Imperial College London
Department of Computing

Inductive Learning of Answer Set Programs

Mark Law

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College London and
the Diploma of Imperial College London, April 2018

Declaration of Originality

I, Mark Law, declare that the work in this thesis is my own. The work of others has been appropriately referenced. A full list of references is given in the bibliography.

Abstract

The goal of Inductive Logic Programming (ILP) is to find a hypothesis that explains a set of examples in the context of some pre-existing background knowledge. Until recently, most research on ILP targeted learning definite logic programs. This thesis constitutes the first comprehensive work on learning answer set programs, introducing new learning frameworks, theoretical results on the complexity and generality of these frameworks, algorithms for learning ASP programs, and an extensive evaluation of these algorithms.

Although there is previous work on learning ASP programs, existing learning frameworks are either *brave* – where examples should be explained by at least one answer set – or *cautious* where examples should be explained by all answer sets. There are cases where brave induction is too weak and cautious induction is too strong. Our proposed frameworks combine brave and cautious learning and can learn ASP programs containing choice rules and constraints. Many applications of ASP use *weak constraints* to express a preference ordering over the answer sets of a program. Learning weak constraints corresponds to *preference learning*, which we achieve by introducing *ordering examples*. We then explore the generality of our frameworks, investigating what it means for a framework to be general enough to distinguish one hypothesis from another. We show that our frameworks are more general than both brave and cautious induction.

We also present a new family of algorithms, called ILASP (Inductive Learning of Answer Set Programs), which we prove to be sound and complete. This work concerns learning from both non-noisy and noisy examples. In the latter case, ILASP returns a hypothesis that maximises the coverage of examples while minimising the length of the hypothesis. In our evaluation, we show that ILASP scales to tasks with large numbers of examples finding accurate hypotheses even in the presence of high proportions of noisy examples.

Acknowledgements

Firstly, I would like to express my gratitude to my PhD supervisors, Professor Alessandra Russo and Dr Krysia Broda for their support and encouragement throughout my PhD, and for their continuous enthusiasm and commitment, even when reading the many drafts of this thesis. Secondly, I would like to thank my parents, who always prioritised and stressed the importance of my education (although I'm not sure they ever expected me to *still* be in education at the age of 27...). I must also thank the members of the SPIKE group at Imperial, particularly Graham Deane for introducing me to Answer Set Programming and Andrea Gaglione for generating the journey simulation data used in Chapters 8 and 11. I would also like to thank Nikos Katzouris, Peter Schüller and Richard Evans for making their datasets available, and for their guidance on how to use them. Finally, I thank Lucy for everything she does.

Dedication

For my grandfather, Howell Phillips.

Contents

List of Figures

Notation

$AS(P)$	The set of all answer sets of P.	37
$AS^{*}(P)$	The set of all optimal answer sets of P.	40
$ApproxCoverage(H, T, CC)$	The approximate coverage of a hypothesis H over the examples in T with respect to the coverage constraints in CC.	191
$Coverage(H, T)$	The set of examples in T which are covered by H.	191
$DISJ(SC)$	$DISJ(SC)$ denotes the set of all rule-disjunctions that appear in SC	204
HB_P	The Herbrand base of P.	34
HB_P^{rel}	The relevant Herbrand base of P.	44
$ILP_X(T_X)$	The set of all inductive solutions of T_X	48
O_{id}	An identifier, which is unique to the object O.	117
$P \equiv_s Q$	P is strongly equivalent to Q.	38
$[\mathtt{id_1} : \lambda_1, \ldots, \mathtt{id_n} : \lambda_n]$	This represents a mapping from rule IDs $\mathtt{id_i}$ to the integers λ_i.	197
$\mathcal{E}_b$ and $\mathcal{E}_c$	$\mathcal{E}_b(P)$ (resp. $\mathcal{E}_c(P)$) denotes the set of conjunctions of literals that are true in at least one (resp. all) answer set of P.	98
$\mathcal{ASP}^R$	The restricted class of ASP programs containing only normal rules, choice rules and hard and weak constraints.	41

$\mathcal{ASP}^{ch}$	The restricted class of ASP programs containing only normal rules, choice rules and hard constraints. 41
$\mathcal{A}$	$\mathcal{A}(P, \texttt{lit})$ is the ASP program generated by appending `lit` to every rule in P 117
$\mathcal{M}^{-1}_{as}$	A function used to extract object-level answer sets from meta-level answer sets 117
$\mathcal{M}^{-1}_{in\_h}$	$\mathcal{M}^{-1}_{in\_h}(A)$ is the hypothesis extracted from the meta-level answer set A 117
$\mathcal{M}^{-1}_{vi}$	$\mathcal{M}^{-1}_{vi}(A)$ is the violating interpretation extracted from the meta-level answer set A 141
$\mathcal{M}^{-1}_{vp}$	$\mathcal{M}^{-1}_{vp}(A)$ is the violating pair extracted from the meta-level answer set A 141
$\mathcal{P}(T)$	The positive hypotheses of T. 125
$\mathcal{P}^n(T)$	The positive hypotheses of T of length n. 125
$\mathcal{S}(H, T)$	The score of the hypothesis H. 177
$\mathcal{S}_{lb}(H, T, CC)$	The lower bound score of H with respect to CC. 192
$\mathcal{V}(T)$	The violating hypotheses of T. 125
$\mathcal{V}^n(T)$	The violating hypotheses of T of length n. 125
$\mathcal{R}$	A function used to reify ASP programs. 117
$\mathcal{NR}$	A function used to reify a conjunction of atoms, while also negating each atom. 143
$\mathcal{R}_{\texttt{in\_as}}$	A shorthand for a common reification used in the ILASP meta-level programs 117

$\uparrow$	The first non-zero element of an ordered list. The function returns 0 if no such element exists.	197
${}^{*}ILP_X(T)$ and ${}^{n}ILP_X(T)$	The set of all optimal inductive solutions of T, and the set of all solutions of T that are of length n, respectively.	70
$agg\_atoms(R)$	The set of atoms that occur in the aggregate elements in $body(R)$.	34
$body(R)$	The set of literals that occur in the body of the rule R.	34
$body^{+}(R)$	The set of atoms that occur as positive naf-literals in the body of the rule R.	34
$body^{-}(R)$	The set of atoms that occur in negative naf-literals in the body of the rule R.	34
$ground(P)$	The complete grounding of P.	35
$ground^{rel}(P)$	The relevant grounding of P.	44
$head(R)$	The head of the rule R.	33
$heads(R)$	The set of atoms that occur in the head of the rule R.	33
$ids(S)$	The set of identifiers of all objects in the set S.	117
$inverse$	$inverse(o)$ is the CDOE o with the operator inverted.	76
$sc_1 \cup sc_2$	The combination of two hypothesis schemas, sc_1 and sc_2	203
$solve(P)$	A function that uses Clingo to find a single optimal answer set of P if P is satisfiable (in the case that P is unsatisfiable, `nil` is returned).	138

Meta Programs

$\mathcal{M}(T)$	This is the main meta-level program used by ILASP1. Its answer sets represent the positive hypotheses of T	127
$\mathcal{M}^n(T)$	Used by the ILASP1 algorithm. The answer sets of this program represent the positive hypotheses of length n.	128
$\mathcal{M}_1(T)$	$\mathcal{M}_1(T)$ is a meta level program which transforms T so that many object-level answer sets can be reasoned about in one meta-level answer set.	118
$\mathcal{M}_{ILASP2}(T, VR)$	Given an $ILP^{context}_{LOAS}$ task T and a set of violating reasons VR, this represents the meta-level representation used by the ILASP2 algorithm.	148
$\mathcal{M}_{fre}(T, H)$	The meta-level program used to check the coverage of H (used by the $findRelevantExamples$ procedure).	152
$\mathcal{M}_{osc}(OSC, S_M)$	The program $\mathcal{M}_{osc}(OSC, S_M)$ is used for deciding which hypotheses $H \subseteq S_M$ conform to which ordering schemas in OSC.	206
$\mathcal{M}_{pair}(T, OSC, o)$	The program $\mathcal{M}_{pair}(T, OSC, o)$ is used to find an accepting pair of answer sets of o, wrt a hypothesis H that does not conform to any schema in OSC.	320
$\mathcal{M}_{reduct}(P, \texttt{int\_id})$	This program-fragment is used to check whether the object-level answer set represented by the `in_int` atoms with id `int_id` is an answer set of the object-level program. It does so by computing the minimal model of the reduct of the object level program.	143

$\mathcal{M}_{sc}(SC, S_M)$	The program $\mathcal{M}_{sc}(SC, S_M)$ is used for deciding which hypotheses $H \subseteq S_M$ conform to which schemas in SC. 205
$\mathcal{M}_{solve}(CC, S_M, E)$	This program is used to find the optimal hypothesis given the schemas and coverage constraints that have been found by the ILASP3 algorithm so far. 208
$\mathcal{M}_{trans}(T, e, SC)$	This program searches for a hypothesis that does not conform to any schema in SC, but does accept the example e. It finds an accepting answer set of e wrt H. 317
$\mathcal{M}_{u}(P, S_M, I, sc)$	Given an interpretation I and a hypothesis schema sc, this meta-level program finds a hypothesis that conforms to sc, but for which I has at least one unfounded set with respect to $B \cup H$. 314
$\mathcal{M}_{violating\_ord}(T)$	This is the program fragment which gives a preference ordering to the answer sets of the meta-level representation in ILASP2. 148
$\mathcal{M}_{vio}(T, S)$	Given a task T and a set of violating reasons S, ILASP2 uses this program fragment to rule out any hypotheses that share any of the violating reasons in S. 145
$\mathcal{M}_{vp}(T, S)$	This program fragment is used in order to determine whether each of a set of violating pairs is a violating pair of another hypothesis. . 146
$check(e, \texttt{as\_id})$	This program fragment is used to check whether the object-level answer set represented by the `in_as` atoms with id `as_id` is an is an accepting answer set of e. 119
$check\_ord(T, o, \texttt{a}_1, \texttt{a}_2)$	This program fragment is used to check whether the pair of object-level answer sets represented by the `in_as` atoms with ids $\texttt{a}_1$ and $\texttt{a}_2$ is an accepting pair of answer sets of o. 121
$constraint(H)$	The constraint used by ILASP1 to rule out a single hypothesis H. 129

List of Tables

Chapter 1

Introduction

Over the last two decades there has been a growing interest in Inductive Logic Programming (ILP) [Mug91, MDRP$^+$12], where the goal is to learn a logic program called a *hypothesis*, which together with an existing background knowledge base, explains a set of observations. Advantages claimed for ILP over statistical machine learning approaches are that the learned hypotheses can be easily expressed in plain English and explained to a human user, and that it is possible to reason with (and correct) learned knowledge. Much of the work on ILP frameworks has focused on learning definite logic programs (e.g. [Mug91, BDR98, Sri01, RBR04, MDRP$^+$12, ML13]) and normal logic programs (e.g. [Sak01, CRL10]).

On the other hand, Answer Set Programming [GL88] is a purely declarative language for knowledge representation and reasoning. Due to its non-monotonicity, ASP is particularly well suited to common-sense reasoning [EIK09, Mue14, GK14]. The typical workflow in ASP is that a real world problem is encoded as an ASP program, whose *answer sets* – a special subset of the models of the program – correspond to the solutions of the original problem. Because of its expressiveness and efficient solving, ASP is also increasingly gaining attention in industry [EGL16]; for example, in decision support systems [NBG$^+$01], in e-tourism [RDG$^+$10] and in product configuration [SN99]. Consequently, the scope of ILP has recently been extended to learning under the answer set semantics [Ote01, Sak01, Ray09, CRL12]. Learning ASP programs allows us to learn a variety of declarative non-monotonic, common-sense theories, including for instance Event Calculus [KS86] theories [KAP15] and user preference models from real user data [Ath15].

1.1 Motivation of Learning ASP Programs

Learning ASP programs has several differences when compared to learning Prolog programs. Firstly, when learning Prolog programs, the goal directed SLDNF procedure of Prolog must be taken into account. Specifically, when learning programs with negation, it must be ensured that the programs are locally stratified, or otherwise the learned program may loop under certain queries. No such consideration needs to be taken into account when learning ASP programs. Another key difference between ASP and Prolog is that ASP is declarative, whereas in Prolog knowledge is represented procedurally. In ASP, logical specifications are separated from control [MT99], meaning that an ASP program can represent the definition of a problem, rather than a procedure to find solutions to that problem. When considering ILP, this distinction is important for two reasons: firstly, explaining a procedure to a non-expert human may be much more difficult than explaining the definition of what that procedure computes; and secondly, in some problems it may be easier to learn a definition, rather than attempting to learn an efficient procedure. In some sense, learning Prolog programs mixes machine learning with program synthesis, where the goal is to learn a procedure to meet a specification. Conversely, because ASP programs separate the procedure from the specification, only the specification needs to be learned by an ASP learner. A third, more fundamental, difference of learning ASP programs is that the theory learned can be expressed using extra types of rules that are not available in Prolog, such as choice rules and weak constraints.

One concept which may be important when learning is the notion of choice. For example, when learning a policy, we may need to learn that in a specific scenario, sc_1, at least one of a set of possible actions, $a_1, \ldots, a_n$, must be executed. This can be expressed in ASP with the following *choice rule*.

$$1\{\mathtt{execute(a_1)}, \ldots, \mathtt{execute(a_n)}\}\mathtt{n\,{:}\text{-}\,holds(sc_1)}.$$

This choice rule expresses that in any answer set where the scenario holds (the answer set contains $\mathtt{holds(sc_1)}$), the answer set must contain between 1 and n of the atoms, $\mathtt{execute(a_1)}, \ldots, \mathtt{execute(a_n)}$. In other words, whenever the scenario holds, at least one (but possibly more) of the actions must be executed. When learning choice rules, we can also learn non-deterministic concepts such as the possible outcomes of tossing a coin. Note that learning choice rules is different from probabilistic ILP settings such as [DRKT07, RBZ14, Nic16] where, in similar coins problems, the focus would be on learning the *probabilities* of the outcomes of tossing a coin. In simple settings when learning ASP programs, the aim is for the answer sets of the learned program to represent the set of possible instances of a problem; for example, the two outcomes of tossing a coin. In these cases, positive and negative examples should be *possible* and *impossible* (partial) instances of a problem. Choice rules are useful to generate the answer sets required to cover all the positive examples of possible instances. In some cases, *hard constraints* are also necessary in the learned program in order to rule out impossible instances (given by the negative examples).

Most of the types of rules in ASP are used for defining the answer sets of a program. Learning these kinds of rules allows us to perform *classification* tasks. Another important area of machine learning is *preference learning*, where the goal is to learn a user's preferences over a set of objects, given examples of which objects the user prefers to other objects. For instance, given a set of a user's pairwise preferences over possible schedules, cars, holiday packages or journeys, it is possible to learn to predict the user's preferences over future schedules, cars, holiday packages or journeys. If these preferences are encoded in logic, it is even possible to search for (or in some cases, construct) an optimal solution. Preferences in ASP can be encoded using *weak constraints*. Unlike other rules in ASP, weak constraints do not affect what is (or is not) an answer set of a program, but instead represent a preference ordering over the answer sets of a program. For instance, a program can be defined to represent a set of possible journeys a user might take to get from one location to another. The weak constraints in this program could express the user's preferences; for instance, the user may like to minimise the walking distance, or minimise the driving distance through congested cities. It is even possible to represent that some preferences are more important than other preferences (and should therefore be given priority).

This thesis aims to provide the first comprehensive work on learning ASP programs including normal rules, choice rules and both hard and weak constraints.

1.2 Contributions

1.2.1 Frameworks for Learning Under the Answer Set Semantics

The idea of learning ASP programs has been considered before [Ote01, SI09], and several frameworks[1] for learning under the answer set semantics have been introduced. The main approaches can be divided into *brave* learning frameworks – where examples should be covered in at least one answer set of the learned program – and *cautious* learning frameworks – where examples should be covered in all answer sets. One immediate question is whether any of these frameworks are general enough to learn ASP programs including constructs such as choice rules or weak constraints. For that matter, what does it even mean for a framework to be general enough to learn a particular class of programs?

We present, as a first contribution of this thesis, a definition of the generality of a learning framework and an investigation of the generalities of existing learning frameworks under the answer set semantics. We prove that none of the existing frameworks is general enough to learn the full class of ASP programs that we target in this thesis.

As a second contribution, this thesis presents a collection of progressively more general frameworks for learning ASP programs [LRB14, LRB15a, LRB16]. In particular, our most expressive framework, called *context-dependent learning from ordered answer sets* ($ILP_{LOAS}^{context}$) [LRB16], is able to learn programs consisting of normal rules, choice rules and both hard and weak constraints. We show that $ILP_{LOAS}^{context}$ is general enough to learn any ASP program (up to strong equivalence in ASP), making it more general than any other existing framework.

An obvious question would be whether the extra generality of $ILP_{LOAS}^{context}$ over other frameworks means that it is computationally harder to solve its tasks. In fact, this is not the case, and we show that on three important decision problems – *verification*, *satisfiability* and *optimum verification* – each

[1]In this thesis, a *learning framework* refers to a general setting for learning, which consists of a definition of the structure of examples and a definition of the inductive solutions of learning tasks in the framework. A *learning task* is a particular learning instance, consisting of a background knowledge, a hypothesis space and a set of examples.

of our learning frameworks has the same computational complexity as cautious induction (the existing learning framework with highest complexity). The verification problem corresponds to deciding whether a given hypothesis is a solution of a given learning task, the satisfiability problem corresponds to deciding whether a learning task has any solutions at all, and the optimum verification problem corresponds to deciding whether a given hypothesis is the optimal (shortest) solution of a given task.

1.2.2 Algorithms for Learning ASP Programs

The third major contribution of this thesis is a collection of new algorithms, called ILASP, for solving $ILP_{LOAS}^{context}$ learning tasks. In recent years, there have been several ILP systems that have used ASP solvers to solve brave induction tasks. XHAIL [Ray09] and ILED [KAP15], for example, use an ASP solver at various stages of their computation. The ASPAL [CRL12] and RASPAL [ACBR13] systems go one step further, and encode an ILP task as a meta-level ASP program, whose optimal answer sets correspond to the optimal inductive solutions of the original task. These existing systems for ILP that use an ASP solver as a computation engine were only designed to solve brave induction tasks. Hence, they cannot learn the full class of ASP programs targeted in this thesis.

We present a collection of progressively more efficient ILASP algorithms for finding optimal solutions to tasks of our learning frameworks. Each of these algorithms is sound and complete with respect to the optimal inductive solutions of a task, meaning that if they are run on a satisfiable task, then they are each guaranteed to return an optimal inductive solution of that task.

In Chapter 8, we evaluate these algorithms on several synthetic datasets and show that the scalability across these algorithms increases with respect to the number of examples. In particular, the ILASP2i algorithm scales to tasks with hundreds of examples. Conversely, we show that at least on these synthetic datasets, the ILASP algorithms are able to learn highly accurate hypotheses using relatively few (tens of) randomly selected examples.

1.2.3 Learning ASP Programs from Noisy Examples

The above three contributions are presented in Part I of this thesis and they refer to learning tasks where all examples are assumed to be perfectly labeled, meaning that any inductive solution of a task must cover every example of that task. In practice, of course, examples are unlikely to be perfectly labeled. In real datasets, there is likely to be *noise*, and a more realistic approach is to search for a hypothesis that covers the majority of examples, and weighs the example coverage against the complexity of the hypothesis – dramatically increasing the hypothesis complexity in order to cover a few more examples is undesirable, as these examples may well be incorrectly labeled.

Part II of this thesis generalises our work, enabling the learning of ASP programs from noisy examples. The first step of this generalisation is to extend our learning frameworks with a notion of penalties on examples – a cost that needs to be paid for not covering a particular example. We also extend our generality and complexity results, showing that the "noisy" extension of $ILP^{context}_{LOAS}$ is more general than similar extensions of other existing learning frameworks, and that the complexity of $ILP^{context}_{LOAS}$ on the three decision problems presented in Part I is unaffected by this extension.

As a second step of this generalisation to noisy tasks, we show that although our non-noisy ILASP algorithms can be extended to solve these tasks, they may not be very efficient. In Chapter 10, we propose a new algorithm, called ILASP3, which uses a very different approach to the other ILASP algorithms, as it is specifically targeted at learning in the presence of noise. ILASP3 is sound and complete with respect to the optimal solutions of any "noisy" $ILP^{context}_{LOAS}$ task.

We evaluate ILASP3 both on synthetic datasets, and on real datasets. On the real datasets, we compare the accuracy of hypotheses learned by ILASP3 with that achieved by the ILP algorithms that have been applied to those datasets (in [KAP16, KSS17, QK17, EG18]). We show that, in most cases, ILASP3 achieves a higher accuracy than the other systems.

1.2.4 Publications

Some of the work in this thesis has appeared in the following publications:

[LRB14] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, pages 311–325, 2014

[LRB15a] Mark Law, Alessandra Russo, and Krysia Broda. Learning weak constraints in answer set programming. *Theory and Practice of Logic Programming*, 15(4-5):511–525, 2015

[LRB16] Mark Law, Alessandra Russo, and Krysia Broda. Iterative learning of answer set programs from context dependent examples. *Theory and Practice of Logic Programming*, 16(5-6):834–848, 2016

[LRB18a] Mark Law, Alessandra Russo, and Krysia Broda. The complexity and generality of learning answer set programs. *Artificial Intelligence*, 259:110–146, 2018

[LRB18b] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs from noisy examples. *Advances in Cognitive Systems*, 2018

1.3 Thesis Structure

We first recall relevant background material in Chapter 2 and discuss the relevant literature related to Inductive Logic Programming in Chapter 3. The remainder of the thesis is spilt into two parts, presenting our work on learning from non-noisy examples in Part I and then from noisy examples in Part II.

Part I: Learning Answer Set Programs from Non-Noisy Examples

Chapter 4 introduces our three frameworks for ILP under the answer set semantics, which were presented in [LRB14, LRB15a, LRB16]. This chapter also contains the complexity results for the learning frameworks considered in this part of the thesis, in addition to complexity results for three existing frameworks (brave induction, cautious induction and induction of stable models). We show that our frameworks can all simulate any of the three existing frameworks.

Chapter 5 introduces three new measures of the generality of a learning framework, which were presented in [LRB18a]. We show that under each measure, $ILP_{LOAS}^{context}$ is the most general learning framework out of the frameworks for learning under the answer set semantics.

Chapter 6 describes the first ILASP algorithm, which was presented in [LRB14], and proves it to be sound and complete with respect to the optimal inductive solutions of any $ILP_{LOAS}^{context}$ task.

Chapter 7 describes two ways of scaling up the first ILASP algorithm. The first, ILASP2 [LRB15a], is based on the new notion of *violating reasons* which can be used to eliminate many *violating hypotheses*. The second, ILASP2i [LRB16], is an iterative version of ILASP2 that incrementally computes a set of *relevant examples*.

Chapter 8 evaluates the ILASP1, ILASP2 and ILASP2i algorithms on several synthetic datasets, and shows that the algorithms are able to find highly accurate hypotheses from relatively few (tens of) examples. We also show that ILASP2i can scale to tasks with hundreds of examples. The experiments in this chapter are based on the evaluations in [LRB15a] and [LRB16].

Part II: Learning Answer Set Programs from Noisy Examples

Chapter 9 extends our learning frameworks to handle noise. We show how this extension affects the complexity and generality results presented in Chapters 4 and 5 (respectively), and explain why our previous ILASP algorithms are not well suited to solving noisy tasks.

Chapter 10 introduces the notion of *hypothesis schemas* and *ordering schemas*, which can be used in order to represent examples in a much more compact way. The ILASP3 algorithm presented in this chapter is based on the idea of translating examples into these schemas, and searching for a hypothesis that conforms to the computed schemas.

Chapter 11 evaluates the ILASP3 algorithm, both on synthetic datasets and on real datasets, and compares it to existing ILP algorithms that have been applied to the same datasets. We show that, in most cases, ILASP3 achieves a higher accuracy than the other systems.

Chapter 12 concludes the thesis, summarising the key contributions and outcomes. We also discuss our intended directions for future work.

Appendix A formalises the ILASP language bias, and gives the language biases of the ILASP tasks described in Chapters 8 and 11.

Appendix B presents the proofs which were omitted from the main body of the thesis.

Chapter 2

Preliminaries

In this chapter we recall the background material used throughout this thesis. In particular, we outline the subset of the language of ASP used in the rest of the thesis. For a more complete overview of the standardised syntax used by the leading ASP solvers, please see [CFG$^+$13]. Our learning systems accept a restricted form of ASP program, which include only normal rules, choice rules and hard and weak constraints. We denote this class of *restricted* programs as $\mathcal{ASP}^R$. As our systems use meta-level ASP programs using a wider class of rules (e.g. summing aggregates), we first present this wider class of programs, and then define the restriction.

2.1 The Syntax of ASP

This section summarises the syntax of the subset of the ASP standard from [CFG$^+$13] that we use. A *basic symbol* is a string of characters that starts with a lower case letter and contains only letters, digits and the '_' (underscore) character. *Predicates* and *functions* are each basic symbols and a *constant* is either an integer or a basic symbol.

A *term* is one of the following:

- A constant $\mathtt{c}$.
- A variable $\mathtt{V}$, where $\mathtt{V}$ is a string of letters and digits and underscores, beginning with an uppercase letter.
- An arithmetic term $-(\mathtt{t_1})$ or $(\mathtt{t_1} \diamond \mathtt{t_2})$, where $\mathtt{t_1}$ and $\mathtt{t_2}$ are terms and $\diamond \in \{+, -, \times, /\}$. When brackets are omitted, standard operator precedences apply (multiplication and division are given higher precedence than addition and subtraction).
- A functional term $\mathtt{f}(\mathtt{t_1}, \ldots, \mathtt{t_n})$, where $\mathtt{f}$ is a function symbol and $\mathtt{t_1}, \ldots, \mathtt{t_n}$ are terms.

Given a functional term $\mathtt{t}$ of the form $\mathtt{f(t_1, \ldots, t_n)}$, we refer to n as the *arity* of $\mathtt{t}$ and $\mathtt{t_1, \ldots, t_n}$ as the arguments of $\mathtt{t}$.

An *atom* is of the form $\mathtt{p(t_1, \ldots, t_n)}$, where $\mathtt{p}$ is a predicate symbol and $\mathtt{t_1, \ldots, t_n}$ are terms. When $\mathtt{n} = 0$, we omit the brackets and write the atom as $\mathtt{p}$. We define the arity and arguments of an atom similarly to functional terms.

2.1.1 Literals

A *naf-literal* is either $\mathtt{a}$ or $\mathtt{not\ a}$ where $\mathtt{a}$ is an atom, and $\mathtt{not}$ denotes *negation as failure* [Cla77]. We do not consider *classical negation* in this thesis, as it can be trivially simulated through additional atoms and constraints in ASP.

The *binary comparison operators* that we consider in this thesis are $\{<, >, \leq, \geq, =, \neq\}$. A *comparison literal* is of the form $\mathtt{t_1} \prec \mathtt{t_2}$, where $\mathtt{t_1}$ and $\mathtt{t_2}$ are both terms and $\prec$ is a binary comparison operator.

There is another form of literal in ASP called an *aggregate*. *Aggregate elements* are of the form $\mathtt{t_1, \ldots, t_j : l_1, \ldots, l_k}$, where $\mathtt{t_1, \ldots, t_j}$ are terms and $\mathtt{l_1, \ldots, l_k}$ are naf-literals. In general, an aggregate is of the form $\#\mathtt{aggr}\{\mathtt{e_1}; \ldots; \mathtt{e_n}\} \prec \mathtt{u}$, where $\mathtt{e_1, \ldots, e_n}$ are *aggregate elements*, $\prec$ is a binary comparison operator, $\mathtt{u} \in \mathbb{Z}$ and $\#\mathtt{aggr}$ is an *aggregate function name*, which specifies the meaning of the aggregate. The two aggregate function names that we use in this thesis are $\#\mathtt{count}$ and $\#\mathtt{sum}$ (we refer to the corresponding aggregate atoms as *counting aggregates* and *summing aggregates*, respectively).

Example 2.1. $\#\mathtt{sum}\{\mathtt{1, X : p(X); 2, Y : q(Y),\ not\ r(Y)}\} > 3$ *is an aggregate atom. Its aggregate elements are* $\mathtt{1, X : p(X)}$ *and* $\mathtt{2, Y : q(Y),\ not\ r(Y)}$*. We will revisit the meaning of aggregates such as this when we discuss the semantics of ASP later in this chapter, but essentially, it is satisfied whenever the number of* $\mathtt{X}$*'s for which* $\mathtt{p(X)}$ *is true plus double the number of* $\mathtt{Y}$*'s for which* $\mathtt{q(Y)}$ *is true and* $\mathtt{r(Y)}$ *is false is greater than 3.*

In this thesis, the term *literal* is used to mean any naf-literal, comparison literal or aggregate.

2.1.2 Rules

There are several different types of rule that we consider in this thesis, each of which has various restrictions on what can occur in the rule. Given a set of atoms $\mathtt{h_1, \ldots, h_j}$ and a set of literals $\mathtt{b_1, \ldots, b_k}$, a *general rule* is of the form:

$$\mathtt{h_1} \vee \ldots \vee \mathtt{h_j} \mathtt{\ :\!-\ } \mathtt{b_1, \ldots, b_k}.$$

$\mathtt{h_1} \vee \ldots \vee \mathtt{h_j}$ is called the *head* of R and is denoted *head*(R). We also write *heads*(R) to represent the set of atoms that occur in *head*(R). The set of literals $\{\mathtt{b_1}, \ldots, \mathtt{b_k}\}$ is called the *body* of R and is denoted *body*(R). We also write *body*$^+(R)$ and *body*$^-(R)$ to denote the atoms that occur in positive and negative naf-literals in *body*(R) (respectively). *agg_atoms*(R) denotes the set of atoms that occur in the aggregate elements in *body*(R).

A *disjunctive rule* is a rule with no aggregate literals. A *hard constraint* is a disjunctive rule R such that *heads*(R) is empty. A *normal rule* is a disjunctive rule such that $|heads(R)| = 1$ and a *definite rule* is a normal rule R such that *body*$^-(R)$ is empty. We refer to sets of disjunctive, normal and definite rules as *disjunctive*, *normal* and *definite* logic programs, respectively.

2.1.3 Weak Constraints

A weak constraint W is of the form $:\sim \mathtt{b_1}, \ldots, \mathtt{b_m}.[\mathtt{wt@lev}, \mathtt{t_1}, \ldots \mathtt{t_n}]$, where $\mathtt{b_1}, \ldots, \mathtt{b_m}$ are naf-literals or comparison literals called (collectively) the body of W, $\mathtt{wt}$ and $\mathtt{lev}$ are terms called (respectively) the *weight* and *priority level* of W and $\mathtt{t_1}, \ldots, \mathtt{t_n}$ are terms. We call $[\mathtt{wt@lev}, \mathtt{t_1}, \ldots \mathtt{t_n}]$ the tail of W, and write *tail*(W) to refer to the tuple $(\mathtt{wt}, \mathtt{lev}, \mathtt{t_1}, \ldots, \mathtt{t_n})$.

Weak constraints have a very different role to other rules in ASP programs. As such, it is often useful to separate them. Given a program P, we write *weak*(P) and *non_weak*(P) to refer to the set of weak constraints in P and the set of rules in P, respectively.

2.2 The Semantics of ASP

The semantics of an ASP program is defined in terms of its *answer sets*. Usually, when we *solve* an ASP program, we compute its answer sets, although sometimes solving an ASP program can mean searching for a single answer set.

The *Herbrand universe* of any ASP program P is the set of all terms that can be constructed from function symbols and constant symbols in P. We write HU_P to denote the Herbrand universe of P. The *Herbrand base* of P, denoted HB_P, is the set of all atoms constructed using a predicate symbol that occurs in P and whose arguments are terms from HU_P.

The first step to solving an ASP program is to *ground* it. This means transforming the program into an equivalent *ground* program, where any term, atom or rule is said to be ground if it contains no variables. Given a program P:

- $vars(P)$ is the set of all variables that occur in P (we extend this notation to terms, atoms and rules).
- A substitution θ over a program P is a partial function $\theta : vars(P) \rightarrow HU_P$ (i.e. the domain of θ is a subset of $vars(P)$ and the range of θ is a subset of HU_P).

- Given any rule, weak constraint, term, literal or aggregate element L in P, and any substitution θ over P, we write $\theta(L)$ to denote the result of applying θ to the variables in L that occur in the domain of θ.
- Given a rule or weak constraint R in P, a *global substitution* of R is a substitution θ such that the domain of θ is equal to those variables in R that occur at least once outside of the aggregate elements in R.

Example 2.2. *Consider the rule R,* $\texttt{p(X)}\,\texttt{:-}\,\texttt{q(X,Y)}, \#\texttt{sum}\{\texttt{Z,X : q(X,Z)}\} < \texttt{Y}$. *Any global substitution of R must map the variables* X *and* Y*, but not the variable* Z *(as* Z only *occurs in an aggregate element).*

One global substitution is $\theta = \{\texttt{X} \rightarrow \texttt{1},\ \texttt{Y} \rightarrow \texttt{2}\}$*. Applying θ to R (denoted $\theta(R)$) yields the rule* $\texttt{p(1)}\,\texttt{:-}\,\texttt{q(1,2)}, \#\texttt{sum}\{\texttt{Z,1 : q(1,Z)}\} < \texttt{2}$.

Having defined substitutions over terms, atoms, rules and weak constraints, we can now describe the ground instances of each. As variables that occur in an aggregate element in a rule (but not in the rest of the rule) are considered "local" to the aggregate element, rules with aggregates are ground in two steps: first, the "global" variables that occur outside the aggregate elements are ground, and then the local variables are ground. We define the ground instances of rules and weak constraints as follows:

- Given an aggregate literal a of the form $\#\texttt{aggr}\{\texttt{e}_1;\ldots;\texttt{e}_\texttt{n}\} \prec \texttt{u}$, the *ground instantiation* of a is the aggregate $\#\texttt{aggr}\{\texttt{g}_1;\ldots;\texttt{g}_\texttt{m}\} \prec \texttt{u}$, where $\{\texttt{g}_1, \ldots, \texttt{g}_\texttt{m}\} = \{\theta(e_i) \mid i \in [1,n], \theta : vars(P) \rightarrow HU_P,\ \text{st } \theta(e_i) \text{ is ground}\}$.
- Given a rule or weak constraint R the *ground instances* of R can be computed in two steps:
  1. Compute the set GR of all rules R' for which there is a global substitution θ of R such that $R' = \theta(R)$.
  2. For each aggregate literal a in GR, replace a with the ground instantiation of a.

Example 2.3. *Consider a program P such that $HB_U = \{\texttt{1},\ \texttt{2}\}$. Reconsider the rule R from Example 2.2. Given this Herbrand universe, there are four global substitutions of R* ($\{\{\texttt{X} \rightarrow \texttt{1},\ \texttt{Y} \rightarrow \texttt{1}\}, \{\texttt{X} \rightarrow \texttt{1},\ \texttt{Y} \rightarrow \texttt{2}\}, \{\texttt{X} \rightarrow \texttt{2},\ \texttt{Y} \rightarrow \texttt{1}\}, \{\texttt{X} \rightarrow \texttt{2},\ \texttt{Y} \rightarrow \texttt{2}\}\}$)*. This leads to four ground instances of R – one for each global substitutions.*

Consider the substitution $\theta = \{\texttt{X} \rightarrow \texttt{1},\ \texttt{Y} \rightarrow \texttt{2}\}$. *$\theta(R)$ contains the aggregate* $\#\texttt{sum}\{\texttt{Z,1 : q(1,Z)}\} < \texttt{2}$. *The substitutions for the single aggregate element in this literal are* $\{\texttt{Z} \rightarrow \texttt{1}\}$ *and* $\{\texttt{Z} \rightarrow \texttt{2}\}$*, meaning that the ground instantiation of this literal is* $\#\texttt{sum}\{\texttt{1,1 : q(1,1); 2,1 : q(1,2)}\} < \texttt{2}$. *Hence, one of the four ground instances of R is the rule* $\texttt{p(1)}\,\texttt{:-}\,\texttt{q(1,2)}, \#\texttt{sum}\{\texttt{1,1 : q(1,1); 2,1 : q(1,2)}\} < \texttt{2}$.

The *complete grounding* of a program P, denoted *ground*(P) is the program consisting of all ground instances of all rules and weak constraints in P. As in many cases this grounding is infinite, in practice ASP solvers use a smaller equivalent ground program called the *relevant grounding.* In Section 2.2.4, we present the notion of *relevant grounding* used in this thesis. We next define the semantics of ground ASP programs.

A *Herbrand interpretation* I of a program P is an assignment from every atom in HB_P to a truth value, $\top$ or $\bot$ (true or false). By convention, we write a Herbrand interpretation I as the set of all atoms in HB_P that I assigns to $\top$. As in some parts of this thesis, it is necessary to treat interpretations separately from any program (e.g. when the program has yet to be learned, and is thus unknown), we will loosely refer to any set of atoms as an interpretation.

We now define what it means for a Herbrand interpretation I to *satisfy* ground naf-literals and comparison literals in a program.

- Let $\mathtt{a}$ be an atom. I satisfies $\mathtt{a}$ if and only if $\mathtt{a} \in I$
- Let $\mathtt{a}$ be an atom. I satisfies $\mathtt{not\ a}$ if and only if $\mathtt{a} \notin I$
- Let l be the comparison literal $\mathtt{t_1} \prec \mathtt{t_2}$. I satisfies l if and only if $\mathtt{t_1} \prec \mathtt{t_2}$, where the $<$, $>$, $\leq$, and $\geq$ operators are defined lexicographically over ground terms.

Given any interpretation I and any aggregate literal $\mathtt{a}$ of the form $\#\mathtt{aggr}\{\mathtt{e_1};\ldots;\mathtt{e_n}\} \prec \mathtt{u}$, we write $tuples(\mathtt{a}, I)$ to denote the set of tuples $(t_1, \ldots, t_j)$ for which there is at least one aggregate element $\mathtt{t_1}, \ldots, \mathtt{t_j} : \mathtt{l_1}, \ldots, \mathtt{l_k}$ in $\{\mathtt{e_1}, \ldots, \mathtt{e_n}\}$ such that $\mathtt{l_1}, \ldots, \mathtt{l_k}$ are all satisfied by I.

- Let c be the ground *counting aggregate* $\#\mathtt{count}\{\mathtt{e_1};\ldots;\mathtt{e_n}\} \prec \mathtt{u}$. I satisfies c if and only if $|tuples(c, I)| \prec u$.
- Let s be the ground *summing aggregate* $\#\mathtt{sum}\{\mathtt{e_1};\ldots;\mathtt{e_n}\} \prec \mathtt{u}$. I satisfies s if and only if $\left(\sum_{(wt,t_1,\ldots,t_j) \in tuples(s,I), wt \in \mathbb{Z}} wt\right) \prec u$.
- Let $\mathtt{l_1}, \ldots, \mathtt{l_n}$ be a set of literals. I satisfies the *conjunction* $\mathtt{l_1}, \ldots, \mathtt{l_n}$ if and only if I satisfies each literal in $\{\mathtt{l_1}, \ldots, \mathtt{l_n}\}$.
- Let $\mathtt{l_1}, \ldots, \mathtt{l_n}$ be a set of literals. I satisfies the *disjunction* $\mathtt{l_1} \vee \ldots \vee \mathtt{l_n}$ if and only if I satisfies at least one literal in $\{\mathtt{l_1}, \ldots, \mathtt{l_n}\}$. Note that this means that the head of a hard constraint cannot be satisfied by any interpretation.
- A rule R is satisfied by I if either *head*(R) is satisfied by I or *body*(R) is not satisfied by I.

A Herbrand interpretation I is said to be a *(Herbrand) model* of a program P if it satisfies every rule in P (i.e. for every rule R such that I satisfies the body of R, I must also satisfy the head of R). A model I of a program P is said to be a *minimal model* of P if no proper subset of I is a model of P.

Fact 2.1. *[VEK76] Any definite logic program P has exactly one minimal model. We call this model the* least Herbrand model *of P, and denote it $M(P)$.*

In order to determine whether an interpretation I is an *answer set* (often called a stable model) of a program P, we construct the *reduct* [GL88] of P with respect to I. Many definitions of the reduct[1] of an ASP program exist [Lif08]. We follow the reduct defined in [CFG+13].

Definition 2.1. ([CFG+13]) The reduct of a ground program P with respect to an interpretation I (denoted P^I) is the program:

$$\left\{ \texttt{h}_1 \vee \ldots \vee \texttt{h}_\texttt{j} \texttt{:-} \texttt{b}_1, \ldots, \texttt{b}_\texttt{k} \in \mathit{non\_weak}(P) \;\middle|\; \texttt{b}_1, \ldots, \texttt{b}_\texttt{k} \text{ is satisfied by } I \right\}$$

An interpretation I is said to be an *answer set* of a program P if and only if I is a subset-minimal model of P^I (i.e I is a model of P^I and no proper subset of I is a model of P^I). The answer sets of a non-ground program P are the answer sets of the complete grounding of P. We write $AS(P)$ to denote the set of all answer sets of P.

Example 2.4. *Consider the program P below.*

$$P = \left\{ \begin{array}{l} \texttt{p :- b, not q.} \\ \texttt{q :- not p.} \\ \texttt{a} \vee \texttt{b :- \#sum\{1 : p; 2 : q\}} < 2. \end{array} \right\}$$

Consider the interpretation $X_1 = \{\texttt{q}\}$, the reduct of P wrt X_1 (denoted P^{X_1}) is as follows:

$P^{X_1} = \left\{ \texttt{q :- not p.} \right\}$

This reduct has two minimal models, $\{\texttt{p}\}$ and $\{\texttt{q}\}$, one of which is X_1, and hence X_1 is an answer set of P. The reducts for all 16 Herbrand interpretations of P (and their minimal models) are summarised by the following table.

X's	P^X	*Minimal models of* P^X	$X \in AS(P)$*?*
{q}	q :- not p.	{p} *and* {q}	*Yes*
{p, b}	p :- b, not q. a ∨ b :- #sum{1 : p; 2 : q} < 2.	{p, b}, {a} *and* {q}	*Yes*
{a, b}, {b}	p :- b, not q. q :- not p. a ∨ b :- #sum{1 : p; 2 : q} < 2.	{p, a}, {p, b} *and* {q}	*No*
∅, {a}	q :- not p. a ∨ b :- #sum{1 : p; 2 : q} < 2.	{p, a}, {p, b} *and* {q}	*No*
{p}, {p, a}	a ∨ b :- #sum{1 : p; 2 : q} < 2.	{a}, {b} *and* {q}	*No*
{p, q}, {p, q, a}, {p, q, b}, {p, q, a, b}	∅	∅	*No*
{p, a, b}	p :- b, not q. a ∨ b :- #sum{1 : p; 2 : q} < 2.	{p, b}, {a}, {q}	*No*
{q, a}, {q, b}, {q, a, b}	q :- not p.	{p} *and* {q}	*No*

[1]For the subset of ASP programs considered in this thesis, these definitions are all equivalent. In general, however, when programs contain unstratified aggregates, there are multiple different semantics for ASP.

We say a program P *bravely entails* an atom $\mathtt{a}$ (written $P \models_b \mathtt{a}$) if there is at least one answer set A of P such that $\mathtt{a} \in A$. Similarly, P *cautiously entails* $\mathtt{a}$ (written $P \models_c \mathtt{a}$) if for every answer set A of P, $\mathtt{a} \in A$. Two ASP programs P and Q are *strongly equivalent* (written $P \equiv_s Q$) if for every ASP program R, $AS(P \cup R) = AS(Q \cup R)$.

Example 2.5. *Consider the programs P_1, P_2 and P_3 below.*

$$P_1 = \left\{ \begin{array}{l} \mathtt{p.} \\ \mathtt{q.} \end{array} \right\} \qquad P_2 = \left\{ \begin{array}{l} \mathtt{p\,:\!-\,q.} \\ \mathtt{q.} \end{array} \right\} \qquad P_3 = \left\{ \begin{array}{l} \mathtt{p\,:\!-\;not\;r.} \\ \mathtt{q.} \end{array} \right\}$$

All three programs have the same answer sets (a single answer set $\{\mathtt{p,\ q}\}$*). P_1 and P_2 are strongly equivalent, as for any program R, $AS(P_1 \cup R) = AS(P_2 \cup R)$ – regardless of the rules R contains, any answer set of each program must contain both* $\mathtt{p}$ *and* $\mathtt{q}$*. P_3 is not strongly equivalent to either P_1 or P_2. To see this, consider the program $R = \{\mathtt{r.}\}$. $AS(P_1 \cup R) = AS(P_2 \cup R) = \{\{\mathtt{p,q,r}\}\}$, but $AS(P_3 \cup R) = \{\{\mathtt{q,r}\}\}$.*

2.2.1 Shorthands

A choice rule is of the form $\mathtt{lb}\{\mathtt{h_1},\ldots,\mathtt{h_m}\}\mathtt{ub}\,\mathtt{:\!-}\,\mathtt{b_1},\ldots,\mathtt{b_n}$, where $\mathtt{lb}$ and $\mathtt{ub}$ are integers, $\mathtt{h_1},\ldots,\mathtt{h_m}$ are atoms and $\mathtt{b_1},\ldots,\mathtt{b_n}$ are naf-literals or comparison literals. It is a shorthand for the rules:

$$\left\{ \begin{array}{l} \mathtt{h_1} \vee \widehat{\mathtt{h_1}}\,\mathtt{:\!-}\,\mathtt{b_1},\ldots\mathtt{b_n}. \\ \qquad\ldots \\ \mathtt{h_m} \vee \widehat{\mathtt{h_m}}\,\mathtt{:\!-}\,\mathtt{b_1},\ldots\mathtt{b_n}. \\ \mathtt{:\!-}\,\mathtt{b_1},\ldots\mathtt{b_n}, \#\mathtt{count}\{\mathtt{h_1} : \mathtt{h_1};\ldots;\mathtt{h_m} : \mathtt{h_m}\} < \mathtt{lb}. \\ \mathtt{:\!-}\,\mathtt{b_1},\ldots\mathtt{b_n}, \#\mathtt{count}\{\mathtt{h_1} : \mathtt{h_1};\ldots;\mathtt{h_m} : \mathtt{h_m}\} > \mathtt{ub}. \end{array} \right\}$$

where $\widehat{\mathtt{h_1}},\ldots,\widehat{\mathtt{h_m}}$ are each new atoms that do not occur anywhere in the rest of the program. Essentially, $\widehat{\mathtt{h_i}}$ stands for the negation of $\mathtt{h_1}$. Note that each $\mathtt{h_i}$ occurs both on the left and right hand side of the :'s in aggregate elements. On the left hand side, it is treated as a term (the predicate symbol is treated as a function symbol[2]). Using the atom as a term on the left hand side ensures that each aggregate element is counted as unique[3].

Example 2.6. *Consider the choice rule* $\mathtt{1\{p(X),q(X),r(X,Y)\}2\,:\!-\,s(X,Y)}$*. This rule represents the set of rules below:*

$$\left\{ \begin{array}{l} \mathtt{p(X)} \vee \widehat{\mathtt{p(X)}}\,\mathtt{:\!-\,s(X,Y).} \\ \mathtt{q(X)} \vee \widehat{\mathtt{q(X)}}\,\mathtt{:\!-\,s(X,Y).} \\ \mathtt{r(X,Y)} \vee \widehat{\mathtt{r(X,Y)}}\,\mathtt{:\!-\,s(X,Y).} \\ \mathtt{:\!-\,s(X,Y)}, \#\mathtt{count}\{\mathtt{p(X) : p(X); q(X) : q(X); r(X,Y) : r(X,Y)}\} < 1. \\ \mathtt{:\!-\,s(X,Y)}, \#\mathtt{count}\{\mathtt{p(X) : p(X); q(X) : q(X); r(X,Y) : r(X,Y)}\} > 2. \end{array} \right\}$$

[2]ASP allows the same symbol to be used as both a predicate and a function symbol.

[3]If two or more aggregate elements share exactly the same list of terms on the left hand side of the :'s then they are counted only once, as the set of tuples that are counted does not allow for duplicate elements.

Loosely speaking, the choice rule states that for every X *and* Y*, if* s(X, Y) *is true then between 1 and 2 of the atoms* p(X)*,* q(X) *and* r(X, Y) *must be true. The first three rules in the translation above state that if* s(X, Y) *is true, then all combinations of the possible truth values of the atoms in the head of the choice rule should be generated as possible answer sets. The final two rules are constraints which enforce the lower and upper bounds of the choice rule (respectively). In Section 2.2.3, we define the semantics of choice rules directly (without the need for a translation). This definition is equivalent to using this translation and the standard definition of the ASP semantics (using the reduct in Definition 2.1) [LRB15b].*

As the summing aggregates used in this thesis take a very specific form, we introduce another shorthand, which is similar to the summing aggregates used in the input of the Clingo 3 solver [GKK+10]. A *simple summing aggregate* is of the form $\#\mathtt{sum}\{\mathtt{a_1} = \mathtt{w_1}, \ldots, \mathtt{a_n} = \mathtt{w_n}\} \prec \mathtt{u}$, where $\mathtt{a_1}, \ldots, \mathtt{a_n}$ are atoms and $\mathtt{w_1}, \ldots, \mathtt{w_n}$ are integers. It is a shorthand for the aggregate $\#\mathtt{sum}\{\mathtt{w_1}, \mathtt{a_1} : \mathtt{a_1}; \ldots; \mathtt{w_n}, \mathtt{a_n} : \mathtt{a_n}\} \prec \mathtt{u}$ (where on the left hand side of the :'s, the $\mathtt{a_i}$ atoms are treated as functional terms rather than as atoms).

Example 2.7. *Consider the simple summing aggregate* $\#\mathtt{sum}\{\mathtt{p(X)} = \mathtt{X}, \mathtt{q(X)} = \mathtt{3}\} < \mathtt{40}$*. The truth value of the aggregate (wrt an interpretation) can be evaluated directly. An interpretation I satisfies this aggregate if and only if the following inequality holds:*

$$\left(3\times \mid \{\mathtt{t} \mid \mathtt{q(t)} \in I\} \mid + \sum_{\mathtt{p(t)} \in I, \mathtt{t} \in \mathbb{Z}} t\right) < 40$$

The standard aggregate AGG represented by this simple summing aggregate is $\#\mathtt{sum}\{\mathtt{X}, \mathtt{p(X)} : \mathtt{p(X)}; \mathtt{3}, \mathtt{q(X)} : \mathtt{q(X)}\} < \mathtt{40}$*. This aggregate is satisfied by any interpretation I if and only if the following inequality holds:*

$$\left(\sum_{(wt, t_1, \ldots, t_n) \in tuples(AGG, I), wt \in \mathbb{Z}} wt\right) < 40$$

For any interpretation I, $tuples(AGG, I) = \{(\mathtt{t}, \mathtt{p(t)}) \mid \mathtt{p(t)} \in I\} \cup \{(\mathtt{3}, \mathtt{q(t)}) \mid \mathtt{q(t)} \in I\}$*. Hence the inequality is equivalent to:*

$$\left(\sum_{\mathtt{q(t)} \in I} 3 + \sum_{\mathtt{p(t)} \in I, \mathtt{t} \in \mathbb{Z}} t\right) < 40$$

This is equivalent to the inequality derived directly from the original simple summing aggregate.

2.2.2 Optimal Answer Sets

Unlike hard constraints in ASP, *weak constraints* do not affect what is (or is not) an answer set of a program P. Hence Definition 2.1 also applies to programs with weak constraints. Weak constraints create an ordering over $AS(P)$ specifying which answer sets are "preferred" to others. At each priority level `lev` that appears in $weak(P)$, the aim is to discard any answer set that does not minimise the sum of the weights of the ground weak constraints with level `lev` whose bodies are true. The higher levels are minimised first. The terms in the tail of a weak constraint specify which ground weak constraints should be considered unique.

Definition 2.2. Let P be a ground program and let I be an interpretation. $weak(P, I)$ is the set:

$$\left\{ (\mathtt{wt}, \mathtt{lev}, \mathtt{t_1}, \ldots, \mathtt{t_n}) \left| \begin{array}{c} :\sim \mathtt{b_1}, \ldots, \mathtt{b_k}.[\mathtt{wt@lev}, \mathtt{t_1}, \ldots, \mathtt{t_n}] \in weak(P) \\ \mathtt{b_1}, \ldots, \mathtt{b_k} \text{ is each satisfied by } I \end{array} \right. \right\}$$

The *score* of I at a priority level `lev` (denoted $P^I_{\mathtt{lev}}$) is equal to the sum:

$$\sum_{(\mathtt{wt}, \mathtt{lev}, \mathtt{t_1}, \ldots, \mathtt{t_n}) \in weak(P,I), wt \in \mathbb{Z}} wt$$

Given a pair of interpretations I_1 and I_2, I_1 is said to *dominate* I_2 (denoted $I_1 \succ_P I_2$) if there is a priority level lev such that $P^{I_1}_{lev} < P^{I_2}_{lev}$ and there is no lev' such that $lev' > lev$ and $P^{I_1}_{lev'} > P^{I_2}_{lev'}$. An answer set A of a program P is said to be *optimal* if there is no answer set $A' \in AS(P)$ that dominates A. We write $AS^*(P)$ to denote the set of optimal answer sets of P.

Example 2.8. *Let P be the program $\{0\{\mathtt{p(1)}, \mathtt{p(2)}, \mathtt{p(3)}\}3.\}$. P has 8 answer sets, which are the various combinations of making each of the three* `p` *atoms true or false. Consider the two weak constraints* $:\sim \mathtt{p(X)}.[\mathtt{1@1}]$ *and* $:\sim \mathtt{p(X)}.[\mathtt{1@1}, \mathtt{X}]$. *The first weak constraint states that if any of the* `p` *atoms is true then a penalty of 1 must be paid. This penalty is only paid once, regardless of whether one, two or three of the* `p` *atoms are true. Conversely, the second weak constraint says that a penalty of 1 must be paid for each* `p` *atom that is true. In both cases, $\emptyset$ is the only optimal answer set; however, in the first case, none of the remaining answer sets dominate each other, whereas in the second case, the answer sets with only one* `p` *atom dominate those with two* `p` *atoms, which in turn each dominate the single answer set with three* `p` *atoms.*

We now introduce some extra notation which will be useful in later chapters. Given a set of interpretations S, the set $ord(P, S)$ captures the ordering of the interpretations given by the weak constraints in P. It generalises the *dominates* relation; so it not only includes $\langle A_1, A_2, < \rangle$ if $A_1 \succ_P A_2$, but it also includes tuples for other binary comparison operators. Formally, $\langle A_1, A_2, < \rangle \in ord(P, S)$ if $A_1, A_2 \in S$ and $A_1 \succ_P A_2$; $\langle A_1, A_2, > \rangle \in ord(P, S)$ if $A_1, A_2 \in S$ and $A_2 \succ_P A_1$; $\langle A_1, A_2, \leq \rangle \in ord(P, S)$ if $A_1, A_2 \in S$ and $A_2 \not\succ_P A_1$; $\langle A_1, A_2, \geq \rangle \in ord(P, S)$ if $A_1, A_2 \in S$ and $A_1 \not\succ_P A_2$; $\langle A_1, A_2, = \rangle \in ord(P, S)$ if

$A_1, A_2 \in S$, $A_1 \not\succ_P A_2$ and $A_2 \not\succ_P A_1$; $\langle A_1, A_2, \neq\rangle \in \mathit{ord}(P, S)$ if $A_1, A_2 \in S$ and $A_1 \succ_P A_2$ or $A_2 \succ_P A_1$. Given an ASP program, we write $\mathit{ord}(P)$ as a shorthand for $\mathit{ord}(P, AS(P))$.

2.2.3 Alternative Definitions of Answer Sets

The meta-level programs we introduce in this thesis use each of the components described so far; however, the programs learned by our algorithms take a slightly more restricted form. We introduce the *restricted* class $\mathcal{ASP}^R$ to denote the class of programs containing only normal rules, choice rules and hard and weak constraints (i.e. programs with no aggregates in the body and no disjunction). Unless stated otherwise, we assume all background knowledges and hypothesis spaces in this thesis to be $\mathcal{ASP}^R$ programs. We also use $\mathcal{ASP}^{ch}$ to denote the class of programs containing normal rules, choice rules and hard constraints – i.e. the $\mathcal{ASP}^R$ programs with no weak constraints. We now present two definitions of answer sets of $\mathcal{ASP}^R$ programs, which are both equivalent to the definition in [CFG+13].

Simplified Reduct for Choice Rules

In this section, we present our simplified semantics for programs in $\mathcal{ASP}^R$. This does not involve the translation of choice rules in Section 2.2.1 before solving; instead it uses an extended definition of reduct. This removes the need to "invent" extra helper ($\widehat{h_i}$) atoms that do not appear in the answer sets of a program. It also has the added benefit that the reduct is guaranteed to have exactly one minimal model.

Definition 2.3. The *simplified reduct* of a ground program P with respect to an interpretation I, is constructed in the following 4 (sequential) steps.

1. Remove any rule whose body contains `not a` for some $\mathtt{a} \in I$ and remove any negative literals from the remaining rules.
2. For any constraint R, $\texttt{:-}\mathit{body}(R)$, replace R with $\bot\texttt{:-}\mathit{body}^+(R)$ ($\bot$ is a new atom which cannot appear in any answer set of P).
3. For any choice rule R, $\mathtt{l}\{\mathtt{h_1}, \ldots, \mathtt{h_n}\}\mathtt{u}\texttt{:-}\mathit{body}(R)$ such that $\mathtt{l} \leq |I \cap \{\mathtt{h_1}, \ldots, \mathtt{h_n}\}| \leq \mathtt{u}$, replace R with the set of rules $\{\mathtt{h_i}\texttt{:-}\mathit{body}^+(R) \mid \mathtt{h_i} \in I \cap \{\mathtt{h_1}, \ldots, \mathtt{h_n}\}\}$.
4. For any remaining choice rule R, replace R with the constraint $\bot\texttt{:-}\mathit{body}^+(R)$.

Example 2.9. *Consider the program P.*

$$P = \left\{ \begin{array}{l} \mathtt{1\{p;q\}1\text{:-}\,r.} \\ \mathtt{r.} \\ \mathtt{\text{:-}\ not\ p, r.} \end{array} \right\}$$

The simplified reduct wrt the interpretation $I = \{\mathtt{p},\ \mathtt{r}\}$ is $\left\{\begin{array}{l}\mathtt{p\,:\text{-}\,r.}\\ \mathtt{r.}\end{array}\right\}$

The minimal models of the reduct is I, and hence, I is an answer set of P.

Consider now the alternative interpretation $I' = \{\mathtt{q},\ \mathtt{r}\}$.

The simplified reduct wrt I' is $\left\{\begin{array}{l}\mathtt{q\,:\text{-}\,r.}\\ \mathtt{r.}\\ \bot\,\mathtt{:\text{-}\,r.}\end{array}\right\}$

The minimal model of the reduct is: $\{\mathtt{q},\ \mathtt{r},\ \bot\}$. Hence, as $\bot \notin I'$, $I' \notin AS(P)$.

Consider the third interpretation $I'' = \{\mathtt{p},\ \mathtt{q},\ \mathtt{r}\}$.

The simplified reduct wrt I' is $\left\{\begin{array}{l}\bot\,\mathtt{:\text{-}\,r.}\\ \mathtt{r.}\end{array}\right\}$

The minimal model is $\{\mathtt{r},\ \bot\}$, and hence, as $\{\bot\} \notin I''$, $I'' \notin AS(P)$.

In contrast, the standard reduct of P wrt I'' is $\left\{\begin{array}{l}\mathtt{p} \vee \widehat{\mathtt{p}}\,\mathtt{:\text{-}\,r.}\\ \mathtt{q} \vee \widehat{\mathtt{q}}\,\mathtt{:\text{-}\,r.}\\ \mathtt{:\text{-}\,r}, \#\mathtt{count}\{\mathtt{p:p;q:q}\} < \mathtt{1}.\\ \mathtt{:\text{-}\,r}, \#\mathtt{count}\{\mathtt{p:p;q:q}\} > \mathtt{1}.\\ \mathtt{r.}\end{array}\right\}$

Similarly it can be seen that I'' is not a minimal model of the standard reduct, as it violates the second constraint.

Theorem 2.10. *([LRB15b]) Given any $\mathcal{ASP}^R$ program P, an interpretation I is in $AS(P)$ if and only if A is equal to the minimal model of the simplified reduct of P with respect to I.*

In the rest of this thesis, whenever P is an $\mathcal{ASP}^R$ program, we use P^I to denote the simplified reduct, rather than the standard reduct.

Unfounded Sets

We now recall an alternative definition of answer sets, based on the notion of unfounded sets [LRS97]. This alternative definition is central to the ILASP3 algorithm presented in Chapter 10.

Definition 2.4. Let P be a program and I be an interpretation. For any subset $U \subseteq I$, U is an unfounded subset of I wrt P iff there is no rule $R \in P$ that satisfies conditions 1-3.

1. $\exists \mathtt{h} \in \mathit{heads}(R)$ such that $\mathtt{h} \in U$
2. $\mathit{body}^+(R) \subseteq I \backslash U$

3. $body^-(R) \cap I = \emptyset$

The answer sets of a program can be defined as the models of the program which have no non-empty unfounded subsets. As the original definitions of unfounded sets do not consider choice rules, for completeness we prove in Lemma 2.11 that this result holds for $\mathcal{ASP}^R$ programs.

Lemma 2.11. *(proof on page 262)*

Let P be an $\mathcal{ASP}^R$ program and I be an interpretation. I is an answer set of P if and only if I is a model of P and there is no non-empty unfounded subset of I wrt P.

2.2.4 Useful Results on ASP

In this section, we present some results about ASP programs that are used throughout the thesis.

Lemma 2.12. *(proof on page 264)*

Let P be any ASP program, and C be an ASP program containing only hard constraints. $A \in AS(P \cup C)$ if and only if $A \in AS(P)$ and A does not satisfy the body of any instance of any constraint in C.

Another useful result that we use in the proofs in this thesis is the Splitting Set Theorem [LT94]. This theorem relies on the notions of a splitting set and the partial evaluation of a logic program. Given a ground disjunctive logic program P, a set $U \subseteq HB_P$ is a splitting set of P if and only if for every rule $R \in P$ such that $heads(R) \cap U \neq \emptyset$, $heads(R) \cup body^+(R) \cup body^-(R) \subseteq U$. Given a ground rule R and a set of atoms U, we write $R\backslash U$ to denote the rule R with all (positive or negative) occurrences of atoms in U removed from the body of R. Given a program P a splitting set U of P and a set $X \subseteq U$, the partial evaluation of P with respect to U and X, written $e_U(P, X)$, is the program $\{R\backslash U \mid R \in P, heads(R) \cap U = \emptyset, (body^+(R) \cap U) \subseteq X, body^-(R) \cap X = \emptyset\}$.

Theorem 2.13. *([LT94]) Given any ground ASP program P, and splitting set U of P:*

$$AS(P) = \left\{ X \cup Y \;\middle|\; \begin{array}{c} X \in AS(\{R \in P \mid heads(R) \cap U \neq \emptyset\}), \\ Y \in AS(e_U(P, X)) \end{array} \right\}$$

The intuition behind the splitting set theorem is that if a set of atoms U is known to *split* the program P, then we can find the answer sets of the subprogram that defines the atoms in U first. For each of these answer sets X, we can partially evaluate P using X and solve this partially evaluated program for answer sets. The splitting set theorem then guarantees that for each answer set Y of the partially evaluated program, $X \cup Y$ is an answer set of P. Furthermore, every answer set of P can be constructed in this way. Sometimes in our proofs, we use the following Corollary of the Splitting Set Theorem.

Corollary 2.14. Given any ground ASP program P, and splitting set U of P:

$$AS(P) = \left\{ Y \,\middle|\, \begin{array}{c} X \in AS(\{R \in P \mid heads(R) \cap U \neq \emptyset\}), \\ Y \in AS(\{R \in P \mid heads(R) \cap U = \emptyset\} \cup \{\mathtt{a.} \mid \mathtt{a} \in X\}) \end{array} \right\}$$

Grounding

Recall that the first step of solving any ASP program is to *ground* it. In principle, this means that a program P should be replaced with the set of all ground instances (within the language of P) of rules in P. In general, this grounding is infinite; however, there are some cases where even though the complete grounding is infinite, there is an equivalent subset of the grounding, called the *relevant* grounding, which is finite.

Definition 2.5. Let P be any ASP program. $f_P : \mathcal{P}(HB_P) \rightarrow \mathcal{P}(HB_P)$, is the function $f_P(I) = I \cup \{\mathtt{h} | R \in ground(P), body^+(R) \text{ is satisfied by } I, \mathtt{h} \in heads(R)\}$

Lemma 2.15. Consider any ASP program P and any two interpretations I_1 and I_2. If both I_1 and I_2 are fixpoints of f_P then $I_1 \cap I_2$ is a fixpoint of f_P.

Proof. Assume that I_1 and I_2 are fixpoints of f_P for some program P. Assume for contradiction that $\exists \mathtt{a} \in f_P(I_1 \cap I_2)$ such that $\mathtt{a} \notin I_1 \cap I_2$. There are two cases: either $\mathtt{a} \notin I_1$ or $\mathtt{a} \notin I_2$. Without loss of generality, assume that $\mathtt{a} \notin I_1$. As $I_1 = f_P(I_1)$, this means that $\nexists R \in ground(P)$ such that $\mathtt{a} \in heads(R)$ and $body^+(R) \subseteq I_1$. Hence, $\nexists R \in ground(P)$ such that $\mathtt{a} \in heads(R)$ and $body^+(R) \subseteq I_1 \cap I_2$. This means that $\mathtt{a}$ can not be in $f_P(I_1 \cap I_2)$, contradicting our initial assumption. $\square$

Corollary 2.16. Given any program P, f_P has exactly one least fixpoint.

Proof. HB_P is a fixpoint, so there must be at least one least fixpoint. Now assume for contradiction that both I_1 and I_2 are least fixpoints and I_2 is not equal to I_1. Then $I_1 \cap I_2$ is also a fixpoint, and it must be a strict subset of at least one of I_1 and I_2. This contradicts that both I_1 and I_2 are least fixpoints. Hence there is exactly one least fixpoint of f_P. $\square$

Given any program P, we call the least fixpoint of f_P the *relevant Herbrand base* of P, and denote it as HB_P^{rel}.

Definition 2.6. Let P be any $\mathcal{ASP}^R$ program. The *relevant grounding* of P, written $ground^{rel}(P)$ is the set of all rules $R \in ground(P)$ such that for each $\mathtt{a} \in body^+(R)$, $\mathtt{a} \in HB_P^{rel}$.

Theorem 2.17 shows that if HB_P^{rel} is finite, then $ground^{rel}(P)$ is also finite and $AS(ground^{rel}(P)) = AS(P)$. We will use this result extensively when proving termination of our algorithms, as any finite ground ASP program can be solved in a finite time.

Theorem 2.17. *(proof on page 265)*

Let P be any $\mathcal{ASP}^R$ program such that $|HB_P^{rel}|$ is finite.

1. *$|ground^{rel}(P)|$ is finite*
2. *$AS(P) = AS(ground^{rel}(P))$*

2.3 Computational Complexity

We assume the reader is familiar with the fundamental concepts of complexity, such as Turing machines and reductions; for a detailed explanation, see [Pap03].

Many of the decision problems for ASP are known to be complete for classes in the polynomial hierarchy [Sto76]. We now summarise the classes of the polynomial hierarchy. P is the class of all problems which can be solved in polynomial time by a Deterministic Turing Machine (DTM); $\Sigma_0^P = \Pi_0^P = \Delta_0^P = P$; $\Delta_{k+1}^P = P^{\Sigma_k^P}$ is the class of all problems which can be solved by a DTM in polynomial time with a Σ_k^P oracle; $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$ is the class of all problems which can be solved by a non-deterministic Turing Machine in polynomial time with a Σ_k^P oracle; and finally, $\Pi_{k+1}^P = \text{co-}NP^{\Sigma_k^P}$ is the class of all problems whose complement which can be solved by a non-deterministic Turing Machine in polynomial time with a Σ_k^P oracle. Σ_1^P and Π_1^P are NP and co-NP (respectively), where NP is the class of problems which can be solved by a non-deterministic Turing machine in polynomial time and co-NP is the class of problems whose complement is in NP.

DP is the class of problems that can be mapped to a pair of problems D_1 and D_2 such that $D_1 \in NP$ and $D_2 \in$ co-NP. It is well known that the following inclusions hold: $P \subseteq NP \subseteq DP \subseteq \Delta_2^P \subseteq \Sigma_2^P$ and $P \subseteq \text{co-}NP \subseteq DP \subseteq \Delta_2^P \subseteq \Pi_2^P$ [Pap03].

Complexity results for two classes of ASP programs are useful for later chapters. First we recall the definition of *aggregate stratification* from [FPL11]. We slightly simplify the definition by considering only ground programs such that every rule has an atom as its head. For any program fragment P, let $atoms(P)$ denote the set of all atoms that occur in P.

Definition 2.7. A ground program P, in which aggregates occur only in bodies of rules, is *stratified on an aggregate* `agg` if there is a level mapping $||\ ||$ from HB_P to ordinals, such that for each rule $R \in P$, the following holds:

1. $\forall \mathtt{b} \in atoms(body(R)) : ||\mathtt{b}|| \leq ||head(R)||$
2. If $\mathtt{agg} \in body(R)$, then for each atom $\mathtt{a} \in atoms(\mathtt{agg})$: $||\mathtt{a}|| < ||head(R)||$

P is said to be *aggregate stratified* if it is stratified on every aggregate in P.

The intuition is that aggregate stratification forbids recursion through aggregates. Aggregate stratification has nothing do with negation as failure, and therefore, whether a program is aggregate stratified is unrelated to whether it is *stratified* in the usual sense.

Note that constraints and choice rules can be added in to any aggregate stratified program without breaking stratification so long as no atoms in the head of the choice rule are on a lower level than any atom in the body. This is illustrated by the following example.

Example 2.18. *Any constraint* $\texttt{:-}\,\mathtt{b_1,\ldots,b_n}$, $\mathtt{not\ c_1,\ldots,}$ $\mathtt{not\ c_m}$ *can be rewritten as* $\mathtt{s\,:\text{-}\,b_1,\ldots,b_n}$, $\mathtt{not\ c_1,\ldots,}$ $\mathtt{not\ c_m}$, $\mathtt{not\ s}$ *where* $\mathtt{s}$ *is a new atom. For any mapping proving that the rest of the program is stratified on an aggregate,* $\mathtt{s}$ *can be added to the mapping, taking a higher level than any other atom.*

A choice rule $\mathtt{lb\{h_1,\ldots,h_o\}ub\,:\text{-}\,b_1,\ldots,b_n}$, $\mathtt{not\ c_1,\ldots,}$ $\mathtt{not\ c_m}$ *can be rewritten as:*

$\mathtt{h_1\,:\text{-}\,b_1,\ldots,b_n,\ not\ c_1,\ldots,\ not\ c_m,\ not\ h'_1.}$
$\mathtt{h'_1\,:\text{-}\,b_1,\ldots,b_n,\ not\ c_1,\ldots,\ not\ c_m,\ not\ h_1.}$
$\ldots$
$\mathtt{h_o\,:\text{-}\,b_1,\ldots,b_n,\ not\ c_1,\ldots,\ not\ c_m,\ not\ h'_o.}$
$\mathtt{h'_o\,:\text{-}\,b_1,\ldots,b_n,\ not\ c_1,\ldots,\ not\ c_m,\ not\ h_o.}$

$\mathtt{s\,:\text{-}\,b_1,\ldots,b_n,\ not\ c_1,\ldots,\ not\ c_m,\#count\{h_1:h_1;\ldots;h_n:h_n\} < lb,\ not\ s.}$
$\mathtt{s'\,:\text{-}\,b_1,\ldots,b_n,\ not\ c_1,\ldots,\ not\ c_m,\#count\{h_1:h_1;\ldots;h_n:h_n\} > ub,\ not\ s'.}$

where $\mathtt{h'_1,\ldots,h'_o,s,s'}$ *are all new atoms. Provided the rest of the program is aggregate stratified, then this new one is too. For any mapping that proves that the rest of the program is stratified on an aggregate, we can extend the mapping by assigning* $\mathtt{s}$ *and* $\mathtt{s'}$ *a new highest level and each* $\mathtt{h'_i}$ *the same level as* $\mathtt{h_i}$ *(if they do not occur in the rest of the program then they should be given a new level one below* $\mathtt{s}$ *and* $\mathtt{s'}$*). Note that the whole program can be shown to be stratified on the two new counting aggregates by using a level mapping that maps* $\mathtt{s}$ *and* $\mathtt{s'}$ *to 2 and every other atom to 1. Hence, the whole program is aggregate stratified, as it is stratified on every aggregate in the program. To avoid constantly using this mapping, we will refer to programs with choice rules and constraints as also being aggregate stratified.*

Lemma 2.19. [FPL11] Deciding whether an aggregate stratified propositional program without disjunction cautiously entails an atom is co-NP-complete.

Corollary 2.20. Deciding whether an aggregate stratified propositional program without disjunction bravely entails an atom is NP-complete.

Proof. We first show that deciding whether an aggregate stratified propositional program without disjunction bravely entails an atom is in NP. We do this by showing that there is a polynomial reduction from this problem to the complement of the problem in Lemma 2.19 (which by definition of co-NP must be in NP). The complement of the problem in Lemma 2.19 is deciding whether a non disjunctive aggregate stratified program does not cautiously entail an atom. Take any non-disjunctive aggregate stratified program P and any atom `a`. $P \models_b$ `a` if and only if $P \cup \{$`neg_a :- not a.`$\} \not\models_c$ `neg_a`. So the decision problem is in NP.

It remains to show that deciding whether an aggregate stratified propositional program without disjunction bravely entails an atom is NP-hard. We do this by showing that any problem in NP can be reduced in polynomial time to deciding the satisfiability of an aggregate stratified propositional program without disjunction.

Consider an arbitrary NP problem D. The complement of D, $\bar{D}$, must be in co-NP (by definition of co-NP). Hence, by Lemma 2.19, there is a polynomial reduction from $\bar{D}$ to deciding whether an aggregate stratified propositional program without disjunction cautiously entails an atom.

We define the polynomial reduction from D to deciding whether an aggregate stratified propositional program without disjunction bravely entails an atom as follows: for any instance I of D, let P and `a` be the program and atom given by the polynomial reduction from the complement of I to cautious entailment; define P' as the program $P \cup \{$`neg_a :- not a.`$\}$ (where `neg_a` is a new atom). I returns true if and only if $P \not\models_c$ `a` if and only if $P' \models_b$ `neg_a`. Hence, as P' is still aggregate stratified (the new atom `neg_a` can be mapped to the highest ordinal used by any level mapping that is used to prove that P is stratified on an aggregate), this is a polynomial reduction from D to deciding whether an aggregate stratified propositional program without disjunction bravely entails an atom. Hence, the decision problem is NP-hard.

$\square$

Lemma 2.21. [EG95] Deciding whether a disjunctive logic program has at least one answer set is Σ_2^P-complete.

We have now summarised the main background material on ASP that is necessary to understand the rest of the thesis. In the next chapter, we review the relevant literature on Inductive Logic Programming before presenting our own work.

Chapter 3

Inductive Logic Programming

Traditionally, Inductive Logic Programming has addressed the problem of learning Prolog programs. Often these programs have no negation as failure (i.e. they consist of definite clauses). In Section 3.1 we give a short overview of ILP, introducing the main frameworks and algorithms for learning. This overview is by no means complete, but serves as a means of comparison between traditional approaches and recent approaches to learning ASP programs. In Section 3.2 we then review the state of the art frameworks and algorithms for learning under the answer set semantics.

3.1 An Overview of ILP

3.1.1 Learning Frameworks

Research in ILP has mainly addressed three learning settings: *learning from entailment*, *learning from interpretations* and *learning from satisfiability*. In this section we present the definitions of each, and give examples of simple learning tasks.

We refer to a theoretical setting for ILP as a *learning framework*. We will usually write ILP_X to denote a framework X. A particular problem associated with a framework X is called a *learning task* of ILP_X and will be denoted T_X[1]. A learning task T_X will usually consist of a *background knowedge* B, which is a logic program, a *hypothesis space* S_M, defining the set of rules that are permitted to be in a *hypothesis* and some *examples*, the form of which varies between the learning frameworks. Every framework X will come with a definition of what it means for a hypothesis (a subset of the hypothesis space) to be an *inductive solution* of a task.

Notation ($ILP_X(T_X)$). Given any ILP framework ILP_X, and any ILP_X task T_X, $ILP_X(T_X)$ is the set of all inductive solutions of T_X.

[1]When it is clear which framework is being discussed, we will omit the X.

Throughout this thesis, given any framework ILP_X, learning task T_X and example e, we say that e is *covered* by a hypothesis if the hypothesis meets all conditions that T_X imposes on e.

Learning from Entailment

The most common framework for ILP is *Learning from Entailment* (LFE). The goal in learning from entailment is to find a hypothesis that (together with the background knowledge) entails each of a set of clauses called the positive examples, and which is consistent with a second set of clauses called the negative examples. Definition 3.1 formalises the Learning from Entailment framework.

Definition 3.1. A *Learning from Entailment* (ILP_{LFE}) *task* T is a tuple $\langle B, S_M, \langle E^+, E^- \rangle\rangle$ where B is a clausal theory, called the background knowledge, S_M is a set of clauses, called the hypothesis space and E^+ and E^- are sets of ground clauses, called the positive and negative examples, respectively. A hypothesis $H \subseteq S_M$ is an inductive solution of T if and only if:

1. $B \cup H \models E^+$
2. $B \cup H \cup E^- \not\models \bot$

As learning from entailment has been researched by many different groups, the language of the background knowledge, hypothesis space and examples differs between papers. In many cases (e.g. [Mug95], [RBR03] and [ML13]) the language of the background knowledge and hypothesis space is restricted to either definite or Horn clauses, whereas [RI07] allowed a language of full clausal theories as in Definition 3.1. In the common special case where both the background knowledge and hypothesis space consists of definite clauses and positive and (resp. negative) examples are positive (resp. negative) literals, $B \cup H$ is guaranteed to have a unique minimal Herbrand model M and H is an inductive solution of the task if and only if M includes all of the positive examples and does not include (the negation of) any negative examples (i.e. if $B \cup H$ entails each of the positive examples and does not entail (the negation of) any negative examples).

Example 3.1. *Consider the* ILP_{LFE} *task* $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$*, where:*

$$B = \left\{ \begin{array}{l} \texttt{parent(X,Y):-father(X,Y).} \\ \texttt{parent(X,Y):-mother(X,Y).} \\ \texttt{father(mike,mark).} \\ \texttt{mother(sue,mark).} \\ \texttt{father(howell,sue).} \\ \texttt{mother(norma,sue).} \\ \texttt{person(mark).} \\ \texttt{person(mike).} \\ \texttt{person(sue).} \\ \texttt{person(howell).} \\ \texttt{person(norma).} \end{array} \right\}$$

$$S_M = \left\{ \begin{array}{ll} \texttt{h}_1: & \texttt{grandfather(X,Y):-parent(X,Z),} \\ & \qquad\qquad\qquad\texttt{parent(Z,Y).} \\ \texttt{h}_2: & \texttt{grandfather(X,Y):-father(X,Z),} \\ & \qquad\qquad\qquad\texttt{parent(Z,Y).} \\ \texttt{h}_3: & \texttt{grandfather(X,Y):-mother(X,Z),} \\ & \qquad\qquad\qquad\texttt{parent(Z,Y).} \end{array} \right\}$$

$$E^+ = \left\{ \texttt{grandfather(howell,mark).} \right\}$$

$$E^- = \left\{ \begin{array}{l} \texttt{:- grandfather(howell,mike).} \\ \texttt{:- grandfather(norma,mark).} \end{array} \right\}$$

- $\emptyset \notin ILP_{LFE}(T)$*, as* B *does not entail* `grandfather(howell,mark)`.
- *For each* $H \in \{\{\texttt{h}_1\}, \{\texttt{h}_3\}, \{\texttt{h}_1, \texttt{h}_2\}, \{\texttt{h}_1, \texttt{h}_3\}, \{\texttt{h}_2, \texttt{h}_3\}, \{\texttt{h}_1, \texttt{h}_2, \texttt{h}_3\}\}$, $H \notin ILP_{LFE}(T)$*, as* $B \cup H$ *entails* `grandfather(norma,mark)`*, so* $B \cup H \cup \{\texttt{:- grandfather(norma,mark).}\}$ *is inconsistent.*
- $\{\texttt{h}_2\} \in ILP_{LFE}(T)$*, as* $B \cup \{\texttt{h}_2\}$ *entails* `grandfather(howell,mark)` *and is consistent with* `:- grandfather(howell,mark)` *and* `:- grandfather(norma,mark)`.

Learning from Interpretations

Another common ILP setting is *Learning from Interpretations* (LFI). Definition 3.2 formalises the ILP_{LFI} framework. In the LFI literature (e.g. [DRVL95]), a set of (possibly incomplete) facts is often called an interpretation. We avoid this usage of the term interpretation to avoid confusion with our own usage of the term.

Definition 3.2. A *Learning from Interpretations* (ILP_{LFI}) *task* T is a tuple $\langle B, S_M, \langle E^+, E^- \rangle\rangle$ where B is a definite clausal theory, S_M is a set of clauses and each element of E^+ and E^- is a set of facts. A hypothesis $H \subseteq S_M$ is an inductive solution of T if and only if:

1. $\forall e^+ \in E^+$: $M(B \cup e^+)$ satisfies H
2. $\forall e^- \in E^-$: $M(B \cup e^-)$ does not satisfy H

Note that with no background knowledge, this definition is equivalent to saying that each positive example (treated as an interpretation) must be a model of H, and no negative example should be a model of H. In ILP_{LFI} systems that allow background knowledge, the background knowledge in ILP_{LFI} is used to "complete" the interpretation, so that not all atoms need to be specified in the example.

Example 3.2. *Consider the* ILP_{LFI} *task* $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$*, where:*

$$B = \left\{ \texttt{father(X,Y):- male(X), parent(X,Y).} \right\}$$

$$E^+ = \left\{ \begin{array}{c} \left\{ \begin{array}{c} \texttt{parent(richard,lucy).} \\ \texttt{male(richard).} \end{array} \right\}, \\ \left\{ \begin{array}{c} \texttt{parent(mike,mark).} \\ \texttt{male(mike). male(mark).} \\ \texttt{son(mark,mike).} \end{array} \right\} \end{array} \right\}$$

$$S_M = \left\{ \begin{array}{ll} \texttt{h}_1: & \texttt{son(X,Y):- father(Y,X).} \\ \texttt{h}_2: & \texttt{son(X,Y):- parent(Y,X).} \\ \texttt{h}_3: & \texttt{son(X,Y):- parent(Y,X), male(X).} \end{array} \right\}
\quad E^- = \left\{ \left\{ \begin{array}{c} \texttt{parent(mike,mark).} \\ \texttt{male(mike). male(mark).} \end{array} \right\} \right\}$$

The minimal models of B combined with each example are M_1 = {`parent(richard, lucy)`, `male(richard)`, `father(richard, lucy)`}, M_2 = {`parent(mike, mark)`, `male(mike)`, `male(mark)`, `father(mike, mark)`, `son(mark, mike)`}, M_3 = {`parent(mike, mark)`, `male(mike)`, `male(mark)`, `father(mike, mark)`}, where M_1 and M_2 have been derived from the positive examples, and M_3 has been derived from the negative example.

The task is to find a hypothesis H such that M_1 and M_2 are both models of H and M_3 is not.

- $\emptyset \notin ILP_{LFI}(T)$ *as M_3 is a model of $\emptyset$.*
- $\{\mathtt{h_1}\} \notin ILP_{LFI}(T)$ *as M_1 is not a model of $\{\mathtt{h_1}\}$. This means that no hypothesis containing* $\mathtt{h_1}$ *can be an inductive solution of T.*
- $\{\mathtt{h_2}\} \notin ILP_{LFI}(T)$ *as M_1 is not a model of $\{\mathtt{h_2}\}$. This means that no hypothesis containing* $\mathtt{h_2}$ *can be an inductive solution of T.*
- $\{\mathtt{h_3}\} \in ILP_{LFI}(T)$ *as M_1 and M_2 are both models of $\{\mathtt{h_3}\}$ but M_3 is not.*

Learning from Satisfiability

The third common setting for ILP is *Learning from Satisfiability* (LFS) [DRD97b].

Definition 3.3. A *Learning from Satisfiability* (ILP_{LFS}) *task* T is a tuple $\langle B, S_M, \langle E^+, E^- \rangle\rangle$ where B is a clausal theory, S_M is a set of definite clauses and E^+ and E^- are sets of clausal theories. A hypothesis $H \subseteq S_M$ is an inductive solution of T if and only if:

1. $\forall e^+ \in E^+$: $B \cup H \cup e^+$ has at least one model
2. $\forall e^- \in E^-$: $B \cup H \cup e^-$ has no models

Example 3.3. *Consider the ILP_{LFS} task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$, where:*

$$B = \left\{\ \mathtt{p\,{:}\text{-}\,q, r.}\ \right\} \qquad E^+ = \left\{\ \{\mathtt{r.\ \ {:}\text{-}\,q.}\}\ \right\}$$

$$S_M = \left\{\ \mathtt{h_1:\ \ p.}\ \right\} \qquad E^- = \left\{\ \{\mathtt{{:}\text{-}\,p.\ \ {:}\text{-}\,q.}\}\ \right\}$$

- $\emptyset \notin ILP_{LFS}(T)$ *as* $B \cup \{\mathtt{{:}\text{-}\,p.\ \ {:}\text{-}\,q.}\}$ *is satisfiable ($\emptyset$ is a model).*
- $\{\mathtt{h_1}\} \in ILP_{LFS}(T)$ *as* $B \cup \{\mathtt{h_1}\} \cup \{\mathtt{r.\ \ {:}\text{-}\,q.}\}$ *is satisfiable, and* $B \cup \{\mathtt{h_1}\} \cup \{\mathtt{{:}\text{-}\,p.\ \ {:}\text{-}\,q.}\}$ *is unsatisfiable.*

In [DR97], it was shown that ILP_{LFS} can simulate ILP_{LFE} by using the fact that $P \models e$ if and only if $P \cup \neg e$ has no models. Example 3.4 shows how the task in Example 3.1 can be encoded as an ILP_{LFS} task.

Example 3.4. *Recall the ILP_{LFE} task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$ from Example 3.1 and consider the task T_{LFS} with the same background knowledge and hypothesis space and the following examples.*

$$E^+ = \left\{ \left\{ \begin{array}{l} \texttt{:-grandfather(howell,mike).} \\ \texttt{:-grandfather(norma,mark).} \end{array} \right\} \right\} \qquad E^- = \left\{ \{\texttt{:-grandfather(howell,mark).}\} \right\}$$

Note that the original negative examples have become a single positive example. The condition for the negative examples under ILP_{LFE} is that $B \cup H \cup E^-$ must be consistent (i.e. it must have at least one model). This is exactly the same as the condition for a positive example under ILP_{LFS}. The original positive examples have been negated, and added as negative examples. For example, the original task said that $B \cup H$ must entail `grandfather(howell,mark)`*, which is equivalent to saying that no model of $B \cup H$ could not contain* `grandfather(howell,mark)`*. This is equivalent to saying that $B \cup H \cup \{$*`:-grandfather(howell,mark).`*$\}$ must have no models.*

3.1.2 Learning Algorithms

This section concentrates on the algorithms proposed to solve the learning tasks of three frameworks presented in Section 3.1.1. In the rest of this thesis we refer to these frameworks and algorithms as *traditional ILP* in order to differentiate them from ASP-based approaches to ILP. Early algorithms for ILP were divided into two classes: *bottom-up* algorithms such as Progol [Mug95], Aleph [Sri01] and HAIL [RBR03, Ray05], which compute a most specific hypothesis to cover an example, and then generalise it; and *top-down* algorithms such as HYPER [Bra99, OB10], which first computes a most general hypothesis and then specialises it. More recently, a new class of *meta-level* algorithms, such as TAL [Cor12] and Metagol [ML13, MLPTN14], has emerged, which encode learning tasks as meta-level logic programs.

Algorithm	Learning Framework	Language of $B \cup H$	Category
Aleph [Sri01]	ILP_{LFE}	Definite clauses (in the standard mode)	Bottom-up
Claudien [DRD97a]	ILP_{LFI}	Clausal theories	Top-down
Claudien-Sat [DRD97b]	ILP_{LFS}	Clausal theories	Top-down
FOIL [Qui90]	ILP_{LFE}	Definite clauses	Top-down
HAIL [RBR03]	ILP_{LFE}	Definite clauses	Bottom-up
HYPER [Bra99]	ILP_{LFE}	Definite clauses	Top-down
ICN [MV96]	ILP_{LFE}	Normal clauses	Top-down
ICL [DRVL95]	ILP_{LFI}	Clausal theories	Top-down
ICL-Sat [DRD97b]	ILP_{LFS}	Clausal theories	Top-down
LOGAN-H [AKM07]	ILP_{LFI}	Horn clauses	Bottom-up
Metagol [ML13]	ILP_{LFE}	Definite clauses	Meta-level
Progol [Mug95]	ILP_{LFE}	Definite clauses	Bottom-up
Toplog [MSTN08]	ILP_{LFE}	Definite clauses	Top-down
TAL [CRL10]	ILP_{LFE}	Normal clauses	Meta-level

Table 3.1: A summary of some of the main algorithms for traditional ILP.

Table 3.1 gives a summary of some available systems along with the learning framework they support, the language of their background knowledge and hypothesis space and the category of the algorithm (bottom-up, top-down or meta-level). As the focus of this thesis is on learning ASP programs, we only go into detail on those algorithms that have influenced later algorithms for induction under the answer set semantics, or that are related to our own approach.

Many of the systems specify hypothesis spaces with a *language bias* consisting of *mode declarations* (which specify which atoms can appear in the head and in the body of rules in the hypothesis space). As the form of language bias varies for different systems, and to avoid confusion with our own language bias, we do not formalise mode declarations here (for an example of a common style of mode declaration, see [Mug95]).

Progol 5

The most well known ILP system is Progol, which solves Learning from Entailment tasks, with the background knowledge, hypothesis space and examples all restricted to Horn clauses. Progol uses a *cover loop*, sequentially picking an uncovered positive example e_i, called a *seed* example, and finding a hypothesis h_i which covers this example and which is consistent with all the negative examples. When all positive examples are covered, Progol terminates with the hypothesis $H = h_1 \cup \ldots \cup h_n$ (where $n \leq |E^+|$).

When processing a seed example e, Progol uses the principle of *Inverse Entailment* ($B \cup H \models e \Leftrightarrow B \cup \neg e \models \neg H$ [Mug95]) in a technique called *Bottom Generalisation.* The first step in Bottom Generalisation is to construct the *bottom clause*, which is the disjunction of all literals which are entailed by $B \cup \bar{e}$, where $\bar{e}$ is the complement of e[2]. Any Horn clause which subsumes the bottom clause is said to be *derivable from e by Bottom Generalisation.* For each seed example e, Progol 5 attempts to compute a clause that is compatible with the language bias, and is derivable from e by Bottom Generalisation.

Bottom Generalisation is not complete, in that it cannot learn multiple clauses from the same example.

Example 3.5. *Consider a background knowledge $B = \{\mathtt{p\,:\!-\,q, r.}\}$, and a seed example* `p`*. The hypothesis $H = \{\mathtt{q.\ \ r.}\}$ covers the example but neither of its two clauses are derivable by Bottom Generalisation – the bottom clause is $\neg\mathtt{p}$, meaning that the only clause that can be derived is "*`p.`*".*

In fact, in [RBR03, Ray05] it was shown that Progol 5 is not even complete with respect to the semantics of Bottom Generalisation. In [RBR03], the incompleteness of Progol 5 and the inability of Bottom Generalisation to learn multiple clauses from the same example was used to motivate the *HAIL* system.

[2]Given a Horn clause e, the complement $\bar{e}$ is the theory: $(\neg head(e) \wedge (\bigwedge_{l \in body(e)} l))\theta$ Where θ replaces each variable in e with a Skolem constant symbol.

Hybrid Abductive Inductive Learning (HAIL)

Kernel Set Subsumption[RBR04] is a generalisation of Bottom Generalisation.

Definition 3.4. [RBR04] Let B be a Horn theory and e be a ground atom, then a definite theory $\mathcal{K}$ is a *Kernel Set* of B and e if and only if $B \cup \{head(R) \mid R \in \mathcal{K}\} \models e$ and for each rule $R \in \mathcal{K}$, $\forall \mathtt{b} \in body(R)$, $B \models \mathtt{b}$.

Let $\mathcal{K}$ be a Kernel Set of a Horn theory B and e st $B \not\models e$. A definite theory H is *derivable by Kernel Set Subsumption* if and only if every clause in $\mathcal{K}$ is θ-subsumed by at least one clause in H.

Note that Kernel Set Subsumption extends Bottom Generalisation by allowing mutiple clauses to be learned from a single example. The *Hybrid Abductive Inductive Learning (HAIL)* algorithm [RBR04] computes hypotheses using a cover loop, using the Kernel Set Subsumption principle to compute new clauses for each seed example. For each seed example e, the procedure is split into 3 phases: an abductive phase, a deductive phase, and a search phase. In the abductive phase, a set of ground atoms Δ is found such that $B \cup \Delta \models e$, each element of Δ is compatible with at least one mode (head) declaration and $B \cup \Delta$ is consistent with all the examples. The atoms in Δ become the heads of the rules in the Kernel set. The deductive phase uses the background knowledge (and the current hypothesis, computed in previous iterations) to deduce the ground atoms that are allowed to appear in the body of rules in the Kernel Set (and which are compatible with the mode body declarations). The search phase then finds the most compressed (shortest) set of clauses that clausally subsumes K (and which is consistent with the full set of examples). This set of clauses is then added to the background knowledge and the next uncovered example is selected as the seed example.

TAL

Top-directed Abductive Learning (TAL) is a meta-level learning technique proposed in [CRL10], which solves an ILP task by mapping it to an equivalent Abductive Logic Programming (ALP) task. The idea is that every rule h in the hypothesis space can be represented by a unique meta-level atom Δ_h. The ILP task $T = \langle B, M, E^+, E^- \rangle$ (where M is a set of mode declarations, rather than a full hypothesis space) is then transformed into a Prolog program T_{meta}, which can be used to find the inductive solutions of T. Each hypothesis H is an inductive solution of T if and only if $T_{meta} \cup \{\Delta_{h_i} \mid h_i \in H\}$ entails the positive examples and does not entail any negative examples. This means that the original ILP task can be represented as an abductive task, where the goal is to *abduce* a minimal subset of the possible Δ_h atoms in order to cover the examples.

Originally this technique was implemented as *TAL* which achieved nonmonotonic ILP under the three valued Fitting semantics [Fit02]. More interestingly (as this thesis addresses learning under the answer set semantics) this same technique, with a different meta-representation, was then implemented under the answer set semantics as *ASPAL* [CRL12]. ASPAL is discussed in greater detail in Section 3.2.2.

Algorithms for Learning from Interpretations and Learning from Satisfiability

Inductive Constraint Logic (ICL) [DRVL95] solves a learning from interpretations task with both positive and negative examples. It iteratively computes a hypothesis, searching in each iteration for a clause that is satisfied by each positive example e^+ (or by $M(B \cup e^+)$ in the case that a background knowledge B is given) and fails to satisfy at least one negative example that is not yet covered. The clauses are constructed using a top-down refinement operator (from general to specific). This approach can be thought of as the reverse of what many learning from entailment algorithms do. For example Progol searches, in each iteration, for a clause that satisfies at least one positive example (the seed example) and does not satisfy any negative examples. Progol also searches in a bottom-up fashion (from specific to general).

Claudien [DRD97a] is a top-down system for learning from interpretations. Unlike many ILP systems, it does not search for a compressed hypothesis, but instead searches for a maximally specific hypothesis that is consistent with a set of positive interpretation examples. This setting is called *characteristic learning from interpretations*. Claudien begins with an initial hypothesis $H = \emptyset$ and a set of clauses $Q = \{\bot\}$. In each iteration it processes each clause in $c \in Q$ as follows: if each example interpretation is a model of c, then c is removed from Q and added to H; if not, then c is replaced in Q with every possible maximally general refinement of c. As soon as Q is empty, H is guaranteed to be a maximally specific hypothesis (in that it accepts as few models as possible). [DRD97a] also presented a parallel version of Claudien, which is made possible because the clauses in Q are independent from each other.

In [DRD97b], ICL and Claudien were extended to ICL-Sat and Claudien-Sat, respectively, in order to allow them to solve learning from satisfiability tasks. As clauses in the hypothesis space cannot be considered independently for learning from satisfiability (in general $h_1 \cup e \not\models \bot$ and $h_2 \cup e \not\models \bot$ does not imply that $h_1 \cup h_2 \cup e \not\models \bot$), both algorithms must be slightly modified. ICL-Sat's search no longer builds a single hypothesis as in the ICL algorithm, but builds a set Q of tuples of the form $\langle H, N, c\rangle$, where H is a hypothesis, N is a set of negative examples which are not covered by N and c is a single clause which can be refined. In each iteration, for each tuple $\langle H, N, c\rangle \in Q$, if each positive example is consistent with $H \cup c$ and at least one negative example is not consistent with $H \cup c$ then $\langle H, N, c\rangle$ is replaced in Q by $\langle H \cup \{c\}, N', \bot\rangle$ (where N' is the subset of examples in N that are not covered by $H \cup \{c\}$); if not, then $\langle H, N, c\rangle$ is replaced in Q with every tuple $\langle H, N, c'\rangle$ such that c' is a maximally general refinement of c. If the algorithm encounters a tuple $\langle H, N, c\rangle$ such that $N = \emptyset$ then H is an inductive solution of the task, and the algorithm terminates. Claudien-Sat is modified such that a clause c is added to H when $H \cup c$ is consistent with each example interpretation, rather than when each interpretation is a model of c. The more important difference is that Claudien-Sat can not be parallelised in the same way as Claudien, as the clauses can no longer be considered independently.

3.2 Approaches to ILP Under the Answer Set Semantics

3.2.1 Learning Frameworks

In ASP, there can be one, many or even no answer sets of a program. This leads to two different standard notions of entailment under the answer set semantics: *brave entailment* and *cautious entailment.* Recall from Chapter 2 that an atom `a` is bravely (resp. cautiously) entailed by a program P iff at least one (resp. every) answer set of P contains `a`.

These two different notions of entailment naturally lead to two different frameworks for learning from entailment under the answer set semantics: *brave induction* and *cautious induction.* Early approaches to ILP under the answer set or stable model semantics tended to adopt cautious induction[3] [IK97, SBP00, Sak00], as this is closer to standard learning from entailment, where examples must be covered in every model. In [SI09], it was argued that in some cases cautious induction can be too strong, and that in those cases a weaker form of induction – brave induction – is needed. It was in [SI09] that the notions of *brave* and *cautious* induction were first defined.

Some of the frameworks in this section were originally presented with no hypothesis space, as they were considered only theoretically. Hypothesis spaces are, however, an integral part of many practical ILP algorithms. For example, although brave induction was originally presented with no hypothesis space, every publicly available system for brave induction requires a hypothesis space to function. The definitions of learning frameworks in this section are therefore reformulations of the original definitions, extended with hypothesis spaces and defined over the language of ASP considered in this thesis.

Cautious Induction

Cautious induction (ILP_c), first presented in [SI09], defines a learning task in which all examples should be covered in every answer set (i.e. entailed under cautious entailment in ASP) and $B \cup H$ should be satisfiable (have at least one answer set)[4].

Definition 3.5. A cautious induction (ILP_c) task T_c is a tuple $\langle B, S_M, \langle E^+, E^- \rangle\rangle$, where B is an ASP program, S_M is a set of ASP rules and E^+ and E^- are sets of ground atoms. A hypothesis $H \subseteq S_M$ is an inductive solution of T_c if and only if $AS(B \cup H) \neq \emptyset$ and $\forall A \in AS(B \cup H)$, $E^+ \subseteq A$ and $E^- \cap A = \emptyset$.

Example 3.6. *Consider the* ILP_c *task* $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$*, where:*

[3]As the notions had not been defined at the time, they did not call it cautious induction, but the definitions are the same.

[4]The original definitions of brave and cautious induction did not consider atoms which should not be present in an answer set (negative examples). Publicly available algorithms that realise brave induction, on the other hand, do allow for negative examples. We therefore upgrade the definitions in this thesis to allow negative examples. Note that a negative example `e` can be easily simulated by adding a rule `a:- not e` to the background knowledge and giving `a` as a positive example (where `a` is a new atom that does not appear anywhere in the original task).

$$B = \left\{ \begin{array}{l} \texttt{bird(X):-penguin(X).} \\ \texttt{bird(X):-sparrow(X).} \\ \texttt{penguin(b1).} \\ \texttt{sparrow(b2).} \end{array} \right\} \qquad S_M = \left\{ \begin{array}{ll} \texttt{h}_1: & \texttt{flies(X):-bird(X).} \\ \texttt{h}_2: & \texttt{flies(X):-bird(X),} \\ & \quad\texttt{not penguin(X).} \\ \texttt{h}_3: & \texttt{0\{flies(X)\}1:-bird(X).} \\ \texttt{h}_4: & \texttt{0\{flies(X)\}1:-bird(X),} \\ & \quad\texttt{not penguin(X).} \end{array} \right\}$$

$$E^+ = \{\texttt{flies(b2)}\}$$

$$E^- = \{\texttt{flies(b1)}\}$$

- *$\emptyset \notin ILP_c(T)$ as B has exactly one answer set, and it does not contain* `flies(b2)`*.*
- *$\{\texttt{h}_1\} \notin ILP_c(T)$ as $B \cup \{\texttt{h}_1\}$ has exactly one answer set, and it contains* `flies(b1)`*.*
- *$\{\texttt{h}_2\} \in ILP_c(T)$ as $B \cup \{\texttt{h}_2\}$ has exactly one answer set, and it contains* `flies(b2)` *but does not contain* `flies(b1)`*.*
- *$\{\texttt{h}_3\}$ and $\{\texttt{h}_4\}$ are not in $ILP_c(T)$, as they both have answer sets (when combined with B) that do not cover the examples.*

Enforcing that examples are covered in every answer set is sometimes too *strong* a requirement, as shown in Example 3.7.

Example 3.7. *Consider the background knowledge $B = \emptyset$ and the hypothesis space S_M:*

$$S_M = \left\{ \begin{array}{ll} \texttt{h}_1: & \texttt{p:- not q.} \\ \texttt{h}_2: & \texttt{q:- not p.} \end{array} \right\}$$

There is no ILP_c task T, with background knowledge B and hypothesis space S_M such that $\{\texttt{h}_1, \texttt{h}_2\}$ is a solution of T and $\emptyset$ is not. This can be seen as follows. There are only two atoms (`p` *and* `q`*) in the Herbrand base of $B \cup S_M$. There are therefore only two atoms which would be meaningful as examples. Neither can be given as a positive example as for each atom there is an answer set of $B \cup \{\texttt{h}_1, \texttt{h}_2\}$ that does not contain it. Similarly, neither can be given as a negative example as for each atom there is an answer set that contains it. This means that the only ILP_c task T (without introducing extra redundant negative examples that are outside the Herbrand base) such that $\{\texttt{h}_1, \texttt{h}_2\} \in ILP_c(T)$ is $\langle B, S_M, \langle \emptyset, \emptyset \rangle \rangle$. As there are no examples in T, $\emptyset$ is clearly also an inductive solution of T. As in practice ILP systems search for the shortest possible hypothesis, and no hypothesis is shorter than $\emptyset$, this means that no examples can be given such that a cautious induction system would return $\{\texttt{h}_1, \texttt{h}_2\}$.*

Brave Induction

Brave induction (*ILP_b*) was also formalised in [SI09]. It defines an inductive task where all of the examples should be covered in at least one answer set (i.e. entailed under brave entailment in ASP). Note that there should be at least one answer set that covers every example (rather than at least one answer set for each example).

Definition 3.6. A brave induction (ILP_b) task T_b is a tuple $\langle B, S_M, \langle E^+, E^- \rangle\rangle$, where B is an ASP program, S_M is a set of ASP rules and E^+ and E^- are sets of ground atoms. A hypothesis $H \subseteq S_M$ is an inductive solution of T_b if and only if $\exists A \in AS(B \cup H)$ such that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$.

Example 3.8. *Consider the ILP_b task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$, where:*

$$B = \left\{ \begin{array}{l} \texttt{bird(X):-penguin(X).} \\ \texttt{bird(X):-sparrow(X).} \\ \texttt{penguin(b1).} \\ \texttt{sparrow(b2).} \end{array} \right\}$$

$$E^+ = \{\texttt{flies(b2)}\}$$

$$E^- = \{\texttt{flies(b1)}\}$$

$$S_M = \left\{ \begin{array}{ll} \texttt{h}_1: & \texttt{flies(X):-bird(X).} \\ \texttt{h}_2: & \texttt{flies(X):-bird(X),} \\ & \qquad \texttt{not penguin(X).} \\ \texttt{h}_3: & \texttt{0\{flies(X)\}1:-bird(X).} \\ \texttt{h}_4: & \texttt{0\{flies(X)\}1:-bird(X),} \\ & \qquad \texttt{not penguin(X).} \end{array} \right\}$$

- $\emptyset \notin ILP_b(T)$ *as B has exactly one answer set, and it does not contain* flies(b2).
- $\{\texttt{h}_1\} \notin ILP_b(T)$ *as $B \cup \{\texttt{h}_1\}$ has exactly one answer set, and it contains* flies(b1).
- $\{\texttt{h}_2\}, \{\texttt{h}_3\}, \{\texttt{h}_4\} \in ILP_b(T)$ *as each of $B \cup \{\texttt{h}_2\}$, $B \cup \{\texttt{h}_3\}$ and $B \cup \{\texttt{h}_4\}$ has the answer set* {penguin(b1), sparrow(b2), bird(b1), bird(b2), flies(b2)}, *which contains* flies(b2) *but does not contain* flies(b1).

Brave induction can only reason about what should be true in at least one answer set of a program. It cannot reason about what should be true in all answer sets of a program. For this reason, brave induction is incapable of learning constraints. Any hypothesis that contains a constraint and is a solution of an ILP_b task T is still a solution of T if the constraint is removed.

Example 3.9. *Consider the background knowledge $B = \{0\{p\}1.\}$ and a hypothesis space S_M, containing only the constraint* :-p. *There is no ILP_b task T, with background knowledge B and hypothesis space S_M such that* {:-p.} *is a solution of T and $\emptyset$ is not. This can be seen as follows. There is only one atom (*p*) in the Herbrand base of $B \cup S_M$. There is therefore one atom which would be meaningful as an example. It must be given as a negative example (as $B \cup$* {:-p.} *has only one answer set, and it does not contain* p*). But $B \cup \emptyset$ also covers this negative example (as it also has the answer set $\emptyset$, which does not contain* p*). Therefore for any ILP_b task such that* {:-p.} *is a solution, $\emptyset$ is also a solution, meaning that in practice brave induction systems (searching for the shortest hypothesis) will not return the constraint.*

Furthermore, brave induction cannot specify other brave learning tasks such as enforcing that two atoms are both bravely entailed, but not necessarily in the same answer set (as brave induction requires all examples to be covered in the same answer set).

Induction of Stable Models

Induction of stable models [Ote01] (ILP_{sm}), generalises ILP_b, in order to allow conditions to be set over multiple answer sets. Its examples are *partial interpretations*.

Definition 3.7. A *partial interpretation* e is a pair of sets of atoms $\langle e^{inc}, e^{exc}\rangle$, we refer to e^{inc} and e^{exc} as the *inclusions* and *exclusions* respectively. An interpretation I is said to *extend* e if and only if $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$.

Example 3.10. *Consider the partial interpretation* $e = \langle\{\mathtt{p},\ \mathtt{q}\}, \{\mathtt{r},\ \mathtt{s}\}\rangle$.

- $\{\mathtt{p}\}$ *does not extend* e, *as it does not contain* q.
- $\{\mathtt{p},\ \mathtt{q},\ \mathtt{r}\}$ *does not extend* e, *as it contains* r.
- $\{\mathtt{p},\ \mathtt{q}\}$ *extends* e, *as it contains all of* e*'s inclusions, and none of* e*'s exclusions.*
- $\{\mathtt{p},\ \mathtt{q},\ \mathtt{t}\}$ *extends* e, *as it contains all of* e*'s inclusions, and none of* e*'s exclusions.*

Induction of stable models is formalised in Definition 3.8.

Definition 3.8. An induction of stable models (ILP_{sm}) task T_{sm} is a tuple $\langle B, S_M, \langle E\rangle\rangle$, where B is an ASP program, S_M is the hypothesis space and E is a set of example partial interpretations. A hypothesis H is an inductive solution of T_{sm} if and only if $H \subseteq S_M$ and $\forall e \in E$, $\exists A \in AS(B \cup H)$ such that A extends e.

Note that a brave induction task can be thought of as a special case of induction of stable models (with $|E| = 1$ and the inclusions and exclusions of the only partial interpretation example being the positive and negative examples of the brave task, respectively).

Example 3.11. *Consider the* ILP_{sm} *task* $T = \langle B, S_M, \langle E^+, E^-\rangle\rangle$, *where:*

$$B = \emptyset \qquad E = \left\{ \begin{array}{l} \langle\{\mathtt{p}\}, \{\mathtt{q}\}\rangle, \\ \langle\{\mathtt{q}\}, \{\mathtt{p}\}\rangle \end{array} \right\}$$

$$S_M = \left\{ \begin{array}{ll} \mathtt{h_1}: & \mathtt{p\text{:- not q.}} \\ \mathtt{h_2}: & \mathtt{q\text{:- not p.}} \end{array} \right\}$$

$\{\mathtt{h_1},\ \mathtt{h_2}\}$ *is the only subset of the hypothesis space that is an inductive solution of* T, *as it is the only hypothesis that has answer sets that extend both of the examples.*

Induction from Answer Sets

Before brave and cautious induction, Sakama put forward a different setting: *Induction from Answer Sets* [Sak05]. Most of [Sak05] focuses on algorithms for induction from single examples. For a positive example the setting aims to find a rule R such that $B \cup R \models_c e^+$, and for a negative example it aims to find a rule R such that $B \cup R \not\models_c e^-$. In [Sak05], it is shown that although in some cases the proposed algorithms can solve tasks with multiple examples, they do not work in general for tasks that contain both positive and negative examples. We therefore consider these cases as two separate settings: ILP_{IAS+} (for positive examples) and ILP_{IAS-} (for negative examples). ILP_{IAS+} is essentially cautious induction (although it is restricted to learn a single rule from each example); whereas ILP_{IAS-} is very similar to brave induction, but with the condition negated. An ILP_{IAS-} task $\langle B, S_M, \langle\{\mathtt{e_1}, \ldots, \mathtt{e_n}\}\rangle\rangle$ could be represented as an ILP_{sm} task $\langle B, S_M, \langle\{\langle\emptyset, \{\mathtt{e_1}\}\rangle, \ldots, \langle\emptyset, \{\mathtt{e_n}\}\rangle\}\rangle\rangle$. Note, this is different to the brave task $\langle B, S_M, \langle\emptyset, E\rangle\rangle$ as the latter requires that there is a single answer set that does not contain any of the examples, whereas the former only requires that there is an answer set for each example that does not contain the example. As the tasks of both frameworks can be easily translated into tasks of other frameworks, we do not consider them any further.

3.2.2 Learning Algorithms

In this section, we present the existing systems for learning ASP programs. Early algorithms [IK97, SBP00, Sak00] for learning ASP programs did so under a cautious induction style task[5]. In [Sak05], together with the two Induction from Answer Sets frameworks, two learning algorithms were presented. The IAS^{pos} algorithm solves the ILP_{IAS+}; whereas the IAS^{neg} algorithm solves an $ILP_{IAS}-$ task. Both algorithms, however, are only proven to be sound for tasks with multiple examples when programs are *categorical* ($B \cup H$ must have exactly one answer set). This is a severe restriction when learning ASP programs, as in general programs can have many answer sets. The state of the art ILP systems that operate under the answer set semantics, summarised in Table 3.2, aim to solve brave induction tasks. We now review these systems.

Algorithm	Learning Framework	Language of $B \cup H$	Category
ASPAL [CRL12]	ILP_b	Normal ASP programs	Meta-level
ILED [KAP15]	ILP_b	Normal ASP programs	Bottom-up
RASPAL [ACBR13]	ILP_b	Normal ASP programs	Meta-level
XHAIL [BR15b]	ILP_b	Normal ASP programs	Bottom-up

Table 3.2: The main systems for ASP-based ILP

XHAIL

eXtended Hybrid Abductive Inductive Learning (XHAIL) [Ray09] generalises the HAIL algorithm in order to solve ILP_b tasks.

Similarly to HAIL, XHAIL computes solutions in 3 phases: an abductive phase; a deductive phase; and an inductive phase. In the abductive phase, XHAIL finds a set of atoms Δ such that $B \cup \Delta \models_b$

[5]Many of these papers predate the term *cautious induction*, which first appeared in [SI09].

$(\bigwedge E^+) \wedge (\bigwedge\{\texttt{ not e} \mid \texttt{e} \in E^-\})$. The atoms in Δ that are ground instances of at least one atom that conforms with some `modeh` declaration become the heads of (ground instances of) rules in the final hypothesis.

Next, in the deductive phase, XHAIL finds the set of all ground literals that could go in the body of rules in the hypothesis. Each of these body atoms b is such that $B \cup \Delta \models_b \texttt{b}$ and `b` is a ground instance of an atom that conforms to at least one `modeb` declaration. The sets of ground rules with the heads from Δ with bodies consisting of literals computed in the deductive phase is referred to as the Kernel Set $\mathcal{K}$.

Example 3.12. *(from [Ray09])*

Consider the ILP_b task $T = \langle B, S_M, \langle E^+, E^-\rangle\rangle$, where B, M, E^+ and E^- are as follows:

$$B = \left\{ \begin{array}{l} \texttt{bird(X):-penguin(X).} \\ \texttt{bird(a).} \\ \texttt{bird(b).} \\ \texttt{bird(c).} \\ \texttt{penguin(d).} \end{array} \right\} \qquad M = \left\{ \begin{array}{l} \texttt{\#modeh(flies(+bird))} \\ \texttt{\#modeb(penguin(+bird))} \\ \texttt{\#modeb(not penguin(+bird))} \end{array} \right\}$$

$$E^+ = \left\{ \begin{array}{l} \texttt{flies(a),} \\ \texttt{flies(b),} \\ \texttt{flies(c)} \end{array} \right\} \qquad E^- = \left\{ \texttt{ flies(d) } \right\}$$

The mode declarations in the task express which predicates can be used in the head and body of the rules in the hypothesis space (the `#modeh`*'s and* `#modeb`*'s, respectively). The arguments of each of these particular mode declarations are all of the form* `+type`*, meaning that the arguments of the corresponding atoms in the rules must all be* input *variables. There are no placeholders for* output *variables (which would be denoted −type) or constants (which would be denoted #type). The restriction on variables is that every input variable in a rule must either occur in the head of the rule, or as an output variable, earlier in the rule.*

One abductive explanation of the examples is $\Delta = \{$`flies(a)`*,* `flies(b)`*,* `flies(c)`*$\}$.*

This leads to the Kernel:

$$\mathcal{K} = \left\{ \begin{array}{l} \texttt{flies(a):- not penguin(a).} \\ \texttt{flies(b):- not penguin(b).} \\ \texttt{flies(c):- not penguin(c).} \end{array} \right\}$$

Note that although there are other potential body literals that are entailed by $B \cup \Delta$ and that do conform to the mode declarations, they are not added to the Kernel as they could not form part of a ground instance of a rule that conforms to the mode delcarations. For example, `penguin(d)` *is not added to any of the three rules, as* `penguin(X)` *can only occur in the body of a rule with* `X` *as an input variable, meaning that* `X` *must occur elsewhere in the rule. In the case of the ground Kernel, this means that* *for* `penguin(d)` *to occur in the body of the rule,* `d` *must occur elsewhere in the rule, which is not the case for any of the three rules.*

The final step of the XHAIL algorithm – the *inductive* step – is to compute a hypothesis that conforms to the mode declarations, subsumes the Kernel and bravely entails the examples. The Kernel is mapped into an abductive task, which is represented as an ASP program. The answer sets of this ASP program can then be mapped to inductive solutions of the task.

Example 3.13. *The first rule in the Kernel produced in Example 3.12 is transformed into the ASP rules:*

$$\left\{\begin{array}{l}\texttt{flies(X):-use(1,0),try(1,1,X).}\\ \texttt{try(1,1,X):-bird(X), not use(1,1).}\\ \texttt{try(1,1,X):-bird(X), not penguin(X).}\end{array}\right\}$$

The first and second arguments of each of the meta-level atoms `use` *and* `try` *indicate a unique identifier for the object-level rule and literal (respectively). So,* `use(1,0)` *means that the head atom* `flies(X)` *is being* used *(i.e. it is in the hypothesis). The* `try` *atoms are for testing whether the rule body is satisfied. If the head is being used, then* `flies(X)` *is true in two cases: (1), the literal* `not penguin(X)` *is not in the hypothesis (indicated by the first* `try` *rule); or (2),* `not penguin(X)` *is true (represented by the second* `try` *rule).*

By using a choice rule, containing the various possible `try` *and* `use` *atoms, together with the transformation of each of the rules in the Kernel, XHAIL uses an ASP solver to compute the minimal hypothesis that subsumes the Kernel, conforms to the mode declarations and bravely entails the examples.*

One major difference between HAIL and XHAIL is that HAIL uses a cover loop approach, whereas XHAIL does not. This is due to the nonmonotonicity of negation as failure: in a cover loop approach, examples that were covered in previous iterations of the cover loop may not be covered in future iterations.

As in general there are many possible abductive solutions Δ, and not all Δ's lead to inductive solutions, XHAIL employs an iterative deepening approach, ordering the Δ's by size and terminating after processing the shortest Δ that leads to a solution. In general, this may not lead to the optimal solution being found, as there may be a large Δ that leads to a shorter hypothesis (e.g. with more individual rules, but fewer overall literals).

ILED ILED [KAP15] is an incremental algorithm, based on XHAIL. It is targeted at learning Event Calculus [KS86] theories, and therefore, its examples are slightly different in that they are grouped into *time windows*. The examples are processed one at a time and at each timepoint the hypothesis is revised so that it covers all examples in all windows that have been processed so far.

ILED has been shown to be much more scalable than XHAIL when processing large numbers of examples divided into time windows [KAP15]. On the other hand, like XHAIL, ILED is not guaranteed to find the optimal solution of a task. In fact, this incompleteness with respect to optimal solutions is more severe in ILED than in XHAIL, as it can also occur because of the incremental nature of the algorithm. Although at each step the revision may be optimal, the combination of every revision may result in a longer hypothesis than could have been found if all examples had been processed together.

ASPAL

The ASPAL [CRL12] algorithm is based on the TAL approach of converting an ILP task to a meta-level logic program, but the key difference of ASPAL (compared to TAL) is that ASPAL's meta-level program is an ASP program.

Given an ILP_b task $T_b = \langle B, S_M, \langle E^+, E^-\rangle\rangle$, where S_M is defined by a given set of mode declarations M, the first step is to compute a set of *skeleton rules* Sk. These are the set of rules R, such that there is an $R' \in S_M$, where each constant in R' is replaced by a variable in R.

Example 3.14. *Consider the mode declarations M.*

$$M = \left\{ \begin{array}{l} \texttt{\#modeh(penguin(+bird))} \\ \texttt{\#modeb(2, not can(+bird, \#ability))} \end{array} \right\}$$

The first argument of the mode body declaration is called the recall *and it expresses a constraint that this mode declaration can be used at most twice per rule in the hypothesis space. There are three skeleton rules:*

$$Sk = \left\{ \begin{array}{l} \texttt{penguin(X):-bird(X)} \\ \texttt{penguin(X):-bird(X), not can(X,C1)} \\ \texttt{penguin(X):-bird(X), not can(X,C1), not can(X,C2)} \end{array} \right\}$$

S_M consists of the rules in Sk where $C1$ and $C2$ have been replaced with constants of type `ability`.

Each skeleton rule R is then associated with a unique meta-level atom $\texttt{rule}(\texttt{R}_{\texttt{id}}, \texttt{C}_1, \ldots, \texttt{C}_\texttt{n})$, where $\texttt{C}_1, \ldots, \texttt{C}_\texttt{n}$ are the "constant placeholder" variables in R. For each rule $R \in Sk$, R_{meta} denotes the meta-level atom that represents R. For each rule $R' \in S_M$, we similarly write R'_{meta} to denote the ground atom representing R' (where each "constant placeholder" variable has been replaced with a constant of the correct type).

Definition 3.9. Let T be the ILP_b task $\langle B, S_M, \langle\{\texttt{e}^+_1, \ldots, \texttt{e}^+_\texttt{n}\}, \{\texttt{e}^-_1, \ldots, \texttt{e}^-_\texttt{m}\}\rangle\rangle$, where S_M is characterised by the set of mode declarations M. Let Sk be the set of skeleton rules derivable from M. The ASPAL meta-representation is the program consisting of the following components:

- B
- $\texttt{h:-b}_1, \ldots \texttt{b}_{\texttt{rl}}, \texttt{R}_{\texttt{meta}}$, for each rule $R \in Sk$, where R is the rule $\texttt{h:-b}_1, \ldots, \texttt{b}_{\texttt{rl}}$.

- A choice rule $\texttt{0\{ab}_\texttt{1},\ldots,\texttt{ab}_\texttt{k}\texttt{\}n.}$, where $\{\texttt{ab}_\texttt{1}, \ldots, \texttt{ab}_\texttt{k}\} = \{R_{meta} \mid R \in S_M\}$[6]
- The rule $\texttt{goal :- e}_\texttt{1}^+, \ldots, \texttt{e}_\texttt{n}^+\texttt{, not e}_\texttt{1}^-, \ldots\texttt{, not e}_\texttt{m}^-\texttt{.}$
- The constraint `:- not goal.`

We refer to the answer sets of this meta representation as meta-level answer sets, and the answer sets of $B \cup H$ as object-level answer sets. Each meta-level answer set A represents a single hypothesis H (defined by the `rule` atoms in A). Each meta-level answer set also contains exactly one object level answer set of $B \cup H$ that contains all of the positive examples and none of the negative examples (enforced by the `goal` constraint).

Example 3.15. *Consider the ILP_b task $T = \langle B, S_M, E^+, E^- \rangle$, where S_M is characterised by the mode declarations in Example 3.14.*

$$B = \left\{ \begin{array}{l} \texttt{bird(a).} \\ \texttt{bird(b).} \\ \texttt{can(a,fly).} \\ \texttt{can(b,swim).} \\ \texttt{ability(fly).} \\ \texttt{ability(swim).} \end{array} \right\} \qquad \begin{array}{l} E^+ = \{\texttt{penguin(b)}\} \\ E^- = \{\texttt{penguin(a)}\} \end{array}$$

The ASPAL meta-level representation is the program:

$$Meta = \left\{ \begin{array}{l} \texttt{bird(a).} \\ \texttt{bird(b).} \\ \texttt{can(a,fly).} \\ \texttt{can(b,swim).} \\ \texttt{ability(fly).} \\ \texttt{ability(swim).} \\ \\ \texttt{penguin(X):-bird(X),rule(1).} \\ \texttt{penguin(X):-bird(X), not can(X,C1),rule(2,C1).} \\ \texttt{penguin(X):-bird(X), not can(X,C1), not can(X,C2),rule(3,C1,C2).} \\ \\ \texttt{0\{rule(1),rule(2,fly),rule(2,swim),rule(3,fly,swim)\}4.} \\ \\ \texttt{goal:-penguin(b), not penguin(a).} \\ \texttt{:- not goal.} \end{array} \right\}$$

[6]This is a slight simplification. In the ASPAL algorithm, this is a choice rule using conditional literals, in order to delegate the grounding of the possible constants to the ASP solver. The ground version of ASPAL's choice rule is identical to the one presented in this definition.

Note that each skeleton rule R has been appended with the atom R_{meta}, where each of the arguments other than the identifier R_{id} is a variable representing a placeholder for a constant. The choice rule on the other hand contains the atoms R'_{meta} such that R' is an instance of a skeleton rule R (instantiating the constant placeholders with the list of constants in R'_{meta}).

The answer sets of this program can be mapped to the inductive solutions of T. For example, the answer set {`bird(a)`, `bird(b)`, `can(a,fly)`, `can(b,swim)`, `ability(fly)`, `ability(swim)`, `penguin(b)`, `rule(2,fly)`, `goal`} *shows that the hypothesis* {`penguin(X):-bird(X), not can(X,fly).`} *is an inductive solution of T.*

In the ASPAL algorithm, this meta representation is combined with an optimisation statement (similar to weak constraints in ASP), which orders the meta-level answer sets by the length of the hypothesis that they represent. ASPAL has been proven to be sound and complete with respect to the optimal inductive solutions of any brave induction task [CR11].

RASPAL ASPAL scales poorly with respect to the size of $ground(B \cup S_M)$ [ACBR13]. One of the main factors in the size of this ground program is the number of body literals that are allowed to appear in a rule in the hypothesis space. RASPAL [ACBR13] iteratively refines a hypothesis until all of the examples in an ILP_b task are covered. At each step, the number of literals that are allowed to be added to the hypothesis is restricted, meaning that the grounding is often significantly smaller than the meta-level program in ASPAL. In [Ath15] it was shown that RASPAL significantly outperforms ASPAL on some learning tasks with large problem domains and large hypothesis spaces.

Summary

In this chapter, we have reviewed the learning frameworks and algorithms for both traditional and ASP-based ILP. In the next chapter, we introduce our own new frameworks for learning ASP.

Part I

Learning Answer Set Programs from Non-Noisy Examples

Chapter 4

Learning from Answer Sets

In the previous chapter, we presented the main frameworks for learning ASP programs, which fall into two categories: either the examples must be covered in at least one answer set of the learned program (brave induction [SI09] and induction of stable models [Ote01]), or the examples must be covered in every answer set of the learned program (cautious induction [SI09]). Work on using brave induction (such as [Ray09] and [CRL12]) has often only considered learning stratified programs[1]. In general, however, ASP programs can have one, many or even no answer sets. Example 4.1 presents a program H describing the rules of Sudoku, and shows that no brave induction, induction of stable models or cautious induction task could possibly have H as an optimal solution.

Example 4.1. *Consider a background knowledge B that contains definitions of the structure of a 4x4 Sudoku grid; i.e. definitions of* `cell`*,* `same_row`*,* `same_col` *and* `same_block` *(where* `same_row`*,* `same_col` *and* `same_block` *are true only for two* different *cells in the same row, column or block).*

$$B = \left\{ \begin{array}{llll}
\texttt{cell((1,1)).} & \texttt{cell((1,2)).} & \ldots & \texttt{cell((4,4)).} \\
\multicolumn{4}{l}{\texttt{same\_row((X1,Y),(X2,Y)):-cell((X1,Y)),cell((X2,Y)),X1} \neq \texttt{X2.}} \\
\multicolumn{4}{l}{\texttt{same\_col((X,Y1),(X,Y2)):-cell((X,Y1)),cell((X,Y2)),Y1} \neq \texttt{Y2.}} \\
\texttt{block((1,1),1).} & \texttt{block((1,2),1).} & \texttt{block((2,1),1).} & \texttt{block((2,2),1).} \\
\texttt{block((3,1),2).} & \texttt{block((3,2),2).} & \texttt{block((4,1),2).} & \texttt{block((4,2),2).} \\
\texttt{block((1,3),3).} & \texttt{block((1,4),3).} & \texttt{block((2,3),3).} & \texttt{block((2,4),3).} \\
\texttt{block((3,3),4).} & \texttt{block((3,4),4).} & \texttt{block((4,3),4).} & \texttt{block((4,4),4).} \\
\multicolumn{4}{l}{\texttt{same\_block(C1,C2):-block(C1,B),block(C2,B),C1} \neq \texttt{C2.}}
\end{array} \right\}$$

[1]Both XHAIL [Ray09] and ASPAL [CRL12] support learning non-stratified programs, but the background knowledge and hypothesis space of each of the example tasks in [Ray09] and [CRL12] is stratified.

One hypothesis H that describes the correct rules of Sudoku is as follows:

$$H = \left\{ \begin{array}{l} \texttt{1\{value(C,1),value(C,2),value(C,3),value(C,4)\}1:-cell(C).} \\ \texttt{:-same\_row(C1,C2),value(C1,V),value(C2,V).} \\ \texttt{:-same\_col(C1,C2),value(C1,V),value(C2,V).} \\ \texttt{:-same\_block(C1,C2),value(C1,V),value(C2,V).} \end{array} \right\}$$

Let S_M be a set of rules which contains the rules in H (for the purposes of this example, it does not matter which other rules it contains). There is no ILP_b, ILP_{sm} or ILP_c task such that H is a solution, and no subset of H is a solution. In practice, as ILP systems tend to search for a solution that is as short as possible (called an optimal solution), this means that no system for ILP_b, ILP_{sm} or ILP_c will return H as the solution. We now show that no task exists, for any of the three frameworks, for which H is an optimal solution.

- *Assume that there is an ILP_b task T_b with background knowledge B such that H is a solution of T_b. Then there must be at least one answer set of $B \cup H$ that contains all of the positive examples of T_b and none of the negative examples of T_b. But this answer set must also be an answer set of $B \cup \{$`1{value(C,1),value(C,2),value(C,3),value(C,4)}1:-cell(C).`$\}$, as the constraints in H only rule out answer sets (by Lemma 2.12). Hence, $H' = \{$`1{value(C,1),value(C,2),value(C,3),value(C,4)}1:-cell(C).`$\}$ must also be an inductive solution of T_b. As H' is shorter than H, this means that H cannot possibly be an optimal solution of T_b.*

- *The argument for ILP_{sm} is similar to ILP_b. Assume there is an ILP_{sm} task T_{sm} with background knowledge B such that H is a solution of T_{sm}. Then for each example e, there must be at least one answer set A_e of $B \cup H$, such that A_e extends e. In each case, A_e must also be an answer set of $B \cup \{$`1{value(C,1),value(C,2),value(C,3),value(C,4)}1:-cell(C).`$\}$, as the constraints in H only rule out answer sets (by Lemma 2.12). Hence, the hypothesis $H' = \{$`1{value(C,1),value(C,2),value(C,3),value(C,4)}1:-cell(C).`$\}$ must also be an inductive solution of T_{sm}. As H' is shorter than H, this means that H cannot possibly be an optimal solution of T_{sm}.*

- *If we use ILP_c to learn H, we have to give examples which are either true in every answer set of $B \cup H$, or false in every answer set. Therefore, we could not give any meaningful examples about the* `value` *predicate – for each atom* `value(x,y)` *(where* `x` *and* `y` *range from 1 to 4), there is at least one answer set of $B \cup H$ that contains* `value(x,y)` *and at least one that does not; this means that if* `value(x,y)` *is given as either a positive or negative example, H will not be a solution of the task. This means that for any ILP_c task $T_c = \langle B, S_M, E^+, E^- \rangle$ such that H is a solution, $E^+ \subseteq \{\texttt{a} \mid \forall A \in AS(B), \texttt{a} \in A\}$ and $E^- \subseteq \{\texttt{a} \mid \forall A \in AS(B), \texttt{a} \notin A\}$. Hence, for any such task, $\emptyset$ must be a solution of T_c, meaning that H cannot be an optimal solution.*

The problem with using either brave or cautious induction to learn general ASP programs is that brave induction can only reason about what should be true in at least one answer set of the learned program, which can be far too weak a condition, and cautious induction can only express what should be true in all answer sets of a program, which can be far too strong a condition. Furthermore, examples in both frameworks are atoms. In ASP it is common [EIK09] to represent a problem such that the answer sets are solutions (see Figure 4.1 (a)). In order to learn ASP programs, examples should therefore be of what should (or should not) be an answer set of the program (Figure 4.1 (b)). In the context of learning the rules of Sudoku using the representation in Example 4.1, this corresponds to giving examples of Sudoku grids rather than the values of individual cells.

In practice, there may be some atoms whose values are unknown before learning. It is therefore more practical to consider learning from partial interpretations rather than full interpretations. This setting, under the answer set semantics is the basis of our first framework, *Learning from Answer Sets*.

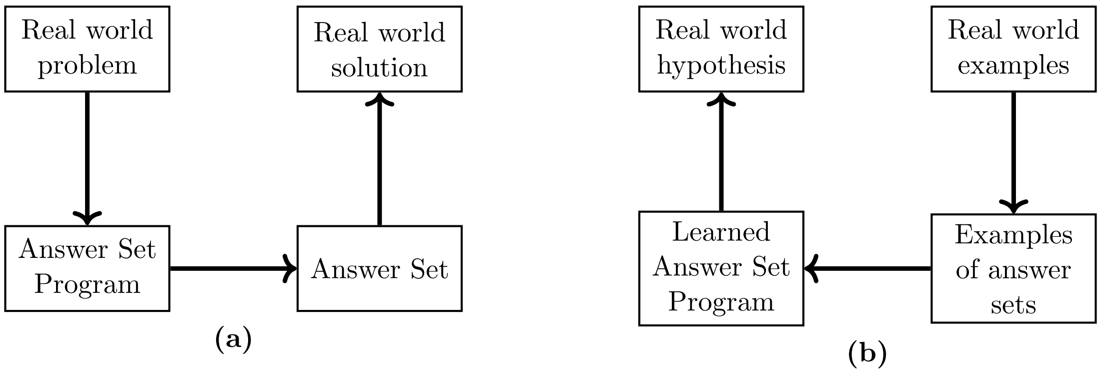


Figure 4.1: (a) shows the general paradigm of answer set programming; (b) shows the general idea of Learning from Answer Sets.

4.1 Language Biases and Hypothesis Spaces

In general ILP systems search for inductive solutions within a *hypothesis space*. One way of defining a hypothesis space is to explicitly give the full set of rules that are allowed to appear in a hypothesis. For the remainder of this thesis, we assume that all *hypothesis spaces* are finite subsets of $\mathcal{ASP}^R$. Given a hypothesis space S_M containing the rules $h_1, \ldots, h_n$, such that for each h_i, $\texttt{id}_\texttt{i} = id(h_i)$, we will often write S_M using the following notation:

$$\left\{ \begin{array}{c} \texttt{id}_\texttt{1} : \ h_1 \\ \ldots \\ \texttt{id}_\texttt{n} : \ h_n \end{array} \right\}$$

In general, it is impractical to explicitly state the set of rules in a hypothesis space, so ILP systems tend to use *mode declarations* [Mug95]. In the examples in this thesis, for simplicity, we define hypothesis spaces in full. The mode declarations used by the ILASP systems are formalised Appendix A.

4.1.1 Hypothesis Length and Optimality

It is common practice in ILP to search for *most compressed* or *optimal* hypotheses [Mug95, Ray09, Sri01]. This is often defined in terms of the number of literals in the hypothesis. This does not apply well to hypotheses that include aggregates: the length of $\texttt{1}\{\texttt{p},\texttt{q}\}\texttt{1}$ (exactly one of `p` and `q` is true) would be the same as the length of $\texttt{0}\{\texttt{p},\texttt{q}\}\texttt{2}$ (none, either or both of `p` and `q` is true), but clearly they do not represent similar concepts. To calculate the length of an aggregate we convert it to disjunctive normal form, as this takes into account both the number of answer sets that the aggregate generates and the number of literals it uses. For example, $\texttt{0}\{\texttt{p},\texttt{q}\}\texttt{2}$ is considered as $(\texttt{p} \wedge \texttt{q}) \vee (\texttt{p} \wedge \texttt{ not q}) \vee (\texttt{ not p} \wedge \texttt{q}) \vee (\texttt{ not p} \wedge \texttt{ not q})$, which has length 8, whereas $\texttt{1}\{\texttt{p},\texttt{q}\}\texttt{1}$ is considered as $(\texttt{p} \wedge \texttt{ not q}) \vee (\texttt{ not p} \wedge \texttt{q})$, which has length 4.

Definition 4.1. Given a hypothesis H, the *length* of the hypothesis, $|H|$, is the number of literals that appear in H^D, where H^D is constructed from H by converting all aggregates in H to disjunctive normal form.

Notation ($^*ILP_X(T)$ and $^nILP_X(T)$). For any learning framework ILP_X and any ILP_X learning task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$, we denote with $^*ILP_X(T)$ the set of all optimal inductive solutions of T, where optimality is defined in terms of the length of the hypotheses. We will also denote with $^nILP_X(T)$ the set of all inductive solutions of T which have length n.

4.2 Learning from Answer Sets

We now present the first of our three learning frameworks, called *Learning from Answer Sets* (ILP_{LAS}). A learning from answer sets task consists of an ASP background knowledge B, a hypothesis space and sets of positive and negative partial interpretation examples. The goal is to find a hypothesis H that has at least one answer set (when combined with B) that extends each positive example, and no answer set that extends any negative examples. Note that each positive example could be extended by a different answer set of the learned program.

Definition 4.2. A *Learning from Answer Sets* task is a tuple $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$ where B is an ASP program, S_M a set of ASP rules and E^+ and E^- are finite sets of partial interpretations. A hypothesis $H \subseteq S_M$ is an *inductive solution* of T if and only if:

1. $\forall e^+ \in E^+ \; \exists A \in AS(B \cup H)$ such that A extends e^+
2. $\forall e^- \in E^- \; \nexists A \in AS(B \cup H)$ such that A extends e^-

Note that this definition combines properties of both brave and cautious frameworks: positive examples must each be extended by at least one answer set, and so they can represent brave induction; whereas negative examples express what should be true in no answer set (and hence what should be true in every answer set), representing cautious induction.

Example 4.2. *Reconsider the Sudoku background knowledge B and hypothesis H from Example 4.1. We showed in Example 4.1 that no ILP_b, ILP_{sm} or ILP_c task has H as an optimal solution, meaning that in practice, systems that use these frameworks cannot learn H. We will now demonstrate that ILP_{LAS} can learn H. First, consider an extremely restrictive hypothesis space S_M that consists of* only *the rules in H. H is an optimal solution of the ILP_{LAS} task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$, where E^+ and E^- are as follows:*

$$E^+ = \left\{ \ \langle\{\texttt{value((1,1),1)}\}, \emptyset\rangle \ \right\}$$

$$E^- = \left\{ \begin{array}{l} \langle\{\texttt{value((1,1),1)}, \texttt{value((1,3),1)}\}, \emptyset\rangle \\ \langle\{\texttt{value((1,1),1)}, \texttt{value((3,1),1)}\}, \emptyset\rangle \\ \langle\{\texttt{value((1,1),1)}, \texttt{value((2,2),1)}\}, \emptyset\rangle \end{array} \right\}$$

We can demonstrate that H is an optimal solution of T by showing that H is a solution, and no hypothesis $H' \subset H$ is also a solution of T.

- *We can show that H is a solution of T by checking that it covers all of the examples. The single positive example means that at least one answer set of $B \cup H$ must contain* `value((1,1),1)`*. This is clearly the case as the answer sets of $B \cup H$ correspond to the valid Sudoku grids, and there is at least one valid grid where 1 appears in the first cell. The first negative example means that no answer set of $B \cup H$ should contain* both `value((1,1),1)` *and* `value((1,3),1)`*. The cells* (1,1) *and* (1,3) *are in the same column, so $B \cup H$ cannot have such an answer set as H contains the constraint "*`:- same_col(C1,C2),value(C1,V),value(C2,V)`*", and hence H covers the example. The other two negative examples are similarly covered due to the other constraints in H.*

- *We now show that no strict subset H' of H can be an inductive solution of T. Any strict subset H' must either be missing the choice rule in H, or one of the three constraints in H.*

  If H' is missing the choice rule, then $B \cup H'$ has no rule with the `value` *predicate in the head, meaning that no answer set of $B \cup H'$ can contain* `value((1,1),1)`*, so H cannot cover the positive example and is therefore not a solution of T.*

  On the other hand, if H' contains the choice rule, but is missing one of the three constraints in H, then $B \cup H'$ will have at least one answer set that extends one of the three negative examples, meaning that H' does not cover all of the negative examples, and so is not a solution of T. For example, if H' is missing the constraint, "`:- same_col(C1,C2),value(C1,V),value(C2,V)`*", then one of the answer sets of $B \cup H'$ (projected over the predicate* `value/2`*) is* {`value((1,1),1)`, `value((2,1),2)`, `value((3,1),3)`, `value((4,1),4)`, `value((1,2),3)`, `value((2,2),4)`, `value((3,2),1)`, `value((4,2),2)`, `value((1,3),1)`, `value((2,3),2)`, `value((3,3),3)`, `value((4,3),4)`, `value((1,4),3)`, `value((2,4),4)`, `value((3,4),1)`, `value((4,4),2)`}. *This answer set extends the first negative example, meaning that H' is not a solution of T.*

The hypothesis space S_M in this example is extremely restrictive. If we consider a wider hypothesis space, then we may need more examples to ensure that H is still an optimal inductive solution of T. However, no matter how many rules are added to S_M, we can always add further examples to T such that for each optimal inductive solution H' of T, $AS(B \cup H') = AS(B \cup H)$. We demonstrate this by showing that for any hypothesis H' such that $AS(B \cup H') \neq AS(B \cup H)$, we can construct an example such that H covers e, but H' does not.

***Case 1**: There is an answer set $A \in AS(B \cup H)$ such that $A \notin AS(B \cup H')$*

Consider the partial interpretation $e = \langle A, HB_{B \cup S_M} \backslash A \rangle$. $B \cup H$ has an answer set that extends e, but $B \cup H'$ does not. Hence, if e is given as a positive example, H will cover e, but H' will not.

***Case 2**: There is an answer set $A \in AS(B \cup H')$ such that $A \notin AS(B \cup H)$*

Consider the partial interpretation $e = \langle A, HB_{B \cup S_M} \backslash A \rangle$. $B \cup H'$ has an answer set that extends e, but $B \cup H$ does not. Hence, if e is given as a negative example, H will cover e, but H' will not.

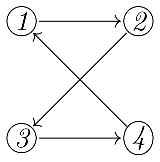
Thus, for any hypothesis space S_M that contains H, there is an ILP_{LAS} task with background knowledge B and hypothesis space S_M such that H, or some hypothesis with exactly the same answer sets when combined with B, is an optimal inductive solution of the task.

Another example of a learning from answer sets task, which we revisit throughout the thesis, is of learning the definition of a Hamiltonian graph.

Example 4.3. *Consider the problem of learning the definition of what it means for a graph to be Hamiltonian. The background knowledge B defines what it means to be a graph, up to size 4.*

$$B = \left\{ \begin{array}{l} \texttt{1\{size(1), size(2), size(3), size(4)\}1.} \\ \texttt{node(1..S) :- size(S).} \\ \texttt{0\{edge(N1, N2)\}1 :- node(N1), node(N2).} \end{array} \right\}$$

The answer sets of B exactly represent the graphs of size 1 to 4. For example, the answer set {size(4), node(1), node(2), node(3), node(4), edge(1,2), edge(2,3), edge(3,4), edge(4,1)} *represents the graph G:*



The program H can be used to determine whether a graph is Hamiltonian or not. The answer sets of $B \cup H$ correspond exactly with the Hamiltonian graphs of size 1 to 4.

$$H = \left\{ \begin{array}{l} \texttt{0\{in(V0, V1)\}1 :- edge(V0, V1).} \\ \texttt{reach(V0) :- in(1, V0).} \\ \texttt{reach(V1) :- in(V0, V1), reach(V0).} \\ \texttt{:- node(V0), not reach(V0).} \\ \texttt{:- in(V0, V1), in(V0, V2), V1! = V2.} \end{array} \right\}$$

The graph G can be represented as a partial interpretation ⟨{`size(4)`, `edge(1,2)`, `edge(2,3)`, `edge(3,4)`, `edge(4,1)`}, {`edge(1,1)` , `edge(1,3)`, `edge(1,4)`, `edge(2,1)`, `edge(2,2)`, `edge(2,4)`, `edge(3,1)`, `edge(3,2)`, `edge(3,3)`, `edge(4,2)`, `edge(4,3)`, `edge(4,4)`}⟩. *Note that as the example is a partial interpretation, the* `reach` *and* `in` *predicates do not need to be specified. In ILP, learning a concept which does not feature in the language of the problem (i.e. the background knowledge and examples) is known as* predicate invention *[Sta93].*

4.3 Learning from Ordered Answer Sets

Preference Learning has received much attention over the last decade from within the machine learning community. A popular approach to preference learning is *learning to rank* [FH03, GHH01], where the goal is to learn to rank any two objects given some examples of pairwise preferences (indicating that one object is preferred to another). While in previous work ILP systems such as TILDE [BDR98] and Aleph [Sri01] have been applied to preference learning [DJJT01, Hor12], this has addressed learning ratings, such as `good`, `poor` and `bad`, rather than rankings over the examples. Ratings are not expressive enough if we want to find an optimal solution as we may rate many objects as `good` when some are better than others. ASP, on the other hand, allows the expression of preferences through *weak constraints.*

Weak constraints do not affect what is, or is not, an answer set of a program. Instead, they create a preference ordering over the answer sets of a program; i.e. they allow us to specify which answer sets are preferred to other answer sets. Example 4.4 shows how a set of preferences can be encoded as weak constraints.

Example 4.4. *Consider the problem of using a user's preferences over alternative journeys, in order to select the optimal journey. Let A, B, C and D be the journeys represented by the following sets of attributes. Each journey is split into a number of* legs, *in which a single mode of transport is used.*

$$\left\{\begin{array}{l}\texttt{leg\_mode(1,walk)},\\ \texttt{leg\_crime\_rating(1,2)},\\ \texttt{leg\_distance(1,500)},\\ \texttt{leg\_mode(2,bus)},\\ \texttt{leg\_crime\_rating(2,4)},\\ \texttt{leg\_distance(2,3000)}\end{array}\right\} \qquad \left\{\begin{array}{l}\texttt{leg\_mode(1,bus)},\\ \texttt{leg\_crime\_rating(1,2)},\\ \texttt{leg\_distance(1,4000)},\\ \texttt{leg\_mode(2,walk)},\\ \texttt{leg\_crime\_rating(2,5)},\\ \texttt{leg\_distance(2,1000)}\end{array}\right\}$$

(A) *(B)*

$$\left\{\begin{array}{l}\texttt{leg\_mode(1,bus)},\\ \texttt{leg\_crime\_rating(1,2)},\\ \texttt{leg\_distance(1,400)},\\ \texttt{leg\_mode(2,bus)},\\ \texttt{leg\_crime\_rating(2,4)},\\ \texttt{leg\_distance(2,3000)}\end{array}\right\} \qquad \left\{\begin{array}{l}\texttt{leg\_mode(1,bus)},\\ \texttt{leg\_crime\_rating(1,5)},\\ \texttt{leg\_distance(1,2000)},\\ \texttt{leg\_mode(2,walk)},\\ \texttt{leg\_crime\_rating(2,1)},\\ \texttt{leg\_distance(2,2000)}\end{array}\right\}$$

(C) *(D)*

The following weak constraints H give a preference ordering to the journeys A to D.

$$H = \left\{\begin{array}{l}:\sim \texttt{leg\_mode(L,walk)}, \texttt{leg\_crime\_rating(L,C)}, \texttt{C} > \texttt{4}.[\texttt{1@3,L,C}]\\ :\sim \texttt{leg\_mode(L,bus)}.[\texttt{1@2,L}]\\ :\sim \texttt{leg\_mode(L,walk)}, \texttt{leg\_distance(L,D)}.[\texttt{D@1,L,D}]\end{array}\right\}$$

The first weak constraint in H means that the user would like to avoid walking through an area with a crime rating higher than 4. A journey pays a penalty of 1 at priority level 3 for each leg of the journey that involves walking though such an area. As there is no weak constraint in H with a priority level higher than 3, this preference is the most important. The second weak constraint (at priority level 2) means that the user would like to take as few buses as possible. The third weak constraint (at priority level 1) means that the user would like to minimise the distance that they have to walk. Note that as a penalty of the distance is paid for each leg where the user has to walk, the total penalty is equal to the total walking distance of the journey. Given these preferences, journey A is the best journey, followed by D, then C and then journey B.

The hypothesis in Example 4.4 could be learned by giving examples of which journeys are preferred to which other journeys. For the preferences to be learned as weak constraints, this would require examples of pairs of answer sets, such that the first is preferred to the second. In fact, each of our ordering examples contains two partial interpretations, rather than two complete answer sets. We also allow examples to be given with any of the operators $<$, $\leq$, $=$, $\neq$, $>$ or $\geq$. The $<$ operator, for example, indicates that the first partial interpretation is preferred to the second; whereas the $=$ operator specifies that the two partial interpretations are equal.

Definition 4.3. An *ordering example* is a tuple $o = \langle e_1, e_2, op\rangle$ where e_1 and e_2 are partial interpretations and op is a binary comparison operator ($<$, $>$, $=$, $\leq$, $\geq$ or $\neq$).

As our orderings contain two partial interpretations, rather than two full interpretations, there are two possible semantics to give to the examples. The *brave* semantics indicates that there should be at least one pair of answer sets extending the pair of partial interpretations, which are ordered according to the operator. The *cautious* semantics, on the other hand, indicates that every pair of answer sets that extend the pair of partial interpretations should be ordered according to the operator.

Definition 4.4. Let $o = \langle e_1, e_2, op\rangle$ be an ordering example. An ASP program P *bravely respects o* iff $\exists A_1, A_2 \in AS(P)$ such that all of the following conditions hold: (i) A_1 extends e_1; (ii) A_2 extends

e_2; and (iii) $\langle A_1, A_2, op\rangle \in ord(P)$. *$P$ cautiously respects o* iff $\nexists A_1, A_2 \in AS(P)$ such that all of the following conditions hold: (i) A_1 extends e_1; (ii) A_2 extends e_2; and (iii) $\langle A_1, A_2, op\rangle \notin ord(P)$.

We now define the notion of *Learning from Ordered Answer Sets* (ILP_{LOAS}).

Definition 4.5. A *Learning from Ordered Answer Sets* task is a tuple $T = \langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$ where B is an ASP program, S_M is a set of ASP rules, E^+ and E^- are finite sets of partial interpretations and O^b and O^c are finite sets of ordering examples over E^+ called brave and cautious orderings. A hypothesis $H \subseteq S_M$ is an inductive solution of T if and only if:

1. $H \in ILP_{LAS}(\langle B, S_M, \langle E^+, E^-\rangle\rangle)$
2. $\forall o \in O^b$ $B \cup H$ bravely respects o
3. $\forall o \in O^c$ $B \cup H$ cautiously respects o

Note that the orderings are only over positive examples. We chose to make this restriction as there does not appear to be any scenario where a hypothesis would need to respect an ordering of a pair of partial interpretations that are not extended by any pair of answer sets of $B \cup H$.

Example 4.5. *Recall the journey preferences in Example 4.4. Consider the background knowedge B, which defines a set of possible journeys.*

$$B = \left\{ \begin{array}{l} \texttt{1\{leg(1),\ldots,leg(5)\}5.} \\ \texttt{1\{leg\_mode(L,walk),leg\_mode(L,bus)\}1:-leg(L).} \\ \texttt{1\{leg\_crime\_rating(L,1),\ldots,leg\_crime\_rating(L,4000)\}1:-leg(L).} \\ \texttt{1\{leg\_distance(L,0),leg\_distance(L,500),\ldots,leg\_distance(L,4000)\}1:-leg(L).} \end{array} \right\}$$

Journeys A to D of Example 4.4 can be represented by the four positive examples e_A to e_D.

$$\left\langle \left\{ \begin{array}{l} \texttt{leg\_mode(1,walk),} \\ \texttt{leg\_crime\_rating(1,2),} \\ \texttt{leg\_distance(1,500),} \\ \texttt{leg\_mode(2,bus),} \\ \texttt{leg\_crime\_rating(2,4),} \\ \texttt{leg\_distance(2,3000)} \end{array} \right\}, \left\{ \begin{array}{l} \texttt{leg(3),} \\ \texttt{leg(4),} \\ \texttt{leg(5)} \end{array} \right\} \right\rangle \quad (e_A)$$

$$\left\langle \left\{ \begin{array}{l} \texttt{leg\_mode(1,bus),} \\ \texttt{leg\_crime\_rating(1,2),} \\ \texttt{leg\_distance(1,4000),} \\ \texttt{leg\_mode(2,walk),} \\ \texttt{leg\_crime\_rating(2,5),} \\ \texttt{leg\_distance(2,1000)} \end{array} \right\}, \left\{ \begin{array}{l} \texttt{leg(3),} \\ \texttt{leg(4),} \\ \texttt{leg(5)} \end{array} \right\} \right\rangle \quad (e_B)$$

$$\left\langle \left\{ \begin{array}{l} \texttt{leg\_mode(1,bus),} \\ \texttt{leg\_crime\_rating(1,2),} \\ \texttt{leg\_distance(1,400),} \\ \texttt{leg\_mode(2,bus),} \\ \texttt{leg\_crime\_rating(2,4),} \\ \texttt{leg\_distance(2,3000)} \end{array} \right\}, \left\{ \begin{array}{l} \texttt{leg(3),} \\ \texttt{leg(4),} \\ \texttt{leg(5)} \end{array} \right\} \right\rangle \quad (e_C)$$

$$\left\langle \left\{ \begin{array}{l} \texttt{leg\_mode(1,bus),} \\ \texttt{leg\_crime\_rating(1,5),} \\ \texttt{leg\_distance(1,2000),} \\ \texttt{leg\_mode(2,walk),} \\ \texttt{leg\_crime\_rating(2,1),} \\ \texttt{leg\_distance(2,2000)} \end{array} \right\}, \left\{ \begin{array}{l} \texttt{leg(3),} \\ \texttt{leg(4),} \\ \texttt{leg(5)} \end{array} \right\} \right\rangle \quad (e_D)$$

As these positive examples completely represent each journey, there is exactly one answer set of B that extends each example. Therefore there is no distinction between brave and cautious orderings in this case. Recall from Example 4.4 that journey A was preferred to journey D, which was preferred to journey C, which was preferred to journey B. This means that to learn the preferences in Example 4.4, we could give the orderings $\langle e_A, e_D, <\rangle$, $\langle e_D, e_C, <\rangle$ and $\langle e_C, e_B, <\rangle$ as either brave or cautious orderings.

4.4 Context-Dependent Learning from Ordered Answer Sets

Common to previous ILP frameworks is the underlying assumption that hypotheses should cover the examples with respect to one fixed given background knowledge. But, in practice, some examples may be context-dependent – different examples may need to be covered using different background knowledges. The journey preferences in Example 4.4 can be extended, for example, with contextual information (e.g. the weather).

Example 4.6. *Reconsider the background knowledge and examples from Example 4.5. It may be that certain attributes of a journey are* context-dependent*; for instance, weather conditions may be important. Any of the ordering examples o in Example 4.5 could be extended with a context such as $C = \{$`raining.`$\}$. This would mean that for a brave ordering o, there should be a pair of answer sets of $B \cup H \cup C$ that extends the partial interpretations in o and that respect the ordering (wrt the weak constraints in $B \cup H \cup C$).*

In fact, the context dependent ordering examples we present are slightly more general than in Example 4.6, as we allow each partial interpretation in the CDOE to have its own context. We will see that in addition to representing genuinely contextual information, in some cases, contexts can be used in order to partition the background knowledge into pieces that are relevant to particular examples.

In this section, we present a generalisation of ILP_{LOAS}, called *context-dependent learning from ordered answer sets* ($ILP^{context}_{LOAS}$). We now formalise the notion of its *context-dependent* examples. Similarly to our previous examples, these are of two types: partial interpretations and ordering examples.

Definition 4.6. A *context-dependent partial interpretation* (CDPI) is a pair $e = \langle e_{pi}, e_{ctx}\rangle$, where e_{pi} is a partial interpretation and e_{ctx} is an $\mathcal{ASP}^{ch}$ program, called a *context.* Given a program P, an interpretation I is said to be an *accepting answer set* of e wrt P if and only if $I \in AS(P \cup e_{ctx})$ and I extends e_{pi}. P is said to *accept* e if there is at least one accepting answer set of e wrt P.

Notation (*inverse*). Given a CDOE $o = \langle e_1, e_2, op\rangle$, $inverse(o) = \langle e_1, e_2, op^{-1}\rangle$, where $<^{-1}$ is $\geq$, $\leq^{-1}$ is $>$, $=^{-1}$ is $\neq$, $\neq^{-1}$ is $=$, $>^{-1}$ is $\leq$ and $\geq^{-1}$ is $>$.

Definition 4.7. A *context-dependent ordering example* (CDOE) o is a tuple $\langle\langle e^1_{pi}, e^1_{ctx}\rangle, \langle e^2_{pi}, e^2_{ctx}\rangle, op\rangle$, where the first two elements are CDPIs and op is a binary comparison operator ($<$, $>$, $=$, $\leq$, $\geq$ or $\neq$).

A pair of interpretations $\langle I_1, I_2 \rangle$ is said to be an *accepting pair of answer sets* of o wrt a program P if all of the following conditions hold: (i) I_1 is an accepting answer set of $\langle e^1_{pi}, e^1_{ctx} \rangle$; (ii) I_2 is an accepting answer set of $\langle e^2_{pi}, e^2_{ctx} \rangle$; and (iii) $\langle I_1, I_2, op \rangle \in \mathit{ord}(P, AS(P \cup e^1_{ctx}) \cup AS(P \cup e^2_{ctx}))$. A program P is said to *bravely respect* o if there is at least one accepting pair of answer sets of o. P is said to *cautiously respect* o if there is no accepting pair of answer sets of $\mathit{inverse}(o)$.

Given an ordering example o, we write o_{eg1}, o_{eg2} to refer to the two CDPIs in o and o_{op} to refer to the binary comparison operator of o.

Definition 4.8. A *Context-dependent Learning from Ordered Answer Sets* ($ILP^{context}_{LOAS}$) task is a tuple $T = \langle B, S_M, \langle E^+, E^-, O^b, O^c \rangle \rangle$ where B is an ASP program, S_M is a set of ASP rules, E^+ and E^- are finite sets of CDPIs, and O^b and O^c are finite sets of CDOEs over E^+ called, respectively, brave and cautious orderings. A hypothesis $H \subseteq S_M$ is an inductive solution of T if and only if:

1. $\forall e \in E^+$, $B \cup H$ accepts e
2. $\forall e \in E^-$, $B \cup H$ does not accept e
3. $\forall o \in O^b$, $B \cup H$ bravely respects o
4. $\forall o \in O^c$, $B \cup H$ cautiously respects o

Example 4.7. *Reconsider the journey preference learning task of Example 4.5. The contextual information in Example 4.6 can be added to the examples* e_1 *and* e_2*, to show the preference "in the case that it is raining* e_1 *is preferred to* e_2*, but otherwise it is the other way around" with the context dependent ordering examples* o_1 *and* o_2*:*

$$o_1 = \left\langle \left\langle e_1, \left\{ \texttt{raining.} \right\} \right\rangle, \left\langle e_2, \left\{ \texttt{raining.} \right\} \right\rangle \right\rangle \qquad o_2 = \langle \langle e_2, \emptyset \rangle, \langle e_1, \emptyset \rangle \rangle$$

4.4.1 Translation to Non-Context-Dependent Tasks

In this section, we show that any $ILP^{context}_{LOAS}$ task can be translated into an ILP_{LOAS} task. Example 4.8 shows that a naïve translation, achieved by moving every context into the background knowledge, does not work in general.

Example 4.8. *In general, it is not the case that an* $ILP^{context}_{LOAS}$ *task can be translated into an* ILP_{LOAS} *task simply by moving all the contexts into the background knowledge* ($B \cup C_1 \cup \ldots \cup C_n$ *where* $C_1, \ldots, C_n$ *are the contexts of the examples). Consider, for instance, the* $ILP^{context}_{LOAS}$ *task* $\langle B, S_M, \langle E^+, E^-, O^b, O^c \rangle \rangle$ *defined as follows:*

- $B = \emptyset$. $E^- = \emptyset$. $O^b = \emptyset$. $O^c = \emptyset$
- $S_M = \{$`go_out :- raining.  go_out :- not raining.`$\}$

- $E^+ = \left\{ \begin{array}{l} \langle\langle\{\texttt{go\_out}\}, \emptyset\rangle, \emptyset\rangle, \\ \langle\langle\emptyset, \{\texttt{go\_out}\}\rangle, \{\texttt{raining.}\}\rangle \end{array} \right\}$

This task has one solution: `go_out :- not raining`. *But, if we were to add all the contexts to the background knowledge, we would get a background knowledge containing the single fact* `raining`. *So, there would be no way of explaining both examples, as every hypothesis would, in this case, lead to a single answer set (either* $\{\texttt{raining}, \texttt{go\_out}\}$ *or* $\{\texttt{raining}\}$*), and therefore cover only one of the examples.*

To capture the meaning of the context-dependent examples accurately, we could augment the background knowledge with the choice rule $\texttt{0}\{\texttt{raining}\}\texttt{1}$ *and define the* ILP_{LOAS} *examples as the pairs* $\langle\{\texttt{go\_out}\}, \{\texttt{raining}\}\rangle$ *and* $\langle\{\texttt{raining}\}, \{\texttt{go\_out}\}\rangle$*. In this way, answer sets of the inductive solution would exclude* `go_out` *when* `raining` *(i.e., in the context of raining), and include* `go_out` *otherwise, which is the correct meaning of the given context-dependent examples.*

The following definition gives a general translation of $ILP^{context}_{LOAS}$ to ILP_{LOAS}. The translation assumes that for any CDPI $e = \langle\langle e^{inc}_{pi}, e^{exc}_{pi}\rangle, e_{ctx}\rangle$, $c(e)$ is the partial interpretation $\langle e^{inc}_{pi} \cup \{\texttt{ctx(e}_{\texttt{id}}\texttt{)}\}, e^{exc}_{pi}\rangle$, where `ctx` is a new predicate (that does not occur in the original task). Also, for any program P and any atom `a`, $append(P, a)$ is the program constructed by appending `a` to the body of every rule in P.

Definition 4.9. For any $ILP^{context}_{LOAS}$ task $T = \langle B_1, S_M, \langle E^+_1, E^-_1, O^b_1, O^c_1\rangle\rangle$, $\mathcal{T}_{LOAS}(T) = \langle B_2, S_M, \langle E^+_2, E^-_2, O^b_2, O^c_2\rangle\rangle$, where the components of $\mathcal{T}_{LOAS}(T)$ are as follows:

- $B_2 = B_1 \cup \{append(e_{ctx}, \texttt{ctx(e}_{\texttt{id}}\texttt{)}) \mid e = \langle e_{pi}, e_{ctx}\rangle \in E^+_1 \cup E^-_1\}$
  $\cup \left\{\texttt{1}\{\texttt{ctx(e}^{\texttt{1}}_{\texttt{id}}\texttt{)}, \ldots, \texttt{ctx(e}^{\texttt{n}}_{\texttt{id}}\texttt{)}\}\texttt{1.} \middle| \{e^1, \ldots, e^n\} = E^+_1 \cup E^-_1\right\}$
- $E^+_2 = \{c(e) \mid e \in E^+_1\}$
- $E^-_2 = \{c(e) \mid e \in E^-_1\}$
- $O^b_2 = \{\langle c(e_1), c(e_2), op\rangle \mid \langle e_1, e_2, op\rangle \in O^b_1\}$
- $O^c_2 = \{\langle c(e_1), c(e_2), op\rangle \mid \langle e_1, e_2, op\rangle \in O^c_1\}$

The intuition of this translation is that the contexts of the examples are in the (new) background knowledge, but each rule in them has been appended with an extra atom (unique to each example). There is a choice rule in the new background knowledge that says that only one context can be "used" at a time, and each example of the translated task contains an extra atom (added by the $c(.)$ function) to ensure that the correct context is used for each example.

Theorem 4.9. *(proof on page 266)*

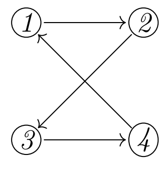
For any $ILP^{context}_{LOAS}$ *learning task* T, $ILP_{LOAS}(\mathcal{T}_{LOAS}(T)) = ILP^{context}_{LOAS}(T)$.

This translation is important for two reasons. Firstly, when we investigate the complexity of each of the frameworks in Section 4.5, the translation enables us to show that the complexity of $ILP_{LOAS}^{context}$ is the same as ILP_{LOAS}. Secondly, it highlights some of the advantages of using context-dependent examples. Specifically, if we do not use context dependent examples then the contextual information of every example must be encoded in the background knowledge in some way. Contexts, on the other hand, can allow for what would have otherwise been background knowledge to be partitioned into the pieces that are relevant for particular examples. Example 4.10 shows how the background knowledge of the non-context dependent task of learning the definition of what it means for a graph to be Hamiltonian (Example 4.3) can be partitioned in this way.

Example 4.10. *Reconsider the background knowledge of the task in Example 4.3. This background knowledge has choice rules which generate all graphs up to size 4. In fact, as each example completely describes a single graph, most of the answer sets of this background knowledge are completely irrelevant to most examples. A context-dependent representation would be to have an empty background knowledge and to have the context of each example be a set of facts describing exactly one graph G. For instance, the context dependent example e_G represents the graph G.*

$$\left\langle \langle\emptyset,\emptyset\rangle, \left\{\begin{array}{l}\texttt{node(1..4).}\\ \texttt{edge(1,2).}\\ \texttt{edge(2,3).}\\ \texttt{edge(3,4).}\\ \texttt{edge(4,1).}\end{array}\right\}\right\rangle$$

(e_G)



(the graph G)

As there is no background knowledge, and the partial interpretation of each example is empty, the goal of this task now becomes to find a hypothesis H such that $H \cup e_{ctx}$ is satisfiable for each positive example e and unsatisfiable for each negative example e.

4.5 Complexity

We now discuss the complexity of the three learning frameworks presented in this chapter, with respect to three decision problems: *verification*, deciding whether a given hypothesis H is an inductive solution of a task T; *satisfiability*, deciding whether a learning task T has any inductive solutions; and *optimum verification*, deciding whether a given hypothesis H is an optimal inductive solution of a task T. We also present the same complexity results for brave induction (ILP_b), induction from stable models (ILP_{sm}) and cautious induction (ILP_c). A summary of the results is shown in Table 4.1. The tasks we consider in this section are propositional (i.e. their background knowledges, hypothesis spaces and the contexts of all examples are ground).

Framework	Verification	Satisfiablity	Optimum Verification
ILP_b	NP-complete	NP-complete	DP-complete
ILP_{sm}	NP-complete	NP-complete	DP-complete
ILP_c	DP-complete	Σ_2^P-complete	Π_2^P-complete
ILP_{LAS}	DP-complete	Σ_2^P-complete	Π_2^P-complete
ILP_{LOAS}	DP-complete	Σ_2^P-complete	Π_2^P-complete
$ILP_{LOAS}^{context}$	DP-complete	Σ_2^P-complete	Π_2^P-complete

Table 4.1: A summary of the complexity of the various learning frameworks.

4.5.1 Learning from Answer Sets with Stratified Summing Aggregates

As there are existing results on the complexity of solving *aggregate stratified* programs, it is useful to introduce a generalisation of ILP_{LAS}. *Learning from Answer Sets with Stratified Aggregates* (ILP_{LAS}^s) is the same as Learning from Answer Sets, except for allowing summing aggregates in the bodies of the rules in B and S_M, so long as $B \cup S_M$ is aggregate stratified. Note that the condition of $B \cup S_M$ being aggregate stratified implies that for any hypothesis $H \subseteq S_M$, $B \cup H$ is aggregate stratified. The existing results on the complexity of these programs allow us to prove the complexity of ILP_{LAS}^s; hence, as we can show that ILP_{LOAS} reduces to ILP_{LAS}^s, this is helpful in proving the complexity of ILP_{LOAS}.

4.5.2 Relationships Between the Learning Tasks

In this section we show that for each of the three decision problems there is a chain of polynomial reductions from ILP_c to ILP_{LAS} to $ILP_{LOAS}^{context}$ to ILP_{LOAS} to ILP_{LAS}^s. This chain of reductions is then used in proving that all four tasks share the same complexity for each decision problem. By proving that ILP_c is $\mathcal{O}$-hard and ILP_{LAS}^s is in $\mathcal{O}$ for some complexity class $\mathcal{O}$, we prove that all four tasks are $\mathcal{O}$-complete. Similarly, as we show that ILP_b and ILP_{sm} both reduce polynomially to each other for each of the three decision problems, if for one of the problems ILP_b is $\mathcal{O}$-complete for some class $\mathcal{O}$ then so is ILP_{sm}. The chains of reductions are shown in Figure 4.2.

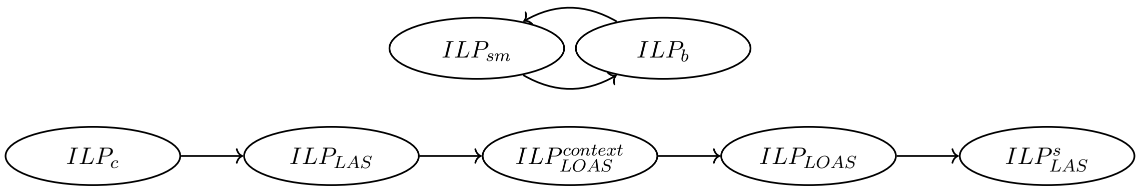

$$\begin{tikzcd} ILP_{sm} \arrow[r, bend right] & ILP_b \arrow[l, bend right] \end{tikzcd}$$
$$\begin{tikzcd} ILP_c \arrow[r] & ILP_{LAS} \arrow[r] & ILP_{LOAS}^{context} \arrow[r] & ILP_{LOAS} \arrow[r] & ILP_{LAS}^s \end{tikzcd}$$

Figure 4.2: Chains of polynomial reductions that hold for the three decision problems considered in this section (each arrow represents that there is a polynomial reduction from one framework to another).

Proposition 4.11 shows that the complexity of ILP_b and of ILP_{sm} coincide for the three decision problems.

Proposition 4.11. *(proof on page 267)*

1. Deciding verification, satisfiability and optimum verification for ILP_b each reduce polynomially to the corresponding ILP_{sm} decision problem.
2. Deciding verification, satisfiability and optimum verification for ILP_{sm} each reduce polynomially to the corresponding ILP_b decision problem.

Proposition 4.12 shows that there is a chain of polynomial reductions from ILP_c to ILP_{LAS} to ILP_{LOAS} to $ILP^{context}_{LOAS}$ to ILP^{s}_{LAS} for the three decision problems.

Proposition 4.12. *(proof on page 268)*

1. Deciding verification, satisfiability and optimum verification for ILP_c each reduce polynomially to the corresponding ILP_{LAS} decision problem.
2. Deciding verification, satisfiability and optimum verification for ILP_{LAS} each reduce polynomially to the corresponding $ILP^{context}_{LOAS}$ decision problem.
3. Deciding verification, satisfiability and optimum verification for $ILP^{context}_{LOAS}$ each reduce polynomially to the corresponding ILP_{LOAS} decision problem.
4. Deciding verification, satisfiability and optimum verification for ILP_{LOAS} each reduce polynomially to the corresponding ILP^{s}_{LAS} decision problem.

4.5.3 Complexity of Deciding Verification, Satisfiability and Optimum Verification for each Framework

In this section we prove the complexity of deciding verification, satisfiability and optimum verification, for each of the learning frameworks. We start with the ILP_b and ILP_{sm} frameworks, for which verification and satisfiability are both NP-complete, and optimum verification is DP-complete.

Proposition 4.13. *(proof on page 271)* Verifying whether a given H is an inductive solution of a general ILP_b task is NP-complete.

Corollary 4.14. Verifying whether a given H is an inductive solution of a general ILP_{sm} task is NP-complete.

Proposition 4.15. *(proof on page 271)* Deciding the satisfiability of a general ILP_b task is NP-complete.

Corollary 4.16. Deciding the satisfiability of a general ILP_{sm} task is NP-complete.

Proposition 4.17. *(proof on page 272)* Verifying whether a given H is an optimal inductive solution of a general ILP_b task is DP-complete.

Corollary 4.18. Deciding whether a hypothesis is an optimal solution of a general ILP_{sm} task is DP-complete.

We have now proved the complexity of deciding verification, satisfiability and optimum verification for ILP_b and ILP_{sm}, proving the corresponding entries in Table 4.1. It remains to show the complexities for ILP_c, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$.

As we have shown that ILP_c reduces (polynomially) to ILP_{LAS} which, in turn, reduces to ILP_{LOAS}, which reduces to $ILP^{context}_{LOAS}$ and that $ILP^{context}_{LOAS}$ reduces to ILP^{s}_{LAS}, to prove the complexity of verifying a hypothesis for each framework, it suffices to show that ILP_c is DP-hard (thus also proving the hardness for each of the other frameworks) and that ILP^{s}_{LAS} is a member of DP (thus proving membership for the other frameworks). This shows that each framework is both a member of DP and also DP-hard, and therefore must be DP-complete.

Proposition 4.19. *(proof on page 273)* Deciding verification for ILP^{s}_{LAS} is a member of DP.

Proposition 4.20. *(proof on page 274)* Deciding verification for ILP_c is DP-hard.

We can now prove the complexity of deciding verification for ILP_c, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$. This proves the corresponding entries in Table 4.1.

Theorem 4.21. *Deciding whether a given H is a solution of any ILP_c, ILP_{LAS}, ILP_{LOAS} or $ILP^{context}_{LOAS}$ task is DP-complete in each case.*

Proof. By Proposition 4.20, deciding the verification for ILP_c is DP-hard. By Proposition 4.12, deciding the verification for ILP_c reduces to deciding verification for ILP_{LAS} which, in turn, reduces to deciding verification for $ILP^{context}_{LOAS}$, which reduces to deciding verification for ILP_{LOAS}, which again reduces to deciding verification for ILP^{s}_{LAS} and by Proposition 4.19, deciding verification for ILP^{s}_{LAS} is a member of DP. Deciding verification for each of these learning frameworks must therefore be both a member of DP and must be DP-hard. Hence, deciding verification for each framework is DP-complete. $\square$

Similarly, to show that deciding satisfiability is Σ^P_2-complete for each framework, we only need to show that ILP^{s}_{LAS} is a member of Σ^P_2 and ILP_c is Σ^P_2-hard.

Proposition 4.22. *(proof on page 274)* Deciding satisfiability for ILP^{s}_{LAS} is in Σ^P_2.

Proposition 4.23. *(proof on page 275)* Deciding satisfiability for ILP_c is Σ^P_2-hard.

We can now prove the complexity of deciding satisfiability for ILP_c, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$. This proves the corresponding entries in Table 4.1.

Theorem 4.24. *Deciding the satisfiability of any ILP_c, ILP_{LAS}, ILP_{LOAS} or $ILP_{LOAS}^{context}$ task is Σ_2^P-complete in each case.*

Proof. (similar to the proof of Theorem 4.21) By Proposition 4.23, deciding satisfiability for ILP_c is Σ_2^P-hard. By Proposition 4.12, deciding satisfiability for ILP_c reduces to deciding satisfiability for ILP_{LAS} which, in turn, reduces to deciding satisfiability for $ILP_{LOAS}^{context}$, which reduces to deciding satisfiability for ILP_{LOAS}, which again reduces to deciding satisfiability of ILP_{LAS}^s. By Proposition 4.22, deciding satisfiability for ILP_{LAS}^s is in Σ_2^P. Deciding satisfiability for each of these learning frameworks is therefore both a member of Σ_2^P and is Σ_2^P-hard. Hence, deciding satisfiability for each framework is Σ_2^P-complete. □

Finally, to show that deciding that optimum verification is Π_2^P-complete for each framework, we only need to show that ILP_{LAS}^s is a member of Π_2^P and ILP_c is Π_2^P-hard.

Proposition 4.25. *(proof on page 276)* Deciding optimum verification for ILP_{LAS}^s is in Π_2^P.

Proposition 4.26. *(proof on page 276)* Deciding whether an arbitrary hypothesis H is an optimal inductive solution of a given ILP_c task is Π_2^P-hard.

We can now prove the complexity of deciding optimum verification for ILP_c, ILP_{LAS}, ILP_{LOAS} and $ILP_{LOAS}^{context}$. This proves the remaining entries in Table 4.1.

Theorem 4.27. *Deciding whether an arbitrary hypothesis H is an optimal solution for any ILP_c, ILP_{LAS}, ILP_{LOAS} or $ILP_{LOAS}^{context}$ task is Π_2^P-complete in each case.*

Proof. (similar to the proof of Theorem 4.21) By Proposition 4.26, deciding optimum verification for ILP_c is Π_2^P-hard. By Proposition 4.12, deciding optimum verification for ILP_c reduces to deciding optimum verification for ILP_{LAS} which, in turn, reduces to deciding optimum verification for $ILP_{LOAS}^{context}$, which reduces to deciding optimum verification for ILP_{LOAS}, which again reduces to deciding optimum verification of ILP_{LAS}^s. By Proposition 4.25, deciding optimum verification for ILP_{LAS}^s is in Σ_2^P. Deciding optimum verification for each of these learning frameworks is therefore both a member of Π_2^P and is Π_2^P-hard. Hence, deciding optimum verification for each framework is Π_2^P-complete. □

4.6 Related Work

4.6.1 Comparison with Traditional ILP

The three most common settings for ILP are *learning from entailment*, *learning from interpretations* and *learning from satisfiability* [DR97]. In this section, we discuss how our learning frameworks compare to each of these settings. Due to the differences in the languages of ASP and Prolog, in this section when comparing to traditional frameworks, we assume that all programs considered are normal logic programs such that HB_P^{rel} is finite (with no lists, constraints or choice rules).

We also demonstrate that our three frameworks can simulate ILP_b, ILP_c and ILP_{sm} – the three previous frameworks for learning under the answer set semantics. In Chapter 5 we give a much more detailed analysis of each of the six frameworks and their generality. We show that $ILP_{LOAS}^{context}$ is the most general of the six learning frameworks.

Learning from Entailment

As discussed in Chapter 3, there are two versions of Learning from Entailment under the answer set semantics. These are *brave* and *cautious* induction. We now show that ILP_{LAS} is capable of simulating both. First, Theorem 4.28 shows that ILP_b can be simulated by ILP_{LAS}. The positive and negative examples of an ILP_b task become the inclusions and exclusions of a single positive (partial interpretation) example in an ILP_{LAS} task.

Theorem 4.28. *Let $T_b = \langle B, S_M, \langle E^+, E^- \rangle\rangle$ be an arbitrary ILP_b task. $ILP_b(T_b) = ILP_{LAS}(\langle B, S_M, \langle \{\langle E^+, E^-\rangle\}, \emptyset\rangle\rangle)$.*

Proof. For any $H \subseteq S_M$:

$H \in ILP_b(T_b)$

$\Leftrightarrow \exists A \in AS(B \cup H)$ such that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$

$\Leftrightarrow \exists A \in AS(B \cup H)$ such that A extends $\langle E^+, E^-\rangle$

$\Leftrightarrow H \in ILP_{LAS}(\langle B, S_M, \langle \{\langle E^+, E^-\rangle\}, \emptyset\rangle\rangle)$

□

Theorem 4.29 shows that ILP_c can be simulated by ILP_{LAS}. The translation from cautious induction is slightly more complicated, in that each positive and negative ILP_c example becomes a distinct negative example in ILP_{LAS}. A positive example $\mathtt{a}$ becomes a negative partial interpretation example $\langle \emptyset, \{\mathtt{a}\}\rangle$, where the ILP_c example says that all answer sets should contain $\mathtt{a}$ and the latter (equivalently) says that no answer set should not contain $\mathtt{a}$ (negative examples are translated similarly). An "empty" partial interpretation is needed as a positive example to enforce the condition of ILP_c that $B \cup H$ must be satisfiable.

Theorem 4.29. *Let $T_c = \langle B, S_M, \langle E^+, E^-\rangle\rangle$ be an arbitrary ILP_c task. $ILP_c(T_c) = ILP_{LAS}(\langle B, S_M, \langle \{\langle \emptyset, \emptyset\rangle\}, \{\langle \emptyset, \{\mathtt{e}^+\}\rangle \mid \mathtt{e}^+ \in E^+\} \cup \{\langle \{\mathtt{e}^-\}, \emptyset\rangle \mid \mathtt{e}^- \in E^-\}\rangle\rangle)$.*

Proof. For any $H \subseteq S_M$:

$H \in ILP_c(T_c)$

$\Leftrightarrow B \cup H$ is satisfiable and $\forall A \in AS(B \cup H)$: $E^+ \subseteq A$ and $E^- \cap A = \emptyset$

$\Leftrightarrow B \cup H$ is satisfiable and
$\forall e^+ \in E^+$: $\forall A \in AS(B \cup H)\ e^+ \in A$ and
$\forall e^- \in E^-$: $\forall A \in AS(B \cup H)\ e^- \notin A$

$\Leftrightarrow B \cup H$ is satisfiable and
$\forall \mathtt{e}^+ \in E^+$: $\forall A \in AS(B \cup H)$ A does not extend $\langle \emptyset, \{\mathtt{e}^+\}\rangle$ and
$\forall \mathtt{e}^- \in E^+$: $\forall A \in AS(B \cup H)$ A does not extend $\langle \{\mathtt{e}^-\}, \emptyset\rangle$

$\Leftrightarrow B \cup H$ has an answer set that extends $\langle \emptyset, \emptyset\rangle$ and
$\forall \mathtt{e}^+ \in E^+$: $\nexists A \in AS(B \cup H)$ such that A extends $\langle \emptyset, \{\mathtt{e}^+\}\rangle$ and
$\forall \mathtt{e}^- \in E^-$: $\nexists A \in AS(B \cup H)$ such that A extends $\langle \{\mathtt{e}^-\}, \emptyset\rangle$

$\Leftrightarrow H \in ILP_{LAS}(\langle B, S_M, \langle \{\langle \emptyset, \emptyset\rangle\}, \{\langle \emptyset, \{\mathtt{e}^+\}\rangle \mid \mathtt{e}^+ \in E^+\} \cup \{\langle \{\mathtt{e}^-\}, \emptyset\rangle \mid \mathtt{e}^- \in E^-\}\rangle\rangle)$

□

Learning from Interpretations

Recall from Chapter 3 that the task of ILP_{LFI} is to find a hypothesis H such that for each example set of facts $e \in E^+$, $M(B \cup e)$ is a model of H and for each set of facts $e \in E^-$, $M(B \cup e)$ is not a model of H. $ILP_{LOAS}^{context}$ can be used to simulate the standard learning from interpretations setting by encoding each example set of facts as the context of an example (with no inclusions or exclusions) and converting the clauses in the hypothesis space into constraints. A set of constraints H' is an inductive solution of such a task if for each positive (resp. negative) example e, with context C, $B \cup H' \cup C$ is satisfiable (resp. unsatisfiable). As B is a definite program, C is a set of facts and H' is a set of constraints, $B \cup H' \cup C$ is satisfiable if and only if $M(B \cup C)$ is a model of H'.

Theorem 4.30. *Let $T_{LFI} = \langle B, S_M, \langle E^+, E^-\rangle\rangle$ be an ILP_{LFI} task. Consider the $ILP_{LOAS}^{context}$ task $T' = \langle B, f(S_M), \{\langle\langle \emptyset, \emptyset\rangle, e\rangle \mid e \in E^+\}, \{\langle\langle \emptyset, \emptyset\rangle, e\rangle \mid e \in E^-\}, \emptyset, \emptyset\rangle$, where f maps any set of clauses into a set of constraints, such that each clause $\neg \mathtt{b_1} \vee \ldots \vee \neg \mathtt{b_m} \vee \mathtt{c_1} \vee \ldots \vee \mathtt{c_n}$ becomes the constraint* `:-`$\mathtt{b_1}, \ldots, \mathtt{b_m}$ `not` $\mathtt{c_1}, \ldots,$ `not` $\mathtt{c_n}$. *Then $ILP_{LFI}(T_{LFI}) = \{f^{-1}(H) \mid H \in ILP_{LOAS}^{context}(T')\}$ (where f^{-1} is the inverse mapping of f).*

Proof. Assume that $H \in ILP_{LFI}(T_{LFI})$

$\Leftrightarrow H \subseteq S_M$,
$\forall e \in E^+$, $M(B \cup e)$ is a model of H and
$\forall e \in E^-$, $M(B \cup e)$ is not a model of H

$\Leftrightarrow H \subseteq S_M$,
$\forall e \in E^+$, $M(B \cup e)$ is a model of $f(H)$ and
$\forall e \in E^-$, $M(B \cup e)$ is not a model of $f(H)$
(for any clause h, the models of h are equal to the models of $f(h)$)

$\Leftrightarrow H \subseteq S_M$,
  $\forall e \in E^+$, $B \cup e \cup f(H)$ has at least one answer set and
  $\forall e \in E^-$, $B \cup e \cup f(H)$ has no answer sets

$\Leftrightarrow H \subseteq S_M$,
  $\forall e' \in \{\langle\langle\emptyset,\emptyset\rangle, e\rangle \mid e \in E^+\}$, $B \cup e'_{ctx} \cup f(H)$ has at least one answer set that extends e'_{pi} and
  $\forall e' \in \{\langle\langle\emptyset,\emptyset\rangle, e\rangle \mid e \in E^-\}$, $B \cup e'_{ctx} \cup f(H)$ has no answer sets that extend e'_{pi}

$\Leftrightarrow f(H) \subseteq f(S_M)$,
  $\forall e' \in \{\langle\langle\emptyset,\emptyset\rangle, e\rangle \mid e \in E^+\}$, $B \cup f(H)$ accepts e' and
  $\forall e' \in \{\langle\langle\emptyset,\emptyset\rangle, e\rangle \mid e \in E^-\}$, $B \cup f(H)$ does not accept e'

$\Leftrightarrow f(H) \in ILP^{context}_{LOAS}(T')$

$\Leftrightarrow H \in \{f^{-1}(H') \mid H' \in ILP^{context}_{LOAS}(T')\}$

□

Learning from Satisfiability

The task of ILP_{LFS} is to find a hypothesis such that for each example program $e^+ \in E^+$, $B \cup H \cup e^+$ has at least one model and for each program $e^- \in E^-$, $B \cup H \cup e^-$ has no models. $ILP^{context}_{LOAS}$ can simulate ILP_{LFS} by using context-dependent examples. In standard ILP_{LFS}, the background knowledge, hypothesis space and examples are each sets of clauses. In our translation we transform all clauses in the background knowledge, hypothesis space and examples of the original task into constraints (using the same mapping f as in Theorem 4.30). We also add a choice rule to the background knowledge for each atom in the language of the task (saying that each atom can either be true or false). The effect is that for any hypothesis H, the models of $B \cup H \cup e$ are equal to the answer sets of $B' \cup f(H) \cup f(e)$, where B' is the background knowledge of the translated task. More specifically, $B \cup H \cup e$ is satisfiable if and only of $B' \cup f(H) \cup f(e)$ is satisifable. Hence, a hypothesis H covers an example if and only if $f(H)$ covers the translated example. Theorem 4.31 formalises our translation from an ILP_{LFS} task to an $ILP^{context}_{LOAS}$ task.

Theorem 4.31. *Let $T_{LFS} = \langle B, S_M, \langle E^+, E^-\rangle\rangle$ be an ILP_{LFS} task. Let HB_T be the relevant Herbrand base of the union of all clauses in T_{LFS}. Consider the $ILP^{context}_{LOAS}$ task $T' = \langle f(B) \cup \{\texttt{0\{a\}1.} \mid \texttt{a} \in HB^{rel}_T\}, f(S_M), \{\langle\langle\emptyset,\emptyset\rangle, f(e)\rangle \mid e \in E^+\}, \{\langle\langle\emptyset,\emptyset\rangle, f(e)\rangle \mid e \in E^-\}, \emptyset, \emptyset\rangle$, where f maps any set of clauses into a set of constraints, such that each clause $\neg \texttt{b}_1 \vee \ldots \vee \neg \texttt{b}_\texttt{m} \vee \texttt{c}_1 \vee \ldots \vee \texttt{c}_\texttt{n}$ becomes the constraint $\texttt{:-}\,\texttt{b}_1, \ldots, \texttt{b}_\texttt{m}\ \texttt{not}\ \texttt{c}_1, \ldots,\ \texttt{not}\ \texttt{c}_\texttt{n}$. Then $ILP_{LFS}(T_{LFS}) = \{f^{-1}(H) \mid H \in ILP^{context}_{LOAS}(T')\}$ (where f^{-1} is the inverse mapping of f).*

Proof.

Note that for any program P, $\forall I \subseteq HB^{rel}_T$, I is a model of P if and only if $I \in AS(f(P) \cup \{\texttt{0\{a\}1.} \mid \texttt{a} \in HB^{rel}_T\})$.

Assume that $H \in ILP_{LFS}(T_{LFS})$

$\Leftrightarrow H \subseteq S_M$, $\forall e \in E^+$, $B \cup H \cup e$ has at least one model and $\forall e \in E^-$, $B \cup H \cup e$ has no models

$\Leftrightarrow H \subseteq S_M$, $\forall e \in E^+$, $\exists I \subseteq HB_T^{rel}$ such that I is a model of $B \cup H \cup e$ and $\forall e \in E^-$, $\forall I \subseteq HB_T^{rel}$, I is not a model of $B \cup H \cup e$

$\Leftrightarrow H \subseteq S_M$, $\forall e \in E^+$, $f(B \cup H \cup e) \cup \left\{ \texttt{0\{a\}1.} \mid \texttt{a} \in HB_T^{rel} \right\}$ is satisfiable and $\forall e \in E^-$, $f(B \cup H \cup e) \cup \left\{ \texttt{0\{a\}1.} \mid \texttt{a} \in HB_T^{rel} \right\}$ is unsatisfiable

$\Leftrightarrow H \subseteq S_M$, $\forall e \in E^+$, $f(B) \cup \left\{ \texttt{0\{a\}1.} \mid \texttt{a} \in HB_T^{rel} \right\} \cup f(H) \cup f(e)$ is satisfiable and $\forall e \in E^-$, $f(B) \cup \left\{ \texttt{0\{a\}1.} \mid \texttt{a} \in HB_T^{rel} \right\} \cup f(H) \cup f(e)$ is unsatisfiable

$\Leftrightarrow f(H) \subseteq f(S_M)$, $\forall e' \in \{\langle\langle \emptyset, \emptyset \rangle, f(e)\rangle \mid e \in E^+\}$, $f(B) \cup \left\{ \texttt{0\{a\}1.} \mid \texttt{a} \in HB_T^{rel} \right\} \cup f(H)$ accepts e' and $\forall e' \in \{\langle\langle \emptyset, \emptyset \rangle, f(e)\rangle \mid e \in E^-\}$, $f(B) \cup \left\{ \texttt{0\{a\}1.} \mid \texttt{a} \in HB_T^{rel} \right\} \cup f(H)$ does not accept e'

$\Leftrightarrow f(H) \in ILP_{LOAS}^{context}(T')$

$\Leftrightarrow H \in \{f^{-1}(H') \mid H' \in ILP_{LOAS}^{context}(T')\}$

$\square$

Induction of Stable Models

Induction of stable models is a generalisation of ILP_b, with multiple partial interpretation examples. An ILP_{sm} task is essentially an ILP_{LAS} task with no negative examples. Theorem 4.32 shows that ILP_{sm} can be (trivially) simulated by ILP_{LAS}.

Theorem 4.32. *Let $T_{sm} = \langle B, S_M, \langle E \rangle \rangle$ be an arbitrary ILP_{sm} task. $ILP_{sm}(T_{sm}) = ILP_{LAS}(\langle B, S_M, \langle \{E, \emptyset \rangle \rangle)$*

Proof. For any $H \subseteq S_M$:

$H \in ILP_{sm}(T_{sm})$

$\Leftrightarrow \forall e \in E$: $\exists A \in AS(B \cup H)$ such that e extends A

$\Leftrightarrow H \in ILP_{LAS}(\langle B, S_M, \langle E, \emptyset \rangle \rangle)$

$\square$

4.6.2 Preference Learning

Another field that is related to our work is that of *preference learning* [FH10]. Specifically, our ordering examples could be thought of as a kind of *learning to rank* [FH03, GHH01], where the task is to learn to rank a set of objects given a set of pairwise preference examples over the objects (this task is often referred to as *object ranking*). Approaches for object ranking often use non logic-based machine learning techniques (e.g. support vector machines [Joa02, KHM05] or artificial neural networks [GHH01, RPMB08, RPMS11]). Compared to these, $ILP_{LOAS}^{context}$ shares the same advantages as any ILP approach versus a non logic-based machine learning technique: learned hypotheses are structured, human readable and can express relational concepts such as minimising the instances of particular combinations of predicates. Existing background knowledge can be taken into account to capture predefined concepts and the search can be steered towards particular types of hypotheses using a language bias. In this section we focus on attempts at preference learning that learn human readable representations of preferences.

Conditional Preference Networks

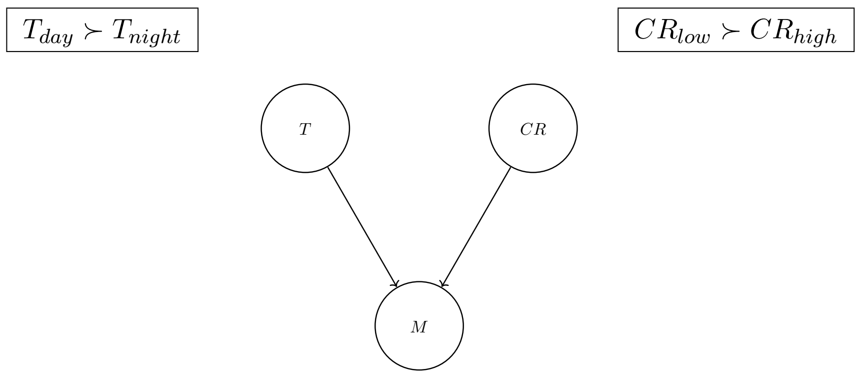
One approach for preference learning that does aim to be human readable is to learn a *conditional preference network* (CP-net) [BBD$^+$04]. Definition 4.10 is the formal definition of a CP-net (first presented in [BBD$^+$04]). Given a set of object instances O with attributes[2] $X_1, \ldots, X_n$, a CP-net induces a partial ordering of the instances in O.

Definition 4.10. A CP-net over the attributes $X_1, \ldots, X_n$ is a directed graph G with the nodes $X_1, \ldots, X_n$. The parent nodes of each node X_i are denoted as $par(X_i)$. Each node X_i is annotated with a *CP table*, which maps each instantiation of the nodes in $par(X_i)$ to a total ordering of the elements in the domain of X_i.

The semantics of CP-nets is that each row in a CP-table specifies which of two objects are preferred if "all other attributes are equal". Informally, a preference ranking over all objects in the domain is said to *satisfy* a CP-net if every row of every CP table is respected. We illustrate this with an example.

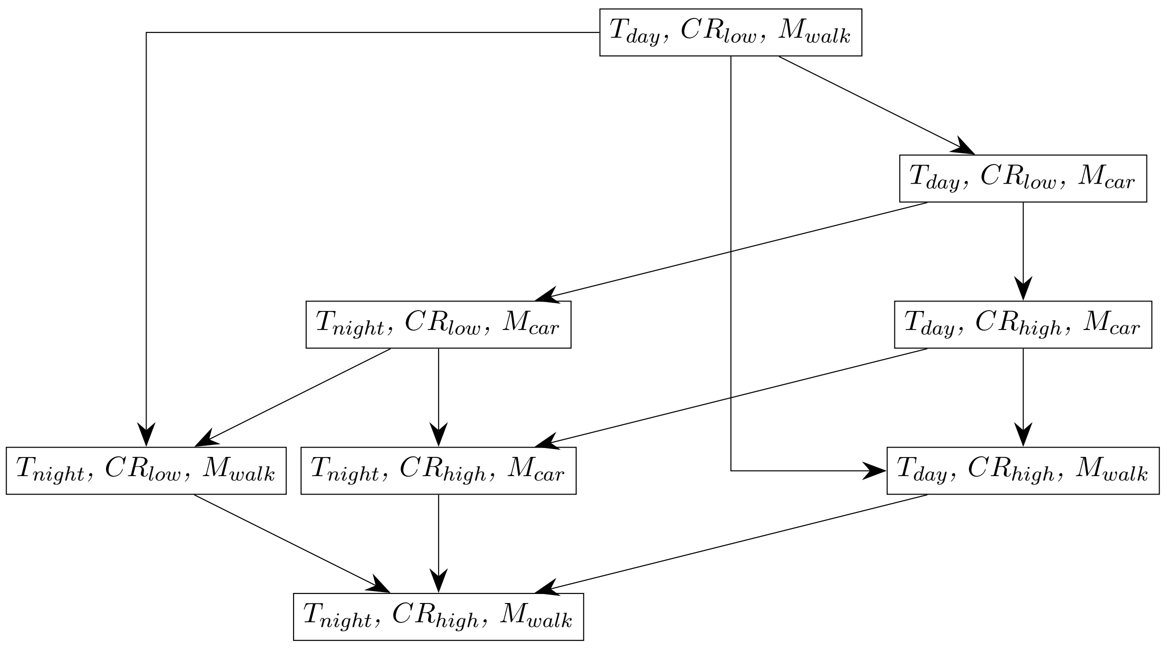
Example 4.33. *Reconsider the task of learning a user's journey preferences, as discussed in Example 4.4. The nodes in the graph below represent the attributes "time of day" (denoted T, with values* `day` *and* `night`*), "crime rating" (denoted CR, with values* `low` *and* `high`*) and "mode of transport" (denoted M, with values* `car` *and* `walk`*). The CP-tables for each node are also shown below. Note that the CP-table for M depends on T and CR, and so it has 4 rows, one for each possible combination of the values for T and CR.*

[2]In the CP-net literature, these are usually referred to as *variables*, but we avoid this term as it has another meaning in this thesis.



$T_{day} \wedge CR_{low}$	$M_{walk} \succ M_{car}$
$T_{night} \wedge CR_{low}$	$M_{car} \succ M_{walk}$
$T_{day} \wedge CR_{high}$	$M_{car} \succ M_{walk}$
$T_{night} \wedge CR_{high}$	$M_{car} \succ M_{walk}$

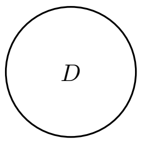
This particular CP-net induces the partial ordering shown below. In the diagram, a path from node A to node B illustrates that A is preferred to B.


$T_{day}, CR_{low}, M_{walk}$
$T_{day}, CR_{low}, M_{car}$
$T_{night}, CR_{low}, M_{car}$
$T_{day}, CR_{high}, M_{car}$
$T_{night}, CR_{low}, M_{walk}$
$T_{night}, CR_{high}, M_{car}$
$T_{day}, CR_{high}, M_{walk}$
$T_{night}, CR_{high}, M_{walk}$

There have been some attempts at learning CP-nets from examples. In [GAG13], a two-phase method is proposed for eliciting CP-nets in an iterative process, by querying a user about their preferences. In the first phase, a user is asked about their preferences over single attributes, and then in the second phase, they are asked about their preferences over full objects. The first phase is used to build an initial CP-net with no edges (all attibutes are assumed to be "unconditional"), whereas the second phase is used to find the conditions and edges. In [LXW^{+}14], CP-nets are learned from a set of pairwise preference examples. Interestingly, this set of examples can be *inconsistent* (i.e. the examples can imply that, given a pair of objects o_1 and o_2, $o_1 \succ o_2$ and $o_2 \succ o_1$). The first step of the algorithm is to find a maximal consistent subset of the examples, together with a partial ordering of the objects in the domain (which implies this maximal subset of the examples). The partial ordering of objects is then transformed into a CP-net.

One drawback to CP-nets is that they are propositional. For example, to represent that an attribute should be minimised, the CP table for that attribute would need to list every value in order. While this is possible, in order to learn such a CP table, examples of every instance would need to be seen.

Example 4.34. *Consider a simplified scenario, with distance (denoted D, with values $1, \ldots, 4$) as the only attribute. A CP-net which expresses a wish to minimise the distance is:*



$$\boxed{D_1 \succ D_2 \succ D_3 \succ D_4}$$

In order to incentivise learning the CP table for D, there would need to be at least 3 examples, to show that $D_1 \succ D_2$, $D_2 \succ D_3$ and $D_3 \succ D_4$. In fact, the number of examples required would grow linearly with the size of the domain of D. In contrast, $ILP^{context}_{LOAS}$ could learn the first order weak constraint "`:~ distance(D).[D@1]`*" from few examples, and does not need more examples as the domain of D grows. This is the case in general for ILP approaches, which generalise from few examples to learn first order theories (rather than learning propositional theories for each example, which tends to result in a longer hypothesis). In [Yd07],* more-or-less *CP-nets were introduced, which allowed some attributes to be labeled as* monotonic*, meaning that their preference ordering was decided by a predetermined total ordering over the domain, either from lowest to highest, or from highest to lowest. In this example, such a framework would be able to represent that there are only two options: either the distance should be maximised, or it should be minimised. At the time of writing, we are not aware of any work on* learning *more-or-less CP-nets.*

Another drawback of CP-nets is that they only reason directly on the attributes, and cannot combine attributes; for instance, in Example 4.4, we were interested in minimising the *total* walking distance, rather than the distance in each leg. The penalties in weak constraints allow us to easily express that the distances of all walking legs should be summed, but a CP-net would require the total walking distance to be given as an attribute.

Relational CP-nets [Kor12] allow the use of *aggregators* such as taking the mode or median of the set of values for an attribute. Although this does allow sets of values to be combined using the aggregators, this does not overcome the issues above in general, as it does not allow different attributes to be combined.

Answer Set Optimisation

Weak constraints are one way of expressing preferences over answer sets. Other constructs include the more complex preferences used by the *asprin* solver [BDRS15], which can express, for example, that the optimal answer sets are those that are subset-minimal. In [BNT03], the notion of *preference programs* was first introduced as part of the ASO framework. Preference programs are sets of rules of the form $\texttt{C}_1 > \ldots > \texttt{C}_\texttt{n}$ `:-body.`, where `body` is a normal ASP rule body, and $C_1, \ldots, C_n$ are boolean formulas such as $(\texttt{a} \vee \texttt{b}) \wedge$ `not c`. The meaning of such a preference rule is that if `body` is satisfied then C_1 is preferred to C_2, and so on. Preference programs are solved together with an ASP program, and the goal is to find the optimal answer set(s) given the preferences in the preference program. We decided in our frameworks to use weak constraints rather than these alternative representations of preferences, as they are part of the ASP-Core-2 standard [CFG+13], and are implemented in standard solvers such as clingo [GKK+11] and DLV [LPF+06]. In principle any representation that creates a preference ordering over answer sets could be learned using the $ILP_{LOAS}^{context}$ framework, but the algorithms in this thesis are specifically targeted at learning weak constraints.

4.6.3 Comparison with Probabilistic ILP

One of the advantages to learning ASP programs rather than Prolog programs is that ASP allows the modeling of non-determinism, either through unstratified negation or through choice rules. For example our ILP_{LAS} framework can learn that a coin can be either heads or tails, but not both (the hypothesis would be {`1{heads,tails}1.`}.

Another method for achieving non-determinism in ILP is by adding probabilities. Probabilistic Inductive Logic Programming [DRK04] is a combination of ILP and with probabilistic reasoning. Its aim is to learn a logic program that is annotated with probabilities. The task of PILP is often divided into *structure learning*, where the underlying logic program is learned, and *parameter estimation* or *weight learning*, where the probabilities are learned. A key difference between ILP_{LAS} and PILP is that while both aim to learn programs which are non-deterministic, ILP_{LAS} aims to learn programs whose answer sets capture the set of possibilities, whereas PILP aims to learn a probability distribution over these possibilities.

Although there has been significant progress in the field of PILP [DRKT07, DRT10, BR15a, RBZ+16, RBZ14] for learning annotated Prolog programs, PILP under the answer set semantics is still relatively young, and thus, there are few approaches. PrASP [NM14, NM15, Nic16] considers the problem of weight learning, and in fact uses a similar example of learning about coins. This example illustrates the difference between weight learning and standard ILP. In ILP our task is to learn that there are exactly two possibilities (`heads` and `tails`); whereas in weight learning, the goal is to estimate probabilities of each possibility. PROBXHAIL [DRB+16] combines structure learning and weight learning, but it can only learn definite logic programs.

4.6.4 Related Complexity Results

The complexity of ILP_b and ILP_c for verification and satisfiability were investigated in [SI09]. However, in that work, the results on satisfiability are for deciding whether or not a task has any solutions with no restrictions on the hypothesis space. This means that for both ILP_b and ILP_c deciding whether a task is satisfiable is equivalent to checking whether there is a model of B in which the examples are covered (a simpler decision problem). For this reason, the complexity of satisfiability for propositional ILP_c in [SI09] was NP-complete, rather than Σ_2^P-complete. The complexities for the verification of a hypothesis given in [SI09] are also different from the ones in this chapter, as they consider a different language for $B \cup H$. [SI09] considers disjunctive logic programs, whereas we investigated the complexity of learning programs without disjuction. The reason we chose not to consider disjunctive logic programs is that the systems available for ILP under the answer set semantics (including our own ILASP systems) are not targeted at learning with disjunction. For example, ASPAL [CRL12] does not allow any disjunction, and although XHAIL [BR15b] allows disjunction in the background knowledge, it does not support learning disjunctive rules.

4.6.5 Summary

In this chapter, we introduced our three frameworks for learning ASP programs. We motivated the need for such frameworks, by showing that the existing frameworks for ILP under the answer set semantics are not capable of learning some ASP programs (or at least, no task of these frameworks could have the programs as optimal solutions). In the next chapter, we explore in much more detail what it means for a framework to be capable of learning a class of ASP programs and introduce three new measures of the *generality* of a learning framework. We present the generality results for our own frameworks and the main three existing frameworks under the answer set semantics.

Chapter 5

Generality

In this chapter, we present a new notion of the *generality* of a learning framework. The aim is to characterise the class of ASP programs that a framework is capable of learning, if given sufficient examples. Language biases tend, in general, to impose their own restrictions on the classes of program that can be learned. They are primarily used to aid the performance of the computation, rather than to capture intrinsic properties of a learning framework. In this chapter we will therefore consider learning tasks with unrestricted hypothesis spaces: hypotheses can be constructed from any set of normal rules, choice rules and hard and weak constraints. We assume each learning framework $\mathcal{F}$ to have a task consisting of a pair $\langle B, E_{\mathcal{F}}\rangle$, where B is the (ASP) background knowledge and $E_{\mathcal{F}}$ is a tuple consisting of the examples for this framework; for example E_{LAS}[1] $= \langle E^{+}, E^{-}\rangle$ where E^{+} and E^{-} are sets of partial interpretations.

Allowing an unrestricted hypothesis space means that we can now focus on analysing whether a learning framework is general enough to define learning tasks that has a particular set of hypotheses as the inductive solutions. In the first instance, we could define that a framework $\mathcal{F}$ is general enough to learn a hypothesis H if there is at least one task $T_{\mathcal{F}}$ in this framework such that H is an inductive solution of $T_{\mathcal{F}}$. However, such a "loose" notion of generality may lead to the trivial learning framework, whose learning tasks have no examples, as the most general framework possible. This is shown in the following example.

Example 5.1. *Consider the trivial learning framework $ILP_{\top}$ whose learning tasks are pairs $\langle B, E_{\top}\rangle$, where $E_{\top}$ is the empty tuple and B is an ASP program. $ILP_{\top}(\langle B, E_{\top}\rangle)$ is then the set of all ASP programs, i.e. every ASP program is a solution of every $ILP_{\top}$ task. For every background knowledge B and hypothesis H there is clearly a task $\langle B, E_{\top}\rangle$ such that $H \in ILP_{\top}(\langle B, E_{\top}\rangle)$. However, every other possible hypothesis H' is also a solution of this same task, which makes it impossible to distinguish one hypothesis from another.*

It is clearly not sufficient to say that a framework is general enough to learn some target hypothesis

[1]Note that to avoid cumbersome notation, we denote this E_{LAS} rather than $E_{ILP_{LAS}}$.

(denoted from now on as H_T) if we can find at least one learning task with H_T as a solution. What this definition lacks is a way to express that H_T is a solution of a task T, but that some other (unwanted) hypothesis is not a solution of T. To capture this property of a learning framework we should be able to say that a task T can *distinguish* a target hypothesis H_T from an unwanted hypothesis. Pairs of target and unwanted hypotheses that can be distinguished from each other, are an interesting starting point when considering the generality of a learning framework. But this again might not be the only property of generality. Frameworks, such as brave induction, may be able to distinguish a target hypothesis H_T from two (or more) unwanted hypotheses, e.g., H'_1 and H'_2, in two separate learning tasks, but they may not have a single learning task capable of accepting H_T as inductive solution but neither H'_1 nor H'_2. Consider for instance the following example.

Example 5.2. *Imagine the scenario where we are observing a coin being tossed several times. Obviously there are two outcomes, and we would like to learn an ASP program whose answer sets correspond to these two different outcomes. Consider the background knowledge B to be empty, and the atoms* `heads` *and* `tails` *to be true when the coin lands on heads or tails respectively. Our target hypothesis H_T is an ASP program such that $AS(B \cup H) = \{\{\texttt{heads}\}, \{\texttt{tails}\}\}$. One such hypothesis could be the program $H_T = \{\texttt{1\{heads, tails\}1.}\}$. Consider now the two hypotheses $H'_1 = \{\texttt{heads.}\}$ and $H'_2 = \{\texttt{tails.}\}$, which correspond to the coin always landing on heads or tails respectively. Neither of these hypothesis correctly represent the behaviour of the coin, so they are unwanted hypotheses. There is one answer set, $\{\texttt{heads}\}$, of $B \cup H'_1$ and one answer set, $\{\texttt{tails}\}$, of $B \cup H'_2$. ILP_b can distinguish H_T from H'_1 and from H'_2 separately with the tasks $\langle B, \langle\{\texttt{tails}\}, \emptyset\rangle\rangle$ and $\langle B, \langle\{\texttt{heads}\}, \emptyset\rangle\rangle$, respectively. But there is, however, no learning task for ILP_b for which H_T is an inductive solution and neither H'_1 nor H'_2 is.*

A more general notion of generality of a learning framework can be considered, which looks at distinguishing a target hypothesis H_T from a set of unwanted hypotheses S. In Section 5.2 we introduce the notion of the *one-to-many-distinguishability class* of a learning framework. This corresponds to the class of pairs consisting of a single hypothesis H_T and a set S of hypotheses for which a learning framework has at least one task that distinguishes H_T from each hypothesis in S. Informally, this notion expresses the generality of a framework in finding a single target hypothesis in the presence of many unwanted hypotheses. In Section 5.3, we also extend the one-to-many-distinguishability class of a learning framework to *many-to-many-distinguishability*, which captures the notion of distinguishing a set of target hypotheses S_1 from another set of unwanted hypotheses S_2, with a single task.

In the remainder of this chapter we explore these three new measures of generality, expressed as three different learning problems. One-to-one-distinguishability determines the hypotheses that a framework is general enough to learn, while ruling out another unwanted hypothesis; one-to-many-distinguishability determines the hypotheses that can be learned from within a space of unwanted hypotheses; and finally, many-to-many-distinguishability determines exactly which sets of hypotheses can be learned. We will prove properties of our three classes of generalities making use of a definition of *strong reduction* from one framework to another. Strong reduction is different from the concept of reduction presented in [DR97]. Definitions 5.1 and 5.2 present, respectively, a reformulation of the notion of reduction introduced in [DR97] and of our new concept of strong reduction.

Definition 5.1. A framework $\mathcal{F}_1$ *reduces to* $\mathcal{F}_2$ (written $\mathcal{F}_1 \rightarrow_r \mathcal{F}_2$) if for every $\mathcal{F}_1$ task $T_{\mathcal{F}_1}$ there is an $\mathcal{F}_2$ task $T_{\mathcal{F}_2}$ such that $ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1}) = ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$. A framework $\mathcal{F}_1$ is *at least as r-general as* $\mathcal{F}_2$ if $\mathcal{F}_2 \rightarrow_r \mathcal{F}_1$; and $\mathcal{F}_1$ is *more r-general than* $\mathcal{F}_2$ if $\mathcal{F}_2 \rightarrow_r \mathcal{F}_1$ and $\mathcal{F}_1 \not\rightarrow_r \mathcal{F}_2$.

Example 5.3. *Consider the ILP_b and ILP_c learning frameworks. $ILP_b \rightarrow_r ILP_c$, as any ILP_b task $\langle B, \langle E^+, E^- \rangle\rangle$ maps to the ILP_c task $\langle B \cup \{\texttt{:- not } e. \mid e \in E^+\} \cup \{\texttt{:-} e. \mid e \in E^-\}, \langle \emptyset, \emptyset \rangle\rangle$. ILP_c does not, however, reduce to ILP_b. Consider, for instance, the ILP_c task $T_c = \langle \emptyset, \langle \{\texttt{p}\}, \emptyset \rangle\rangle$ and assume that there is a task $T_b = \langle B, \langle E^+, E^- \rangle\rangle$ such that $ILP_b(T_b) = ILP_c(T_c)$. The hypothesis $H_1 = \{\texttt{p.}\} \in ILP_c(T_c)$, and, given the assumption, H_1 is also in $ILP_b(T_b)$. But consider now the hypothesis $H_2 = \{\texttt{0\{p\}1.}\}$. Since $AS(B \cup H_1) \subseteq AS(B \cup H_2)$, if $B \cup H_1$ has an answer set extending $\langle E^+, E^- \rangle$, then so does $B \cup H_2$. Thus, if $H_1 \in ILP_b(T_b)$ then $H_2 \in ILP_b(T_b)$. But, although $H_2 \in ILP_b(T_b)$, it is easy to see that $H_2 \notin ILP_c(T_c)$, so $ILP_b(T_b)$ is not equal to $ILP_c(T_c)$. Hence, ILP_c does not reduce to ILP_b, and ILP_c is therefore more r-general than ILP_b.*

We discuss the relationship between reductions and our own measures of generality in Section 5.5. Our notion of strong reduction differs from the above notion of reduction, in the fact that the reduced task must have the same background knowledge as the original task.

Definition 5.2. A framework $\mathcal{F}_1$ *strongly reduces to* $\mathcal{F}_2$ (written $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$) if for every $\mathcal{F}_1$ task $T_{\mathcal{F}_1} = \langle B, E_{\mathcal{F}_1} \rangle$ there is an $\mathcal{F}_2$ task $T_{\mathcal{F}_2} = \langle B, E_{\mathcal{F}_2} \rangle$ such that $ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1}) = ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$. A framework $\mathcal{F}_1$ is *at least as sr-general as* $\mathcal{F}_2$ if $\mathcal{F}_2 \rightarrow_{sr} \mathcal{F}_1$; and $\mathcal{F}_1$ is *more sr-general than* $\mathcal{F}_2$ if $\mathcal{F}_2 \rightarrow_{sr} \mathcal{F}_1$ and $\mathcal{F}_1 \not\rightarrow_{sr} \mathcal{F}_2$.

Proposition 5.4 shows the strong reduction relations between the frameworks considered in this chapter. Note that although ILP_c is more r-general than ILP_b (as shown in Example 5.3), it is not more sr-general than ILP_b. This is because without changing the background knowledge, ILP_c cannot represent the same ILP_b tasks.

Proposition 5.4.

1. $ILP_b \rightarrow_{sr} ILP_{sm} \rightarrow_{sr} ILP_{LAS} \rightarrow_{sr} ILP_{LOAS} \rightarrow_{sr} ILP_{LOAS}^{context}$
2. $ILP_c \rightarrow_{sr} ILP_{LAS}$

Proof.

1. For any ILP_b task $T_b = \langle B, \langle E^+, E^- \rangle\rangle$, $ILP_b(T_b) = ILP_{sm}(\langle B, \langle \{\langle E^+, E^- \rangle\}\rangle\rangle)$
   For any ILP_{sm} task $T_{sm} = \langle B, \langle \{e_1, \ldots, e_n\}\rangle\rangle$, $ILP_{sm}(T_{sm}) = ILP_{LAS}(\langle B, \langle \{e_1, \ldots, e_n\}, \emptyset \rangle\rangle)$
   For any ILP_{LAS} task $T_{LAS} = \langle B, \langle E^+, E^- \rangle\rangle$, $ILP_{LAS}(T_{LAS}) = ILP_{LOAS}(\langle B, \langle E^+, E^-, \emptyset, \emptyset \rangle\rangle)$

For any partial interpretation e, let $c(e)$ be the CDPI $\langle e, \emptyset\rangle$. For any ILP_{LOAS} task $T_{LOAS} = \langle B, \langle E^{+}, E^{-}, O^{b}, O^{c}\rangle\rangle$, $ILP_{LOAS}(T_{LOAS}) = ILP^{context}_{LOAS}(\langle B, \langle\{c(e) \mid e \in E^{+}\}, \{c(e) \mid e \in E^{-}\}, \{\langle c(e_1), c(e_2)\rangle \mid \langle e_1, e_2\rangle \in O^{b}\}, \{\langle c(e_1), c(e_2)\rangle \mid \langle e_1, e_2\rangle \in O^{c}\}\rangle\rangle)$

2. For any ILP_c task $T_c = \langle B, \langle\{e^{+}_{1}, \ldots, e^{+}_{m}\}, \{e^{-}_{1}, \ldots, e^{-}_{n}\}\rangle\rangle$, $ILP_c(T_c) = ILP_{LAS}(\langle B, \langle\{\langle\emptyset, \emptyset\rangle\}, \{\langle\emptyset, \{e^{+}_{1}\}\rangle, \ldots, \langle\emptyset, \{e^{+}_{m}\}\rangle, \langle\{e^{-}_{1}\}, \emptyset\rangle, \ldots, \langle\{e^{-}_{n}\}, \emptyset\rangle\}\rangle\rangle)$. Note that the empty ILP_{LAS} positive example enforces that there is at least one answer set, and both the ILP_c positive and negative examples are mapped to ILP_{LAS} negative examples which enforce in the case of positive (resp. negative) examples that they are not false (resp. not true) in any answer set, and hence true (resp. false) in every answer set.

$\square$

5.1 Distinguishability

A one-to-one-distinguishability class captures those pairs of hypotheses H_1 and H_2 that can be distinguished from each other with respect to a given possible background knowledge.

Definition 5.3. The *one-to-one-distinguishability class* of a learning framework $\mathcal{F}$ (denoted $\mathcal{D}^{1}_{1}(\mathcal{F})$) is the set of tuples $\langle B, H_1, H_2\rangle$ of ASP programs for which there is at least one task $T_{\mathcal{F}} = \langle B, E_{\mathcal{F}}\rangle$ such that $H_1 \in ILP_{\mathcal{F}}(T_{\mathcal{F}})$ and $H_2 \notin ILP_{\mathcal{F}}(T_{\mathcal{F}})$. For each $\langle B, H_1, H_2\rangle \in \mathcal{D}^{1}_{1}(\mathcal{F})$, $T_{\mathcal{F}}$ is said to *distinguish* H_1 from H_2 with respect to B. Given two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, we say that $\mathcal{F}_1$ is at least as (resp. more) $\mathcal{D}^{1}_{1}$-general as (resp. than) $\mathcal{F}_2$ if $\mathcal{D}^{1}_{1}(\mathcal{F}_2) \subseteq \mathcal{D}^{1}_{1}(\mathcal{F}_1)$ (resp. $\mathcal{D}^{1}_{1}(F_2) \subset \mathcal{D}^{1}_{1}(F_1)$).

Note that the one-to-one-distinguishability relationship is not symmetric; i.e there are pairs of hypotheses H_1 and H_2 such that, given a background knowledge B, H_1 can be distinguished from H_2, but H_2 can not be distinguished from H_1. This is illustrated by Example 5.5.

Example 5.5. *Consider a background knowledge B that defines the concepts of cell, same_block, same_row and same_column for a* 4x4 *Sudoku grid.*

Let H_1 be the incomplete description of the Sudoku rules:

```
1 { value(C, 1), value(C, 2), value(C, 3), value(C, 4) } 1 :- cell(C).
:- value(C1, V), value(C2, V), same_row(C1, C2).
:- value(C1, V), value(C2, V), same_col(C1, C2).
```

Also let H_2 be the complete description of the Sudoku rules:

```
1 { value(C, 1), value(C, 2), value(C, 3), value(C, 4) } 1 :- cell(C).
:- value(C1, V), value(C2, V), same_row(C1, C2).
:- value(C1, V), value(C2, V), same_col(C1, C2).
:- value(C1, V), value(C2, V), same_block(C1, C2).
```

ILP_b can distinguish H_1 from H_2 with respect to B. This can be seen using the task $\langle B, \langle \{\texttt{value((1,1),1)}, \texttt{value((2,2),1)}\}, \emptyset\rangle\rangle$. On the other hand, ILP_b cannot distinguish H_2 from H_1. Whatever examples are given in a learning task to learn H_2, it must be the case that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$, where A is an answer set of $B \cup H_2$. But answer sets of $B \cup H_2$ are also answer sets of $B \cup H_1$. So A is also an answer set of $B \cup H_1$, which implies that H_1 satisfies the same examples and is a solution of the same learning task.

In fact, Proposition 5.6 generalises Example 5.5 showing that ILP_b cannot distinguish any program containing a constraint from the same program without the constraint.

Proposition 5.6. ILP_b cannot distinguish any hypothesis H which contains a constraint C from $H\backslash\{C\}$, with respect to any background knowledge.

Proof. Assume for contradiction that there is a hypothesis $H = H' \cup C$ where C is a constraint and an ILP_b task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H \in ILP_b(T_b)$ and $H' \notin ILP_b(T_b)$.

$\Rightarrow \exists A \in AS(B \cup H)$ such that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$. But as C is a constraint $AS(B \cup H) \subseteq AS(B \cup H')$ and so $A \in AS(B \cup H')$.

$\Rightarrow \exists A \in AS(B \cup H')$ such that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$.

$\Rightarrow H' \in ILP_b(T_b)$. Contradiction! $\square$

One useful property is that if there is a strong reduction from one framework $\mathcal{F}_1$ to another framework $\mathcal{F}_2$ then $\mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2)$. Note that $\mathcal{F}_2$ is not guaranteed to be more $\mathcal{D}^1_1$-general than $\mathcal{F}_1$, even in the case when there is no reduction from $\mathcal{F}_2$ to $\mathcal{F}_1$.

Proposition 5.7. For any two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$: $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2 \Rightarrow \mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2)$.

Proof. Assume that $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$. Take any $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_1)$. There must be some task $T_{\mathcal{F}_1}$, with background knowledge B, such that $H_1 \in ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1})$ and $H_2 \notin ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1})$. Hence, as $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$, there must be some task $T_{\mathcal{F}_2}$, with background knowledge B, such that $H_1 \in ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$ and $H_2 \notin ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$. So $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_2)$. Hence, $\mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2)$. $\square$

As there are clear strong reductions (shown by Proposition 5.4), an ordering of the one-to-one-distinguishability classes of the frameworks emerges (shown in Corollary 5.8).

Corollary 5.8.

1. $\mathcal{D}^1_1(ILP_b) \subseteq \mathcal{D}^1_1(ILP_{sm}) \subseteq \mathcal{D}^1_1(ILP_{LAS}) \subseteq \mathcal{D}^1_1(ILP_{LOAS}) \subseteq \mathcal{D}^1_1(ILP^{context}_{LOAS})$
2. $\mathcal{D}^1_1(ILP_c) \subseteq \mathcal{D}^1_1(ILP_{LAS})$

Framework $\mathcal{F}$	Sufficient/necessary condition for $\langle B, H_1, H_2\rangle$ to be in $\mathcal{D}^1_1(\mathcal{F})$
$ILP_{\top}$	$\bot$
ILP_b	$AS(B \cup H_1) \not\subseteq AS(B \cup H_2)$
ILP_{sm}	$AS(B \cup H_1) \not\subseteq AS(B \cup H_2)$
ILP_c	$AS(B \cup H_1) \neq \emptyset \wedge (AS(B \cup H_2) = \emptyset \vee (\mathcal{E}_c(B \cup H_1) \not\subseteq \mathcal{E}_c(B \cup H_2)))$
ILP_{LAS}	$AS(B \cup H_1) \neq AS(B \cup H_2)$
ILP_{LOAS}	$(AS(B \cup H_1) \neq AS(B \cup H_2)) \vee (ord(B \cup H_1) \neq ord(B \cup H_2))$
$ILP^{context}_{LOAS}$	$(B \cup H_1 \not\equiv^s B \cup H_2) \vee (\exists C \in \mathcal{ASP}^{ch}$ st $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C))$

Table 5.1: A summary of the sufficient and necessary conditions in each learning framework for a hypothesis H_1 to be distinguishable from another hypothesis H_2 with respect to a background knowledge B.

While this does give us information about the ordering of the power of the frameworks to distinguish between hypotheses, it does not tell us, for example, what the relationship is between the distinguishability classes of ILP_b and ILP_c. It does not tell us which of the $\subseteq$'s are strict (in fact, $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm})$, but the rest are strict subset relations). For each framework, Table 5.1 shows the necessary and sufficient condition needed to be able to distinguish hypotheses. In the case of the cautious induction framework, the condition makes use of a new notation.

Notation ($\mathcal{E}_b$ and $\mathcal{E}_c$). Given a program P, $\mathcal{E}_b(P) = \{\texttt{i}_1 \wedge \ldots \wedge \texttt{i}_\texttt{m}, \wedge$ `not` $\texttt{e}_1 \wedge \ldots \wedge$ `not` $\texttt{e}_\texttt{n} \mid \exists A \in AS(P)$ st $\texttt{i}_1, \ldots, \texttt{i}_\texttt{m} \in A$ and $\texttt{e}_1, \ldots, \texttt{e}_\texttt{n} \not\in A\}$, i.e. $\mathcal{E}_b(P)$ denotes the set of conjunctions of literals that are true in at least one answer set of P. Similarly, we use $\mathcal{E}_c(P)$ to denote the set of conjunctions of literals that are true in every answer set of P.

Proposition 5.9. *(proof on page 278)*

For any programs P_1 and P_2, $\mathcal{E}_b(P_1) \subseteq \mathcal{E}_b(P_2)$ if and only if $AS(P_1) \subseteq AS(P_2)$.

Propositions 5.10 to 5.17 prove the one-to-one-distinguishability classes of ILP_b, ILP_{sm}, ILP_c, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$, showing also the sufficient and necessary conditions for distinguishability presented in Table 5.1.

Proposition 5.10. *(proof on page 278)*

$\mathcal{D}^1_1(ILP_b) = \{\langle B, H_1, H_2\rangle \mid AS(B \cup H_1) \not\subseteq AS(B \cup H_2)\}$

Interestingly, although $ILP_{sm} \not\rightarrow_{sr} ILP_b$, $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm})$. This is shown by Proposition 5.11. The reason for this is that if ILP_{sm} can distinguish one hypothesis H_1 from another hypothesis H_2 then there must be some task T_{sm} such that H_1 is a solution of T_{sm} and H_2 is not. This means that H_1 must cover all of the examples of T_{sm} and there must be at least one (partial interpretation) example of T_{sm} which is not covered by H_2. This partial interpretation example can be given as the set of positive and negative examples in an ILP_b task. This ILP_b task will then distinguish H_1 from H_2.

Proposition 5.11. *(proof on page 278)* $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm})$.

To better compare the conditions for ILP_b and ILP_c, we can express the necessary and sufficient condition of ILP_b in terms of the notion $\mathcal{E}_b(P)$. Specifically, in ILP_b for one hypothesis H_1 to be distinguishable from another hypothesis H_2 (with respect to a background knowledge B) it is both necessary and sufficient for $\mathcal{E}_b(B \cup H_1)$ to contain at least one conjunction that is not in $\mathcal{E}_b(B \cup H_2)$. This is because the extra conjunction can be used to generate a set of examples that are covered by H_1 but not H_2. This is demonstrated by Example 5.12.

Example 5.12. *Consider again the programs $B = \emptyset$, $H_1 = \{\texttt{1\{heads,tails\}1.}\}$ and $H_2 = \{\texttt{heads.}\}$. $\mathcal{E}_b(B \cup H_1)$ contains the conjunction* `not heads`$\wedge$`tails`*, whereas $\mathcal{E}_b(B \cup H_2)$ does not. This conjunction can be mapped into the positive example* `tails` *and the negative example* `heads`*, which $B \cup H_1$ covers, but $B \cup H_2$ does not – i.e. the task $\langle B, \langle \{\texttt{tails}\}, \{\texttt{heads}\} \rangle \rangle$ distinguishes H_1 from H_2.*

So, as the one-to-one-distinguishability condition for ILP_b could also be expressed as $\mathcal{E}_b(B \cup H_1) \not\subseteq \mathcal{E}_b(B \cup H_2)$, it might be expected that the one-to-one-distinguishability condition for ILP_c would be that $\mathcal{E}_c(B \cup H_1) \not\subseteq \mathcal{E}_c(B \cup H_2)$. Indeed this would be the case, if it were not for the extra condition that ILP_c imposes on any inductive solution: that is, any inductive solution H must be such that $B \cup H$ is satisfiable. Although this extra condition may seem unnecessary at first sight, its importance becomes clear when considering distinguishability. Without this extra condition, no hypothesis would be distinguishable from the hypothesis given by the empty constraint ":-." – i.e. there would be no hypothesis H such that $\langle B, H, \{\texttt{:-.}\} \rangle \in \mathcal{D}^1_1(ILP_c)$ (for any B). This is because there cannot be any answer set of $B \cup \{\texttt{:-.}\}$ that does not cover the examples (as there are no answer sets). As ILP_c has the extra condition that $B \cup H$ must be satisfiable, its distinguishability condition is slightly more complicated than $\mathcal{E}_c(B \cup H_1) \not\subseteq \mathcal{E}_c(B \cup H_2)$, as shown in Proposition 5.13.

Proposition 5.13. *(proof on page 279)*

$$\mathcal{D}^1_1(ILP_c) = \left\{ \langle B, H_1, H_2 \rangle \middle| \begin{array}{c} AS(B \cup H_1) \neq \emptyset \wedge \\ (AS(B \cup H_2) = \emptyset \vee \mathcal{E}_c(B \cup H_2) \not\subseteq \mathcal{E}_c(B \cup H_1)) \end{array} \right\}$$

We now prove the one-to-one-distinguishability classes of our own frameworks, ILP_{LAS} and ILP_{LOAS}. $\mathcal{D}^1_1(ILP_{LAS})$ contains both $\mathcal{D}^1_1(ILP_b)$ and $\mathcal{D}^1_1(ILP_c)$ as ILP_{LAS} can distinguish any two hypotheses which, combined with the background knowledge, have different answer sets.

Proposition 5.14. *(proof on page 279)* $\mathcal{D}^1_1(ILP_{LAS}) = \{\langle B, H_1, H_2 \rangle | AS(B \cup H_1) \neq AS(B \cup H_2)\}$

As shown in Theorem 5.18, ILP_{LOAS} is more $\mathcal{D}^1_1$-general than ILP_{LAS}. This is because ILP_{LOAS} is able to use its ordering examples to distinguish any two hypotheses that, when combined with the background knowledge, order their answer sets differently, even if the two programs have the same answer sets.

Proposition 5.15. *(proof on page 280)*

$$\mathcal{D}^1_1(ILP_{LOAS}) = \left\{ \langle B, H_1, H_2\rangle \;\middle|\; \begin{array}{c} AS(B \cup H_1) \neq AS(B \cup H_2) \text{ or} \\ ord(B \cup H_1) \neq ord(B \cup H_2) \end{array} \right\}$$

Note that we assume ILP_{LOAS} to be able to give ordering examples with any of the binary ordering operators. The slightly more restrictive version of ILP_{LOAS}, presented in [LRB15a] where the operator is always $<$, has a smaller one-to-one-distinguishability class. This is shown in Example 5.16.

Example 5.16. *Consider the heads and tails problem again, where* $B = \left\{ \texttt{1\{heads, tails\}1.} \right\}$*, and two potential hypotheses:*

- $H_1 = \emptyset$
- $H_2 = \{$:~ `heads.`$[1@1]\}$

$AS(B \cup H_1) = AS(B \cup H_2) = \{\{\texttt{heads}\}, \{\texttt{tails}\}\}$. *If we consider the restricted* ILP_{LOAS} *where only the operator* $<$ *is used to express ordering over the examples,* H_2 *can be distinguished from* H_1*, but* H_1 *cannot be distinguished from* H_2*. This is because all answer sets of* $B \cup H_1$ *are equally optimal – neither* $\langle\{\texttt{tails}\}, \{\texttt{heads}\}, <\rangle$ *nor* $\langle\{\texttt{heads}\}, \{\texttt{tails}\}, <\rangle$ *is in* $ord(B \cup H_1)$*. In contrast, if we allow the use of any of the binary ordering operators, we can consider a task with the ordering example* $\langle\langle\{\texttt{tails}\}, \emptyset\rangle, \langle\{\texttt{heads}\}, \emptyset\rangle, =\rangle$ *and be able to distinguish* H_1 *from* H_2*. The learned hypothesis* H_1 *has no weak constraints, so the two answer sets are equally optimal and the ordering example is respected by* H_1*, whereas* H_2 *prefers* $\{\texttt{tails}\}$ *to* $\{\texttt{heads}\}$*.*

ILP_{LOAS} can distinguish any two hypotheses that, when combined with a fixed background knowledge, behave differently. It cannot distinguish hypotheses that are different but behave the same with respect to the background knowledge. This means that there are some hypotheses that are not strongly equivalent (when combined with the background knowledge), but ILP_{LOAS} cannot distinguish one from the other. We now show that $ILP^{context}_{LOAS}$ *can* distinguish between any two hypotheses, H_1 and H_2, that, when combined with the background knowledge, are not strongly equivalent, or for which there is at least one program $C \in \mathcal{ASP}^{ch}$ (consisting of normal rules, choice rules and hard constraints), such that $ord(B \cup H_1 \cup C) \neq ord(B, \cup H_2 \cup C)$.

Proposition 5.17. *(proof on page 281)*

$$\mathcal{D}^1_1(ILP^{context}_{LOAS}) = \left\{ \langle B, H_1, H_2\rangle \;\middle|\; \begin{array}{c} B \cup H_1 \not\equiv^s B \cup H_2 \text{ or} \\ \exists C \in \mathcal{ASP}^{ch} \textit{ such that } ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C) \end{array} \right\}$$

Now that we have proven the distinguishability classes for each learning framework, we can strengthen the statement of Corollary 5.8 and more precisely state the relationship between the distinguishability classes of the frameworks. Apart from the case of ILP_b and ILP_{sm}, each of the subset relations in Corollary 5.8 are in fact strict subsets.

Theorem 5.18. *Consider the learning frameworks ILP_b, ILP_c, ILP_{sm}, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$.*

1. $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm}) \subset \mathcal{D}^1_1(ILP_{LAS}) \subset \mathcal{D}^1_1(ILP_{LOAS}) \subset \mathcal{D}^1_1(ILP^{context}_{LOAS})$
2. $\mathcal{D}^1_1(ILP_c) \subset \mathcal{D}^1_1(ILP_{LAS})$

Proof.

1. The fact that $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm})$ was shown in Proposition 5.11. By Corollary 5.8, $\mathcal{D}^1_1(ILP_{sm}) \subseteq \mathcal{D}^1_1(ILP_{LAS}) \subseteq \mathcal{D}^1_1(ILP_{LOAS}) \subseteq \mathcal{D}^1_1(ILP^{context}_{LOAS})$; hence, it remains to show that $\mathcal{D}^1_1(ILP_{sm}) \neq \mathcal{D}^1_1(ILP_{LAS}) \neq \mathcal{D}^1_1(ILP_{LOAS}) \neq \mathcal{D}^1_1(ILP^{context}_{LOAS})$
   - Consider the tuple $\langle B, H_1, H_2\rangle$, where $B = \emptyset$, $H_1 = \{\texttt{p.}\}$ and $H_2 = \{\texttt{1\{p,q\}1}\}$. $AS(B \cup H_1) \subset AS(B \cup H_2)$, hence $\langle B, H_1, H_2\rangle$ does not satisfy the condition, given in Table 5.1, necessary for it to be in $\mathcal{D}^1_1(ILP_{sm})$. It does, however, satisfy the condition for it to be in $\mathcal{D}^1_1(ILP_{LAS})$. Hence, $\mathcal{D}^1_1(ILP_{sm}) \neq \mathcal{D}^1_1(ILP_{LAS})$.
   - Consider the tuple $\langle B, H_1, H_2\rangle$, where $B = \{\texttt{1\{p,q\}1}\}$, $H_1 = \emptyset$ and $H_2 = \{\texttt{:\sim p.[1@1]}\}$. $AS(B \cup H_1) = AS(B \cup H_2)$ and $ord(B \cup H_1) \neq ord(B \cup H_2)$. Hence, by the conditions in Table 5.1, $\langle B, H_1, H_2\rangle$ is in $\mathcal{D}^1_1(ILP_{LOAS})$ but is not in $\mathcal{D}^1_1(ILP_{LAS})$. Therefore, $\mathcal{D}^1_1(ILP_{LAS}) \neq \mathcal{D}^1_1(ILP_{LOAS})$.
   - Consider the tuple $\langle B, H_1, H_2\rangle$, where $B = \emptyset$, $H_1 = \emptyset$ and $H_2 = \{\texttt{p:-q.}\}$. Also consider the program $P = \{\texttt{q.}\}$. $AS(B \cup H_1) = AS(B \cup H_2)$ and $ord(B \cup H_1) = ord(B \cup H_2)$, but $AS(B \cup H_1 \cup P) \neq AS(B \cup H_2 \cup P)$; this shows that $B \cup H_1 \not\equiv_s B \cup H_2$. Hence, by the conditions in Table 5.1, $\langle B, H_1, H_2\rangle$ is in $\mathcal{D}^1_1(ILP^{context}_{LOAS})$ but is not in $\mathcal{D}^1_1(ILP_{LOAS})$. Therefore, $\mathcal{D}^1_1(ILP_{LOAS}) \neq \mathcal{D}^1_1(ILP^{context}_{LOAS})$.
2. By Corollary 5.8, $\mathcal{D}^1_1(ILP_c) \subseteq \mathcal{D}^1_1(ILP_{LAS})$. Hence, it remains to show that $\mathcal{D}^1_1(ILP_c) \neq \mathcal{D}^1_1(ILP_{LAS})$. Consider the tuple $\langle B, H_1, H_2\rangle$, where $B = \{\texttt{p:- not p}\}$, $H_1 = \emptyset$ and $H_2 = \{\texttt{p.}\}$. $AS(B \cup H_1) = \emptyset$ and $AS(B \cup H_1) \neq AS(B \cup H_2)$. By the conditions in Table 5.1, $\langle B, H_1, H_2\rangle$ is in $\mathcal{D}^1_1(ILP_{LAS})$ but is not in $\mathcal{D}^1_1(ILP_c)$. Hence, $\mathcal{D}^1_1(ILP_c) \neq \mathcal{D}^1_1(ILP_{LAS})$.

□

5.2 The One-To-Many-Distinguishability Class of a Learning Framework

In practice, an ILP task has a search space of possible hypotheses, and it is important to know the cases in which one particular hypothesis can be distinguished from the rest. In what follows, we analyse the conditions under which a learning framework can distinguish a hypothesis from *a set* of other hypotheses. As mentioned at the beginning of the chapter, this corresponds to the new notion called the *one-to-many-distinguishability class* of a learning framework, which is a generalisation of the notion of the *one-to-one-distinguishability class* described in the previous section.

Definition 5.4. The *one-to-many-distinguishability class* of a learning framework $\mathcal{F}$ (denoted $\mathcal{D}^1_m(\mathcal{F})$) is the set of all tuples $\langle B, H, \{H_1, \ldots, H_n\}\rangle$ such that there is a task $T_{\mathcal{F}}$ that distinguishes H from each H_i with respect to B. Given two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, we say that $\mathcal{F}_1$ is at least as (resp. more) $\mathcal{D}^1_m$-general as (resp. than) $\mathcal{F}_2$ if $\mathcal{D}^1_m(\mathcal{F}_2) \subseteq \mathcal{D}^1_m(\mathcal{F}_1)$ (resp. $\mathcal{D}^1_m(F_2) \subset \mathcal{D}^1_m(F_1)$).

The one-to-many-distinguishability class tells us the circumstances in which a framework is general enough to distinguish some target hypothesis from a set of unwanted hypotheses. Note that, although the tuples in a one-to-many-distinguishability class that have a singleton set as third argument correspond to the tuples in a one-to-one-distinguishability class of that framework, it is not always the case that if $\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$ then $\mathcal{F}_1$ is also more $\mathcal{D}^1_1$-general than $\mathcal{F}_2$. For example, we will see that ILP_{sm} is more $\mathcal{D}^1_m$-general than ILP_b, but we have already shown in Proposition 5.11 that the ILP_b and ILP_{sm} are equally $\mathcal{D}^1_1$-general. Proposition 5.19 shows, however, that if $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$ then $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$.

Proposition 5.19. For any two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$ such that $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$, $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$ (i.e. $\mathcal{D}^1_m(\mathcal{F}_2) \subseteq \mathcal{D}^1_m(\mathcal{F}_1) \Rightarrow \mathcal{D}^1_1(\mathcal{F}_2) \subseteq \mathcal{D}^1_1(\mathcal{F}_1)$).

Proof. Assume that $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$ and let $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_2)$. To show that $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$, we must show that $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_1)$.

As $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_2)$, $\langle B, H_1, \{H_2\}\rangle \in \mathcal{D}^1_m(\mathcal{F}_2)$; hence, as $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$, $\langle B, H_1, \{H_2\}\rangle \in \mathcal{D}^1_m(\mathcal{F}_1)$; and hence, $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_1)$.

□

We have already seen that if there is a strong reduction from $\mathcal{F}_1$ to $\mathcal{F}_2$ then $\mathcal{F}_2$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_1$. Proposition 5.20 shows that a similar result holds for $\mathcal{D}^1_m$-generality. Similarly to $\mathcal{D}^1_1$-generality, however, a strong reduction from $\mathcal{F}_1$ to $\mathcal{F}_2$ does not imply that $\mathcal{F}_2$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_1$, even in the case that there is no strong reduction from $\mathcal{F}_2$ to $\mathcal{F}_1$.

Proposition 5.20. For any two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$: $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2 \Rightarrow \mathcal{D}^1_m(\mathcal{F}_1) \subseteq \mathcal{D}^1_m(\mathcal{F}_2)$.

Proof. Assume that $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$. Take any $\langle B, H, S\rangle \in \mathcal{D}^1_m(\mathcal{F}_1)$. There must be some task $T_{\mathcal{F}_1}$, with background knowledge B, such that $H \in ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1})$ and $S \cap ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1}) = \emptyset$. Hence, as $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$, there must be some $\mathcal{F}_2$ task $T_{\mathcal{F}_2}$, with background knowledge B, such that $H \in ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$ and $S \cap ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2}) = \emptyset$. So $\langle B, H, S\rangle \in \mathcal{D}^1_m(\mathcal{F}_2)$. Hence, $\mathcal{D}^1_m(\mathcal{F}_1) \subseteq \mathcal{D}^1_m(\mathcal{F}_2)$. □

Due to the strong reductions shown in Proposition 5.4, an ordering of the one-to-many-distinguishability classes of the frameworks emerges (shown in Corollary 5.21).

Corollary 5.21.

1. $\mathcal{D}^1_m(ILP_b) \subseteq \mathcal{D}^1_m(ILP_{sm}) \subseteq \mathcal{D}^1_m(ILP_{LAS}) \subseteq \mathcal{D}^1_m(ILP_{LOAS}) \subseteq \mathcal{D}^1_m(ILP^{context}_{LOAS})$
2. $\mathcal{D}^1_m(ILP_c) \subseteq \mathcal{D}^1_m(ILP_{LAS})$

This time, we will see that each of the $\subseteq$'s in Corollary 5.21 can be upgraded to a strict $\subset$. Rather than proving the one-to-many-distinguishability classes from scratch, we now present a useful result. For some frameworks, the one-to-one-distinguishability class of a learning framework can be used to construct the one-to-many-distinguishability class. This is the case when the framework has *closed one-to-many-distinguishability* (formalised by Definition 5.5). Proposition 5.22 and Corollary 5.23 show how the one-to-many-distinguishability class of a framework can be constructed using its one-to-one-distinguishability class if it has closed one-to-many-distinguishability.

Definition 5.5. Given any learning framework $\mathcal{F}$, the *closure* of the one-to-many-distinguishability class, written $\overline{\mathcal{D}^1_m(\mathcal{F})}$ is the set $\left\{\langle B, H, S_1 \cup \ldots \cup S_n\rangle \middle| \langle B, H, S_1\rangle, \ldots, \langle B, H, S_n\rangle \in \mathcal{D}^1_m(\mathcal{F})\right\}$. We say that $\mathcal{F}$ has *closed one-to-many-distinguishability* if and only if $\overline{\mathcal{D}^1_m(\mathcal{F})} = \mathcal{D}^1_m(\mathcal{F})$.

Proposition 5.22. *(proof on page 283)* For any framework $\mathcal{F}$:

$$\overline{\mathcal{D}^1_m(\mathcal{F})} = \left\{ \langle B, H, \{H_1, \ldots, H_n\}\rangle \middle| \begin{array}{c} \langle B, H, H_1\rangle \in \mathcal{D}^1_1(\mathcal{F}), \\ \ldots, \\ \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F}) \end{array} \right\}$$

Corollary 5.23. For any framework $\mathcal{F}$:

$$\mathcal{D}^1_m(\mathcal{F}) \subseteq \left\{ \langle B, H, \{H_1, \ldots, H_n\}\rangle \middle| \begin{array}{c} \langle B, H, H_1\rangle \in \mathcal{D}^1_1(\mathcal{F}), \\ \ldots, \\ \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F}) \end{array} \right\}$$

The equality holds if and only if $\mathcal{F}$ has closed one-to-many-distinguishability.

Note that not all learning frameworks have closed one-to-many-distinguishability; for instance, Example 5.24 shows that brave induction does not. We will show that induction of stable models, on the other hand, does have closed one-to-many-distinguishability.

Example 5.24. *ILP_b does not have closed one-to-many-distinguishability. We can see this by reconsidering the programs $B = \emptyset$, $H = \{\texttt{1\{heads, tails\}1.}\}$, $H_1 = \{\texttt{heads.}\}$ and $H_2 = \{\texttt{tails.}\}$. $\langle B, H, \{H_1\}\rangle \in \mathcal{D}^1_m(ILP_b)$ ($\langle B, \langle\{\texttt{tails}\}, \emptyset\rangle\rangle$ distinguishes H from H_1 wrt the background knowledge*

B). Similarly $\langle B, H, \{H_2\}\rangle \in \mathcal{D}^1_m(ILP_b)$ ($\langle B, \langle \{\texttt{heads}\}, \emptyset\rangle\rangle$ distinguishes H from H_2 wrt the background knowledge B). If ILP_b had closed one-to-many-distinguishability then $\langle B, H, \{H_1, H_2\}\rangle$ would be in $\mathcal{D}^1_m(ILP_b)$; hence, to show that ILP_b does not have closed one-to-many-distinguishability it is sufficient to show that $\langle B, H, \{H_1, H_2\}\rangle \notin \mathcal{D}^1_m(ILP_b)$. Hence it remains to show that there is no task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H \in ILP_b(T_b)$ and $\{H_1, H_2\} \cap ILP_b(T_b) = \emptyset$.

Assume for contradiction that there is such a task T_b. As $H \in ILP_b(T_b)$ and $AS(B \cup H) = \{\{\texttt{heads}\}, \{\texttt{tails}\}\}$, $E^+ \subset \{\texttt{heads}, \texttt{tails}\}$ and $E^- \subset \{\texttt{heads}, \texttt{tails}\}$ (neither can be equal to $\{\texttt{heads}, \texttt{tails}\}$ or H would not be a solution).

Case 1: $E^+ = \emptyset$

Case a: $E^- = \emptyset$

Then H_1 and H_2 would be inductive solutions. This is a contradiction as $\{H_1, H_2\} \cap ILP_b(T_b) = \emptyset$.

Case b: $E^- = \{\texttt{heads}\}$

Then H_2 would be an inductive solution of T_b. Contradiction.

Case c: $E^- = \{\texttt{tails}\}$

Then H_1 would be an inductive solution of T_b. Contradiction.

Case 2: $E^+ = \{\texttt{heads}\}$

$\texttt{heads} \notin E^-$ *as otherwise the task would have no solutions (and we know that H is a solution). In this case H_1 would be an inductive solution (regardless of what else is in E^-). Contradiction.*

Case 3: $E^+ = \{\texttt{tails}\}$

Similarly to above case, $\texttt{tails} \notin E^-$ *as otherwise the task would have no solutions. In this case H_2 would be an inductive solution (regardless of what else is in E^-). Contradiction.*

Hence, there is no such task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H \in ILP_b(T_b)$ and $\{H_1, H_2\} \cap ILP_b(T_b) = \emptyset$. ILP_b does not have closed one-to-many-distinguishability.

In contrast to ILP_b, ILP_{sm} (which we will see does have closed one-to-many-distinguishability), can distinguish H from H_1 and H_2 with the task $\langle B, \langle \{\langle \{\texttt{heads}\}, \emptyset\rangle, \langle \{\texttt{tails}\}, \emptyset\rangle\}\rangle\rangle$. Note that this is a combination of the two brave tasks which distinguish H from H_1 and from H_2. We will show that the ability to combine tasks in this way is a sufficient condition for a framework to have closed one-to-many-distinguishability. Proposition 5.25 shows the one-to-many-distinguishability class of ILP_b.

Proposition 5.25. $\mathcal{D}^1_m(ILP_b) = \{\langle B, H, \{h_1, \ldots, h_m\}\rangle \mid AS(B \cup H) \not\subseteq AS(B \cup h_1) \cup \ldots \cup AS(B \cup h_m)\}$

Proof.

1. Let $B, H, h_1, \ldots, h_m$ be ASP programs such that $AS(B \cup H) \not\subseteq AS(B \cup h_1) \cup \ldots \cup AS(B \cup h_m)$. This implies that there is an interpretation A that is an answer set of $B \cup H$ but not an answer set of any of the programs $B \cup h_1, \ldots, B \cup h_m$. Let L be the set of atoms which occur in at least one answer set of at least one of the programs $B \cup H, B \cup h_1, \ldots B \cup h_m$; then $B \cup H$ has an answer set that extends $\langle A, L \backslash A\rangle$, but none of $B \cup h_1, \ldots B \cup h_m$ do. So the task $\langle B, \langle A, L\backslash A\rangle\rangle$ distinguishes H from h_1 to h_m. Hence, $\langle B, H, \{h_1, \ldots, h_m\}\rangle \in \mathcal{D}^1_m(ILP_b)$.
2. Assume $\langle B, H, \{h_1, \ldots, h_m\}\rangle \in \mathcal{D}^1_m(ILP_b)$. Then there is an ILP_b task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $B \cup H$ has an answer set extending $\langle E^+, E^-\rangle$ and none of $B \cup h_1, \ldots, B \cup h_m$ do. Hence, there must be at least one answer set of $B \cup H$, which is not an answer set of any of $B \cup h_1, \ldots, B \cup h_m$. Therefore $AS(B \cup H) \not\subseteq AS(B \cup h_1) \cup \ldots \cup AS(B \cup h_m)$.

□

For a framework $\mathcal{F}$ to have closed one-to-many-distinguishability it is sufficient (but not necessary) that for every two $\mathcal{F}$ tasks, there is a third $\mathcal{F}$ task whose solutions are exactly those hypotheses which are solutions to both of the original two tasks. This is formalised and proved in Lemma 5.26. This condition is not necessary in general, but it holds for the frameworks considered in this chapter that have closed one-to-many-distinguishability.

Lemma 5.26. For any learning framework $\mathcal{F}$ to have closed one-to-many-distinguishability, it is sufficient that for every pair of learning tasks $T^1_{\mathcal{F}} = \langle B, E^1_{\mathcal{F}}\rangle$ and $T^2_{\mathcal{F}} = \langle B, E^2_{\mathcal{F}}\rangle$ it is possible to construct a new learning task $T^3_{\mathcal{F}} = \langle B, E^3_{\mathcal{F}}\rangle$ such that $ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = ILP_{\mathcal{F}}(T^1_{\mathcal{F}}) \cap ILP_{\mathcal{F}}(T^2_{\mathcal{F}})$.

Proof. Assume that for every pair of learning tasks $T^1_{\mathcal{F}} = \langle B, E^1_{\mathcal{F}}\rangle$ and $T^2_{\mathcal{F}} = \langle B, E^2_{\mathcal{F}}\rangle$ it is possible to construct a new learning task $T^3_{\mathcal{F}} = \langle B, E^3_{\mathcal{F}}\rangle$ such that $ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = ILP_{\mathcal{F}}(T^1_{\mathcal{F}}) \cap ILP_{\mathcal{F}}(T^2_{\mathcal{F}})$. Let $\langle B, H, S_1\rangle, \ldots, \langle B, H, S_n\rangle \in \mathcal{D}^1_m(\mathcal{F})$. To prove that $\mathcal{F}$ has closed one-to-many-distinguishability, we must show that $\langle B, H, S_1 \cup \ldots \cup S_n\rangle \in \mathcal{D}^1_m(\mathcal{F})$. We prove this by showing (by mathematical induction) that for each $k \in [1..n]$, $\langle B, H, S_1 \cup \ldots \cup S_k\rangle \in \mathcal{D}^1_m(\mathcal{F})$.

Base Case: $k = 1$. $\langle B, H, S_1\rangle \in \mathcal{D}^1_m(\mathcal{F})$ by the initial assumptions.

Inductive Hypothesis: Assume that for some $1 \leq k < n$, $\langle B, H, S_1 \cup \ldots \cup S_k\rangle \in \mathcal{D}^1_m(\mathcal{F})$

Inductive Step: We must show that $\langle B, H, S_1 \cup \ldots \cup S_{k+1}\rangle \in \mathcal{D}^1_m(\mathcal{F})$.

As $\langle B, H, S_1 \cup \ldots \cup S_k\rangle \in \mathcal{D}^1_m(\mathcal{F})$ (by the inductive hypothesis), there must be a learning task $T^1_{\mathcal{F}}$ such that $H \in ILP_{\mathcal{F}}(T^1_{\mathcal{F}})$ and $(S_1 \cup \ldots \cup S_k) \cap ILP_{\mathcal{F}}(T^1_{\mathcal{F}}) = \emptyset$. As $\langle B, H, S_{k+1}\rangle \in \mathcal{D}^1_m(\mathcal{F})$, there must also be a learning task $T^2_{\mathcal{F}}$ such that $H \in ILP_{\mathcal{F}}(T^2_{\mathcal{F}})$ and $S_{k+1} \cap ILP_{\mathcal{F}}(T^2_{\mathcal{F}}) = \emptyset$. By our initial assumption, there is a learning task $T^3_{\mathcal{F}} = \langle B, E^3_{\mathcal{F}}\rangle$ such that $ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = ILP_{\mathcal{F}}(T^1_{\mathcal{F}}) \cap ILP_{\mathcal{F}}(T^2_{\mathcal{F}})$. So, $H \in ILP_{\mathcal{F}}(T^3_{\mathcal{F}})$, $(S_1 \cup \ldots \cup S_k) \cap ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = \emptyset$ and $S_{k+1} \cap ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = \emptyset$. Therefore, $H \in ILP_{\mathcal{F}}(T^3_{\mathcal{F}})$ and $(S_1 \cup \ldots \cup S_{k+1}) \cap ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = \emptyset$. Hence, $\langle B, H, S_1 \cup \ldots \cup S_{k+1}\rangle \in \mathcal{D}^1_m(\mathcal{F})$.

□

Proposition 5.27. *(proof on page 283)* ILP_c, ILP_{sm}, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$ all have closed one-to-many-distinguishability.

Theorem 5.28. *Given two frameworks* $\mathcal{F}_1$ *and* $\mathcal{F}_2$, $\overline{\mathcal{D}^1_m(\mathcal{F}_1)} \subseteq \overline{\mathcal{D}^1_m(\mathcal{F}_2)}$ *if and only if* $\mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2)$.

Proof.

$$\overline{\mathcal{D}^1_m(\mathcal{F}_1)} \subseteq \overline{\mathcal{D}^1_m(\mathcal{F}_2)}$$
$$\Leftrightarrow \left\{ \langle B, H, \{H_1, \ldots, H_n\}\rangle \left| \begin{array}{c} \langle B, H, H_1\rangle \in \mathcal{D}^1_1(\mathcal{F}_1) \\ \ldots \\ \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F}_1) \end{array} \right.\right\}$$
$$\subseteq \left\{ \langle B, H, \{H_1, \ldots, H_n\}\rangle \left| \begin{array}{c} \langle B, H, H_1\rangle \in \mathcal{D}^1_1(\mathcal{F}_2) \\ \ldots \\ \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F}_2) \end{array} \right.\right\} \text{ (by Prop. 5.22)}$$
$$\Leftrightarrow \mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2).$$

□

Corollary 5.29. Given two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$ with closed one-to-many-distinguishability: $\mathcal{D}^1_m(\mathcal{F}_1) \subset \mathcal{D}^1_m(\mathcal{F}_2)$ if and only if $\mathcal{D}^1_1(\mathcal{F}_1) \subset \mathcal{D}^1_1(\mathcal{F}_2)$.

Theorem 5.30. *Consider the learning frameworks* ILP_b, ILP_c, ILP_{sm}, ILP_{LAS}, ILP_{LOAS} *and* $ILP^{context}_{LOAS}$.

1. $\mathcal{D}^1_m(ILP_b) \subset \mathcal{D}^1_m(ILP_{sm}) \subset \mathcal{D}^1_m(ILP_{LAS}) \subset \mathcal{D}^1_m(ILP_{LOAS}) \subset \mathcal{D}^1_m(ILP^{context}_{LOAS})$
2. $\mathcal{D}^1_m(ILP_c) \subset \mathcal{D}^1_m(ILP_{LAS})$

Proof. Firstly, as shown in Example 5.24, $\mathcal{D}^1_m(ILP_b)$ is a strict subset of $\overline{\mathcal{D}^1_m(ILP_b)}$. Hence, by Theorem 5.28, $\mathcal{D}^1_m(ILP_b) \subset \overline{\mathcal{D}^1_m(ILP_{sm})}$; and hence as ILP_{sm} has closed one-to-many-distinguishability, $\mathcal{D}^1_m(ILP_b) \subset \mathcal{D}^1_m(ILP_{sm})$. The other results all follow from Corollary 5.29 and Proposition 5.27. □

Even if two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$ both have closed one-to-many-distinguishability, it might not be the case that their combination has closed one-to-many-distinguishability. Example 5.31 shows, for example, that this is not the case for ILP_{sm} and ILP_c. We first define what we mean by the combination framework constructed from two given frameworks.

Definition 5.6. Given two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, a task of the combination framework $comb(\mathcal{F}_1, \mathcal{F}_2)$ is either of the form $\langle B, \langle 1, E_1\rangle\rangle$, where $\langle B, E_1\rangle$ is an $\mathcal{F}_1$ task, or of the form $\langle B, \langle 2, E_2\rangle\rangle$, where $\langle B, E_2\rangle$ is an $\mathcal{F}_2$ task.

Given any $comb(\mathcal{F}_1, \mathcal{F}_2)$ task $T = \langle B, \langle x, E\rangle\rangle$: $ILP_{comb(\mathcal{F}_1,\mathcal{F}_2)}(T) = \left\{ \begin{array}{l} ILP_{\mathcal{F}_1}(\langle B, E\rangle) \text{ if } x = 1 \\ ILP_{\mathcal{F}_2}(\langle B, E\rangle) \text{ if } x = 2 \end{array} \right.$

Example 5.31. *Consider the frameworks ILP_{sm} and ILP_c (both of which have closed one-to-many-distinguishability). Also consider the programs $B = \emptyset, H = \{\texttt{0\{p\}1.}\}, H_1 = \emptyset, H_2 = \{\texttt{0\{p,q\}1.}\}$. $\langle B, H, H_1\rangle \in \mathcal{D}^1_1(ILP_{sm})$ (using the task $\langle B, \langle\{\langle\{\texttt{p}\}, \emptyset\rangle\}\rangle\rangle$), and $\langle B, H, H_2\rangle \in \mathcal{D}^1_1(ILP_c)$ (using the task $\langle B, \langle\emptyset, \{\texttt{q}\}\rangle\rangle$). This shows that both $\langle B, H, H_1\rangle$ and $\langle B, H, H_2\rangle$ are in $\mathcal{D}^1_1(ILP_{sm}) \cup \mathcal{D}^1_1(ILP_c)$. Hence by Definition 5.6 they must be in $\mathcal{D}^1_1(comb(ILP_{sm}, ILP_c))$. But using the distinguishability conditions proven in the previous section, it can be seen that neither framework can distinguish H from both H_1 and H_2. Therefore, $\langle B, H, \{H_1, H_2\}\rangle \notin \mathcal{D}^1_m(comb(ILP_{sm}, ILP_c))$. This also means that $\mathcal{D}^1_m(ILP_{sm}) \cup \mathcal{D}^1_m(ILP_c)$ is a strict subset of $\mathcal{D}^1_m(ILP_{LAS})$. This is because $\mathcal{D}^1_m(ILP_{LAS})$ contains both $\langle B, H, \{H_1\}\rangle$ and $\langle B, H, \{H_2\}\rangle$ (as it contains both $\mathcal{D}^1_m(ILP_{sm})$ and $\mathcal{D}^1_m(ILP_c)$) and has closed one-to-many-distinguishability, so must also contain $\langle B, H, \{H_1, H_2\}\rangle$.*

5.3 The Many-To-Many-Distinguishability Class of a Learning Framework

So far, we have considered two main classes to define how general a learning framework is. Firstly, we discussed the one-to-one-distinguishability class, which is made up of tuples $\langle B, H, H'\rangle$ such that the framework can distinguish H from H' with respect to B. We showed that this has limitations and cannot separate ILP_b and ILP_{sm} even though ILP_b is clearly a special case of ILP_{sm}. This motivated upgrading the notion of a one-to-one-distinguishability class, changing the third element of each tuple from a single hypothesis to a set of hypotheses to give the notion of a one-to-many-distinguishability class.

This naturally leads to the question of whether it is possible to upgrade generality classes by allowing the second element of the tuple to also be a set of hypotheses. Each tuple would then be of the form $\langle B, S_1, S_2\rangle$, where B is a background knowledge, and S_1 and S_2 are sets of hypotheses. For each tuple in this new class, a framework would be required to have at least one task T with the background knowledge B such that every hypothesis in S_1 is an inductive solution of T, and no hypothesis in S_2 is an inductive solution of T. Definition 5.7 formalises this *many-to-many-distinguishability class.*

Definition 5.7. The *many-to-many-distinguishability class* of a learning framework $\mathcal{F}$ (denoted $\mathcal{D}^m_m(\mathcal{F})$) is the set of all tuples $\langle B, S_1, S_2\rangle$, where B is a program and S_1 and S_2 are sets of hypotheses for which there is a task $T_{\mathcal{F}}$, with background knowledge B, such that $S_1 \subseteq ILP_{\mathcal{F}}(T_{\mathcal{F}})$ and $S_2 \cap ILP_{\mathcal{F}}(T_{\mathcal{F}}) = \emptyset$. Given two frameworks, $\mathcal{F}_1$ and $\mathcal{F}_2$, we say that $\mathcal{F}_1$ is at least as (resp. more) $\mathcal{D}^m_m$-general than $\mathcal{F}_2$ if and only if $\mathcal{D}^m_m(\mathcal{F}_2) \subseteq \mathcal{D}^m_m(\mathcal{F}_1)$ (resp. $\mathcal{D}^m_m(\mathcal{F}_2) \subset \mathcal{D}^m_m(\mathcal{F}_1)$).

We have already seen that for any two frameworks, $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2 \Rightarrow \mathcal{D}^1_m(\mathcal{F}_1) \subseteq \mathcal{D}^1_m(\mathcal{F}_2) \Rightarrow \mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2)$. We have also seen that for $\mathcal{D}^1_1$-generality and $\mathcal{D}^1_m$-generality, even if there is no corresponding strong reduction from $\mathcal{F}_2$ to $\mathcal{F}_1$ these subset relations are not necessarily strict. Proposition 5.32 and Corollary 5.33 show that $\mathcal{D}^m_m$-generality is equivalent to strong reductions.

Proposition 5.32. For any two learning frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2 \Leftrightarrow \mathcal{D}^m_m(\mathcal{F}_1) \subseteq \mathcal{D}^m_m(\mathcal{F}_2)$

Proof.

1. Assume that $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$. Let $\langle B, S_1, S_2\rangle$ be an arbitrary element of $\mathcal{D}^m_m(\mathcal{F}_1)$. By definition of $\mathcal{D}^m_m(\mathcal{F}_1)$, there is a task $T_{\mathcal{F}_1}$ with background knowledge B such that $S_1 \subseteq ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1})$ and $S_2 \cap ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1}) = \emptyset$. Hence, as $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$, there is an $\mathcal{F}_2$ task $T_{\mathcal{F}_2}$ with background knowledge B such that $S_1 \subseteq ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$ and $S_2 \cap ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2}) = \emptyset$. Hence $\langle B, S_1, S_2\rangle \in \mathcal{D}^m_m(\mathcal{F}_2)$.

2. Assume that $\mathcal{D}^m_m(\mathcal{F}_1) \subseteq \mathcal{D}^m_m(\mathcal{F}_2)$. Let $T_{\mathcal{F}_1}$ be an arbitrary $\mathcal{F}_1$ task. We must show that there is a $\mathcal{F}_2$ task with the same background knowledge and the same inductive solutions. Let B be the background knowledge of $T_{\mathcal{F}_1}$, $S_1 = ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1})$ and S_2 be the (possibly infinite) set of ASP programs which are not in S_1. $\langle B, S_1, S_2\rangle \in \mathcal{D}^m_m(\mathcal{F}_1)$; and hence, $\langle B, S_1, S_2\rangle \in \mathcal{D}^m_m(\mathcal{F}_2)$. Therefore, there must be at least one task $T_{\mathcal{F}_2}$ with the background knowledge B such that $ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2}) = S_1$. Hence, $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$.

□

Corollary 5.33. For any two learning frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1$ is more $\mathcal{D}^m_m$-general than $\mathcal{F}_2$ if and only if $\mathcal{F}_2 \rightarrow_{sr} \mathcal{F}_1$ and $\mathcal{F}_1 \not\rightarrow_{sr} \mathcal{F}_2$.

Proposition 5.34. For any two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$: $\mathcal{D}^m_m(\mathcal{F}_1) \subseteq \mathcal{D}^m_m(\mathcal{F}_2) \Rightarrow \mathcal{D}^1_m(\mathcal{F}_1) \subseteq \mathcal{D}^1_m(\mathcal{F}_2)$

Proof. Assume that $\mathcal{D}^m_m(\mathcal{F}_1) \subseteq \mathcal{D}^m_m(\mathcal{F}_2)$ and let $\langle B, H, S\rangle \in \mathcal{D}^1_m(\mathcal{F}_1)$. Then $\langle B, \{H\}, S\rangle \in \mathcal{D}^m_m(\mathcal{F}_1)$, and so $\langle B, \{H\}, S\rangle \in \mathcal{D}^m_m(\mathcal{F}_2)$. Hence, $\langle B, H, S\rangle \in \mathcal{D}^1_m(\mathcal{F}_2)$. □

Theorem 5.35 shows that one framework being more $\mathcal{D}^1_m$-general than another implies that it is also more $\mathcal{D}^m_m$-general if there is a strong reduction from the second framework to the first.

Theorem 5.35. *For any two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, if $\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$ and $\mathcal{F}_2 \rightarrow_{sr} \mathcal{F}_1$ then $\mathcal{F}_1$ is more $\mathcal{D}^m_m$-general than $\mathcal{F}_2$.*

Proof. Assume that $\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$ and that $\mathcal{F}_2 \rightarrow_{sr} \mathcal{F}_1$. By Proposition 5.32, $\mathcal{F}_1$ is at least as $\mathcal{D}^m_m$-general as $\mathcal{F}_2$. It remains to show that $\mathcal{F}_2$ is not at least as $\mathcal{D}^m_m$-general as $\mathcal{F}_1$. Assume for contradiction that $\mathcal{F}_2$ is at least as $\mathcal{D}^m_m$-general as $\mathcal{F}_1$. Then by Proposition 5.34, $\mathcal{F}_2$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_1$, contradicting the fact that $\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$. □

Corollary 5.36. Consider the learning frameworks ILP_b, ILP_c, ILP_{sm}, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$.

1. $\mathcal{D}^m_m(ILP_b) \subset \mathcal{D}^m_m(ILP_{sm}) \subset \mathcal{D}^m_m(ILP_{LAS}) \subset \mathcal{D}^m_m(ILP_{LOAS}) \subset \mathcal{D}^m_m(ILP^{context}_{LOAS})$

2. $\mathcal{D}^m_m(ILP_c) \subset \mathcal{D}^m_m(ILP_{LAS})$

Proof. Each result follows directly from Theorem 5.30, Theorem 5.35 and Proposition 5.4. □

Note that although for each pair of frameworks discussed in this chapter, one being more $\mathcal{D}^1_m$-general than another implies that it is also more $\mathcal{D}^m_m$-general, this result does not hold in general. Example 5.37 shows such a pair of frameworks.

Example 5.37. *Consider a new learning framework ILP_d that takes as examples a pair of sets of atoms E^+ and E^- such that a hypothesis H is an inductive solution of a task if $B \cup H$ has exactly one answer set and this answer set contains all of the E^+'s and none of the E^-'s. The one-to-one-distinguishability class $\mathcal{D}^1_1(ILP_d) \subseteq \mathcal{D}^1_1(ILP_c)$. This can be seen as follows: assume that $\langle B, H, H' \rangle \in \mathcal{D}^1_1(ILP_d)$. Then there is a task $T_d = \langle B, \langle E^+, E^- \rangle \rangle$ such that $H \in ILP_d(T_d)$ but $H' \notin ILP_d(T_d)$.*

Case 1: *$AS(B \cup H') = \emptyset$*

Let $T_c = \langle B, \langle E^+, E^- \rangle \rangle$. As $B \cup H$ has exactly one answer set, and this answer set covers the examples, $H \in ILP_c(T_c)$. As $AS(B \cup H') = \emptyset$, $H' \notin ILP_c(T_c)$. Hence, $\langle B, H, H' \rangle \in \mathcal{D}^1_1(ILP_c)$.

Case 2: *$B \cup H'$ has exactly one answer set, and this answer set does not cover the examples.*

Let $T_c = \langle B, \langle E^+, E^- \rangle \rangle$. As $B \cup H$ has exactly one answer set, and this answer set covers the examples, $H \in ILP_c(T_c)$. As $B \cup H'$ has an answer set that does not cover the examples, $H' \notin ILP_c(T_c)$. Hence, $\langle B, H, H' \rangle \in \mathcal{D}^1_1(ILP_c)$.

Case 3: *$B \cup H'$ has multiple answer sets.*

There must be at least one answer set A^* of $B \cup H'$ that is not an answer set of $B \cup H$ (as $B \cup H$ only has one answer set). There must either be an atom $\mathtt{a} \in A^*$ that is not in the unique answer set of $B \cup H$, or an atom $\mathtt{a}$ that is not in A^*, but is in the unique answer set of $B \cup H$. In the first case, let $E^+_c = \emptyset$ and $E^-_c = \{\mathtt{a}\}$. In the second case, let $E^+_c = \{\mathtt{a}\}$ and $E^-_c = \emptyset$. Then let $T_c = \langle B, \langle E^+_c, E^-_c \rangle \rangle$. $H \in ILP_c(T_c)$ as the only answer set of $B \cup H$ covers the examples, whereas $H' \notin ILP_c(T_c)$ as $B \cup H'$ has at least one answer set that does not cover the examples. Hence, $\langle B, H, H' \rangle \in \mathcal{D}^1_1(ILP_c)$.

In fact, $\mathcal{D}^1_1(ILP_d)$ is a strict subset of $\mathcal{D}^1_1(ILP_c)$ as $\mathcal{D}^1_1(ILP_d)$ has no elements $\langle B, H, H' \rangle$ where $B \cup H$ has multiple answer sets. As ILP_c is closed under one-to-many-distinguishability, and all one-to-many-distinguishability classes are subsets of their own closure, this means that ILP_c is more $\mathcal{D}^1_m$-general than ILP_d (by Theorem 5.28).

ILP_c is not, however, more $\mathcal{D}^m_m$-general than ILP_d. Take, for instance, the tuple $t = \langle \emptyset, \{\{\mathtt{heads.}\}, \{\mathtt{tails.}\}\}, \{\{\mathtt{1\{heads, tails\}1.}\}\} \rangle$. The empty set of examples are sufficient for ILP_d to distinguish both hypotheses containing facts from the choice rule (as the choice rule has multiple answer sets). However, there is no ILP_c task such that both facts are solutions, but the choice rule is not. Hence, $t \in \mathcal{D}^m_m(ILP_d)$ but $t \notin \mathcal{D}^m_m(ILP_c)$; and so, ILP_c is not at least as $\mathcal{D}^m_m$-general as ILP_d. In fact, as ILP_d is not as $\mathcal{D}^m_m$-general as ILP_c either, the two have incomparable $\mathcal{D}^m_m$-generalities.

Example 5.37 shows that $\mathcal{D}^m_m$-generality may not be able to compare two frameworks even when there is a clear $\mathcal{D}^1_m$-generality relation between the two. In the next section, we discuss relationships between, and the relative merits of using, each measure of generality.

5.4 Discussion

Property	Consequences of property
$\mathcal{F}_1$ and $\mathcal{F}_2$ have equal $\mathcal{D}^1_m$-generality	$\mathcal{F}_1$ and $\mathcal{F}_2$ have equal $\mathcal{D}^1_1$-generality
$\mathcal{F}_1$ and $\mathcal{F}_2$ have equal $\mathcal{D}^m_m$-generality	1) $\mathcal{F}_1$ and $\mathcal{F}_2$ have equal $\mathcal{D}^1_1$-generality 2) $\mathcal{F}_1$ and $\mathcal{F}_2$ have equal $\mathcal{D}^1_m$-generality
$\mathcal{F}_1$ is more $\mathcal{D}^1_1$-general than $\mathcal{F}_2$	Either $\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$ or $\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^1_m$-generality
$\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$	1) Either $\mathcal{F}_1$ is more $\mathcal{D}^m_m$-general than $\mathcal{F}_2$ or $\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^m_m$-generality 2) $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$
$\mathcal{F}_1$ is more $\mathcal{D}^m_m$-general than $\mathcal{F}_2$	1) $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$ 2) $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$
$\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$	$\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$
$\mathcal{F}_1$ is at least as $\mathcal{D}^m_m$-general as $\mathcal{F}_2$	1) $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$ 2) $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$
$\mathcal{F}_1$ and $\mathcal{F}_2$ have different $\mathcal{D}^1_1$-generality	1) $\mathcal{F}_1$ and $\mathcal{F}_2$ have different $\mathcal{D}^1_m$-generality 2) $\mathcal{F}_1$ and $\mathcal{F}_2$ have different $\mathcal{D}^m_m$-generality
$\mathcal{F}_1$ and $\mathcal{F}_2$ have different $\mathcal{D}^1_m$-generality	$\mathcal{F}_1$ and $\mathcal{F}_2$ have different $\mathcal{D}^m_m$-generality
$\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^1_1$-generality	1) $\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^1_m$-generality 2) $\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^m_m$-generality
$\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^1_m$-generality	$\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^m_m$-generality

Table 5.2: A summary of the relationships between the different measures of generality.

Table 5.2 summarises the relationships between the different measures of generality. It shows that equal distinguishability is weaker than equal one-to-many-distinguishability, which is weaker than equal many-to-many-distinguishability. This can be seen from the first section of the table, as equal many-to-many-distinguishability implies equal one-to-many-distinguishability, which implies equal distinguishability, but the converse implications do not hold in general. On the other hand different distinguishability is stronger than different one-to-many-distinguishability, which is stronger than different many-to-many-distinguishability. This means that many-to-many-distinguishability (resp. one-to-many-distinguishability) will be able to "separate" frameworks that one-to-many-distinguishability (resp. distinguishability) can not; but, there are more frameworks that are incomparable under many-to-many-distinguishability (resp. one-to-many-distinguishability) than one-to-many-distinguishability (resp. distinguishability).

The different notions of generalities will never be inconsistent, in the sense that one will never say that $\mathcal{F}_1$ is more general than $\mathcal{F}_2$, while the other says that $\mathcal{F}_2$ is more general than $\mathcal{F}_1$. It is useful, however, to explain the tasks that the different measures of generality correspond to.

1. One-to-one-distinguishability describes how general a framework is at distinguishing one hypothesis from another.
2. One-to-many-distinguishability describes how general a framework is at the task of identifying one target hypothesis within a space of unwanted hypotheses.
3. Many-to-many-distinguishability describes how general a framework is for the task of identifying a set of target hypotheses – for any background knowledge B and set of hypotheses S, there is a task $T_{\mathcal{F}}$ with background knowledge B such that $ILP_{\mathcal{F}}(T_{\mathcal{F}}) = S$ if and only if $\langle B, S, \bar{S} \rangle \in \mathcal{D}^m_m(\mathcal{F})$, where $\bar{S}$ is the (possibly infinite) set of hypotheses which are not in S.

In practice, as ILP usually addresses the task of finding a single target hypothesis from a space of other hypotheses, one-to-many-distinguishability is likely to be the most useful measure; however, distinguishability classes are useful for finding the one-to-many-distinguishability classes of frameworks, and many-to-many-distinguishability is interesting as a theoretical property.

More General Learning Frameworks

We have shown in this section that $ILP^{context}_{LOAS}$ is more general (under every measure) than any of the other tasks presented for learning under the answer set semantics. The obvious question is whether it is possible to go further and define more general learning tasks.

The most $\mathcal{D}^1_1$-general learning task possible would be able to distinguish between any two different ASP programs H_1 and H_2 with respect to any background knowledge B. This would require the learning task to distinguish between programs which are strongly equivalent, such as {`p.  q:-p.`} and {`p:-q.  q.`}. We would argue that this level of one-to-one-distinguishability is unnecessary as in ILP we aim to learn programs whose output explains the examples. As two strongly equivalent programs will always have the same output, even when combined with additional programs providing "context", we can not see any reason for going further under $\mathcal{D}^1_1$-generality. As $ILP^{context}_{LOAS}$ has closed one-to-many-distinguishability, the same argument can be made for $\mathcal{D}^1_m$-generality.

One outstanding question is whether it is worth going any further under $\mathcal{D}^m_m$-generality. Note that it is possible to define the notion of the closure of many-to-many-distinguishability classes; however, none of the frameworks considered in this chapter have closed many-to-many-distinguishability. It is unclear whether having closed many-to-many-distinguishability is a desirable property for a framework. Closed one-to-many-distinguishability means that a framework can distinguish a target hypothesis H from any set of hypotheses S such that it can distinguish H from each element of S: this means that the sets of examples that distinguish H from each element of S can be combined to form a single set of examples, ruling out each element of S. For a framework to have closed many-to-many-distinguishability, however, given two (or more) target hypotheses H_1, H_2 that can be distinguished from an undesirable hypothesis H_3, it would need to be able to find a task which distinguishes both H_1 and H_2 from H_3. For example, as both $\langle\emptyset, \{\texttt{heads}\}, \{\texttt{1\{heads, tails\}1.}\}\rangle$ and $\langle\emptyset, \{\texttt{tails}\}, \{\texttt{1\{heads, tails\}1.}\}\rangle$ are in $\mathcal{D}_1^1(ILP_{LAS})$, for ILP_{LAS} to have closed many-to-many-distinguishability it would need to be able to find a task with an empty background knowledge that distinguishes both $\{\texttt{heads.}\}$ and $\{\texttt{tails.}\}$ from $\{\texttt{1\{heads, tails\}1.}\}$. It is difficult to imagine a scenario, however, where we should learn either the hypothesis that a coin is always `heads` or always `tails`, when the choice rule is not a desirable hypothesis.

5.5 Related Work

As discussed at the beginning of this chapter, the generality of a learning framework has been investigated before. In [DR97], generality was defined in terms of reductions – one framework $\mathcal{F}_1$ was said to be more general than another framework $\mathcal{F}_2$ if and only if $\mathcal{F}_2 \rightarrow_r \mathcal{F}_1$ and $\mathcal{F}_1 \not\rightarrow_r \mathcal{F}_2$. We showed in Section 5.3 that our notion of many-to-many-distinguishability coincides with a similar notion of strong reductions. The difference with strong reductions, as compared to the reductions in [DR97], is that strong reductions do not allow the background knowledge to be modified as part of the reduction. We showed in Example 5.3 that ILP_b reduces to ILP_c, but ILP_b does not strongly reduce to ILP_c. This is because any reduction from ILP_b to ILP_c must encode the examples in the background knowledge, which we would argue abuses the purpose of the background knowledge.

Aside from the differences between strong reductions and reductions, we have discussed that one-to-many-distinguishability is more relevant when comparing the generalities of frameworks with respect to the task of finding a single hypothesis within a space of hypotheses. The reductions of [DR97] are closer to the notion of many-to-many-distinguishability, because they compare the set of solutions.

One key advantage to using our three notions of generality, rather than strong reductions or reductions, is for comparing the relative generalities of frameworks that do not strongly reduce to one another. For instance, we have seen that ILP_b and ILP_c are incomparable under $\mathcal{D}_1^1$-generality, but we can still reason that ILP_b is never $\mathcal{D}_1^1$-general enough to distinguish a hypothesis containing a constraint from the same hypothesis without the constraint. On the other hand, ILP_c may be $\mathcal{D}_1^1$-general enough to do so (for example, ILP_c can distinguish $\{\texttt{:- p.}\}$ from $\emptyset$ with respect to $\{\texttt{0\{p\}1.}\}$, with the task $\langle\{\texttt{0\{p\}1.}\}, \langle\emptyset, \{\texttt{p}\}\rangle\rangle$).

We have already discussed the main frameworks for ILP which work under the answer set semantics and shown generality of these frameworks compare to our own frameworks. In particular, we have shown that although the complexities of our three learning frameworks (ILP_{LAS}, ILP_{LOAS} and $ILP_{LOAS}^{context}$) are the same as cautious induction, there are some learning problems which can be represented in learning from answer sets that cannot be represented in either brave or cautious induction. One example of this is the learning of the rules of Sudoku. This is because brave induction cannot incentivise learning the constraints in the rules of Sudoku, and there are no useful examples that can be given to a cautious learner about the values of cells, since no cell has the same value in every valid Sudoku grid.

Another early work on learning frameworks under the answer set semantics is *Induction from Answer Sets* [Sak05]. In [Sak05], two learning algorithms IAS^{pos} and IAS^{neg} are presented. The task of IAS^{pos} is to learn a hypothesis that cautiously entails a set of examples. This corresponds to the task of cautious induction. IAS^{neg} on the other hand aims to find a hypothesis that does not cautiously entail each of a set of examples (i.e. there should be at least one answer set that does not contain each example). This is (in some sense reversed) brave induction. As shown [Sak05], in general the IAS^{pos} and IAS^{neg} procedures cannot be combined in general to compute a correct hypothesis.

Another framework, under the supported model semantics rather than the answer set semantics, is *Learning from Interpretation Transitions* (LFIT) [IRS14]. In LFIT, the examples are pairs of interpretations $\langle I, J\rangle$ where J is the set of immediate consequences of I given $B \cup H$. In [LRB16], we presented a mapping from any LFIT task to an $ILP^{context}_{LOAS}$ task. This shows that the complexity of deciding both satisfiability and verification for LFIT is at most Σ^P_2-complete. The generality, on the other hand would be different to the tasks we have considered, since there are programs that are strongly equivalent under the answer sets semantics that have different supported models. Example 5.38 demonstrates a pair of such programs, and an example that learning from interpretation transitions could use to distinguish between them.

Example 5.38. *Consider the programs P_1 and P_2.*

$P_1 = \{\texttt{p :- p.}\}$

$P_2 = \emptyset$

P_1 and P_2 are strongly equivalent under the answer set semantics. However, P_1 has the supported model $\{\texttt{p}\}$, whereas P_2 does not. LFIT can distinguish P_1 from P_2 (with respect to an empty background knowledge) with the example $\langle\{\texttt{p}\}, \{\texttt{p}\}\rangle$.

Example 5.38 shows that $ILP^{context}_{LOAS}$ has a distinguishability class which does not contain LFIT's distinguishability class. Conversely, LFIT cannot have a distinguishability class which contains $\mathcal{D}^1_1(ILP^{context}_{LOAS})$, as it cannot distinguish hypotheses containing weak constraints from the same hypotheses without the weak constraints. In fact, it does not even contain $\mathcal{D}^1_1(ILP_{LAS})$, as shown in Example 5.39.

Example 5.39. *Consider the programs P_1 and P_2.*

$P_1 = \left\{ \texttt{p.} \right\}$

$P_2 = \left\{ \begin{array}{l} \texttt{p :- p.} \\ \texttt{p :- not p.} \end{array} \right\}$

For both programs P_1 and P_2, the immediate consequences of any interpretation I is the set $\{\texttt{p}\}$. This means that no example could possibly distinguish P_1 from P_2 with respect to an empty background knowledge. Under the answer set semantics, however, P_1 has one answer set $\{\texttt{p}\}$, but P_2 has no *answer sets.* ILP_{LAS} *can therefore distinguish* P_1 *from* P_2 *(with respect to the empty background knowledge), with the positive example* $\langle\{\texttt{p}\},\emptyset\rangle$.

ILP_{LAS}, ILP_{LOAS} and $ILP_{LOAS}^{context}$ have different distinguishability classes to LFIT, but none is either more or less $\mathcal{D}_1^1$-general than LFIT. This is an interesting observation, as it demonstrates that even when two frameworks are incomparable under our measures of generality, we can still reason about their individual distinguishability classes and discuss hypotheses which one framework is powerful enough to distinguish between and another is not. For instance, ILP_{LAS} cannot distinguish between any two hypotheses that are strongly equivalent under the answer set semantics, but Example 5.38 shows that there are some cases where ILP_{LFIT} can.

5.5.1 Summary

In this chapter we have presented three measures of the generality of a learning framework. We have shown that under each measure, all three of our learning frameworks are more general than any of the existing frameworks, and that $ILP_{LOAS}^{context}$ is the most general of all the frameworks considered. In the rest of Part I, we present our work on algorithms for solving $ILP_{LOAS}^{context}$ tasks concluding in Chapter 8 with an evaluation of our algorithms.

Chapter 6

Using ASP for Inductive Learning of Answer Set Programs

In Chapter 4 we introduced three frameworks for learning answer set programs, the most general of which is $ILP^{context}_{LOAS}$. In this chapter we describe the first algorithm for solving $ILP^{context}_{LOAS}$ tasks, called ILASP1 (Inductive Learning of Answer Set Programs) and prove its soundness and completeness results. Several previous ILP systems [Ray09, CRL12, ACBR13] have used ASP as a computation mechanism for computing inductive solutions. The ASPAL [CRL12] algorithm, for example, translates a brave induction task into an ASP program whose answer sets correspond to the inductive solutions of the original ILP task. We refer to these kinds of ASP encodings and their answer sets as *meta-level programs* and *meta-level answer sets*. We will also refer to answer sets of $B \cup H \cup e_{ctx}$ (for some hypothesis H and some context e_{ctx}) as *object-level answer sets*.

The ILASP algorithms also use a meta-level ASP program in their computation, but as these algorithms are designed to compute solutions to $ILP^{context}_{LOAS}$ tasks, the meta-level ASP programs are more complicated than ASPAL's meta-level program. ASPAL was designed to solve brave induction tasks, in which all examples must be covered in one answer set of the (object-level) program $B \cup H$. For this reason, ASPAL only needs to reason about one object level answer set at a time, and so one answer set of ASPAL's meta-level program encodes a single answer set of the object level program. In $ILP^{context}_{LOAS}$ (and in ILP_{LAS} and ILP_{LOAS}) different examples can be covered by different object-level answer sets, and as a result the meta-level programs used by the ILASP algorithms consider multiple object-level answer sets in a single meta-level answer set. This is achieved through *reification.*

Example 6.1. *Consider the ILP_{LAS} task $T = \langle B, S_M, E^+, E^- \rangle$, where:*

$$B = \left\{ \texttt{p :- not q.} \right\} \qquad S_M = \left\{ \begin{array}{ll} 1: & \texttt{q.} \\ 2: & \texttt{q :- not p.} \end{array} \right\}$$

$$E^+ = \left\{ \begin{array}{ll} e^1: & \langle \{\texttt{p}\}, \{\texttt{q}\} \rangle \\ e^2: & \langle \{\texttt{q}\}, \{\texttt{p}\} \rangle \end{array} \right\}$$

This task can be represented by the following meta-level ASP program P. As there are two positive examples, which can be extended by two different object-level answer sets, each answer set of our meta-level program considers two object-level answer sets, which are given the identifiers `1` *and* `2`*.*

$$P = \left\{\begin{array}{l}
\texttt{\%B} \\
1:\ \ \texttt{as(1).}\quad \texttt{as(2).} \\
2:\ \ \texttt{in\_as(p,AS\_ID) :- not in\_as(q,AS\_ID),as(AS\_ID).} \\
\\
\texttt{\%S}_\texttt{M} \\
3:\ \ \texttt{0\{in\_h(1),in\_h(2)\}2.} \\
4:\ \ \texttt{in\_as(q,AS\_ID) :- as(AS\_ID),in\_h(1).} \\
5:\ \ \texttt{in\_as(q,AS\_ID) :- as(AS\_ID), not in\_as(p,AS\_ID),in\_h(2).} \\
\\
\texttt{\%Examples} \\
6:\ \ \texttt{cov(1) :- in\_as(p,1), not in\_as(q,1).} \\
7:\ \ \texttt{cov(2) :- in\_as(q,2), not in\_as(p,2).} \\
8:\ \ \texttt{:- not cov(1).} \\
9:\ \ \texttt{:- not cov(2).}
\end{array}\right\}$$

The program P uses a similar technique to the meta-level program used in the ASPAL algorithm (see Section 3.2.2). Each rule h in the hypothesis space is appended with an atom $\texttt{in\_h}(\texttt{h}_\texttt{id})$*, which represents whether or not h is in the hypothesis represented by a meta-level answer set. The key difference between this program and the one used by ASPAL is that in this case both the background knowledge and the hypothesis space have been reified. The effect of this is that each answer set of the meta-level program P contains the representation of two (reified) answer sets of the object-level program. The* `as` *facts in lines 1 and 2 introduce two object-level answer sets, with identifiers* `1` *and* `2`*, respectively.*

For example, consider the meta-level answer set $A = \{\texttt{as(1)}, \texttt{as(2)}, \texttt{in\_h(2)}, \texttt{in\_as(p,1)}, \texttt{in\_as(q,2)}, \texttt{cov(1)}, \texttt{cov(2)}\}$ *of P. It shows that there are two object level answer sets* $A_1 = \{\texttt{p}\}$ *(represented by the* `in_as` *atoms with the identifier* `1`*) and* $A_2 = \{\texttt{q}\}$ *(represented by the* `in_as` *atoms with the identifier* `2`*). A_1 extends the first example, and A_2 extends the second example (which is ensured by lines 6-9 of P). Note that sometimes a single object level answer set will extend two different examples. In this case a meta-level answer set may contain the same object level answer set more than once (reified with different identifiers).*

The program in Example 6.1 uses several techniques which are used extensively in the ILASP meta-level programs. For this reason, we introduce some notation which will considerably simplify the definitions of these meta-level programs.

Notation ($\mathcal{A}$). Let P be an ASP program and lit be a literal. $\mathcal{A}(P, \mathtt{lit})$ is the program constructed by adding `lit` to the body of each rule in P.

Notation ($\mathcal{M}^{-1}_{in\_h}$). Let A be a meta-level answer set. $\mathcal{M}^{-1}_{in\_h}(A)$ is the hypothesis $\{h \mid \mathtt{in\_h}(\mathtt{h_{id}}) \in A\}$

We use the following general notation to refer to the reification of a given ASP program.

Notation ($\mathcal{R}$). Let P be an ASP program, `pred` be a predicate symbol and $\{\mathtt{t_1}, \ldots, \mathtt{t_n}\}$ be a (possibly empty) set of terms. $\mathcal{R}(P, \mathtt{pred}, \mathtt{t_1}, \ldots, \mathtt{t_n})$ is the program constructed by replacing every atom `a` in P with the reified atom $\mathtt{pred}(\mathtt{a}, \mathtt{t_1}, \ldots, \mathtt{t_n})$.

In the ILASP encodings, most of the reification uses the same predicate and arguments, and will also append every rule with the same atom. We therefore introduce a shorthand notation for applying these operations. We also introduce a notation for extracting object level answer sets from meta-level answer sets.

Notation ($\mathcal{R}_{\mathtt{in\_as}}$). Given a program P, $\mathcal{R}_{\mathtt{in\_as}}(P) = \mathcal{A}(\mathcal{R}(P, \mathtt{in\_as}, \mathtt{AS\_ID}), \mathtt{as}(\mathtt{AS\_ID}))$.

Notation ($\mathcal{M}^{-1}_{as}$). Given an answer set A of a meta-level program and a constant `id`, we write $\mathcal{M}^{-1}_{as}(A, \mathtt{id})$ to denote the set $\{\mathtt{a} \mid \mathtt{in\_as}(\mathtt{a}, \mathtt{id}) \in A\}$.

Throughout the rest of this thesis, we also assume that every rule, example, interpretation, and other kind of object O has a unique identifier, denoted O_{id}. Furthermore, given a set S of such objects, we use the notation $ids(S)$ to denote the set of identifiers, $\{O_{id} \mid O \in S\}$.

6.1 Meta Representation

In this section, we introduce the main components of the meta-level programs used by ILASP1 to compute inductive solutions of $ILP^{context}_{LOAS}$ tasks[1].

[1]An initial version of ILASP1, targeted at solving ILP_{LAS} tasks, was presented in [LRB14].

Meta-program 6.1 ($\mathcal{M}_1(T)$)**.** Given a learning task T, $\mathcal{M}_1(T)$ is the program consisting of the following components:

- $\mathcal{R}_{\texttt{in\_as}}(r)$ for each rule $r \in non\_weak(B)$
- $\mathcal{A}(\mathcal{R}_{\texttt{in\_as}}(r), \texttt{in\_h}(\texttt{r}_{\texttt{id}}))$ for each rule $r \in non\_weak(S_M)$
- $\mathcal{A}(\mathcal{R}_{\texttt{in\_as}}(e_{ctx}), \texttt{ctx}(\texttt{e}_{\texttt{id}}, \texttt{AS\_ID}))$ for each $e \in E^+ \cup E^-$
- The choice rule $\texttt{1}\{\texttt{ctx}(\texttt{e}^{\texttt{1}}_{\texttt{id}}, \texttt{AS\_ID}), \ldots, \texttt{ctx}(\texttt{e}^{\texttt{n}}_{\texttt{id}}, \texttt{AS\_ID})\}\texttt{1 :- as}(\texttt{AS\_ID})$.
  (where $\{e^1, \ldots, e^n\} = E^+ \cup E^-$)

$\mathcal{M}_1(T)$ consists of the reification of the background knowledge and hypothesis space (where each rule in the hypothesis space has been appended with an `in_h` atom as in Example 6.1). As different object-level answer sets may use different contexts, $\mathcal{M}_1(T)$ contains a choice rule to indicate that exactly one context is used by each object level answer set, and the rules in each context are added with an extra atom appended. Example 6.2 shows how $\mathcal{M}_1$ can be used to reason about multiple object-level answer sets, within a single meta-level answer set.

Example 6.2. *Consider the $ILP^{context}_{LOAS}$ task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$, where:*

$$B = \left\{ \texttt{p :- not q, r.} \right\} \qquad S_M = \left\{ \begin{array}{ll} 1: & \texttt{q.} \\ 2: & \texttt{q :- not p.} \end{array} \right\}$$

$$E^+ = \left\{ \begin{array}{l} e_1 : \langle\langle\{\texttt{p}\}, \{\texttt{q}\}\rangle, \{\texttt{r.}\}\rangle \\ e_2 : \langle\langle\{\texttt{q}\}, \{\texttt{p}\}\rangle, \emptyset\rangle \end{array} \right\} \qquad O^b = \emptyset$$

$$E^- = \emptyset \qquad O^c = \emptyset$$

Consider the program $P = \mathcal{M}_1(T) \cup \{\texttt{as(1). as(2).}\} \cup \{\texttt{in\_h(2).}\}$. The set of facts $\{\texttt{in\_h(2).}\}$ represents the hypothesis $H = \{\texttt{q :- not p.}\}$. In full, P is the program:

```
in_as(p, AS_ID) :- not in_as(q, AS_ID), in_as(r, AS_ID), as(AS_ID).
in_as(q, AS_ID) :- as(AS_ID), in_h(1).
in_as(q, AS_ID) :- not in_as(p, AS_ID), as(AS_ID), in_h(2).
in_as(r, AS_ID) :- as(AS_ID), ctx(1, AS_ID).
1 { ctx(1, AS_ID), ctx(2, AS_ID) } 1 :- as(AS_ID).
as(1).          as(2).          in_h(2).
```

Each answer set of P represents a pair of object-level answer sets of $B \cup H \cup ctx$, where ctx is the context of one of the two examples. For instance, consider the answer set $A = \{$`as(1)`, `as(2)`, `in_h(2)`, `ctx(2, 1)`, `ctx(1, 2)`, `in_as(q, 1)`, `in_as(p, 2)`, `in_as(r, 2)`*$\}$. A represents two object level answer sets: firstly* {`q`}*, which is an answer set of $B \cup H \cup e^2_{ctx}$; and secondly* {`p, r`}*, which is an answer set of $B \cup H \cup e^1_{ctx}$. As there are 3 possible object level answer sets ($B \cup H \cup e^1_{ctx}$ also has the answer set* {`q,`

$\mathtt{r}\}$), and each meta-level answer set represents two object-level answer sets, there are $3^2 = 9$ answer sets of P. Let $Facts$ be the set $\{\mathtt{as(1)},\ \mathtt{as(2)},\ \mathtt{in\_h(2)}\}$. The other 8 answer sets of P, together with each pair of object level answer sets that they represent are shown in the following table:

A	$\mathcal{M}_{as}^{-1}(A, 1)$	$\mathcal{M}_{as}^{-1}(A, 2)$
$Facts \cup \{\mathtt{ctx(1,1)},\ \mathtt{ctx(1,2)},\ \mathtt{in\_as(p,1)},\ \mathtt{in\_as(r,1)},\ \mathtt{in\_as(p,2)},\ \mathtt{in\_as(r,2)}\}$	$\{\mathtt{p},\ \mathtt{r}\}$	$\{\mathtt{p},\ \mathtt{r}\}$
$Facts \cup \{\mathtt{ctx(1,1)},\ \mathtt{ctx(1,2)},\ \mathtt{in\_as(p,1)},\ \mathtt{in\_as(r,1)},\ \mathtt{in\_as(q,2)},\ \mathtt{in\_as(r,2)}\}$	$\{\mathtt{p},\ \mathtt{r}\}$	$\{\mathtt{q},\ \mathtt{r}\}$
$Facts \cup \{\mathtt{ctx(1,1)},\ \mathtt{ctx(1,2)},\ \mathtt{in\_as(q,1)},\ \mathtt{in\_as(r,1)},\ \mathtt{in\_as(p,2)},\ \mathtt{in\_as(r,2)}\}$	$\{\mathtt{q},\ \mathtt{r}\}$	$\{\mathtt{p},\ \mathtt{r}\}$
$Facts \cup \{\mathtt{ctx(1,1)},\ \mathtt{ctx(1,2)},\ \mathtt{in\_as(q,1)},\ \mathtt{in\_as(r,1)},\ \mathtt{in\_as(q,2)},\ \mathtt{in\_as(r,2)}\}$	$\{\mathtt{q},\ \mathtt{r}\}$	$\{\mathtt{q},\ \mathtt{r}\}$
$Facts \cup \{\mathtt{ctx(1,1)},\ \mathtt{ctx(2,2)},\ \mathtt{in\_as(p,1)},\ \mathtt{in\_as(r,1)},\ \mathtt{in\_as(q,2)}\}$	$\{\mathtt{p},\ \mathtt{r}\}$	$\{\mathtt{q}\}$
$Facts \cup \{\mathtt{ctx(1,1)},\ \mathtt{ctx(2,2)},\ \mathtt{in\_as(q,1)},\ \mathtt{in\_as(r,1)},\ \mathtt{in\_as(q,2)}\}$	$\{\mathtt{q},\ \mathtt{r}\}$	$\{\mathtt{q}\}$
$Facts \cup \{\mathtt{ctx(2,1)},\ \mathtt{ctx(1,2)},\ \mathtt{in\_as(q,1)},\ \mathtt{in\_as(q,2)},\ \mathtt{in\_as(r,2)}\}$	$\{\mathtt{q}\}$	$\{\mathtt{q},\ \mathtt{r}\}$
$Facts \cup \{\mathtt{ctx(2,1)},\ \mathtt{ctx(2,2)},\ \mathtt{in\_as(q,1)},\ \mathtt{in\_as(q,2)}\}$	$\{\mathtt{q}\}$	$\{\mathtt{q}\}$

Theorem 6.3 generalises Example 6.2 to show how $\mathcal{M}_1(T)$ can be used to reason about a set of object-level answer sets.

Theorem 6.3. *(proof on page 284) Let T be the $ILP_{LOAS}^{context}$ task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$, $AS_{ids} = \{\mathtt{t_1}, \ldots, \mathtt{t_n}\}$ be a set of terms and let $H \subseteq S_M$. Consider the program $P = \mathcal{M}_1(T) \cup \{\mathtt{as(t).} \mid \mathtt{t} \in AS_{ids}\} \cup \{\mathtt{in\_h(h_{id}).} \mid h \in H\}$. For any list $[\langle I^1, e^1\rangle, \ldots, \langle I^n, e^n\rangle]$ (of length $|AS_{ids}|$), where each I^i is an interpretation and each e^i is selected from $E^+ \cup E^-$, $\exists A \in AS(P)$ such that $\forall i \in [1, n]$, $\mathtt{ctx(e^i_{id}, t_i)} \in A$ and $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i}) = I^i$ if and only if $\forall i \in [1, n]$, $I^i \in AS(B \cup H \cup e^i_{ctx})$.*

The meta-level programs we have considered so far are useful for generating many object-level answer sets within a single meta-level answer set. We now show how to constrain these answer sets to check whether certain examples are accepted.

The *check* program, which is formalised in Meta-program 6.2, contains a single rule, whose body is satisfied if a particular object-level answer set of $B \cup H \cup e_{ctx}$ (with identifier $\mathtt{as\_id}$) extends e_{pi} and the meta-level answer set contains $\mathtt{ctx(e_{id}, as\_id)}$.

Meta-program 6.2 (*check*$(e, \mathtt{as\_id})$)**.** Given any CDPI e, and constant $\mathtt{as\_id}$, *check*$(e, \mathtt{as\_id})$ is the program:

$$\left\{\begin{array}{ll} \mathtt{cov(e_{id}, as\_id)\,\text{:-}} & \mathtt{ctx(e_{id}, as\_id), in\_as(e^1_{inc}, as\_id), \ldots, in\_as(e^m_{inc}, as\_id)} \\ & \mathtt{not\ in\_as(e^1_{exc}, as\_id), \ldots,\ not\ in\_as(e^n_{exc}, as\_id)} \end{array}\right\}$$

Theorem 6.4 shows that we can use the *check* program in conjunction with $\mathcal{M}_1$ in order to determine which object-level answer sets are accepting answer sets of which examples.

Theorem 6.4. *(proof on page 285) Let T be an $ILP_{LOAS}^{context}$ task with background knowledge B and hypothesis space S_M, and let $H \subseteq S_M$. Let $AS_{ids} = \{\mathtt{t_1}, \ldots, \mathtt{t_n}\}$ be a set of terms. For each $\mathtt{t} \in AS_{ids}$ let $E_{\mathtt{t}}$ be a set of CDPIs.*

Consider the program $P = \mathcal{M}_1(T) \;\cup\; \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h \in H\} \;\cup\; \{\texttt{as}(\texttt{t}).\mid \texttt{t} \in AS_{ids}\} \;\cup\; \{check(e, \texttt{t}) \mid \texttt{t} \in AS_{ids}, e \in E_{\texttt{t}}\}$.

1. *For any list* $[\langle I^1, e^1\rangle, \ldots, \langle I^n, e^n\rangle]$ *(of length* $|AS_{ids}|$*) such that each* e^i *is selected from* $E^+ \cup E^-$ *and each* I^i *is an interpretation:* $\exists A \in AS(P)$ *such that* $\forall i \in [1, n]$, $\texttt{ctx}(\texttt{e}^{\texttt{i}}_{\texttt{id}}, \texttt{t}_\texttt{i}) \in A$ *and* $\mathcal{M}^{-1}_{as}(A, \texttt{t}_\texttt{i}) = I^i$ *if and only if* $\forall i \in [1, n]$, $I^i \in AS(B \cup H \cup e^i_{ctx})$.

2. *For any answer set* $A \in AS(P)$, $\forall i \in [1, n]$, $\forall e \in E^+ \cup E^-$, $\texttt{cov}(\texttt{e}_{\texttt{id}}, \texttt{t}_\texttt{i}) \in A$ *if and only if* $\texttt{ctx}(\texttt{e}_{\texttt{id}}, \texttt{t}_\texttt{i}) \in A$, $e \in E_{\texttt{t}_\texttt{i}}$ *and* $\mathcal{M}^{-1}_{as}(A, \texttt{t}_\texttt{i})$ *is an accepting answer set of* e *wrt* $B \cup H$.

We now introduce a similar program, which can be used to check whether or not an ordering example is accepted. We first define a meta-level representation of the weak constraints in $B \cup S_M$. This is given in Definition 6.1. Intuitively, this meta representation expresses that if the body of a weak constraint in a program P is satisfied by an interpretation I, then $(\texttt{w}, \texttt{lev}, \texttt{t}_1, \ldots, \texttt{t}_\texttt{n}) \in weak(P, I)$ (where $[\texttt{w@lev}, \texttt{t}_1, \ldots, \texttt{t}_\texttt{n}]$ is the tail of the weak constraint). The $\texttt{w}$ predicate represents the tuples in $weak(P, I)$[2].

Definition 6.1. Let $\texttt{as\_id}$ be a term. Given a weak constraint, R of the form $:\sim \texttt{b}_1, \ldots, \texttt{b}_\texttt{k}, \texttt{ not c}_1, \ldots, \texttt{ not c}_\texttt{m}.[\texttt{wt@lev}, \texttt{t}_1, \ldots, \texttt{t}_\texttt{n}]$, $\mathcal{R}_{weak}(R, \texttt{as\_id})$ is the rule:

$$\begin{array}{l}\texttt{w}(\texttt{wt}, \texttt{lev}, \texttt{args}(\texttt{t}_1, \ldots, \texttt{t}_\texttt{n}), \texttt{as\_id})\texttt{ :- }\\ \quad \texttt{as}(\texttt{as\_id}), \texttt{in\_as}(\texttt{b}_1, \texttt{as\_id}), \ldots, \texttt{in\_as}(\texttt{b}_\texttt{k}, \texttt{as\_id}),\\ \quad \texttt{not in\_as}(\texttt{c}_1, \texttt{as\_id}), \ldots, \texttt{ not in\_as}(\texttt{c}_\texttt{m}, \texttt{as\_id}).\end{array}$$

For a set of weak constraints W, $\mathcal{R}_{weak}(W, \texttt{as\_id}) = \{\mathcal{R}_{weak}(R, \texttt{as\_id}) \mid R \in W\}$.

Example 6.5. *Consider the program* W *containing one weak constraint* $:\sim \texttt{p}(\texttt{X}, \texttt{Y}).[\texttt{1@1}, \texttt{X}]$ *and the interpretation* $I = \{\texttt{p}(1, 1), \texttt{p}(2, 2), \texttt{p}(1, 2), \texttt{p}(2, 1)\}$.

We can calculate $weak(W, I)$ *using the program* P.

$$P = \left\{\begin{array}{l}\texttt{as(1).}\\ \texttt{in\_as(p(1,1),1).}\\ \texttt{in\_as(p(1,2),1).}\\ \texttt{in\_as(p(2,1),1).}\\ \texttt{in\_as(p(2,2),1).}\\ \texttt{w(1,1,args(X),1) :- as(1),in\_as(p(X,Y),1)}\end{array}\right\}$$

P *has one answer set* $\{\texttt{as(1)}$, $\texttt{in\_as(p(1,1),1)}$, $\texttt{in\_as(p(1,2),1)}$, $\texttt{in\_as(p(1,1),1)}$, $\texttt{in\_as(p(2,2),1)}$, $\texttt{w(1,1,args(1),1)}$, $\texttt{w(1,1,args(2),1)}\}$. *This indicates that* $weak(W, I) = \{(\texttt{1,1,1}), (\texttt{1,1,2})\}$.

[2]Recall from Chapter 2 that given a program P and an interpretation I, a tuple $(\texttt{w}, \texttt{l}, \texttt{t}_1, \ldots, \texttt{t}_\texttt{n}) \in weak(P, I)$ if and only if there is a weak constraint $:\sim \texttt{body}.[\texttt{w@l}, \texttt{t}_1, \ldots, \texttt{t}_\texttt{n}]$ in $ground(P)$ such that I satisfies *body*.

In order to check whether or not an ordering example o is respected, we require two object-level answer sets. The *check_ord* program can be used to check that an ordering example is respected by using two answer set identifiers. The program, given in Meta-program 6.3, is split into 6 parts. Part 1 uses the *check* program from Meta-program 6.2 to check whether the two answer sets are accepted by the two examples in o. The next two parts use $\mathcal{R}_{weak}$ to represent the weak constraints in $B \cup S_M$ (where the weak constraints in S_M have been appended with `in_h` atoms, as usual. Part 4 checks at each priority level whether one answer set dominates the other. The obvious rule that could be used to check whether one answer set dominates another at priority level `lev`, is:

$\texttt{dom\_at\_lv}(\texttt{a}_1, \texttt{a}_2, \texttt{lev})\texttt{:-}\,\texttt{S}_1 = \#\texttt{sum}\{\texttt{w}(\texttt{W}, \texttt{lev}, \texttt{A}, \texttt{a}_1) = \texttt{W}\}, \texttt{S}_2 = \#\texttt{sum}\{\texttt{w}(\texttt{W}, \texttt{lev}, \texttt{A}, \texttt{a}_2) = \texttt{W}\}, \texttt{S}_1 < \texttt{S2}.$

This naive encoding has many ground instances, however, as there is one ground instance for each pair of values $\texttt{s}_1$ and $\texttt{s}_2$ that the variables $\texttt{S}_1$ and $\texttt{S}_2$ could possibly take. The check_ord program therefore instead uses the equivalent rule:

$\texttt{dom\_at\_lv}(\texttt{a}_1, \texttt{a}_2, \texttt{lev})\texttt{:-}\,\#\texttt{sum}\{\texttt{w}(\texttt{W}, \texttt{lev}, \texttt{A}, \texttt{a}_1) = \texttt{W}, \texttt{w}(\texttt{W}, \texttt{lev}, \texttt{A}, \texttt{a}_2) = -\texttt{W}\} < 0.$

Part 5 of the *check_ord* program expresses that one answer set A_1 dominates another answer set A_2 if there is a priority level `lev` such that A_1 dominates A_2 at `lev`, and A_2 does not dominate A_1 at any level which is higher than `lev`. Finally, part 6 expresses that o is accepted if the two answer sets extend the two examples referred to in o and the domination relation between the two answer sets is the same as specified by the operator in o.

Meta-program 6.3 (*check_ord*$(T, o, \texttt{a}_1, \texttt{a}_2)$)**.** Let T be the task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$. Given an ordering example $o \in O^b \cup O^c$, such that $o_{eg1} = e^1$ and $o_{eg2} = e^2$ and any two terms $\texttt{a}_1$ and $\texttt{a}_2$, *check_ord*$(T, o, \texttt{a}_1, \texttt{a}_2)$ is the program comprised of the following components:

1. $check(e^1, \texttt{a}_1) \cup check(e^2, \texttt{a}_2)$
2. $\mathcal{R}_{weak}(B, \texttt{a}_1) \cup \mathcal{R}_{weak}(B, \texttt{a}_2)$
3. For each $W \in weak(S_M)$, the rules $\mathcal{A}(\mathcal{R}_{weak}(W, \texttt{a}_1), \texttt{in\_h}(\texttt{W}_{\texttt{id}}))$ and $\mathcal{A}(\mathcal{R}_{weak}(W, \texttt{a}_2), \texttt{in\_h}(\texttt{W}_{\texttt{id}}))$.
4. For each priority level `lev` that occurs in at least one weak constraint in $B \cup S_M$, the rules:
$$\left\{\begin{array}{l}\texttt{dom\_at\_lv}(\texttt{a}_1, \texttt{a}_2, \texttt{lev})\texttt{:-}\,\#\texttt{sum}\{\texttt{w}(\texttt{W}, \texttt{lev}, \texttt{A}, \texttt{a}_1) = \texttt{W}, \texttt{w}(\texttt{W}, \texttt{lev}, \texttt{A}, \texttt{a}_2) = -\texttt{W}\} < 0.\\ \texttt{dom\_at\_lv}(\texttt{a}_2, \texttt{a}_1, \texttt{lev})\texttt{:-}\,\#\texttt{sum}\{\texttt{w}(\texttt{W}, \texttt{lev}, \texttt{A}, \texttt{a}_1) = \texttt{W}, \texttt{w}(\texttt{W}, \texttt{lev}, \texttt{A}, \texttt{a}_2) = -\texttt{W}\} > 0.\end{array}\right\}$$
5. For each priority level `lev` that occurs in at least one weak constraint in $B \cup S_M$, the rules:
$$\left\{\begin{array}{lll}\texttt{dom}(\texttt{a}_1, \texttt{a}_2) & \texttt{:-} & \texttt{dom\_at\_lv}(\texttt{a}_1, \texttt{a}_2, \texttt{lev}),\\ & & \;\texttt{not dom\_at\_lv}(\texttt{a}_2, \texttt{a}_1, \texttt{l}_1), \ldots, \texttt{ not dom\_at\_lv}(\texttt{a}_2, \texttt{a}_1, \texttt{l}_\texttt{n}).\\ \texttt{dom}(\texttt{a}_2, \texttt{a}_1) & \texttt{:-} & \texttt{dom\_at\_lv}(\texttt{a}_2, \texttt{a}_1, \texttt{lev}),\\ & & \;\texttt{not dom\_at\_lv}(\texttt{a}_1, \texttt{a}_2, \texttt{l}_1), \ldots, \texttt{ not dom\_at\_lv}(\texttt{a}_1, \texttt{a}_2, \texttt{l}_\texttt{n}).\end{array}\right\}$$
(where $\{\texttt{l}_1, \ldots, \texttt{l}_\texttt{n}\}$ are the levels that occur in $B \cup S_M$ that are higher than `lev`)

6. If o_{op} is $<$, the rule:
   $\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{a}_1, \texttt{a}_2)\texttt{:-}\,\texttt{dom}(\texttt{a}_1, \texttt{a}_2), \texttt{cov}(\texttt{e}^1_{\texttt{id}}, \texttt{a}_1), \texttt{cov}(\texttt{e}^2_{\texttt{id}}, \texttt{a}_2).$

   If o_{op} is $>$ the rule:
   $\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{a}_1, \texttt{a}_2)\texttt{:-}\,\texttt{dom}(\texttt{a}_2, \texttt{a}_1), \texttt{cov}(\texttt{e}^1_{\texttt{id}}, \texttt{a}_1), \texttt{cov}(\texttt{e}^2_{\texttt{id}}, \texttt{a}_2).$

   If o_{op} is $\neq$, the rules:
   $\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{a}_1, \texttt{a}_2)\texttt{:-}\,\texttt{dom}(\texttt{a}_1, \texttt{a}_2), \texttt{cov}(\texttt{e}^1_{\texttt{id}}, \texttt{a}_1), \texttt{cov}(\texttt{e}^2_{\texttt{id}}, \texttt{a}_2).$
   $\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{a}_1, \texttt{a}_2)\texttt{:-}\,\texttt{dom}(\texttt{a}_2, \texttt{a}_1), \texttt{cov}(\texttt{e}^1_{\texttt{id}}, \texttt{a}_1), \texttt{cov}(\texttt{e}^2_{\texttt{id}}, \texttt{a}_2).$

   If o_{op} is $=$, the rule:
   $\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{a}_1, \texttt{a}_2)\texttt{:- not dom}(\texttt{a}_1, \texttt{a}_2), \texttt{ not dom}(\texttt{a}_2, \texttt{a}_1), \texttt{cov}(\texttt{e}^1_{\texttt{id}}, \texttt{a}_1), \texttt{cov}(\texttt{e}^2_{\texttt{id}}, \texttt{a}_2).$

   If o_{op} is $\leq$, the rule:
   $\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{a}_1, \texttt{a}_2)\texttt{:- not dom}(\texttt{a}_2, \texttt{a}_1), \texttt{cov}(\texttt{e}^1_{\texttt{id}}, \texttt{a}_1), \texttt{cov}(\texttt{e}^2_{\texttt{id}}, \texttt{a}_2).$

   If o_{op} is $\geq$, the rule:
   $\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{a}_1, \texttt{a}_2)\texttt{:- not dom}(\texttt{a}_1, \texttt{a}_2), \texttt{cov}(\texttt{e}^1_{\texttt{id}}, \texttt{a}_1), \texttt{cov}(\texttt{e}^2_{\texttt{id}}, \texttt{a}_2).$

We now demonstrate that we can combine $\mathcal{M}_1$ with instances of the *check* and *check_ord* programs in order to determine which examples and ordering examples are accepted by a hypothesis H. The program presented in Theorem 6.6 generates a set of object-level answer sets and checks which answer sets are accepting answer sets of which examples, and which pairs of object-level answer sets are accepting pairs of answer sets of o. In this and subsequent chapters we use programs of this form, but for efficiency we often do not need to check whether the object level answer set (resp. pair of object level answer sets) for every term (resp. pair of terms) is an accepting answer set (resp. accepting pair of answer sets) of every CDPI (resp. CDOE) in the task. We therefore specify a set of CDPIs (resp. CDOEs) for each term (resp. pair of terms) that should be checked. The first point of the theorem shows that the meta-level answer sets correspond to the set of all combinations of object-level answer sets. Point (2) shows that given a meta-level answer set A of the program, for each CDPI e such that $\texttt{cov}(\texttt{e}_{\texttt{id}}, \texttt{t}) \in A$, the object level answer set represented by $\texttt{t}$ in A is guaranteed to be an accepting answer set of e. Similarly, for each CDOE o such that $\texttt{ord\_respected}(\texttt{o}, \texttt{t}_1, \texttt{t}_2) \in A$, the pair of object-level answer sets $\langle A_1, A_2\rangle$ represented by the terms $\texttt{t}_1$ and $\texttt{t}_2$ (respectively) is guaranteed to be a pair of accepting answer sets of o.

Theorem 6.6. *(proof on page 286) Let T be an $ILP^{context}_{LOAS}$ task with background knowledge B and hypothesis space S_M, and let $H \subseteq S_M$. Let $AS_{ids} = \{\texttt{t}_1, \ldots, \texttt{t}_\texttt{n}\}$ be a set of terms and $Pair_{ids}$ be a set of pairs $\langle \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}\rangle$, where $\texttt{t}_\texttt{i}$ and $\texttt{t}_\texttt{j}$ are terms in AS_{ids} such that no term occurs more than once in $Pair_{ids}$.*

For each $\texttt{t} \in AS_{ids}$ let $E_\texttt{t}$ be a set of CDPIs and for each tuple $p \in Pair_{ids}$ let O_p be a set of CDOEs.

$$\begin{array}{lll} \textit{Let } P = \mathcal{M}_1(T) & \cup & \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}) \mid h \in H\} \\ & \cup & \{\texttt{as}(\texttt{t}). \mid \texttt{t} \in AS_{ids}\} \\ & \cup & \{\mathit{check}(e, \texttt{t}) \mid \langle \texttt{t}, e\rangle \in E_{\texttt{t}}, \texttt{t} \in AS_{ids}\} \\ & \cup & \{\mathit{check\_ord}(T, o, \texttt{t}_{\texttt{i}}, \texttt{t}_{\texttt{j}}) \mid p = \langle \texttt{t}_{\texttt{i}}, \texttt{t}_{\texttt{j}}\rangle \in Pair_{ids}, o \in O_p\} \end{array}$$

For each term $\texttt{t} \in AS_{ids}$*, let* $E(\texttt{t}) = E_{\texttt{t}} \cup \{e \mid p = \langle \texttt{t}, \_\rangle \in Pair_{ids}, \langle e, \_, \_\rangle \in O_p\} \cup \{e \mid p = \langle \_, \texttt{t}\rangle \in Pair_{ids}, \langle \_, e, \_\rangle \in O_p\}$.

1. *For any list* $[\langle I^1, e^1\rangle, \ldots, \langle I^n, e^n\rangle]$ *(of length* $|AS_{ids}|$*) such that each* e^i *is selected from* $E^+ \cup E^-$ *and each* I^i *is an interpretation:* $\exists A \in AS(P)$ *such that* $\forall i \in [1, n]$, $\texttt{ctx}(\texttt{e}^{\texttt{i}}_{\texttt{id}}, \texttt{t}_{\texttt{i}}) \in A$ *and* $\mathcal{M}^{-1}_{as}(A, \texttt{t}_{\texttt{i}}) = I^i$ *if and only if* $\forall i \in [1, n]$, $I^i \in AS(B \cup H \cup e^i_{ctx})$.

2. *For any answer set* $A \in AS(P)$, $\forall i \in [1, n]$, $\forall e \in E^+ \cup E^-$, $\texttt{cov}(\texttt{e}_{\texttt{id}}, \texttt{t}_{\texttt{i}}) \in A$ *if and only if* $\texttt{ctx}(\texttt{e}_{\texttt{id}}, \texttt{t}_{\texttt{i}}) \in A$, $e \in E(\texttt{t}_{\texttt{i}})$ *and* $\mathcal{M}^{-1}_{as}(A, \texttt{t}_{\texttt{i}})$ *is an accepting answer set of* e *wrt* $B \cup H$.

3. *For any* $A \in AS(P)$*, for any ordering example* $o = \langle oe^1, oe^2, op\rangle$ *and for any* $i, j \in [1, n]$, $\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{t}_{\texttt{i}}, \texttt{t}_{\texttt{j}}) \in A$ *if and only if* $p = \langle \texttt{t}_{\texttt{i}}, \texttt{t}_{\texttt{j}}\rangle \in Pair_{ids}$, $o \in O_p$, $\texttt{cov}(\texttt{oe}^{1}_{\texttt{id}}, \texttt{t}_{\texttt{i}}) \in A$, $\texttt{cov}(\texttt{oe}^{2}_{\texttt{id}}, \texttt{t}_{\texttt{j}}) \in A$ *and* $\langle \mathcal{M}^{-1}_{as}(A, \texttt{t}_{\texttt{i}}), \mathcal{M}^{-1}_{as}(A, \texttt{t}_{\texttt{j}})\rangle$ *is an accepting pair of answer sets of* o *wrt* $B \cup H$.

6.2 Searching for Positive and Violating Hypotheses

The programs presented in the previous section provide a way to check whether a hypothesis accepts a set of CDPIs and CDOEs. A hypothesis can be represented as a set of `in_h` facts added to the meta-level program and a set of examples are accepted if there is an answer set that contains `cov` and `ord_respected` atoms for each of the examples. In order to use this program to search for hypotheses that accept all of these examples, we must make two changes to the program. First, rather than adding a set of facts describing the hypothesis, we must add a choice rule which expresses that any rule in the hypothesis space can either be in, or not in the hypothesis. Second, we must add constraints expressing that each of the required `cov` and `ord_respected` atoms must occur in any answer set of the program. For the case with only CDPIs, this program would be similar to the program P in Example 6.1.

The programs presented so far only generate accepting answer sets of CDPIs and accepting pairs of answer sets of CDOEs. This allows us to easily generate the hypotheses which cover all of the positive examples and brave orderings; however, enforcing the coverage of negative examples and cautious orderings is more complicated. Example 6.7 illustrates why a naive approach, using meta-level constraints to enforce that negative examples are covered, does not work.

Example 6.7. *Consider the* $ILP^{context}_{LOAS}$ *task* $T = \langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$*, where:*

$B = \left\{ \texttt{p :- not q.} \right\}$

$S_M = \left\{ \begin{array}{l} \texttt{q.} \\ \texttt{q :- not p.} \end{array} \right\}$

$E^+ = \left\{ e_1 : \ \langle\langle\{\texttt{p}\},\{\texttt{q}\}\rangle,\emptyset\rangle \right\}$

$E^- = \left\{ e_2 : \ \langle\langle\{\texttt{q}\},\{\texttt{p}\}\rangle,\emptyset\rangle \right\}$

$O^b = \emptyset$

$O^c = \emptyset$

It might be thought that this task could be represented by the meta-level ASP program P.

$$P = \mathcal{M}_1(T) \cup \left\{ \begin{array}{l} \texttt{as(1).} \\ \texttt{as(2).} \\ \texttt{:- not cov(1,1).} \\ \texttt{:- cov(2,2).} \end{array} \right\} \cup \mathit{check}(e_1, 1) \cup \mathit{check}(e_2, 2).$$

P is shown in full below.

```
     %B
     1 :  as(1).    as(2).
     2 :  in_as(p, AS_ID) :- not in_as(q, AS_ID).

     %S_M
     3 :  0{in_h(1), in_h(2)}2.
P =  4 :  in_as(q, AS_ID) :- as(AS_ID), in_h(1).
     5 :  in_as(q, AS_ID) :- as(AS_ID), not in_as(p, AS_ID), in_h(2).

     %Examples
     6 :  cov(1, 1) :- in_as(p, 1), not in_as(q, 1).
     7 :  cov(2, 2) :- in_as(q, 2), not in_as(p, 2).
     8 :  :- not cov(1, 1).
     9 :  :- cov(2, 2).
```

P has one answer set $\{\texttt{as(1)},\texttt{as(2)},\texttt{in\_h(2)},\texttt{in\_as(p,1)},\texttt{in\_as(p,2)},\texttt{cov(1,1)}\}$. *This would indicate that* `q :- not p` *is an inductive solution of the task, but this is not the case (as the hypothesis has the answer set* $\{\texttt{q}\}$*, when combined with the background knowledge). The problem arises because the negative example e_2 is covered only if there are no accepting answer sets of e_2 with respect to $B \cup H$, whereas the program P ensures only that there is at least one object level answer set of $B \cup H$ that is not an accepting answer set of e_2.*

In fact, as shown in Section 4.5, deciding the satisfiability of propositional $ILP_{LOAS}^{context}$ tasks is Σ_2^P-complete. This means that unless the polynomial hierarchy collapses, in general we cannot write an ASP program using only normal rules, choice rules and constraints whose answer sets can be mapped to the inductive solutions an $ILP_{LOAS}^{context}$ task (as deciding satisfiability of these programs is only Σ_1^P-complete). In ASP there are two ways of solving Σ_2^P-complete problems. ASP solvers such as clingo (starting with clasp-D [GKS13]) permit the use of disjunction in ASP. Deciding the satisfiability of disjunctive logic programs and programs containing unstratified aggregates is Σ_2^P-complete. In practice, however, encoding Σ_2^P-complete problems in ASP requires advanced techniques such as saturation [EG95]. Such encodings can be unintuitive, and are often only understandable by experts in ASP [GKS11]. Another method for accessing higher levels of the polynomial hierarchy in ASP is to solve an ASP program iteratively (i.e. solving the program many times, each time amending the program, based on the previous iteration).

The algorithms in this thesis explore the second direction. By using an iterative approach, we have been able to develop algorithms that are specifically targeted at solving $ILP_{LOAS}^{context}$ tasks, rather than delegating the entire Σ_2^P problem to a general-purpose ASP solver.

In each iteration, ILASP1 searches for hypotheses which cover all of the positive examples and brave orderings – we call these the *positive hypotheses*. It may find some hypotheses which do not cover some negative examples or cautious orderings. These are still positive hypotheses, but they are not inductive solutions – we call these the *violating hypotheses*. The idea of the ILASP1 algorithm is to rule these hypotheses out by adding constraints to the meta-level program, before solving it again. Once there are no violating hypotheses left, the remaining positive hypotheses are in fact the inductive solutions. Definition 6.2 formalises the notions of positive and violating hypotheses.

Definition 6.2. Let $T = \langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$ be an $ILP_{LOAS}^{context}$ task. A hypothesis $H \subseteq S_M$ is said to be *positive* if $\forall e \in E^+$, H accepts e and $\forall o \in O^b$, H accepts o. A positive hypothesis H is said to be *violating* if there is at least one $e \in E^-$ such that H accepts e or at least one $o \in O^c$ such that H does not cautiously respect o. We write $\mathcal{P}(T)$ and $\mathcal{V}(T)$ to denote the positive and violating hypotheses of T. We also write $\mathcal{P}^n(T)$ and $\mathcal{V}^n(T)$ to denote the positive and violating hypotheses of length n (for any $n \in \mathbb{N}$).

Example 6.8. *Consider the learning task $T = \langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$, where:*

$$B = \left\{ \begin{array}{l} \texttt{0\{p\}1.} \\ \texttt{q:- not r.} \end{array} \right\} \qquad E^+ = \left\{ \begin{array}{ll} e^1: & \langle\langle\{\texttt{q}\}, \emptyset\rangle, \emptyset\rangle \\ e^2: & \langle\langle\{\texttt{r}\}, \emptyset\rangle, \emptyset\rangle \end{array} \right\}$$

$$S_M = \left\{ \begin{array}{ll} \texttt{h}_1: & \texttt{r:- not q.} \\ \texttt{h}_2: & \texttt{:-p,q.} \\ \texttt{h}_3: & \texttt{:}\sim\texttt{p.[1@1]} \\ \texttt{h}_4: & \texttt{:}\sim\texttt{r.[1@1]} \end{array} \right\} \qquad \begin{array}{l} E^- = \left\{ e^3: \ \langle\langle\{\texttt{p}\}, \{\texttt{r}\}\rangle, \emptyset\rangle \right\} \\ O^b = \left\{ o^1: \ \langle e^1, e^2, <\rangle \right\} \\ O^c = \left\{ o^2: \ \langle e^1, e^2, <\rangle \right\} \end{array}$$

Note that in practice, we would not usually have identical CDOEs in O^b and O^c (if an ordering is cautiously respected then it is also bravely respected).

$\emptyset$ is not a positive hypothesis, as it does not cover the positive example e_2. In fact, to be a positive hypothesis (given the hypothesis space S_M), a hypothesis must contain $\texttt{h}_1$, because otherwise the example e_2 cannot be covered (no rule in B, e^2_{ctx} or $S_M\backslash\{\texttt{h}_1\}$ has $\texttt{r}$ in the head). The hypothesis $\{\texttt{h}_1\}$ is still not a positive hypothesis, as it does not cover the brave ordering o_1 (there are no weak constraints in

$B \cup H$, so all answer sets are equally preferred). The hypothesis $\{$`h`$_1$, `h`$_3\}$ is a positive hypothesis, as it prefers the answer set $\{$`q`$\}$ to the answer set $\{$`p`, `r`$\}$. $\{$`h`$_1$, `h`$_3\}$ is not an inductive solution, however; instead it is violating, as it does not cover the negative example e_3. The hypothesis $\{$`h`$_1$, `h`$_2$, `h`$_3\}$ covers this negative example, as the extra constraint eliminates the answer set $\{$`p`, `q`$\}$, but it is still violating as it does not cautiously respect the cautious ordering o_2 (it does not prefer the answer set $\{$`q`$\}$ to the answer set $\{$`r`$\}$ – as neither pays any penalty). The hypothesis $\{$`h`$_1$, `h`$_2$, `h`$_4\}$ does cautiously respect o_2, and is not a violating hypothesis. Hence, as it is a positive hypothesis that is not violating, it is an inductive solution of T.

We now prove that the inductive solutions of any task are indeed those positive hypotheses which are not violating hypotheses. The proof follows from the definitions of an inductive solution, and a positive and violating hypothesis.

Theorem 6.9. *Let T be an $ILP^{context}_{LOAS}$ task. Then $ILP^{context}_{LOAS}(T) = \mathcal{P}(T)\backslash\mathcal{V}(T)$.*

Proof. Let $H \in ILP^{context}_{LOAS}(T)$

$\Leftrightarrow H \subseteq S_M$, $\forall e \in E^+, B \cup H$ accepts e, $\forall e \in E^-$, $B \cup H$ does not accept e, $\forall o \in O^b$, $B \cup H$ bravely respects o and $\forall o \in O^c$, $B \cup H$ cautiously respects o.

$$\Leftrightarrow H \in \left\{ H \subseteq S_M \,\middle|\, \begin{array}{c} \forall e \in E^+, B \cup H \text{ accepts } e \text{ and} \\ \forall o \in O^b, B \cup H \text{ bravely respects } o \end{array} \right\}$$

$$\text{and } H \notin \left\{ H \subseteq S_M \,\middle|\, \begin{array}{c} \exists e \in E^-, B \cup H \text{ accepts } e \text{ or} \\ \exists o \in O^c, B \cup H \text{ does not} \\ \text{cautiously respect } o \end{array} \right\}$$

$$\Leftrightarrow H \in \mathcal{P}(T) \text{ and } H \notin \left\{ H \subseteq S_M \,\middle|\, \begin{array}{c} \exists e \in E^-, B \cup H \text{ accepts } e \text{ or} \\ \exists o \in O^c, B \cup H \text{ does not cautiously respect } o \end{array} \right\}$$

$$\Leftrightarrow H \in \mathcal{P}(T) \text{ and } H \notin \left\{ H \in \mathcal{P}(T) \,\middle|\, \begin{array}{c} \exists e \in E^-, B \cup H \text{ accepts } e \text{ or} \\ \exists o \in O^c, B \cup H \text{ does not cautiously respect } o \end{array} \right\}$$

$\Leftrightarrow H \in \mathcal{P}(T)$ and $H \notin \mathcal{V}(T)$

$\Leftrightarrow H \in \mathcal{P}(T)\backslash\mathcal{V}(T)$

□

We can now present the main meta-level program, $\mathcal{M}(T)$, of the ILASP1 algorithm, which builds on the partial meta-level programs we have presented so far. $\mathcal{M}(T)$ introduces one object-level answer set per positive example and two object-level answer sets per brave ordering. We use $\mathcal{M}(T)$ in two different ways: searching for positive solutions, and searching for violating solutions – in the second case, we add the fact "`check_violating.`" to the program. In the case that we are searching for violating solutions it is sufficient to find a negative example or cautious ordering that is not covered. We therefore only need to introduce two extra object-level answer sets (`v1` and `v2`) in this case. If these two answer sets are an accepting pair of answer sets for any cautious ordering, or if `v1` is an accepting answer set of any negative example, then the hypothesis represented by the meta-level answer set is guaranteed to be a violating hypothesis.

There is also a constraint expressing that if the fact `check_violating` is present, then `violating` must be present in any answer set A of $\mathcal{M}(T)$, where `violating` is true if and only if at least one of the following conditions hold:

1. There is a negative example e such that $\mathcal{M}_{as}^{-1}(A, \texttt{v1})$ is an accepting answer set of e. In this case, we call $\mathcal{M}_{as}^{-1}(A, \texttt{v1})$ a *violating interpretation* of e. A will contain the atom $\texttt{v\_i}(\texttt{e}_{\texttt{id}})$.
2. There is a cautious ordering o such that $\langle \mathcal{M}_{as}^{-1}(A, \texttt{v1}), \mathcal{M}_{as}^{-1}(A, \texttt{v2})\rangle$ is an accepting pair of answer sets of $inverse(o)$. In this case, we call $\langle \mathcal{M}_{as}^{-1}(A, \texttt{v1}), \mathcal{M}_{as}^{-1}(A, \texttt{v2})\rangle$ a *violating pair of interpretations* of o. A will contain the atom $\texttt{v\_p}(\texttt{o}_{\texttt{id}})$.

Note that each brave ordering example requires a unique pair of object-level answer sets to be represented in each meta-level answer set. If there are two ordering examples o_1 and o_2 that "share" the same positive examples, a pair of answer sets proving that o_1 is respected may not prove that o_2 is respected. We introduce an extra pair of unique identifiers o_{id1} and o_{id2} for each ordering example o. Note that these are not equal to $(o_{eg1})_{id}$ and $(o_{eg2})_{id}$.

Meta-program 6.4 ($\mathcal{M}(T)$)**.** Let T be the $ILP_{LOAS}^{context}$ task, $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$. $\mathcal{M}(T)$ is the program comprised of the following components:

- The choice rule $\texttt{0}\{\texttt{in\_h}(\texttt{h}_{\texttt{id}})\}\texttt{1}$ for each $h \in S_M$.
- $\mathcal{M}_1(T)$
- $\left\{ \begin{array}{l} \texttt{as}(\texttt{e}_{\texttt{id}}). \\ \texttt{:- not cov}(\texttt{e}_{\texttt{id}}, \texttt{e}_{\texttt{id}}). \end{array} \right\} \cup check(e, e_{id})$, for each $e \in E^+$.
- $\left\{ \begin{array}{l} \texttt{as}(\texttt{o}_{\texttt{id1}}). \\ \texttt{as}(\texttt{o}_{\texttt{id2}}). \\ \texttt{:- not ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{o}_{\texttt{id1}}, \texttt{o}_{\texttt{id2}}). \end{array} \right\} \cup check\_ord(o, o_{id1}, o_{id2})$, for each $o \in O^b$.
- $\left\{ \begin{array}{l} \texttt{as(v1) :- check\_violating.} \\ \texttt{as(v2) :- check\_violating.} \\ \texttt{:- check\_violating, not violating.} \\ \texttt{violating :- v\_i(\_), check\_violating.} \\ \texttt{violating :- v\_p(\_), check\_violating.} \end{array} \right\}$
- $\left\{ \texttt{v\_i}(\texttt{e}_{\texttt{id}}) \texttt{ :- cov}(\texttt{e}_{\texttt{id}}, \texttt{v1}). \right\} \cup check(e, \texttt{v1})$, for each $e \in E^-$.

- $\left\{\begin{array}{l}\texttt{v\_p(o}_{\texttt{id}}\texttt{) :-}\\ \quad\texttt{ord\_respected(o}_{\texttt{id}}\texttt{, v1, v2).}\end{array}\right\} \cup \mathit{check\_ord}(\mathit{inverse}(o), \texttt{v1}, \texttt{v2})$, for each $o \in O^c$.

$\mathcal{M}(T)$ has several key properties that we show in Theorem 6.10. For ILASP1, the most important of these properties are (1) that the answer sets of $\mathcal{M}(T)$ correspond to the positive hypotheses of T, and (4), that if $\mathcal{M}(T)$ is combined with an extra fact (`check_violating`), then the answer sets correspond to the violating hypotheses of T. Properties (2) and (3) will be used in later chapters.

Theorem 6.10. *(proof on page 290) Let T be an $ILP_{LOAS}^{context}$ task, and H be a hypothesis.*

1. *$\exists A \in AS(\mathcal{M}(T))$ such that $H = \mathcal{M}_{in\_h}^{-1}(A)$ if and only if $H \in \mathcal{P}(T)$.*
2. *$\forall e \in E^-$, for any $A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$ such that $\texttt{v\_i(e}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}_{in\_h}^{-1}(A)$, $\mathcal{M}_{as}^{-1}(A, \texttt{v1})$ is an accepting answer set of e wrt $B \cup H$.*
3. *$\forall o \in O^c$, for any $A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$ such that $\texttt{v\_p(o}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}_{in\_h}^{-1}(A)$, $\langle \mathcal{M}_{as}^{-1}(A, \texttt{v1}), \mathcal{M}_{as}^{-1}(A, \texttt{v2})\rangle$ is an accepting pair of answer sets of $inverse(o)$ wrt $B \cup H$.*
4. *$\exists A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$ such that $H = \mathcal{M}_{in\_h}^{-1}(A)$ if and only if $H \in \mathcal{V}(T)$.*

6.3 The ILASP1 Algorithm

Algorithm 6.1 shows the ILASP1 algorithm. The main idea of the approach is to iteratively compute the set of inductive solutions of length n, starting at $n = 0$. The first time this set is non-empty, it must be the set of optimal inductive solutions, and so it is returned. In each iteration, there are two steps: firstly, we compute the set of violating hypotheses at the current length, using Meta-program 6.4; these violating hypotheses are then converted into meta-level constraints, which are added to our meta-level program before solving it again. The answer sets of the second meta-level program correspond to the positive hypotheses which are not violating. By Theorem 6.9, these are the inductive solutions. The meta-level programs are solved using the clingo [GKK$^+$11] ASP solver.

In order to restrict the answer sets of our meta-level program to hypotheses of a given length n, we add a constraint with a summing aggregate in the body. The augmented program is formalised in Meta-program 6.5.

Meta-program 6.5 ($\mathcal{M}^n(T)$)**.** Let T be the $ILP_{LOAS}^{context}$ task $\langle B, S_M, E\rangle$ and let $n \in \mathbb{N}$, where $S_M = \{R^1, \ldots, R^m\}$.
$\mathcal{M}^n(T) = \mathcal{M}(T) \cup \left\{\texttt{:-}\ \texttt{\#sum}\{\texttt{in\_h(R}^{\texttt{1}}_{\texttt{id}}\texttt{)} = |\texttt{R}^{\texttt{1}}|, \ldots, \texttt{in\_h(R}^{\texttt{m}}_{\texttt{id}}\texttt{)} = |\texttt{R}^{\texttt{m}}|\} \neq \texttt{n.}\right\}$

We now show that for any n the answer sets of $\mathcal{M}^n(T)$ are the answer sets A of $\mathcal{M}(T)$ such that the hypothesis represented by A has length n.

Algorithm 6.1 ILASP1

```
1: procedure ILASP1($T, max$)
2:     $solutions = \emptyset$;
3:     for $n = 0$; $solutions == \emptyset$ && $n \leq max$; $n$++
4:         $AS_1 = AS(\mathcal{M}^n(T) \cup \{\texttt{check\_violating.}\})$;
5:         $vs = \{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS_1\}$;
6:         $AS_2 = AS(\mathcal{M}^n(T) \cup \{constraint(V) \mid V \in vs\})$;
7:         $solutions = \{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS_2\}$;
8:     end for
9:     return $solutions$;
10: end procedure
```

Proposition 6.11. *(proof on page 295)* Let T be any $ILP^{context}_{LOAS}$ task and $n \in \mathbb{N}$. $AS(\mathcal{M}^n(T)) = \{A \in \mathcal{M}(T) \mid |\mathcal{M}^{-1}_{in\_h}(A)| = n\}$.

Similarly to $\mathcal{M}(T)$, $\mathcal{M}^n(T) \cup \{\texttt{check\_violating.}\}$ allows us to compute violating hypotheses – specifically, it allows us to compute the violating hypotheses of a given length n. Once we have computed these violating hypotheses, we use the meta-level constraints formalised in Meta-program 6.6 in order to rule them out.

Meta-program 6.6 ($constraint(H)$)**.** Let H be the hypothesis $\{h^1, \ldots, h^n\}$. $constraint(H) = \{\texttt{:- in\_h}(\texttt{h}^1_{\texttt{id}}), \ldots, \texttt{in\_h}(\texttt{h}^{\texttt{n}}_{\texttt{id}}).\}$

In Proposition 6.12, we show that given a set of hypotheses V, we can use the constraint representation in Meta-program 6.6 combined with the $\mathcal{M}^n(T)$ program to find the positive solutions that are not in V.

Proposition 6.12. *(proof on page 295)* Let T be an $ILP^{context}_{LOAS}$ task, $n \in \mathbb{N}$ and V be a set of hypotheses of length n. Consider the program $P = \mathcal{M}^n(T) \cup \{constraint(v) \mid v \in V\}$. Then $\mathcal{P}^n(T) \backslash V = \{\mathcal{M}^{-1}_{in\_h}(A) | A \in AS(P)\}$.

We have now proved the intermediate results necessary to show the soundness and completeness of ILASP1. To ensure termination, ILASP1 is called with an upper bound (max) on the length of a hypothesis. Without this upper bound, ILASP would only terminate for satisfiable tasks – for unsatisfiable tasks, *solutions* would be empty at the end of every iteration.

In each iteration, ILASP1 first computes the violating hypotheses vs of length n, and then computes the positive hypotheses ps of length n, which are not in vs (i.e. the inductive solutions of length n). If this set is non-empty then the algorithm terminates and returns the set of optimal inductive solutions of the task. Theorem 6.14 shows that ILASP1 is guaranteed to terminate and is both sound and complete with respect to the optimal solutions (with length less than or equal to the given upper bound) of any well-defined $ILP^{context}_{LOAS}$ task.

In order to prove the termination of the ILASP1 algorithm, we must first show that each call to the ASP solver terminates. In theory, this should be the case, provided the relevant groundings of the programs being solved are finite. Proposition 6.13, shows that this is the case for the meta-level programs used by ILASP1 to solve any task T, so long as all of the object-level programs that can be constructed from the components of T have a finite grounding. We say that a task T is *well-defined* if every program consisting of rules in the background knowledge, hypothesis space and contexts of T has a finite relevant grounding.

Proposition 6.13. *(proof on page 296)* Let T be any well-defined $ILP^{context}_{LOAS}$ task, $n \in \mathbb{N}$, and HS be any finite set of hypotheses.

1. $ground^{rel}(\mathcal{M}^n(T) \cup \{\texttt{check\_violating.}\})$ is finite.
2. $ground^{rel}(\mathcal{M}^n(T) \cup \{constraint(H) \mid H \in HS\})$ is finite.

We are now ready to prove the termination, soundness and completeness results for the ILASP1 algorithm. Theorem 6.14 shows that given any well-defined $ILP^{context}_{LOAS}$ task T and any upper bound max, $ILASP1(T, max)$ terminates in a finite time and returns the set of optimal inductive solutions H of T such that $|H| \leq max$; i.e. if there is an inductive solution H of T such that $|H| \leq max$ then $ILASP1(T, max)$ returns the set of optimal inductive solutions of T; otherwise, $ILASP1(T, max)$ returns $\emptyset$.

Theorem 6.14. *Let $max \in \mathbb{N}$ and let T be an $ILP^{context}_{LOAS}$ task.*

1. *$ILASP1(T, max)$ terminates for any well-defined task.*
2. *$ILASP1(T, max) = \{H \in {}^*ILP^{context}_{LOAS}(T) \mid |H| \leq max\}$*

Proof.

1. As there are clearly a finite number of iterations, it remains to show that any arbitrary iteration of the for loop terminates. Let $n \in \mathbb{N}$. Since by Proposition 6.13 (1), $ground^{rel}(\mathcal{M}^n(T) \cup \{\texttt{check\_violating.}\})$ is finite, the first call to the ASP solver terminates. In the next line vs must be finite, as vs must be a subset of the powerset of S_M. Hence by Proposition 6.13 (2), the second call to the ASP solver must also terminate. Hence, the iteration must terminate.
2. We first show that at the end of each iteration, *solutions* is equal to the inductive solutions of length n.

   In each iteration, after line 5, $vs = \{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS(\mathcal{M}^n(T) \cup \{\texttt{check\_violating.}\})\}$. Hence, by Proposition 6.11, $vs = \{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\}), |\mathcal{M}^{-1}_{in\_h}(A)| = n\}$. By Theorem 6.10 (part 4), this means that $vs = \{H \in \mathcal{V}(T) \mid |H| = n\}$ (i.e. $vs = \mathcal{V}^n(T)$).

Hence, at the end of each iteration, $solutions = \{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS(\mathcal{M}^n(T) \cup \{constraint(V) \mid V \in \mathcal{V}^n(T)\})\}$. By Proposition 6.12, this means that $solutions = \mathcal{P}^n(T)\backslash\mathcal{V}^n(T)$.

Hence, at the end of each iteration, *solutions* is equal to the set of inductive solutions of length n. We can now show that $ILASP1(T, max) = \{H \in {}^*ILP^{context}_{LOAS}(T) \mid |H| \leq max\}$.

Case 1: $\exists H \in ILP^{context}_{LOAS}$ such that $|H| \leq max$

In this case, the set of optimal solutions must all have length m for some integer $m \leq max$. In all iterations where $n < m$, *solutions* must have been empty at the end of the iteration (otherwise there would be an inductive solution with length smaller than m). Hence, as $ILASP1$ is guaranteed to terminate (by part (1)), iteration m must be reached. At the end of iteration m, *solutions* must therefore be the (non-empty) set of optimal inductive solutions of the task.

Hence, in this case, $ILASP1(T, max)$ returns $\{H \in {}^*ILP^{context}_{LOAS}(T) \mid |H| \leq max\}$ (as all optimal inductive solutions have length m which is less than or equal to max).

Case 2: $\nexists H \in ILP^{context}_{LOAS}$ such that $|H| \leq max$

In this case, as there are no inductive solutions H such that $|H| \leq max$, $\{H \in {}^*ILP^{context}_{LOAS}(T) \mid |H| \leq max\}$ is empty. Hence, it remains to show that $ILASP1(T, max)$ returns $\emptyset$.

At the end of each iteration *solutions* must be empty, as otherwise there would be a solution with length less than or equal to max. Hence, $ILASP1(T, max)$ returns $\emptyset$.

$\square$

6.3.1 Scalability Issues with the ILASP1 Approach

There are two main scalability issues with the ILASP1 algorithm. The first is that, in principle, there may be an extremely large number of violating hypotheses that need to be found before the optimal inductive solutions are found. We address this in the ILASP2 algorithm (Section 7.1), by introducing the notion of *violating reasons*, which enable us to rule out many violating hypotheses which are all violating for the same "reason".

The second issue for scalability is the time it takes to solve the meta-level programs. Generally speaking, ASP programs with larger relevant groundings take longer to solve. The size of the relevant grounding of $\mathcal{M}(T)$ is roughly proportional to $as \times |ground^{rel}(B \cup S_M \cup CS)|$ (where as is the number of object level answer sets considered and CS is the union of the all of the contexts of examples). This is because the relevant grounding of $\mathcal{M}(T)$ essentially contains as copies of the relevant grounding of $B \cup S_M \cup CS$. Note that when the meta-level program is ground, the contexts "mix" – the relevant grounding does not consider whether two atoms can appear in the same answer set, but only whether each of them can occur in at least one answer set. Due to the size of the grounding, the performance is affected by each of the following factors:

1. The size of the problem domain (i.e. the Herbrand domain), as this affects the size of the relevant grounding of $B \cup S_M \cup CS$.

2. The number of positive examples. This is because $as = |E^{+}| + 2 \times |O^{b}| + 2$ (there is one meta-level answer set for each positive example, two for each brave ordering and two extra – represented by `v1` and `v2`).

3. The size of B, S_M and CS. It should be noted that the sizes of S_M and CS usually have a greater effect than the size of B. This is because in the meta-level programs, B does not have an extra condition (an `in_h` or `ctx` atom), and so the ASP solver can use the rules in B to make simplifications. For example, if a fact "`a.`" occurs in B, then clingo will remove any positive occurrence of `a` from the bodies of ground instances of other rules.

In Section 7.2, we show how we can dramatically reduce the number of examples and contexts used in the meta encoding, so that the grounding becomes significantly smaller. In Chapter 10, we go further, and show how we can consider examples independently from all other examples, meaning that that version of $\mathcal{M}(T)$ considers at most two object-level answer sets, and therefore its grounding is not proportional in size to any number of examples.

6.4 Related Work

We now discuss how two of the other ILP systems which use ASP solvers, ASPAL and XHAIL, relate to the ILASP approach. As both ASPAL and XHAIL are systems for brave induction, the results in Chapter 5 mean that there are some programs which can be learned using ILASP that are out of reach for these two systems; for instance, neither will ever learn a hypothesis that contains a constraint.

6.4.1 ASPAL

For brave induction tasks, the closest ILP system to ILASP is the ASPAL system, which works by encoding an ILP_b task as a meta-level ASP program, whose optimal answer sets correspond exactly to the optimal inductive solutions of the task. We now reconsider the learning task in Example 3.15.

Example 6.15. *Consider the following (equivalent) tasks for the ASPAL and ILASP systems. The first is a brave induction task T_1 with traditional mode declarations, and the second is an ILP_{LAS} task T_2 with a fully specified hypothesis space. Note that the positive and negative examples in T_1 are combined into a single positive partial interpretation example in T_2.*

$$B_1 = \left\{ \begin{array}{l} \texttt{bird(a).} \\ \texttt{bird(b).} \\ \texttt{can(a, fly).} \\ \texttt{can(b, swim).} \\ \texttt{ability(fly).} \\ \texttt{ability(swim).} \end{array} \right\} \qquad B_2 = \left\{ \begin{array}{l} \texttt{bird(a).} \\ \texttt{bird(b).} \\ \texttt{can(a, fly).} \\ \texttt{can(b, swim).} \\ \texttt{ability(fly).} \\ \texttt{ability(swim).} \end{array} \right\}$$

$$M = \left\{ \begin{array}{l} \texttt{\#modeh(penguin(+bird))} \\ \texttt{\#modeb(2, not can(+bird, \#ability))} \end{array} \right\} \qquad S_M = \left\{ \begin{array}{l} \texttt{penguin(X) :- bird(X).} \\ \texttt{penguin(X) :- bird(X), not can(X, fly).} \\ \texttt{penguin(X) :- bird(X), not can(X, swim).} \\ \texttt{penguin(X) :- bird(X), not can(X, fly),} \\ \qquad\qquad\quad \texttt{not can(X, swim).} \end{array} \right\}$$

$$E_2^+ = \{\texttt{penguin(b)}\} \qquad E_2^+ = \{\langle\{\texttt{penguin(b)}\}, \{\texttt{penguin(a)}\}\rangle\}$$

$$E_2^- = \{\texttt{penguin(a)}\} \qquad E^- = \emptyset$$

The two meta-level programs produced by the two algorithms for solving this task are shown below; first, the meta-level program of ASPAL, and then of ILASP. In the case of ASPAL, we have replaced its choice rule with an equivalent one that conforms to the subset of ASP considered in this thesis.

$$ASPAL\_Meta = \left\{ \begin{array}{l} \texttt{bird(a).} \\ \texttt{bird(b).} \\ \texttt{can(a, fly).} \\ \texttt{can(b, swim).} \\ \texttt{ability(fly).} \\ \texttt{ability(swim).} \\ \\ \texttt{penguin(X) :- bird(X), rule(1).} \\ \texttt{penguin(X) :- bird(X), not can(X, C1), rule(2, C1).} \\ \texttt{penguin(X) :- bird(X), not can(X, C1), not can(X, C2), rule(3, C1, C2).} \\ \\ \texttt{0\{rule(1), rule(2, fly), rule(2, swim), rule(3, fly, swim)\}4.} \\ \\ \texttt{goal :- penguin(b), not penguin(a).} \\ \texttt{:- not goal.} \end{array} \right\}$$

ILASP_Meta =

```
0{in_h(1), in_h(2), in_h(3), in_h(4)}4.

in_as(bird(a), AS_ID) :- as(AS_ID).
in_as(bird(b), AS_ID) :- as(AS_ID).
in_as(can(a, fly)) :- as(AS_ID).
in_as(can(b, swim), AS_ID) :- as(AS_ID).
in_as(ability(fly), AS_ID) :- as(AS_ID).
in_as(ability(swim), AS_ID) :- as(AS_ID).

in_as(penguin(X), AS_ID) :- as(AS_ID), in_as(bird(X), AS_ID), in_h(1).
in_as(penguin(X), AS_ID) :- as(AS_ID), in_as(bird(X), AS_ID),
   not in_as(can(fly, C1), AS_ID), in_h(2).
in_as(penguin(X), AS_ID) :- as(AS_ID), in_as(bird(X), AS_ID),
   not in_as(can(swim, C1), AS_ID), in_h(3).
in_as(penguin(X), AS_ID) :- as(AS_ID), in_as(bird(X), AS_ID),
   not in_as(can(fly, C1), AS_ID), not in_as(can(swim, C2), AS_ID), in_h(4).

as(1).
cov(1, AS_ID) :- in_as(penguin(b), AS_ID), not in_as(penguin(a), AS_ID), ctx(1, 1).
:- not cov(1, 1).

1{ctx(1, AS_ID)}1 :- as(AS_ID).
as(v1) :- check_violating.
as(v2) :- check_violating.
:- check_violating, not violating.
violating :- v_i(_), check_violating.
violating :- v_p(_), check_violating.
```

Other than the last set of rules in the ILASP_Meta program, the two programs have very similar structures (although ILASP_Meta has been reified and uses the `cov` *predicate rather than a single* `goal` *atom). The additional rules in the ILASP program are there because the ILASP system is more general and these extra rules are needed for solving tasks that have other kinds of examples (such as context-dependent examples, negative (partial interpretation) examples and ordering examples). Similarly, the reification is needed for ILASP to be able to solve tasks with multiple positive examples.*

Another difference between the two meta-level programs is that ASPAL "delegates" the filling in of constants in the hypothesis space to the ASP grounder. ILASP, on the other hand, enumerates the hypothesis space in full. The two ground programs both contain the full hypothesis space, so there is little difference between the two approaches for tasks with only one positive example. In general, when ILASP has negative (partial interpretation) examples, this program could be solved many times, meaning that it is more efficient to compute the hypothesis space once, rather than having the ASP grounder compute it multiple times.

In general, these differences in representation have a negligible impact on the performance of the two algorithms, as the size of the ground programs is almost exactly the same. One large difference between *the two is that the ILASP1 algorithm presented in this chapter solves this program twice; once with the* `check_violating` *fact, and once without, which means that it is likely to take twice as long as ASPAL. For this reason, the implementation of ILASP1 first checks whether the task contains any negative examples or cautious orderings, and if there are none it skips the first step of solving the program with the* `check_violating` *fact.*

6.4.2 XHAIL

XHAIL is another algorithm for brave induction, and so, similarly to ASPAL, is unable to learn some of the hypotheses that can be learned by ILASP. For the tasks that can be expressed by XHAIL, XHAIL can be more scalable than ILASP (and ASPAL) as it does not enumerate the hypothesis space in full, but first abduces the (ground instances of) heads of the rules in the hypothesis, then deduces the atoms that can appear in the bodies of (ground instances of) the rules. Finally, it uses its inductive phase to find the final hypothesis.

Both the abductive and inductive phases of the XHAIL algorithm are performed using meta-level ASP programs. In problems with large hypothesis spaces these programs have considerably smaller groundings than the ILASP meta-level programs, as the XHAIL programs do not contain the full hypothesis space. For problems with large (groundings of) background knowledges, however, XHAIL has similar scalability issues to ILASP, as its meta-level program in the abductive phase still contains the grounding of the background knowledge over the entire problem domain.

Unlike ILASP (and ASPAL), XHAIL is not guaranteed to return the optimal inductive solution of any task. This is caused by XHAIL choosing the "wrong" abductive solution in the abductive phase. Example 6.16 shows a task where this occurs.

Example 6.16. *Consider the following XHAIL task.*

```
#example p(a).
#example not p(b).
#example not p(c).
#example not p(d).

% Atoms defining types used in the
% mode bias.
qt(a).  qt(b).
rt(a).  rt(c).
pt(a).  pt(b).  pt(c).  pt(d).
int(1..3).
```

```
cond(1, a).  cond(1, b).  cond(1, c).
cond(2, a).  cond(2, b).  cond(2, d).
cond(3, a).  cond(3, c).  cond(3, d).

p(X) :- q(X), r(X).


#modeh p(+pt).
#modeh q(+qt).
#modeh r(+rt).
#modeb cond($int,+pt).
```

Unlike ILASP, XHAIL adds "types" for each variable that occurs in a hypothesis. The optimal inductive solution of this task would be the two rules "`q(V1):-qt(V1).`*" and "*`r(V1):-rt(V1).`*". XHAIL* *returns the longer hypothesis: "*`p(V1):-pt(V1),cond(1,V1),cond(2,V1),cond(3,V1).`*". This is because XHAIL picks the smallest abductive solution in the first phase, which is* {`p(a)`}*. The smallest inductive solution that can be constructed from this solution is the one that is returned by XHAIL. If it had instead picked the larger abductive solution* {`q(a)`*,* `r(a)`} *then it could have found the true optimal inductive solution.*

Summary

In this chapter we have presented the first of our ILASP algorithms for solving $ILP_{LOAS}^{context}$ tasks. As discussed in Section 6.3.1, there are several scalability issues with the ILASP1 approach. The next chapter introduces the ILASP2 and ILASP2i algorithms, which are specifically targeted at solving some of these scalability issues.

Chapter 7

Scalable Learning of Answer Set Programs

In the previous chapter, we highlighted some potential efficiency and scalability issues of the ILASP1 algorithm. The main cause of these issues is that the size of the grounding of the meta-level programs being solved can become very large. In this chapter, we present two improved versions of the ILASP algorithm that address these issues. ILASP2 replaces ILASP1's inefficient notion of violating hypotheses with a more general notion of *violating reason*, which can lead to more compact meta-level programs when solving tasks with negative examples. ILASP2i is an iterative algorithm that considers a subset of the examples, called the *relevant examples*. This can have a dramatic effect on the size of the grounding of meta-level programs, as this grounding size is proportional to the number of examples being considered. The effect is that ILASP2i is not only much more scalable than ILASP1 and ILASP2 with respect to the number of examples, but also with respect to the size of the problem domain and hypothesis space, which both have an additional impact on the size of the grounding of the meta-programs.

7.1 ILASP2

One of the major scalability issues of the ILASP1 algorithm is that there may be many violating hypotheses that need to be found and converted to constraints, before the first inductive solution is found.

Example 7.1. *Consider the ILP_{LAS} task $T = \langle B, S_M, \langle E^+, E^- \rangle \rangle$, where S_M is defined by the mode declarations M and:*

$$B = \left\{ \begin{array}{l} \texttt{0\{p\}1.} \\ \texttt{0\{q\}1.} \\ \texttt{0\{r\}1.} \end{array} \right\} \qquad E^+ = \left\{ \begin{array}{ll} e_1: & \langle \{\texttt{p}\}, \emptyset \rangle \\ e_2: & \langle \{\texttt{q}\}, \emptyset \rangle \\ e_3: & \langle \{\texttt{r}\}, \emptyset \rangle \end{array} \right\}$$

$$M = \left\{ \begin{array}{ll} \texttt{\#modeh(1,p).} & \texttt{\#modeb(1,p).} \\ \texttt{\#modeh(1,q).} & \texttt{\#modeb(1,q).} \\ \texttt{\#modeh(1,r).} & \texttt{\#modeb(1,r).} \\ \texttt{\#modeh(1,s).} & \texttt{\#modeb(1,s).} \end{array} \right\} \qquad E^- = \left\{ \begin{array}{ll} e_4: & \langle\{\texttt{p},\texttt{q}\},\emptyset\rangle \\ e_5: & \langle\{\texttt{p},\texttt{r}\},\emptyset\rangle \\ e_6: & \langle\{\texttt{q},\texttt{r}\},\emptyset\rangle \end{array} \right\}$$

One optimal inductive solution of this task is the set of constraints C:

$$C = \left\{ \begin{array}{l} \texttt{:- p,q.} \\ \texttt{:- q,r.} \\ \texttt{:- p,r.} \end{array} \right\}$$

C has length 6, and so before ILASP1 finds C, it must rule out every positive hypothesis of length 5 or lower (as they are all violating). There are 97352 such positive hypotheses, and each of them must be converted to a constraint. ILASP1 takes 41.3s to solve the task.

This simple example illustrates a scalability issue of the ILASP1 algorithm, related to the number of violating hypotheses that need to be ruled out before the first inductive solution is found. In practice, with larger hypothesis spaces and larger optimal solutions, there can be many more violating hypotheses, and so ILASP1 would require more computation time. The root cause of the problem is that ILASP1 rules out each of the violating hypotheses as an individual constraint, meaning that its meta-level program becomes very large. However, it is possible that many hypotheses could be violating for the same "reason". In this chapter, we introduce the notion of violating reasons *and present an improved algorithm, ILASP2, which is able to make use of this notion, leading to much more compact meta-level programs. In the task above, for instance, ILASP2 only requires 8 violating reasons to rule out all of the 97352 violating hypotheses computed by ILASP1, thus reducing the computational time from 41.3s to just 0.16s.*

Algorithm 7.1 shows the ILASP2 algorithm. The notation $solve(P)$ means that Clingo is used to find a single optimal answer set of P (if P is unsatisfiable then `nil` is returned). In ILASP1, each iteration includes two main steps: firstly, we search for the violating hypotheses of a particular length, and secondly we search for the remaining positive hypotheses, once the violating hypotheses found in the first step have been ruled out. In ILASP2, we accumulate a set of *violating reasons*, VR (initialised to $\emptyset$ in line 2 of Algorithm 7.1, and expanded in each iteration in line 5). In each iteration, we solve the meta-level program, $\mathcal{M}_{ILASP2}$, searching for the shortest positive hypothesis that does not violate any of the violating reasons in VR. If the meta-level answer set A that we find contains the atom `violating`, then we can extract a new violating reason that rules out the hypothesis represented by A, and any other hypotheses that share this violating reason. In this way, ILASP2 can rule out many violating hypotheses of different lengths in a single iteration, whereas in each iteration of ILASP1, the violating hypotheses of a particular length are converted to constraints. Furthermore, the grounding of the representation of violating reasons can be much more compact than the grounding of ILASP1's meta-level program, which contains an individual constraint for each violating hypothesis that has been computed.

Algorithm 7.1 ILASP2

```
1: procedure ILASP2($T$)
2:     $VR = \emptyset$;
3:     $A = solve(\mathcal{M}_{ILASP2}(T, VR))$;
4:     while $A \neq$ nil && violating $\in A$
5:         $VR.insert(extractVR(A))$;
6:         $A = solve(\mathcal{M}_{ILASP2}(T, VR))$
7:     end while
8:     return $\{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS^*(\mathcal{M}_{ILASP2}(T, VR))\}$
9: end procedure
```

In the rest of this section we formalise the notion of violating reasons and show how we can both find and eliminate these violating reasons using meta-level ASP programs.

7.1.1 Violating Reasons

We now formalise the notion of a violating reason. In general, there can be two reasons why a hypothesis H is violating: it could accept a negative example, or it could not respect a cautious ordering. More specifically, there could be an interpretation I that is an accepting answer set of some negative example with respect to H – in this case, we call I a *violating interpretation*; or there could be a pair of interpretations $\langle I_1, I_2\rangle$ that is an accepting pair of answer sets of $inverse(o)$ for some cautious ordering o – in this case, we call $\langle I_1, I_2\rangle$ a *violating pair* of interpretations. The set of all violating interpretations and violating pairs of a task are collectively called the violating reasons. Definition 7.1 formalises the notion of violating interpretations.

Definition 7.1. Let T be the $ILP^{context}_{LOAS}$ task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$. Let $e \in E^-$ and $H \subseteq S_M$. An interpretation I is said to be a *violating interpretation* of H wrt e if I extends e_{pi} and $I \in AS(B \cup H \cup e_{ctx})$.

An interpretation I is said to be a violating interpretation of T if there is at least one $H \subseteq S_M$ and at least one $e \in E^-$ such that I is a violating interpretation of H wrt e.

Example 7.2 revisits the task from Example 6.8, and exemplifies the concept of violating interpretations.

Example 7.2. *Consider the learning task $T = \langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$, where:*

$$B = \left\{ \begin{array}{l} \texttt{0\{p\}1.} \\ \texttt{q :- not r.} \end{array} \right\} \qquad S_M = \left\{ \begin{array}{ll} \texttt{h}_1: & \texttt{r :- not q.} \\ \texttt{h}_2: & \texttt{:- p, q.} \\ \texttt{h}_3: & \texttt{:}\sim\texttt{ p.[1@1]} \\ \texttt{h}_4: & \texttt{:}\sim\texttt{ r.[1@1]} \end{array} \right\}$$

$$E^{+} = \left\{ \begin{array}{ll} e_1: & \langle\langle\{\texttt{q}\}, \emptyset\rangle, \emptyset\rangle \\ e_2: & \langle\langle\{\texttt{r}\}, \emptyset\rangle, \emptyset\rangle \end{array} \right\} \qquad O^b = \left\{ \; o_1: \; \langle e^1, e^2, < \rangle \; \right\}$$

$$E^{-} = \left\{ \; e_3: \; \langle\langle\{\texttt{p}\}, \{\texttt{r}\}\rangle, \emptyset\rangle \; \right\} \qquad O^c = \left\{ \; o_2: \; \langle e^1, e^2, < \rangle \; \right\}$$

The hypothesis $\{\texttt{h}_1, \texttt{h}_3\}$ *is a positive hypothesis, as it covers the positive examples and it prefers the answer set* $\{\texttt{q}\}$ *to the answer set* $\{\texttt{p, r}\}$*. As stated in Example 6.8,* $\{\texttt{h}_1, \texttt{h}_3\}$ *is not an inductive solution, as it does not cover the negative example* e_3*. More specifically,* $\{\texttt{p, q}\}$ *is a violating interpretation of* $\{\texttt{h}_1, \texttt{h}_3\}$ *with respect to* e_3*.*

Definition 7.2 formalises the notion of violating pairs of interpretations.

Definition 7.2. Let T be the $ILP^{context}_{LOAS}$ task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$. Let $o \in O^c$ and $H \subseteq S_M$. A pair of interpretations $\langle I_1, I_2\rangle$ is said to be a *violating pair* of interpretations of H wrt o if each of the following conditions hold:

1. I_1 extends $(o_{eg1})_{pi}$ and $I_1 \in AS(B \cup H \cup (o_{eg1})_{ctx})$
2. I_2 extends $(o_{eg2})_{pi}$ and $I_2 \in AS(B \cup H \cup (o_{eg2})_{ctx})$
3. $\langle I_1, I_2, o_{op}\rangle \notin ord(B \cup H, \{I_1, I_2\})$

A pair of interpretations $\langle I_1, I_2\rangle$ is said to be a violating pair of T if there is at least one $H \subseteq S_M$ and at least one $o \in O^c$ such that $\langle I_1, I_2\rangle$ is a violating pair of H wrt o.

Example 7.3 revisits the task from Example 6.8, and exemplifies the concept of violating pairs.

Example 7.3. *Reconsider the task T from Example 7.2. The hypothesis* $\{\texttt{h}_1, \texttt{h}_2, \texttt{h}_3\}$ *is a positive hypothesis, with no violating interpretations. It is not an inductive solution, as it does not cover the cautious ordering* o_2*. More specifically,* $\langle\{\texttt{p, q}\}, \{\texttt{r}\}\rangle$ *is a violating pair of* $\{\texttt{h}_1, \texttt{h}_2, \texttt{h}_3\}$ *wrt* o_2*.*

Definition 7.3 formalises the notion of *violating reasons.* A violating reason is a pair $\langle R, e\rangle$, where e is an example and R is either a violating interpretation or violating pair of at least one hypothesis with respect to e.

Definition 7.3. Let T be the $ILP^{context}_{LOAS}$ task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$. A *violating reason* of T is a tuple $\langle R, e\rangle$, such that one of the following holds:

1. $e \in E^-$ and $\exists H \subseteq S_M$ such that R is a violating interpretation of H with respect to e.
2. $e \in O^c$ and $\exists H \subseteq S_M$ such that R is a violating pair of H with respect to e.

Searching for Violating Reasons

The meta-level program $\mathcal{M}(T)$ can be used to find violating reasons. We now introduce two pieces of notation which are useful when extracting violating interpretations and violating pairs from meta-level answer sets. Recall that `v1` and `v2` are identifiers for object-level answer sets in the program $\mathcal{M}(T)$, which are used to identify violating hypotheses.

Notation ($\mathcal{M}^{-1}_{vi}$). Let A be a meta-level answer set. $\mathcal{M}^{-1}_{vi}(A) = \mathcal{M}^{-1}_{as}(A, \texttt{v1})$

Notation ($\mathcal{M}^{-1}_{vp}$). Let A be a meta-level answer set. $\mathcal{M}^{-1}_{vp}(A) = \langle \mathcal{M}^{-1}_{as}(A, \texttt{v1}), \mathcal{M}^{-1}_{as}(A, \texttt{v2})\rangle$

In iterations of the ILASP2 algorithm $\mathcal{M}(T)$ is solved once per iteration with the choice rule $\texttt{0\{check\_violating\}1}$, instead of solving it twice with and without the fact `check_violating`. Theorem 7.4 shows that this program can be used both to find positive hypotheses, and to find violating reasons.

Theorem 7.4. *(proof on page 297) Let T be an $ILP^{context}_{LOAS}$ task, and H be a hypothesis. Consider the program $P = \{\texttt{0\{check\_violating\}1.}\} \cup \mathcal{M}(T)$.*

1. *$\exists A \in AS(P)$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ if and only if $H \in \mathcal{P}(T)$.*
2. *For any $A \in AS(P)$ and any $e \in E^-$, if $\texttt{v\_i(e}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$ then $\mathcal{M}^{-1}_{vi}(A)$ is a violating interpretation of H with respect to e.*
3. *For any $A \in AS(P)$ and any $o \in O^c$, if $\texttt{v\_p(o}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$ then $\mathcal{M}^{-1}_{vp}(A)$ is a violating pair of interpretations of H with respect to o.*
4. *$\exists A \in AS(P)$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ and $\texttt{violating} \in A$ if and only if $H \in \mathcal{V}(T)$.*

Algorithm 7.2 uses the result of Theorem 7.4 to extract violating reasons from meta-level answer sets. If the meta-level answer set contains $\texttt{v\_i(e}_{\texttt{id}}\texttt{)}$ for some $e \in E^-$, then a violating interpretation is extracted for e; otherwise, if there is an $o \in O^c$ such that the meta-level answer set contains $\texttt{v\_p(o}_{\texttt{id}}\texttt{)}$, then a violating pair is extracted.

7.1.2 Representing Violating Reasons in ASP

In the previous section, we showed that the meta-level $\mathcal{M}(T)$ program can be used to find violating reasons. In this section, we show that given a set of violating reasons VR, we can augment $\mathcal{M}(T)$

Algorithm 7.2 extractVR

```
1: procedure EXTRACTVR($A$)
2:     if $\exists e$ st $\texttt{v\_i}(\texttt{e}_{\texttt{id}}) \in A$
3:         return $\langle \mathcal{M}_{vi}^{-1}(A), e\rangle$;
4:     else if $\exists o$ such that $\texttt{v\_p}(\texttt{o}_{\texttt{id}}) \in A$
5:         return $\langle \mathcal{M}_{vp}^{-1}(A), o\rangle$;
6:     end if
7: end procedure
```

to constrain away any meta-level answer set which represents a hypothesis that is violating for any of the reasons in VR. This augmented version of $\mathcal{M}(T)$ is the program $\mathcal{M}_{ILASP2}(T, VR)$, used by Algorithm 7.1.

Representing Violating Interpretations

Consider a hypothesis H and a pair $\langle I, e\rangle$, where e is a negative example, and I is an interpretation that was found to be a violating interpretation of a different hypothesis H' wrt e, but is not necessarily a violating interpretation of H wrt e. We now describe a meta-level ASP program that enables us to check whether I is a violating interpretation of H wrt e. We do this by checking whether I is an answer set of $B \cup H \cup e_{ctx}$, which can be done by computing the minimal model of the reduct of $B \cup H \cup e_{ctx}$ with respect to I. Example 7.5 exemplifies the way in which we compute the minimal model of a reduct within our meta-level program.

Example 7.5. *Consider the normal logic program P.*

$$P = \left\{ \begin{array}{l} \texttt{p :- r, not q.} \\ \texttt{q :- r, not p.} \\ \texttt{r.} \end{array} \right\}$$

Given an interpretation I, the reduct P^I consists of the rules head(R):-body$^+(R)$ such that $R \in P$ and body$^-(R) \cap I = \emptyset$. Given an interpretation I, we can represent this in a meta-level ASP program, by adding the interpretation as a (reified) set of facts, and reifying the negative literals in the body of P. The program P' demonstrates this with the interpretation $\{\texttt{p}, \texttt{r}\}$.

$$P' = \left\{ \begin{array}{l} \texttt{in\_int(p).} \\ \texttt{in\_int(r).} \\ \texttt{p :- r, not in\_int(q).} \\ \texttt{q :- r, not in\_int(p).} \\ \texttt{r.} \end{array} \right\}$$

Using the splitting set theorem, it can be shown that the answer sets of this program are of the form $A_1 \cup A_2$, where $A_1 = \mathcal{R}(I, \texttt{in\_int})$ and A_2 is an answer set of the reduct of P^I. As the reduct is a definite logic program, this means that the program has a single answer set $\mathcal{R}(I, \texttt{in\_int}) \cup M(P^I)$. In *this case, P' has the single answer set* $\{\texttt{in\_int(p)}, \texttt{in\_int(r)}, \texttt{p}, \texttt{r}\}$*, indicating that the minimal model of* P^I *is* $\{\texttt{p}, \texttt{r}\}$.

When using this idea in the ILASP2 algorithm, we will need to reason about multiple interpretations, and so we use a slightly more complicated reification, introducing an identifier for each interpretation that we are testing, and reifing the positive body literals and heads of rules with the predicate `mmr` *(standing for* the Minimal Model of the Reduct*).*

The program P'' demonstrates how we can compute the minimal models of the reduct of P with respect to the interpretations $\{\texttt{p}, \texttt{r}\}$ *and* $\{\texttt{q}, \texttt{r}\}$.

$$P'' = \left\{ \begin{array}{ll} \texttt{int(1).} & \texttt{int(2).} \\ \texttt{in\_int(p, 1).} & \texttt{in\_int(q, 1).} \\ \texttt{in\_int(q, 2).} & \texttt{in\_int(r, 2).} \\ \multicolumn{2}{l}{\texttt{mmr(p, INT\_ID) :- int(INT\_ID), mmr(p, INT\_ID), not in\_int(q, INT\_ID).}} \\ \multicolumn{2}{l}{\texttt{mmr(q, INT\_ID) :- int(INT\_ID), mmr(q, INT\_ID), not in\_int(p, INT\_ID).}} \\ \multicolumn{2}{l}{\texttt{mmr(r, INT\_ID) :- int(INT\_ID).}} \end{array} \right\}$$

P'' has a single answer set { `int(1)`, `int(2)`, `in_int(p, 1)`, `in_int(r, 1)`, `in_int(q, 2)`, `in_int(r, 2)`, `mmr(p, 1)`, `mmr(r, 1)`, `mmr(q, 2)`, `mmr(r, 2)` }*, which indicates that the minimal model of the reduct of P with respect to* $\{\texttt{p}, \texttt{r}\}$ *is* $\{\texttt{p}, \texttt{r}\}$ *and the minimal model of the reduct of P with respect to* $\{\texttt{q}, \texttt{r}\}$ *is* $\{\texttt{q}, \texttt{r}\}$*. This means that both interpretations are answer sets of P.*

Meta-program 7.1 formalises the general notion of the meta-level reduct representation. As our programs in general contain choice rules and constraints (in addition to the normal rules considered in Example 7.5), the definition must also consider the transformation of these rules in a reduct. Before presenting the meta-level program, we introduce a notation to represent reifying a conjunction of atoms, while also negating each atom.

Notation ($\mathcal{NR}$). Let $\{\texttt{a}_1, \ldots, \texttt{a}_n\}$ be a set of atoms, `pred` be a predicate symbol and $\{\texttt{t}_1, \ldots, \texttt{t}_m\}$ be a (possibly empty) set of terms. $\mathcal{NR}(\{\texttt{a}_1, \ldots, \texttt{a}_n\}, \texttt{pred}, \texttt{t}_1, \ldots, \texttt{t}_m)$ is the conjunction $\texttt{not pred}(\texttt{a}_1, \texttt{t}_1, \ldots, \texttt{t}_m), \ldots, \texttt{not pred}(\texttt{a}_n, \texttt{t}_1, \ldots, \texttt{t}_m)$.

Meta-program 7.1 ($\mathcal{M}_{reduct}(P, \texttt{int\_id})$)**.** Let P be an ASP program containing normal rules, choice rules and hard constraints, and let `int_id` be a term. $\mathcal{M}_{reduct}(P, \texttt{int\_id})$ is the program consisting of the following rules:

1. For each normal rule $R \in P$, the rule:
   $$\begin{array}{lll} \texttt{mmr(head(R), int\_id)} & \texttt{:-} & \texttt{int(int\_id)}, \mathcal{R}(\texttt{body}^+\texttt{(R)}, \texttt{mmr}, \texttt{int\_id}), \\ & & \mathcal{NR}(\texttt{body}^-\texttt{(R)}, \texttt{in\_int}, \texttt{int\_id}). \end{array}$$
2. For each hard constraint $R \in P$, the rule:

$$\begin{aligned}\texttt{mmr}(\bot, \texttt{int\_id}) \;\texttt{:-}\; & \texttt{int(int\_id)}, \mathcal{R}(\texttt{body}^{+}(\texttt{R}), \texttt{mmr}, \texttt{int\_id}),\\ & \mathcal{NR}(\texttt{body}^{-}(\texttt{R}), \texttt{in\_int}, \texttt{int\_id}).\end{aligned}$$

3. For each choice rule $R \in P$, where $head(R) = \texttt{lb}\{\texttt{h}_1, \ldots, \texttt{h}_\texttt{n}\}\texttt{ub}$, the rules:

$\texttt{mmr}(\bot, \texttt{int\_id})\texttt{:-}\texttt{RB}, \texttt{ub} + 1\{\texttt{in\_int}(\texttt{h}_1, \texttt{int\_id}), \ldots, \texttt{in\_int}(\texttt{h}_\texttt{n}, \texttt{int\_id})\}.$

$\texttt{mmr}(\bot, \texttt{int\_id})\texttt{:-}\texttt{RB}, \{\texttt{in\_int}(\texttt{h}_1, \texttt{int\_id}), \ldots, \texttt{in\_int}(\texttt{h}_\texttt{n}, \texttt{int\_id})\}\texttt{lb} - 1.$

$\texttt{mmr}(\texttt{h}_1, \texttt{int\_id})\texttt{:-}\texttt{RB}, \texttt{in\_int}(\texttt{h}_1, \texttt{int\_id}), \texttt{lb}\{\texttt{in\_int}(\texttt{h}_1, \texttt{int\_id}), \ldots, \texttt{in\_int}(\texttt{h}_\texttt{n}, \texttt{int\_id})\}\texttt{ub}.$

$\ldots$

$\texttt{mmr}(\texttt{h}_\texttt{n}, \texttt{int\_id})\texttt{:-}\texttt{RB}, \texttt{in\_int}(\texttt{h}_\texttt{n}, \texttt{int\_id}), \texttt{lb}\{\texttt{in\_int}(\texttt{h}_1, \texttt{int\_id}), \ldots, \texttt{in\_int}(\texttt{h}_\texttt{n}, \texttt{int\_id})\}\texttt{ub}.$

where $\texttt{RB} = \texttt{int(int\_id)}, \mathcal{R}(\texttt{body}^{+}, \texttt{mmr}, \texttt{int\_id}), \mathcal{NR}(\texttt{body}^{-}, \texttt{in\_int}, \texttt{int\_id})$

Example 7.6. *Consider the program P.*

$$P = \left\{\begin{array}{l}\texttt{1\{p, q\}1 :- r.}\\ \texttt{:- r, not p.}\\ \texttt{r.}\end{array}\right\}$$

$$\mathcal{M}_{reduct}(P, 1) = \left\{\begin{array}{l}\texttt{mmr}(\bot, 1)\texttt{:-}\texttt{int}(1), \texttt{mmr}(\texttt{r}, 1), 2\{\texttt{in\_int}(\texttt{p}, 1), \texttt{in\_int}(\texttt{q}, 1)\}.\\ \texttt{mmr}(\bot, 1)\texttt{:-}\texttt{int}(1), \texttt{mmr}(\texttt{r}, 1), \{\texttt{in\_int}(\texttt{p}, 1), \texttt{in\_int}(\texttt{q}, 1)\}0.\\ \texttt{mmr}(\texttt{p}, 1)\texttt{:-}\texttt{int}(1), \texttt{mmr}(\texttt{r}, 1), \texttt{in\_int}(\texttt{p}, 1), 1\{\texttt{in\_int}(\texttt{p}), \texttt{in\_int}(\texttt{q})\}1.\\ \texttt{mmr}(\texttt{q}, 1)\texttt{:-}\texttt{int}(1), \texttt{mmr}(\texttt{r}, 1), \texttt{in\_int}(\texttt{q}, 1), 1\{\texttt{in\_int}(\texttt{p}), \texttt{in\_int}(\texttt{q})\}1.\\ \texttt{mmr}(\bot, 1)\texttt{:-}\texttt{int}(1), \texttt{mmr}(\texttt{r}, 1), \texttt{ not in\_int}(\texttt{p}, 1).\\ \texttt{mmr}(\texttt{r}, 1)\texttt{:-}\texttt{int}(1).\end{array}\right\}$$

This program can be combined with a (reified) interpretation in order to compute the minimal model of the reduct of the interpretation with respect to P. Consider the interpretation $I = \{\texttt{p, q, r}\}$. The program $\mathcal{M}_{reduct}(P, 1) \cup \{\texttt{int}(1)\} \cup \mathcal{R}(I, \texttt{in\_int}, 1)$ has a single answer set $\{\texttt{int}(1)$, $\texttt{in\_int}(\texttt{p}, 1)$, $\texttt{in\_int}(\texttt{q}, 1)$, $\texttt{in\_int}(\texttt{r}, 1)$, $\texttt{mmr}(\texttt{r}, 1)$, $\texttt{mmr}(\bot, 1)\}$. This demonstrates that the minimal model of $P^I = \{\texttt{r}, \bot\}$, and so I is not an answer set of P.

We now introduce a program $\mathcal{M}_{vio}$, which enables us to take a set of interpretations, a set of examples and a hypothesis H and check whether each interpretation is an answer set of $B \cup H \cup e_{ctx}$ (for the example e). We consider a set S of tuples of the form $\langle I, e, \texttt{int\_id}\rangle$, where we wish to check whether I is an accepting answer set of e with respect to some hypothesis (the $\texttt{int\_id}$'s are unique terms in each tuple, and are used in the meta representation). Meta-program 7.2 formalises the meta-level program $\mathcal{M}_{vio}$, which consists of 4 components. The first three components ensure that, for each interpretation, $\mathcal{M}_{vio}$ contains $\mathcal{M}_{reduct}(B \cup S_M \cup e_{ctx}, \texttt{int\_id})$, where each rule in S_M is appended with the usual $\texttt{in\_h}$ atom, and $\langle I, e, \texttt{int\_id}\rangle \in S$. Given a hypothesis H, the final component ensures that for each tuple $\langle I, e, \texttt{int\_id}\rangle \in S$, $\texttt{not\_as}(\texttt{int\_id})$ is in the unique meta-level answer set of $\mathcal{M}_{vio} \cup \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}). \mid h \in H\}$ if and only if $I \notin AS(B \cup H \cup e_{ctx})$.

Meta-program 7.2 ($\mathcal{M}_{vio}(T, S)$). Let T be the $ILP^{context}_{LOAS}$ task $\langle B, S_M, E\rangle$ and let S be a set of tuples $\langle I, e, \texttt{int\_id}\rangle$, where I is an interpretation, e is a CDPI and `int_id` is a ground term. $\mathcal{M}_{vio}(T, S)$ is the program consisting of the following rules:

1. For each $\langle I, e, \texttt{int\_id}\rangle \in S$, the facts:
   `int(int_id).`
   $\{\texttt{in\_int(atom, int\_id).} \mid \texttt{atom} \in I\}$
   and the rules: $\mathcal{M}_{reduct}(e_{ctx}, \texttt{int\_id})$
2. $\mathcal{M}_{reduct}(B, \texttt{INT\_ID})$
3. For each rule $R \in S_M$:
   $\mathcal{A}(\mathcal{M}_{reduct}(R, \texttt{INT\_ID}), \texttt{in\_h}(\texttt{R}_{\texttt{id}}))$
4. `not_as(INT_ID):-int(INT_ID),in_int(ATOM,INT_ID), not mmr(ATOM,INT_ID).`
   `not_as(INT_ID):-int(INT_ID), not in_int(ATOM,INT_ID),mmr(ATOM,INT_ID).`

Theorem 7.7 demonstrates that given a set of tuples of the form $\langle I, e, \texttt{int\_id}\rangle$ as described in Meta-program 7.2, $\mathcal{M}_{vio}$ can be used to compute the minimal model of the reduct of each I wrt $B \cup H \cup e_{ctx}$ (for the corresponding e), and that the unique meta-level answer set will contain `not_as(int_id)` if and only if $I \notin AS(B \cup H \cup e_{ctx})$. For a negative example e, this can be used to check whether I is a violating interpretation of H wrt e.

Theorem 7.7. *(proof on page 299)*

Let T be an $ILP^{context}_{LOAS}$ task with hypothesis space S_M and let $H \subseteq S_M$. Let S be a set of tuples of the form $\langle I, e, \texttt{int\_id}\rangle$, where I is an interpretation, e is a CDPI and `int_id` *is a unique ground term.*

$\mathcal{M}_{vio}(T, S) \cup \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}) \mid h \in H\}$ has exactly one answer set, which consists of:

- *The atom* $\texttt{in\_h}(\texttt{h}_{\texttt{id}})$ *for each* $h \in H$
- *For each $\langle I, e, \texttt{int\_id}\rangle \in S$, the atoms:*
  - `int(int_id)`
  - `in_int(a, int_id)` *for each* $\texttt{a} \in I$
  - `mmr(a, int_id)` *for each* $\texttt{a} \in M(ground^{rel}(B \cup H \cup e_{ctx})^I)$
  - *If $I \notin AS(B \cup H \cup e_{ctx})$, the atom* `not_as(int_id)`

Representing Violating Pairs

Given a set of pairs of the form $\langle VP, o\rangle$, where VP is a violating pair of some hypothesis with respect to o, Meta-program 7.3 formalises a meta-level program which can be used, in conjunction with the

$\mathcal{M}_{vio}$ program in order to determine whether or not each VP is a violating pair of another hypothesis wrt o. Meta-program 7.3 relies on a meta-level encoding of a weak constraint, similar to the $\mathcal{R}_{weak}$ representation used in ILASP1. Definition 7.4 formalises the $\mathcal{R}^{vio}_{weak}$ representation.

Definition 7.4. Let `int_id` be a term. Given a weak constraint, R of the form $:\sim$ $\mathtt{b_1, \ldots, b_k}$, $\mathtt{not\ c_1, \ldots, not\ c_m}$.$[\mathtt{wt@lev, t_1, \ldots, t_n}]$, $\mathcal{R}^{vio}_{weak}(R, \mathtt{int\_id})$ is the rule:

$\mathtt{vio\_w(wt, lev, args(t_1, \ldots, t_n), int\_id)}$:-
  $\mathtt{int(int\_id), in\_int(b_1, int\_id), \ldots, in\_int(b_k, int\_id),}$
  $\mathtt{not\ in\_int(c_1, int\_id), \ldots, not\ in\_int(c_m, int\_id).}$

For a set of weak constraints W, $\mathcal{R}^{vio}_{weak}(W, \mathtt{int\_id}) = \{\mathcal{R}^{vio}_{weak}(R, \mathtt{int\_id}) \mid R \in W\}$.

Meta-program 7.3 ($\mathcal{M}_{vp}(T, S)$)**.** Let T be the task $\langle B, S_M, E\rangle$, and let S be a set of tuples of the form $\langle \mathtt{p_{id}}, \mathtt{a_1}, \mathtt{a_2}, op\rangle$, where $\mathtt{p_{id}}$, $\mathtt{a_1}$ and $\mathtt{a_2}$ are ground terms and $op \in \{<, \leq, >, \geq, =, \neq\}$. $\mathcal{M}_{vp}(T, S)$ is the program:

1. $\mathcal{R}^{vio}_{weak}(weak(B), \mathtt{a_1}) \cup \mathcal{R}^{vio}_{weak}(weak(B), \mathtt{a_2})$
2. For each $W \in weak(S_M)$: $\mathcal{A}(\mathcal{R}^{vio}_{weak}(W, \mathtt{a_1}), \mathtt{in\_h(W_{id})}) \cup \mathcal{A}(\mathcal{R}^{vio}_{weak}(W, \mathtt{a_2}), \mathtt{in\_h(W_{id})})$.
3. For each priority level `lev` that occurs in at least one weak constraint in $B \cup S_M$, the rules:
$$\left\{\begin{array}{l} \mathtt{v\_dom\_lv(a_1, a_2, lev)} \text{ :- } \mathtt{\#sum\{vio\_w(W, lev, A, a_1) = W, vio\_w(W, lev, A, a_2) = -W\} < 0.} \\ \mathtt{v\_dom\_lv(a_2, a_1, lev)} \text{ :- } \mathtt{\#sum\{vio\_w(W, lev, A, a_1) = W, vio\_w(W, lev, A, a_2) = -W\} > 0.} \end{array}\right\}$$
4. For each priority level `lev` that occurs in at least one weak constraint in $B \cup S_M$, the rules:
$$\left\{\begin{array}{lll} \mathtt{v\_dom(a_1, a_2)} & \text{:-} & \mathtt{v\_dom\_lv(a_1, a_2, lev),} \\ & & \mathtt{not\ v\_dom\_lv(a_2, a_1, l_1), \ldots, not\ v\_dom\_lv(a_2, a_1, l_n).} \\ \mathtt{v\_dom(a_2, a_1)} & \text{:-} & \mathtt{v\_dom\_lv(a_2, a_1, lev),} \\ & & \mathtt{not\ v\_dom\_lv(a_1, a_2, l_1), \ldots, not\ v\_dom\_lv(a_1, a_2, l_n).} \end{array}\right\}$$
(where $\{\mathtt{l_1, \ldots, l_n}\}$ are the levels that occur in $B \cup S_M$ that are higher than `lev`)
5. If op is $<$, the rule:
$\mathtt{vp\_not\_resp(p_{id})}$:- $\mathtt{not\ v\_dom(a_1, a_2),\ not\ not\_as(a_1),\ not\ not\_as(a_2).}$

   If op is $>$ the rule:
$\mathtt{vp\_not\_resp(p_{id})}$:- $\mathtt{not\ v\_dom(a_2, a_1),\ not\ not\_as(a_1),\ not\ not\_as(a_2).}$

   If op is $\neq$, the rule:
$\mathtt{vp\_not\_resp(p_{id})}$:- $\mathtt{not\ v\_dom(a_1, a_2),\ not\ v\_dom(a_2, a_1),}$
  $\mathtt{not\ not\_as(a_1),\ not\ not\_as(a_2).}$

   If op is $=$, the rules:
$\mathtt{vp\_not\_resp(p_{id})}$:- $\mathtt{v\_dom(a_1, a_2),\ not\ not\_as(a_1),\ not\ not\_as(a_2).}$
$\mathtt{vp\_not\_resp(p_{id})}$:- $\mathtt{v\_dom(a_2, a_1),\ not\ not\_as(a_1),\ not\ not\_as(a_2).}$

If *op* is $\leq$, the rule:
$\mathtt{vp\_not\_resp(p_{id})\text{:-}\,v\_dom(a_2, a_1),\ not\ not\_as(a_1),\ not\ not\_as(a_2).}$

If *op* is $\geq$, the rule:
$\mathtt{vp\_not\_resp(p_{id})\text{:-}\,v\_dom(a_1, a_2),\ not\ not\_as(a_1),\ not\ not\_as(a_2).}$

Theorem 7.8 shows that given a hypothesis H and a set of violating reasons VR (which were computed with other hypotheses and are not necessarily violating reasons for H), we can determine whether each $vr \in VR$ is indeed a violating reason of H by using the $\mathcal{M}_{vio}$ and $\mathcal{M}_{vp}$ programs. Violating interpretations can be checked using the $\mathcal{M}_{vio}$ program, as shown in Theorem 7.7, and violating pairs can be checked using a combination of both $\mathcal{M}_{vio}$ and $\mathcal{M}_{vp}$. Theorem 7.8 shows that $\mathtt{vp\_not\_resp(p_{id})}$ is in the unique answer set of the combined meta-level program if and only if the pair of interpretations represented by $\mathtt{p_{id}}$ is a violating pair of interpretations of H.

Theorem 7.8. *(proof on page 302)*

Let T be the task $\langle B, S_M, E\rangle$, let S_1 be a list of tuples $[\langle I^1, e^1, \mathtt{int\_id^1}\rangle, \ldots, \langle I^n, e^n, \mathtt{int\_id^n}\rangle]$, where each I^i is an interpretation, each e^i is a CDPI st I^i extends e^i and each $\mathtt{int\_id^i}$ is a unique ground term, and let S_2 be a set of tuples of the form $\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle$, where $\mathtt{p_{id}}$ is a unique ground term, $i, j \in [1, n]$ and $op \in \{<, \leq, >, \geq, =, \neq\}$. Let $H \subseteq S_M$.

The program $\{\mathtt{in\_h(h_{id}).} \mid h \in H\} \cup \mathcal{M}_{vio}(T, S_1) \cup \mathcal{M}_{vp}(T, S_2)$ has exactly one answer set, consisting of:

- *All atoms in the unique answer set of $\{\mathtt{in\_h(h_{id}).} \mid h \in H\} \cup \mathcal{M}_{vio}(T, S_1)$.*
- *For each $\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2$:*
  - $\mathtt{vio\_w(wt, lv, args(t_1, \ldots, t_n), int\_id^i)}$ *(resp.* $\mathtt{vio\_w(wt, lv, args(t_1, \ldots, t_n), int\_id^j)}$*) for each weak constraint $W \in ground(B \cup H)$, with tail $[\mathtt{wt@lv, t_1, \ldots, t_n}]$ such that $body(W)$ is satisfied by I^i (resp. I^j).*
  - $\mathtt{v\_dom\_lv(int\_id^i, int\_id^j, lev)}$ *(resp.* $\mathtt{v\_dom\_lv(int\_id^j, int\_id^i, lev)}$*) for each level* $\mathtt{lev}$ *that occurs in $B \cup S_M$ such that $(B \cup H)^{I^i}_{lev} < (B \cup H)^{I^j}_{lev}$ (resp. $(B \cup H)^{I^i}_{lev} > (B \cup H)^{I^j}_{lev}$).*
  - $\mathtt{v\_dom(int\_id^i, int\_id^j)}$ *(resp.* $\mathtt{v\_dom(int\_id^j, int\_id^i)}$*) if $I^i \succ_{B\cup H} I^j$ (resp. $I^j \succ_{B\cup H} I^i$).*
  - $\mathtt{vp\_not\_resp(p_{id})}$ *if I^i is an accepting answer set of e^i wrt $B \cup H$, I^j is an accepting answer set of e^j wrt $B \cup H$ and $\langle I^i, I^j, op\rangle \notin ord(B \cup H, AS(B \cup H \cup e^i_{ctx}) \cup AS(B \cup H \cup e^j_{ctx}))$*

7.1.3 The ILASP2 Meta-Level Program

We have now presented ASP programs which can be added to our previous meta-level program $\mathcal{M}(T)$ to check whether or not a hypothesis is violating for a reason that we have already identified.

In the case that the meta-level answer set found in any iteration does not contain a violating reason, we need to be sure that there is no other answer set of the meta-level program corresponding to the same hypothesis that does contain a violating reason. We can achieve this with a weak constraint, indicating that we prefer meta-level answer sets that contain a violating reason to those that do not contain a violating reason. This means that any meta-level answer set containing a violating reason will be returned before any that does not, and so if an optimal meta-level answer set does not contain a violating reason, then the hypothesis represented by this answer set must not be violating.

One possible preference ordering over the meta-level answer sets would be the one represented by the weak constraints $:\sim \texttt{in\_h}(\texttt{h}_{\texttt{id}}).[|\texttt{h}|@1, \texttt{in\_h}(\texttt{h}_{\texttt{id}})]$, for each $h \in S_M$ and $:\sim$ `not violating`$.[1@2]$. Given two answer sets that either both contain `violating` or that both do not contain `violating`, the answer set representing the shortest hypothesis is preferred. While this preference ordering is valid, in general it leads to an inefficient algorithm, as every violating hypothesis must be ruled out before the inductive solution is found. If the optimal inductive solution is relatively short, then much time could be wasted ruling out longer violating hypotheses. The preference ordering represented by the weak constraints in Meta-program 7.4 only prefers a meta-level answer set containing `violating` over one that does not if the hypotheses represented by both meta-level answer sets are the same length – the score of any interpretation at the only priority level in the program is equal to $2 \times |\mathcal{M}^{-1}_{in\_h}(A)|$ if `violating` $\in A$ and $2 \times |\mathcal{M}^{-1}_{in\_h}(A)| + 1$ if `violating` $\notin A$, so shorter hypotheses are always preferred to longer ones, but an answer set that contains `violating` is preferred to one that does not contain `violating` if both answer sets represent hypotheses of the same length.

Meta-program 7.4 ($\mathcal{M}_{violating\_ord}(T)$)**.** Let T be the $ILP^{context}_{LOAS}$ task $\langle B, S_M, E\rangle$. $\mathcal{M}_{violating\_ord}(T)$ is the program containing:

- For each rule $r \in S_M$, the weak constraint:
  $:\sim \texttt{in\_h}(\texttt{r}_{\texttt{id}}).[2 \times |\texttt{r}|@1, \texttt{in\_h}(\texttt{r}_{\texttt{id}})]$
- The weak constraint:
  $:\sim$ `not violating`$.[1@1, \texttt{violating}]$

We can now present the full meta-level program used by ILASP2. Meta-program 7.5 formalises $\mathcal{M}_{ILASP2}$, which consists of 4 parts. The first is the $\mathcal{M}(T)$ program of ILASP1, which ensures that any meta-level answer set corresponds to a positive hypothesis and also identifies violating reasons. The next two components rule out violating reasons which have already been found. Finally, the fourth component gives a prefence ordering to the meta-level answer sets. The definition assumes that each interpretation I has its own unique identifier, written $\texttt{I}_{\texttt{id}}$, and that each violating pair VP has its own unique identifier, written $\texttt{VP}_{\texttt{id}}$.

Meta-program 7.5 ($\mathcal{M}_{ILASP2}(T, VR)$)**.** Let T be the $ILP^{context}_{LOAS}$ task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$ and let VR be a set of violating reasons. $\mathcal{M}_{ILASP2}(T, VR)$ is the program consisting of the following components:

- $\mathcal{M}(T) \cup \{\texttt{0\{check\_violating\}1.}\}$
- $\mathcal{M}_{vio}(T, S_1) \cup \left\{ \texttt{:- not not\_as(I}_{\texttt{id}}\texttt{).} \;\middle|\; \begin{array}{l} \langle I, e\rangle \in VR, \\ e \in E^- \end{array} \right\}$, where S_1 is the set of tuples $\langle I, e, I_{id}\rangle$, where I is an interpretation such that $\langle I, e\rangle \in VR$, or $\exists \langle\langle I^1, I^2\rangle, o\rangle \in VR$ such that $o_{eg1} = e$ or $o_{eg2} = e$ and $I \in \{I^1, I^2\}$.
- $\mathcal{M}_{vp}(T, S_2) \cup \left\{ \texttt{:-vp\_not\_resp(VP}_{\texttt{id}}\texttt{).} \;\middle|\; \begin{array}{l} \langle VP, o\rangle \in VR, \\ o \in O^c \end{array} \right\}$, where S_2 is the set of tuples $\langle \texttt{VP}_{\texttt{id}}, \texttt{I}^1_{\texttt{id}}, \texttt{I}^2_{\texttt{id}}, o_{op}\rangle$, where $\langle VP, o\rangle \in VR$ and $VP = \langle I^1, I^2\rangle$,
- $\mathcal{M}_{violating\_ord}(T)$

Theorem 7.9 shows three useful properties about the answer sets of the $\mathcal{M}_{ILASP2}(T, VR)$ program (given any task T and set of violating reasons VR). The first is that the set of hypotheses that are represented by at least one answer set of the meta program are those positive hypotheses of T that do not violate any of the reasons in VR. Secondly, for any optimal meta-level answer set that contains `violating`, $extractVR(A)$ returns a violating reason of the hypothesis represented by A. Finally, if no optimal meta-level answer set contains `violating` then the set of optimal remaining hypotheses can be extracted from the optimal meta-level answer sets. Furthermore, as none of these is violating, they are guaranteed to be the optimal inductive solutions. This means that if the while loop in the ILASP2 algorithm terminates then the returned hypotheses must be the set of optimal inductive solutions.

Theorem 7.9. *(proof on page 304) Given an $ILP^{context}_{LOAS}$ task and a set of violating reasons VR, let AS be the set of optimal answer sets of $\mathcal{M}_{ILASP2}(T, VR)$*

1. *For any hypothesis H, $\exists A \in AS(\mathcal{M}_{ILASP2}(T, VR))$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ if and only if $H \in \mathcal{P}(T)$ and $\forall vr \in VR$, vr is not a violating reason of H.*
2. *For any $A \in AS$ such that `violating` $\in A$, $extractVR(A)$ is a violating reason of $\mathcal{M}^{-1}_{in\_h}(A)$.*
3. *If no $A \in AS$ contains `violating`, then the set of optimal remaining hypotheses (none of which is violating) is exactly equal to the set $\{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS\}$.*

Before proving termination of the ILASP2 algorithm, we must first show that each call to the ASP solver terminates. In theory, this should be the case, provided the relevant groundings of the programs being solved are finite. Proposition 7.10, shows that this is the case for the meta-level programs used by ILASP2.

Proposition 7.10. *(proof on page 305)* Let T be any well-defined $ILP^{context}_{LOAS}$ task, and VR be a finite set of violating reasons. $ground^{rel}(\mathcal{M}_{ILASP2}(T, VR))$ is finite.

We can now prove the main results of the ILASP2 algorithm. Theorem 7.11 shows that for any well defined $ILP^{context}_{LOAS}$ task, ILASP2 is guaranteed to terminate, and is also sound and complete with respect to the optimal inductive solutions of the task.

Theorem 7.11. *(proof on page 306) Let T be any well-defined $ILP^{context}_{LOAS}$ task.*

1. *$ILASP2(T)$ terminates in a finite time.*
2. *$ILASP2(T) = {}^*ILP^{context}_{LOAS}(T)$*

7.2 ILASP2i – Incremental Mode

The ILASP2 algorithm presented in Section 7.1 addresses one of the scalability issues of ILASP1, through the use of violating reasons, and it performs significantly better on tasks with many violating hypotheses. However, ILASP2 still scales poorly with respect to the number of examples. This is because the relevant grounding of the $\mathcal{M}(T)$ program, which is contained in $\mathcal{M}_{ILASP2}$, is still roughly proportional in size to $|E^+| + 2 \times |O^b| + 2$.

ILASP1 and ILASP2 are examples of *batch learners*, which consider all examples simultaneously. Some older ILP systems, such as ALEPH [Sri01], Progol [Mug95] and HAIL [RBR03], incrementally consider each positive example in turn, employing a *cover loop.* The idea behind a cover loop is that the algorithm starts with an empty hypothesis H, and in each iteration adds new rules to H such that a single positive example e is covered, and none of the negative examples are covered. Example 7.12 demonstrates why cover loops do not work in a non-monotonic setting, even for the task of brave induction.

Example 7.12. *Consider the brave induction task $\langle B, S_M, \langle E^+, E^- \rangle\rangle$, where:*

$$B = \left\{ \begin{array}{l} \texttt{p(1).} \\ \texttt{p(2).} \\ \texttt{q(2).} \\ \texttt{r(2).} \end{array} \right\} \qquad S_M = \left\{ \begin{array}{ll} \texttt{h}_1: & \texttt{q(1).} \\ \texttt{h}_2: & \texttt{s(X):-p(X).} \\ \texttt{h}_3: & \texttt{s(X):-p(X), not q(X).} \\ \texttt{h}_4: & \texttt{s(X):-p(X), not r(X).} \end{array} \right\}$$

$$E^+ = \{\texttt{s(1), q(1)}\} \qquad E^- = \{\texttt{s(2)}\}$$

If we were to employ a cover loop strategy, we would start with our hypothesis as $H = \emptyset$. We would then consider the first positive example s(1)*, and attempt to find a set of rules H' in S_M such that $B \cup \emptyset \cup H'$ has at least one answer set that contains* s(1) *and does not contain* s(2)*. We could choose the rule* $\texttt{h}_3$*.*

So in the next iteration, we would start with the hypothesis $\{\texttt{h}_3\}$. We would then search for an H' such that $B \cup \{\texttt{h}_3\} \cup H'$ has at least one answer set that contains q(1) *and does not contain* s(2)*. The only rule that we could possibly add to cover* q(1)*, is* $\texttt{h}_1$*.*

So after the second iteration, our hypothesis would be $\{\mathtt{h_1}, \mathtt{h_3}\}$*, but this does not cover the first example. The problem is that due to the non-monotonicity of ASP, even though our partial hypothesis* $\{\mathtt{h_3}\}$ *did cover the example* $\mathtt{p(1)}$*,* $\mathtt{p(1)}$ *has become uncovered by the addition of* $\mathtt{h_1}$ *to the hypothesis.*

Under the answer set semantics, most learners are batch learners due to the non-monotonicity. In fact, it is worth noting that, in particular, although the HAIL algorithm for learning definite clauses employs a cover loop, the later XHAIL algorithm is a batch learner as it learns non-monotonic programs [Ray09]. In this section we introduce our ILASP2i algorithm, which uses a "middle ground" between batch learning and a cover loop.

ILASP2i iteratively computes a "*relevant*" subset of the examples, and in each iteration searches for the smallest hypothesis H that covers every example in this relevant subset. ILASP2i then searches for an example which is not covered by H. If no such example exists, then H is an optimal inductive solution. If there is such an example, it is added to the set of relevant examples, and a new iteration is started. Definition 7.5 formalises the notion of a relevant example.

Definition 7.5. Consider an $ILP^{context}_{LOAS}$ task $T = \langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$ and a hypothesis $H \subseteq S_M$. An example ex is *relevant* to H given T if $ex \in E^+ \cup E^- \cup O^b \cup O^c$ and H does not cover ex.

In each iteration, the search for a hypothesis that covers the relevant examples is performed by the batch learner ILASP2. It is for this reason that we consider ILASP2i to be a "middle ground" between a cover loop and a batch learner. In each iteration ILASP2i does consider multiple examples, and employs a batch learner to solve them; however, in practice the final set of relevant examples is often significantly smaller than the entire set of examples.

7.2.1 Finding the Relevant Examples

We now introduce a function $findRelevantExamples(\langle B, S_M, E\rangle, H)$, which returns the set of examples in E that are not covered by H. It works by using the $\mathcal{M}_1$ program defined in Chapter 6 augmented with some additional components to check the coverage of each example. This meta-level program is formalised in Meta-program 7.6. The important thing to note about $\mathcal{M}_{fre}$ is that it only considers two object-level answer sets in each meta-level answer set, represented by $\mathtt{as(1)}$ and $\mathtt{as(2)}$, meaning that the grounding of the program is much smaller than the main ILASP2 meta-level program. One very useful property of $\mathcal{M}_{fre}$ is that for each $e \in E^+ \cup E^-$, $\mathtt{cov(e_{id}, 1)}$ is bravely entailed by the meta program if and only if $B \cup H$ accepts e, meaning that we can use $\mathcal{M}_{fre}$ to check which examples in E^+ and E^- are covered; similarly, for each $o \in O^b \cup \{inverse(o') \mid o' \in O^c\}$, $\mathtt{ord\_respected(o_{id}, 1, 2)}$ is bravely entailed if and only if $B \cup H$ accepts o. Note that we check whether the inverse of each cautious ordering is accepted because this is exactly the negation of the coverage condition for cautious orderings – i.e. those cautious orderings whose inverse is accepted are exactly those cautious orderings that are *not* covered.

Meta-program 7.6 **($\mathcal{M}_{fre}(T, H)$).** Let T be the $ILP^{context}_{LOAS}$ task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$, and H be a hypothesis. $\mathcal{M}_{fre}(T, H)$ is the program consisting of the following components:

- $\mathcal{M}_1(T) \cup \{\texttt{in\_h(h}_{\texttt{id}}\texttt{).} \mid h \in H\} \cup \{\texttt{as(1). as(2).}\}$
- $\{check(e, \texttt{1}) \cup check(e, \texttt{2}) \mid e \in E^+ \cup E^-\}$
- $\{check\_ord(T, o, \texttt{1}, \texttt{2}) \mid o \in O^b\}$
- $\{check\_ord(T, inverse(o), \texttt{1}, \texttt{2}) \mid o \in O^c\}$

Algorithm 7.3 findRelevantExamples

```
1: procedure findRelevantExamples($\langle B, S_M, E\rangle, H$)
2:     $S = BraveConsequences(\mathcal{M}_{fre}(T, H))$;
3:     $CDPIs = \{e \in E^+ \cup E^- \mid \texttt{cov(e}_{\texttt{id}}\texttt{, 1)} \in S\}$;
4:     $CDOEs = \{o \in O^b \cup O^c \mid \texttt{ord\_respected(o}_{\texttt{id}}\texttt{, 1, 2)} \in S\}$;
5:     return $(E^+ \backslash CDPIs) \cup (E^- \cap CDPIs) \cup (O^b \backslash CDOEs) \cup (O^c \cap CDOEs)$;
6: end procedure
```

The ILASP2i algorithm uses the function $findRelevantExample(\langle B, S_M, E\rangle, H)$, which returns an example in E that is not covered by H, or returns `nil` if no such example exists. The function $findRelevantExample(\langle B, S_M, E\rangle, H)$ calls the $findRelevantExamples(\langle B, S_M, E\rangle, H)$ procedure presented in Algorithm 7.3 and returns the first example in the set returned by $findRelevantExamples$ set if one exists, and `nil` otherwise.

Proposition 7.13. *(proof on page 307)* Let T be any well-defined $ILP^{context}_{LOAS}$ task with a hypothesis space S_M, and let H be any hypothesis $H \subseteq S_M$. $ground^{rel}(\mathcal{M}_{fre}(T, H))$ is finite.

Corollary 7.14. $findRelevantExample(T, H)$ terminates for any well-defined task T and finite hypothesis H.

Theorem 7.15. *(proof on page 307) Let T be an $ILP^{context}_{LOAS}$ task, and let H be a hypothesis.*

1. *If $H \in ILP^{context}_{LOAS}(T)$ then $findRelevantExample(T, H)$ returns* `nil`
2. *If $H \notin ILP^{context}_{LOAS}(T)$ then $findRelevantExample(T, H)$ returns an example that is relevant to H given T.*

7.2.2 The ILASP2i Algorithm

The intuition of ILASP2i (Algorithm 7.4) is that we start with an empty set of *relevant* examples and an empty hypothesis. At each step of the search we look for an example which is relevant to our current hypothesis (i.e. an example that $B \cup H$ does not cover) using the $findRelevantExample$ method. If

Algorithm 7.4 ILASP2i

```
1: procedure ILASP2i($\langle B, S_M, E\rangle$)
2:     $Relevant = \langle\emptyset, \emptyset, \emptyset, \emptyset\rangle$;  $H = \emptyset$;
3:     $re = findRelevantExample(\langle B, S_M, E\rangle, H)$;
4:     while $re \neq$ nil
5:         $Relevant << re$;
6:         $H = ILASP2(\langle B, S_M, Relevant\rangle)$;
7:         if $H ==$ nil
8:             return UNSATISFIABLE;
9:         else
10:            $re = findRelevantExample(\langle B, S_M, E\rangle, H)$;
11:        end if
12:    end while
13:    return $H$;
14: end procedure
```

findRelevantExample returns `nil`, then no such example exists, and so we return our current hypothesis as it is guaranteed to be an optimal inductive solution of the task. If *findRelevantExample* returns an example, then we add the example to our relevant set of examples[1] and use ILASP2 to compute a new hypothesis. If at any stage in the algorithm ILASP2 returns `nil` then $\langle B, S_M, Relevant\rangle$ has no solutions and, as $ILP_{LOAS}^{context}(\langle B, S_M, E\rangle)$ is a subset of $ILP_{LOAS}^{context}(\langle B, S_M, Relevant\rangle)$, the task $\langle B, S_M, E\rangle$ is unsatisfiable.

Increased Performance Over ILASP2

We will show in Chapter 8 that ILASP2i can significantly outperform ILASP2 on tasks with large numbers of examples. The reason for this is that when tasks have many examples, there can be some subsets of examples that are covered (and not covered) by the same hypotheses. For any of these sets of similar examples, ILASP2i will only add one example to the set of relevant examples, as once one of the examples is considered relevant, the whole set will be covered by the hypotheses in any future iterations of the computation. This means that the set of relevant examples can be significantly smaller than the entire set of examples, which in turn leads to the size of the grounding of the meta-level program in each call to ILASP2 being significantly smaller than the meta-level program which ILASP2 would generate for the full set of examples.

It should be noted that in the worst case the final set of relevant examples is equal to the entire set of examples. In this case, ILASP2i is slower than ILASP2. In real settings, however, where examples would not be carefully constructed, there is likely to be overlap between examples, so the relevant set will be much smaller than the whole set. Theorem 7.16 shows that ILASP2i terminates for any well-defined $ILP_{LOAS}^{context}$ task.

[1]The notation <<, in line 5 of algorithm 7.4, means to add the relevant example *re* to the correct set in *Relevant* (the first set if it is a positive example etc).

Theorem 7.16. *(proof on page 308) ILASP2i terminates for any well-defined $ILP_{LOAS}^{context}$ task.*

Note that although ILASP2i is sound, it is complete only in the sense that it always returns an optimal solution if one exists (rather than returning the full set).

Theorem 7.17. *(proof on page 308) Let T be a well-defined $ILP_{LOAS}^{context}$ task.*

1. *If T is satisfiable, then $ILASP2i(T)$ returns an optimal inductive solution of T.*
2. *If T is unsatisfiable, then $ILASP2i(T)$ returns* `UNSATISFIABLE`.

7.3 Related Work

7.3.1 Violating Reasons

The notion of a violating reason allows us to rule out an entire class of hypotheses at once. In some ways they perform a similar function to the *nogoods* or *learned constraints* used in many SAT [LMS03] and ASP [GKNS07, GKK^{+}11, ADF^{+}13] solvers.

In clasp [GKK^{+}11], nogoods are sets of literals which are recorded during the search for an answer set, such that for each set N, it is known that there is no answer set that satisfies every literal in N. Thus, nogoods allow clasp to rule out any interpretation that entails the nogood. Although violating reasons have a very different structure, they are also used to rule out solutions (hypotheses which are violating for the same reason). Recording nogoods is also referred to as *clause learning*, but we avoid this term, as it is very different to ILP, which aims at generalising examples.

7.3.2 The Relationship Between ILASP2i and Other Incremental Techniques

As discussed earlier in the chapter, the standard method for solving an ILP task incrementally is to use a *cover loop*. ILASP2i's method of using relevant examples can essentially be thought of as a non-monotonic version of the cover loop. There are three main differences:

1. In cover loop approaches, in each iteration a previous hypothesis H is extended with extra rules, giving a new hypothesis H' that contains H. In ILASP2i, a completely new hypothesis is learned in each iteration. This is necessary to guarantee that optimal hypotheses are computed. Many cover loop approaches make no guarantee about the optimality of the final hypothesis.
2. In ILASP2i, the set of relevant examples is maintained and used in every iteration, whereas in cover loop approaches, only one example is considered per iteration.

3. In cover loop approaches, once an example has been processed, even if it did not cause any changes to the current hypothesis, it is guaranteed to be covered by any future hypothesis and so it is not checked again. In ILASP2i, this is not the case. This is why ILASP2i calls the *findRelevantExamples* method on the full set of examples, even if some were previously known to be covered.

There are two incremental approaches to ILP under the answer set semantics. ILED [KAP15], is an incremental version of the XHAIL algorithm, which is specifically targeted at learning Event Calculus theories. ILED's examples are split into *windows*, and ILED incrementally computes a hypothesis through Theory Revision [Wro96] to cover the examples. In an arbitrary iteration, ILED revises the previous hypothesis H (which is guaranteed to cover the first n examples), to ensure that it covers the first $n + 1$ examples. As the final hypothesis is the outcome of the series of revisions, although each revision may have been optimal, ILED may terminate with a sub-optimal inductive solution. In contrast, ILASP2i will always terminate with an optimal inductive solution if one exists.

The other incremental ILP system under the answer set semantics is RASPAL [ACBR13, Ath15], which uses an ASPAL-like approach to iteratively revise a hypothesis until it is an optimal inductive solution of a task. RASPAL's incremental approach is successful as it often only needs to consider small parts of the hypothesis space, rather than the full hypothesis space. Unlike ILED and ILASP2i, however, RASPAL considers the full set of examples in every iteration. For RASPAL, this is less of an issue than for ILASP2i as, similarly to ASPAL, RASPAL uses a single rule to encode the full set of examples. For ILASP2i on the other hand, the size of the full meta representation is proportional to the number of examples.

Outside of the ILP community the L^* algorithm for learning deterministic finite automata (DFA) was presented in [Ang87]. This procedure relies on a *learner* and a *teacher*. In each iteration of the L^* algorithm, the learner queries the teacher as to whether the DFA it has learned so far is correct. If it is, L^* terminates, if not, the teacher presents a counter-example and the learner attempts to find a DFA that is consistent with the set of counter examples it has seen so far. ILASP2i's set of relevant examples are very similar to L^*'s counter examples. In the case of ILASP2i, the *findRelevantExample* method could be seen as the teacher, and the call to ILASP2 could be seen as the learner.

Summary

In this chapter, we have presented two algorithms aimed at improving the scalability of the ILASP approach. In the next chapter we investigate the performance of ILASP1, ILASP2 and ILASP2i on various classes of learning task.

Chapter 8

Evaluation

In this chapter, we evaluate the three ILASP algorithms presented so far on synthetically generated data. The reason for using synthetic data is that ILASP1, ILASP2 and ILASP2i each solve learning tasks where all of the examples must be covered by the learned hypothesis. We address learning from real data, where there is likely to be noise, in Part II of this thesis.

We consider four problem settings in this chapter, designed to evaluate various aspects of the ILASP approach. The first two settings are the Hamiltonian and journey preference problems which were used to exemplify the learning from answer sets frameworks in Chapter 4. The third setting investigates the learning of preferences from partial examples, and compares the relative usefulness of brave and cautious orderings. Our final setting concerns an agent navigation problem and allows us to mix rule learning with preference learning, showing that ILASP is capable of doing both simultaneously.

The hypothesis spaces (S_M) in these experiments are defined using ILASP mode declarations. For full details of these mode declarations and their meaning, please see Appendix A. Note that a hypothesis space containing n rules allows for many more than just n possible hypotheses. Theoretically, there are 2^n possible hypotheses, but in practice ILASP imposes an upper limit on the length of hypotheses. Still, even with a hypothesis space of 100 rules, where the upper limit means that at most 5 rules can be learned, this still results in over 10 billion possible hypotheses.

8.1 Hamilton Graphs

The first problem setting we consider is learning the definition of whether a graph is Hamiltonian or not (i.e. whether it contains a Hamilton cycle). A Hamilton cycle is a cycle that visits each node of the graph exactly once. In our experiments, we investigated two versions of this problem, one (Hamilton A) with the set of possible graphs encoded in the background knowledge, and the other (Hamilton B) using context dependent examples to represent the graphs. These two representations were discussed in Chapter 4 (Example 4.3 and Example 4.10, respectively).

Hamilton A is an ILP_{LOAS} representation of the problem, where the background knowledge B consists of two choice rules:

```
0 { node(1), node(2), node(3), node(4) } 4.
0 { edge(N1, N2)} 1 :- node(N1), node(N2).
```

These two choice rules mean that the answer sets of B correspond to all possible graphs of size 0 to 4. Each example corresponds to exactly one graph, by specifying which `node` and `edge` atoms should be true in the inclusions and exclusions; for instance, the partial interpretation $\langle\{\texttt{node(1)}, \texttt{edge(1,1)}\}, \{\texttt{node(2)}, \texttt{node(3)}, \texttt{node(4)}\}\rangle$ represents a graph with one node and one edge (from the only node to itself). Positive examples correspond to Hamiltonian graphs, and negative examples correspond to non-Hamiltonian graphs; for instance, the example $\langle\{\texttt{node(1)}\}, \{\texttt{node(2)}, \texttt{node(3)}, \texttt{node(4)}, \texttt{edge(1,1)}\}\rangle$ represents a graph with a single node and no edges.

Hamilton B is the $ILP^{context}_{LOAS}$ representation of the same problem. The background knowledge is empty, and each example has a context consisting of the `node` and `edge` atoms representing a single graph (and no inclusions or exclusions). For instance, the Hamilton A positive example $\langle\{\texttt{node(1)}, \texttt{edge(1,1)}\}, \{\texttt{node(2)}, \texttt{node(3)}, \texttt{node(4)}\}\rangle$ is equivalent to the Hamilton B positive example $\langle\langle\emptyset, \emptyset\rangle, \{\texttt{node(1).}\ \texttt{edge(1,1).}\}\rangle$.

Preliminary Experiment

In the first experiment, we randomly generated 100 positive and 100 negative examples and used them to create equivalent Hamilton A and Hamilton B learning tasks. Table 8.1 shows the computation time[1] and peak memory usage of the three ILASP algorithms on these two learning tasks.

Learning		#examples				time/s			Memory/kB		
task	S_M	E^+	E^-	O^b	O^c	1	2	2i	1	2	2i
Hamilton A	104	100	100	0	0	TO	8.5	4.8	TO	9.4×10^4	1.2×10^4
Hamilton B	104	100	100	0	0	TO	31.2	3.6	TO	3.6×10^5	1.3×10^4

Table 8.1: The running times and peak memory usages of ILASP1, ILASP2 and ILASP2i on a single instance of Hamilton A and Hamilton B. TO stands for time out (30 minutes). The two learning tasks in this table are available to download from `https://www.doc.ic.ac.uk/~ml1909/ILASP/`.

ILASP1 was unable to solve either task within the time limit of 30 minutes. This was due to the many violating solutions that ILASP1 needed to rule out. ILASP2 on the other hand, due to its improved handling of violating solutions based on violating reasons, was able to solve both tasks in under a minute. ILASP2i was faster still, solving both tasks in under 5 seconds.

[1]All experiments in this thesis were run on an Ubuntu 14.04 desktop machine with a 3.4 GHz Intel® Core™ i7-3770 processor and with 16GB RAM. All meta-level ASP programs were solved using clingo 4.3 [GKK+11], unless stated otherwise.

The results of this experiment also highlight the difference in performance of the latter two algorithms on the two representations of the problem. ILASP2 performed better on Hamilton A than on Hamilton B, whereas ILASP2i performed better on Hamilton B than Hamilton A. ILASP2i's increase in performance on Hamilton B was due to the meta-representation only needing to be ground over the contexts of the relevant examples (the meta-representation was only ground over the problem domain used by the relevant contexts, rather than the full problem domain). ILASP2 on the other hand considered the contexts of the full set of examples, so there was no similar reduction in the grounding for Hamilton B. In fact, Hamilton A resulted in a more efficient meta-representation of the problem for ILASP2 as the contexts of the examples in Hamilton B covered the full problem domain.

Comparison of ILASP2 and ILASP2i with Varying Graph Sizes

To test how the size of the contexts affects the performance of the ILASP2 and ILASP2i algorithms, we ran the Hamilton A and B experiments with the maximum size of the graphs varying from 4 to 10. Each experiment was run 100 times with randomly generated sets of positive and negative examples (100 of each in each experiment). The results (in Figure 8.1) show that ILASP2i performed best in both cases – notably, on average, there was almost no difference between Hamilton A and Hamilton B at first for ILASP2i, but as the maximum graph size increased, the domain of the background knowledge in Hamilton A increased and so ILASP2i performed better on Hamilton B.

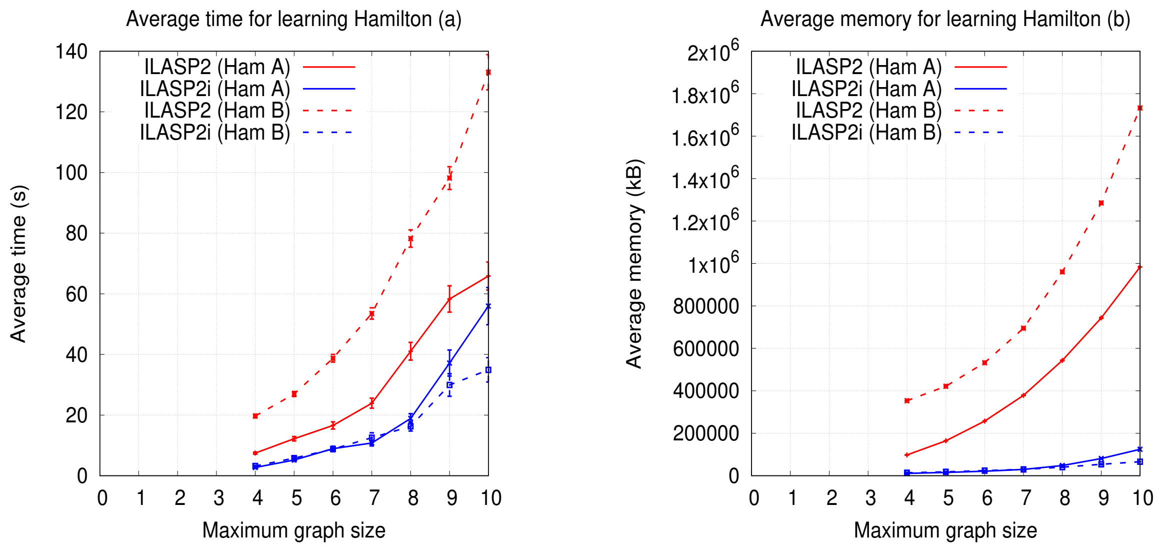


Figure 8.1: (a) the average computation time and (b) the memory usage of ILASP2, ILASP2i and ILASP2i_pt for Hamilton A and B.

8.2 Urban Mobility: User Journey Preferences

The second setting we consider is a preference learning problem, where the examples are completely specified (i.e. they are not partial). We use these experiments to investigate how the accuracy and computational performance of the ILASP algorithms varies with the number of examples, and also show that giving ordering examples with both the $<$ and $=$ operators can lead to a higher accuracy than giving ordering examples with only the $<$ operator.

Recall the problem of learning a user's preferences over alternative journeys, which was first presented in Example 4.4. In this scenario, a user makes requests to a journey planner to get from one location to another. The user then chooses a journey from the alternatives returned by the planner. A journey consists of one or more legs, in each of which the user uses a single mode of transport.

We used data generated by a simulator [PBL14] to give realistic examples of journeys. In the experiments, we used a set of journey requests from one simulated day. The attributes of journey legs in these experiments were: `mode`, which took one of the values `bus`, `car`, `walk` or `bicycle`; `distance`, which took an integer value between `1` and `20000`; and `crime_rating`. As the crime ratings were not readily available from the simulator, we used a randomly generated value between `1` and `5` for each journey leg.

In the experiments, we assumed that a user's preferences could be represented by a set of weak constraints based on the attributes of a leg. S_J denotes the set of possible weak constraints that we used in the experiments, each of which includes at most 3 literals (characterised by a mode bias, given in Appendix A). Most of these literals capture the leg's attributes, e.g., `mode(L, bus)` or `crime_rating(L, R)` (if the attribute's values range over integers this is represented by a variable, otherwise each possible value is used as a constant). For the crime rating (`crime_rating(L, R)`), we also allow comparisons of the form $R >$ `c` where `c` is an integer from 1 to 4. The weight of each weak constraint is a variable representing the distance of the leg in the body of the weak constraint, or 1 and the priority is 1, 2 or 3. One possible set of preferences is represented by the weak constraints W^*.

$$W^* = \left\{ \begin{array}{l} :\sim \texttt{leg\_mode(L, walk), leg\_crime\_rating(L, C), C} > \texttt{3.[1@3, L, C]} \\ :\sim \texttt{leg\_mode(L, car).[1@2, L]} \\ :\sim \texttt{leg\_mode(L, walk), leg\_distance(L, D).[D@1, L, D]} \end{array} \right\}$$

These preferences represent that the user's top priority is to avoid walking through areas with a high crime rating. Second, the user would like to avoid driving, and finally, the user would like to minimise the total walking distance of the journey.

We now describe how to represent the journey preferences scenario in $ILP_{LOAS}^{context}$. We assume that each journey is encoded as a set of attributes of the legs of the journey; for example the journey {`distance(leg(1), 2000)`, `distance(leg(2), 100)`, `mode(leg(1), bus)`, `mode(leg(2), walk)`} has two legs; in the first leg, the person must take a bus for 2000m and in the second, he/she must walk 100m.

Given a set of such journeys $J = \{j_1, \ldots, j_n\}$ and a partial ordering O over J, $\mathcal{M}(J, O, S_J)$ is the $ILP^{context}_{LOAS}$ task $\langle \emptyset, S_J, E^+, \emptyset, O^b, \emptyset \rangle$, where $E^+ = \{\langle\langle \emptyset, \emptyset \rangle, j_i\rangle \mid j_i \in J\}$ and $O^b = \{\langle\langle\langle \emptyset, \emptyset\rangle, j_1\rangle, \langle\langle \emptyset, \emptyset\rangle, j_2\rangle, <\rangle \mid \langle j_1, j_2\rangle \in O\}$. Each solution of $\mathcal{M}(J, O, S_J)$ is a subset of the weak constraints in S_J, representing preferences which explain the ordering of the journeys. Note that the positive examples are automatically satisfied as the (empty) background knowledge (combined with the context) already covers them. Also, as the background knowledge together with each context has exactly one answer set, the notions of brave and cautious orderings coincide; hence, we do not need cautious ordering examples for this task. Furthermore, since only weak constraints are being learned, the task also has no negative examples (a negative example would correspond to an invalid journey).

Preliminary Experiment

In the first experiment, we created a single learning task, using the set of weak constraints in W^* as a "target hypothesis". We used the simulated journeys to generate a set of 200 pairs of journeys $\langle j_1, j_2 \rangle$ such that j_1 was one of the optimal journeys, given W^*, and j_2 was an non-optimal alternative to j_1. As some journeys occurred in more than one ordering example, there were only 386, and not 400, positive examples in this experiment. The performance results for ILASP1, ILASP2 and ILASP2i are shown in Table 8.2.

Learning		#examples				time/s			Memory/kB		
task	S_M	E^+	E^-	O^b	O^c	1	2	2i	1	2	2i
Journey	117	386	0	200	0	OOM	OOM	5.4	OOM	OOM	2.9×10^4

Table 8.2: The running time and peak memory usage of ILASP1, ILASP2 and ILASP2i on a single instance of the journey preference learning problem. OOM stands for out of memory. The learning task in this table is available to download from `https://www.doc.ic.ac.uk/~ml1909/ILASP/`.

Both ILASP1 and ILASP2 ran out of memory before returning a solution. This is due to the size of the grounding of their meta-level representations being proportional to $|E^+| + 2 \times |O^b| + 2$. This means that each answer set of the meta-level programs of ILASP1 and ILASP2 represented 788 object-level answer sets. On the other hand, as ILASP2i's meta-level representation only considers the relevant examples, even in its final iteration, ILASP2i's meta-level answer sets only represented 26 object-level answer sets.

Comparison of ILASP's Accuracy with and Without Equality Ordering Examples

We next investigated how the accuracy of ILASP2i varied with the number of examples. Note that as each ILASP algorithm returns an arbitrary optimal inductive solution, their average accuracies are the same. We therefore only report accuracy results for ILASP2i, but refer to these results as the accuracy of the ILASP approach.

We randomly selected 100 test hypotheses, each of which consisted of between 1 and 3 weak constraints from S_J. For each test hypothesis H_T, we then used the simulated journeys to generate a set of pairs

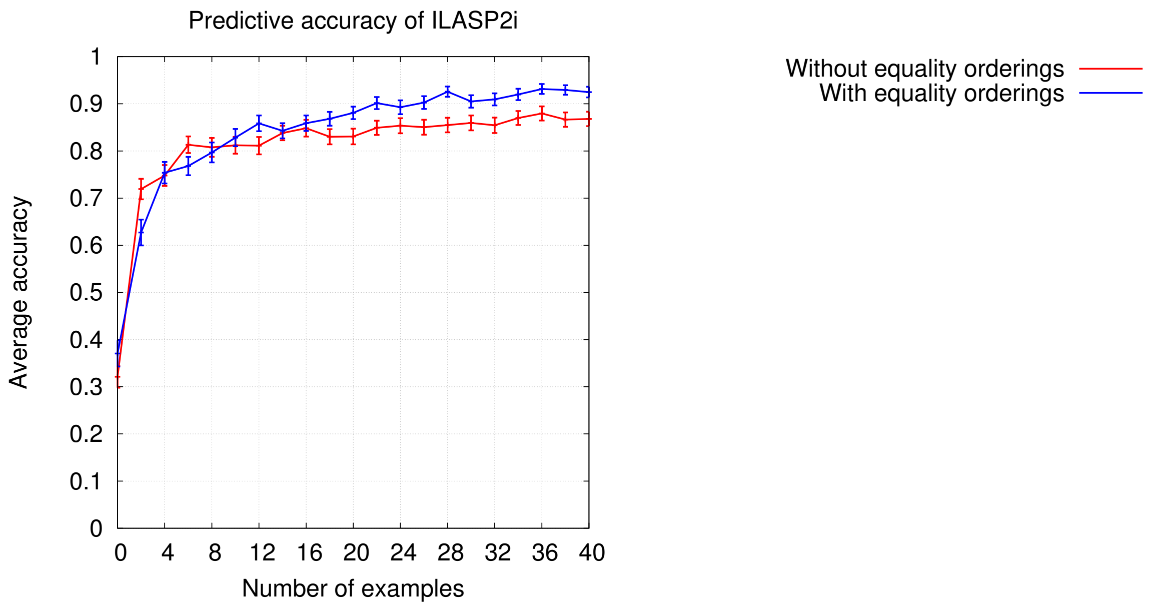


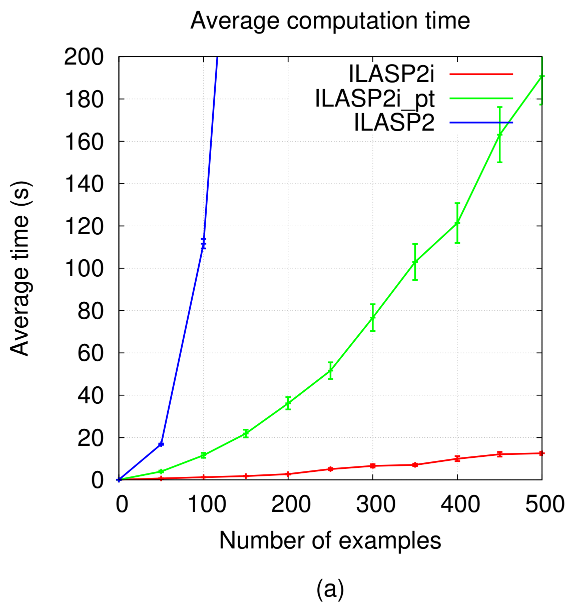
Figure 8.2: Average accuracy of ILASP2i with and without equality orderings on the task of learning journey preferences.

of journeys $\langle j_1, j_2 \rangle$ such that j_1 was one of the optimal journeys, given H_T, and j_2 was an non-optimal alternative to j_1. We then tested the algorithms on tasks with varying numbers of ordering examples by taking a random sample of the complete set of ordering examples.

We measured the accuracy of each learned hypothesis by generating a further set of pairs of journeys, and testing whether the learned hypothesis ordered these pairs correctly; i.e. if j_1 was preferred (resp. equally preferred) to j_2 according to the target hypothesis, then j_1 should be preferred (resp. equally preferred) to j_2 according to the learned hypothesis. The average accuracy of the hypotheses learned by ILASP for varying numbers of examples is shown in Figure 8.2. The average accuracy converged to around 85% after roughly 20 examples. As we only gave examples of journeys such that one was preferred to the other, the hypotheses were often incorrect at predicting the cases where two journeys were equally preferred. We therefore introduced brave ordering examples that used the $=$ operator, meaning that two journeys are equally optimal. We ran the same experiment with half of the ordering examples as “equality” orderings. The average accuracy increased to around 93% after 40 examples.

Comparison of ILASP2 and ILASP2i on Varying Numbers of Examples

Finally, we investigated how the computational performance (running time and peak memory usage) of ILASP2 and ILASP2i varied with the number of examples. We again randomly selected 100 test hypotheses, each of which consisted of between 1 and 3 weak constraints from S_J, and again used the simulated journeys to generate ordered pairs of journeys. The results showed a dramatic difference between the performance of ILASP2 and ILASP2i. ILASP2i has two (related) advantages



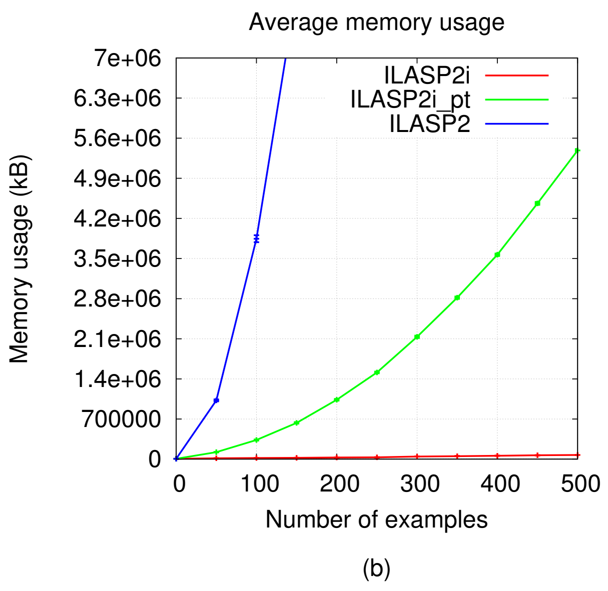


Figure 8.3: (a) the average computation time and (b) the peak memory usage of ILASP2, ILASP2i and ILASP2i_pt for learning journey preferences.

over ILASP2, when it comes to solving context-dependent tasks with large numbers of examples. Firstly, it only needs to consider a small subset of relevant examples, and secondly, it only needs to consider a small subset of the contexts. To investigate how much of the speed up was caused by the context-dependent representation we implemented a variation of the ILASP2i algorithm, which puts the context of every example in the meta-representation (regardless of whether the example has been selected as a relevant example). We call this variation ILASP2i_pt, where the pt stands for "pre-translate", as this can be thought of as pre-translating the $ILP_{LOAS}^{context}$ task into a ILP_{LOAS} task (the resulting meta-representation is indeed very similar to the translation given in Chapter 4 – Definition 4.9).

Figure 8.3(a) and (b) show the running times and peak memory usage (respectively) for up to 500 examples for ILASP2, ILASP2i and ILASP2i_pt. For experiments with more than 200 examples, ILASP2 ran out of memory. By 200 examples, ILASP2i is already over 2 orders of magnitude faster and uses over 2 orders of magnitude less memory than ILASP2, showing a significant improvement in scalability. The fact that by 500 examples ILASP2i is an order of magnitude faster without the pre-translation shows that, in this problem domain, the context is a large factor in this improvement; however, ILASP2i_pt's significantly improved performance over ILASP2 shows that even without context, the relevant examples are also a large factor.

8.3 Scheduling Preferences

The previous experiment considered a preference learning problem in which all examples were completely specified. We now investigate the ILASP algorithms' ability to learn from partial examples. We consider an interview timetabling problem and the task of learning weak constraints that capture an academic's preferences for scheduling undergraduate interviews. In this setting, we use the following language: `slot(D,T)` represents slot T on day D; the `type` predicate represents the course type of each slot; and the `assigned` predicate represents the slots that have been assigned to the interviewer. Using this language we can express, for example, that an academic prefers interviewing for one course to interviewing for another, or prefers not to have many interviews on the same day, or holds both of these preferences but regards the former as more important. These preferences can be encoded using the following two weak constraints:

$$\begin{array}{l} :\sim \texttt{assigned(D,S),type(D,S,c2).[1@2,D,S]} \\ :\sim \texttt{assigned(D,S1),assigned(D,S2),S1!=S2.[1@1,D,S1,S2]} \end{array}$$

The experiments test whether ILP_{LOAS} can successfully learn these kinds of preferences, encoded as weak constraints, from examples of brave and cautious orderings representing ordered pairs of partial timetables. The learning task uses the following background knowledge B (in which it is assumed that there are three interview slots per day):

$$B = \left\{ \begin{array}{lll} \texttt{slot(1..3,1..3).} & & \\ & & \\ \texttt{type(1,1,c2).} & \texttt{type(2,1,c1).} & \texttt{type(3,1,c1).} \\ \texttt{type(1,2,c1).} & \texttt{type(2,2,c1).} & \texttt{type(3,2,c2).} \\ \texttt{type(1,3,c2).} & \texttt{type(2,3,c1).} & \texttt{type(3,3,c1).} \\ & & \\ \texttt{0\{assigned(X,Y)\}1 :- slot(X,Y).} & & \end{array} \right\}$$

In each of the following experiments S_M consisted of 180 weak constraints (characterised by a mode bias, given in Appendix A). As S_M only contained weak constraints, for any $H \subseteq S_M$, $AS(B \cup H) = AS(B)$. Each learning task described in these experiments therefore corresponds to the task of learning to rank the answer sets of B.

The only atoms that vary in B are the `assigned` atoms. For three day timetables, as there are 9 different slots, there are 2^9 answer sets of B (and many more partial interpretations which are extended by these answer sets). We say an example partial interpretation is *full* if it specifies the truth value of all `assigned` atoms, otherwise we describe the *fullness* as the percentage of the atoms which are specified.

In each experiment, given some target hypothesis H_T, ordering examples $o = \langle e_1, e_2, < \rangle$ were randomly generated such that o was bravely respected by H_T. If o was also cautiously respected by H_T, then it was given as a cautious example (otherwise it was given as a brave example). Examples in each learning task corresponded to pairs of partial timetables; however, the weak constraints in each learned program induced an ordering over complete timetables. Therefore, in each experiment, we tested the accuracy of the learned weak constraints on every possible pair of complete timetables (checking whether the learned hypothesis ordered the pair in the same way as the target hypothesis H_T).

Preliminary Experiments

In the first experiment, we generated three learning tasks, with 3, 4 and 5 day timetables, each using the 2 weak constraints in the previous section as a target hypothesis. The examples were of random fullness, each with between half and all of the `assigned` atoms specified.

Learning		#examples				time/s			Memory/kB		
task	S_M	E^+	E^-	O^b	O^c	1	2	2i	1	2	2i
Scheduling (3 day)	180	400	0	110	90	231.6	759.3	112.11	2.9×10^6	2.9×10^6	2.0×10^5
Scheduling (4 day)	180	400	0	128	72	776.8	TO	92.2	5.6×10^6	TO	3.4×10^5
Scheduling (5 day)	180	400	0	133	67	TO	1130.6	107.5	TO	9.0×10^6	4.7×10^5

Table 8.3: The running times and peak memory usages of ILASP1, ILASP2 and ILASP2i on a three instances of the scheduling preference learning problem, with pairwise examples of 3, 4 and 5 day timetables. TO stands for time out (30 minutes). The two learning tasks in this table are available to download from `https://www.doc.ic.ac.uk/~ml1909/ILASP/`.

In each of the three learning tasks, ILASP2i was significantly faster than both ILASP1 and ILASP2. ILASP1 was faster than ILASP2 on the first two tasks (with ILASP2 timing out on the second task), but ILASP2 was faster than ILASP1 (which timed out) on the task with 5 day timetables. The poor performance of both ILASP1 and ILASP2 was again caused by the number of examples, which caused the meta-level programs used by each algorithm to have large groundings. This issue was exacerbated by the size of the problem domain (which increases as the number of days in the example timetables increases). This explains why both ILASP1 and ILASP2 performed best on the task with 3 day timetables. ILASP2i on the other hand performed similarly on each of the three tasks.

Investigating ILASP's Accuracy with Respect to the Number of Examples

We next investigated the relationship between the number of examples and the accuracy of the hypothesis learned by ILASP. Similarly to the journey preference experiments, as each ILASP algorithm returns an arbitrary optimal inductive solution (if it terminates), the average accuracy of each algorithm is the same. In this section, we therefore only present the average accuracy results for ILASP2.

A set of 100 target hypotheses were randomly selected, each with between 1 to 3 weak constraints from S_M, omitting hypotheses that ranked all answer sets equally. We performed this 20 times for each target hypothesis H_T. Each time, ILASP2 was used to learn a hypothesis H_L which covered all of a set of examples (of pairs of three day timetables). We then calculated the accuracy of H_L in predicting the pairwise ordering of the answer sets of B – for each pair of answer sets $A_1, A_2 \in AS(B)$ we tested whether H_T and H_L agreed on the preference ordering between A_1 and A_2.

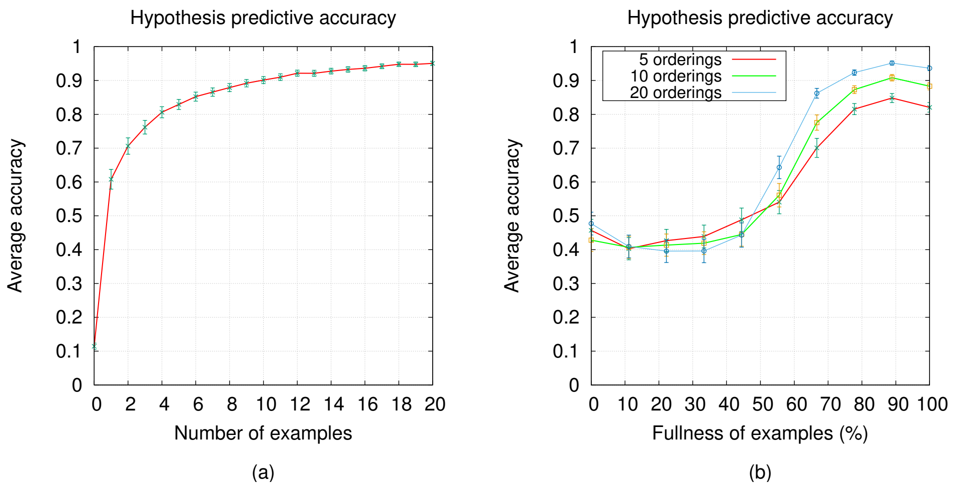


Figure 8.4: Average accuracy with varying (a) numbers of examples; (b) fullness of examples.

The examples were of random fullness, each with between 5 to 9 `assigned` atoms specified. Figure 8.4(a) shows the average predictive accuracy of ILASP2, with the number of ordering examples ranging from 0 to 20. Each point on the graph corresponds to 2000 learning tasks (100 target hypotheses with 20 different sets of examples). The results show that ILASP2 achieved 90% accuracy for this experiment with around 10 or more ordering examples. Note that unlike the journey preference learning setting, we did not need to use equality orderings. The difference between the two settings is that in the journey preference setting two journeys were more likely to be equally preferred (than two schedules in this setting) given the possible target hypotheses, which meant that there was a greater need for equality orderings.

Investigating ILASP's Accuracy with Respect to the Fullness of Examples

In the next experiment we again tested ILASP2 on 100 randomly generated hypotheses with 20 different sets of randomly generated examples. This time, however, the number of examples was fixed at 5, 10 and 20 and it was the fullness of the examples that varied. The results are shown in Figure 8.4(b). The graph shows that examples are only useful if they are more than 50% full. One interesting point to note is that the peak performance is with examples of around 90% fullness. This is caused by a trade off between the usefulness of cautious orderings, and the likelihood of them being generated. A partial interpretation with 8 `assigned` atoms specified is extended by 2 answer sets. So a cautious ordering with $\frac{8}{9}$ fullness is extended by 4 ordered pairs of answer sets, and the ordering means that every one of those 4 pairs must be ordered correctly. Such a cautious ordering is therefore more informative than a cautious ordering with 100% fullness, which is only extended by one pair of ordered answer sets. In general, cautious orderings are more useful if they are less full, as they rule out more hypotheses. This explains why ILASP2 achieved a higher accuracy with examples of 90% fullness. However, as the fullness decreases further, ILASP2's accuracy decreases. This is because it is less likely that the randomly generated orderings would be cautiously respected by the target hypothesis (as there are more pairs that need to be ordered correctly). Hence, the generated tasks with lower fullness had fewer cautious orderings and more brave orderings. Unlike cautious orderings, brave orderings become *less* powerful as the fullness of the examples decreases. This is because brave orderings are more ambiguous when they are less full, and so they rule out fewer hypotheses.

Investigating ILASP's Scalability with Respect to the Number of Examples

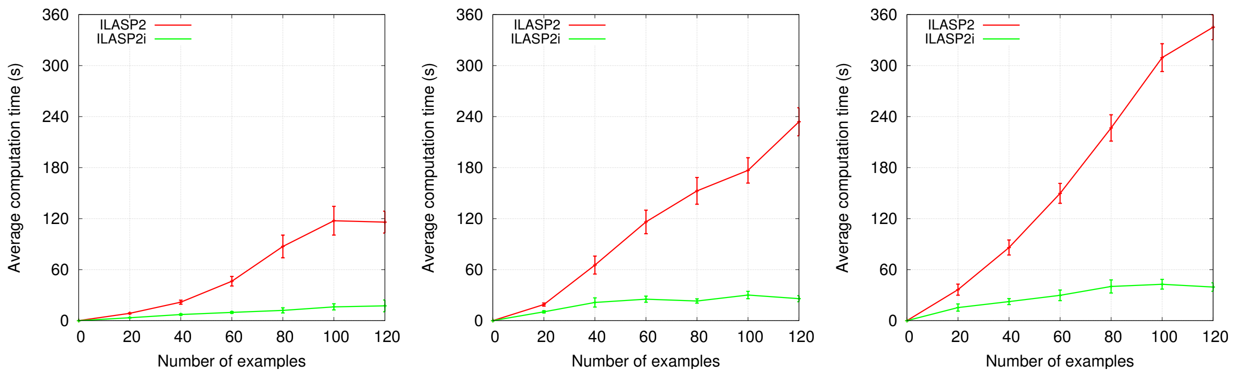


Figure 8.5: Average running time of ILASP2 and ILASP2i with varying numbers of examples. Each experiment had a timeout of 600s, after which the computation was terminated (each timeout was counted as 600s). ILASP2 timed out in 107 of the 2100 experiments and ILASP2i timed out in 3 experiments.

In the final experiment, we investigated the scalability of ILASP2 and ILASP2i by varying both the number of days in the timetables and the number of examples. Figure 8.5 shows the average running time of ILASP2 and ILASP2i with 3, 4 and 5 day timetables (each with 3 slots) with up to 120 ordering examples. Ordering examples were randomly generated, as in the previous experiments, with each example having a random fullness, as in the first experiment. The results show that ILASP2 scales poorly, not only with the number of examples, but also with the size of the problem domain. In comparison, ILASP2i's computation time was very similar across the three domain sizes. Its computation time also increased much more slowly than ILASP2's as the number of examples increased.

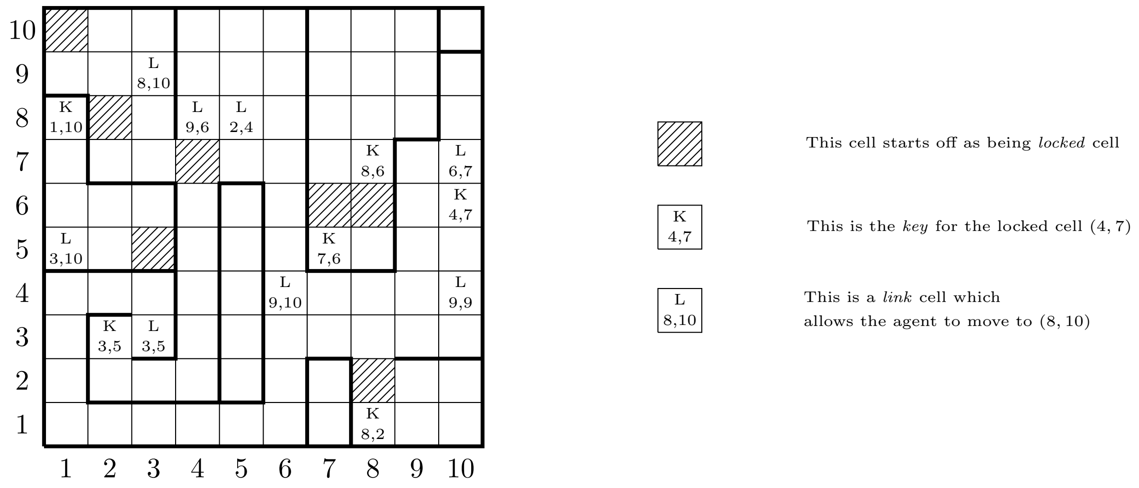
8.4 Agent Navigation Problem

In this section we investigate the problem of an agent learning how to navigate a grid. The agent starts with complete knowledge of the map, but no knowledge of which moves it will be able to make in future time points. At each time point, the agent is informed of which moves it can make by an oracle. The idea is that the agent must learn the rules defining the moves that are valid at each time point. In [LRB14] we showed how ILASP could be used in conjunction with an ASP solver to perform a cycle of planning and learning in order to reach a goal.

We consider four scenarios, which allow us to investigate ILASP's ability to solve tasks with different challenges. In Scenario A, the agent learns just the concept of valid move; in Scenario B, part of the existing background knowledge is removed and the agent has to also learn a new concept that does not appear in the examples or in the background knowledge, showing that ILASP is capable of supporting predicate invention [Sta93]. In Scenario C, the agent must also learn a constraint; and in Scenario D, the agent must also learn a preference ordering over the paths it can take.

In this learning setting each of our examples is of a path taken by the agent through the maze, and the set of moves it was allowed to make at each time point. The rules which should be learned are different in each scenario, and so for some scenarios, different types of examples are needed.

Figure 8.6 gives a graphical representation of the grid and the legend describes its main features. The agent has complete knowledge of the grid map, but it does not know the meaning of the various cell features. For instance, it knows which cells are locked, but not that to go through a locked cell it must first visit the key to that cell.



10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10
L 8,10
K 1,10
L 9,6
L 2,4
K 8,6
L 6,7
K 4,7
L 3,10
K 7,6
L 9,10
L 9,9
K 3,5
L 3,5
K 8,2
This cell starts off as being *locked* cell
This is the *key* for the locked cell (4, 7)
This is a *link* cell which allows the agent to move to (8, 10)

Figure 8.6: Cells with diagonal lines are *locked* and the agent must visit the corresponding *key* before it can enter these cells. *Link* cells allow the agent to jump to the indicated destination cell. The thick black lines represent walls.

We now present the four scenarios, followed by the performance results of each ILASP algorithm on learning tasks from each scenario.

Scenario A

In this simplest scenario, the background knowledge contains a complete description of the map, encoded as facts using the language {`cell/1`, `locked/1`, `key/2`, `link/2`, `wall/2`, `time/1`}, together with the following set of rules:

```
unlocked(C, T) :- visited_cell(Key, T), key(Key, C).
unlocked(C, T) :- cell(C), not locked(C), time(T).

adjacent(cell(X, Y), cell(X + 1, Y)) :- cell(cell(X, Y)), cell(cell(X + 1, Y)).
adjacent(cell(X + 1, Y), cell(X, Y)) :- cell(cell(X, Y)), cell(cell(X + 1, Y)).
adjacent(cell(X, Y + 1), cell(X, Y)) :- cell(cell(X, Y)), cell(cell(X, Y + 1)).
adjacent(cell(X, Y), cell(X, Y + 1)) :- cell(cell(X, Y)), cell(cell(X, Y + 1)).

visited_cell(C,T) :- time(T), T2 <= T, agent_at(C, T2).
```

In this scenario, the agent is expected to learn the following definition of valid move, which means that an agent can move to any unlocked cell that is either linked to its current cell or is adjacent to its current cell and not blocked by a wall.

```
valid_move(C1, T) :- adjacent(C1, C2), agent_at(C2, T),
                     unlocked(C1, T), not wall(C1, C2).
valid_move(C1, T) :- link(C2, C1), agent_at(C2, T), unlocked(C1, T).
```

This problem can be specified in $ILP_{LOAS}^{context}$ as follows. The background knowledge contains the description of the map and the definition of unlocked. Due to the structure of the background knowledge and the hypothesis space, every possible hypothesis is guaranteed to be stratified (when combined with the background knowledge and examples contexts). There is therefore no need for negative examples in this task ($B \cup e_{ctx} \cup H$ is guaranteed to have exactly one answer set for each e_{ctx}, so it is sufficient to give examples of what should, or should not, be in this answer set). For each trace, we constructed a single positive CDPI example, with a context containing the agent's history (its path through the maze so far) and the inclusions and exclusions being a random subset of the moves which were valid and invalid[2], respectively, at each step in the trace; for instance, $\langle\{\texttt{valid\_move}(\texttt{cell}(9,1),1)$, $\texttt{valid\_move}(\texttt{cell}(10,2),1), \ldots\}, \{\texttt{valid\_move}(\texttt{cell}(1,3),1), \ldots\}\rangle, C\rangle$, where C is the set of facts:

[2]Each valid move was given as an inclusion with probability 0.9 and each invalid move was given as an exclusion with probability 0.1. The probability for the exclusions was much lower, as there were many more invalid moves than valid moves.

```
agent_at(cell(10,1),1).    agent_at(cell(9,1),2).    agent_at(cell(8,1),3).
agent_at(cell(8,2),4).     agent_at(cell(8,3),5).    agent_at(cell(8,4),6).
agent_at(cell(7,4),7).     agent_at(cell(6,4),8).    agent_at(cell(9,10),9).
agent_at(cell(8,10),10).   agent_at(cell(8,9),11).   agent_at(cell(8,8),12).
agent_at(cell(8,7),13).
```

Note that this demonstrates the advantage of $ILP_{LOAS}^{context}$ in giving different contexts to different examples. We are able to give the set of valid moves for each example history. For the experiments based on this scenario, the hypothesis space consisted of 531 normal rules, characterised by a set of mode declarations given in Appendix A. In this experiment, the number of body literals permitted in rules in the hypothesis space was restricted to 4.

Scenario B

This scenario differs from the previous one in that the background knowledge does not contain `unlocked`. The language bias of this learning task is augmented with a new predicate with the same structure as `unlocked`, called `extra` added to both M_h and M_b. Note that the predicate `extra/2` does not occur in either the background knowledge or the examples. In the ILP community, this is called *predicate invention* [Sta93]. ILASP supports predicate invention, as long as the structure (the predicate name and arity) of each invented predicate is specified in the hypothesis space. We call this *prescriptive* predicate invention.

Other than the need for predicate invention, this scenario is identical to Scenario A. The agent is expected to learn an equivalent definition of valid move. One might expect the solution to be the previous hypothesis augmented with the definition of `unlocked` that was removed from the background knowledge, with the only difference being that the predicate `unlocked` would be replaced by the predicate `extra`. This would be the hypothesis:

```
valid_move(C1, T) :- adjacent(C1, C2), agent_at(C2, T),
                     extra(C1, T), not wall(C1, C2).
valid_move(C1, T) :- link(C2, C1), agent_at(C2, T), extra(C1, T).

extra(C, T) :- visited_cell(Key, T), key(Key, C).
extra(C, T) :- cell(C), not locked(C), time(T).
```

In fact, ILASP learns the shorter hypothesis:

```
valid_move(V0, V1) :- extra(V0, V1), visited_cell(V2, V1), key(V2, V0).
valid_move(V0, V1) :- extra(V0, V1), not locked(V0).
```

```
extra(V2, V1) :- agent_at(V0, V1), link(V0, V2).
extra(V2, V1) :- agent_at(V0, V1), not wall(V0, V2), adjacent(V0, V2).
```

In this definition, `extra` essentially means "connected to the agent's current cell", where a cell is "connected" to another cell if the two cells are either adjacent to one another and there is no wall between them or if there is a link from the first cell to the second cell. The learned definition of valid move then says that the agent can move to any cell which is connected to its current cell and that was either not locked to begin with, or for which the agent has visited the corresponding key cell.

The task uses a hypothesis space of 146 normal rules (characterised by a set of mode declarations given in Appendix A). In this experiment, the number of body literals permitted in rules in the hypothesis space was restricted to 3. The examples in this task are of the same structure as those in Scenario A.

Scenario C

In this scenario, the agent must learn the same definition of valid move from Scenario A and an additional constraint expressing that the agent cannot visit the same cell more than once. Note that this does not change the rule definition of `valid_move`. If the agent were to visit the same cell more than once, the individual moves would still be valid according to the rule definition, but the trace as a whole would be invalid. The background knowledge in this scenario is the same as in Scenario A, but with the following additional rule for `already_visited_cell`.

```
already_visited_cell(C,T) :- time(T), T2 < T, agent_at(C, T2).
```

Note that this is different to the rule for `visited_cell`, as it requires that the cell has been visited in a previous time point (excluding the current time point). The full target hypothesis is as follows:

```
valid_move(C1, T) :- adjacent(C1, C2), agent_at(C2, T),
                     unlocked(C1, T), not wall(C1, C2).
valid_move(C1, T) :- link(C2, C1), agent_at(C2, T), unlocked(C1, T).

:- agent_at(C, T), already_visited_cell(C, T).
```

To learn this definition, ILASP needs positive examples of the same form as those in Scenarios A and B, but as the hypothesis contains a constraint, the task must also include negative examples that rule out agent histories where the agent has visited the same cell more than once. The negative examples are similar to the positive examples, but contain invalid agent histories in the context. One such example could be:

```
⟨⟨{valid_move(cell(9,1),1), ...}, {valid_move(cell(3,3),1), ...}⟩, {
   agent_at(cell(10,1),1).    agent_at(cell(9,1),2).     agent_at(cell(8,1),3).
   agent_at(cell(8,2),4).     agent_at(cell(8,3),5).     agent_at(cell(7,3),6).
   agent_at(cell(7,4),7).     agent_at(cell(7,3),8).     agent_at(cell(7,4),9).
   agent_at(cell(6,4),10).    agent_at(cell(9,10),11).   agent_at(cell(8,10),12).
   agent_at(cell(7,10),13).   agent_at(cell(7,9),14).
}⟩
```

The hypothesis rule consists of 160 normal rules and hard constraints (characterised by a set of mode declarations given in Appendix A). In this experiment, the number of body literals permitted in rules in the hypothesis space was restricted to 4.

Scenario D

The final scenario combines rule learning, constraint learning and preference learning. In this scenario, the map is augmented with two new concepts. Each cell is given a danger rating of 1, 2 or 3 and some cells are given coins with a value of between 1 and 5. The agent should collect the coins, but avoid dangerous cells at night. The background knowledge also contains facts that state when it is daytime. The additional predicates in the language are {`danger_rating/2`, `coin/2`, `daytime/1`}.

The agent is expected to learn the same definition of valid move and the same constraint as in Scenario C and, additionally, a set of weak constraints expressing a preference ordering over the possible paths that an agent can take. The weak constraints in the target hypothesis express the following: the agent's top priority is to minimise the danger the agent is exposed to at night; its second priority is to maximise the value of the coins it collects; and its final priority is to minimise the length of its path. The full target hypothesis is as follows:

```
valid_move(V1, V2) :- adjacent(V0, V1), agent_at(V0, V2),
                      unlocked(V1, V2), not wall(V0, V1).
valid_move(V2, V1) :- agent_at(V0, V1), unlocked(V2, V1), link(V0, V2).

:- agent_at(V0, V1), already_visited_cell(V0, V1).

:~ agent_at(V0, V1), not daytime(V1), danger_rating(V0, V2).[V2@3, V0, V1, V2]
:~ agent_at(V0, V1), coin(V0, V2).[V2@2, V0, V1, V2]
:~ agent_at(V0, V1).[1@1, V0, V1]
```

In the experiments, the learning tasks contained similar positive and negative examples to those in Scenario C, but also contained brave ordering examples to learn the weak constraints. The ordering examples were over the previously described positive examples, and described agent histories which were preferred (given the weak constraints) to other agent histories. As the agent histories were complete (and the weak constraints in the hypothesis space did not depend on the partially specified `valid_move` predicate), the notions of brave and cautious orderings coincided, so there was no need for any cautious orderings.

The hypothesis space in these tasks contains 244 rules and weak constraints (characterised by a set of mode declarations in Appendix A). In this experiment, the number of body literals permitted in rules in the hypothesis space was restricted to 4.

Results

Similarly to the other problem settings, we first ran a small preliminary experiment, testing the three ILASP algorithms on a single learning task for each of the four scenarios.

Learning		#examples				time/s			Memory/kB		
task	S_M	E^+	E^-	O^b	O^c	1	2	2i	1	2	2i
Scenario A	531	200	0	0	0	851.9	432.3	18.2	OOM	OOM	9.0×10^4
Scenario B	146	50	0	0	0	OOM	OOM	5.4	OOM	OOM	1.8×10^5
Scenario C	160	80	120	0	0	283.7	226.5	16.5	2.7×10^6	2.9×10^6	9.6×10^4
Scenario D	244	172	228	390	0	OOM	OOM	850.4	OOM	OOM	1.2×10^6

Table 8.4: The running time and peak memory usage of ILASP1, ILASP2 and ILASP2i on single instances of each of the four agent navigation scenarios. OOM stands for out of memory. The learning tasks in this table are available to download from `https://www.doc.ic.ac.uk/~ml1909/ILASP/`.

ILASP1 and ILASP2 ran out of memory in two out of the four problems. This was due to the large number of examples in the learning tasks. In the comparison of ILASP1 and ILASP2, the most noteworthy task is Scenario A. It has no negative examples or cautious orderings, so ILASP1's usual efficiency issue caused by violating hypotheses was not to blame for its performance. In fact, as this task only had positive examples, the only difference between the ILASP1 and ILASP2 algorithms in solving the task was that ILASP2 only used a single call to the ASP solver, which returned an optimal answer set, corresponding to an optimal inductive solution; ILASP1 on the other hand made $2n$ calls (where n is the length of the optimal inductive solution), searching for hypotheses of each length up to n. ILASP2i performed best on each of the four learning tasks, but took rather longer to solve Scenario D than any of the other scenarios, as in this task the optimal inductive solution was much longer, which meant that more relevant examples (and hence, more iterations) were required to learn it. In general, the higher the length of the optimal solution of a task, the more iterations of ILASP2i are needed to learn it.

Figure 8.7 shows the performance of ILASP2i on each of the four scenarios. Each point in the graph is an average over 20 randomly generated learning tasks for this scenario (the background knowledge is fixed in each case and the examples are randomly generated). In Scenarios A and B, the number of CDPIs corresponds to the number of positive examples (as these are the only kind of examples in the task). In Scenarios C and D, it corresponds to the number of positive and negative examples. In Scenario D there is also an equal number of brave orderings.

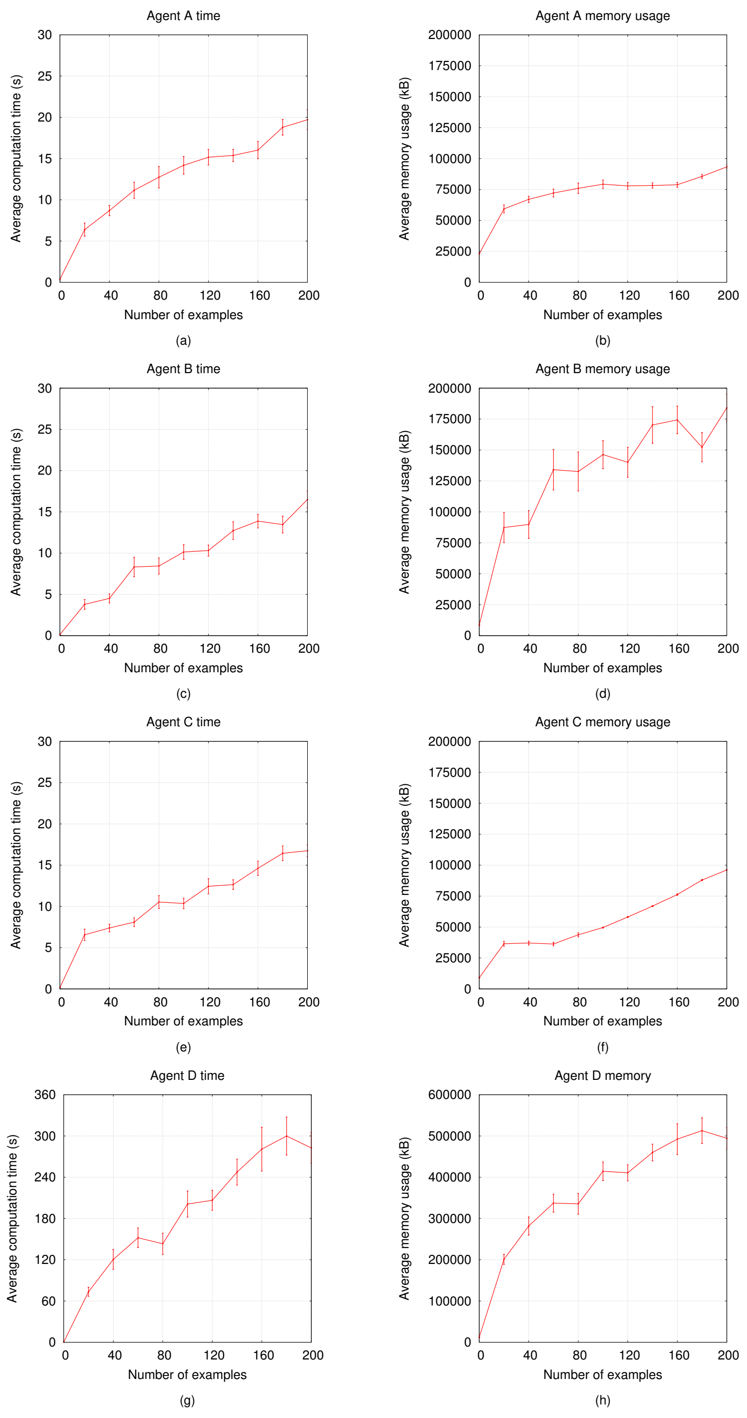


Figure 8.7: Average computation time and peak memory usage used by ILASP2i with varying numbers of examples on each of the four scenarios.

For the first three scenarios, ILASP2i terminated in under 30s on average, even with 200 examples. For the final scenario, however, ILASP2i took over 3 minutes with 100 CDPIs (so 200 examples overall).

This is expected, as the target hypothesis is considerably longer than in the other scenarios.

Summary

This chapter concludes our work on learning ASP programs from non-noisy examples. In this part of the thesis we have presented new frameworks and algorithms for learning from perfectly labeled data. We have also conducted a thorough study of the notion of the generality of learning frameworks under the answer set semantics. From this point on, we devote our attention to learning from noisy examples. The next chapter presents extensions of our learning frameworks in order to handle examples that may or may not be covered and investigates what this means for the generality and complexity results of the frameworks.

Part II

Learning Answer Set Programs from Noisy Examples

Chapter 9

Learning from Noisy Answer Sets

In the first part of this thesis, we addressed the problem of learning ASP programs under the assumption that all examples were perfectly labeled – i.e. we assumed that there was no noise in the data. In practice, this assumption can be unrealistic, as data can come from unreliable sources. In Part II, we now address the more general case of learning ASP programs from examples that may not be perfectly labeled.

This chapter introduces the ILP_{LOAS}^{noise} framework, which extends the $ILP_{LOAS}^{context}$ framework to enable learning from noisy examples. We show that the complexity and generality results presented in Chapters 4 and 5 are unaffected by this extension. Section 9.4 shows how the ILASP2 and ILASP2i algorithms can be upgraded to solve noisy tasks, but illustrates with an example why neither algorithm is particularly well suited to solving this type of learning problem. Motivated by this, Chapter 10 introduces the ILASP3 algorithm, which is specifically targeted at solving ILP_{LOAS}^{noise} tasks. Finally, in Chapter 11 we evaluate the performance of these algorithms.

9.1 Context-Dependent Noisy Learning from Ordered Answer Sets

In this section, we present our noisy framework, ILP_{LOAS}^{noise}, for learning from noisy ordered answer sets. It extends the most general of our previous non-noisy learning frameworks, $ILP_{LOAS}^{context}$, by allowing examples to be *weighted context-dependent partial interpretations* and *weighted context-dependent ordering examples.* These are essentially the same as context-dependent partial interpretations and context-dependent ordering examples, but now additionally weighted with a notion of *penalty.* If a hypothesis does not cover an example, then we say that it *pays the penalty* of that example. Informally, penalties are used to calculate the *cost* associated with a hypothesis for not covering examples. The cost function of a hypothesis H is the sum over the penalties of all the examples that are not *covered* by H, augmented with the length of the hypothesis. The goal of ILP_{LOAS}^{noise} is to find a hypothesis that minimises the cost function over a given hypothesis space with respect to a given set of examples.

Definition 9.1. A weighted context-dependent partial interpretation e is a tuple $\langle e_{id}, e_{pen}, \langle e_{pi}, e_{ctx}\rangle\rangle$, where e_{id} is a constant, called the *identifier* of e (unique to each example), e_{pen} is the penalty of e, e_{pi} is a partial interpretation and e_{ctx} is an ASP program called the *context* of e. The penalty e_{pen} is either a positive integer, or ∞. A program P *accepts* e iff it accepts $\langle e_{ctx}, e_{pi}\rangle$.

Definition 9.2. A weighted context-dependent ordering example o is a tuple $\langle o_{id}, o_{pen}, o_{ord}\rangle$, where o_{id} is a constant, called the *identifier* of o, o_{pen} is the penalty of o and o_{ord} is a CDOE. The penalty o_{pen} is either a positive integer, or ∞. A program P *bravely* (resp. *cautiously*) *respects* o iff it bravely (resp. cautiously) respects o_{ord}.

In a learning task without noise, each example must be covered by any inductive solution. However, when examples are noisy (i.e. they have a weight), solutions may not necessarily cover every example, but will incur penalties instead. Notice that multiple occurrences of the same context-dependent partial interpretation will have different identifiers and so hypotheses that do not cover the example will pay the penalty multiple times (for instance, if a context-dependent partial interpretation occurs twice then a hypothesis will have to pay twice the penalty for not covering it).

In most of the learning tasks in our evaluation in Chapter 11, we assign the same penalty to each example. In some cases, however, the penalties can be used to simulate *oversampling*; for example, if we have far more positive examples than negative examples, we may choose to give the negative examples a higher weight – this makes it less likely that we will learn a hypothesis that treats all negative examples as noisy, just because they are under represented in the data. Another potential use of penalties would be to use them to represent some sort of confidence in each example[1]. Definition 9.3 formalises our ILP_{LOAS}^{noise} framework.

Definition 9.3. An ILP_{LOAS}^{noise} task T is a tuple of the form $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$, where B is an ASP program, S_M is a hypothesis space, E^+ and E^- are sets of weighted context-dependent partial interpretations and O^b and O^c are sets of weighted context-dependent ordering examples. Given a hypothesis $H \subseteq S_M$,

1. *uncov*(H, T) is the set consisting of:
   (a) all positive examples $e \in E^+$ such that $B \cup H$ does not accept e.
   (b) all negative examples $e \in E^-$ such that $B \cup H$ accepts e.
   (c) all brave ordering examples $o \in O^b$ such that $B \cup H$ does not bravely respect o.
   (d) all cautious ordering examples $o \in O^c$ such that $B \cup H$ does not cautiously respect o.
2. the penalty of H, denoted as *pen*(H, T), is the sum $\sum_{e \in uncov(H,T)} e_{pen}$.
3. the score of H, denoted as $\mathcal{S}(H, T)$, is the sum $|H| + pen(H, T)$.

[1]We do not explore this use of penalties. Any such use would first require a formalisation of what is meant by the confidence in an example.

4. H is an inductive solution of T (written $H \in ILP^{noise}_{LOAS}(T)$) if and only if $\mathcal{S}(H,T)$ is finite.

5. H is an *optimal inductive solution* of T (written $H \in {}^*ILP^{noise}_{LOAS}(T)$) if and only if $\mathcal{S}(H,T)$ is finite and $\nexists H' \subseteq S_M$ such that $\mathcal{S}(H,T) > \mathcal{S}(H',T)$.

Note that an example with infinite penalty *must* be covered by any inductive solution of the task, as any hypothesis that does not cover such an example will have an infinite score. An ILP^{noise}_{LOAS} task T is said to be *satisfiable* if $ILP^{noise}_{LOAS}(T)$ is non-empty. If $ILP^{noise}_{LOAS}(T)$ is empty, then T is said to be *unsatisfiable.*

Example 9.1. *Consider a simple ILP^{noise}_{LOAS} learning task T, with background knowledge $B = \{$`r(X):-s(X),p. s(1..98).`$\}$, a hypothesis space consisting of the two facts "*`p.`*" and "*`q.`*" and examples as defined below:*

$E^+ = \{\langle i, 1, \langle\langle\{\texttt{r(i)}\}, \emptyset\rangle, \emptyset\rangle\rangle | i \in [1..50]\} \cup \{\langle i, 1, \langle\langle\emptyset, \{\texttt{r(i)}\}\rangle, \emptyset\rangle | i \in [51..98]\}$

$E^- = \{\langle 99, \infty, \langle\langle\{\texttt{q}\}, \emptyset\rangle, \emptyset\rangle\}$

$O^b = \emptyset$

$O^c = \emptyset$

There are four possible hypotheses in the hypothesis space: $H_1 = \emptyset$, $H_2 = \{$`p.`$\}$, $H_3 = \{$`q.`$\}$ and $H_4 = \{$`p. q.`$\}$. H_3 and H_4 have an infinite score, as they accept the negative example. $\mathcal{S}(H_2,T) = 49$, as H_2 pays a penalty of 48 for not covering the examples with id greater than 50 and less than 99, and $|H_2| = 1$. $\mathcal{S}(H_1,T) = 50$ as H_1 pays a penalty of 50 for not covering any of the examples with id less than 51, and $|H_1| = 0$. Hence, the only optimal inductive solution of T is H_2; i.e. ${}^*ILP^{noise}_{LOAS}(T) = \{H_2\}$. The full set of inductive solutions of T, $ILP^{noise}_{LOAS}(T)$, is $\{H_1, H_2\}$.

9.2 The Complexity of Noisy Learning Tasks

In this section, we show that the complexity of ILP^{noise}_{LOAS} is the same as $ILP^{context}_{LOAS}$ for the three decision problems of *verification*, *satisfiability* and *optimum verification* presented in Section 4.5.

Proposition 9.2. Deciding verification, satisfiability and optimum verification for $ILP^{context}_{LOAS}$ each reduce polynomially to the same problem for ILP^{noise}_{LOAS}.

Proof. Let T be the $ILP^{context}_{LOAS}$ task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$. Consider the ILP^{noise}_{LOAS} task $T' = \langle B, S_M, \langle E^+_2, E^-_2, O^b_2, O^c_2\rangle\rangle$, where the examples are defined as follows:

- $E^+_2 = \{\langle e_{id}, \infty, e\rangle | e \in E^+\}$
- $E^-_2 = \{\langle e_{id}, \infty, e\rangle | e \in E^-\}$

- $O_2^b = \{\langle o_{id}, \infty, o\rangle | o \in O^b\}$
- $O_2^c = \{\langle o_{id}, \infty, o\rangle | o \in O^c\}$

First note that $H \in ILP^{context}_{LOAS}(T) \Leftrightarrow \mathcal{S}(H, T')$ is finite. Hence $H \in ILP^{context}_{LOAS}(T) \Leftrightarrow H \in ILP^{noise}_{LOAS}(T')$. So verification for $ILP^{context}_{LOAS}$ reduces to verification for ILP^{noise}_{LOAS}. As this also means that $ILP^{context}_{LOAS}(T) = \emptyset \Leftrightarrow ILP^{noise}_{LOAS}(T') = \emptyset$, this also shows that satisfiability for $ILP^{context}_{LOAS}$ reduces to satisfiability for ILP^{noise}_{LOAS}.

It remains to show that optimum verification for $ILP^{context}_{LOAS}$ reduces to optimum verification for ILP^{noise}_{LOAS}. We do this by showing that $^*ILP^{context}_{LOAS}(T) = {}^*ILP^{noise}_{LOAS}(T')$. As $ILP^{context}_{LOAS}(T) = ILP^{noise}_{LOAS}(T')$, to do this, it suffices to show that $\forall H, H' \in ILP^{noise}_{LOAS}(T')$, $\mathcal{S}(H, T) < \mathcal{S}(H', T)$ if and only if $|H| < |H'|$. Well, as both $\mathcal{S}(H, T)$ and $\mathcal{S}(H', T)$ are finite, both $uncov(H, T)$ and $uncov(H', T)$ must be empty (as every example in T' has an infinite penalty). So $\mathcal{S}(H, T) = |H|$ and $\mathcal{S}(H', T) = |H'|$. Hence, $\mathcal{S}(H, T) < \mathcal{S}(H', T)$ if and only if $|H| < |H'|$. □

It remains to show that each of the decision problems for ILP^{noise}_{LOAS} reduces to the corresponding decision problem for $ILP^{context}_{LOAS}$. For verification and satisfiability this is trivial. Examples with finite penalty can be ignored, as they do not affect whether any hypothesis is an inductive solution.

Proposition 9.3. Deciding verification and satisfiability for ILP^{noise}_{LOAS} each reduce polynomially to the same decision problem for $ILP^{context}_{LOAS}$.

Proof. Let T be the ILP^{noise}_{LOAS} task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$. Consider the $ILP^{context}_{LOAS}$ task $T' = \langle B, S_M, \langle E_2^+, E_2^-, O_2^b, O_2^c\rangle\rangle$, where the examples are defined as follows:

- $E_2^+ = \{\langle e_{pi}, e_{ctx}\rangle | \langle e_{id}, e_{pen}, \langle e_{pi}, e_{ctx}\rangle\rangle \in E^+, e_{pen} = \infty\}$
- $E_2^- = \{\langle e_{pi}, e_{ctx}\rangle | \langle e_{id}, e_{pen}, \langle e_{pi}, e_{ctx}\rangle\rangle \in E^-, e_{pen} = \infty\}$
- $O_2^b = \{o_{ord} | \langle o_{id}, o_{pen}, o_{ord}\rangle \in O^b, o_{pen} = \infty\}$
- $O_2^c = \{o_{ord} | \langle o_{id}, o_{pen}, o_{ord}\rangle \in O^c, o_{pen} = \infty\}$

$\forall H \subseteq S_M$, $H \in ILP^{context}_{LOAS}(T')$ if and only if H covers all examples in T that have a finite penalty. Hence, $ILP^{context}_{LOAS}(T') = ILP^{noise}_{LOAS}(T)$. This means that both verification and satisfiability for ILP^{noise}_{LOAS} reduce to verification and satisfiability for $ILP^{context}_{LOAS}$ (as $H \in ILP^{noise}_{LOAS}(T) \Leftrightarrow H \in ILP^{context}_{LOAS}(T')$ and $ILP^{noise}_{LOAS}(T) = \emptyset \Leftrightarrow ILP^{context}_{LOAS}(T') = \emptyset$). □

We now show that optimum verification for ILP^{noise}_{LOAS} is in Π_2^P (which is the same complexity class as optimum verification for $ILP^{context}_{LOAS}$).

Proposition 9.4. Deciding optimum verification for ILP^{noise}_{LOAS} is in Π_2^P.

Proof. Let T be the ILP^{noise}_{LOAS} task $\langle B, S_M, E\rangle$ and H be a subset of S_M. We will show that whether H is *not* an optimal solution of T can be verified by a non-deterministic Turing Machine with an NP-oracle in polynomial time.

Firstly deciding whether H is an inductive solution of T reduces to deciding verification for $ILP^{context}_{LOAS}$, so it is in DP (it can be reduced to two decision problems D_1 and D_2 in NP such that H is a solution of T if and only if D_1 answers yes and D_2 answers no). Hence, the Turing Machine can start by making two calls to the NP oracle, and if either D_1 answers no or D_2 answers yes, the Turing Machine can return yes (as H is not an optimal solution of T). The Turing Machine then searches for a hypothesis $H' \subseteq H$ such that $\mathcal{S}(H,T) > \mathcal{S}(H',T)$ (as this is the only remaining way that H can not be an optimal solution of T).

Let $|E|$ be the number of examples in the tuple E. The Turing Machine makes $|S_M|$ choices to decide which rules are in H', then makes $|E|$ choices to decide which examples are not covered by H and a further $|E|$ choices to decide which examples are covered by H'. Each branch corresponds to a hypothesis H' and two tuples of examples E_H and $E_{H'}$. The branch must now verify three things: (1) that H covers none of the examples in E_H; (2) that H' covers every example in $E_{H'}$; and (3), that $|H| + \mathcal{SP}(E_H) > |H'| + \mathcal{SP}(E\backslash E_{H'})$, where $\mathcal{SP}$ computes the sum of the penalties in a tuple of examples. If all of these properties hold, then $\mathcal{S}(H',T) < \mathcal{S}(H,T)$[2], so the branch can return yes, otherwise it will return no. It remains to show that these three properties can be verified in polynomial time using an NP oracle.

1. To check that H covers none of a set of examples in E_H, we take each example *ex* in turn and consider an $ILP^{context}_{LOAS}$ task T' containing only that example (and the relevant positive CDPI examples if *ex* is an ordering example), with its penalty and id removed. H covers *ex* if and only of H is a solution of T'. By Theorem 4.21, verifying that H is a solution of T' is DP-complete. Hence, there are two NP-complete decision problems D_1 and D_2 such that H covers *ex* if and only if D_1 answers yes and D_2 answers no. As we are verifying that no example in E_H is covered, if D_1 returns yes and D_2 returns no, then the branch returns no, otherwise it carries on to the next example in E_H until none are left. Checking all of the examples requires at most $2 \times |E_H|$ calls to the oracle.

2. To check that H' covers every example in $E_{H'}$, we simply verify that it is a solution of the $ILP^{context}_{LOAS}$ task $\langle B, S_M, E_{H'}\rangle$ (where all penalties and ids have again been removed from the examples). Again, by Theorem 4.21, verifying that H' is a solution of this task is DP-complete. Hence, the verification can be achieved with two calls to the NP oracle.

3. As this is a simple sum, it can be computed in polynomial time by the Turing Machine without even needing to use the oracle.

[2] $\mathcal{S}(H',T) \leq |H'| + \mathcal{SP}(E\backslash E_{H'}) < |H| + \mathcal{SP}(E_H) \leq \mathcal{S}(H,T)$.

If there is at least one branch that returns yes then H is not an optimal solution of T. If there is a hypothesis H' such that $\mathcal{S}(H,T) > \mathcal{S}(H',T)$ then the branch where $E_H = uncov(T,H)$ and $E_{H'} = E \backslash uncov(T,H')$ will return yes. Hence the non-deterministic Turing Machine returns yes if and only if H is not an optimal solution of T. Hence the optimum verification problem for ILP^{noise}_{LOAS} is in Π^P_2.

□

We now show that for each of the decision problems in Section 4.5 ILP^{noise}_{LOAS} has the same complexity as the corresponding decision problem for $ILP^{context}_{LOAS}$.

Theorem 9.5.

1. *Deciding verification for an arbitrary ILP^{noise}_{LOAS} task is DP-complete*
2. *Deciding satisfiability for an arbitrary ILP^{noise}_{LOAS} task is Σ^P_2-complete*
3. *Deciding optimum verification for an arbitrary ILP^{noise}_{LOAS} task is Π^P_2-complete*

Proof.

1. As verification for $ILP^{context}_{LOAS}$ is DP-complete (by Theorem 4.21), and there are polynomial reductions to and from verification for ILP^{noise}_{LOAS} (by Propositions 9.2 and 9.3), verification for ILP^{noise}_{LOAS} must also be DP-complete.
2. Similarly, as satisfiability for $ILP^{context}_{LOAS}$ is Σ^P_2-complete (by Theorem 4.24), and there are polynomial reductions to and from satisfiability for ILP^{noise}_{LOAS} (by Propositions 9.2 and 9.3), satisfiability for ILP^{noise}_{LOAS} must also be Σ^P_2-complete.
3. Optimum verification for $ILP^{context}_{LOAS}$ reduces to optimum verification for ILP^{noise}_{LOAS} (by Proposition 9.2), hence as optimum verification for $ILP^{context}_{LOAS}$ is Π^P_2 complete (by Theorem 4.27), optimum verification for ILP^{noise}_{LOAS} must be Π^P_2-hard. Hence, as optimum verification for ILP^{noise}_{LOAS} is a member of Π^P_2 (by Proposition 9.4), it must be Π^P_2-complete.

□

9.3 The Generality of Noisy Learning Tasks

In this section we consider the generality of the ILP^{noise}_{LOAS} framework and compare it to the generality of noisy versions of the frameworks that we analysed in Chapter 5. These noisy frameworks are defined in the same way as ILP^{noise}_{LOAS}, by taking the original definition of an example e, and replacing it with a tuple $\langle e_{id}, e_{pen}, e\rangle$. For any task $T = \langle B, E\rangle$ of any framework, we write $pen(H,T)$ to denote the sum of the penalties of all examples in E that are not covered by H. Given a framework $\mathcal{F}$, we denote the noisy equivalent of the framework as $n(\mathcal{F})$. Note that $n(ILP^{context}_{LOAS}) = ILP^{noise}_{LOAS}$.

Noise is not captured in the definition of inductive solution, but is instead captured by the penalty paid by a hypothesis. We therefore slightly alter the notion of distinguishability.

Definition 9.4. The *noisy one-to-one-distinguishability class* of a noisy learning framework $\mathcal{F}$ (denoted $\mathcal{ND}^1_1(\mathcal{F})$) is the set of tuples $\langle B, H_1, H_2\rangle$ of ASP programs for which there is at least one task $T_{\mathcal{F}} = \langle B, E_{\mathcal{F}}\rangle$ such that $pen(H_1, T_{\mathcal{F}}) < pen(H_2, T_{\mathcal{F}})$. For each $\langle B, H_1, H_2\rangle \in \mathcal{ND}^1_1(\mathcal{F})$, $T_{\mathcal{F}}$ is said to *distinguish* H_1 from H_2 with respect to B. Given two noisy frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, we say that $\mathcal{F}_1$ is at least as (resp. more) $\mathcal{ND}^1_1$-general as (resp. than) $\mathcal{F}_2$ if $\mathcal{ND}^1_1(\mathcal{F}_2) \subseteq \mathcal{ND}^1_1(\mathcal{F}_1)$ (resp. $\mathcal{ND}(F_2) \subset \mathcal{ND}^1_1(F_1)$).

We now show that for any framework $\mathcal{F}$, the noisy one-to-one-distinguishability class of $n(\mathcal{F})$ is equal to the one-to-one-distinguishability class of $\mathcal{F}$.

Theorem 9.6. *For any non-noisy framework $\mathcal{F}$, $\mathcal{ND}^1_1(n(\mathcal{F})) = \mathcal{D}^1_1(\mathcal{F})$.*

Proof.

1. We first show that $\mathcal{ND}^1_1(n(\mathcal{F})) \subseteq \mathcal{D}^1_1(\mathcal{F})$.

   Assume that $\langle B, H_1, H_2\rangle \in \mathcal{ND}^1_1(n(ILP_{\mathcal{F}}))$. Then there is an $n(ILP_{\mathcal{F}})$ task $T_{\mathcal{F}}$ with background knowledge B such that $pen(H_1, T_{\mathcal{F}}) < pen(H_2, T_{\mathcal{F}})$. There must be at least one example e^* in $T_{\mathcal{F}}$ that H_1 covers but H_2 does not. Let $T^2_{\mathcal{F}}$ be the $ILP_{\mathcal{F}}$ task with background knowledge B and only the example e^*. Then $H_1 \in ILP_{\mathcal{F}}(T^2_{\mathcal{F}})$ and $H_2 \notin ILP_{\mathcal{F}}(T^2_{\mathcal{F}})$. Hence $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_{\mathcal{F}})$.

2. Now we show that $\mathcal{D}^1_1(\mathcal{F}) \subseteq \mathcal{ND}^1_1(n(\mathcal{F}))$.

   Assume that $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F})$. Then there is an $ILP_{\mathcal{F}}$ task $T_{\mathcal{F}}$ with background knowledge B such that $H_1 \in ILP_{\mathcal{F}}(T_{\mathcal{F}})$ and $H_2 \notin ILP_{\mathcal{F}}(T_{\mathcal{F}})$. Let $T^2_{\mathcal{F}}$ be the $n(ILP_{\mathcal{F}})$ task constructed from $T_{\mathcal{F}}$ by adding infinite penalties (and ids) to each example. For any H, $H \in ILP_{\mathcal{F}}(T_{\mathcal{F}})$ if and only if H covers every example in T, which is true if and only if $pen(H, T^2_{\mathcal{F}})$ is finite. Hence $pen(H_1, T^2_{\mathcal{F}}) < pen(H_2, T^2_{\mathcal{F}})$ (as $pen(H_1, T^2_{\mathcal{F}})$ is finite and $pen(H_2, T^2_{\mathcal{F}})$ is infinite). So $\langle B, H_1, H_2\rangle \in \mathcal{ND}^1_1(n(ILP_{\mathcal{F}}))$.

□

As we have already proven the one-to-one-distinguishability classes of the six frameworks in Chapter 5, this result leads to the following ordering of noisy one-to-one-distinguishability classes.

Corollary 9.7.

1. $\mathcal{ND}^1_1(n(ILP_b)) = \mathcal{ND}^1_1(n(ILP_{sm})) \subset \mathcal{ND}^1_1(n(ILP_{LAS})) \subset \mathcal{ND}^1_1(n(ILP_{LOAS})) \subset \mathcal{ND}^1_1(ILP^{noise}_{LOAS})$
2. $\mathcal{ND}^1_1(n(ILP_c)) \subset \mathcal{ND}^1_1(n(ILP_{LAS}))$

Similarly, we can upgrade the definition of the one-to-many-distinguishability class.

Definition 9.5. The *noisy one-to-many-distinguishability class* of a noisy learning framework $\mathcal{F}$ (denoted $\mathcal{ND}^1_m(\mathcal{F})$) is the set of all tuples $\langle B, H, \{H_1, \ldots, H_n\}\rangle$ such that there is a task $T_\mathcal{F}$ which distinguishes H from each H_i with respect to B. Given two noisy frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, we say that $\mathcal{F}_1$ is at least as (resp. more) $\mathcal{ND}^1_m$-general as (resp. than) $\mathcal{F}_2$ if $\mathcal{ND}^1_m(\mathcal{F}_2) \subseteq \mathcal{ND}^1_m(\mathcal{F}_1)$ (resp. $\mathcal{ND}^1_m(F_2) \subset \mathcal{ND}^1_m(F_1)$).

Similarly to noisy one-to-one-distinguishability, the noisy one-to-many-distinguishability class of any framework $n(\mathcal{F})$ is equal to the one-to-many-distinguishability class of $\mathcal{F}$.

Theorem 9.8. *For any non-noisy framework $\mathcal{F}$, $\mathcal{ND}^1_m(n(\mathcal{F})) = \mathcal{D}^1_m(\mathcal{F})$.*

Proof.

1. We first show that $\mathcal{ND}^1_m(n(\mathcal{F})) \subseteq \mathcal{D}^1_m(\mathcal{F})$.

   Assume that $\langle B, H, S\rangle \in \mathcal{ND}^1_m(n(ILP_\mathcal{F}))$. Then there is an $n(ILP_\mathcal{F})$ task $T_\mathcal{F}$ with background knowledge B such that $\forall H' \in S$, $pen(H, T_\mathcal{F}) < pen(H', T_\mathcal{F})$. Hence, $\forall H' \in S$, there must be at least one example $e^*_{H'}$ in $T_\mathcal{F}$ that H covers but H' does not. Let $T^2_\mathcal{F}$ be the $ILP_\mathcal{F}$ task with background knowledge B and each of the examples $e^*_{H'}$. Then $H \in ILP_\mathcal{F}(T^2_\mathcal{F})$ and $S \cap ILP_\mathcal{F}(T^2_\mathcal{F}) = \emptyset$. Hence $\langle B, H, S\rangle \in \mathcal{D}^1_m(ILP_\mathcal{F})$.

2. Now we show that $\mathcal{D}^1_m(\mathcal{F}) \subseteq \mathcal{ND}^1_m(n(\mathcal{F}))$.

   Assume that $\langle B, H, S\rangle \in \mathcal{D}^1_m(\mathcal{F})$. Then there is an $ILP_\mathcal{F}$ task $T_\mathcal{F}$ with background knowledge B such that $H \in ILP_\mathcal{F}(T_\mathcal{F})$ and $S \cap ILP_\mathcal{F}(T_\mathcal{F}) = \emptyset$. Let $T^2_\mathcal{F}$ be the $n(ILP_\mathcal{F})$ task constructed from $T_\mathcal{F}$ by adding infinite penalties (and ids) to each example. For any H^*, $H^* \in ILP_\mathcal{F}(T_\mathcal{F})$ if and only if H^* covers every example in $T_\mathcal{F}$, which is true if and only if $pen(H^*, T^2_\mathcal{F})$ is finite. Hence $pen(H, T^2_\mathcal{F})$ is finite, and $\forall H' \in S$, $pen(H', T^2_\mathcal{F})$ is infinite (as there is at least one example in $T^2_\mathcal{F}$ that H' does not cover). Hence $pen(H, T^2_\mathcal{F}) < pen(S, T^2_\mathcal{F})$. So $\langle B, H, S\rangle \in \mathcal{ND}^1_m(n(ILP_\mathcal{F}))$.

□

As we have already proven the one-to-many-distinguishability classes of the six frameworks in Chapter 5, this result leads to the following ordering of noisy one-to-one-distinguishability classes.

Corollary 9.9.

1. $\mathcal{ND}^1_m(n(ILP_b)) \subset \mathcal{ND}^1_m(n(ILP_{sm})) \subset \mathcal{ND}^1_m(n(ILP_{LAS})) \subset \mathcal{ND}^1_m(n(ILP_{LOAS})) \subset \mathcal{ND}^1_m(ILP^{noise}_{LOAS})$
2. $\mathcal{ND}^1_m(n(ILP_c)) \subset \mathcal{ND}^1_m(n(ILP_{LAS}))$

We can again upgrade the notion of $\mathcal{D}^m_m$ to the noisy setting.

Definition 9.6. The *noisy many-to-many-distinguishability class* of a noisy learning framework $\mathcal{F}$ (denoted $\mathcal{ND}^m_m(\mathcal{F})$) is the set of all tuples $\langle B, S_1, S_2\rangle$, where B is a program and S_1 and S_2 are sets of hypotheses for which there is a task $T_{\mathcal{F}}$, with background knowledge B, such that $\forall H_1 \in S_1$, $\forall H_2 \in S_2$, $pen(H_1, T_{\mathcal{F}}) < pen(H_2, T_{\mathcal{F}})$. Given two frameworks, $\mathcal{F}_1$ and $\mathcal{F}_2$, we say that $\mathcal{F}_1$ is at least as (resp. more) $\mathcal{ND}^m_m$-general as (resp. than) $\mathcal{F}_2$ if and only if $\mathcal{ND}^m_m(\mathcal{F}_2) \subseteq \mathcal{ND}^m_m(\mathcal{F}_1)$ (resp. $\mathcal{ND}^m_m(\mathcal{F}_2) \subset \mathcal{ND}^m_m(\mathcal{F}_1)$).

Unlike the previous two distinguishability classes, the noisy many-to-many-distinguishability class is not guaranteed to be equal to its non-noisy counterpart.

Example 9.10. *Take for instance the tuple* $t = \langle \emptyset, \{\{\texttt{heads.}\}, \{\texttt{tails.}\}\}, \{\{\texttt{1\{heads, tails\}1.}\}\}\rangle$. $t \notin \mathcal{D}^m_m(ILP_{LAS})$. *However, we will now show that* $t \in \mathcal{ND}^m_m(n(ILP_{LAS}))$.

Consider the (weighted) partial interpretations $E^- = \{\langle \texttt{id1}, 1, \langle \{\texttt{heads}\}, \emptyset\rangle\rangle, \langle \texttt{id2}, 1, \langle \{\texttt{tails}\}, \emptyset\rangle\rangle\}$. *Both hypotheses containing single facts accept exactly one of the examples in* E^-*; whereas the choice rule accepts both. Hence, if we consider the* $n(ILP_{LAS})$ *task* $T = \langle \emptyset, \langle \emptyset, E^-\rangle\rangle$ *then* $pen(\{\texttt{heads.}\}, T) = pen(\{\texttt{tails.}\}, T) = 1$ *and* $pen(\{\texttt{1\{heads, tails\}1.}\}, T) = 2$. *Thus, t is in* $\mathcal{ND}^m_m(n(ILP_{LAS}))$.

The fact that noisy many-to-many-distinguishability classes are not necessarily equal to their non-noisy counterparts raises the question of whether the noisy many-to-many-generality relation is the same for the six frameworks considered in Chapter 5. In fact, it is the same. The proof is simple for most of the pairs of frameworks, as the examples can be directly translated to the examples of the next framework in the chain. The difficulty comes with mapping ILP_b to ILP_{sm}. Although the non-noisy ILP_b examples of any task can be represented as a single partial interpretation example in ILP_{sm}, this does not work in general for $n(ILP_b)$ and $n(ILP_{sm})$. This is because the penalties for $n(ILP_b)$ are on the single atom examples, whereas in $n(ILP_{sm})$ the penalties are on each partial interpretation. Nevertheless, we can still show that $\mathcal{ND}(n(ILP_b)) \subset \mathcal{ND}(n(ILP_{sm}))$.

Theorem 9.11.

1. $\mathcal{ND}^m_m(n(ILP_b)) \subset \mathcal{ND}^m_m(n(ILP_{sm})) \subset \mathcal{ND}^m_m(n(ILP_{LAS})) \subset \mathcal{ND}^m_m(n(ILP_{LOAS})) \subset \mathcal{ND}^m_m(ILP^{noise}_{LOAS})$
2. $\mathcal{ND}^m_m(n(ILP_c)) \subset \mathcal{ND}^m_m(n(ILP_{LAS}))$

Proof. The fact that the classes are not equal follows from Corollary 9.9. (If any pair of classes were equal then the corresponding $\mathcal{ND}^1_m$ classes would also be equal). Thus it remains to show the $\subseteq$ relation for each pair. In each case, we show this by proving that an arbitrary element of the first framework's many-to-many-distinguishability class is also a member of the second framework's many-to-many-distinguishability class.

1. • Consider any element $\langle B, S_1, S_2\rangle \in \mathcal{ND}^m_m(n(ILP_b))$. There must be some task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $\forall H_1 \in S_1, \forall H_2 \in S_2 : pen(H_1, T_b) < pen(H_2, T_b)$. Hence, there must be a positive integer n such that $\forall H_1 \in S_1 : pen(H_1, T_b) < n$ and $\forall H_2 \in S_2 : pen(H_2, T_b) \geq n$.

   Let $T = \langle B, E_{sm}\rangle$, where E_{sm} is the set of all weighted $n(ILP_{sm})$ examples (all with penalty 1) with partial interpretation $\langle e^{inc}, e^{exc}\rangle$ such that:

(a) $e^{inc} \cup e^{exc} = E^{+}_{atoms} \cup E^{-}_{atoms}$, where E^{+}_{atoms} (resp. E^{-}_{atoms}) is the set consisting of all atoms $\mathtt{a}$ for which there is a tuple $\langle e_{id}, e_{pen}, \mathtt{a}\rangle \in E^{+}$ (resp. E^{-}).

(b) $\left(\sum\limits_{\mathtt{a} \in E^{+}_{atoms} \cap e^{exc}} \mathtt{a}^{+}_{\mathtt{pen}} + \sum\limits_{a \in E^{-}_{atoms} \cap e^{inc}} \mathtt{a}^{-}_{\mathtt{pen}}\right) < n$

(where for each atom $\mathtt{a} \in E^{+}_{atoms} \cup E^{-}_{atoms}$, $\mathtt{a}^{+}_{\mathtt{pen}}$ (resp. $\mathtt{a}^{-}_{\mathtt{pen}}$) is the sum of all penalties for $\mathtt{a}$ that occur in E^{+} (resp. E^{-})).

For any hypothesis H, $pen(H, T_b) < n$ if and only if $B \cup H$ accepts at least one example in E_{sm}. Hence $\forall H_1 \in S_1$, $pen(H_1, T) < |E_{sm}|$ and $\forall H_2 \in S_2$, $pen(H_2, T) = |E_{sm}|$. Thus, $\forall H_1 \in S_1$, $\forall H_2 \in S_2$, $pen(H_1, T) < pen(H_2, T)$.

Hence $\langle B, S_1, S_2\rangle \in \mathcal{ND}^{m}_{m}(n(ILP_{sm}))$.

- Assume $\langle B, S_1, S_2\rangle \in \mathcal{ND}^{m}_{m}(n(ILP_{sm}))$. Then there is an $n(ILP_{sm})$ task $T_1 = \langle B, \langle E\rangle\rangle$ such that $\forall H_1 \in S_1, \forall H_2 \in S_2$, $pen(H_1, T_1) < pen(H_2, T_1)$. Let T_2 be the $n(ILP_{LAS})$ task $\langle B, \langle E, \emptyset\rangle\rangle$. Then $\forall H_1 \in S_1, \forall H_2 \in S_2$, $pen(H_1, T_2) < pen(H_2, T_2)$ (as $\forall H$, $pen(H, T_1) = pen(H, T_2)$). Hence, $\langle B, S_1, S_2\rangle \in \mathcal{ND}^{m}_{m}(n(ILP_{LAS}))$

- Assume $\langle B, S_1, S_2\rangle \in \mathcal{ND}^{m}_{m}(n(ILP_{LAS}))$. Then there is an $n(ILP_{LAS})$ task $T_1 = \langle B, \langle E^{+}, E^{-}\rangle\rangle$ such that $\forall H_1 \in S_1, \forall H_2 \in S_2$, $pen(H_1, T_1) < pen(H_2, T_1)$. Let T_2 be the $n(ILP_{LOAS})$ task $\langle B, \langle E^{+}, E^{-}, \emptyset, \emptyset\rangle\rangle$. Then $\forall H_1 \in S_1, \forall H_2 \in S_2$, $pen(H_1, T_2) < pen(H_2, T_2)$ (as $\forall H$, $pen(H, T_1) = pen(H, T_2)$). Hence, $\langle B, S_1, S_2\rangle \in \mathcal{ND}^{m}_{m}(n(ILP_{LOAS}))$

- Similarly to above, any $n(ILP_{LOAS})$ task that distinguishes a set of hypotheses S_1 from a set of hypotheses S_2 (wrt some background knowledge B) can be mapped into an ILP^{noise}_{LOAS} task that scores every hypothesis the same as the original task (by adding empty contexts to each example). Hence any tuple in $\mathcal{ND}^{m}_{m}(n(ILP_{LOAS}))$ is also in $\mathcal{ND}^{m}_{m}(ILP^{noise}_{LOAS})$

2. Assume $\langle B, S_1, S_2\rangle \in \mathcal{ND}^{m}_{m}(n(ILP_{c}))$. Then there is an $n(ILP_{c})$ task $T_1 = \langle B, \langle E^{+}, E^{-}\rangle\rangle$, such that $\forall H_1 \in S_1, \forall H_2 \in S_2$, $pen(H_1, T_1) < pen(H_2, T_1)$. Let `extra_id` be an example identifier that does not occur in T_1 and consider the $n(ILP_{LAS})$ task $\langle B, \langle\{\langle \mathtt{extra\_id}, \infty, \langle\emptyset, \emptyset\rangle\rangle\}$, $\{\langle e_{id}, e_{pen}, \langle\emptyset, \{\mathtt{a}\}\rangle\rangle \mid \langle e_{id}, e_{pen}, \mathtt{a}\rangle \in E^{+}\} \cup \{\langle e_{id}, e_{pen}, \langle\{\mathtt{a}\}, \emptyset\rangle\rangle \mid \langle e_{id}, e_{pen}, \mathtt{a}\rangle \in E^{-}\}\rangle\rangle$. For any hypothesis H, $pen(H, T_1) = pen(H, T_2)$, and hence, $\forall H_1 \in S_1, \forall H_2 \in S_2$, $pen(H_1, T_2) < pen(H_2, T_2)$. Hence, $\langle B, S_1, S_2\rangle \in \mathcal{ND}^{m}_{m}(n(ILP_{LAS}))$.

$\square$

We have now shown that for each of the three measures of generality, the six noisy frameworks are ordered in the same way as was the case for the six non-noisy frameworks in Chapter 5.

9.4 Solving ILP^{noise}_{LOAS} Tasks with ILASP2 and ILASP2i

The immediate question that may come to mind is whether it is possible to use existing ILASP algorithms to solve noisy tasks. In this section, we demonstrate that both ILASP2 and ILASP2i can indeed be slightly modified in order to solve ILP^{noise}_{LOAS} tasks. Even so, we argue that neither would perform particularly well on noisy tasks, as their algorithms were originally designed for tasks with no noise.

9.4.1 ILASP2

ILASP2 can be extended to solve ILP^{noise}_{LOAS} tasks by slightly modifying the meta-level program $\mathcal{M}_{ILASP2}$. Given an ILP^{noise}_{LOAS} task T and a set of violating reasons VR, the modification consists of the following steps:

1. For each example eg with finite penalty, a choice rule $\texttt{0\{noisy(eg}_{\texttt{id}}\texttt{)\}1}$ and a weak constraint $:\sim$ $\texttt{noisy(eg}_{\texttt{id}}\texttt{)}.[2 \times \texttt{e}_{\texttt{pen}}\texttt{@1}]$ are added to $\mathcal{M}_{ILASP2}(T, VR)$.
2. Each constraint "$\texttt{:- not cov(e}_{\texttt{id}}\texttt{, e}_{\texttt{id}}\texttt{)}$." in $\mathcal{M}(T)$ is appended with the literal $\texttt{not noisy(e}_{\texttt{id}}\texttt{)}$ (indicating that if the penalty of the example has been paid then it does not need to be covered).
3. Similarly each constraint "$\texttt{:- not ord\_respected(o}_{\texttt{id}}\texttt{, o}_{\texttt{id1}}\texttt{, o}_{\texttt{id2}}\texttt{)}$." in $\mathcal{M}(T)$ is appended with the literal $\texttt{not noisy(o}_{\texttt{id}}\texttt{)}$.
4. Each rule "$\texttt{v}_{\texttt{i}}\texttt{(e}_{\texttt{id}}\texttt{) :- cov(e}_{\texttt{id}}\texttt{, v1)}$." in $\mathcal{M}(T)$ is appended with the literal $\texttt{not noisy(e}_{\texttt{id}}\texttt{)}$.
5. Each rule "$\texttt{v}_{\texttt{p}}\texttt{(o}_{\texttt{id}}\texttt{) :- ord\_respected(o}_{\texttt{id}}\texttt{, v1, v2)}$." in $\mathcal{M}(T)$ is appended with the literal $\texttt{not noisy(o}_{\texttt{id}}\texttt{)}$.
6. The sub-program $\left\{\texttt{:- not not\_as(I}_{\texttt{id}}\texttt{).} \;\middle|\; \langle I, e\rangle \in VR, e \in E^{-}\right\}$ is replaced with the sub-program $\left\{\texttt{:- not not\_as(I}_{\texttt{id}}\texttt{), not noisy(e}_{\texttt{id}}\texttt{).} \;\middle|\; \langle I, e\rangle \in VR, e \in E^{-}\right\}$.
7. The sub-program $\left\{\texttt{:- vp\_not\_resp(VP}_{\texttt{id}}\texttt{).} \;\middle|\; \langle VP, o\rangle \in VR, o \in O^{c}\right\}$ is replaced with the sub-program $\left\{\texttt{:- vp\_not\_resp(VP}_{\texttt{id}}\texttt{), not noisy(o}_{\texttt{id}}\texttt{).} \;\middle|\; \langle VP, o\rangle \in VR, o \in O^{c}\right\}$.

The intuition behind this modification is that every example eg which is permitted not to be covered (i.e. those with finite penalties) is assigned a new atom $\texttt{noisy(eg}_{\texttt{id}}\texttt{)}$ in the meta-level program. If this atom is true in a meta-level answer set, then anything to do with this example (including previous violating reasons associated with this example) may be ignored. Due to the extra weak constraints any answer set A of the meta-level program will have the optimisation score (at level 1) $2 \times \mathcal{S}(\mathcal{M}^{-1}_{hyp}(A), T)$ if A contains $\texttt{violating}$ and $2 \times \mathcal{S}(\mathcal{M}^{-1}_{hyp}(A), T) + 1$ if not. This has the effect that ILASP2 will now find the optimal inductive solutions of the ILP^{noise}_{LOAS} task T.

Although we have demonstrated that ILASP2 can solve ILP^{noise}_{LOAS} tasks, the main issue with using ILASP2 for solving these tasks is that when examples have noise, more examples are often necessary in order to learn an accurate hypothesis. This is an issue because ILASP2 scales poorly with respect to the number of examples, as discussed in Chapter 7.

9.4.2 ILASP2i

As discused in Chapter 7, ILASP2i addresses some of the scalability issues in ILASP2 as it often allows ILASP2 to be called with a much smaller *relevant* set of examples. ILASP2i could be extended to address ILP^{noise}_{LOAS} tasks in the following way. Firstly, the call to ILASP2 should be replaced with a call to the modified ILASP2 introduced in the previous section (so that ILASP2 can handle the noisy examples). The only other change that is needed is to the call to *findRelevantExamples*. Originally *findRelevantExamples* was called on the full set of examples E. However, if ILASP2 returns a hypothesis that does not cover an example that has already been added to the set of relevant examples (as is possible now that ILASP2 is allowed to ignore examples that it considers noisy), this will cause ILASP2i to go into an infinite loop, as it will continually add the same example to the set of relevant examples. This can be overcome with a simple modification: the call to *findRelevantExamples* should use $E\backslash Relevant$ (where *Relevant* is the set of relevant examples), rather than the full set of examples E.

We have shown in Chapter 8 that ILASP2i can give a large performance boost to learning tasks with large numbers of examples (with respect to ILASP2), as it is often the case that the final set of relevant examples is much smaller than the whole set of examples. But, when considering ILP^{noise}_{LOAS} tasks, this may not be the case, as illustrated by the following example.

Example 9.12.

Let us reconsider the ILP^{noise}_{LOAS} task T from Example 9.1. The ILASP2i algorithm would start with the relevant examples being empty and the hypothesis $H = \emptyset$. Examples 51 to 98 are covered, but examples 1 to 50 are not, so ILASP2i would pick an uncovered example (the first) and add it to the relevant example set. The call to ILASP2 might then find the hypothesis $H' = \{$`p.`$\}$, which covers the only relevant example. As this hypothesis does not cover examples 51 to 98, one of these uncovered examples would be added to the set of relevant examples. Now, every hypothesis is guaranteed to cover one of the relevant examples and not cover the other. Hence, the next call to ILASP2 would then return $H = \emptyset$ as the next hypothesis, which means that example with id 2 would be added to the relevant set. This would continue until all of the examples with ids 1 to 49 and 51 to 98 had been added. At this point the hypothesis H' would again be returned by the call to ILASP2, and as there are no examples that are not covered by H' that are not already in the relevant example set, H' would be returned by ILASP2i.

Example 9.12 demonstrates that for ILP^{noise}_{LOAS} tasks, the set of relevant examples may not be significantly smaller than the full set of examples. When this is the case, ILASP2i may even be slower than ILASP2, as all iterations before the last iteration are essentially wasted computations. In the next chapter, we introduce a new algorithm, ILASP3, that is able to efficiently solve ILP^{noise}_{LOAS} tasks with large numbers of examples. In Chapter 11 we then demonstrate that ILASP3 indeed significantly outperforms both (the extended) ILASP2 and ILASP2i on noisy tasks with large numbers of examples.

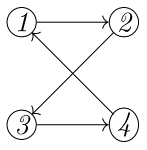
9.5 Related Work

Most ILP systems have been designed for the task of learning from example atoms. In order to search for best hypotheses, such systems normally use a scoring function, defined in terms of the coverage of the examples and the length of the hypothesis (e.g. ALEPH [Sri01], Progol [Mug95], and the implementation of XHAIL [BR15b]). When examples are noisy, this scoring function is sometimes combined with a notion of maximum threshold, and the search is not for an optimal solution that minimises the number of uncovered examples, but for a hypothesis that does not fail to cover more than a defined maximum threshold number of examples (e.g. [Sri01, OB10, ACBR13]). In this way, once an acceptable hypothesis (i.e. a hypothesis that covers a sufficient number of examples) is computed the system does not search for a better one. As such, the computational task is simpler, and therefore the time needed to compute a hypothesis is shorter, but the learned hypothesis is not optimal. Furthermore, to guess the "correct" maximum threshold requires some idea of how much noise there is in the given set of examples. For instance, one of the inputs to the HYPER/N [OB10] system is the proportion of noise in the examples. When the proportion of noise is unknown, too small a threshold could result in the learning task being unsatisfiable, or in learning a hypothesis that overfits the data. On the other hand, too high a threshold could result in poor hypothesis accuracy, as the hypothesis may not cover many of the examples. Our ILP^{noise}_{LOAS} framework addresses the problem of computing optimal solutions and in doing so does not require any knowledge a priori of the level of noise in the data.

Another difference when compared to many ILP approaches that support noise is that our examples are partial interpretations. We do not consider penalising individual atoms within these partial interpretations. This is somewhat similar to what traditional ILP approaches do (it is only the notion of examples that is different in the two approaches). In fact, while penalising individual atoms within partial interpretations would certainly be an interesting avenue for future work, Example 9.13 shows that this could be seen as analogous to penalising the arguments of atomic examples in traditional ILP approaches.

Example 9.13. *Consider the problem of learning the definition of what it means for a graph to be Hamiltonian. We showed in Chapter 4 that there are two ways to represent example graphs. A graph G can either be represented as partial interpretations $\langle e^{inc}, e^{exc}\rangle$, where e^{inc} contains the set of edges in G, and e^{inc} contains the set of edges not in G, or it can be represented as a context dependent example $\langle\langle\emptyset,\emptyset\rangle, C\rangle$, where the context C is a set of facts that represent the nodes and edges in G.*

There are also two ways of representing an example graph G in a traditional learning from entailment setting: as a single atom with a list containing the nodes in G and a list containing all the edges in G, and as a clause with the edges in the body.



$$e_{atom} = \texttt{hamiltonian}([1,2,3,4],[(1,2),(2,3),(3,4),(4,1)])$$

$$e_{clause} = \begin{array}{ll} \texttt{hamiltonian:-} & \texttt{node}(1),\ldots,\texttt{node}(4), \\ & \texttt{edge}(1,2),\ldots,\texttt{edge}(4,1). \end{array}$$

If we compare this to the two ILP^{noise}_{LOAS} representations, we can see that the way of scoring a hypothesis is the same in both cases. As hypotheses are scored on the number of examples that are covered, in both frameworks a hypothesis is penalised for the number of graphs that are misclassified. Having penalties on individual atoms in the partial interpretation examples of ILP^{noise}_{LOAS} would mean that individual edges or nodes would be penalised. In traditional ILP_{LFE} this would mean penalising either the arguments of atomic examples, or the body literals of clausal examples. Although potentially interesting, we are not aware of any ILP system that allows for this level of granularity.

Summary

In this chapter we have presented the noisy extensions of our learning frameworks and investigated the complexities and generalities of these frameworks. We have explained why our previous ILASP systems are not well suited to solving noisy tasks. In the next chapter we present ILASP3, which is specifically targeted at learning from noisy examples.

Chapter 10

The ILASP3 Algorithm for Scalable Learning from Noisy Examples

In the previous chapter we presented the ILP^{noise}_{LOAS} framework for learning from noisy examples. We explained why our existing ILASP2 and ILASP2i algorithms are not well suited to solving the tasks of this framework. In this chapter we present ILASP3, an alternative algorithm specifically targeted at solving ILP^{noise}_{LOAS} tasks.

The main intuition of ILASP3 is to incrementally build a function for approximating the coverage of hypotheses. This allows us to approximate the score of any hypothesis. Given any hypothesis H, the approximation function used by ILASP3 will always contain the true coverage of H, but may also contain examples that are not covered by H. This means that for any hypothesis H, the approximate score of H is less than or equal to the real score of H. In each iteration, ILASP performs the following steps:

1. Find an optimal hypothesis H^* with respect to the notion of approximate score. This is a hypothesis H that minimises the sum of $|H|$ plus the penalties for all examples that are not in the approximation of the coverage.
2. Search for an example that is in the approximate coverage of H^* but that is not covered by H^*.
   - If such an example exists, then the approximate coverage of H^* is incorrect, so H^* might not be a true optimal hypothesis. The algorithm therefore updates the approximation function and goes back to step (1).
   - If no such example exists, then H^* is an optimal inductive solution of the task (so ILASP3 terminates returning H^*).

The approximation function used by ILASP3 is based on a notion of *coverage constraints*. A coverage constraint is a pair consisting of a CDPI example and a set of *hypothesis schemas*. A hypothesis schema is a set of structural conditions defining a class of hypotheses; for example a hypothesis schema may define the class of all hypotheses that contain at least one rule from the set $\{\texttt{h}_1, \texttt{h}_2\}$ and no rules from the set $\{\texttt{h}_3, \texttt{h}_4\}$. A similar notion of coverage constraints is used for ordering examples, where the pair consists of a CDOE and a set of schemas called *ordering schemas*. Any hypothesis which obeys every condition of a schema is said to *conform* to that schema.

Definition 10.1. Let T be the ILP^{noise}_{LOAS} task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$, $H \subseteq S_M$ and $\langle ex, SC\rangle$ be a coverage constraint. H is said to *satisfy* $\langle ex, SC\rangle$ if one of the following conditions holds:

- $ex \in E^+$ or $ex \in O^b$ and H conforms to at least one schema in SC.
- $ex \in E^-$ or $ex \in O^c$ and H does not conform to any schema in SC.

If H does not satisfy $\langle ex, SC\rangle$, we say that H *violates* $\langle ex, SC\rangle$.

ILASP3 incrementally builds a set of coverage constraints CC such that for each constraint $\langle ex, SC\rangle \in CC$, every hypothesis that covers ex satisfies $\langle ex, SC\rangle$. In the case that ex is a positive example and SC is empty there is no way of satisfying the coverage constraint. Such a coverage constraint is generated by ILASP3 when ex is impossible to cover (or at least there is no hypothesis in the hypothesis space that covers ex). Conversely if there is a negative example ex such that $\langle ex, \emptyset\rangle$ is in CC then this constraint is automatically satisfied. In practice, ILASP3 only generates coverage constraints for negative examples with a non-empty set of hypothesis schemas.

The coverage constraints CC can be used to define the approximate coverage of a hypothesis. Given a hypothesis H this approximation assumes that examples are covered unless there is evidence that the example is not covered (i.e. a violated coverage constraint). This approximation is formalised in Definition 10.2.

Definition 10.2. Let T be the ILP^{noise}_{LOAS} task $\langle B, S_M, \langle E^+, E^-, O^b, O^b\rangle\rangle$, $H \subseteq S_M$ and CC be a set of coverage constraints. The *approximate coverage* wrt CC, denoted $ApproxCoverage(H, T, CC)$, is the set of examples $ex \in E^+ \cup E^- \cup O^b \cup O^c$ such that there is no coverage constraint $\langle ex, SC\rangle \in CC$ that is violated by H.

We will demonstrate that any set of coverage constraints CC computed in the ILASP3 algorithm is such that for any $H \subseteq S_M$, $Coverage(H, T) \subseteq ApproxCoverage(H, T, CC)$ (where $Coverage(H, T)$ denotes the set of examples in T which are covered by H). The examples which are not in $ApproxCoverage(H, T, CC)$ must therefore not be covered by H. Note that it is not necessarily the case that any example in $ApproxCoverage(H, T, CC)$ *is* covered – it is only that given the coverage constraints CC, we have no evidence that the example is *not* covered. For instance, when ILASP3 begins CC is empty, meaning that for any hypothesis H, $ApproxCoverage(H, T, CC)$ contains every example in T. As the algorithm progresses, more coverage constraints are added to CC, and $ApproxCoverage(H, T, CC)$ gets closer to $Coverage(H, T)$.

We can use the approximation of the coverage of a hypothesis to compute an approximation of the score of that hypothesis. As $ApproxCoverage(H, T, CC)$ is never an under-estimate of the coverage of H (i.e. $Coverage(H, T) \subseteq ApproxCoverage(H, T, CC)$), this approximate score is never an over-estimate of the score of H. We call the approximate score of a hypothesis the *lower bound score* of H wrt CC. This is formalised by the following definition.

Definition 10.3. Let T be the ILP^{noise}_{LOAS} task $\langle B, S_M, \langle E^+, E^-, O^b, O^b \rangle \rangle$, $H \subseteq S_M$ and CC be a set of coverage constraints. Let $NotCov$ be the set $(E^+ \cup E^- \cup O^b \cup O^c) \backslash ApproxCoverage(H, T, CC)$. The *lower bound score* of H wrt CC, denoted as $\mathcal{S}_{lb}(H, T, CC)$, is the sum $|H| + \sum_{e \in NotCov} e_{pen}$. A hypothesis $H \subseteq S_M$ is said to be *optimal with respect* to CC if and only if $\mathcal{S}_{lb}(H, T, CC)$ is finite and there is no $H' \subseteq S_M$ such that $\mathcal{S}_{lb}(H', T, CC) < \mathcal{S}_{lb}(H, T, CC)$.

Note that for any hypothesis H and any set of coverage constraints CC computed by ILASP3, $\mathcal{S}_{lb}(H, T, CC) \leq \mathcal{S}(H, T)$.

When ILASP3 starts, the set of coverage constraints CC is initialised to be empty. In each iteration, the algorithm first finds a hypothesis H^* that is optimal with respect to the current CC. It then searches for an example ex that is in $ApproxCoverage(H^*, T, CC)$ but that is not in $Coverage(H^*, T)$. If such an example exists then $\mathcal{S}_{lb}(H^*, T, CC) < \mathcal{S}(H^*, T)$, so H^* may not be an optimal inductive solution of T. In this case a new coverage constraint, computed for the example ex, is added to CC and a new iteration is started. If no such example exists then $\mathcal{S}_{lb}(H^*, T, CC) = \mathcal{S}(H^*, T)$. Hence, as for any other hypothesis H', $\mathcal{S}_{lb}(H^*, T, CC) \leq \mathcal{S}_{lb}(H', T, CC) \leq \mathcal{S}(H', T)$, it must be the case that H^* is an optimal solution of T. In this case ILASP3 terminates and returns H^*.

ILASP3 is similar to ILASP2i in that it iteratively computes a hypothesis and searches for counter examples to its current hypothesis. When compared with ILASP2i, the main scalability gain in ILASP3 comes from the fact that the task of finding intermediate hypotheses is simpler. In ILASP2i, the grounding of the background knowledge, hypothesis space and the context of each relevant example must be considered. In ILASP3, only the coverage constraints are considered. The only point at which ILASP3 considers the grounding of the background knowledge, hypothesis space, and contexts is when it translates a detected counter example into a new coverage constraint. At this point ILASP3 only needs to consider the specific counter example rather than the set of relevant examples considered by ILASP2i.

In the next section, we formalise the notion of hypothesis schemas that form the main component of our coverage constraints. We then present the methods used for translating detected counter examples into coverage constraints and show how schemas can be represented in meta-level ASP programs. Next we present our method for finding an optimal hypothesis with respect to a set of coverage constraints and formalise the ILASP3 algorithm, together with its soundness and completeness results. After defining the main ILASP3 algorithm, we discuss some extra extensions of ILASP3 that can result in a significant boost in performance on noisy tasks. We conclude the chapter with a discussion of the related work.

10.1 Hypothesis Schemas

The intuition behind a hypothesis schema is to identify a set of constraints on the rules allowed in a hypothesis such that a hypothesis conforms to this set of constraints only if it covers a particular example. Recall that each rule h in the hypothesis space is associated with the unique identifier $\mathtt{h_{id}}$. Informally, a hypothesis schema can be thought of as a set of *disjunctions* over the rule ids and one negated disjunction of rule ids. Each disjunction of rule ids in the set captures the fact that for at least one of the ids, the corresponding rule must be in a hypothesis in order for the hypothesis to conform to the schema. The negated disjunction of rule ids expresses that for each id in this negated disjunction, the corresponding rule should not be in a hypothesis. This is formally expressed in the following definition, using the notion of a *rule-disjunction.* Given a hypothesis space S_M, a *rule-disjunction* d is a subset of $ids(S_M)$, and a hypothesis $H \subseteq S_M$ is said to *satisfy* d if and only if $d \cap ids(H) \neq \emptyset$.

Definition 10.4. Let S_M be a hypothesis space. A *hypothesis schema* is a pair $sc = \langle D, V\rangle$, where D is a set of rule-disjunctions and V is a single rule-disjunction. A hypothesis H is said to conform to sc if and only if $\forall d \in D, H$ satisfies d (i.e. $d \cap ids(H) \neq \emptyset$), and H does not satisfy V (i.e. $V \cap ids(H) = \emptyset$).

This notion of a hypothesis schema is related to the definition of an answer set based on unfounded sets (Definition 2.4). Specifically, given an interpretation I, a background knowledge B and a hypothesis space S_M, it is possible to generate a hypothesis schema $sc = \langle D, V\rangle$ such that any $H \subseteq S_M$ conforms to sc if and only if I is an answer set of $B \cup H$. Each non-empty unfounded subset of I with respect to B can be used to generate a rule-disjunction containing the identifiers of the rules in S_M that, added to B, would prevent I from being unfounded. These rule-disjunctions form the set D of a hypothesis schema. The set V of a schema is the set of all rules that prevent a particular interpretation I from being a model. Requiring that a hypothesis conforms to a schema constructed from an interpretation I, essentially corresponds to requiring that I is a model of $B \cup H$ and has no non-empty unfounded subsets (i.e. requiring that I is an answer set of $B \cup H$). We illustrate this correspondence between hypothesis schemas and unfounded sets in the following example.

Example 10.1. *Let $B = \emptyset$ and S_M be the following hypothesis space:*

$$\left\{\begin{array}{ll}
\mathtt{h_1:} & \mathtt{p.}\\
\mathtt{h_2:} & \mathtt{q.}\\
\mathtt{h_3:} & \mathtt{p:\text{-}q.}\\
\mathtt{h_4:} & \mathtt{p:\text{-}\ not\ q.}\\
\mathtt{h_5:} & \mathtt{q:\text{-}p.}\\
\mathtt{h_6:} & \mathtt{q:\text{-}\ not\ p.}\\
\mathtt{h_7:} & \mathtt{r.}
\end{array}\right\}$$

Let sc be the hypothesis schema $\langle D, V\rangle$, where $D = \{\{\mathtt{h_1}, \mathtt{h_4}\}, \{\mathtt{h_7}\}\}$ and $V = \{\mathtt{h_2}, \mathtt{h_5}\}$. To conform to sc, a hypothesis H must contain at least one of the rules $\mathtt{h_1}$ or $\mathtt{h_4}$ and contain $\mathtt{h_7}$ and contain neither

$\mathtt{h}_2$ *nor* $\mathtt{h}_5$*. A hypothesis* H *that conforms to sc would correspond to having* $I = \{\mathtt{p}, \mathtt{r}\}$ *as an answer set of* $B \cup H$*. In fact, the two rule-disjunctions in* D *come from the two subsets of* I *that could be unfounded with respect to some hypotheses in the hypothesis space:* $\{\mathtt{p}\}$ *is unfounded with respect to* $B \cup H$ *if* H *does not contain either* $\mathtt{h}_1$ *or* $\mathtt{h}_4$*; and* $\{\mathtt{r}\}$ *is unfounded with respect to* $B \cup H$ *if* H *does not contain* $\mathtt{h}_7$*. If neither of the two rule-disjunctions are "violated" then there are no other possible unfounded sets. Finally, for* I *to be an answer set of* $B \cup H$*,* I *has to be a model of* $B \cup H$*. It is a model if and only if* H *does not contain either* $\mathtt{h}_2$ *or* $\mathtt{h}_5$*. Hence,* I *is an answer set of* $B \cup H$ *if and only if* H *conforms to sc.*

10.1.1 Ordering Schemas

The hypothesis schemas presented so far are useful in expressing which hypotheses accept particular answer sets, and therefore which positive or negative examples are covered. In order to express which ordering examples are covered, we introduce a second type of schema called an *ordering schema*. Recall that a (context-dependent) ordering example $o = \langle e^1, e^2, op\rangle$ specifies the preference relation between answer sets of the learned program. In the case of a context-dependent brave ordering, there must be a pair of accepting answer sets $\langle A_1, A_2\rangle$ of o – meaning that (i) A_1 is an accepting answer set of e^1; (ii) A_2 is an accepting answer set of e^2; and (iii) $\langle A_1, A_2, op\rangle \in ord(B \cup H, AS(B \cup H \cup e^1_{ctx}) \cup AS(B \cup H \cup e^2_{ctx}))$. The first two of these conditions can be ensured using the hypothesis schemas presented in the previous section. Condition (iii) motivates a generalisation of these hypothesis schemas, which depends on the new notion of *hypothesis weightings*.

Just as hypothesis schemas were based on the notion of unfounded sets and models, which can be used to define what it means to be an answer set, hypothesis weightings are based on a notion of *optimisation differences*, which can be used to define what it means for one answer set to be preferred to another. Definition 10.5 introduces the notion of optimisation differences.

Definition 10.5. Let P be an ASP program and I_1 and I_2 be interpretations. For any integer l, the *optimisation difference* between I_1 and I_2 at l with respect to P (denoted $\Delta^P_l(I_1, I_2)$) is equal to $P^{I_1}_l - P^{I_2}_l$.

Definition 10.5 defines optimisation differences for programs. We also define optimisation differences for single weak constraints. For any weak constraint W and any interpretations I_1 and I_2, $\Delta^W_l(I_1, I_2)$ is defined as $\Delta^{\{W\}}_l(I_1, I_2)$, where $\{W\}$ is the program containing only W. The optimisation difference of a weak constraint is crucial to our notion of ordering schemas. In order to use the optimisation differences of individual weak constraints, we must slightly restrict the language of weak constraints in our learning tasks. Definition 10.6 introduces the notion of a set of weak constraints being *independent* from each other.

Definition 10.6. Let WC be a set of weak constraints. The weak constraints in WC are *independent* if there are no two weak constraints W_1 and W_2 in WC with ground instances W^g_1 and W^g_2 such that the tail of W^g_1 is equal to the tail of W^g_2.

In the rest of this thesis, we assume that for any learning task, the weak constraints in $B \cup S_M$ are independent. In practice, this can be achieved by adding a unique constraint to each weak constraint as the first term occurring in the tail after the weight and priority level. The benefit of this assumption is that we can now compute the optimisation difference (between two interpretations) with respect to a program by summing the optimisation difference with respect to each weak constraint in the program. This result is formalised in Lemma 10.2.

Lemma 10.2. *(proof on page 309)*

Let P be any ASP program whose weak constraints are independent, l be any integer and I_1 and I_2 be any interpretations. Then $\Delta^P_l(I_1, I_2) = \sum\limits_{W \in weak(P)} \Delta^W_l(I_1, I_2)$.

When learning, this result means that we can compute the optimisation difference of $B \cup H$ by summing the optimisation differences for each of the weak constraints in the learned program. Given a pair of interpretations, we can compute the individual optimisation difference for each weak constraint in the hypothesis space once, and can compute the optimisation difference for any hypothesis by summing the optimisation differences of the weak constraints in the hypothesis (and background knowledge). The following definition introduces the notion of the *deciding optimisation difference*, which can be used to check the ordering between two interpretations, given a set of weak constraints.

Definition 10.7. Let P be any program and I_1 and I_2 be any interpretations. Let L be the set of integers l such that $\Delta^P_l(I_1, I_2) \neq 0$. We define the *deciding optimisation difference* between I_1 and I_2 with respect to P (denoted $\Delta^P(I_1, I_2)$) as follows:

- If $L = \emptyset$ then $\Delta^P(I_1, I_2) = 0$
- If $L \neq \emptyset$ then $\Delta^P(I_1, I_2) = \Delta^P_l(I_1, I_2)$, where l is the maximum of L

Lemma 10.3. *(proof on page 309)*

Let S be a set of interpretations and P be an ASP program. Given any pair of interpretations I_1 and I_2 in S and any binary operator $op \in \{<, >, \leq, \geq, =, \neq\}$, $\langle I_1, I_2, op\rangle \in ord(P, S)$ if and only if $\Delta^P(I_1, I_2)\ op\ 0$.

Example 10.4. *Consider the program P and the interpretations I_1 and I_2.*

$$P = \left\{ \begin{array}{l} \texttt{0\{p(1),p(2),p(3)\}3.} \\ \texttt{:}\sim \texttt{p(X).[1@2,X,1]} \\ \texttt{:}\sim \texttt{p(X).[X@1,X,2]} \end{array} \right\} \qquad \begin{array}{l} I_1 = \{\texttt{p(1)},\ \texttt{p(2)}\} \\ \\ I_2 = \{\texttt{p(2)},\ \texttt{p(3)}\} \end{array}$$

Let the two weak constraints be denoted as W_1 and W_2, respectively. W_1 represents that the highest priority in the program is to minimise the number of `p` *atoms in an answer set (as a penalty of 1 is paid for each* `p` *atom). Given this weak constraint, both interpretations I_1 and I_2 pay the same penalty.*

As there are no other weak constraints in P at priority level 2, we should expect the optimisation difference $\Delta_2^P(I_1, I_2)$ to equal 0. We can calculate $\Delta_2^P(I_1, I_2)$ as follows:

$$\begin{aligned}\Delta_2^P(I_1, I_2) &= \Delta_2^{W_1}(I_1, I_2) + \Delta_2^{W_2}(I_1, I_2)\\ &= (\{W_1\}_2^{I_1} - \{W_1\}_2^{I_2}) + (\{W_2\}_2^{I_1} - \{W_2\}_2^{I_2})\\ &= (2-2) + (0-0)\\ &= 0\end{aligned}$$

Note that for each weak constraint W_i, $\{W_i\}$ represents the program containing only W_i, and recall that $\{W_i\}_{lev}^I$ is the sum $\sum_{(\mathtt{wt},\mathtt{lev},\mathtt{t_1}\ldots,\mathtt{t_n}) \in weak(\{W_i\},I)} \mathtt{wt}$. I.e. $\{W_i\}_{lev}^I$ is the sum of the first elements of the tuples in $weak(\{W_i\}, I)$ whose second element is the priority level `lev`. For example, as $weak(\{W_1\}, I_1) = \{(\mathtt{1},\mathtt{2},\mathtt{1},\mathtt{1}), (\mathtt{1},\mathtt{2},\mathtt{2},\mathtt{1})\}$, $\{W_1\}_2^{I_1} = 1 + 1 = 2$. Similarly, $\{W_1\}_1^{I_1} = 0$, as there are no elements of $weak(\{W_1\}, I)$ whose second element is 1.

The weak constraint W_2 corresponds to preferring answer sets where the sum of the arguments of the `p`'s is as small as possible. A penalty of `X` is paid for each `p(X)` in an answer set. This means that I_1 pays a penalty of 3, whereas I_2 pays a penalty of 5. So I_2's penalty at level 1 is 2 less than I_1's. We would therefore expect $\Delta_1^P(I_1, I_2)$ to equal -2. We can calculate Δ_1^P as follows:

$$\begin{aligned}\Delta_1^P(I_1, I_2) &= \Delta_1^{W_1}(I_1, I_2) + \Delta_1^{W_2}(I_1, I_2)\\ &= (\{W_1\}_1^{I_1} - \{W_1\}_1^{I_2}) + (\{W_2\}_1^{I_1} - \{W_2\}_1^{I_2})\\ &= (0-0) + (3-5)\\ &= -2\end{aligned}$$

As $\Delta_2^P(I_1, I_2) = 0$, the deciding optimisation difference ($\Delta^P(I_1, I_2)$) is -2. As $\Delta^P(I_1, I_2) < 0$, this means that $\langle I_1, I_2, < \rangle \in ord(P)$. Hence I_1 is preferred to I_2.

We now introduce *hypothesis weightings*, which are the fundamental new part of ordering schemas. A hypothesis weighting can be thought of as a function from a hypothesis to an integer, expressing the optimisation difference between two interpretations at a given level. The idea is that we can transform a pair of answer sets into an ordered list ws of weightings $[\omega_1, \ldots, \omega_n]$, one for each priority level in the hypothesis space, such that each weighting can be used to compute the optimisation difference at that level; thus, we can compute the deciding optimisation difference of the two answer sets. Similarly to hypothesis schemas, the structure of the weak constraints is "compiled away", so that when we are searching for the best hypothesis, we only need to reason about the optimisation differences. A hypothesis weighting ω consists of two components: ω^f, which is a mapping from $ids(S_M)$ to $\mathbb{Z}$ and ω^k, which is a constant that is used to represent the optimisation difference of weak constraints in the background knowledge.

Definition 10.8. Let S_M be a hypothesis space. A *hypothesis weighting* ω is a pair $\langle \omega^f, \omega^k \rangle$, where ω^f is a mapping from $ids(S_M)$ onto $\mathbb{Z}$ and ω^k is an integer. Given a hypothesis $H \subseteq S_M$, we write $\omega[H]$ to denote the sum $\omega^k + \sum\limits_{h \in H} \omega^f(\mathtt{h_{id}})$.

Notation ($[\mathtt{id_1} : \lambda_1, \ldots, \mathtt{id_n} : \lambda_n]$). Let $\{\mathtt{id_1}, \ldots, \mathtt{id_n}\}$ be a set of rule identifiers and $\{\lambda_1, \ldots, \lambda_n\}$ be a set of integers. $[\mathtt{id_1} : \lambda_1, \ldots, \mathtt{id_n} : \lambda_n]$ represents a mapping f such that for each $i \in [1, n]$, $f(\mathtt{id_i}) = \lambda_i$, and for each rule identifier $\mathtt{id}$ such that $\mathtt{id} \notin \{\mathtt{id_1}, \ldots, \mathtt{id_n}\}$, $f(\mathtt{id}) = 0$.

Example 10.5. *Consider the background knowledge B, hypothesis space S_M and interpretations I_1 and I_2.*

$$B = \left\{ \begin{array}{l} \mathtt{0\{p(1), p(2), p(3)\}3.} \\ \mathtt{0\{q(1), q(2)\}2.} \\ \mathtt{:\sim p(X).[1@2, X, b1]} \\ \mathtt{:\sim p(X).[X@1, X, b2]} \end{array} \right\}$$

$$S_M = \left\{ \begin{array}{ll} \mathtt{h_1}: & \mathtt{:\sim q(X).[1@2, X, 1]} \\ \mathtt{h_2}: & \mathtt{:\sim q(X).[X@2, X, 2]} \\ \mathtt{h_3}: & \mathtt{:\sim q(X),\ not\ p(X).[1@2, X, 3]} \\ \mathtt{h_4}: & \mathtt{:\sim q(X),\ not\ p(X).[X@2, X, 4]} \\ \mathtt{h_5}: & \mathtt{:\sim q(X).[1@1, X, 5]} \\ \mathtt{h_6}: & \mathtt{:\sim q(X).[X@1, X, 6]} \\ \mathtt{h_7}: & \mathtt{:\sim q(X),\ not\ p(X).[1@1, X, 7]} \\ \mathtt{h_8}: & \mathtt{:\sim q(X),\ not\ p(X).[X@1, X, 8]} \end{array} \right\}$$

$I_1 = \{\mathtt{p(1)},\ \mathtt{p(2)},\ \mathtt{q(1)},\ \mathtt{q(3)}\}$ $\qquad$ $I_2 = \{\mathtt{p(2)},\ \mathtt{p(3)},\ \mathtt{q(1)},\ \mathtt{q(2)}\}$

Consider the hypothesis weightings $\omega_2 = \langle [\mathtt{h_2} : 1, \mathtt{h_4} : 2], 0 \rangle$ and $\omega_1 = \langle [\mathtt{h_6} : 1, \mathtt{h_8} : 2], -2 \rangle$. For $i \in \{1, 2\}$, and each $h \in S_M$, $\omega_i^f(\mathtt{h_{id}}) = \Delta_i^h(I_1, I_2)$. Hence, as for each $i \in \{1, 2\}$, $\Delta_i^B(I_1, I_2) = \omega_i^k$, we can use ω_1 and ω_2 to compute the optimisation differences of $B \cup H$ for any $H \subseteq S_M$. More specifically, for any $H \subseteq S_M$, and any $i \in \{1, 2\}$, $\omega_i[H] = \Delta_i^{B \cup H}(I_1, I_2)$.

Hence for any operator op, the set of hypotheses H such that $\langle I_1, I_2, op \rangle \in ord(B \cup H)$ are those such that $I_1, I_2 \in AS(B \cup H)$ and x op 0, where x is the first non-zero element of $[\omega_2[H], \omega_1[H]]$ (or 0 if $\omega_2[H] = \omega_1[H] = 0$).

Given a hypothesis H, we can use an ordered list of hypothesis weightings to compute the list of optimisation differences for the answer sets A_1 and A_2. As we are interested in computing the deciding optimisation difference, we introduce a notation for the first non-zero element of an ordered list.

Notation ($\uparrow$). Let L be an ordered list of integers. If L contains at least one non-zero element then $\uparrow(L)$ is equal to the first non-zero element in L. If every element of L is equal to zero, then $\uparrow(L) = 0$.

We can now define *ordering schemas*. An ordering schema consists of three elements: a hypothesis schema, which is used to ensure that a pair of interpretations are both answer sets of the learned program; an ordered list of hypothesis weightings, which can be used to compute the optimisation differences of the pair of interpretations with respect to the weak constraints in the learned program; and finally, a comparison operator. The last two components are used to ensure that the ordering of the pair of interpretations respects the operator (with respect to the learned program).

Definition 10.9. Let S_M be a hypothesis space. An *ordering schema osc* is a tuple $\langle sc, ws, op\rangle$, where sc is a hypothesis schema, ws is an ordered list of hypothesis weightings and op is a binary operator. A hypothesis H *conforms to osc* if and only if H conforms to sc and $\uparrow([\omega_i[H] \mid \omega_i \in ws])\ op\ 0$.

In the next section, we present a method for translating a brave ordering example into a set of ordering schemas. The result in Lemma 10.6 means that we can use our method for translating a brave ordering example into a set of ordering schemas for translating both brave and cautious orderings (translating $inverse(o)$ instead of any cautious ordering o).

Lemma 10.6. *(proof on page 311)*

Consider a background knowledge B, hypothesis space S_M and a context-dependent ordering example o. Then for any hypothesis $H \subseteq S_M$, $B \cup H$ cautiously respects o if and only if $B \cup H$ does not bravely respect $inverse(o)$.

10.2 Translating Examples to Schemas

In this section we show how to translate (weighted) CDPIs and CDOEs into coverage constraints. In both cases the translation methods construct sets of schemas incrementally, at each step searching for a hypothesis H that covers the example, but which does not conform to any of the current set of schemas. In the case of a CDPI e, this means that $B \cup e_{ctx} \cup H$ must have at least one answer set that extends e_{pi}. In Section 10.2.1, we show how such an answer set A can be translated into a hypothesis schema that is conformed to by exactly those hypotheses that (together with B and e_{ctx}) accept A as an answer set. In Section 10.2.2, we present our method for translating a CDPI into a coverage constraint. Similarly, in Section 10.2.3 we show that, given an ordering example o, a pair of answer sets $\langle A_1, A_2\rangle$ can be translated into an ordering schema osc such that the hypotheses that conform to osc are exactly those hypotheses H for which $\langle A_1, A_2\rangle$ is an accepting pair of answer sets of o wrt of $B \cup H$. Finally, in Section 10.2.4, we show how to translate an ordering example into a coverage constraint.

10.2.1 The Translation of an Answer Set

At each iteration step, the ILASP3 algorithm translates an example into a set of hypothesis schemas, which is used to form a coverage constraint. The intuition is to start with an empty set of schemas and in each iteration to search for a hypothesis that accepts a current example, but which does not conform to any of the schemas computed so far. To prove that the example is accepted by such a hypothesis H the algorithm tries to find an answer set of $B \cup e_{ctx} \cup H$ that extends e_{pi}. If such an answer set exists, it is then translated into a hypothesis schema sc, which is in turn added to the current set SC of hypothesis schemas. We will show that when there are no hypotheses that accept e but do not conform to any of the already computed schemas, the set of computed schemas is guaranteed to be complete (e is accepted by H if and only if H conforms to at least one of the computed schemas). This incremental computation of hypothesis schemas relies upon a method that generates, from an interpretation I, a hypothesis schema that is conformed to by exactly those hypotheses H such that $I \in AS(B \cup e_{ctx} \cup H)$. We describe first, in this section, the method of computing hypothesis schemas from a given interpretation and in the next section we show how it can be used to iteratively compute the translation of an example.

Definition 10.10. Given a program P, hypothesis space S_M and an interpretation I, a hypothesis schema sc is said to be a *translation* of I (with respect to P and S_M) iff the set of hypotheses that conform to sc are exactly those hypotheses H such that $I \in AS(P \cup H)$.

Example 10.7. *Consider a learning task with background knowledge B and hypothesis space S_M defined as follows.*

$B = \{\texttt{q :- p.}\}$

$$S_M = \left\{ \begin{array}{ll} \texttt{h}_1: & \texttt{p.} \\ \texttt{h}_2: & \texttt{p :- q.} \\ \texttt{h}_3: & \texttt{q.} \\ \texttt{h}_4: & \texttt{r :- q.} \end{array} \right\}$$

Consider the interpretation $I = \{\texttt{p}, \texttt{q}\}$. One translation of I (with respect to B and S_M) is $sc = \langle \{\{\texttt{h}_1, \texttt{h}_3\}, \{\texttt{h}_1, \texttt{h}_2\}\}, \{\texttt{h}_4\}\rangle$. This can be seen as follows.

1. *Either $\texttt{h}_1$ or $\texttt{h}_3$ must be present in any hypothesis H for which $I \in AS(B \cup H)$, or $\{\texttt{p}, \texttt{q}\}$ would be an unfounded subset of I. Similarly, either $\texttt{h}_1$ or $\texttt{h}_2$ must be present, or $\{\texttt{p}\}$ would be unfounded. $\texttt{h}_4$ can not be in any hypothesis H such that $I \in AS(B \cup H)$, as I is not a model of $\texttt{h}_4$. Hence, any hypothesis H for which $I \in AS(B \cup H)$ must conform to sc.*

2. *Any hypothesis that conforms to sc must either contain $\texttt{h}_1$ or both $\texttt{h}_2$ and $\texttt{h}_3$. Either way, there are no unfounded subsets of I wrt $B \cup H$. As I is a model of B, $\texttt{h}_1$, $\texttt{h}_2$ and $\texttt{h}_3$, and no hypothesis H that conforms to sc can contain $\texttt{h}_4$, I is a model of $B \cup H$ for any hypothesis H that conforms to sc. Hence, for any hypothesis H that conforms to sc, $I \in AS(B \cup H)$.*

The *translateAS* algorithm finds a translation $\langle D, V\rangle$ of an interpretation I in two main steps. First it computes the set V of rules in S_M for which I is not a model (these rules cannot occur in any hypothesis H where $I \in AS(P \cup H)$). The second step is to iteratively compute the set of rule-disjunctions D.

D is initialised to be empty, and in each iteration a *potential unfounded subset* U is computed, and the rules in S_M which prevent U from being unfounded are added as a new rule-disjunction (for I to be an answer set of $P \cup H$, H must contain at least one such rule). Definition 10.11 formalises the notion of a potential unfounded set.

Definition 10.11. Given a program P, a hypothesis schema *sc* and an interpretation I, a set $U \subseteq I$ is said to be a *potential unfounded subset* of I wrt *sc* and P if and only if $U \neq \emptyset$ and $\exists H \subseteq S_M$ such that H conforms to *sc* and U is an unfounded subset of I wrt $P \cup H$.

Algorithm 10.1 *translateAS*(P, S_M, I)

```
1: procedure TRANSLATEAS$(P, S_M, I)$
2:     $V = \{\mathtt{h_{id}} \mid h \in S_M, I \models body(h), I \not\models head(h)\}$;
3:     $D = \emptyset$
4:     while $\exists U \subseteq I$ such that $U$ is a potential unfounded subset of $I$ wrt $sc$ and $P$
5:         Fix $U$ to be an arbitrary potential unfounded subset of $I$ wrt $sc$ and $P$
6:         $D = D \cup \{\{\mathtt{h_{id}} \mid \mathtt{h_{id}} \in ids(S_M)\backslash V, I \models body(h), U \cap body^+(h) = \emptyset, heads(h) \cap U \neq \emptyset\}\}$;
7:     end while
8:     return $\langle D, V \rangle$;
9: end procedure
```

Lines 4 and 5 of the *translateAS* algorithm rely on the computation of an arbitrary potential unfounded subset if one exists (and the ability to determine if at least one exists). In the ILASP3 implementation, this is done by representing the search for a potential unfounded subset in a meta-level ASP program. This meta-program (along with a proof of its correctness) can be found in Appendix B.1.

Theorem 10.8. *(proof on page 316)*
Let P be a program, S_M be a hypothesis space and I be an interpretation. The procedure translateAS(P, S_M, I) terminates and returns a translation of I.

Example 10.9. *Reconsider the learning task from Example 10.7.*

Consider the interpretation $I = \{\mathtt{p}, \mathtt{q}\}$. We now show how translateAS can compute the schema $sc = \langle\{\{\mathtt{h_1}, \mathtt{h_3}\}, \{\mathtt{h_1}, \mathtt{h_2}\}\}, \{\mathtt{h_4}\}\rangle$, which is a translation of I.

The first step of translateAS(B, I, S_M) is to compute the set V. The only rule in S_M whose body is satisfied by I and whose head is not satisfied by I is $\mathtt{h_4}$; hence, $V = \{\mathtt{h_4}\}$

translateAS now searches for potential unfounded subsets of I wrt the schema $\langle\emptyset, \{\mathtt{h_4}\}\rangle$. $\{\mathtt{p}\}$ is such a potential unfounded subset ($H = \emptyset$ conforms to the schema and $\{\mathtt{p}\}$ is an unfounded subset of I wrt $B \cup \emptyset$). This causes translateAS to add the new rule-disjunction $\{\mathtt{h_1},\ \mathtt{h_2}\}$ to D.

In the next iteration, translateAS searches for a potential unfounded subset of I wrt the schema $\langle\{\{\mathtt{h_1},\ \mathtt{h_2}\}\}, \{\mathtt{h_4}\}\rangle$. $\{\mathtt{p},\ \mathtt{q}\}$ is a potential unfounded subset of I wrt this schema (consider the hypothesis $H_{\{\mathtt{h_2}\}}$). This causes the rule-disjunction $\{\mathtt{h_1},\ \mathtt{h_3}\}$ to be added to D.

In the final iteration, there are no potential unfounded subsets of I wrt $\langle D, V\rangle$, and hence the schema $\langle\{\{\mathtt{h}_1, \mathtt{h}_2\}, \{\mathtt{h}_1, \mathtt{h}_3\}\}, \{\mathtt{h}_4\}\rangle$ is returned.

10.2.2 Translating an Example

We now present the method used by ILASP3 to translate examples into sets of hypothesis schemas. This method makes use of Algorithm 10.1 to translate each computed interpretation into a hypothesis schema.

Definition 10.12. Let $T = \langle B, S_M, E\rangle$ be an ILP^{noise}_{LOAS} task, e be an example and SC be a set of hypothesis schemas. SC is a *partial translation* of e iff $\forall H \subseteq S_M$ such that H conforms to at least one schema in SC, $B \cup H$ accepts e. Furthermore, a partial translation SC of e is said to be *complete* if for each $H \subseteq S_M$ such that $B \cup H$ accepts e, H conforms to at least one schema in SC.

Example 10.10. *Consider an ILP^{noise}_{LOAS} task with background knowledge B and hypothesis space S_M as defined below.*

$$B = \left\{ \begin{array}{l} \mathtt{p\!:\!-\ not\ q.} \\ \mathtt{q\!:\!-\ not\ p.} \end{array} \right\} \qquad S_M = \left\{ \begin{array}{ll} \mathtt{h}_1: & \mathtt{r\!:\!-t.} \\ \mathtt{h}_2: & \mathtt{t\!:\!-q.} \\ \mathtt{h}_3: & \mathtt{r.} \end{array} \right\}$$

Let e be an example such that $e_{pi} = \langle\{\mathtt{r}\}, \emptyset\rangle$, and $e_{ctx} = \emptyset$. The interpretations which extend e_{pi} and are answer sets of $B \cup H$ for some $H \subseteq S_M$ are $\{\{\mathtt{p}, \mathtt{r}\}, \{\mathtt{q}, \mathtt{r}\}, \{\mathtt{q}, \mathtt{r}, \mathtt{t}\}\}$. We refer to these interpretations as A_1, A_2 and A_3, respectively. The translation sc_1 of A_1 (using Algorithm 10.1) is $\langle\{\{\mathtt{h}_3\}\}, \{\mathtt{h}_2\}\rangle$. $\{sc_1\}$ is a partial translation of e, but it is not complete. To see this, consider the hypothesis $H = \{\mathtt{h}_1, \mathtt{h}_2\}$. $B \cup H$ has the answer set A_3, but H does not conform to sc_1. The translation sc_3 of A_3 (using Algorithm 10.1) is $\langle\{\{\mathtt{h}_2\}, \{\mathtt{h}_1, \mathtt{h}_3\}\}, \emptyset\rangle$. $\{sc_1, sc_3\}$ is a complete translation. Note that every hypothesis that has the answer set A_2 (when combined with the background knowledge) also has the answer set A_1, and so there is no need to translate A_2.

Algorithm 10.2 formalises the procedure *translateExample*. The idea in this procedure is to start from an empty set of schemas SC and, in each iteration, search for a hypothesis that does not conform to any schema in SC, but has an answer set I that extends e. If such an answer set I is found, it is translated, using the *translateAS* algorithm, into a hypothesis schema that is then added to SC. If no such answer set is found then SC is a complete translation of e, and so the algorithm returns SC. Theorem 10.11 proves that the algorithm is guaranteed to terminate, and always returns a complete translation of the example.

Details of the computation of the interpretation I (or determining that no such I exists) can be found in Appendix B.1. The computation encodes the search as a meta-level ASP program.

Algorithm 10.2 *translateExample*(T, e)

```
procedure TRANSLATEEXAMPLE($T, e$)
    $SC = \emptyset$;
    while $\exists I$ st $\exists H \subseteq S_M$ st $\forall sc \in SC$, $H$ does not conform to $sc$ and $I$ is an accepting answer set of $e$ wrt $B \cup H$
        Fix an arbitrary such $I$
        $SC = SC \cup \{translateAS(B \cup e_{ctx}, S_M, I)\}$;
    end while
    return $\langle e, SC\rangle$;
end procedure
```

Theorem 10.11. *(proof on page 319)*

Let T be an ILP^{noise}_{LOAS} task and e be an example. translateExample(T, e) terminates and returns a pair $\langle e, SC\rangle$ where SC is a complete translation of e.

The pair $\langle e, SC\rangle$ computed by the procedure *translateExample* is a *coverage constraint*. Its meaning is that e is accepted by a hypothesis H if and only if H conforms to at least one schema in SC.

10.2.3 Converting a Pair of Answer Sets into an Ordering Schema

We can convert any pair of interpretations into a set of hypothesis weightings. This translation is formalised in Definition 10.13.

Definition 10.13. Let T be an ILP^{noise}_{LOAS} task with background knowledge B and hypothesis space S_M and let A_1 and A_2 be a pair of interpretations. For any integer l, $\omega(T, A_1, A_2, l)$ is the hypothesis weighting $\langle \omega^f, \omega^k\rangle$, where:

- For each $h \in S_M$, $\omega^f(h_{id}) = \Delta^h_l(A_1, A_2)$
- $\omega^k = \Delta^B_l(A_1, A_2)$

Let $[l_1, \ldots, l_n]$ be the list of priority levels in $B \cup S_M$ (in descending order). Then the *translation* of $\langle A_1, A_2\rangle$ wrt T (denoted $\omega(T, A_1, A_2)$) is the list $[\omega(T, A_1, A_2, l_1), \ldots, \omega(T, A_1, A_2, l_n)]$.

Example 10.12. *Consider the ILP^{noise}_{LOAS} task $T = \langle B, S_M, E\rangle$, where weak$(B)$ and weak(S_M) are as below:*

$$weak(B) = \left\{ \;:\sim \texttt{p(X).[X@1,X,1]} \right\} \qquad weak(S_M) = \left\{ \begin{array}{ll} \texttt{h}_1: & :\sim \texttt{q(X).[X@1,2]} \\ \texttt{h}_2: & :\sim \texttt{r(X).[X@2,X,3]} \end{array} \right\}$$

Also, consider the two interpretations $A_1 = \{$`p(1)`, `p(2)`, `r(1)`, `r(2)`, `q(3)`*$\}$ and $A_2 = \{$*`r(2)`, `r(3)`, `q(2)`*$\}$.*

As there are two priority levels in the task, $\omega(T, A_1, A_2)$ is the list $[\omega(T, A_1, A_2, 2), \omega(T, A_1, A_2, 1)]$. This is equal to the list $[\langle[\texttt{h}_2 : -2], 0\rangle, \langle[\texttt{h}_1 : 1], 3\rangle]$.

Theorem 10.13 shows how we can use the translation of a pair of answer sets A_1 and A_2 (with respect to some task T) to compute the deciding optimisation difference of A_1 and A_2 with respect to any hypothesis in the hypothesis space of T.

Theorem 10.13. *(proof on page 319)*

Let T be a ILP^{noise}_{LOAS} task and let A_1 and A_2 be interpretations. Let $[\omega_1, \ldots, \omega_n] = \omega(T, A_1, A_2)$. For any $H \subseteq S_M$, $\Delta^{B \cup H}(A_1, A_2) = \uparrow([\omega_1[H], \ldots, \omega_n[H]])$

We now present the *translatePairAS* procedure, which given a pair of answer sets $\langle A_1, A_2\rangle$, computes an ordering schema *osc* such that the hypotheses H that conform to *osc* are exactly those for which $\langle A_1, A_2\rangle$ is an accepting pair of answer sets of o wrt H. It works by computing a pair of hypothesis schemas, which are translations of the two answer sets, and then translating the pair of answer sets into a list of hypothesis weightings. It combines the two hypothesis schemas into one, making use of the following notation.

Notation ($sc_1 \cup sc_2$). Let $sc_1 = \langle D_1, V_1\rangle$ and $sc_2 = \langle D_2, V_2\rangle$ be hypothesis schemas. The *combination* of sc_1 and sc_2 (denoted $sc_1 \cup sc_2$) is the hypothesis schema $\langle D_1 \cup D_2, V_1 \cup V_2\rangle$.

Algorithm 10.3 *translatePairAS*(T, o, A_1, A_2)

```
1: procedure TRANSLATEPAIRAS$(T, o, A_1, A_2)$
2:     $sc_1 = translateAS(B \cup (o_{eg1})_{ctx}, S_M, A_1)$
3:     $sc_2 = translateAS(B \cup (o_{eg2})_{ctx}, S_M, A_2)$
4:     return $\langle sc_1 \cup sc_2, \omega(T, A_1, A_2), o_{op}\rangle$;
5: end procedure
```

Theorem 10.14. *(proof on page 321)*

Let T be the ILP^{noise}_{LOAS} task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$. Also let o be a CDOE and A_1 and A_2 be interpretations that extend $(o_{eg1})_{pi}$ and $(o_{eg2})_{pi}$, respectively. translatePairAS(T, o, A_1, A_2) terminates and returns an ordering schema osc such that $\forall H \subseteq S_M$, H conforms to osc if and only if $\langle A_1, A_2\rangle$ is an accepting pair of answer sets of o wrt $B \cup H$.

10.2.4 The Translation of an Ordering Example

In the ILASP3 algorithm, we translate CDOE's into sets of ordering schemas. Definition 10.14 formalises the notion of the translation of a CDOE.

Definition 10.14. Let T be any ILP^{noise}_{LOAS} task with background knowledge B and hypothesis space S_M, let o be a CDOE and let OSC be a set of ordering schemas. OSC is a *partial translation* of o wrt T if $\forall H \subseteq S_M$ such that H conforms to at least one schema in OCS, $B \cup H$ bravely respects o. Furthermore, a partial translation OSC of o is said to be *complete* if for each $H \subseteq S_M$ such that $B \cup H$ bravely respects o, H conforms to at least one schema in OSC.

Algorithm 10.4 formalises our method for translating an ordering example o into a set of ordering schemas. Similarly to the *translateExample* algorithm, it works by incrementally computing pairs of interpretations that are accepting pairs of answer sets of o with respect to $B \cup H$, for at least one hypothesis H that does not conform to any schema in OSC. Theorem 10.15 shows that $translateOrderingExample(T, o)$ always terminates and returns a complete translation of o.

Algorithm 10.4 $translateOrderingExample(T, o)$

```
1: procedure TRANSLATEORDERINGEXAMPLE($T, o$)
2:     $OSC = \emptyset$;
3:     while $\exists\langle I_1, I_2\rangle$ st $\exists H \subseteq S_M$ st $\forall osc \in OSC$, $H$ does not conform to $osc$ and $\langle I_1, I_2\rangle$ is an accepting pair of answer sets of $o$ wrt $B \cup H$
4:         Fix an arbitrary such $\langle I_1, I_2\rangle$
5:         $OSC = OSC \cup \{translatePairAS(T, o, I_1, I_2)\}$;
6:     end while
7:     return $\langle o, OSC\rangle$;
8: end procedure
```

Details of the computation of a pair of accepting answer sets of o wrt $B \cup H$ (or determining that no such pair exists) can be found in Appendix B.1. The computation encodes the search as a meta-level ASP program.

Theorem 10.15. *(proof on page 322)*

Let T be any ILP^{noise}_{LOAS} task and o be any CDOE. $translateOrderingExample(T, o)$ terminates and returns a pair $\langle o, OSC\rangle$, where OSC is a complete translation of o.

10.3 Representing Hypothesis Schemas

In this section we present the meta-level representation adopted in ILASP3 to determine, for a given set of hypothesis schemas, which hypotheses conform to which schemas. For convenience we introduce the following notation.

Notation ($DISJ(SC)$). Given a set SC of hypothesis schemas, $DISJ(SC)$ denotes the set of all rule-disjunctions that appear in SC (i.e. $\{d \mid \langle D, V\rangle \in SC, d \in D \cup \{V\}\}$)

The formal definition of the meta-level program $\mathcal{M}_{sc}$ is given in Meta-program 10.1. Informally, for each rule-disjunction d that appears in $DISJ(SC)$, the meta-level program includes a rule that states that if none of the rules of the hypothesis space with id's in d are in a hypothesis, then the disjunction d is not satisfied. This rule defines the predicate `not_disj(`$\mathtt{d_{id}}$`)`. For each schema $\langle D, V\rangle$, the meta-level program includes a rule stating that if each $d \in D$ is satisfied (expressed as `not not_disj(d_id)` in the definition above), and V is *not* satisfied then the schema is conformed to (by the current hypothesis). Finally, for each rule h in the hypothesis space, the meta-level program includes a choice rule to state that h is either in the hypothesis (expressed by the predicate `in_h(h_id)`) or not.

Meta-program 10.1 ($\mathcal{M}_{sc}(SC, S_M)$)**.** Let SC be a set of hypothesis schemas and S_M be a hypothesis space. $\mathcal{M}_{sc}(SC, S_M)$ is a program consisting of the following rules:

- $\texttt{not\_disj(d}_{\texttt{id}}\texttt{):- not in\_h(d}_{\texttt{1}}\texttt{)}, \ldots, \texttt{ not in\_h(d}_{\texttt{n}}\texttt{).}$
  for each rule-disjunction $\{\texttt{d}_1, \ldots, \texttt{d}_n\} \in DISJ(SC)$ with identifier $\texttt{d}_{\texttt{id}}$
- $\texttt{conforms(sc}_{\texttt{id}}\texttt{):- not not\_disj(D}^{\texttt{1}}_{\texttt{id}}\texttt{)}, \ldots, \texttt{ not not\_disj(D}^{\texttt{n}}_{\texttt{id}}\texttt{),not\_disj(V}_{\texttt{id}}\texttt{).}$
  for each schema $sc \in SC$, where $sc = \langle\{D^1, \ldots, D^n\}, V\rangle$
- $\texttt{0\{in\_h(h}_{\texttt{id}}\texttt{)\}1.}$
  for each rule $h \in S_M$

Theorem 10.16. *(proof on page 311) Let SC be a set of hypothesis schemas and S_M be a hypothesis space. For each $H \subseteq S_M$, there is exactly one answer set A_H of $\mathcal{M}_{sc}(SC, S_M)$ such that $H = \mathcal{M}^{-1}_{in\_h}(A_H)$. Furthermore, $\forall sc \in SC$, $\texttt{conforms(sc}_{\texttt{id}}\texttt{)} \in A_H$ if and only if H conforms to sc.*

We give now an example of Meta-program 10.1.

Example 10.17. *Consider a hypothesis space S_M composed of 4 rules with ids $\texttt{h}_1, \ldots, \texttt{h}_4$ and the hypothesis schemas sc_1 and sc_2.*

$sc_1 = \langle\{\{\texttt{h}_1, \texttt{h}_2\}, \{\texttt{h}_1, \texttt{h}_3\}\}, \{\texttt{h}_4\}\rangle$ $\qquad$ $sc_2 = \langle\{\{\texttt{h}_1, \texttt{h}_2\}, \{\texttt{h}_4\}\}, \{\}\rangle$

$DISJ(\{sc_1, sc_2\}) = \{\{\texttt{h}_1, \texttt{h}_2\}, \{\texttt{h}_1, \texttt{h}_3\}, \{\texttt{h}_4\}, \{\}\}$, in which rule-disjunctions have, respectively, identifiers $\texttt{d}_1, \ldots \texttt{d}_4$. The meta-level program $\mathcal{M}_{sc}(SC, S_M)$ is then given by the program:

$$\left\{\begin{array}{l}
\texttt{conforms(sc}_1\texttt{):- not not\_disj(d}_1\texttt{), not not\_disj(d}_2\texttt{),not\_disj(d}_3\texttt{).}\\
\texttt{conforms(sc}_2\texttt{):- not not\_disj(d}_1\texttt{), not not\_disj(d}_3\texttt{),not\_disj(d}_4\texttt{).}\\
\texttt{not\_disj(d}_1\texttt{):- not in\_h(h}_1\texttt{), not in\_h(h}_2\texttt{).}\\
\texttt{not\_disj(d}_2\texttt{):- not in\_h(h}_1\texttt{), not in\_h(h}_3\texttt{).}\\
\texttt{not\_disj(d}_3\texttt{):- not in\_h(h}_4\texttt{).}\\
\texttt{not\_disj(d}_4\texttt{).}\\
\texttt{0\{in\_h(h}_1\texttt{)\}1. 0\{in\_h(h}_2\texttt{)\}1. 0\{in\_h(h}_3\texttt{)\}1. 0\{in\_h(h}_4\texttt{)\}1.}
\end{array}\right\}$$

Using Theorem 10.16, we can determine which hypotheses $H \subseteq S_M$ conform to which schemas in SC by using $\mathcal{M}_{sc}(SC, S_M)$. For example, consider $H_{\{1,4\}}$ (the hypothesis containing the rules with ids $\texttt{h}_1$ and $\texttt{h}_4$). $\mathcal{M}_{sc}(SC, S_M)$ has exactly one answer set A such that $\{\texttt{h}_{\texttt{id}} \mid \texttt{in\_h(h}_{\texttt{id}}\texttt{)} \in A\} = \{\texttt{h}_1, \texttt{h}_4\}$. This *is the answer set* $A = \{\mathtt{in\_h(h_1)}, \mathtt{in\_h(h_4)}, \mathtt{not\_disj(d_4)}, \mathtt{conforms(sc_2)}\}$*. We can therefore conclude that* $H_{\{1,4\}}$ *conforms to* sc_2 *but does not conform to* sc_1*. This is indeed true:* $H_{\{1,4\}}$ *satisfies* $\mathtt{d_1}$ *and* $\mathtt{d_2}$ *as it contains the rule with id* $\mathtt{h_1}$*, satisfies* $\mathtt{d_3}$ *as the hypothesis contains the rule with id* $\mathtt{h_4}$*, but does not satisfy* $\mathtt{d_4}$ *as it is empty and thus is impossible to satisfy; hence the hypothesis conforms to* sc_2 *but does not conform to* sc_1*.*

10.3.1 Representing Ordering Schemas

Similarly to hypothesis schemas, we represent ordering schemas as an ASP program. Given a set of ordering schemas OSC and a hypothesis schema S_M, $\mathcal{M}_{osc}(OSC, S_M)$ consists of two components. The first component is used to check, for each ω that occurs in any ordering schema, whether the hypothesis H represented by the `in_h` atoms yields a positive or negative value for $\omega[H]$. The second component uses this information to determine whether each ordering schema is conformed to (represented by the `osc_conforms` predicate). $\mathcal{M}_{osc}$ is used in conjunction with the $\mathcal{M}_{sc}(SC, S_M)$ program, where SC contains each hypothesis schema that occurs in the ordering schemas in OSC. $\mathcal{M}_{sc}$ defines the `in_h` and `conforms` predicates.

Meta-program 10.2 ($\mathcal{M}_{osc}(OSC, S_M)$)**.** Let OSC be a set of ordering schemas and S_M be a hypothesis space such that $ids(weak(S_M)) = \{\mathtt{h_1}, \ldots, \mathtt{h_n}\}$. $\mathcal{M}_{osc}(OSC, S_M)$ is the program consisting of the following components:

- The rules $\left\{\begin{array}{l} \mathtt{diff}(\omega^{\mathtt{id}}, 1) \,\mathtt{:-}\, \#\mathtt{sum}\{\mathtt{in\_h(h_1)} = \omega^{\mathtt{f}}(\mathtt{h_1}), \ldots, \mathtt{in\_h(h_n)} = \omega^{\mathtt{f}}(\mathtt{h_n})\} > -\omega^{\mathtt{k}}. \\ \mathtt{diff}(\omega^{\mathtt{id}}, -1) \,\mathtt{:-}\, \#\mathtt{sum}\{\mathtt{in\_h(h_1)} = \omega^{\mathtt{f}}(\mathtt{h_1}), \ldots, \mathtt{in\_h(h_n)} = \omega^{\mathtt{f}}(\mathtt{h_n})\} < -\omega^{\mathtt{k}}. \end{array}\right\}$
  for each ω such that $\exists\langle sc, ws, op\rangle \in OSC$ such that $\omega \in ws$

- For each $osc = \langle sc, [\omega_1, \ldots \omega_m], op\rangle \in OSC$, the rules:
  $$\left\{\begin{array}{c} \mathtt{osc\_diff(osc_{id}, D)} \,\mathtt{:-}\, \mathtt{diff}(\omega_{\mathtt{i}}, \mathtt{D}),\ \mathtt{not\ diff}(\omega^{\mathtt{id}}_{1}, -1),\ \mathtt{not\ diff}(\omega^{\mathtt{id}}_{1}, 1), \\ \ldots, \\ \mathtt{not\ diff}(\omega^{\mathtt{id}}_{\mathtt{i-1}}, -1),\ \mathtt{not\ diff}(\omega^{\mathtt{id}}_{\mathtt{i-1}}, 1). \end{array}\middle|\, i \in [1, m]\right\}$$
  And the rules:
  $$\left\{\begin{array}{l} \mathtt{osc\_diff(osc_{id}, 0)} \,\mathtt{:-}\ \mathtt{not\ osc\_diff(osc_{id}, 1)},\ \mathtt{not\ osc\_diff(osc_{id}, -1)}. \\ \mathtt{osc\_conforms(osc_{id})} \,\mathtt{:-}\, \mathtt{conforms(sc_{id})}, \mathtt{osc\_diff(osc_{id}, D)}, \mathtt{D\,op\,0}. \end{array}\right\}$$

Theorem 10.18 shows that $\mathcal{M}_{osc}$ can be used in conjunction with $\mathcal{M}_{sc}$ in order to determine which ordering schemas are conformed to by each hypothesis in a given hypothesis space.

Theorem 10.18. *(proof on page 312)*

Let S_M be a hypothesis space, SC be a set of hypothesis schemas and OSC be a set of ordering schemas such that $\{sc \mid \langle sc, ws, op\rangle \in OSC\} \subseteq SC$. For each $H \subseteq S_M$, there is exactly one answer set A_H of $\mathcal{M}_{sc}(SC, S_M) \cup \mathcal{M}_{osc}(OSC, S_M)$ such that $\mathcal{M}^{-1}_{in\_h}(A_H) = H$. Furthermore:

1. *$\forall sc \in SC$, $\texttt{conforms}(\texttt{sc}_{\texttt{id}}) \in A_H$ if and only if H conforms to sc*
2. *$\forall osc \in OSC$, $\texttt{osc\_conforms}(\texttt{osc}_{\texttt{id}}) \in A_H$ if and only if H conforms to osc*

We now extend Example 10.17 to exemplify our representation of ordering schemas.

Example 10.19. *Consider a hypothesis space S_M composed of 8 rules with ids $1, \ldots, 8$, the hypothesis schema sc and the hypothesis weightings ω_1 and ω_2.*

$$sc_1 = \langle \{\{\texttt{h}_1, \texttt{h}_2\}, \{\texttt{h}_1, \texttt{h}_3\}\}, \{\texttt{h}_4\}\rangle \qquad \omega_2 = \langle [\texttt{h}_5 : 6, \texttt{h}_6 : -6, \texttt{h}_7 : 1, \texttt{h}_8 : -3], 0\rangle$$

$$\omega_1 = \langle [\texttt{h}_5 : 6, \texttt{h}_7 : -10, \texttt{h}_8 : -3], 4\rangle$$

Let $OSC = \{\langle sc_1, [\omega_1, \omega_2], <\rangle\}$ The meta-level program $\mathcal{M}_{osc}(OSC, S_M)$ is then given by the program:

$$\left\{\begin{array}{l}
\texttt{diff}(\omega_1,1)\texttt{:-}\#\texttt{sum}\{\texttt{in\_h}(\texttt{h}_5)=6,\texttt{in\_h}(\texttt{h}_7)=-10,\texttt{in\_h}(\texttt{h}_8)=-3\}>-4.\\
\texttt{diff}(\omega_1,-1)\texttt{:-}\#\texttt{sum}\{\texttt{in\_h}(\texttt{h}_5)=6,\texttt{in\_h}(\texttt{h}_7)=-10,\texttt{in\_h}(\texttt{h}_8)=-3\}<-4.\\
\texttt{diff}(\omega_2,1)\texttt{:-}\#\texttt{sum}\{\texttt{in\_h}(\texttt{h}_5)=6,\texttt{in\_h}(\texttt{h}_6)=-6,\texttt{in\_h}(\texttt{h}_7)=1,\texttt{in\_h}(\texttt{h}_8)=-3\}>0.\\
\texttt{diff}(\omega_2,-1)\texttt{:-}\#\texttt{sum}\{\texttt{in\_h}(\texttt{h}_5)=6,\texttt{in\_h}(\texttt{h}_6)=-6,\texttt{in\_h}(\texttt{h}_7)=1,\texttt{in\_h}(\texttt{h}_8)=-3\}<0.\\
\texttt{osc\_diff}(\texttt{osc}_1,\texttt{D})\texttt{:-}\,\texttt{diff}(\omega_1,\texttt{D}).\\
\texttt{osc\_diff}(\texttt{osc}_1,\texttt{D})\texttt{:-}\,\texttt{diff}(\omega_2,\texttt{D}),\ \texttt{not diff}(\omega_1,-1),\ \texttt{not diff}(\omega_1,1).\\
\texttt{osc\_diff}(\texttt{osc}_1,0))\texttt{:- not osc\_diff}(\texttt{osc}_1,1),\ \texttt{not osc\_diff}(\texttt{osc}_1,-1).\\
\texttt{osc\_conforms}(\texttt{osc}_1))\texttt{:-}\,\texttt{conforms}(\texttt{sc}_1),\texttt{osc\_diff}(\texttt{osc}_1,\texttt{D}),\texttt{D}<0.
\end{array}\right\}$$

$\mathcal{M}_{sc}(SC, S_M)$ is the program:

$$\left\{\begin{array}{l}
\texttt{conforms}(\texttt{sc}_1)\texttt{:- not not\_disj}(\texttt{d}_1),\ \texttt{not not\_disj}(\texttt{d}_2),\texttt{not\_disj}(\texttt{d}_3).\\
\texttt{not\_disj}(\texttt{d}_1)\texttt{:- not in\_h}(\texttt{h}_1),\ \texttt{not in\_h}(\texttt{h}_2).\\
\texttt{not\_disj}(\texttt{d}_2)\texttt{:- not in\_h}(\texttt{h}_1),\ \texttt{not in\_h}(\texttt{h}_3).\\
\texttt{not\_disj}(\texttt{d}_3)\texttt{:- not in\_h}(\texttt{h}_4).\\
\texttt{0}\{\texttt{in\_h}(\texttt{h}_1)\}\texttt{1.}\ \ldots\ \texttt{0}\{\texttt{in\_h}(\texttt{h}_8)\}\texttt{1.}
\end{array}\right\}$$

We can use the result of Theorem 10.18 to test which hypotheses conform to the ordering schema. For example, consider the hypothesis $H_{\{1,7\}}$. $\mathcal{M}_{sc}(SC, S_M) \cup \mathcal{M}_{osc}(OSC, S_M)$ has exactly one answer set corresponding to the hypothesis, $\{\texttt{in\_h}(\texttt{h}_1)$, $\texttt{in\_h}(\texttt{h}_7)$, $\texttt{not\_disj}(\texttt{d}_3)$, $\texttt{conforms}(\texttt{sc}_1)$, $\texttt{diff}(\texttt{w}_1, -1)$, $\texttt{diff}(\texttt{w}_2, 1)$, $\texttt{osc\_diff}(\texttt{osc}_1, -1)$, $\texttt{osc\_conforms}(\texttt{osc}_1)\}$, which shows that the hypothesis conforms to both sc_1 and osc_1.

10.4 Solving Intermediate Tasks

In each iteration, the ILASP3 algorithm finds an optimal hypothesis with respect to the coverage constraints computed so far in the iterative cycle. This is done by the procedure *solve_current_task* described in Algorithm 10.5. This procedure simply solves a meta-level program $\mathcal{M}_{solve}$, representing the coverage constraints, and extracts the hypothesis and its approximate coverage from the answer set. Meta-program 10.3 shows the program $\mathcal{M}_{solve}$.

The program $\mathcal{M}_{solve}$ specifies that each rule in the hypothesis space is either in the hypothesis or not, and determines which examples can be proved not to be covered, given the schemas and constraints computed so far. These examples are represented by the `uncov` predicate. The weak constraints in this program minimise the sum of the penalties of the examples which are known to be uncovered plus the length of the hypothesis (i.e. the lower bound score of the hypothesis wrt the coverage constraints).

Meta-program 10.3 ($\mathcal{M}_{solve}(CC, S_M, E)$)**.** Let $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$ be an ILP^{noise}_{LOAS} task and CC be a set of coverage constraints. Let ALL_SC be the set of all hypothesis schemas that occur in CC and let ALL_OSC be the set of all ordering schemas that occur in CC. $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$ is the program that extends $\mathcal{M}_{sc}(ALL\_SC, S_M) \cup \mathcal{M}_{ocs}(ALL\_OSC, S_M)$ with the following rules:

- $\texttt{uncov}(\texttt{e}_{\texttt{id}})\texttt{:-}\ \texttt{not conforms}(\texttt{sc}^{\texttt{1}}_{\texttt{id}}), \ldots, \texttt{not conforms}(\texttt{sc}^{\texttt{n}}_{\texttt{id}})$.
  for each $\langle e, \{sc^1, \ldots, sc^n\}\rangle \in CC$ and $e \in E^+$
- $\texttt{uncov}(\texttt{e}_{\texttt{id}})\texttt{:-}\texttt{conforms}(\texttt{sc}_{\texttt{id}})$.
  for each $\langle e, SC\rangle \in CC$, $e \in E^-$ and $sc \in SC$
- $\texttt{uncov}(\texttt{o}_{\texttt{id}})\texttt{:-}\ \texttt{not osc\_conform}(\texttt{osc}^{\texttt{1}}_{\texttt{id}}), \ldots, \texttt{not osc\_conform}(\texttt{osc}^{\texttt{n}}_{\texttt{id}})$.
  for each $\langle o, \{osc^1, \ldots, osc^n\}\rangle \in CC$ and $o \in O^b$
- $\texttt{uncov}(\texttt{o}_{\texttt{id}})\texttt{:-}\texttt{osc\_conform}(\texttt{osc}_{\texttt{id}})$.
  for each $\langle o, OSC\rangle \in CC$, $inverse(o) \in O^c$ and $osc \in OSC$
- $:\sim \texttt{uncov}(\texttt{e}_{\texttt{id}}).\ [\texttt{e}_{\texttt{pen}}@0, \texttt{noise}(\texttt{e}_{\texttt{id}})]$
  for each $e \in E^+ \cup E^- \cup O^b \cup O^c$, $e_{pen} < \infty$
- $: -\texttt{uncov}(\texttt{e}_{\texttt{id}})$.
  for each $e \in E^+ \cup E^- \cup O^b \cup O^c$, $e_{pen} = \infty$
- $:\sim \texttt{in\_h}(\texttt{R}_{\texttt{id}}).[|\texttt{R}|@0, \texttt{h\_length}(\texttt{R}_{\texttt{id}})]$
  for each $R \in S_M$

Theorem 10.20 shows that the procedure *solve_current_task* returns an optimal hypothesis wrt the coverage constraints, if such a hypothesis exists. If no such hypothesis exists then *solve_current_task*

Algorithm 10.5

```
1: procedure SOLVE_CURRENT_TASK($CC, \langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$)
2:     $A = solve(\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle))$;
3:     if $A ==$ nil
4:         return $\langle$nil, nil$\rangle$;
5:     end if
6:     $H = \{h \in S_M \mid \texttt{in\_h}(\texttt{h}_{\texttt{id}}) \in A\}$;
7:     $ApproxCov = \{e \in E^+ \cup E^- \cup O^b \cup O^c \mid \texttt{uncov}(\texttt{e}_{\texttt{id}}) \not\in A\}$;
8:     return $\langle H, ApproxCov\rangle$;
9: end procedure
```

returns $\langle \texttt{nil}, \texttt{nil}\rangle$. Note that in this case, there is no hypothesis that covers every example with a finite penalty, and so the task is unsatisfiable.

Theorem 10.20. *(proof on page 322)*

For any set CC of coverage constraints, and any ILP^{noise}_{LOAS} task T:

1. *$solve\_current\_task(CC, T)$ terminates.*
2. *If there is no hypothesis $H \subseteq S_M$ such that $\mathcal{S}_{lb}(H, T, CC)$ is finite, then $solve\_current\_task(CC, T)$ returns $\langle \texttt{nil}, \texttt{nil}\rangle$.*
3. *If there is a hypothesis $H \subseteq S_M$ such that $\mathcal{S}_{lb}(H, T, CC)$ is finite, then $solve\_current\_task(CC, T)$ returns a pair $\langle H^*, ApproxCov\rangle$, where H^* is optimal with respect to CC and $ApproxCov = ApproxCoverage(H, T, CC)$.*

10.5 ILASP3

Algorithm 10.6 formalises our ILASP3 algorithm, which makes use of the methods described in the chapter so far.

Before the first iteration, there are no coverage constraints and so CC is initialised to be empty. As there are no coverage constraints, at this point the optimal hypothesis with respect to CC is also empty, and every example is assumed to be covered by H (hence, the variable *ApproxCoverage* is initialised to be the full set of examples). The next step is to use the $findRelevantExample$ method of ILASP2i, to search for an example that is incorrectly assumed to be covered by H. If such an example is found, then the approximation of the coverage for H is incorrect, and so the example is translated into a coverage constraint, which is added to CC. The *solve_current_task* method is then used to search for an optimal hypothesis with respect to CC. If no such hypothesis exists then the task is unsatisfiable, and the pair $\langle \texttt{nil}, \texttt{nil}\rangle$ is returned. If such a hypothesis does exist, the *solve_current_task* method returns a pair containing the hypothesis and the corresponding

Algorithm 10.6 *$ILASP3(T)$*

```
procedure ILASP3($T$)
    let $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle = T$
    $CC = \emptyset; H = \emptyset; ApproxCoverage = E^+ \cup E^- \cup O^b \cup O^c$
    $eg = findRelevantExample(\langle B, S_M, ApproxCoverage\rangle, H)$
    while $eg \neq$ nil
        if $eg \in E^+ \cup E^-$
            $CC.insert(translateExample(T, eg))$
        else if $eg \in O^b$
            $CC.insert(translateOrderingExample(T, eg))$
        else
            $CC.insert(translateOrderingExample(T, inverse(eg)))$
        end if
        $\langle H, ApproxCoverage\rangle = solve\_current\_task(CC, T)$
        if $H ==$ nil
            return UNSATISFIABLE;
        end if
        $eg = findRelevantExample(\langle B, S_M, ApproxCoverage\rangle, H)$
    end while
    return $H$;
end procedure
```

approximation of the coverage with respect to CC. ILASP3 then searches for an example that is incorrectly assumed to be covered by the new H. This process continues until no such example can be found. When no such example can be found, the approximation of the coverage for the current H must be equal to the true coverage of the example. Therefore $\mathcal{S}_{lb}(H, T, CC)$ must be equal to $\mathcal{S}(H, T)$. Hence, as for every other hypothesis $H' \in S_M$, $\mathcal{S}_{lb}(H, T, CC) \leq \mathcal{S}_{lb}(H', T, CC) \leq \mathcal{S}(H', T)$, there can be no hypothesis H' such that $\mathcal{S}(H', T) < \mathcal{S}(H, T)$. This means that the hypothesis returned by ILASP3 must be an optimal inductive solution of T.

We now present some theoretical results about the ILASP3 algorithm. Theorem 10.21 shows that ILASP3 is sound and complete with respect to the optimal inductive solutions of an ILP_{LOAS}^{noise} task and will always terminate. Note that as ILASP3 returns a single optimal solution, rather than the full set, the completeness is in terms of being guaranteed to find an optimal solution if one exists.

Theorem 10.21. *(proof on page 324)*

Let T be a well-defined ILP_{LOAS}^{noise} task.

1. *$ILASP3(T)$ terminates.*
2. *If T is satisfiable, then $ILASP3(T)$ returns an optimal inductive solution of T.*
3. *If T is unsatisfiable, then $ILASP3(T)$ returns* UNSATISFIABLE.

Feature	Symbol	Enabled by default	Notes
Propagation	$\mathcal{P}$	Yes	Can be disabled using the flag `-np`
Implication	$\mathcal{I}$	Yes	Can be disabled using the flag `-ni`
Schema generalisation	$\mathcal{SG}$	Yes	Can be disabled using the flag `-ng`
Maximum translation	$\mathcal{MT}$	No	Can be enabled using the flag `--max-translate`
Single weak constraint per level	$\mathcal{SWC}$	Yes	Can be disabled using the flag `-mwc`

Table 10.1: The optional features of ILASP3. For any subset S of $\{\mathcal{P}, \mathcal{I}, \mathcal{SG}, \mathcal{MT}, \mathcal{SWC}\}$, ILASP3^S denotes ILASP3 with the features in S enabled.

10.6 Optional Features of ILASP3

So far in this chapter we have presented the fundamentals of the ILASP3 algorithm. The implementation of the system also includes some extra optional features that can be used to further improve the scalability of ILASP3 on some tasks. Table 10.1 lists the optional features of ILASP3, and explains how to disable (or enable them) in the implementation of ILASP3.

In the rest of this section we explain the effects of each feature. Most of the features work using meta-level ASP programs similar to those used throughout the thesis. For brevity, we omit details of these meta-level programs.

10.6.1 Propagation

The process of translating an example into a set of hypothesis schemas may include some redundant computation, as the same schema may be computed for different examples. This extra computational time could be avoided. The ILASP3 algorithm includes a special feature called *propagation* that essentially aims at *propagating* a schema computed for one example to other examples, so avoiding re-computation of the same schema.

Once an example has been translated into a set SC of schemas, for each schema $sc \in SC$ we can compute the set of other examples e that would give rise to the same schema. For a schema sc to be computed during the translation of an example e, there must be an interpretation I extending e_{pi} such that for every $H \subseteq S_M$ conforming to sc, $I \in AS(B \cup e_{ctx} \cup H)$. We call such interpretations I the *relevant guaranteed answer sets* of sc.

Definition 10.15. Let E be a set of examples, B be a background knowledge, S_M be a hypothesis space and sc be a hypothesis schema. An interpretation I is a *relevant guaranteed answer set* of sc wrt E, B and S_M iff $\exists e \in E$ such that I extends e_{pi} and $I \in AS(B \cup e_{ctx} \cup H)$ for each $H \subseteq S_M$ that conforms to sc.

For any CDPI e such that there is a relevant guaranteed answer set of the schema sc which is also an accepting answer set of e, we can reuse sc if we need to translate e later in the ILASP3 algorithm.

When propagation is enabled in ILASP3, it takes place between lines 12 and 13 of Algorithm 10.6. We test each schema that was found when translating the previous example, to check whether it can be propagated to any other examples. The result is then stored for use in future iterations; specifically, when translating an example, we start from the set of previous schemas that guarantee it, rather than the empty set (in line 2 of Algorithm 10.2). For negative examples, we can also use the schemas immediately to approximate the coverage: if $\langle e, sc\rangle$ is in the set returned by propagate, and e is a negative example, we know that if sc is conformed to by a hypothesis H, then H does not cover e.

Propagating schemas has two key benefits: firstly, translating later examples will require fewer schemas to be computed. Secondly, propagation can reduce the number of iterations of the final ILASP3 algorithm (which fully translates one example per iteration), for the following reason. For negative examples a single schema can have a big impact (as conforming to a single schema can imply that *many* negative examples are not covered, and so a large penalty needs to be paid). Without propagation, many negative examples would need to be translated in order to prove that a certain hypothesis has a very high penalty, and so the search may not discount this hypothesis for many iterations. Note that for positive examples, the schemas do not have an immediate effect, as the set of schemas SC found for an example e may still be partial, and so it is not necessarily the case that any hypothesis that does not conform to any schema in SC does not cover e (so ILASP3 cannot use these schemas until e is translated).

10.6.2 Implication

We showed in Chapter 8 that the concept of relevant examples can be used to dramatically increase the performance of ILASP for tasks with large numbers of non-noisy examples. The main reason for this improvement is that there may be many examples which are covered (or not covered) for the same reasons (i.e. by the same classes of hypotheses). When the examples are known not to be noisy, adding a single example to the set of relevant examples means that it *must* be covered by any hypothesis found by the algorithm in future iterations. If other examples are covered by exactly the same class of hypotheses, then they will not be added to the relevant example set, as they are now guaranteed to be covered. With noisy examples, this technique of using a set of relevant examples is not so successful. This is because even when an example is added to the relevant example set, it does not *have* to be covered (hypotheses may pay a penalty in order to not cover the example). Consequently, this may result in adding large numbers of examples to the relevant example set (and thus wasting time translating many examples into hypothesis schemas).

The *implication* feature of ILASP3 allows us to "boost" the penalty of a single translated positive example e (or brave ordering o), in order to avoid translating other positive examples (or brave orderings) that are not covered by any hypothesis that does not cover e (or o).

Definition 10.16. Let T be the ILP_{LOAS}^{noise} task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$, $\langle e, SC\rangle$ be a coverage constraint and ex be an example in $E^+ \cup O^b$. ex is said to *imply* SC if and only if every hypothesis that covers ex conforms to at least one schema in SC.

For any positive example (or brave ordering) ex that implies a coverage constraint $\langle e, SC\rangle$, we can add the coverage constraint $\langle ex, SC\rangle$ to our set of coverage constraints. We do this between lines 12 and 13 in Algorithm 10.6. The advantage of this is that our approximation of coverage considers more of the examples earlier (we do not necessarily need to wait for a positive example or brave ordering to be translated to know that it is not covered by a hypothesis). Thus, the ILASP3 algorithm generally requires fewer iterations with this feature than without it.

Figure 10.1 shows the output of ILASP3 with implication and propagation enabled. ILASP3 was run with the debug ($-$`d`) option, so that it printed the intermediate hypotheses found in each iterations, along with their expected and actual penalties (the score of the hypothesis minus the length of the hypothesis). This output shows the power of both advanced features. Consider Iteration 2. The iteration begins with the hypothesis $H = \{$`:- edge(V0, V1).`$\}$ being found. This has length 1 and expected penalty 0, meaning that given the coverage constraints that have been computed so far, we have no reason to believe that this hypothesis does not cover every example. However, in reality, there are 93 examples that are not covered by H (this is shown in the output as the *unexpected penalty*). ILASP3 picks one of these uncovered examples (`p63`) and translates it. The line that says "`Translated p63 into 6 + 71 schemas.`", means that there were already 6 schemas for `p63` (which had been propagated to `p63` in the first iteration), and 71 new schemas needed to be computed in order to complete the translation. The final line shows the result of the implication step. The line says that if `p63` is not covered, then none of the examples in $\{$`p30, p35, p45, p61, p63, p71, p76, p79, p88, p89, p92, p93`$\}$ are covered. This dramatically reduces the number of iterations, as it means that every hypothesis must either cover `p63` or pay the penalty for all 12 examples. This means that in the next iteration, the computed hypothesis has the expected score of 12 (11 higher than in the previous iteration). In later iterations, the effect of the propagation step becomes more significant; for example, in iteration 8, the line "`Translated p77 into 154 + 8 schemas.`" means that before the iteration, `p77` already had 154 schemas which had been propagated from other examples, meaning that only 8 further schemas needed to be computed to complete the translation of `p77`. Propagation also has the effect that only one negative example is fully translated during the whole execution. This is because the schemas found throughout the execution for the positive examples (and the one fully translated negative example) are propagated to negative examples and are used immediately.

10.6.3 Schema Generalisation

The schemas computed by ILASP3 are often more specific than is necessary. We illustrate this in Example 10.22.

Example 10.22. *Consider a task with background knowledge $B = \emptyset$, the hypothesis space $S_M = \{\mathtt{h_1} : \mathtt{p.}\}$. Consider an "empty" CDPI example $\langle\langle\emptyset, \emptyset\rangle, \emptyset\rangle$. ILASP3 will compute the translation $\{\langle\emptyset, \{\mathtt{h_1}\}\rangle, \langle\{\{\mathtt{h_1}\}\}, \emptyset\rangle\}$ corresponding to the two possible answer sets $\emptyset$ and $\{\mathtt{p}\}$. The translation means*

```
%% Iteration 1
%% Found Hypothesis:
%% |H| = 0
%% expected penalty = 0
%% unexpected penalty = 110
%% Adding example: n50

%% Iteration 2
%% Found Hypothesis:
%%  :- edge(V0, V1).
%% |H| = 1
%% expected penalty = 0
%% unexpected penalty = 93
%% Adding example: p63
%% Translated p63 into 6 + 71 schemas.
%% I = { p30, p35, p45, p61, p63, p71,
        p76, p79, p88, p89, p92, p93 }

%% Iteration 3
%% Found Hypothesis:
%%  :- in(V0,V1), in(1,V0).
%%  :- node(V0), not reach(V0).
%% 0 {in(V0,V1) } 1 :- edge(V0, V1).
%% reach(V1) :- in(V0,V1), in(1,V2).
%% |H| = 10
%% expected penalty = 2
%% unexpected penalty = 50
%% Adding example: p44
%% Translated p44 into 48 + 43 schemas.
%% I = { p44 }

%% Iteration 4
%% Found Hypothesis:
%%  :- in(V0,V1), in(1,V0).
%%  :- node(V0), not reach(V0).
%% 0 {in(V0,V1) } 1 :- edge(V0, V1).
%% reach(V1) :- in(V0,V1), in(1,V2).
%% |H| = 10
%% expected penalty = 3
%% unexpected penalty = 49
%% Adding example: p43
%% Translated p43 into 74 + 11 schemas.
%% I = { p21, p43 }

%% Iteration 5
%% Found Hypothesis:
%%  :- in(V0,V1), in(1,V0).
%%  :- node(V0), not reach(V0).
%% 0 {in(V0,V1) } 1 :- edge(V0, V1).
%% reach(V1) :- in(V0,V1), in(1,V2).
%% |H| = 10
%% expected penalty = 5
%% unexpected penalty = 47
%% Adding example: p17
%% Translated p17 into 120 + 4 schemas.
%% I = { p17, p22, p28, p41, p57 }

%% Iteration 6
%% Found Hypothesis:
%%  :- in(V0,V1), in(1,V0).
%%  :- node(V0), not reach(V0).
%% 0 {in(V0,V1) } 1 :- edge(V0, V1).
%% reach(V1) :- in(V0,V1), in(1,V2).
%% |H| = 10
%% expected penalty = 10
%% unexpected penalty = 42
%% Adding example: p82
%% Translated p82 into 86 + 9 schemas.
%% I = { p18, p82 }

%% Iteration 7
%% Found Hypothesis:
%%  :- in(V0,V1), in(1,V0).
%%  :- node(V0), not reach(V0).
%% 0 {in(V0,V1) } 1 :- edge(V0, V1).
%% reach(V1) :- in(V0,V1), in(1,V2).
%% |H| = 10
%% expected penalty = 12
%% unexpected penalty = 40
%% Adding example: p45
%% Translated p45 into 80 + 13 schemas.
%% I = { p35, p45, p93 }

%% Iteration 8
%% Found Hypothesis:
%%  :- edge(V0, V1).
%% |H| = 1
%% expected penalty = 22
%% unexpected penalty = 71
%% Adding example: p77
%% Translated p77 into 154 + 8 schemas.
%% I = { p34, p62, p70, p77 }

%% Iteration 9
%% Found Hypothesis:
%%  :- node(V0), not reach(V0).
%% reach(V0) :- in(1,V0).
%%  :- in(V0,V1), in(V0,V2), V1 != V2.
%% 0 {in(V0,V1) } 1 :- edge(V0, V1).
%% reach(V1) :- in(V0,V1), reach(V0).
%% |H| = 13
%% expected penalty = 10
%% unexpected penalty = 0

%% Final hypothesis:
:- node(V0), not reach(V0).
reach(V0) :- in(1,V0).
:- in(V0,V1), in(V0,V2), V1 != V2.
0 {in(V0,V1) } 1 :- edge(V0, V1).
reach(V1) :- in(V0,V1), reach(V0).
```

Figure 10.1: The output of ILASP3 with the two advanced features on an ILP_{LOAS}^{noise} task. The target hypothesis describes what it means to be a Hamiltonian graph.

that every hypothesis must either contain $\mathtt{h_1}$ *or not contain* $\mathtt{h_1}$*. This is equivalent to the translation* $\{\langle\emptyset,\emptyset\rangle\}$ *which accepts all hypotheses.*

Overly specific schemas can lead to the translation process taking longer than necessary. If *schema generalisation* is enabled then after a schema $\langle D, V\rangle$ is computed as the translation of an answer set A, ILASP3 will attempt to remove unnecessary elements of V.

10.6.4 Maximum Translation

When ILASP3 finds that its approximation for the score of the current hypothesis is incorrect, it translates a single example that it has incorrectly predicted as covered. Once this example has been translated, ILASP3 searches for another hypothesis and continues. An alternative approach would be to continue translating examples until the approximation of the score for the current hypothesis is correct. Note that if this approach is combined with implication, this does not mean translating every incorrectly predicted example: if two examples e_1 and e_2 were mispredicted e_1 not being covered may imply that e_2 is not covered; hence if e_1 is translated first, e_2 will not need to be translated[1].

10.6.5 Restricting Hypotheses to a Single Weak Constraint at each Priority Level

Unlike the other optional features, this feature changes the hypotheses that can be learned by ILASP3, restricting hypotheses so that they cannot contain more than one weak constraint at the same level. This can significantly improve the performance of ILASP3, as it means that checking whether an ordering schema is conformed to no longer involves computing the optimisation differences at each level (summing the optimisation difference of each individual weak constraint). Instead we only need to check whether the optimisation difference of a single weak constraint (plus the optimisation difference of the weak constraints in the background knowledge at that level) is positive or negative.

10.7 Related Work

XHAIL is a brave induction system that avoids the need to enumerate the entire hypothesis space. XHAIL has three phases: abduction, deduction and induction. In the first phase, XHAIL uses abduction to find a minimal subset of some specified ground atoms. These atoms, or a generalisation of them, will appear in the head of some rule in the hypothesis. The deduction phase determines the set of ground literals which could be added to the body of the rules in the hypothesis. The set of ground rules constructed from these head and body literals is called a kernel set. The final induction

[1]In the current implementation of ILASP3, we have only implemented this feature for positive examples (ILASP3 will continue translating positive examples until the approximation of the score of the current hypothesis is correct over the positive examples). In principle, this could be extended to negative examples, but as implication only works for positive examples, this could lead to needing to translate many more examples.

phase is used to find a hypothesis which is a generalisation of a subset of the kernel set that proves the examples. The public implementation of XHAIL [BR15b] has been extended to handle noise by setting penalties for the examples similarly to ILP^{noise}_{LOAS}. However, as shown in Example 10.23 XHAIL is not guaranteed to find an optimal inductive solution of a task.

Example 10.23. *Consider the following noisy task, in the XHAIL input format:*

```
p(X) :- q(X, 1), q(X, 2).          #modeh r(+s).
p(X) :- r(X).                      #modeh q(+s2, +t).
s(a).   s(b).   s2(b).             #example not p(a)=50.
t(1).   t(2).                      #example p(b)=100.
```

This corresponds to a hypothesis space that contains two facts F_1 = `r(X)`, F_2 = `q(X, Y)` (in XHAIL, these facts are implicitly "typed", so the first fact, for example, can be thought of as the rule `r(X) :- s(X)`*). The two examples have penalties 50 and 100 respectively. There are four possible hypotheses: $\emptyset$, F_1, F_2 and $F_1 \cup F_2$, with scores 100, 51, 1 and 52 respectively. XHAIL terminates and returns F_1, which is a suboptimal hypothesis.*

The issue is with the first step. The system finds the smallest abductive solution, {`r(b)`*} and as there are no body declarations in the task, the kernel set contains only one rule:* `r(b) :- s(b)`*. XHAIL then attempts to generalise to a first order hypothesis that covers the examples. There are two hypotheses which are subsets of a generalisation of* `r(b)` *(F_1 and $\emptyset$); as F_1 has a lower score than $\emptyset$, XHAIL terminates and returns F_1. The system does not find the abductive solution {*`q(b, 1), q(b, 2)`*}, which is larger than {*`r(b)`*} and is therefore not chosen, even though it would eventually lead to a better solution than {*`r(b)`*}.*

It should be noted that XHAIL does have an iterative deepening *feature for exploring non-minimal abductive solutions, but in this case using this option XHAIL still returns F_1, even though F_2 is a more optimal hypothesis. Even when iterative deepening is enabled, XHAIL only considers non-minimal abductive solutions if the minimal abductive solutions do not lead to any non-empty inductive solutions.*

In comparison to ILASP3, in some problem domains, XHAIL is more scalable as it does not start by enumerating the hypothesis space in full. On the other hand, as shown by Example 10.23, XHAIL is not guaranteed to find the optimal hypothesis, whereas ILASP3 is. ILASP3 also solves ILP^{noise}_{LOAS} tasks, whereas XHAIL solves $n(ILP_b)$ tasks, which means that due to the generality results in the previous chapter ILASP3 is capable of learning programs which are out of reach for XHAIL no matter what examples are given.

Inspire [KSS17] is an ILP system based on XHAIL, but with some modifications to aid scalability. The main modification is that some rules are "pruned" from the kernel set before XHAIL's inductive phase. Both XHAIL and Inspire use a meta-level ASP program to perform the inductive phase, and the ground kernel set is generalised into a first order kernel set (using the mode declarations to determine which arguments of which predicates should become variables). Inspire prunes rules which have fewer than Pr instances in the ground kernel set (where Pr is a parameter of Inspire). The intuition is that if a rule is necessary to cover many examples then it is likely to have many ground instances in the kernel. Clearly this is an approximation, so Inspire is not guaranteed to find the optimal hypothesis in the inductive phase. In fact, as XHAIL is not guaranteed to find the optimal inductive solution of the task (as it may pick the "wrong" abductive solution), this means that Inspire may be even further from the optimal. In Chapter 11 we evaluate ILASP3 on the same sentence chunking dataset that Inspire was evaluated on in [KSS17], and show that ILASP3 achieves a higher F_1 score than Inspire.

Summary

In this chapter we have presented the ILASP3 algorithm, which is specifically targeted at learning ASP programs from noisy examples. In the next chapter, we evaluate the scalability of ILASP3 on a variety of learning tasks and compare its performance to other ILP systems such as δILP, OLED and Inspire.

Chapter 11

Evaluation

In this chapter we evaluate the performance of ILASP3, comparing it with previous ILASP systems and other ILP systems on noisy datasets. Firstly, we reinvestigate the Hamilton Graph and Journey Preference learning settings from Chapter 8, extended with noise. The value of using these purely synthetic datasets is that we can control the amount of noise and investigate how the performance of ILASP3 varies with the amount of noise.

The remaining experiments in this chapter compare the accuracy of ILASP3 with the accuracy of other algorithms. In Section 11.2, we evaluate ILASP3 on several datasets that have been used to evaluate other ILP algorithms for learning from noisy data.

Section 11.2.1 compares ILASP3 with OLED [KAP16] on the CAVIAR dataset, which contains data gathered from a video stream, annotated to indicate the interactions of people in the video. We show that ILASP's F_1 score compares favourably to OLED's. Next, in Section 11.2.2 we evaluate ILASP3 on a dataset for *sentence chunking*, which has been used to evaluate the Inspire algorithm. We show that the F_1 score of the hypotheses learned by ILASP3 is higher than the corresponding F_1 score for Inspire in every experiment.

In Sections 11.2.3 and 11.2.4, we evaluate the accuracy of ILASP3 on two real user preference datasets (the *car preference* dataset [ASB$^+$13] and the *SUSHI* dataset [KKA10]), and show that ILASP3's accuracy is higher than that reported in a recent paper [QK17] applying ILP to preference learning.

Finally, in Section 11.2.5, we evaluate ILASP3 on a recent set of definite clause learning problems, which have been used to evaluate the δILP [EG18] algorithm. We refute the claims, made in [EG18], that ILP approaches are unable “to handle noisy, erroneous, or ambiguous data” and that “If the positive or negative examples contain any mislabelled data, [ILP approaches] will not be able to learn the intended rule.”

11.1 Synthetic Datasets

11.1.1 Hamilton Graphs

In this experiment, we used ILASP3 to learn the definition of what it means for a graph to be Hamiltonian. For $n = 20, 40, \ldots, 200$, we generated n random graphs of size 1 to 4, half of which were Hamiltonian. We then labeled the graphs as either positive or negative, where positive indicates that the graph is Hamiltonian. In this experiment, learning tasks were encoded using the context-dependent representation (Hamilton B) from Chapter 8.

Comparison Between Different ILASP Versions

As the ILASP systems are the only systems capable of learning constraints and choice rules, we cannot compare the performance of ILASP3 with the performance of other ILP systems on this problem. We can, however, compare ILASP3 to previous ILASP systems. To test the computational time of ILASP3 against ILASP2 and ILASP2i, and to show the effect of the additional features of ILASP3 presented in Section 10.6, we first ran a small set of experiments on the Hamilton dataset. Table 11.1 shows the results of these experements[1].

Each row in Table 11.1 was generated by running ILASP2, ILASP2i, and 16 variations of the ILASP3 algorithm on a single ILP^{noise}_{LOAS} task with 5% noise. With only 20 examples, ILASP2 was the fastest of the 18, and ILASP2i was faster than each of the ILASP3 algorithms. This demonstrates the expected result, that for tasks with few examples, where the size of the ILASP2 meta program is not an issue, ILASP2 returns a solution much quicker than the other algorithms, which perform computations that are not necessary for such small tasks (e.g. computing the hypothesis schemas). On the other hand, as the number of examples increased, both ILASP2 and ILASP2i's computation times increased rapidly, and in every task with more than 100 examples, both systems timed out (i.e., they did not solve the tasks within the 30 minute time limit).

The last 16 columns of Table 11.1 show the performance of the different variations of ILASP3. Implication and propagation on their own (without any other optional features) have a positive effect, particularly on large tasks; whereas schema generalisation and maximum translation on their own have a negative effect, showing that they are not well suited to this kind of task. In general, maximum translation does not work well without implication and propagation enabled, as it can result in many of the examples being unnecessarily translated, and many extra hypothesis schemas being generated. In the Hamilton learning setting, after the first iteration, a common hypothesis for ILASP3 to learn is `:-node(X).`, which results in all of the positive examples being covered and none of the negative examples being covered. This means that ILASP3 with maximum translation enabled needs to translate many of the positive examples, especially if implication is not enabled. In general, schema generalisation is useful when some atoms in the language of the task are irrelevant to particular examples,

[1]The tasks in this table are available at `https://www.doc.ic.ac.uk/~ml1909/ILASP/`

#examples	time/s						
	2	2i	3	$3^{\mathcal{I}}$	$3^{\mathcal{P}}$	$3^{\mathcal{SG}}$	$3^{\mathcal{MT}}$
20	**2.4**	9.0	48.0	46.9	17.7	78.5	54.0
40	60.8	156.1	92.5	52.1	**28.0**	137.9	125.4
60	131.6	226.2	127.5	93.3	42.6	192.2	218.6
80	974.9	1130.5	156.3	102.0	54.3	241.4	300.5
100	542.3	363.1	157.8	96.6	70.6	261.0	394.3
120	-	-	212.3	163.9	**71.8**	335.6	450.6
140	-	-	258.4	174.6	99.4	430.2	600.8
160	-	-	234.2	144.6	114.0	421.5	654.9
180	-	-	280.6	184.9	168.9	456.0	747.7
200	-	-	270.6	162.4	120.7	441.7	741.7

#examples	time/s					
	$3^{\mathcal{I},\mathcal{P}}$	$3^{\mathcal{I},\mathcal{SG}}$	$3^{\mathcal{I},\mathcal{MT}}$	$3^{\mathcal{P},\mathcal{SG}}$	$3^{\mathcal{P},\mathcal{MT}}$	$3^{\mathcal{SG},\mathcal{MT}}$
20	18.2	66.6	48.3	31.7	25.1	101.3
40	28.2	93.1	86.7	40.7	32.2	213.7
60	**37.5**	141.2	122.3	47.9	49.1	354.8
80	**48.3**	148.9	142.3	58.1	65.4	518.7
100	**61.5**	166.5	151.3	88.3	84.4	675.1
120	**71.8**	244.7	176.4	83.4	97.8	731.9
140	**94.7**	292.8	296.2	107.6	120.9	1061.2
160	**78.7**	232.3	227.0	112.7	141.4	1147.7
180	153.3	308.7	239.9	176.6	191.0	1330.6
200	**99.0**	267.9	271.7	126.8	150.8	1340.1

#examples	time/s				
	$3^{\mathcal{I},\mathcal{P},\mathcal{SG}}$	$3^{\mathcal{I},\mathcal{P},\mathcal{MT}}$	$3^{\mathcal{I},\mathcal{SG},\mathcal{MT}}$	$3^{\mathcal{P},\mathcal{SG},\mathcal{MT}}$	$3^{\mathcal{I},\mathcal{P},\mathcal{SG},\mathcal{MT}}$
20	24.2	18.0	83.1	25.9	24.9
40	33.2	30.0	145.1	41.3	41.4
60	44.8	42.4	194.0	55.5	50.1
80	52.2	51.5	226.8	79.0	61.9
100	80.6	71.1	248.4	99.4	85.3
120	73.6	74.4	289.5	101.3	77.8
140	97.4	100.5	486.3	129.5	111.2
160	86.3	95.8	384.5	136.7	107.2
180	**153.0**	158.1	391.7	200.0	168.6
200	107.7	110.0	461.4	158.8	126.5

Table 11.1: The running times of ILASP2 (extended), ILASP2i (extended) and ILASP3 for Hamilton problems with 5% noise and varying numbers of examples. "-" represents the case of a time out (where the system did not return a solution in 30 minutes). Note that we do not test the effect of the "single weak constraint" feature, as this task does not use weak constraints.

and lead to multiple specific schemas, where one more general schema would be sufficient. In the Hamilton setting, for every example e, the relevant Herbrand base of $B \cup H_T \cup e_{ctx}$ (where H_T is the target hypothesis) only contains atoms that are relevant to whether the example represents a Hamilton graph. This may mean that not many schemas are actually being significantly generalised, and that there is a very little positive effect (or no positive effect at all) of using schema generalisation in this learning setting. The negative effects of schema generalisation (i.e. the extra computation involved in attempting to generalise the schema) therefore cost more time than is saved by computing more general schemas.

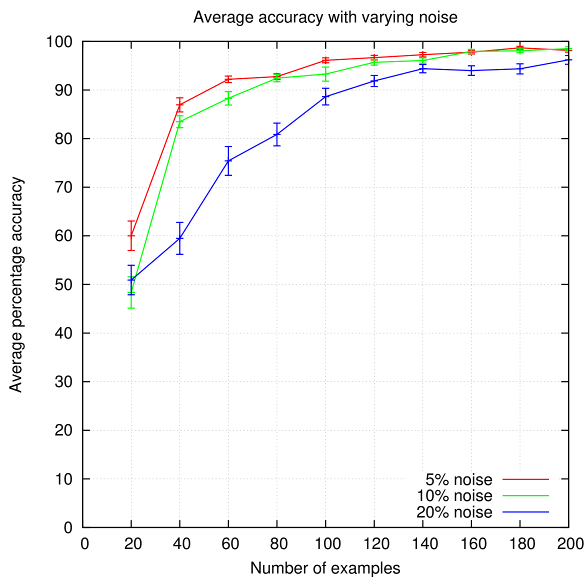
The negative effects of schema generalisation and maximum translation can also be seen for other variations of the algorithm – in general, the different variations perform better on these tasks without these two features. This difference is much less significant when the features are combined with propagation, which significantly reduces the number of schemas that are computed, and thus lowers the negative impact of these two features. For example, in nine out of ten cases, there is less than ten seconds difference between ILASP3$^{\mathcal{I},\mathcal{P}}$ and ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG}}$. In contrast, in general the different variations perform better with implication and propagation than without.

Both implication and propagation make a significant difference to the computation time as both reduce the number of hypothesis schemas that need to be computed by ILASP3. Interestingly, although the ILASP3$^{\mathcal{I},\mathcal{P}}$ algorithm performs the best on most tasks, there are some tasks where ILASP3$^{\mathcal{I},\mathcal{P}}$'s computation time is very close to ILASP3$^{\mathcal{P}}$'s (e.g. with 120 and 140 examples). This demonstrates that for some tasks, although implication might reduce the number of iterations needed, propagation without implication can be sufficient, as the propagation can make these extra iterations so short that the process of checking for implications becomes an overhead. This is not always the case however, as there are some tasks in the table on which $ILASP3^{\mathcal{I},\mathcal{P}}$ significantly outperforms ILASP3$^{\mathcal{P}}$ (e.g. with 160 and 200 examples).

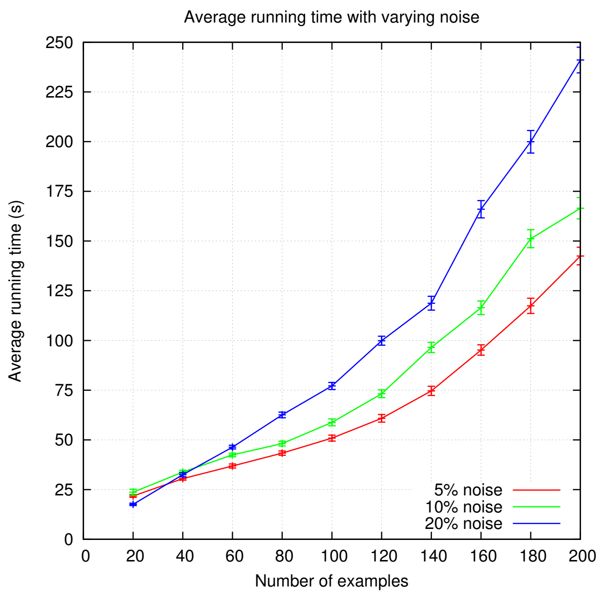
Hamilton Experiments with Varying Degrees of Noise

In this experiment, we used ILASP3$^{\mathcal{I},\mathcal{P}}$ (the fastest variation on our "one-off" experiments[2]) with varying degrees of noise. We ran three sets of experiments to evaluate the performance of ILASP3$^{\mathcal{I},\mathcal{P}}$ on the Hamilton learning problem with 5%, 10% and 20% of the examples being labeled incorrectly. In each experiment, an equal number of Hamiltonian graphs and non-Hamiltonian graphs were randomly generated. n% of the examples were chosen at random to be labeled incorrectly. These n% of examples were labeled as positive (resp. negative) if the graph was not (resp. was) Hamiltonian. The remaining examples were labeled correctly (positive if the graph was Hamiltonian; negative if the graph was not Hamiltonian). Figure 11.1 shows the average accuracy and running time of ILASP3$^{\mathcal{I},\mathcal{P}}$ with up to 200 example graphs. Each experiment was repeated 50 times (with different randomly generated examples). In each case, the accuracy was tested by generating a further 1000 graphs and using the learned hypothesis to classify the graphs as either Hamiltonian or non-Hamiltonian (based on whether the hypothesis was satisfiable when combined with the representation of the graph).

[2]Note that on average, the different variations affect only the computation time, and not the accuracy of the solution, as they each return an optimal solution of the task if they terminate.



(a)



(b)

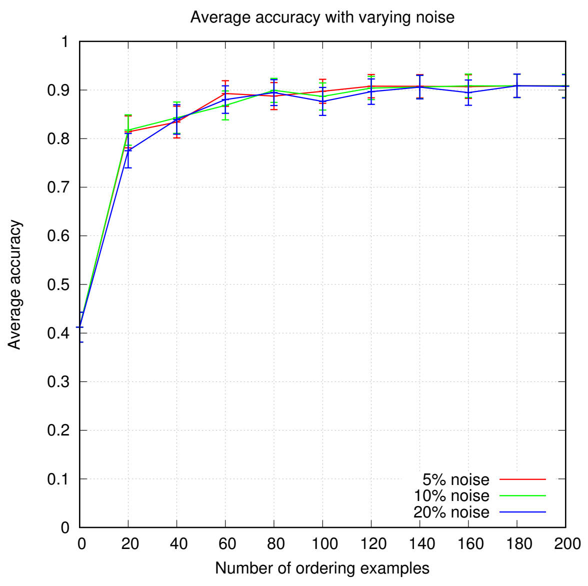
Figure 11.1: (a) the average computation time and (b) average accuracy of ILASP3$^{\mathcal{I},\mathcal{P}}$ for the Hamilton learning task, with varying numbers of examples, and varying noise.

The experiments show that on average ILASP3$^{\mathcal{I},\mathcal{P}}$ is able to achieve a high accuracy (of well over 90%), even with 20% of the examples labeled incorrectly. A larger percentage of noise means that ILASP3$^{\mathcal{I},\mathcal{P}}$ requires a larger number of examples to achieve a high accuracy. This is to be expected, as with few examples, the hypothesis is more likely to "overfit" to the noise, or pay the penalty of some non-noisy examples. With large numbers of examples, it is more likely that ignoring some non-noisy examples would mean not covering others, and thus paying a larger penalty. The computation time rises in all three graphs as the number of examples increases. This is because larger numbers of examples are likely to require larger numbers of iterations of the ILASP3$^{\mathcal{I},\mathcal{P}}$ algorithm. Similarly, the more noise in the example, the longer the computation time, as more noise is also likely to mean a larger number of iterations.

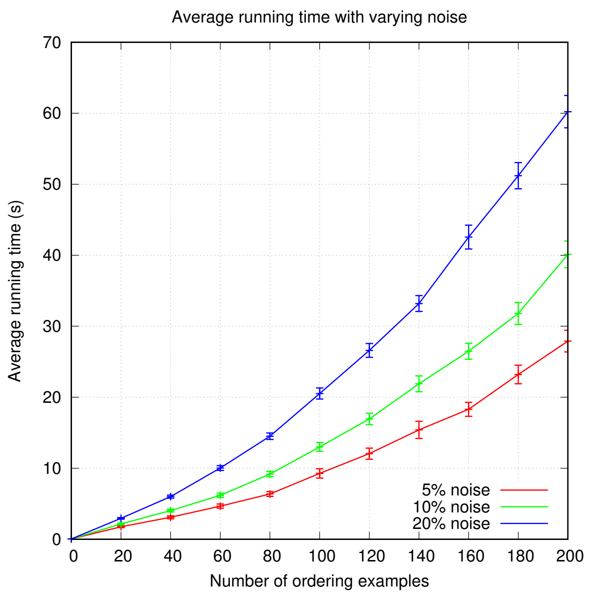
11.1.2 Noisy Journey Preferences

Recall the journey preferences setting from Chapter 8, where the goal is to learn a user's preferences from a set of ordered pairs of journeys. In this section, we consider a noisy version of the experiment, testing how ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG},\mathcal{SWC}3}$ performs on tasks with different percentages of noise and numbers of examples.

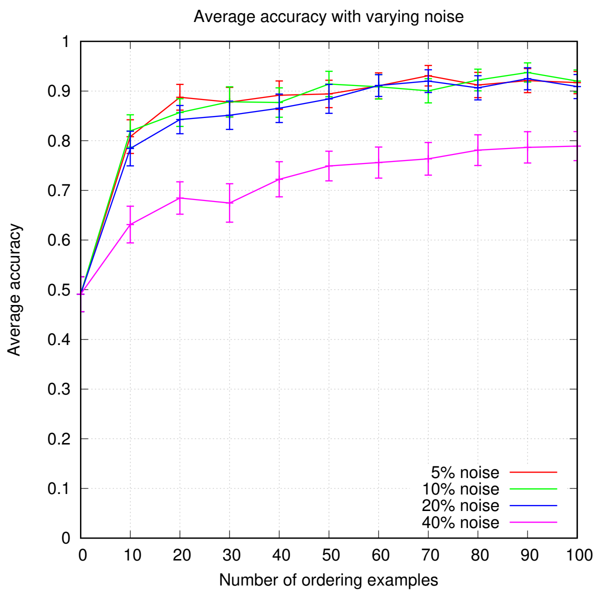
[3]In this experiment, ILASP used clingo 5 to solve meta-level ASP programs, with unsatisfiable core optimisation enabled.



(a)



(b)



(c)

Figure 11.2: (a) and (c) show the average computation time and (b) shows average accuracy of ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG},\mathcal{SWC}}$ for the journey preference learning task, with varying numbers of examples, and varying noise. Each of the point in the graphs is an average of ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG},\mathcal{SWC}}$'s performance on 50 different tasks.

In each experiment, we selected between 1 and 3 weak constraints from the same hypothesis space as was used in Chapter 8. For each set of weak constraints, we then ran learning tasks with 0, 20, ..., 200 examples and with 5%, 10% and 20% noise. The examples for these learning tasks were generated from the weak constraints such that half of the ordering examples represented journeys J_1 and J_2 such that J_1 was strictly prefered to J_2 given the weak constraints, and the other half represented journeys such that J_1 was equally prefered to J_2. Depending on the level of noise, either 5%, 10% or 20% of the examples were given with the wrong operator ($>$ instead of $<$ and $\neq$ instead of $=$). In each of these learning tasks, each ordering example was given a penalty of 1.

Our results (Figure 11.2 (a)) show that even with 20% noise, ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG},\mathcal{SWC}}$ was able to learn hypotheses with an average accuracy of over 90%. We found that there was not much difference between ILASP's accuracy with 5%, 10% and 20% noise, although the noisier tasks have a higher computation time (this is shown in Figure 11.2 (b)), as in general ILASP3 requires more iterations on noisier tasks because more examples need to be translated. Even with 20% noise and 200 ordering examples, ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG},\mathcal{SWC}}$ terminated in just over 60 seconds on average.

As the results for 20% noise were so close to the results for 5% noise, we ran a further set of examples to check that there was some limit to the level of noise where ILASP3 would no longer learn such an accurate hypothesis[4]. In this second set of experiments, we tested ILASP with up to 40% noise, and investigated with 0, 10, . . ., 100 examples. With 40% noise, the accuracy was lower, but we still achieved an average accuracy of just under 80%.

11.2 Comparisons with Other Algorithms

In addition to our own synthetically generated datasets, we also experimented with using ILASP3 on several datasets that have previously been used to evaluate other systems. Several of these consist of "real" data (in contrast to the synthetically generated datasets that we have presented so far). In real datasets there is no way of knowing what proportion of the examples are noisy, and we do not know what the "target" hypothesis is. These datasets allow us to evaluate ILASP3 more realistically – in practical settings, it is unlikely that we would know the target hypothesis (if we did, there would be no point using a system to learn it), or the proportion of noise.

11.2.1 CAVIAR Dataset

In this experiment we tested ILASP3$^{\mathcal{I}}$ on the recent CAVIAR dataset that has been used to evaluate the ILED [KAP15] and OLED [KAP16] systems, which are extensions of the XHAIL [Ray09] algorithm aimed at learning Event Calculus [KS86] theories. The dataset contains data gathered from a video stream. Information such as coordinates of people has been extracted from the stream, and humans have annotated the data to specify when any pairs of people are interacting. Note that we use ILASP3$^{\mathcal{I}}$

[4]If there was no such limit, and we could achieve a high accuracy, even with very high levels of noise, then this would indicate that our hypothesis space was too restrictive, and it was impossible to learn anything other than an accurate hypothesis.

in this section because all examples lead to exactly one hypothesis schema, and this is unlikely to be the same hypothesis schema for any other example. For this reason, propagation would be an overhead that is unlikely to give any gain in performance and schema generalisation is unnecessary.

Specifically, we consider a task from [KAP16], in which the aim is to learn rules to define initiating and terminating conditions for two people meeting. The background knowledge (Figure 11.3 (a)) of this task has rules determining whether a fluent (in this case two people meeting) holds at a given time. This can hold in two cases: either the two people were meeting at the previous timepoint and their meeting was not terminated at this previous timepoint; or their meeting could have been initiated at the current timepoint.

In the evaluation of the OLED system, examples were generated for every pair of consecutive timepoints t and $t+1$. Each example is a pair $\langle N \cup A_t, A_{t+1}\rangle$, where N is the "narrative" at time t (a collection of information about the people in the video stream, such as their location and direction), and A_i is the "annotation" at time i (exactly which pairs of people in the video have been labeled as meeting). This is very simple to express using context-dependent examples. The context of an example is simply the narative and annotation of time t together with a set of constraints that enforce that the meetings at time t are exactly those in the annotation (denoted by the *goal* predicate). One such context-dependent example is shown in Figure 11.3 (b).

```
  holdsAt(F,Te) :- initiatedAt(F,Ts), next_time(Ts, Te).
  holdsAt(F,Te) :- holdsAt(F,Ts), not terminatedAt(F,Ts), next_time(Ts, Te).
```

(a)

```
<p_12440, 1, <<{}, {}>, {

  goal(holdsAt(meeting(id0,id1),2)).  goal(holdsAt(meeting(id1,id0),2)).

  :- holdsAt(meeting(P1,P2),2), not goal(holdsAt(meeting(P1,P2),2)).
  :- not holdsAt(meeting(P1,P2),2), goal(holdsAt(meeting(P1,P2),2)).

  person(id0).                        person(id1).
  happensAt(walking(id0),1).          happensAt(active(id1),1).
  holdsAt(meeting(id0,id1),1).        holdsAt(meeting(id1,id0),1).
  dist(id0,id1,1,36).                 dist(id1,id0,1,36).
}>>.
```

(b)

Figure 11.3: (a) An extract from the background knowledge, and (b) a CDPI example for the CAVIAR dataset.

In total there are 24530 consecutive pairs in the dataset[5]. We performed 10 fold cross validation by randomly partitioning the dataset. As there were only 22 timepoints where the group of people meeting was different to the timepoint before, we gave a high penalty (of 100) to not covering each of these examples. Effectively this is the same as oversampling this class of examples. If we had given all examples a penalty of 1, then we would have most likely learned the empty hypothesis, as the 22 examples in a task comprised of many thousands of examples are likely to be treated as noise.

We compare ILASP3$^{\mathcal{I}}$ to OLED on the measures of precision, recall and the F_1 score[6]. We achieved a precision of 0.832 and a recall of 0.853, giving an F_1 score of 0.842, compared with OLED's precision of 0.678 and recall of 0.953, with an average F_1 score of 0.792. Our average running time was significantly higher at 576.3s compared with OLED's 107s. This is explained by the fact that the OLED system computes hypotheses through theory revision, iteratively processing examples in sequence to continuously revise its hypothesis. This means that OLED is not guaranteed to find an optimal solution of a learning task. In contrast, ILASP3$^{\mathcal{I}}$ is guaranteed to find an optimal solution of the task.

We note several key differences between our experiments and those reported in [KAP16]. Firstly, to reduce the number of irrelevant possible answer sets (which lead to irrelevant schemas, and slow computation), we put a constraint on the hypothesis space, stating that any rule for `terminatedAt(meeting(V1, V2), T)` had to contain `holdsAt(meeting(V1, V2), T)` in the body, which ensures that a fluent can only be terminated if it is currently happening. Similarly, any rule for `initiatedAt(meeting(V1, V2), T)` had to contain `not holdsAt(meeting(V1, V2), T)` in the body. OLED does not employ this constraint, but when processing an example pair of timepoints, only considers learning a new rule for `initiatedAt`, for example, if two people are meeting at time $t + 1$, but not at time t.

The second difference in our experiment is that we enumerate the hypothesis space in full. As the hypothesis space in this task is potentially very large, we placed several "common sense" constraints on the rules in the hypothesis space; for instance, two people cannot be close to each other at the same time as being far away from each other (we did not generate rules with both conditions in the body). In total our hypothesis space contained 3370 rules. OLED does not enumerate the hypothesis space in full, but uses an approach similar to XHAIL, and derives a "bottom clause" from the background knowledge and the example. In most cases (unless there is noise in the narrative, suggesting that two people are both close to and far away from each other) OLED will therefore only consider rules that respect our "common sense" constraints, as other rules would not be derivable from the background knowledge and example.

[5]We used the data available from `http://users.iit.demokritos.gr/~nkatz/OLED-data/caviar.json.tar.gz`

[6]Let *tp*, *tn*, *fp*, *fn* represent the number of true positives, true negatives, false positives and false negatives achieved by a classifier on some test data. The *precision* of the classifier (on this test data) is equal to $\frac{tp}{tp+fp}$ and the *recall* is equal to $\frac{tp}{tp+fn}$. The F_1 score is equal to $\frac{2 \times precision \times recall}{precision + recall}$.

11.2.2 Sentence Chunking

In [KSS17] the Inspire system was evaluated on a sentence chunking [TKSB00] dataset [AGALG$^+$16]. The task in this setting is to learn to split a sentence into short phrases called chunks. For instance, according to the dataset [AGALG$^+$16], the sentence "Thai opposition party to boycott general election." should be split into the three chunks "Thai opposition party", "to boycott" and "general election". [KSS17] describes how to transform each sentence into a set of facts consisting of part of speach (POS) tags. We use each of these sets of facts as the context of a context dependent example. In Inspire (which is a brave induction system), the facts are all put into the background knowledge. The task is to learn a predicate `split/1`, which expresses where sentences should be split.

Note that the Inspire tasks in [KSS17] group the muliple `split` examples for a chunk into a single example (using a `goodchunk` predicate); for example, the background knowledge may contain a rule `goodchunk(1):-split(1), not split(2), not split(3), split(4)` expressing that there is a chunk between words 1 and 4 of a sentence. It is noted in [KSS17] that this increased performance. This is because there is no benefit in covering some of the `split` atoms that make up a chunk, as hypotheses are tested over full chunks rather than splits. The ILASP task represents this directly with no need for the `goodchunk` rules, with the individual split atoms being inclusions and exclusions in the example and the penalty being on the full example rather than each individual inclusion and exclusion. In the ILASP task, the example corresponding to the rule for `goodchunk(1)` would have the partial interpretation $\langle\{$`split(1)`, `split(4)`$\}, \{$`split(2)`, `split(3)`$\}\rangle$.

In [KSS17], 11-fold cross validation was performed on five different datasets, with 100 and 500 examples. As Inspire has a parameter which determines how aggressive the pruning should be (discussed in Section 10.7), [KSS17] presents many F_1 scores. Each entry for Inspire in Table 11.2 is for the pruning parameter for which Inspire yielded the best F_1 score.

		Inspire F_1 score	ILASP F_1 score	ILASP computation time (s)
100 examples	Headlines S1	73.1	74.2	351.2
	Headlines S2	70.7	73.0	388.3
	Images S1	81.8	83.0	144.9
	Images S2	73.9	75.2	187.2
	Students S1/S2	67.0	72.5	264.5
500 examples	Headlines S1	69.7	75.3	1616.6
	Headlines S2	73.4	77.2	1563.6
	Images S1	75.3	80.8	929.8
	Images S2	71.3	78.9	935.8
	Students S1/S2	66.3	75.6	1451.3

Table 11.2: The F_1 scores for Inspire and ILASP3$^{\mathcal{I},\mathcal{MT}}$ on the various sentence chunking tasks. We also show the average computation time for ILASP3$^{\mathcal{I},\mathcal{MT}}$.

Inspire aims to approximate the optimal inductive solution of the task. The hypothesis can be suboptimal for three reasons: firstly, the abductive phase may find an abductive solution which leads to a suboptimal inductive solution; secondly, Inspire's pruning may remove some hypotheses from the hypothesis space; and finally, Inspire was set to interrupt the inductive phase after 1800 seconds, returning the most optimal hypothesis found so far – in contrast, ILASP3$^{\mathcal{I},\mathcal{MT}}$[7] terminated in less than 1800s on every task. ILASP3$^{\mathcal{I},\mathcal{MT}}$ achieved a higher average F_1 score than Inspire on every one of the ten tasks. This shows that computing the optimal inductive solution of a task can lead to a higher accuracy than approximating the optimal solution. Although there can certainly be a trade off between accuracy and computation time, ILASP3$^{\mathcal{I},\mathcal{MT}}$ terminates in less than Inspire's timeout of 1800s in every case.

Note that for 4 out of the 5 datasets, Inspire performs better with 100 examples than with 500 examples. A possible explanation for this is that with more examples, Inspire does not get as close to the optimal solution as it does with fewer examples, thus leading to a lower F_1 score on the test data. With 500 examples, ILASP3$^{\mathcal{I},\mathcal{MT}}$ does take longer to terminate than it does for 100 examples, but in 4 out of the 5 cases, ILASP's average F_1 score is higher. This is the expected result: more data should tend to lead to a better hypothesis.

11.2.3 Car Preference Learning

We tested ILASP3's ability to learn real user preferences with the *car preference dataset* from [ASB+13]. This dataset consists of responses from 60 different users, who were each asked to give their preferences about 10 cars. They were asked to order each (distinct) pair of cars, leading to 45 orderings. The cars had 4 attributes, shown in Table 11.3.

Attribute	Values
Body type	`sedan`(1), `suv`(2)
Transmission	`manual`(1), `automatic`(2)
Engine Capacity	2.5L, 3.5L, 4.5L, 5.5L, 6.2L
Fuel Consumed	`hybrid`(1), `non_hybrid`(2)

Table 11.3: The attributes of the car preference dataset, along with the possible range of values for each attribute. The integer next to each value is how that value is represented in the data.

Our initial experiment was based on an experiment in [QK17], where the ALEPH [Sri01] system was used to learn the preferences of each user in the dataset and compared with support vector machines (SVM) and decision trees (DT). 10-fold cross validation was performed for each of the 60 users on the 45 orderings. In each fold, 10% of the orderings were omitted from the training data and used to test the learned hypothesis. The flaw in this approach is that in many cases the omitted examples will be implied by the rest of the examples (i.e. if $a \prec b$ and $b \prec c$ are given as examples it does not make much sense to omit $a \prec c$). For this reason, we also experimented with leaving out all the examples for a single car in each fold (i.e. every pair that contains that car), and using these examples

[7]In this experiment, ILASP used clingo 5 to solve meta-level ASP programs, with unsatisfiable core optimisation enabled.

to test (again leading to 10 folds). Essentially this new task corresponds to learning preferences from a complete ordering of 9 cars, and then testing the preferences on a new unseen car.

SVM[QK17] A	DT[QK17] A	ALEPH[QK17] A	ILASP3$^{\mathcal{SWC}}$ A	ILASP3$^{\mathcal{SWC}}$ B
0.832	0.747	0.729	0.880	0.863

Table 11.4: The accuracy results of ILASP3$^{\mathcal{SWC}}$ compared with the method in [QK17] on the car preference dataset.

Table 11.4 shows the accuracy of the approach in [QK17] and ILASP3$^{\mathcal{SWC}}$'s accuracy on the two versions of the experiment. The easier task (with a random 10% of the orderings omitted) is denoted as experiment A in the table, and the harder task is denoted as experiment B. In fact, even on the harder version of the task with all examples for a particular car omitted, ILASP3$^{\mathcal{SWC}}$ performs better than the approaches in [QK17] perform on the easier version of the task.

The following set of weak constraints are an example of the preferences learned by ILASP3$^{\mathcal{SWC}}$ in one of the folds for user 1 (in experiment A):

```
:~ fuel(2).[1@4, 2]
:~ body(1), transmission(2).[-1@3, 4]
:~ engine_cap(V0).[V0@2, 3, V0]
:~ body(1).[-1@1, 1]
```

Note that the first term in each weak constraint (after the priority level) is a unique identifier inserted by ILASP3 to guarantee that all weak constraints in the hypothesis are independent. This set of weak constraints corresponds to the following set of prioritised preferences (ordered from most to least important):

- The user prefers hybrid cars to non-hybrid cars.
- The user likes automatic sedans.
- The user would like to minimise the engine capacity of the car.
- The user prefers sedans to SUVs.

The noise in this experiment comes from the fact that some of the answers given by participants in the survey may contradict other answers. Some participants gave inconsistent orderings (breaking transitivity) meaning that there is no set of weak constraints that cover all of the ordering examples.

11.2.4 SUSHI Preference Learning

Another dataset for preference learning is the SUSHI dataset [KKA10]. The dataset is comprised of peoples' preference orderings over different types of sushi. Each type of sushi has several attributes, which are described in Table 11.5.

Attribute	Values
Style	`maki(0)`, `non_maki(1)`
Major group	`seafood(0)`, `non_seafood(1)`
Minor group	`0, ..., 11`
Oiliness	$[0, 4]$
Frequency Eaten	$[0, 3]$
Normalised Price	$[0, 5]$
Frequency Sold	$[0, 1]$

Table 11.5: The attributes of the SUSHI preference dataset, along with the possible range of values for each attribute.

There is a mix of categorical and continuous attributes. In the language bias for our experiments (see Appendix A), the categorical attributes are used as constants, whereas the continuous attributes are variables that can be used as the weight of the weak constraint. This allows weak constraints to express that the continuous attributes should be minimised or maximised. An example of a hypothesis learned by ILASP is as follows:

```
:~ minor_group(8).[1@5, 2]
:~ minor_group(7).[1@4, 1]
:~ minor_group(3).[1@3, 5]
:~ value(oil,V0), minor_group(1).[V0@2, 3, V0]
:~ value(oil,V0).[-V0@1, 4, V0]
```

This set of weak constraints corresponds to the following set of prioritised preferences (ordered from most to least importance):

- This person does not like sushi which is of minor group 8 (this corresponds to “other seafood”).
- This person does not like sushi which is of minor group 7 (this corresponds to “roe”).
- This person does not like sushi which is of minor group 3 (this corresponds to “tare”).
- This person prefers sushi which is of minor group 1 (this corresponds to “akami”) when it is less oily.
- This person likes all other types of sushi to be as oily as possible.

The dataset was constructed from a survey in which people were asked to order 10 different types of sushi. This ordering leads to 45 ordering examples per person. Our experiment is based on a similar experiment in [QK17]. For each of the first 60 people in the dataset we perform 10-fold cross validation, omitting 10% of the orderings in each fold. This experiment suffers from the same flaw as Experiment A on the car dataset in that some of the omitted examples may be implied by the training examples, but we give the results for a comparison to [QK17]. As shown in Table 11.6, ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SWC}}$ achieved an average accuracy of 0.84, which compares favourably to each result from [QK17].

SVM[QK17]	DT[QK17]	Aleph[QK17]	ILASP3
0.76	0.81	0.78	0.84

Table 11.6: The average accuracy results of ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SWC}}$ compared with the methods used in [QK17] on the sushi preference dataset.

Although in this experiment each participant gave a consistent total ordering of the 10 types of sushi, it might be the case that there is no hypothesis in the hypothesis space that covers all of the examples. This could be the case when we are not modelling a feature of the sushi that the participant considers to be important. For this reason, we treated this as a noisy learning setting, and used ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SWC}}$ to maximise the coverage of the examples.

11.2.5 Comparison to δILP

Although the work in this thesis concerns learning ASP programs from noisy examples, work has been done in the area of extending definite clause learning to handle noisy examples (for example, [SJ93, Sri01, OB10]).

A recent paper [EG18] claims that ILP systems can not handle noisy data. In order to learn from noisy data, [EG18] introduces the δILP algorithm, based on artificial neural networks. It demonstrates that δILP is able to achieve a high accuracy even with a large proportion of noise in the examples. In [EG18], δILP is evaluated on six synthetic datasets, and the noise is varied from 0% to 90%. In these experiments, we investigated the performance of ILASP3 on five of these six datasets[8]. In the original experiments, examples were atoms, and noise corresponded to swapping positive and negative examples. In each of our tasks, we ensured that the hypothesis space was such that for each $H \subseteq S_M$, $B \cup H \cup e_{ctx}$ was stratified for each example e. This allowed atomic examples to be represented as (positive) partial interpretations – a positive example $\mathtt{e}$ was represented as a partial interpretation $\langle \{\mathtt{e}\}, \emptyset \rangle$, and a negative example $\mathtt{e}$ was represented as a partial interpretation $\langle \emptyset, \{\mathtt{e}\} \rangle$. Note that each of the atomic examples must be represented as a separate partial interpretation example, as the penalties of examples must be separate.

The language bias of δILP is more prescriptive than ILASP's in the overall structure of the hypothesis; for example, it forces a maximum of two rules per learned predicate, and these two rules have to

[8]The authors of [EG18] provided us with the training and test data for these five problems.

conform to given rule templates, which state how many existential variables are allowed in the rules, and whether the rule is allowed to use learned predicates in the body. δILP also puts an upper bound of two literals in the body of any rule in the hypothesis. Technically, this does not restrict the expressivity of what can be learned, providing the language bias allows sufficient predicate invention, since any definite rule can be translated into a set of definite rules with at most two body literals by using extra predicates. ILASP on the other hand can be restrictive on the structure of individual rules in the hypothesis space (specifying the number of times a predicate can appear in the body of a single rule, or the type of certain variables). Due to the differences in language biases used by ILASP and δILP, the hypothesis spaces of the two systems are not equivalent. The ILASP language biases for each of the five tasks are in Appendix A.

Due to the imbalance of positive and negative examples in many of the tasks, we weight the positive examples at $w \times |E^-|/(|E^+|+|E^-|)$ and the negative examples at $w \times |E^+|/(|E^+|+|E^-|)$, where in this experiment w is 100. This essentially specifies the scoring function $\mathcal{S}(H, T)$, which is used by ILASP3 to choose between potential hypotheses. The weight for each example class (positive or negative) is equal to w multiplied by the proportion of the whole set of examples which are in the other class. This "corrects" any imbalance between positive and negative examples (i.e. the penalty for not covering a certain proportion of the positive examples is the same as the penalty for not covering the same proportion of negative examples). The constant w can be thought of as the difference in importance between the hypothesis length and the number of examples covered. In this these experiments we chose 100, as it is a high enough number to ensure that coverage is considered far more important than hypothesis length.

Figure 11.4 shows the mean squared error of the two systems. The results for δILP are taken from [EG18]. The results in Figure 11.4 show that in most tasks ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG}}$ achieves similar results to δILP in the range of 0% to 40%. However, at the other end of the scale (with more than 50% noise), there are some tasks where ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG}}$ finds hypotheses with close to 100% error, where δILP's error is much lower (less than 20% in the "member" problem). We would argue that when the noisy examples outnumber the correctly labeled examples, the learner should start learning the negation of the target hypothesis; for instance, in the case of "less than", ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG}}$ correctly learned the "greater than or equal to" relation. This corresponds to the following definition of `p`:

```
p(V0, V0) :- succ(V0, V1). % succ/2 is the successor relation.
p(V1, V1) :- succ(V0, V1).
p(V1, V2) :- succ(V0, V1), p(V0, V2).
```

We would argue that the ideal outcome of these kinds of experiments, where the proportion of noise is varied, is that the learner achieves close to 0% error until around 50% noise and close to 100% error thereafter. This is roughly what seems to happen for ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG}}$ in the "predecessor", "less than", "member" and "undirected edge" experiments. In "predecessor", the graph is less symmetric, with the "crossover" from low to high error occurring later. This is likely because the hypothesis for "not

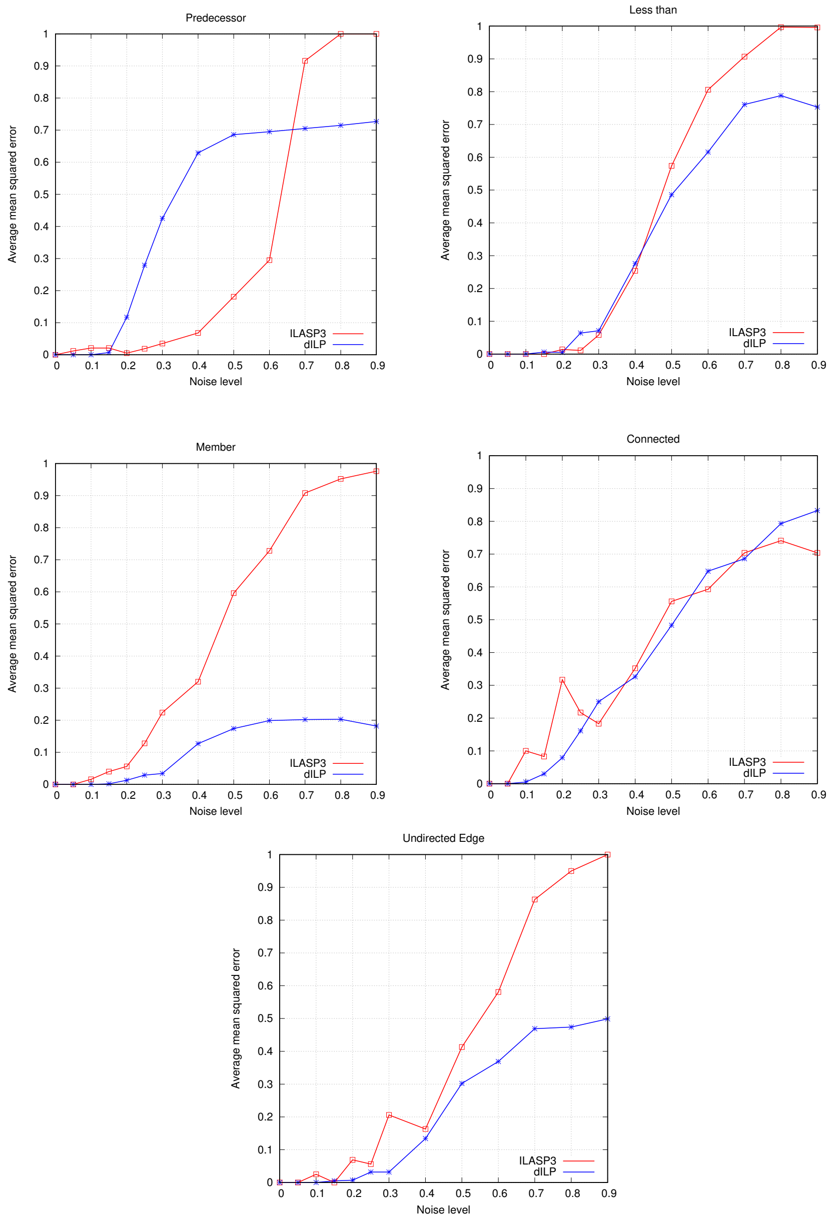


Figure 11.4: The average mean squared error of δILP and ILASP3$^{\mathcal{I},\mathcal{P},\mathcal{SG}}$ on five synthetic datasets from [EG18].

predecessor" is longer than the hypothesis for "predecessor". The failure of δILP to get close to 100% error in many of the tasks (for example in the "member" task, δILP has an error of less than 20% with the noise level at 90%) indicates that the negation of the target hypothesis is not representable given the language bias used by δILP in these experiments. In some cases (such as "member"), this is most likely because the negation of the concept requires negation as failure (which is not supported by δILP), but in others such as "less than", the negation of the target concept is expressible without negation as failure.

Summary

In this chapter we have evaluated ILASP3 on various datasets with noisy examples. We used several synthetic datasets to show that ILASP3 can learn even in the presence of high proportions of data that is inconsistent with the target hypothesis. We have also tested ILASP3's performance on several datasets used by other ILP systems aimed at learning under the answer set semantics, one ILP system aimed at preference learning, and one system for definite clause learning.

Chapter 12

Conclusion

The goal we set at the beginning of this work was to design a framework and learning algorithm that enables the learning of general ASP programs. We have shown that pre-existing frameworks and systems for learning under the answer set semantics are not general enough to learn ASP constructs such as choice rules or hard and weak constraints.

The contributions in this thesis fall into two categories. Firstly, we have extended the theoretical work on ILP under the answer set semantics, both with new learning frameworks, and with a new notion of the generality of a learning framework. Secondly, we have designed and implemented several new algorithms for learning ASP programs and evaluated them on a collection of synthetic and real-world datasets to demonstrate their accuracy and computational performance. We have shown that our algorithms are able to learn highly accurate hypotheses, from both non-noisy and noisy data, and that our ILASP2i and ILASP3 algorithms are highly scalable with respect to the number of examples on non-noisy and noisy learning tasks, respectively.

Theoretical Contributions

We have introduced a novel set of frameworks for learning ASP programs. They are the first frameworks that are capable of learning any ASP program consisting of normal rules, choice rules and hard constraints, up to strong equivalence. We are also able to learn weak constraints, and have shown that this allows us to bring the advantages of ILP to the field of preference learning.

We have introduced three novel measures of generality, based on the notion of distinguishability. For each of the three measures $\mathcal{D}$, we have shown that the following orderings of generality hold:

- $\mathcal{D}(ILP_b) \subseteq \mathcal{D}(ILP_{sm}) \subset \mathcal{D}(ILP_{LAS}) \subset \mathcal{D}(ILP_{LOAS}) \subset \mathcal{D}(ILP_{LOAS}^{context})$
- $\mathcal{D}(ILP_c) \subset \mathcal{D}(ILP_{LAS})$

This shows that $ILP_{LOAS}^{context}$ is the most general of the six frameworks, with respect to each of the three measures. Although each of our new frameworks (ILP_{LAS}, ILP_{LOAS} and $ILP_{LOAS}^{context}$) is more general than the previous frameworks (ILP_b, ILP_{sm} and ILP_c), we have also shown that this does not come with any additional price to pay in terms of computational complexity when compared with cautious induction (which, among the previous frameworks, is the one with highest complexity). For each of the three decision problems considered in Section 4.5, we have shown that the decision problem for $ILP_{LOAS}^{context}$ is in the same complexity class as the decision problem for ILP_c.

Practical Contributions

We have developed four new algorithms for learning ASP. The first new algorithm, ILASP1, showed the feasibility of our approach to learning ASP programs, although it had several scalability issues. These were addressed by ILASP2 and ILASP2i, which perform significantly better on tasks with negative examples and large numbers of examples, respectively. The first three algorithms were targeted at learning from non-noisy examples. In practice, however, examples can be noisy, and although ILASP2 and ILASP2i can be extended to solve noisy learning tasks, they are not well suited to do so. We therefore introduced the ILASP3 algorithm, which is specifically targeted at learning from noisy data and takes a very different approach, based on a notion of approximate coverage expressed through hypothesis schemas. We have shown that all of these algorithms are sound and complete with respect to the optimal inductive solutions of an $ILP_{LOAS}^{context}$ task (or ILP_{LOAS}^{noise} task in the case of ILASP3), and are guaranteed to terminate provided the task is well-defined.

Through our evaluation, we have shown that the ILASP2i algorithm (on non-noisy tasks) and the ILASP3 algorithm (for noisy tasks) scale well with respect to the number of examples, both on synthetic and real datasets. As there are no other systems which can solve $ILP_{LOAS}^{context}$ tasks, a direct comparison with related systems is difficult. For example, many of the other systems can only solve brave induction tasks. However, we have tested ILASP3 on five datasets that have been used for evaluating other ILP systems. None of these datasets require the full expressive power of $ILP_{LOAS}^{context}$, but nonetheless in most cases ILASP3 is still able to perform favourably compared to the other systems.

12.1 Completeness

In addition to the theoretical differences in the learning framework used by the ILASP algorithms and the simpler frameworks used by other ILP algorithms, there is a major difference in ILASP's approach compared to other algorithms. Each of the ILASP algorithms presented in this thesis is sound and complete with respect to the optimal inductive solutions of an $ILP_{LOAS}^{context}$ task. This means that they are guaranteed to return an optimal inductive solution (if the task is satisfiable). In practical settings, ILASP's termination depends on the resources available (these are computation time and memory).

Many ILP algorithms (e.g. Progol 5, Aleph, XHAIL, Inspire) are not complete, and are only sound in the sense that they return an inductive solution (with no guarantee on the optimality of this solution). Some of these algorithms do have a notion of hypothesis "compression", but they are not guaranteed to find the most compressed hypothesis. XHAIL, for instance, does compute the "locally" optimal inductive solution with respect to its Kernel set, but this may not be the "globally" optimal solution. Similarly, Inspire computes the optimal solution of the pruned Kernel. For this reason, we say that these algorithms return *approximations* of the optimal solution.

Computing the optimal inductive solution of a task is certainly a more computationally expensive task than finding an arbitrary inductive solution, so the obvious question is whether striving for optimality is really worth it. We would argue that it is, as the results in Chapter 11 have proved that ILASP can achieve a higher accuracy than the current approximate systems. The fact that ILASP's accuracy results are often better than those achieved by approximate systems would seem to indicate that there is an advantage to finding the optimal inductive solution, in that this solution may be of higher quality than sub-optimal inductive solutions.

In some larger scale applications, there is likely to be a trade-off between accuracy and scalability of the approach; however, a major problem with the current approximate systems is that they have no way of knowing how far their approximation is from the true optimal solution of the task. It would be possible to extend the ILASP3 approach to stop after finding the first hypothesis whose score was known to be within some threshold of the optimal score. In each iteration the lower bound score of the current hypothesis $\mathcal{S}_{lb}$ is less than or equal to the score of the optimal inductive solution. So if a hypothesis is found whose true score minus the lower bound score is below the threshold, it could be returned. However, this is outside of the scope of this thesis, which has presented algorithms for finding optimal solutions.

12.1.1 Predicate Invention

One of the challenging aspects of ILP is *predicate invention*, where a predicate that does not occur in the background knowledge or examples must be learned. The ILASP algorithms support *prescriptive* predicate invention, where the structure (predicate name and arity) of any invented predicates are specified in the language bias of the learning task.

One way of supporting *automatic* predicate invention, where invented predicates are not specified in the mode bias is to use *iterative deepening* over the number of invented predicates that are permitted. In each iteration, further invented predicates are added to the mode bias, until the task becomes satisfiable. Essentially, each iteration involves calling a prescriptive predicate invention algorithm. In principle, this prescriptive predicate invention algorithm could be any one of the ILASP algorithms, thus enabling automatic predicate invention in ILASP.

There are two main reasons why we have chosen not to implement this as part of the ILASP systems. Firstly, it can be achieved trivially, with a wrapper outside of the ILASP systems. More fundamentally, the approach is not guaranteed to find an optimal solution to the task, as the optimal inductive solution for the first satisfiable task is always returned (there may be shorter solutions of tasks that allow further invented predicates). In non-noisy settings, the iterative deepening approach will eventually converge on a correct hypothesis if it is given sufficient examples – for any incorrect hypothesis there is a counter example, meaning that there is a set of examples that rules out any hypothesis that uses fewer invented predicates than the target hypothesis; i.e. if 5 invented predicates are required, it should be possible to construct examples such that no hypothesis with fewer than 5 invented predicates satisfies every example.

The natural extension of the iterative deepening approach in a noisy setting is to continue adding invented predicates until there is a solution that covers a certain proportion of the examples. The issue with this is that there may be exceptional kinds of examples that only occur in rare settings, but require extra predicate invention. If the stopping criterion of the system is that a hypothesis is found that covers 95% of the examples, but the exceptional examples only occur 1% of the time, no matter how many examples are given the correct hypothesis is unlikely to be learned (even though it is within the full hypothesis space of the system).

In such noisy tasks, the trade-off between using an iterative deepening approach and using ILASP directly is similar to the trade-off between complete and approximate systems discussed in the previous section. If the correct solution is within the hypothesis space, and sufficient examples are given, each of the ILASP algorithms will return a solution that is equivalent to the correct solution (computational resources permitting). On the other hand, iterative deepening approaches may be more scalable, but may also return sub-optimal solutions, no matter how many examples are given.

12.2 Future Work

In this thesis, we have made significant progress towards a scalable approach to learning ASP programs. We have shown that our ILASP algorithms are scalable with respect to the number of examples, even for noisy tasks. However, none of the current ILASP algorithms scales well with the size of the hypothesis space. The main reason for this scalability issue is that each of the ILASP algorithms begins by computing the hypothesis space in full. The next step of future work, will be to design a new version of the ILASP algorithm that only computes the parts of the hypothesis space that are necessary to solve the task.

Other future work may include the expansion of the subset of ASP that is considered by ILASP. Including constructs such as (stratified) summing aggregates and conditional literals would not in principle pose any greater challenge for the algorithms, beyond extending the meta-level programs used by the ILASP algorithms (and dealing with the potential increase in the size of the hypothesis spaces). However, they do introduce some interesting theoretical questions; for instance, what is the "length" of a rule containing a summing aggregate or a conditional literal? And, how can we generalise mode declarations to allow the expression of these wider hypothesis spaces?

In our current ILASP algorithms, we have already slightly diverged from a traditional view of hypothesis "length" (i.e. counting the literals), in our calculation of the length of hypotheses containing choice rules. Before the computation of the length of a choice rule, the aggregate in the head of the rule is first converted into disjunctive normal form. The intuition behind this is that a choice rule that has a larger number of disjuncts leads to more possibilities, and is therefore less compressed than a choice rule with fewer possibilities. This is linked to the number of potential answer sets of the choice rule. It may be that other measures of a program, such as the number of even and odd loops in the program (which is also linked to the number of answer sets of the program) give a better measure of hypothesis compression than simply counting the number of literals. The motivation of exploring more advanced compression metrics in future work is twofold: firstly, these metrics may allow for the definition of hypothesis length for other constructs, such as conditional literals and summing aggregates; and secondly, they may lead to better quality hypotheses.

Bibliography

[ACBR13] Duangtida Athakravi, Domenico Corapi, Krysia Broda, and Alessandra Russo. Learning through hypothesis refinement using answer set programming. In *International Conference on Inductive Logic Programming*, pages 31–46. Springer, 2013.

[ADF$^+$13] Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. WASP: A native ASP solver based on constraint learning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 54–66. Springer, 2013.

[AGALG$^+$16] Eneko Agirre, Aitor Gonzalez Agirre, Inigo Lopez-Gazpio, Montserrat Maritxalar, German Rigau Claramunt, and Larraitz Uria. Semeval-2016 task 2: Interpretable semantic textual similarity. In *SemEval-2016 Task 2: Interpretable semantic textual similarity. SemEval-2016. 10th International Workshop on Semantic Evaluation; 2016 Jun 16-17; San Diego, CA. Stroudsburg (PA): ACL; 2016. p. 512-24.* ACL (Association for Computational Linguistics), 2016.

[AKM07] Marta Arias, Roni Khardon, and Jérôme Maloberti. Learning horn expressions with logan-h. *Journal of Machine Learning Research*, 8(Mar):549–587, 2007.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[ASB$^+$13] Ehsan Abbasnejad, Scott Sanner, Edwin V Bonilla, Pascal Poupart, et al. Learning community-based preferences via dirichlet process mixtures of gaussian processes. In *IJCAI*, pages 1213–1219, 2013.

[Ath15] Duangtida Athakravi. *Inductive logic programming using bounded hypothesis space.* PhD thesis, Imperial College London, 2015.

[BBD$^+$04] Craig Boutilier, Ronen I Brafman, Carmel Domshlak, Holger H Hoos, and David Poole. Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res.(JAIR)*, 21:135–191, 2004.

[BDR98] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial intelligence*, 101(1):285–297, 1998.

[BDRS15] Gerhard Brewka, James P Delgrande, Javier Romero, and Torsten Schaub. asprin: Customizing answer set preferences without a headache. In *AAAI*, pages 1467–1474, 2015.

[BNT03] Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski. Answer set optimization. In *IJCAI*, volume 3, pages 867–872, 2003.

[BR15a] Elena Bellodi and Fabrizio Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming*, 15(02):169–212, 2015.

[BR15b] Stefano Bragaglia and Oliver Ray. Nonmonotonic learning in large biological networks. In *Inductive Logic Programming*, pages 33–48. Springer, 2015.

[Bra99] Ivan Bratko. Refining complete hypotheses in ILP. In *International Conference on Inductive Logic Programming*, pages 44–55. Springer, 1999.

[CFG$^+$13] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. Asp-core-2 input language format, 2013.

[Cla77] Keith L Clark. Negation as failure. *Logic and data bases*, 1:293–322, 1977.

[Cor12] Domenico Corapi. *Nonmonotonic inductive logic programming as abductive search*. PhD thesis, Imperial College London, 2012.

[CR11] Domenico Corapi and Alessandra Russo. ASPAL. proof of soundness and completeness. Technical report, Technical Report DTR11-5, Department of Computing, Imperial College, London, 2011.

[CRL10] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming as abductive search. In *ICLP (Technical Communications)*, pages 54–63, 2010.

[CRL12] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In *Inductive Logic Programming*, pages 91–97. Springer, 2012.

[DJJT01] Mehdi Dastani, Nico Jacobs, Catholijn M Jonker, and Jan Treur. Modeling user preferences and mediating agents in electronic commerce. In *Agent Mediated Electronic Commerce*, pages 163–193. Springer, 2001.

[DR97] Luc De Raedt. Logical settings for concept-learning. *Artificial Intelligence*, 95(1):187–201, 1997.

[DRB$^+$16] Stanislav Dragiev, Alessandra Russo, Krysia Broda, Mark Law, and Rares Turliuc. An abductive-inductive algorithm for probabilistic inductive logic programming. In *26th International Conference on Inductive Logic Programming (Short papers)*, 2016.

[DRD97a] Luc De Raedt and Luc Dehaspe. Clausal discovery. *Machine Learning*, 26(2):99–146, 1997.

[DRD97b] Luc De Raedt and Luc Dehaspe. Learning from satisfiability. In *Ninth Dutch Conference on Artificial Intelligence (NAIC'97)*, pages 303–312, 1997.

[DRK04] Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In *International Conference on Algorithmic Learning Theory*, pages 19–36. Springer, 2004.

[DRKT07] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467, 2007.

[DRT10] Luc De Raedt and Ingo Thon. Probabilistic rule learning. In *International Conference on Inductive Logic Programming*, pages 47–58. Springer, 2010.

[DRVL95] Luc De Raedt and Wim Van Laer. Inductive constraint logic. In *Algorithmic Learning Theory*, pages 80–94. Springer, 1995.

[EG95] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[EG18] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[EGL16] Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Magazine*, 37(3):53–68, 2016.

[EIK09] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web. Semantic Technologies for Information Systems*, pages 40–110. Springer, 2009.

[FH03] Johannes Fürnkranz and Eyke Hüllermeier. Pairwise preference learning and ranking. In *Machine Learning: ECML 2003*, pages 145–156. Springer, 2003.

[FH10] Johannes Fürnkranz and Eyke Hüllermeier. Preference learning: An introduction. In *Preference learning*, pages 1–17. Springer, 2010.

[Fit02] Melvin Fitting. Fixpoint semantics for logic programming a survey. *Theoretical computer science*, 278(1-2):25–51, 2002.

[FPL11] Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011.

[GAG13] Joshua T Guerin, Thomas E Allen, and Judy Goldsmith. Learning cp-net preferences online from user queries. In *International Conference on Algorithmic DecisionTheory*, pages 208–220. Springer, 2013.

[GHH01] Ben Geisler, Vu Ha, and Peter Haddawy. Modeling user preferences via theory refinement. In *Proceedings of the 6th international conference on Intelligent user interfaces*, pages 87–90. ACM, 2001.

[GK14] Michael Gelfond and Yulia Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach.* Cambridge University Press, 2014.

[GKK$^+$10] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. A user's guide to gringo, clasp, clingo, and iclingo. 2010.

[GKK$^+$11] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.

[GKNS07] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-driven answer set solving. In *IJCAI*, volume 7, pages 386–392, 2007.

[GKS11] Martin Gebser, Roland Kaminski, and Torsten Schaub. Complex optimization in answer set programming. *Theory and Practice of Logic Programming*, 11(4-5):821–839, 2011.

[GKS13] M. Gebser, B. Kaufmann, and T. Schaub. Advanced conflict-driven disjunctive answer set solving. In F. Rossi, editor, *Proceedings of the Twenty-theird International Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 912–918. IJCAI/AAAI, 2013.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.

[Hor12] Tomáš Horváth. A model of user preference learning for content-based recommender systems. *Computing and informatics*, 28(4):453–481, 2012.

[IK97] Katsumi Inoue and Yoshimitsu Kudoh. Learning extended logic programs. In *IJCAI (1)*, pages 176–181, 1997.

[IRS14] Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. Learning from interpretation transition. *Machine Learning*, 94(1):51–79, 2014.

[Joa02] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[KAP15] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. Incremental learning of event definitions with inductive logic programming. *Machine Learning*, 100(2-3):555–585, 2015.

[KAP16] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. Online learning of event definitions. *Theory and Practice of Logic Programming*, 16(5-6):817–833, 2016.

[KHM05] Hideto Kazawa, Tsutomu Hirao, and Eisaku Maeda. Order svm: a kernel method for order learning based on generalized order statistics. *Systems and Computers in Japan*, 36(1):35–43, 2005.

[KKA10] Toshihiro Kamishima, Hideto Kazawa, and Shotaro Akaho. A survey and empirical comparison of object ranking methods. In *Preference learning*, pages 181–201. Springer, 2010.

[Kor12] Frédéric Koriche. Relational networks of conditional preferences. *Machine learning*, 89(3):233–255, 2012.

[KS86] R Kowalski and M Sergot. A logic-based calculus of events. *New generation computing*, 4(1):67–95, 1986.

[KSS17] Mishal Kazmi, Peter Schüller, and Yücel Saygın. Improving scalability of inductive logic programming via pruning and best-effort optimisation. *Expert Systems with Applications*, 87:291–303, 2017.

[Lif08] Vladimir Lifschitz. Twelve definitions of a stable model. In *ICLP*, volume 5366, pages 37–51. Springer, 2008.

[LMS03] Inês Lynce and Joao Marques-Silva. The effect of nogood recording in dpll-cbj sat algorithms. In *Recent Advances in Constraints*, pages 144–158. Springer, 2003.

[LPF$^+$06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7(3):499–562, 2006.

[LRB14] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, pages 311–325, 2014.

[LRB15a] Mark Law, Alessandra Russo, and Krysia Broda. Learning weak constraints in answer set programming. *Theory and Practice of Logic Programming*, 15(4-5):511–525, 2015.

[LRB15b] Mark Law, Alessandra Russo, and Krysia Broda. Simplified reduct for choice rules in ASP. Technical report, Tech. Rep. DTR2015-2, Imperial College of Science, Technology and Medicine, Department of Computing, 2015.

[LRB16] Mark Law, Alessandra Russo, and Krysia Broda. Iterative learning of answer set programs from context dependent examples. *Theory and Practice of Logic Programming*, 16(5-6):834–848, 2016.

[LRB18a] Mark Law, Alessandra Russo, and Krysia Broda. The complexity and generality of learning answer set programs. *Artificial Intelligence*, 259:110–146, 2018.

[LRB18b] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs from noisy examples. *Advances in Cognitive Systems*, 2018.

[LRS97] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and computation*, 135(2):69–112, 1997.

[LT94] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *ICLP*, volume 94, pages 23–37, 1994.

[LXW$^+$14] Juntao Liu, Yi Xiong, Caihua Wu, Zhijun Yao, and Wenyu Liu. Learning conditional preference networks from inconsistent examples. *IEEE transactions on knowledge and data engineering*, 26(2):376–390, 2014.

[MDRP$^+$12] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. ILP turns 20. *Machine Learning*, 86(1):3–23, 2012.

[ML13] Stephen Muggleton and Dianhuan Lin. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1551–1557. AAAI Press, 2013.

[MLPTN14] Stephen H Muggleton, Dianhuan Lin, Niels Pahlavi, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94(1):25–49, 2014.

[MSTN08] Stephen H Muggleton, José Carlos Almeida Santos, and Alireza Tamaddoni-Nezhad. Toplog: ILP using a logic program declarative bias. In *International Conference on Logic Programming*, pages 687–692. Springer, 2008.

[MT99] Victor W Marek and Miroslaw Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*, pages 375–398. Springer, 1999.

[Mue14] Erik T Mueller. *Commonsense reasoning: An event calculus based approach.* Morgan Kaufmann, 2014.

[Mug91] Stephen Muggleton. Inductive logic programming. *New generation computing*, 8(4):295–318, 1991.

[Mug95] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 13(3-4):245–286, 1995.

[MV96] Lionel Martin and Christel Vrain. A three-valued framework for the induction of general logic programs. *Advances in Inductive Logic Programming*, pages 219–235, 1996.

[NBG$^+$01] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An A-Prolog decision support system for the space shuttle. In *Practical Aspects of Declarative Languages*, pages 169–183. Springer, 2001.

[Nic16] Matthias Nickles. PrASP report. *arXiv preprint arXiv:1612.09591*, 2016.

[NM14] Matthias Nickles and Alessandra Mileo. Probabilistic inductive logic programming based on answer set programming. *arXiv preprint arXiv:1405.0720*, 2014.

[NM15] Matthias Nickles and Alessandra Mileo. A system for probabilistic inductive answer set programming. In *International Conference on Scalable Uncertainty Management*, pages 99–105. Springer International Publishing, 2015.

[OB10] Andrej Oblak and Ivan Bratko. Learning from noisy data using a non-covering ILP algorithm. In *International Conference on Inductive Logic Programming*, pages 190–197. Springer, 2010.

[Ote01] Ramón P Otero. Induction of stable models. In *Inductive Logic Programming*, pages 193–205. Springer, 2001.

[Pap03] Christos H Papadimitriou. *Computational complexity.* John Wiley and Sons Ltd., 2003.

[PBL14] Andreas Poxrucker, Gernot Bahle, and Paul Lukowicz. Towards a real-world simulator for collaborative distributed learning in the scenario of urban mobility. In *Proceedings of the Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 44–48. IEEE Computer Society, 2014.

[QK17] Nunung Nurul Qomariyah and Dimitar Kazakov. Learning binary preference relations. In *4th Joint Workshop on Interfaces and Human Decision Making for Recommender Systems (IntRS) 2017*, page 30, 2017.

[Qui90] J Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 5(3):239–266, 1990.

[Ray05] Oliver Ray. *Hybrid abductive inductive learning.* PhD thesis, Imperial College London, 2005.

[Ray09] Oliver Ray. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329–340, 2009.

[RBR03] Oliver Ray, Krysia Broda, and Alessandra Russo. Hybrid abductive inductive learning: A generalisation of progol. In *Inductive Logic Programming*, pages 311–328. Springer, 2003.

[RBR04] Oliver Ray, Krysia Broda, and Alessandra Russo. A hybrid abductive inductive proof procedure. *Logic Journal of IGPL*, 12(5):371–397, 2004.

[RBZ14] Fabrizio Riguzzi, Elena Bellodi, and Riccardo Zese. A history of probabilistic inductive logic programming. *Frontiers in Robotics and AI*, 1:6, 2014.

[RBZ$^+$16] Fabrizio Riguzzi, Elena Bellodi, Riccardo Zese, Giuseppe Cota, and Evelina Lamma. Scaling structure learning of probabilistic logic programs by mapreduce. In *European Conference on Artificial Intelligence*, 2016.

[RDG$^+$10] Francesco Ricca, Antonella Dimasi, Giovanni Grasso, Salvatore Maria Ielpa, Salvatore Iiritano, Marco Manna, and Nicola Leone. A logic-based system for e-tourism. *Fundamenta Informaticae*, 105(1-2):35–55, 2010.

[RI07] Oliver Ray and Katsumi Inoue. Mode-directed inverse entailment for full clausal theories. In *International Conference on Inductive Logic Programming*, pages 225–238. Springer, 2007.

[RPMB08] Leonardo Rigutini, Tiziano Papini, Marco Maggini, and Monica Bianchini. A neural network approach for learning object ranking. In *International Conference on Artificial Neural Networks*, pages 899–908. Springer, 2008.

[RPMS11] Leonardo Rigutini, Tiziano Papini, Marco Maggini, and Franco Scarselli. Sortnet: Learning to rank by a neural preference function. *IEEE transactions on neural networks*, 22(9):1368–1380, 2011.

[Sak00] Chiaki Sakama. Inverse entailment in nonmonotonic logic programs. In *ILP*, pages 209–224. Springer, 2000.

[Sak01] Chiaki Sakama. Nonmonotomic inductive logic programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 62–80. Springer, 2001.

[Sak05] Chiaki Sakama. Induction from answer sets in nonmonotonic logic programs. *ACM Transactions on Computational Logic (TOCL)*, 6(2):203–231, 2005.

[SBP00] Jennifer Seitzer, James P Buckley, and Yi Pan. Inded: A distributed knowledge-based learning system. *IEEE Intelligent Systems and their Applications*, 15(5):38–46, 2000.

[SI09] Chiaki Sakama and Katsumi Inoue. Brave induction: a logical framework for learning from incomplete information. *Machine Learning*, 76(1):3–35, 2009.

[SJ93] E Sandewall and CG Jansson. Handling imperfect data in inductive logic programming. In *Scandinavian Conference on Artificial Intelligence-93: Proceedings of the Fourth Scandinavian Conference on Artificial Intelligence Electrum, Stockholm, Sweden, May 4-7, 1993*, volume 18, page 111. IOS Press, 1993.

[SN99] Timo Soininen and Ilkka Niemelä. Developing a declarative rule language for applications in product configuration. In *International Symposium on Practical Aspects of Declarative Languages*, pages 305–319. Springer, 1999.

[Sri01] Ashwin Srinivasan. The aleph manual. *Machine Learning at the Computing Laboratory, Oxford University*, 2001.

[Sta93] Irene Stahl. Predicate invention in ILP – an overview. In *European Conference on Machine Learning*, pages 311–322. Springer, 1993.

[Sto76] Larry J Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[TKSB00] Erik F Tjong Kim Sang and Sabine Buchholz. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 127–132. Association for Computational Linguistics, 2000.

[VEK76] Maarten H Van Emden and Robert A Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM (JACM)*, 23(4):733–742, 1976.

[Wro96] Stefan Wrobel. First order theory refinement. *Advances in inductive logic programming*, 32:14–33, 1996.

[Yd07] Fusun Yaman and Marie desJardins. More-or-less cp-networks. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 434–441. AUAI Press, 2007.

Appendix

Appendix A

Language Biases used in the evaluations of ILASP

The example learning tasks used throughout this thesis have mostly used small hypothesis spaces, presented as the full set of rules in the hypothesis space. In practice, however, hypothesis spaces are usually defined by a mode bias. In this chapter we formalise ILASP's particular form of mode bias and present hypothesis spaces used in the experiments in Chapter 8 and Chapter 11 of this thesis.

A.1 The ILASP Mode Bias

Definition A.1 formalises the notion of mode bias used by our ILASP systems.

Definition A.1. An *ILASP mode bias M* is a set of meta-level atoms, which take the following forms:

- *Mode declarations*, each of which is a meta-level atom of the form $\mathtt{class}(\mathtt{r}, \mathtt{a}, (\mathtt{opt_1}, \ldots, \mathtt{opt_n}))$, such that:
  - r is an integer called the *recall*. If r is omitted, this corresponds to an unrestricted (infinite) recall.
  - $\mathtt{class} \in \{\#\mathtt{modeh}, \#\mathtt{modeha}, \#\mathtt{modeb}, \#\mathtt{modeo}\}$.
  - `a` is a term (representing a reified atom) which may contain *placeholder* arguments of the form $\mathtt{var}(\mathtt{t})$ or $\mathtt{const}(\mathtt{t})$, where `t` is a constant called the *type* of the argument.
  - $\mathtt{opt_1}, \ldots, \mathtt{opt_n}$ is a list of constants called *options*. If $n = 0$, the list of options is omitted.
- *Constant declarations*, each of which is a meta-level atom of the form $\#\mathtt{constant}(\mathtt{t}, \mathtt{c})$ where `t` and `c` are both constants (`t` is called a type).

- *Weight declarations*, each of which is a meta-level atom of the form `#weight(c)` where `c` is a constant.
- *Parameters* of the form `#param(i)`, where `#param` $\in$ {`#minhl`, `#maxhl`, `#maxv`, `#maxp`, `#maxbl`, `#maxrl`}[1] and i is an integer. There is at most one atom in M per `#param` predicate.
- The *flag* {`#no_constraints`}.

An ILASP mode bias can be used to specify a full hypothesis space. Definition A.3 formalises the notion of the expansion of a set of mode declarations. We first formalise the notion of an atom being an *instance* of a mode declaration in Definition A.2. Each variable in a rule is assigned a constant called its *type*. These types correspond to the types in the variable placeholders. No variable is allowed to take more than one type.

Definition A.2. Given an ILASP mode bias M, let $\mathtt{m}$ be a mode declaration $\mathtt{c(r, a, (opt_1, \ldots, opt_n))} \in M$. An atom $\mathtt{a'}$ is an *instance* of $\mathtt{m}$ if each of the following conditions hold:

- $\mathtt{a'}$ can be constructed from `a` by replacing each placeholder `var(t)` is with a variable of type `t` and each placeholder `const(t)` with a constant `c` such that $\mathtt{\#constant(t, c)} \in M$.
- If `anti_reflexive` $\in \{\mathtt{opt_1}, \ldots, \mathtt{opt_n}\}$ and $\mathtt{a'}$ has arity 2, then its two arguments are non-equal.
- If `symmetric` $\in \{\mathtt{opt_1}, \ldots, \mathtt{opt_n}\}$ and $\mathtt{a'}$ has exactly two arguments $\mathtt{arg_1}$ and $\mathtt{arg_2}$, then $\mathtt{arg_1} \prec \mathtt{arg_2}$ (where $\prec$ is a lexicographical ordering of all possible terms).

If $\mathtt{a'}$ is an instance of `m` and `positive` $\notin \{\mathtt{opt_1}, \ldots, \mathtt{opt_n}\}$, then `not` $\mathtt{a'}$ is also an instance of `m`.

Note that unlike the mode declarations in other systems, there is no need for the types to correspond to predicates in the background knowledge.

Definition A.3. Given a mode bias M, we define the rules that *conform to M* as follows:

- A conjunction of naf-literals C is said to be valid over a class `c` $\in$ {`#modeb`, `#modeo`} given M iff each of the following conditions hold:
  1. Each literal in C is an instance of at least one mode declaration in M with class `c`.
  2. If there is a parameter atom `#maxv(i)` $\in M$, then the number of variables in C is at most `i`.
  3. If there is a parameter atom `#maxbl(i)` $\in M$, then the number of literals in C is at most `i`.

[1]In the ILASP implementation, `#maxbl` and `#maxrl` are passed to the system as the command line flags `-ml=[integer]` and `--max-rule-length=[integer]`, respectively.

4. Every variable in C occurs in at least one positive literal in C (i.e. C is *safe*).

- A hard constraint R is said to conform to M iff $\#\texttt{no\_constraints} \not\in M$ and $body(R)$ is a valid conjunction over `#modeb` given M.

- A normal rule R is said to conform to M iff each of the following conditions hold:

  1. $body(R)$ is a valid conjunction over `#modeb` given M,
  2. $head(R)$ is an instance of at least one mode declaration in M with class `#modeh`.
  3. each variable in $head(R)$ occurs in $body(R)$.
  4. If there is a parameter atom $\#\texttt{maxrl(i)} \in M$, then the number of literals in R is at most `i`.

- A choice rule R is said to conform to M iff each of the following conditions hold:

  1. $body(R)$ is a valid conjunction over `#modeb` given M.
  2. each atom in $heads(R)$ is an instance of at least one mode declaration in M with class `#modeha`.
  3. If there is a parameter atom $\#\texttt{minhl(i)} \in M$, then $|heads(R)| \geq \texttt{i}$.
  4. If there is a parameter atom $\#\texttt{maxhl(i)} \in M$, then $|heads(R)| \leq \texttt{i}$.
  5. If there is a parameter atom $\#\texttt{maxrl(i)} \in M$, then the number of literals in $R \leq \texttt{i}$.
  6. The lower bound `lb` and upper bound `ub` of $head(R)$ are such that $0 \leq \texttt{lb} \leq \texttt{ub} \leq |heads(R)|$.
  7. All variables in $head(R)$ occur in $body(R)$.

- A weak constraint W with the tail $[\texttt{wt@lev}, \texttt{t}_1, \ldots, \texttt{t}_\texttt{n}]$ is said to conform to M iff each of the following conditions hold:

  1. $body(W)$ is a valid conjunction over `#modeo` given M.
  2. `wt` is either an integer such that $\#\texttt{weight(wt)} \in M$ or a variable `V` that occurs in $body(W)$ (or the negation $-\texttt{V}$ of such a variable) such that $\texttt{weight(t)} \in M$, where `t` is the type of `V`.
  3. `lev` is a positive integer and if there is a parameter atom $\#\texttt{maxp(i)} \in M$, then $\texttt{lev} \leq \texttt{i}$.
  4. $\texttt{t}_1 = W_{id}$ (the unique identifier of W).
  5. $\texttt{t}_2, \ldots, \texttt{t}_\texttt{n}$ is the list of variables that occur in $body(W)$.

A hypothesis space S_M is called an *expansion* of S_M if each rule in S_M conforms to M, and every rule that conforms to M is strongly equivalent to at least one rule in S_M.

We use S_M to denote an arbitrary (finite) expansion of M[2].

The latest versions of the ILASP system also support *bias constraints*, which are used to cut rules out of the hypothesis space. Rather than fully specify the semantics of these bias constraints, we have commented on the effect of any bias constraints that we use in language biases throughout the rest of the section.

A.2 Language Biases Used in the Evaluation Chapters

A.2.1 Hamilton Graphs

The Hamilton problem was first run before bias constraints were implemented in ILASP. We used a mode bias to generate a hypothesis space and then cut out rules which were equivalent to other rules in the hypothesis space (or at least were guaranteed to give exactly the same results), or which were guaranteed not to appear in an optimal inductive solution of the task. In modern versions of ILASP, we could achieve similar results using bias constraints. The original language bias was:

```
#modeha(1, in(var(node1),var(node1))).
#modeb(1, edge(var(node1), var(node1)), (positive)).

#modeb(2, in(var(node), var(node)), (positive, anti_reflexive)).
#modeb(1, var(node) != var(node), (positive, symmetric, anti_reflexive)).

#modeh(1, reach(var(node))).
#modeb(1, reach(var(node))).
#modeb(1, node(var(node)), (positive)).
#modeb(1, in(const(node),var(node)), (positive)).
#constant(node, 1).

#maxhl(1).
```

The final set of rules (where $n \sim R$ denotes that R is in the hypothesis space, and $|R| = n$) were:

```
1 ~  :- edge(V0, V0).
1 ~  :- edge(V0, V1).
1 ~  :- in(1,V0).
```

[2]Note that there is guaranteed to be at least one finite expansion of any finite mode bias M for which M contains values for each of the parameters (the ILASP implementation assigns default values if these are unspecified). This is because there must be a finite set of (non-equivalent) valid conjunctions of any length given M. This fixes the set of possible bodies, and as the variables in the rest of a rule must all occur in the body of the rule, this means there is a finite set of non-equivalent rules that conform to M.

```
1 ~  :- in(V0,V1).
1 ~  :- reach(V0).
2 ~  :- edge(V0, V0), in(1,V1).
2 ~  :- edge(V0, V0), in(V1,V2).
2 ~  :- edge(V0, V0), reach(V1).
2 ~  :- edge(V0, V1), in(1,V2).
2 ~  :- edge(V0, V1), reach(V2).
2 ~  :- in(1,V0), reach(V0).
2 ~  :- in(1,V0), reach(V1).
2 ~  :- in(V0,V1), V0 != V1.
2 ~  :- in(V0,V1), in(1,V0).
2 ~  :- in(V0,V1), in(1,V1).
2 ~  :- in(V0,V1), in(1,V2).
2 ~  :- in(V0,V1), in(V0,V2).
2 ~  :- in(V0,V1), in(V1,V2).
2 ~  :- in(V0,V1), reach(V0).
2 ~  :- in(V0,V1), reach(V1).
2 ~  :- in(V0,V1), reach(V2).
2 ~  :- node(V0), not reach(V0).
2 ~ reach(V0) :- in(1,V0).
2 ~ reach(V0) :- in(V0,V1).
2 ~ reach(V1) :- in(V0,V1).
3 ~  :- edge(V0, V0), in(1,V1), reach(V1).
3 ~  :- edge(V0, V0), in(1,V1), reach(V2).
3 ~  :- edge(V0, V0), in(V1,V2), V1 != V2.
3 ~  :- edge(V0, V0), in(V1,V2), in(1,V1).
3 ~  :- edge(V0, V0), in(V1,V2), in(1,V2).
3 ~  :- edge(V0, V0), in(V1,V2), reach(V1).
3 ~  :- edge(V0, V0), in(V1,V2), reach(V2).
3 ~  :- edge(V0, V0), node(V1), not reach(V1).
3 ~  :- edge(V0, V1), in(1,V2), reach(V2).
3 ~  :- edge(V0, V1), node(V2), not reach(V2).
3 ~  :- in(1,V0), node(V0), not reach(V0).
3 ~  :- in(1,V0), node(V1), not reach(V1).
3 ~  :- in(V0,V1), V0 != V1, in(1,V0).
3 ~  :- in(V0,V1), V0 != V1, in(1,V1).
3 ~  :- in(V0,V1), V0 != V1, in(1,V2).
3 ~  :- in(V0,V1), V0 != V1, reach(V0).
3 ~  :- in(V0,V1), V0 != V1, reach(V1).
3 ~  :- in(V0,V1), V0 != V1, reach(V2).
3 ~  :- in(V0,V1), in(1,V0), reach(V0).
3 ~  :- in(V0,V1), in(1,V0), reach(V1).
3 ~  :- in(V0,V1), in(1,V0), reach(V2).
3 ~  :- in(V0,V1), in(1,V1), reach(V0).
```

```
3 ~  :- in(V0,V1), in(1,V1), reach(V1).
3 ~  :- in(V0,V1), in(1,V1), reach(V2).
3 ~  :- in(V0,V1), in(1,V2), reach(V0).
3 ~  :- in(V0,V1), in(1,V2), reach(V1).
3 ~  :- in(V0,V1), in(1,V2), reach(V2).
3 ~  :- in(V0,V1), in(V0,V2), V0 != V1.
3 ~  :- in(V0,V1), in(V0,V2), V0 != V2.
3 ~  :- in(V0,V1), in(V0,V2), V1 != V2.
3 ~  :- in(V0,V1), in(V0,V2), in(1,V0).
3 ~  :- in(V0,V1), in(V0,V2), in(1,V1).
3 ~  :- in(V0,V1), in(V0,V2), in(1,V2).
3 ~  :- in(V0,V1), in(V0,V2), reach(V0).
3 ~  :- in(V0,V1), in(V0,V2), reach(V1).
3 ~  :- in(V0,V1), in(V0,V2), reach(V2).
3 ~  :- in(V0,V1), in(V1,V2), V0 != V1.
3 ~  :- in(V0,V1), in(V1,V2), V0 != V2.
3 ~  :- in(V0,V1), in(V1,V2), V1 != V2.
3 ~  :- in(V0,V1), in(V1,V2), in(1,V0).
3 ~  :- in(V0,V1), in(V1,V2), in(1,V1).
3 ~  :- in(V0,V1), in(V1,V2), in(1,V2).
3 ~  :- in(V0,V1), in(V1,V2), reach(V0).
3 ~  :- in(V0,V1), in(V1,V2), reach(V1).
3 ~  :- in(V0,V1), in(V1,V2), reach(V2).
3 ~  :- in(V0,V1), node(V0), not reach(V0).
3 ~  :- in(V0,V1), node(V1), not reach(V1).
3 ~  :- in(V0,V1), node(V2), not reach(V2).
3 ~  :- reach(V0), node(V0), not reach(V0).
3 ~  :- reach(V0), node(V1), not reach(V1).
3 ~ 0 {in(V0,V0) } 1 :- edge(V0, V0).
3 ~ 0 {in(V0,V0) } 1 :- edge(V0, V1).
3 ~ 0 {in(V0,V1) } 1 :- edge(V0, V1).
3 ~ 0 {in(V1,V0) } 1 :- edge(V0, V1).
3 ~ 0 {in(V1,V1) } 1 :- edge(V0, V1).
3 ~ reach(V0) :- in(1,V0), reach(V1).
3 ~ reach(V0) :- in(V0,V1), V0 != V1.
3 ~ reach(V0) :- in(V0,V1), in(1,V0).
3 ~ reach(V0) :- in(V0,V1), in(1,V1).
3 ~ reach(V0) :- in(V0,V1), in(1,V2).
3 ~ reach(V0) :- in(V0,V1), in(V0,V2).
3 ~ reach(V0) :- in(V0,V1), in(V1,V2).
3 ~ reach(V0) :- in(V0,V1), reach(V1).
3 ~ reach(V0) :- in(V0,V1), reach(V2).
3 ~ reach(V1) :- edge(V0, V0), in(1,V1).
3 ~ reach(V1) :- edge(V0, V0), in(V1,V2).
```

```
3 ~ reach(V1) :- in(V0,V1), V0 != V1.
3 ~ reach(V1) :- in(V0,V1), in(1,V0).
3 ~ reach(V1) :- in(V0,V1), in(1,V1).
3 ~ reach(V1) :- in(V0,V1), in(1,V2).
3 ~ reach(V1) :- in(V0,V1), in(V0,V2).
3 ~ reach(V1) :- in(V0,V1), in(V1,V2).
3 ~ reach(V1) :- in(V0,V1), reach(V0).
3 ~ reach(V1) :- in(V0,V1), reach(V2).
3 ~ reach(V2) :- edge(V0, V0), in(V1,V2).
3 ~ reach(V2) :- edge(V0, V1), in(1,V2).
3 ~ reach(V2) :- in(V0,V1), in(1,V2).
3 ~ reach(V2) :- in(V0,V1), in(V0,V2).
3 ~ reach(V2) :- in(V0,V1), in(V1,V2).
```

A.2.2 Scheduling

```
#modeo(3, assigned(var(day), var(time)), (positive)).
#modeo(1, type(var(day),var(time),const(type)), (positive)).
#modeo(1, neq(var(day),var(day)), (positive, symmetric, anti_reflexive)).
#constant(type, jmc).
#weight(1).
#weight(-1).
#maxv(4).
```

A.2.3 Journey Preferences

```
#modeo(1,leg_mode(var(leg), const(mode_of_transport)), (positive)).
#modeo(1,leg_distance(var(leg), var(distance)), (positive)).
#modeo(1,leg_crime_rating(var(leg), var(crime_rate)), (positive)).
#modeo(1, var(crime_rate) > const(crime_rate)).
#weight(distance).
#weight(1).
#constant(mode_of_transport, walk).
#constant(mode_of_transport, bus).
#constant(mode_of_transport, bicycle).
#constant(mode_of_transport, car).
#constant(crime_rate, 1).
#constant(crime_rate, 2).
#constant(crime_rate, 3).
#constant(crime_rate, 4).
#maxp(3).
#maxv(2).
```

A.2.4 Agent A

```
#modeh(valid_move(var(cell), var(time))).
#modeb(1, adjacent(var(cell), var(cell)), (symmetric, anti_reflexive)).
#modeb(1, agent_at(var(cell), var(time))).
#modeb(1, unlocked(var(cell), var(time))).
#modeb(1, wall(var(cell), var(cell)), (symmetric, anti_reflexive)).
#modeb(1, link(var(cell), var(cell)), (anti_reflexive)).
```

A.2.5 Agent B

```
#modeh(valid_move(var(cell2), var(time))).
#modeh(extra(var(cell2), var(time))).

#modeb(1, extra(var(cell2), var(time)), (positive)).
#modeb(1, visited_cell(var(cell), var(time)), (positive)).

#modeb(1, locked(var(cell2))).

#modeb(1, agent_at(var(cell), var(time)), (positive)).
#modeb(1, wall(var(cell), var(cell2))).
#modeb(1, adjacent(var(cell), var(cell2)), (positive)).
#modeb(1, link(var(cell), var(cell2)), (positive)).
#modeb(1, key(var(cell), var(cell2)), (positive)).
```

A.2.6 Agent C

```
#modeh(valid_move(var(cell2), var(time))).
#modeb(1, adjacent(var(cell), var(cell2)), (positive, symmetric, anti_reflexive)).
#modeb(1, agent_at(var(cell), var(time)), (positive)).
#modeb(1, already_visited_cell(var(cell), var(time)), (positive)).
#modeb(1, unlocked(var(cell2), var(time)), (positive)).
#modeb(1, wall(var(cell), var(cell2)), (symmetric, anti_reflexive)).
#modeb(1, link(var(cell), var(cell2)), (positive, anti_reflexive)).
```

A.2.7 Agent D

```
#modeo(1, agent_at(var(cell), var(time)), (positive)).
#modeo(1, daytime(var(time))).
#modeo(1, danger_rating(var(cell), var(danger)), (positive)).
#modeo(1, coin(var(cell), var(value))).
#weight(1).
#weight(danger).
#weight(value).
#maxp(3).

#modeh(valid_move(var(cell2), var(time))).
#modeb(1, adjacent(var(cell), var(cell2)), (positive, symmetric, anti_reflexive)).
#modeb(1, agent_at(var(cell), var(time)), (positive)).
#modeb(1, already_visited_cell(var(cell), var(time)), (positive)).
#modeb(1, unlocked(var(cell2), var(time)), (positive)).
#modeb(1, wall(var(cell), var(cell2)), (symmetric, anti_reflexive)).
#modeb(1, link(var(cell), var(cell2)), (positive, anti_reflexive)).
```

A.2.8 Predecessor

```
#modeh(predecessor(var(any),var(any))).
#modeb(succ(var(any),var(any))).
#modeb(number(var(any)), (positive)).
#maxv(2).
#maxbl(3).
```

A.2.9 Less Than

```
#modeh(less_than(var(any), var(any)), (positive)).
#modeb(1, succ(var(any), var(any)), (positive)).
#modeb(1, less_than(var(any), var(any)), (positive)).
#maxbl(2).
```

A.2.10 Member

```
#modeh(member(var(number), var(list))).
#modeh(p(var(number), var(list))).
#modeb(1, p(var(number), var(list))).
#modeb(1, value(var(list), var(number)), (positive)).
#modeb(1, cons(var(list), var(list)), (positive)).
#modeb(list(var(list)), (positive)).
#modeb(number(var(number)), (positive)).
```

```
#maxbl(5).

% enforces that the list/number nodes are used as typing atoms (i.e. they occur
% if and only if a variable of that type occurs).
#bias(":- body(p(V1, V2)), not body(number(V1)).").
#bias(":- body(p(V1, V2)), not body(list(V2)).").
#bias(":- body(value(V1, V2)), not body(list(V1)).").
#bias(":- body(value(V1, V2)), not body(number(V2)).").
#bias(":- body(cons(V1, V2)), not body(list(V1)).").
#bias(":- body(cons(V1, V2)), not body(list(V2)).").
#bias(":- body(number(V1)), not body(p(V1, _)), not body(naf(p(V1, _))),
          not body(naf(value(_, V1))), not body(value(_, V1)).").
#bias(":- body(list(V1)), not body(p(_, V1)), not body(naf(p(_, V1))),
          not body(naf(value(V1, _))), not body(value(V1, _)),
          not body(cons(V1, _)), not body(cons(_, V1)).").
#bias(":- head(p(_, _)), body(naf(p(_, _))).").
```

A.2.11 Connected

```
#modeh(connected(var(any), var(any)), (positive)).
#modeb(1, edge(var(any), var(any)), (positive)).
#modeb(1, connected(var(any), var(any)), (positive)).
#maxbl(2).
```

A.2.12 Undirected Edge

```
#modeh(undirected_edge(var(node), var(node))).
#modeb(edge(var(node), var(node))).
#modeb(node(var(node)), (positive)).
#maxbl(5).

% enforces that the node atoms are used as typing atoms (i.e. they occur if and
% only if a variable of that type occurs).
#bias(":- body(edge(V1, V2)), not body(node(V1)).").
#bias(":- body(edge(V1, V2)), not body(node(V2)).").
#bias(":- body(naf(edge(V1, V2))), not body(node(V1)).").
#bias(":- body(naf(edge(V1, V2))), not body(node(V2)).").
#bias(":- body(node(V)), not body(edge(V, _)), not body(edge(_, V)),
          not body(naf(edge(V, _))), not body(naf(edge(_, V))).").
```

A.2.13 Sentence Chunking

```
#constant(postype, c). % for each part of speach tag that occurs in the task.

#modeh(split(var(token))).
#modeb(1, pos(const(postype),var(token)), (positive)).
#modeb(1, prevpos(const(postype),var(token)), (positive)).
#modeb(1, test_split(var(token)), (positive)).
#maxv(1).

% This constraint means that test_split(V0) must occur in the body of every
% rule. As each context contains test_split(X) for each X such that split(X)
% occurs in the inclusions or exclusions of the corresponding partial
% interpretation, this just means that the only ground instances of rules in
% the hypothesis space that are considered are those that could affect whether
% or not the example is covered.
#bias(":- not body(test_split(_)).").
```

A.2.14 Cars

```
#weight(cap).
#weight(1).
#weight(-1).
#modeo(1,body(const(bool)), (positive)).
#modeo(1,transmission(const(bool)), (positive)).
#modeo(1,fuel(const(bool)), (positive)).
#modeo(1,engine_cap(var(cap))).
#constant(bool, 1).
#constant(bool, 2).
#maxp(5).
```

A.2.15 SUSHI

```
#modeo(1, value(const(val),var(val))).
#modeo(1, minor_group(const(mg)), (positive)).
#modeo(1, seafood, (positive)).
#modeo(1, maki, (positive)).
#maxp(5).
#weight(val).
#weight(1).
#weight(-1).
#constant(val, oil).
#constant(val, price).
#constant(val, freq).
```

```
#constant(val, freq2).

% The constants for minor groups 2, 4 and 10 were omitted as none of the
% sushi's in this part of the dataset have any of these minor groups.
#constant(mg, 1).
#constant(mg, 3).
#constant(mg, 5).
#constant(mg, 6).
#constant(mg, 7).
#constant(mg, 8).
#constant(mg, 9).
#constant(mg, 11).
```

Appendix B

Proofs and Meta-programs Omitted from the Main Thesis

Proofs from Chapter 2

Lemma 2.11.

Let P be an $\mathcal{ASP}^R$ program and I be an interpretation. I is an answer set of P if and only if I is a model of P and there is no non-empty unfounded subset of I wrt P.

Proof. Let P be an $\mathcal{ASP}^R$ program and I be an interpretation.

1. Assume that $I \in AS(P)$. We must show that (a) I is a model of P and (b) I has no non-empty unfounded sets.

   (a) Assume for contradiction that I is not a model of P. This means that there is a rule R in P such that I satisfies $body(R)$ but does not satisfy $head(R)$. In the case that R is a choice rule, P^I will contain the rule $\bot$`:-body`$^+$`(R)`, which means that $M(P^I)$ must either contain $\bot$ (which it cannot do, if I is an answer set), or must not satisfy $body^+(R)$. Hence $I \neq M(P^I)$. This contradicts the assumption that I is an answer set.

   (b) Assume for contradiction that I has a non-empty unfounded set U.
   $\Rightarrow$ There is no $R \in P$ st $heads(R) \cap U \neq \emptyset$, $body^+(R) \subseteq I \backslash U$ and $body^-(R) \cap I = \emptyset$.
   $\Rightarrow$ There is no rule R in P^I such that $head(R) \in U$, $body(R)^+ \subseteq I \backslash U$ (as any rule in P that satisfies these two conditions must contain a negative literal `not a` for some `a` $\in I$, and will therefore be removed when constructing the reduct).

   As I is an answer set of P, $I = M(P^I)$. But consider the smaller interpretation $I' = I \backslash U$. This must also be a model of P^I: for every rule R in P^I with an element of U in the head, I' does not satisfy $\mathit{body}^+(R)$ and hence I' satisfies the rule; and for every other rule $R \in P^I$ (st $\mathit{head}(R) \notin U$), if I' satisfies the body, then I must satisfy the body (as R is definite), and hence $\mathit{head}(R) \in I$, so $\mathit{head}(R) \in I'$ (as $\mathit{head}(R) \notin U$ and $I' = I \backslash U$). Contradiction! (as I is an answer set of P, I must be the minimal model of P^I).

2. Assume that I is a model of P and there is no non-empty unfounded subset U of I wrt P. We must show that $I = M(P^I)$. We first show that (a) it is a model of P^I and then show that (b) nothing smaller can be a model.

   (a) I is a model of P.

   $\Rightarrow \nexists R \in P$ st $\mathit{body}^+(R) \subseteq I$, $\mathit{body}^-(R) \cap I = \emptyset$ and $\mathit{head}(R)$ is not satisfied by R.

   Let R be in P^I. From the definition of the reduct for $\mathcal{ASP}^R$ programs, there are four possible cases:

   Case 1: There is a normal rule $R' \in P$ such that $\mathit{body}^-(R') \cap I = \emptyset$, $\mathit{body}^+(R') = \mathit{body}^+(R)$ and $\mathit{head}(R') = \mathit{head}(R)$. Hence if the body of R is satisfied by I, then the head of R must be satisfied by I.

   Case 2: $\mathit{head}(R) = \bot$ and there is a constraint $R' \in P$ such that $\mathit{body}^-(R') \cap I = \emptyset$ and $\mathit{body}^+(R') = \mathit{body}^+(R)$. As I is a model of P, $\mathit{body}^+(R')$ must contain at least one element that is not in I. Hence the body of R is not satisfied by I, and so R is satisfied by I.

   Case 3: $\mathit{head}(R) = \bot$ and there is a choice rule $R' \in P$ such that $\mathit{body}^-(R') \cap I = \emptyset$, $\mathit{body}^+(R') = \mathit{body}^+(R)$ and $\mathit{head}(R')$ is not satisfied by I. As I is a model of P, $\mathit{body}^+(R')$ must contain at least one element that is not in I. Hence the body of R is not satisfied by I, and so R is satisfied by I.

   Case 4: There is a choice rule $R' \in P$ such that $\mathit{body}^-(R') \cap I = \emptyset$, $\mathit{body}^+(R') = \mathit{body}^+(R)$, $\mathit{head}(R')$ is satisfied by I, $\mathit{head}(R) \in \mathit{heads}(R')$ and $\mathit{head}(R) \in I$. As $\mathit{head}(R) \in I$, I satisfies R.

   Hence I satisfies every rule in P^I, and so must be a model of P^I.

   (b) We now show that no subset $I' \subset I$ can be a model of P^I. We assume the converse for contradiction. Let $I' \subset I$ be a model of P^I. Consider the set $U = I \backslash I'$. As U is non-empty, it cannot be unfounded; and hence, there must be at least one rule $R \in P$ such that $\mathit{heads}(R) \cap U \neq \emptyset$, $\mathit{body}^+(R) \subseteq I'$ and $\mathit{body}^-(R) \cap I = \emptyset$. As I is a model of P, I must satisfy $\mathit{head}(R)$, and hence, there is a definite rule $\mathtt{u}$:- $\mathit{body}^+(R)$ in P^I, where $\mathit{body}^+(R) \subseteq I'$ and $\mathtt{u} \in U$. Hence I' cannot be a model of P^I, as it satisfies the body of this rule, but $\mathtt{u} \notin I'$.

□

Lemma 2.12.

Let P be any ASP program, and C be an ASP program containing only hard constraints. $A \in AS(P \cup C)$ if and only if $A \in AS(P)$ and A does not satisfy the body of any instance of any constraint in C.

Proof.

1. We first show that $A \in AS(P \cup C) \Rightarrow A$ does not satisfy the body of any ground instance of any constraint in C.

   Assume $A \in AS(P \cup C)$. Assume for contradiction that A satisfies the body of a ground instance g of some constraint in C. Then $\bot \text{:-} \mathit{body}^{+}(g)$ would be in the reduct $\mathit{ground}(P \cup C)^{A}$. Hence, as A must be a model of $\mathit{ground}(P \cup C)^{A}$, A must contain $\bot$. This is a contradiction, as no answer set is allowed to contain $\bot$. So A does not satisfy the body of any ground instance of any constraint in C.

2. We now show that $A \in AS(P \cup C) \Rightarrow A \in AS(P)$

   Assume $A \in AS(P \cup C)$. First note that $\mathit{ground}(P)^{A} \subseteq \mathit{ground}(P \cup C)^{A}$. Hence, as A is a model of $\mathit{ground}(P \cup C)^{A}$, it is also a model of $\mathit{ground}(P)^{A}$. It remains to show that no subset of A is a model of $\mathit{ground}(P)^{A}$. Assume for contradiction that some $A' \subset A$ is a model of $\mathit{ground}(P)^{A}$. We know from part (1) that A does not satisfy the body of any constraint in $\mathit{ground}(C)$, so it cannot satisfy the body of any constraint in $\mathit{ground}(C)^{A}$. As $A' \subset A$ and the constraints in $\mathit{ground}(C)^{A}$ contain only positive body literals, this means that A' does not satisfy the body of any constraint in $\mathit{ground}(C)^{A}$. Hence, A' is a model of $\mathit{ground}(C)^{A}$. As A' is also a model of $\mathit{ground}(P)^{A}$, and $\mathit{ground}(P \cup C)^{A} = \mathit{ground}(P)^{A} \cup \mathit{ground}(C)^{A}$, A' must be a model of $\mathit{ground}(P \cup C)^{A}$. This contradicts A being an answer set of $P \cup C$.

3. Finally, we show that any answer set A of $AS(P)$ that does not satisfy any ground instance of any constraint in C must be an answer set of $P \cup C$.

   We know that $A = M(\mathit{ground}(P)^{A})$. As A does not satisfy the body of any constraint in $\mathit{ground}(C)$, A cannot satisfy the body of any constraint in $\mathit{ground}(C)^{A}$. Hence, A is a model of $\mathit{ground}(C)^{A}$. Hence, A is a model of $\mathit{ground}(P \cup C)^{A}$ (as $\mathit{ground}(P \cup C)^{A} = \mathit{ground}(P)^{A} \cup \mathit{ground}(C)^{A}$). It remains to show that no subset A' of A is a model of $\mathit{ground}(P \cup C)^{A}$. But if there were such an A' it would also be a model of $\mathit{ground}(P)^{A}$, which would contradict A being an answer set of P.

□

Theorem 2.17.

Let P be any $\mathcal{ASP}^{R}$ program such that $|HB^{rel}_{P}|$ is finite.

1. $|\mathit{ground}^{rel}(P)|$ is finite

2. $AS(P) = AS(\mathit{ground}^{rel}(P))$

Proof.

1. As any ASP program P contains a finite set of rules, it remains to show that for an arbitrary rule R, there are a finite number of ground instances R^g of R such that $body^+(R^g) \subseteq HB_P^{rel}$.

   As R must be safe, no two distinct ground instances R_1^g and R_2^g of R can share the same positive body literals. Hence it remains to show that there are a finite number of ground instances of $body^+(R)$ that are subsets of HB_P^{rel}. In fact, there are at most $|HB_P^{rel}|^{|body^+(R)|}$ such ground instances. Hence as both $|HB_P^{rel}|$ and $|body^+(R)|$ are finite, there are a finite number of ground instances of R.

2. (a) We first show that $AS(P) \subseteq AS(ground^{rel}(P))$.

   Assume that $I \in AS(P)$

   $\Rightarrow I$ is a model of $ground(P)$ and no non-empty subset U of I is unfounded wrt $ground(P)$.

   $\Rightarrow I$ is a model of $ground^{rel}(P)$ and no non-empty subset U of I is unfounded wrt $ground(P)$ (as $ground^{rel}(P) \subseteq ground(P)$).

   It remains to show that I does not have any non-empty unfounded subsets wrt $ground^{rel}(P)$.

   Assume for contradiction that $I \not\subseteq HB_P^{rel}$. As HB_P^{rel} is a fixpoint of f_P, there is no rule $R \in ground(P)$ such that $heads(R) \not\subseteq HB_P^{rel}$ and $body^+(R) \subseteq HB_P^{rel}$. Hence, there is no rule in $ground(P)$ such that $heads(R) \cap (I \backslash HB_P^{rel}) \neq \emptyset$ and $body^+(R) \subseteq (I \backslash (I \backslash HB_P^{rel}))$, meaning that $(I \backslash HB_P^{rel})$ is a non-empty unfounded subset of I wrt $ground(P)$ (contradicting that I is an answer set of P). Hence, $I \subseteq HB_P^{rel}$.

   Let U be a non-empty subset of I. As U cannot be an unfounded subset of I wrt $ground(P)$, $\exists R \in ground(P)$ such that $heads(R) \cap U \neq \emptyset$, $body^+(R) \subseteq (I \backslash U)$ and $body^-(R) \cap I = \emptyset$. In order to show that U is not an unfounded subset of I wrt $ground^{rel}(P)$, it suffices to show that $R \in ground^{rel}(P)$. We can do this by showing that $body^+(R) \subseteq HB_P^{rel}$. This is clearly the case as $body^+(R) \subseteq (I \backslash U) \subseteq I \subseteq HB_P^{rel}$.

   (b) It remains to show that $AS(ground^{rel}(P)) \subseteq AS(ground(P)$.

   Assume $I \in AS(ground^{rel}(P))$.

   $\Rightarrow I$ is a model of $ground^{rel}(P)$ and I has no non-empty unfounded subsets wrt $ground^{rel}(P)$

   $\Rightarrow I$ is a model of $ground^{rel}(P)$ and I has no non-empty unfounded subsets wrt $ground(P)$ (as $ground^{rel}(P) \subseteq ground(P)$, any rule in $ground^{rel}(P)$ that prevents a non-empty subset from being unfounded must also be in $ground(P)$).

   It remains to show that there is no rule R in $ground(P)$ such that I is not a model of R. Assume for contradiction that such a rule R does exist. Then there is an $R \in ground(P)$ such that $body(R)$ is satisfied by I but $head(R)$ is not satisfied by I. This R cannot occur in $ground^{rel}(P)$ (or I would not be a model of $ground^{rel}(P)$), and so there must be an atom $\mathtt{a} \in body^+(R)$ such that $\mathtt{a} \notin HB_P^{rel}$. As I is not a model of R, $\mathtt{a} \in I$. Hence there is an atom $\mathtt{a} \in I$ such that there is no rule $R' \in ground^{rel}(P)$ for which $\mathtt{a} \in heads(R')$. Hence $\{\mathtt{a}\}$ is an unfounded subset of I wrt $ground^{rel}(P)$. Contradiction (as $I \in AS(ground^{rel}(P))$).

□

Proofs from Chapter 4

Before proving Theorem 4.9, it is useful to introduce the following lemma.

Lemma B.1. *(proof on page 266)*

For any program P (consisting of normal rules, choice rules and constraints) and any set of pairs $S = \{\langle C_1, \mathtt{a_1}\rangle, \ldots, \langle C_n, \mathtt{a_n}\rangle\}$ such that none of the atoms $\mathtt{a_i}$ appear in P (or in any of the C's) and each $\mathtt{a_i}$ atom is unique:

$$AS\left(\begin{array}{ll} P & \cup \ \{\mathtt{1}\{\mathtt{a_1}, \ldots, \mathtt{a_n}\}\mathtt{1.}\} \\ & \cup \ \{append(C_i, \mathtt{a_i}) | \langle C_i, \mathtt{a_i}\rangle \in S\}\end{array}\right) = \left\{ A \cup \{\mathtt{a_i}\} \middle| \begin{array}{c} A \in AS(P \cup C_i), \\ \langle C_i, \mathtt{a_i}\rangle \in S \end{array}\right\}$$

Proof. The answer sets of $\{\mathtt{1}\{\mathtt{a_1}, \ldots, \mathtt{a_n}\}\mathtt{1.}\}$ are $\{\mathtt{a_1}\}, \ldots, \{\mathtt{a_n}\}$, hence by the splitting set theorem (using $U = \{\mathtt{a_1}, \ldots, \mathtt{a_n}\}$ as a splitting set):

$$\begin{array}{l} AS(P \cup \{\mathtt{1}\{\mathtt{a_1}, \ldots, \mathtt{a_n}\}\mathtt{1.}\} \cup \{append(C_i, \mathtt{a_i}) | \langle C_i, \mathtt{a_i}\rangle \in S\}) \\ \quad = \left\{ A' \cup \{\mathtt{a_j}\} \middle| \begin{array}{c} \mathtt{a_j} \in \{\mathtt{a_1}, \ldots, \mathtt{a_n}\}, \\ A' \in AS(e_U(P \cup \{append(C_i, \mathtt{a_i}) \mid \langle C_i, \mathtt{a_i}\rangle \in S\}, \{\mathtt{a_j}\})) \end{array}\right\} \\ \quad = \{A \cup \{\mathtt{a_i}\} | A \in AS(P \cup C_i), \langle C_i, \mathtt{a_i}\rangle \in S\}. \end{array}$$

□

Theorem 4.9.

For any $ILP^{context}_{LOAS}$ learning task T, $ILP_{LOAS}(\mathcal{T}_{LOAS}(T)) = ILP^{context}_{LOAS}(T)$.

Proof. Let $T = \langle B_1, S_M, \langle E^+_1, E^-_1, O^b_1, O^c_1\rangle\rangle$ and $\mathcal{T}_{LOAS}(T) = \langle B_2, S_M, \langle E^+_2, E^-_2, O^b_2, O^c_2\rangle\rangle$.

Assume $H \in ILP^{context}_{LOAS}(T)$

$\Leftrightarrow$ $H \subseteq S_M$; $\forall \langle e_{pi}, e_{ctx}\rangle \in E^+_1, \exists A \in AS(B_1 \cup e_{ctx} \cup H)$ st A extends e_{pi}; $\forall \langle e_{pi}, e_{ctx}\rangle \in E^-_1, \nexists A \in AS(B_1 \cup e_{ctx} \cup H)$ st A extends e_{pi}; $\forall o \in O^b_1, B_1 \cup H$ bravely respects o; $\forall o \in O^c_1, B_1 \cup H$ cautiously respects o

$\Leftrightarrow$ $H \subseteq S_M$; $\forall e \in E^+_1, \exists A \in AS(B_2 \cup H)$ st A extends $c(e)$; $\forall e \in E^-_1, \nexists A \in AS(B_2 \cup H)$ st A extends $c(e)$; $\forall \langle e_1, e_2, op\rangle \in O^b_1, B_2 \cup H$ bravely respects $\langle c(e_1), c(e_2), op\rangle$; $\forall \langle e_1, e_2, op\rangle \in O^c_1, B_2 \cup H$ cautiously respects $\langle c(e_1), c(e_2), op\rangle$ (by Lemma B.1)

$\Leftrightarrow$ $H \subseteq S_M$; $\forall e \in E^+_2, \exists A \in AS(B_2 \cup H)$ st A extends e; $\forall e \in E^-_2, \nexists A \in AS(B_2 \cup H)$ st A extends e; $\forall o \in O^b_2, B_2 \cup H$ bravely respects o; $\forall o \in O^c_2, B_2 \cup H$ cautiously respects o

$\Leftrightarrow$ $H \in ILP_{LOAS}(\mathcal{T}_{LOAS}(T))$

□

Proposition 4.11.

1. Deciding verification, satisfiability and optimum verification for ILP_b each reduce polynomially to the corresponding ILP_{sm} decision problem.

2. Deciding verification, satisfiability and optimum verification for ILP_{sm} each reduce polynomially to the corresponding ILP_b decision problem.

Proof.

1. Let $T_b = \langle B, S_M, \langle E^+, E^- \rangle \rangle$ be any arbitrary ILP_b task.

   Consider the task $T_{sm} = \langle B, S_M, \langle \{\langle E^+, E^- \rangle\} \rangle \rangle$. $\forall H$, $H \in ILP_{sm}(T_{sm})$ if and only if $H \in ILP_b(T_b)$ and hence deciding verification for ILP_b reduces polynomially to deciding verification for ILP_{sm}. Similarly, as $ILP_{sm}(T_{sm}) = ILP_b(T_b)$, T_{sm} is satisfiable if and only if T_b is satisfiable; hence, deciding satisfiability for ILP_b reduces to deciding satisfiability for ILP_{sm}. Finally, as the two tasks give the same length to every possible hypothesis, $^*ILP_b(T_b) = {}^*ILP_{sm}(T_{sm})$. Hence, any hypothesis H is an optimal solution of T_b if and only if it is an optimal solution of T_{sm}; and hence, deciding optimum verification for ILP_b reduces to deciding optimum verification for ILP_{sm}.

2. Let $T_{sm} = \langle B, S_M, \langle E \rangle \rangle$ be any arbitrary ILP_{sm} task and let $E = \{e_1, \ldots, e_n\}$.

   For each integer i from 1 to n, let f_i be a function which maps each atom $\texttt{a}$ in $B \cup S_M$ to a new atom $\texttt{a}_\texttt{i}$. We also extend this notation to work on sets of atoms and rules (and parts of rules) by replacing each atom $\texttt{a}$ in the set or rule with $f_i(\texttt{a})$.

   For each rule $R \in S_M$, define a new atom $\texttt{in\_h}_\texttt{R}$.

   Consider the task $T_b = \langle B^b, S_M^b, \langle E^+, E^- \rangle \rangle$ where the components of the task are as follows ($append(R, \texttt{a})$ is the rule R with the atom $\texttt{a}$ appended to the body).

   $$B^b = \left\{ \begin{array}{c} f_1(R), \\ \ldots, \\ f_n(R) \end{array} \middle| R \in B \right\} \bigcup \left\{ \begin{array}{c} append(f_1(R), \texttt{in\_h}_\texttt{R}), \\ \ldots, \\ append(f_n(R), \texttt{in\_h}_\texttt{R}) \end{array} \middle| R \in S_M \right\}$$

   $S_M^b = \left\{ \texttt{in\_h}_\texttt{R} \middle| R \in S_M \right\}$ such that the length of each $\texttt{in\_h}_\texttt{R}$ atom is defined as $|R|$.

   $E^+ = \left\{ \texttt{f}_\texttt{i}(\texttt{inc}) \middle| e_i \in E, e_i = \langle e_i^{inc}, e_i^{exc} \rangle, \texttt{inc} \in e_i^{inc} \right\}$

   $E^- = \left\{ \texttt{f}_\texttt{i}(\texttt{exc}) \middle| e_i \in E, e_i = \langle e_i^{inc}, e_i^{exc} \rangle, \texttt{exc} \in e_i^{exc} \right\}$

   For any solution H of T_b, define $g(H)$ to be $\{R \mid \texttt{in\_h}_\texttt{R} \in H\}$. We now show that $ILP_{sm}(T_{sm}) = \{g(H') \mid H' \in ILP_b(T_b)\}$.

   Assume $H \in ILP_{sm}(T_{sm})$

   $\Leftrightarrow H \subseteq S_M$ and $\forall e_i \in E, \exists A \in AS(B \cup H)$ such that A extends e_i.

   $\Leftrightarrow H \subseteq S_M$ and $\forall e_i \in E, \exists A \in AS(f_i(B \cup H))$ such that A extends $\langle \{f_i(\texttt{inc}) \mid \texttt{inc} \in e_i^{inc}\}, \{f_i(\texttt{exc}) \mid \texttt{exc} \in e_i^{exc}\} \rangle$.

   $\Leftrightarrow H \subseteq S_M$ and $\exists A \in AS(\{f_i(B \cup H) \mid 1 \leq i \leq n\})$ such that A extends $\langle E^+, E^- \rangle$ (as the atoms in each sub program are disjoint).

   $\Leftrightarrow H \subseteq S_M$ and $\exists A \in AS(B^b \cup \{\texttt{in\_h}_\texttt{R} \mid R \in H\})$ such that A extends $\langle E^+, E^- \rangle$ (by the splitting set theorem, using $\{\texttt{in\_h}_\texttt{R} \mid R \in H\}$ as a splitting set).

   $\Leftrightarrow \exists H' \subseteq S_M^b$ such that $g(H') = H$ and $\exists A \in AS(B^b \cup H')$ such that A extends $\langle E^+, E^- \rangle$

   $\Leftrightarrow \exists H' \in ILP_b(T_b)$ such that $g(H') = H$

   $\Leftrightarrow H \in \{g(H') \mid H' \in ILP_b(T_b)\}$

$\forall H$, $H \in ILP_{sm}(T_{sm})$ if and only if $\{\texttt{in\_h}_\texttt{R} \mid R \in H\} \in ILP_b(T_b)$ and hence deciding verification for ILP_{sm} reduces polynomially to deciding verification for ILP_b. Similarly, as $ILP_{sm}(T_{sm}) = \{g(H) \mid H \in ILP_b(T_b)\}$, T_{sm} is satisfiable if and only if T_b is satisfiable; hence, deciding satisfiability for ILP_{sm} reduces to deciding satisfiability for ILP_b.

Finally, for any arbitrary H, assume that $H \in {}^*ILP_{sm}(T_{sm})$

$\Leftrightarrow H \in ILP_{sm}(T_{sm})$ and $\forall H' \in S_M$ such that $|H'| < |H|$, $H' \notin ILP_{sm}(T_{sm})$

$\Leftrightarrow \{\texttt{in\_h}_\texttt{R} \mid R \in H\} \in ILP_b(T_b)$ and $\forall H' \in S_M$ such that $|H'| < |H|$, $\{\texttt{in\_h}_\texttt{R} \mid R \in H'\} \notin ILP_b(T_b)$

$\Leftrightarrow \{\texttt{in\_h}_\texttt{R} \mid R \in H\} \in ILP_b(T_b)$ and $\forall H' \in S_M$ such that $|\{\texttt{in\_h}_\texttt{R} \mid R \in H'\}| < |\{\texttt{in\_h}_\texttt{R} \mid R \in H\}|$, $\{\texttt{in\_h}_\texttt{R} \mid R \in H'\} \notin ILP_b(T_b)$

$\Leftrightarrow \{\texttt{in\_h}_\texttt{R} \mid R \in H\} \in {}^*ILP_b(T_b)$

Hence, deciding optimum verification for ILP_{sm} reduces to deciding optimum verification for ILP_b.

□

Proposition 4.12.

1. Deciding verification, satisfiability and optimum verification for ILP_c each reduce polynomially to the corresponding ILP_{LAS} decision problem.

2. Deciding verification, satisfiability and optimum verification for ILP_{LAS} each reduce polynomially to the corresponding $ILP^{context}_{LOAS}$ decision problem.

3. Deciding verification, satisfiability and optimum verification for $ILP^{context}_{LOAS}$ each reduce polynomially to the corresponding ILP_{LOAS} decision problem.

4. Deciding verification, satisfiability and optimum verification for ILP_{LOAS} each reduce polynomially to the corresponding ILP^{s}_{LAS} decision problem.

Proof.

First note that if we can show that there is a polynomial mapping M, such that for any $ILP_{\mathcal{F}_1}$ task T, $M(T)$ is an $ILP_{\mathcal{F}_2}$ task such that such that $ILP_{\mathcal{F}_1}(T) = ILP_{\mathcal{F}_2}(M(T))$ (and the length of all hypotheses is preserved by M), then this suffices to show all three polynomial reductions. For verification, we can check that H is an inductive solution of T, by checking that it is an inductive solution of $M(T)$; for satisfiability, we can check that T is satisfiable by checking that $M(T)$ is satisfiable; and finally, as the length of all hypotheses is preserved by M, ${}^*ILP_{\mathcal{F}_1}(T) = {}^*ILP_{\mathcal{F}_2}(M(T))$, and hence, we can check that H is an optimal inductive solution of T, by checking that it is an optimal inductive solution of $M(T)$.

1. Let T_c be any ILP_c task $\langle B, S_M, \langle E^+, E^- \rangle\rangle$.

   Let $M(T_c) = \langle B, S_M, \langle \{\langle \emptyset, \emptyset\rangle\}, \{\langle \emptyset, \{e^+\}\rangle \mid e^+ \in E^+\} \cup \{\langle \{e^-\}, \emptyset\rangle \mid e^- \in E^-\}\rangle\rangle$.

   By the definition of ILP_{LAS}, $H \in ILP_{LAS}(M(T_c))$ if and only if $H \subseteq S_M$; $\exists A \in AS(B \cup H)$ such that A extends $\langle \emptyset, \emptyset\rangle$; $\forall e^+ \in E^+$, $\nexists A \in AS(B \cup H)$ such that A extends $\langle \emptyset, \{e^+\}\rangle$; and finally, $\forall e^- \in E^-$, $\nexists A \in AS(B \cup H)$ such that A extends $\langle \{e^-\}, \emptyset\rangle$.

This is true if and only if $H \subseteq S_M$, $B \cup H$ is satisfiable, $\forall e^+ \in E^+$ $\forall A \in AS(B \cup H)$, $e^+ \in A$ and $\forall e^- \in E^-$ $\forall A \in AS(B \cup H)$ $e^- \notin A$. This is the definition of H being a member of $ILP_c(T_c)$; hence, $ILP_c(T_c) = ILP_{LAS}(M(T_c))$.

Hence, there is a polynomial mapping from ILP_c tasks to ILP_{LAS} tasks.

2. Let T_{LAS} be any ILP_{LAS} task $\langle B, S_M, \langle E^+, E^- \rangle\rangle$. Let $M(T_{LAS}) = \langle B, S_M, \langle E^+, E^-, \emptyset, \emptyset\rangle\rangle$. Clearly, $ILP_{LAS}(T_{LAS}) = ILP_{LOAS}(M(T_{LAS}))$.

3. For any $ILP^{context}_{LOAS}$ task $T^{context}_{LOAS}$, let $M(T^{context}_{LOAS}) = \mathcal{T}_{LOAS}(T^{context}_{LOAS})$.

   By Theorem 4.9, $ILP^{context}_{LOAS}(T^{context}_{LOAS}) = ILP_{LOAS}(M(T^{context}_{LOAS}))$.

4. Consider the ILP_{LOAS} task $T_{LOAS} = \langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$. Before defining our translation M, we define several new atoms used in its meta-level representation.

   For $i \in \{1, 2\}$, let f_i be a function which maps each atom $\mathtt{a}$ in $B \cup S_M$ to a new atom $\mathtt{a_i}$. We also extend this notation to work on sets of atoms and rules (and parts of rules) by replacing each atom $\mathtt{a}$ in the set or rule with $f_i(\mathtt{a})$.

   For each rule $R \in S_M$, define a new atom in_h_R.

   For each weak constraint $W \in B \cup S_M$ let $id_1(W)$ and $id_2(W)$ be two new (propositional) atoms and let $wt(W)$ be the weight of W and $priority(W)$ be the priority level of W.

   For any two terms $\mathtt{t_1}$ and $\mathtt{t_2}$, $dominates(\mathtt{t_1}, \mathtt{t_2})$ is defined as below.

$$dominates(\mathtt{t_1},\mathtt{t_2}) = \left\{ \begin{array}{l|l} \begin{array}{l} \mathtt{dom\_lv(t_1,t_2,l)\,:\!-} \\ \quad \mathtt{\#sum\{id_1(W_1) = wt(W_1), \ldots, id_1(W_n) = wt(W_n),} \\ \qquad \mathtt{id_2(W_1) = -wt(W_1), \ldots, id_2(W_n) = -wt(W_n)\} < 0.} \\ \mathtt{non\_dom\_lv(t_1,t_2,l)\,:\!-} \\ \quad \mathtt{\#sum\{id_1(W_1) = wt(W_1), \ldots, id_1(W_n) = wt(W_n),} \\ \qquad \mathtt{id_2(W_1) = -wt(W_1), \ldots, id_2(W_n) = -wt(W_n)\} > 0.} \\ \mathtt{dom(t_1,t_2)\,:\!-\,dom\_lv(t_1,t_2,l),} \\ \qquad \mathtt{not\ non\_bef(t_1,t_2,l).} \end{array} & \begin{array}{l} l \text{ is a priority level in } B \cup S_M, \\ W_1, \ldots, W_n \text{ are the weak} \\ \text{constraints in } B \cup S_M \text{ with level } l \end{array} \end{array} \right\}$$

$$\cup \left\{ \mathtt{non\_bef(t_1,t_2,l_1)\,:\!-\,non\_dom\_lv(t_1,t_2,l_2).} \;\middle|\; \begin{array}{l} l_1, l_2 \text{ are levels in } B \cup S_M, \\ l_1 < l_2 \end{array} \right\}$$

   Let $M(T_{LOAS})$ be the ILP^s_{LAS} task $\langle B', S'_M, \langle E^{+'}, E^{-'}\rangle\rangle$ where the individual components are defined below. For the positive and negative examples, it is a simple reification so that the examples relate to the new B' and S'_M. The brave orderings are mapped to positive examples which can only be covered by a hypothesis H if $B \cup H$ bravely respects the ordering example. For any hypothesis $H' \in S'_M$, the f_1 and f_2 in a single answer set of $B' \cup H'$ represent two answer sets of $B \cup H$ where H is the hypothesis in S_M corresponding to H'. Similarly, cautious orderings are mapped to negative examples, such that there is an answer set of $B' \cup H'$ which extends the example if there is a pair of answer sets of the corresponding $B \cup H$ which are ordered incorrectly (i.e. if $B \cup H$ does not cautiously respect the ordering).

$$\begin{array}{rl} B' = & \{f_i(R) | R \in B, R \text{ is not a weak constraint, } i \in \{1,2\}\} \\ & \cup \{id_i(W)\,:\!-\,f_i(body(W)). | W \text{ is a weak constraint in } B, i \in \{1,2\}\} \\ & \cup \{append(f_i(R), in\_h_R) | R \in S_M, \text{ R is not a weak constraint }\} \\ & \cup \{id_i(W)\,:\!-\,append(f_i(body(W)), in\_h_W). | W \text{ is a weak constraint in } S_M\} \\ & \cup\, dominates(1,2) \cup dominates(2,1) \\ & \cup \{\mathtt{dom\,:\!-\,dom(1,2).\ \ dom\,:\!-\,dom(2,1).}\} \end{array}$$

$$
\begin{aligned}
&S'_M = \{in\_h_R | R \in S_M\}\\
&\quad E^{+'} = \{f_1(e^+) | e^+ \in E^+\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom(1,2)}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc})\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, >\rangle \in O^b\right\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom(2,1)}\}\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, \geq\rangle \in O^b\right\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc})\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, \neq\rangle \in O^b\right\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom(2,1)}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc})\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, <\rangle \in O^b\right\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom(1,2)}\}\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, \leq\rangle \in O^b\right\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom}\}\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, =\rangle \in O^b\right\}\\
&\quad E^{-'} = \{f_1(e^-) | e^- \in E^-\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom(1,2)}\}\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, >\rangle \in O^c\right\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom(2,1)}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc})\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, \geq\rangle \in O^c\right\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom}\}\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, \neq\rangle \in O^c\right\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom(2,1)}\}\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, <\rangle \in O^c\right\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom(1,2)}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc})\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, \leq\rangle \in O^c\right\}\\
&\qquad \cup \left\{\langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc})\rangle \middle| \langle\langle e_1^{inc}, e_1^{exc}\rangle, \langle e_2^{inc}, e_2^{exc}\rangle, =\rangle \in O^c\right\}
\end{aligned}
$$

By using the splitting set theorem [LT94], it can be shown that for any $H' \in S'_M$:

$$
AS(B' \cup H') = \left\{ A \,\middle|\, A \in AS\left(\begin{array}{l} f_1(A_1) \cup f_2(A_2) \\ \cup dominates(1,2) \cup dominates(2,1) \\ \cup \{\texttt{dom :- dom(1,2).}\ \texttt{dom :- dom(2,1).}\} \end{array}\right), A_1, A_2 \in AS(B \cup H) \right\}
$$

Hence, as the rules in $dominates(\texttt{t}_1, \texttt{t}_2)$ describe exactly the behaviour of the weak constraints in $B \cup H$ for two answer sets (with $\texttt{dom}(\texttt{t}_1, \texttt{t}_2)$ being true if and only if the first answer set dominates the second): $AS(B' \cup H') = \{A' | A = f_1(A_1) \cup f_2(A_2), A_1, A_2 \in AS(B \cup H)\}$, where A' is A augmented with $\texttt{dom}$ and $\texttt{dom(1,2)}$ when A_1 dominates A_2 and $\texttt{dom}$ and $\texttt{dom(2,1)}$ when A_2 dominates A_1.

For any hypothesis $H' \in S'_M$, Let H be the corresponding hypothesis in S_M. The answer sets of $B' \cup H'$ correspond to the pairs of answer sets of $B \cup H$.

Each positive example $e^+ \in E^+$ is mapped to an example in $E^{+'}$ ensuring that at least one of the pairs of answer sets' first answer set covers e^+. Note that as each answer set of $B \cup H$ must be the first element of one of these pairs at least once, this is true if and only if $B \cup H$ covers each positive example.

Similarly each negative example $e^- \in E^-$ is mapped to an example in $E^{-'}$ ensuring that none of the pairs of answer sets' first answer set covers e^-. This is true if and only if $B \cup H$ does not cover any negative examples.

Each brave ordering example $\langle e_1, e_2, op\rangle \in O^b$ is mapped to a positive example ensuring that there is a pair of answer sets $\langle A_1, A_2\rangle$ of $B \cup H$ such that A_1 covers e_1, A_2 covers e_2 and $\langle A_1, A_2, op\rangle \in ord(B \cup H)$. This is true if and only if $B \cup H$ bravely respects the ordering example.

Each cautious ordering example $\langle e_1, e_2, op\rangle \in O^c$ is mapped to a negative example ensuring that there is no pair of answer sets $\langle A_1, A_2\rangle$ of $B \cup H$ such that A_1 covers e_1, A_2 covers e_2 and $\langle A_1, A_2, op\rangle \notin ord(B \cup H)$. This is true if and only if $B \cup H$ cautiously respects the ordering example.

Hence, H' is an inductive solution of $ILP^s_{LAS}(M(T_{LOAS}))$ if and only if H is an inductive solution of $ILP_{LOAS}(T_{LOAS})$.

$\square$

Proposition 4.13. Verifying whether a given H is an inductive solution of a general ILP_b task is NP-complete.

Proof. Let T_b be any ILP_b task $\langle B, S_M, \langle E^+, E^- \rangle\rangle$. For any $H \subseteq S_M$, $H \in ILP_b$ if and only if $B \cup H \cup \{\texttt{:- not e}^+\texttt{.} \mid e^+ \in E^+\} \cup \{\texttt{:-e}^-\texttt{.} \mid e^- \in E^-\}$ is satisfiable. As deciding the satisfiability of this program is NP-complete ($B \cup H$ contains only normal rules, choice rules and constraints), this means that deciding verification for ILP_b is in NP.

It remains to show that deciding verification is NP-hard. We do this by showing that deciding satisfiability for any ASP program P containing normal rules choice rules and constraints can be reduced polynomially to deciding verification for an ILP_b task. Consider the ILP_b task $T_b = \langle P, \emptyset, \langle \emptyset, \emptyset \rangle\rangle$. Let $H = \emptyset$. $H \in ILP_b(T_b)$ if and only if there is an answer set of $P \cup H$, and hence, if and only if P is satisfiable. □

Proposition 4.15. Deciding the satisfiability of a general ILP_b task is NP-complete.

Proof. First we will show that deciding the satisfiability of a general ILP_b task is in NP. We do this by mapping an arbitrary task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$ to an ASP program whose answer sets can be mapped to the solutions of T. This program will be satisfiable if and only if T is satisfiable and as the program is aggregate stratified, checking whether the program is satisfiable is in NP. Hence, if we can construct such a program then we will have proved that deciding satisfiability for ILP_b is in NP.

For each $R^i \in S_M$ we define a new atom $\texttt{in\_h}_{\texttt{R}^\texttt{i}}$. Also, let $meta(R^i)$ be the rule R^i with the additional atom $\texttt{in\_h}_{\texttt{R}^\texttt{i}}$ added to the body.

We define the meta encoding T_{meta} as follows:

$$\begin{aligned} T_{meta} = B &\cup \{meta(R^i) \mid R^i \in S_M\} \cup \{\ \texttt{0}\{\texttt{in\_h}_{\texttt{R}^\texttt{1}}, \ldots, \texttt{in\_h}_{\texttt{R}^{|\texttt{S}_\texttt{M}|}}\}|\texttt{S}_\texttt{M}|.\ \} \\ &\cup \{\texttt{:- not e.} \mid e \in E^+\} \cup \{\texttt{:-e.} \mid e \in E^-\} \end{aligned}$$

For any answer set A, let $\mathcal{M}^{-1}(A) = \{R^i \mid R^i \in S_M, \texttt{in\_h}_{\texttt{R}^\texttt{i}} \in A\}$.

$A \in AS(T_{meta})$ if and only if $(A \backslash \{\texttt{in\_h}_{\texttt{R}^\texttt{i}} \mid R^i \in S_M\}) \in AS(B \cup \mathcal{M}^{-1}(A) \cup \{\texttt{:- not e.} \mid e \in E^+\} \cup \{\texttt{:-e.} \mid e \in E^-\})$. (This can be seen by using the splitting set theorem, with $\{\texttt{in\_h}_{\texttt{R}^\texttt{i}} \mid R^i \in S_M\}$ as the splitting set).

Hence $A \in AS(T_{meta})$ if and only if $\exists H \subseteq S_M$ such that $H = \mathcal{M}^{-1}(A)$, $(A \backslash \{\texttt{in\_h}_{\texttt{R}^\texttt{i}} \mid R^i \in S_M\}) \in AS(B \cup H)$ and A extends $\langle E^+, E^- \rangle$.

Hence T_{meta} is satisfiable if and only if $\exists H \subseteq S_M$ such that $\exists A \in AS(B \cup H)$ such that A extends $\langle E^+, E^- \rangle$. This is the case if and only if T is satisfiable.

It remains to show that deciding the satisfiability of a general ILP_b task is NP-hard. Deciding the satisfiability of a normal logic program is NP-hard, so demonstrating that deciding the satisfiability of a normal program P can be mapped to an ILP_b task is sufficient.

Let P be any normal logic program. Let T be the ILP_b task $\langle P, \emptyset, \langle \emptyset, \emptyset \rangle\rangle$. T is satisfiable if and only if $\exists H \subseteq \emptyset$ such that $\exists A \in AS(P \cup H)$ such that $\emptyset \subseteq A$ and $A \cap \emptyset = \emptyset$. This is true if and only if P is satisfiable.

Hence, deciding the satisfiability of a general ILP_b task is NP-complete.

□

Proposition 4.17. Verifying whether a given H is an optimal inductive solution of a general ILP_b task is DP-complete.

Proof.

1. We first show that the decision problem is in DP. We do so by showing that it can be reduced to two decision problems; one of which is in NP and the other is in co-NP.

   Let $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$ be an arbitrary brave induction task and assume that H is an optimal inductive solution of T.

   $\Leftrightarrow$ H is a brave inductive solution of T and there is no H' such that $|H| > |H'|$ and $|H'|$ is a brave inductive solution of T.

   $\Leftrightarrow$ H is a brave inductive solution of T and the program:
   $B \cup \{append(R, \texttt{in\_h(R}_{\texttt{id}}\texttt{)})\ \texttt{0\{in\_h(R}_{\texttt{id}}\texttt{)\}1.} \mid R \in S_M\}$
   $$\left\{\begin{array}{l} \texttt{cov:-}\ \texttt{e}_1^+, \ldots, \texttt{e}_m^+,\ \texttt{not e}_1^-, \ldots,\ \texttt{not e}_n^-. \\ \texttt{:- not cov.} \\ \texttt{:-}\ |\texttt{H}|\#\texttt{sum}\{\texttt{in\_h(R}^1_{\texttt{id}}\texttt{)} = |\texttt{R}^1|, \ldots, \texttt{in\_h(R}^k_{\texttt{id}}\texttt{)} = |\texttt{R}^k|\}. \end{array}\right\}$$
   is unsatisfiable (where $S_M = \{R^1, \ldots, R^k\}$, $E^+ = \{\texttt{e}_1^+, \ldots, \texttt{e}_m^+\}$ and $E^- = \{\texttt{e}_1^-, \ldots, \texttt{e}_n^-\}$)

   Hence, the decision problem can be reduced to two problems: verifying a hypothesis is a brave inductive solution; and deciding that a ground aggregate stratified program is unsatisfiable. Hence, as the former is in NP and the latter is in co-NP, verifying that a given hypothesis is an optimal brave inductive solution of a given task is a member of DP.

2. We now show that the decision problem is DP hard. To prove that optimum verification for ILP_b is DP-hard, we must prove that any problem in DP can be reduced to this decision problem.

   Let D be any arbitrary decision problem which is in DP.

   By the definition of DP, this is the case if and only if there exist two decision problems D_1 and D_2 such that D_1 is in NP, D_2 is in co-NP and D returns yes if and only if both D_1 and D_2 return yes.

   By Lemma 2.19 and Corollary 2.20, this is the case if and only if there are two programs P_1 and P_2 and two atoms $\texttt{a}_1$ and $\texttt{a}_2$ such that both $P_1 \models_b \texttt{a}_1$ and $P_2 \models_c \texttt{a}_2$ if and only if D returns yes. Without loss of generality we can assume that the atoms in P_1 (together with $\texttt{a}_1$) are disjoint from the atoms in P_2 (together with $\texttt{a}_2$). Let $\texttt{a}$ be a new atom that does not occur in P_1 or P_2.

   Let B be the program $P_1 \cup append(P_2, \texttt{not a}) \cup \{\texttt{:-}\ \texttt{a}_2\}$. We now show that $\{\texttt{a.}\}$ is an optimal solution of the ILP_b task $T = \langle B, \{\texttt{a.}\}, \langle \{\texttt{a}_1\}, \emptyset\rangle\rangle$ if and only if D returns yes.

   Assume $\{\texttt{a.}\}$ is an optimal solution of T

   $\Leftrightarrow$ $\{\texttt{a.}\} \in ILP_b(T)$ and $\emptyset \notin ILP_b(T)$.

   $\Leftrightarrow$ $B \cup \{\texttt{a.}\}$ has an answer set that contains $\texttt{a}_1$, but B has no such answer set.

   $\Leftrightarrow$ P_1 has an answer set that contains $\texttt{a}_1$, but B has no such answer set.
   (by the splitting set theorem, using $HB_{P_1} \cup \{\texttt{a}\}$ as a splitting set).

   $\Leftrightarrow$ P_1 has an answer set that contains $\texttt{a}_1$, but $P_1 \cup P_2 \cup \{\texttt{:-}\ \texttt{a}_2.\}$ has no such answer set (by the splitting set theorem, using $\{\texttt{a}\}$ as a splitting set).

   $\Leftrightarrow$ P_1 has an answer set that contains $\texttt{a}_1$, and $P_2 \cup \{\texttt{:-}\ \texttt{a}_2\}$ is unsatisfiable (as the atoms in P_1 and P_2 are disjoint).

$\Leftrightarrow P_1$ has an answer set that contains $\mathtt{a_1}$, and P_2 has no answer set that does not contain $\mathtt{a_2}$.

$\Leftrightarrow P_1 \models_b \mathtt{a_1}$, and $P_2 \models_c \mathtt{a_2}$.

$\Leftrightarrow D$ returns yes.

$\square$

Proposition 4.19. Deciding verification for ILP^s_{LAS} is a member of DP.

Proof. Checking whether H is an inductive solution of an ILP^s_{LAS} task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$ can be achieved by mapping T to two aggregate stratified ASP programs P^+ and P^-, such that $H \in ILP_{LAS}(T)$ if and only if P^+ bravely entails an atom and P^- cautiously entails an atom.

1. Let n be the integer $|E^+|$.

   For any integer $i \in [1, n]$, let f_i be a function mapping the atoms $\mathtt{a}$ in $B \cup H$ to new atoms $\mathtt{a_i}$. We extend the notation to allow f_i to act on ASP programs (substituting all atoms in the program).

   Let P^+ be the program:

   $$\left\{ f_i(B \cup H) \cup \left\{ \begin{array}{c} \mathtt{covered(i)\text{:-}\,f_i(e^{inc}_1), \ldots, f_i(e^{inc}_m),} \\ \mathtt{not\ f_i(e^{exc}_1), \ldots,\ not\ f_i(e^{exc}_o).} \end{array} \right\} \middle| \begin{array}{c} e^i \in E^+ \\ e^i = \langle \{\mathtt{e^{inc}_1}, \ldots, \mathtt{e^{inc}_m}\}, \{\mathtt{e^{exc}_1}, \ldots, \mathtt{e^{exc}_o}\}\rangle \end{array} \right\}$$

   P^+ can be split into n sub programs $P_1 \ldots P_n$ where each program P_i contains the rules containing the atoms generated by f_i.

   As the atoms in each sub program are disjoint from the atoms of all other subprograms, $AS(P^+) = \{A_1 \cup \ldots \cup A_n \mid A_1 \in AS(P_1), \ldots, A_n \in AS(P_n)\}$. (This follows from applying the splitting set theorem $n - 1$ times).

   For each $i \in [1, n]$, $P_i \models_b \mathtt{covered(i)}$ if and only if $\exists A \in B \cup H$ such that A extends e_i (where e_i is the i^{th} positive example). Hence $P^+ \cup \{\mathtt{covered\text{:-}\,covered(1), \ldots, covered(n).}\} \models_b \mathtt{covered}$ if and only if all the positive examples are covered. Therefore, checking whether all the positive examples are covered is in NP by Corollary 2.20.

   As checking that $H \subseteq S_M$ can be done in polynomial time, this means that checking both that $H \in S_M$ and all the positive examples are covered is in NP.

2. Let P^- be the program:

   $$B \cup H \cup \{\mathtt{covered\text{:-}\ not\ neg\_violated.}\}$$
   $$\bigcup \left\{ \begin{array}{c} \mathtt{neg\_violated\text{:-}\,e^{inc}_1, \ldots, e^{inc}_m,} \\ \mathtt{not\ e^{exc}_1, \ldots,\ not\ e^{exc}_o.} \end{array} \middle| \langle \{\mathtt{e^{inc}_1}, \ldots, \mathtt{e^{inc}_m}\}, \{\mathtt{e^{exc}_1}, \ldots, \mathtt{e^{exc}_o}\}\rangle \in E^- \right\}$$

   $P^- \models_c \mathtt{covered}$ if and only if $\nexists A \in AS(B \cup H)$ such that $\exists e^- \in E^-$ such that A extends e^-. Hence checking that all negative examples are covered is in co-NP by Lemma 2.19.

Hence as $H \in ILP^s_{LAS}(T)$ if and only if $H \subseteq S_M$, all the positive examples are covered and all the negative examples are covered, verifying that $H \in ILP^s_{LAS}(T)$ can be reduced to checking one problem in NP and another problem in co-NP. This means that verifying a hypothesis is a solution of an ILP^s_{LAS} task is in DP.

$\square$

Proposition 4.20. Deciding verification for ILP_c is DP-hard.

Proof. To prove that verification for ILP_c is DP-hard, we must prove that any problem in DP can be reduced to the verification task. Let D be any arbitrary decision problem which is in DP. By the definition of DP, this is the case if and only if there exist two decision problems D_1 and D_2 such that D_1 is in NP, D_2 is in co-NP and D returns yes if and only if both D_1 and D_2 return yes.

By Lemma 2.19 and Corollary 2.20, this is the case if and only if there are two programs P_1 and P_2 and two atoms $\mathtt{a}_1$ and $\mathtt{a}_2$ such that both $P_1 \models_b \mathtt{a}_1$ and $P_2 \models_c \mathtt{a}_2$ if and only if D returns yes. Without loss of generality we can assume that the atoms in P_1 (together with $\mathtt{a}_1$) are disjoint from the atoms in P_2 (together with $\mathtt{a}_2$).

Take T_c to be the ILP_c task $\langle B, S_M, \langle E^+, E^- \rangle\rangle$, where the individual components of the task are defined as follows:

- $B = P_1 \cup \mathcal{A}(P_2, \mathtt{a}_3) \cup \{\mathtt{:-\ not\ a_1.}\quad \mathtt{0\{a_3\}1.}\quad \mathtt{a_2 :-\ not\ a_3.}\}$ (where we assume $\mathtt{a}_3$ to be a new atom and $\mathcal{A}(P, \mathtt{a})$ to add the atom $\mathtt{a}$ to the body of all rules in P)
- $S_M = \emptyset$
- $E^+ = \{\mathtt{a}_2\}$
- $E^- = \emptyset$

$\emptyset \in ILP_c(T_c)$ if and only if $(P_1 \cup \{\mathtt{:-\ not\ a_1.}\})$ is satisfiable and $\mathcal{A}(P_2, \mathtt{a}_3) \cup \{\mathtt{0\{a_3\}1.}\quad \mathtt{a_2 :-\ not\ a_3.}\} \models_c \mathtt{a}_2$. This is the case as the two subprograms $P_1 \cup \{\mathtt{:-\ not\ a_1.}\}$ and $\mathcal{A}(P_2, \mathtt{a}_3) \cup \{\mathtt{0\{a_3\}1.}\quad \mathtt{a_2 :-\ not\ a_3.}\}$ are disjoint, and the latter is guaranteed to be satisfiable (it has the answer set $\{\mathtt{a}_2\}$). Hence $\emptyset \in ILP_c(T_c)$ if and only if $P_1 \models_b \mathtt{a}_1$. and $P_2 \models_c \mathtt{a}_2$. But this is the case if and only if D returns yes. Hence any problem in DP can be reduced to verifying that a hypothesis is an inductive solution of an ILP_c task.

Hence verification for ILP_c is DP-hard. $\square$

Proposition 4.22. Deciding satisfiability for ILP^s_{LAS} is in Σ^P_2.

Proof. Given an ILP^s_{LAS} task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$, we show that a non-deterministic Turing Machine with access to an NP oracle could check satisfiability of T in polynomial time.

A non-deterministic Turing Machine can have $|S_M|$ choices to make (corresponding to selecting each rule as part of the hypothesis). As verification for ILP^s_{LAS} is in DP (by Proposition 4.19), this hypothesis can then be verified in polynomial time using an NP oracle, with two queries, answering yes if and only if the first query returned yes and the second query returned no.

Such a Turing Machine would terminate answering yes if and only if the task is satisfiable (as there is a path through the Turing Machine which answers yes if and only if there is a hypothesis in S_M which is an inductive solution of the task).

Hence, deciding the existence of a solution for an ILP^s_{LAS} task is in Σ^P_2. $\square$

Proposition 4.23. Deciding satisfiability for ILP_c is Σ^P_2-hard.

Proof. We show this by reducing a known Σ_2^P-complete problem (deciding the existence of an answer set for a ground disjunctive logic program [EG95]) to an ILP_c task.

Take any ground disjunctive logic program P. We will define an ILP_c task $T(P)$ which has a solution if and only if P has an answer set.

Let P' be the program constructed by replacing each negative literal `not a` with the literal `not in_as(a)` (where `in_as` is a new predicate) and replacing each head $\mathtt{h_1} \vee \ldots \vee \mathtt{h_m}$ with $\mathtt{1\{h_1, \ldots, h_m\}m}$ (empty heads are mapped to $\mathtt{1\{\}0}$ – this is equivalent to $\bot$).

We define the learning task $T(P)$ as follows (`not_minimal` is a new atom):

$$B = P' \cup \left\{ \begin{array}{l|l} \mathtt{:\text{-}\ a,\ not\ in\_as(a).} & \\ \mathtt{not\_minimal :\text{-}\ not\ a, in\_as(a).} & \mathtt{a} \in HB_P \end{array} \right\}$$
$$S_M = \{\mathtt{in\_as(a).} \mid \mathtt{a} \in HB_P\}$$
$$E^+ = \emptyset$$
$$E^- = \{\mathtt{not\_minimal}\}$$

This task has a solution if there exists an $H \subseteq S_M$ such that $B \cup H$ is satisfiable and no answer set of $B \cup H$ contains `not_minimal`.

$$\Leftrightarrow \exists H \subseteq S_M \text{ st } \exists A \in AS\left(\left\{ \mathtt{1\{h_1, \ldots, h_m\}m :\text{-}\ b_1, \ldots, b_n} \;\middle|\; \begin{array}{c} \mathtt{1\{h_1, \ldots, h_m\}m :\text{-}\ b_1, \ldots, b_n,} \\ \mathtt{not\ in\_as(c_1), \ldots,\ not\ in\_as(c_o).} \\ \{\mathtt{in\_as(c_1)},\ \ldots,\ \mathtt{in\_as(c_o)}\} \cap H = \emptyset \end{array} \in P', \right\}\right)$$

such that $A \subseteq \{\mathtt{a} \mid \mathtt{in\_as(a)} \in H\}$ and no answer set of $B \cup H$ contains `not_minimal`.

$$\Leftrightarrow \exists H \subseteq S_M \text{ st } \exists A \in AS\left(\left\{ \mathtt{1\{h_1, \ldots, h_m\}m :\text{-}\ b_1, \ldots, b_n} \;\middle|\; \begin{array}{c} \mathtt{1\{h_1, \ldots, h_m\}m :\text{-}\ b_1, \ldots, b_n,} \\ \mathtt{not\ in\_as(c_1), \ldots,\ not\ in\_as(c_o).} \\ \{\mathtt{in\_as(c_1)},\ \ldots,\ \mathtt{in\_as(c_o)}\} \cap H = \emptyset \end{array} \in P', \right\}\right)$$

such that $A = \{\mathtt{a} \mid \mathtt{in\_as(a)} \in H\}$ and there is no strict subset of A which is also an answer set (or there would be an answer set of $B \cup H$ which contains `not_minimal`).

$\Leftrightarrow \exists H \subseteq S_M$ st $\{\mathtt{a} \mid \mathtt{in\_as(a)} \in H\}$ is a minimal model of

$$\left\{ \mathtt{h_1 \vee \ldots \vee h_m :\text{-}\ b_1, \ldots, b_n} \;\middle|\; \begin{array}{c} \mathtt{1\{h_1, \ldots, h_m\}m :\text{-}\ b_1, \ldots, b_n,} \\ \mathtt{not\ in\_as(c_1), \ldots,\ not\ in\_as(c_o).} \\ \{\mathtt{in\_as(c_1)},\ \ldots,\ \mathtt{in\_as(c_o)}\} \cap H = \emptyset \end{array} \in P', \right\}$$

$\Leftrightarrow \exists H \subseteq S_M$ st $\{\mathtt{a} \mid \mathtt{in\_as(a)} \in H\}$ is a minimal model of

$$\left\{ \mathtt{h_1 \vee \ldots \vee h_m :\text{-}\ b_1, \ldots, b_n} \;\middle|\; \begin{array}{c} \mathtt{h_1 \vee \ldots \vee h_m :\text{-}\ b_1, \ldots, b_n,} \\ \mathtt{not\ c_1, \ldots,\ not\ c_o.} \\ \{\mathtt{in\_as(c_1)},\ \ldots,\ \mathtt{in\_as(c_o)}\} \cap H = \emptyset \end{array} \in P, \right\}$$

$\Leftrightarrow \exists A \subseteq HB_P$ st A is a minimal model of

$$\left\{ \mathtt{h_1 \vee \ldots \vee h_m :\text{-}\ b_1, \ldots, b_n} \;\middle|\; \begin{array}{c} \mathtt{h_1 \vee \ldots \vee h_m :\text{-}\ b_1, \ldots, b_n,} \\ \mathtt{not\ c_1, \ldots,\ not\ c_o.} \\ \{\mathtt{c_1},\ \ldots,\ \mathtt{c_o}\} \cap A = \emptyset \end{array} \in P, \right\}$$

$\Leftrightarrow \exists A \subseteq HB_P$ such that A is a minimal model of P^A

$\Leftrightarrow \exists A \subseteq HB_P$ such that A an answer set of P.

$\Leftrightarrow P$ is satisfiable.

Hence, deciding whether a disjunctive logic program is satisfiable can in general be mapped to the decision problem of checking the existence of solutions of a cautious induction task.

Therefore, deciding the existence of solutions of a ground ILP_c task is Σ^P_2-hard.

□

Proposition 4.25. Deciding optimum verification for ILP^s_{LAS} is in Π^P_2.

Proof. Given an ILP^s_{LAS} task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$, we show that a non-deterministic Turing Machine with access to an NP oracle could check whether a hypothesis H is not an optimal solution of T in polynomial time.

H can be verified in polynomial time using an NP oracle, with two queries to the oracle (as verification for ILP^s_{LAS} is in DP, by Proposition 4.19). H is a solution if and only if the first query returns yes and the second no. If the first query returns no, or if the second query returns yes, then H is not an optimal solution (as it is not a solution), and so the Turing machine can return yes.

If the first query returns yes and the second returns no, then H is a solution, so in order to prove that H is not an optimal solution, we must show that there is a shorter hypothesis H' such that H' is a solution of T.

A non-deterministic Turing Machine can then have $|S_M|$ choices to make (corresponding to selecting each rule as part of the hypothesis). This hypothesis H' can then be verified in polynomial time using an NP oracle (by Proposition 4.19), answering yes if and only if the first query returned yes and the second query returned no and $|H'| < |H|$ (which can be checked in polynomial time).

Such a Turing Machine would terminate answering yes if and only if H is not an optimal solution of T (as there is a path through the Turing Machine which answers yes if and only if either H is not a solution of T or if there is a hypothesis H' in S_M such that $|H'| < |H|$ and such that H' is an inductive solution of the task).

Hence, the Turing Machine terminates answering no if and only if H' is an optimal solution of T.

Hence, optimum verification for ILP^s_{LAS} is in Π^P_2. □

Proposition 4.26. Deciding whether an arbitrary hypothesis H is an optimal inductive solution of a given ILP_c task is Π^P_2-hard.

Proof. We show this by reducing a known Π^P_2-complete problem (deciding whether an atom is cautiously entailed by a ground disjunctive logic program [EG95]) to deciding whether a hypothesis is an optimal solution of an ILP_c task.

Take any ground disjunctive logic program P and any atom $\mathtt{a}^*$. We will define an ILP_c task $T(P, \mathtt{a}^*)$ and a hypothesis H such that $H \in {}^*ILP_c(T(P, \mathtt{a}^*))$ if and only if $P \models_c \mathtt{a}^*$.

Let P' be the program constructed by replacing each negative literal `not a` with the literal `not in_as(a)` (where `in_as` is a new predicate) and replacing each head $\mathtt{h_1} \vee \ldots \vee \mathtt{h_m}$ with the counting aggregate $\mathtt{1}\{\mathtt{h_1}, \ldots, \mathtt{h_m}\}\mathtt{m}$ (empty heads are mapped to $\mathtt{1}\{\}\mathtt{0}$ – this is equivalent to $\bot$).

We define the learning task $T(P, \mathtt{a}^*)$ as follows (`not_minimal` and `b` are new atoms):

$$B = \mathcal{A}(P', \texttt{not b}) \cup \left\{ \begin{array}{l|l} \texttt{:- a, not in\_as(a).} & \\ \texttt{:- not n\_in\_as(a), not in\_as(a).} & \\ \texttt{:- n\_in\_as(a), in\_as(a).} & a \in HB_P \\ \texttt{not\_minimal :- not a, in\_as(a), not b.} & \\ \texttt{:- in\_as(a}^*\texttt{), not b.} & \end{array} \right\}$$

$S_M = \{\texttt{in\_as(a)} \mid \texttt{a} \in HB_P\} \cup \{\texttt{n\_in\_as(a)} \mid \texttt{a} \in HB_P\} \cup \{\texttt{b.}\}$
$E^+ = \emptyset$
$E^- = \{\texttt{not\_minimal}\}$

We now show that $\{\texttt{in\_as(a).} \mid \texttt{a} \in HB_P\} \cup \{\texttt{b.}\}$ is an optimal solution of $T(P, \texttt{a}^*)$ if and only if $P \models_c \texttt{a}$.

Assume that $\{\texttt{in\_as(a).} \mid \texttt{a} \in HB_P\} \cup \{\texttt{b.}\}$ is an optimal solution of $T(P, \texttt{a}^*)$

$\Leftrightarrow$ $\{\texttt{in\_as(a).} \mid \texttt{a} \in HB_P\} \cup \{\texttt{b.}\} \in ILP_c(T(P, \texttt{a}^*))$ and $\nexists H \subseteq S_M$ such that $|H| \leq |HB_P|$ and $H \in ILP_c(T(P, \texttt{a}^*))$

$\Leftrightarrow$ $B \cup \{\texttt{in\_as(a).} \mid \texttt{a} \in HB_P\} \cup \{\texttt{b.}\}$ is satisfiable and has no answer set that contains $\texttt{not\_minimal}$ and $\nexists H \subseteq S_M$ such that $|H| \leq |HB_P|$ and $H \in ILP_c(T(P, \texttt{a}^*))$

$\Leftrightarrow$ $\nexists H \subseteq S_M$ such that $|H| \leq |HB_P|$ and $H \in ILP_c(T(P, \texttt{a}^*))$
(as $\{\texttt{in\_as(a)} \mid \texttt{a} \in HB_P\} \cup \{\texttt{b}\} \in AS(B \cup \{\texttt{in\_as(a).} \mid \texttt{a} \in HB_P\} \cup \{\texttt{b.}\})$)

$\Leftrightarrow$ $\forall H \subseteq S_M$ such that $|H| \leq |HB_P|$, and $B \cup H$ is satisfiable, $B \cup H \models_b \texttt{not\_minimal}$

$\Leftrightarrow$ $\forall A \subseteq HB_P$ such that P^* is satisfiable, $P^* \models_b \texttt{not\_minimal}$

$$\text{where} \quad P^* = P' \cup \left\{ \begin{array}{l} \texttt{:- a, not in\_as(a).} \\ \texttt{not\_minimal :- not a, in\_as(a).} \\ \texttt{:- in\_as(a}^*\texttt{).} \end{array} \right\} \begin{array}{l} \cup\{\texttt{in\_as(a).} \mid \texttt{a} \in A\} \\ \cup\{\texttt{n\_in\_as(a).} \mid \texttt{a} \in HB_P \backslash A\} \end{array}$$

$\Leftrightarrow$ $\forall A \subseteq HB_P$ such that $\texttt{a}^* \notin A$ and P^* is satisfiable, $P^* \models_b \texttt{not\_minimal}$

$$\text{where} \quad P^* = P' \cup \left\{ \begin{array}{l} \texttt{:- a, not in\_as(a).} \\ \texttt{not\_minimal :- not a, in\_as(a).} \end{array} \right\} \begin{array}{l} \cup\{\texttt{in\_as(a).} \mid \texttt{a} \in A\} \\ \cup\{\texttt{n\_in\_as(a).} \mid \texttt{a} \in HB_P \backslash A\} \end{array}$$

$\Leftrightarrow$ $\forall A \subseteq HB_P$ such that $\texttt{a}^* \notin A$ and P^* is satisfiable, $P^* \models_b \texttt{not\_minimal}$
where $P^* = P^A \cup \{\texttt{:- a.} \mid \texttt{a} \notin A\} \cup \{\texttt{not\_minimal :- not a.} \mid \texttt{a} \in A\}$ (similar to the proof of Proposition 4.23).

$\Leftrightarrow$ $\forall A \subseteq HB_P$ such that $\texttt{a}^* \notin A$ and $\exists A' \in M(P^A)$ such that $A' \subset A$.

$\Leftrightarrow$ $\forall A \subseteq HB_P$ such that $\texttt{a}^* \notin A$, $A \notin M(P^A)$.

$\Leftrightarrow$ $\forall A \subseteq HB_P$ such that $\texttt{a}^* \notin A$, $A \notin AS(P)$.

$\Leftrightarrow$ $P \models_c \texttt{a}^*$

$\square$

Proofs from Chapter 5

Proposition 5.9.

For any programs P_1 and P_2, $\mathcal{E}_b(P_1) \subseteq \mathcal{E}_b(P_2)$ if and only if $AS(P_1) \subseteq AS(P_2)$.

Proof.

- Assume that $AS(P_1) \subseteq AS(P_2)$

  $\Leftrightarrow \forall A \in AS(P_1),\ A \in AS(P_2)$.

  Assume $c = \mathtt{i_1} \wedge \ldots \wedge \mathtt{i_m}, \wedge\ \mathtt{not\ e_1}, \ldots,\ \mathtt{not\ e_n} \in \mathcal{E}_b(P_1)$. Then there must be an answer set A of P_1 which contains all of the $\mathtt{i}$'s and none of the $\mathtt{e}$'s. Hence, there is also such an answer set of P_2. Hence, $c \in \mathcal{E}_b(P_2)$.

- Conversely, assume that $\mathcal{E}_b(P_1) \subseteq \mathcal{E}_b(P_2)$. Let $A \in AS(P_1)$, we must show that $A \in AS(P_2)$.

  Let $\mathcal{L}$ be the set $HB_{P_1} \cup HB_{P_2}$.

  As $A \in AS(P_1)$, $c = \mathtt{i_1} \wedge \ldots \wedge \mathtt{i_m}, \wedge\ \mathtt{not\ e_1}, \ldots,\ \mathtt{not\ e_n} \in \mathcal{E}_b(P_1)$, where $\{\mathtt{i_1}, \ldots, \mathtt{i_m}\} = A$ and $\{\mathtt{e_1}, \ldots, \mathtt{e_n}\} = \mathcal{L} \backslash A$. As $c \in \mathcal{E}_b(P_1)$, $c \in \mathcal{E}_b(P_2)$ and hence there is an answer set A' of P_2 which contains each $\mathtt{i} \in A$ but no atom $\mathtt{e} \in \mathcal{L} \backslash A$, and hence as $HB_{P_2} \subseteq \mathcal{L}$, $A' = A$. Hence $A \in AS(P_2)$.

$\square$

Proposition 5.10.

$\mathcal{D}^1_1(ILP_b) = \{\langle B, H_1, H_2\rangle |\ AS(B \cup H_1) \not\subseteq AS(B \cup H_2)\}$

Proof. We prove this by showing that $\mathcal{D}^1_1(ILP_b) = \{\langle B, H_1, H_2\rangle | \mathcal{E}_b(B \cup H_1) \not\subseteq \mathcal{E}_b(B \cup H_2)\}$, which is equal to the set $\{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \not\subseteq AS(B \cup H_2)\}$ by Proposition 5.9.

- We first show that if $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_b)$ then there must be a conjunction in $\mathcal{E}_b(B \cup H_1)$ that is not in $\mathcal{E}_b(B \cup H_2)$.

  As $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_b)$, there is an ILP_b task $T = \langle B, \langle\{\mathtt{i_1}, \ldots, \mathtt{i_m}\}, \{\mathtt{e_1}, \ldots, \mathtt{e_n}\}\rangle\rangle$ such that $H_1 \in ILP_b(T)$ and $H_2 \notin ILP_b(T)$.

  Hence, there must be an answer set of $B \cup H_1$ such that $\{\mathtt{i_1}, \ldots, \mathtt{i_m}\} \subseteq A$ and $\{\mathtt{e_1}, \ldots, \mathtt{e_m}\} \cap A = \emptyset$, but no such answer set of $B \cup H_2$.

  Hence the conjunction $c = \mathtt{i_1} \wedge \ldots \wedge \mathtt{i_m} \wedge\ \mathtt{not\ e_1} \wedge \ldots \wedge\ \mathtt{not\ e_n} \in \mathcal{E}_b(B \cup H_1)$ but $c \notin \mathcal{E}_b(B \cup H_2)$.

- Next we show that if there exists a conjunction $c = \mathtt{i_1} \wedge \ldots \wedge \mathtt{i_m} \wedge\ \mathtt{not\ e_1} \wedge \ldots \wedge\ \mathtt{not\ e_n}$ such that $c \in \mathcal{E}_b(B \cup H_1)$ but $c \notin \mathcal{E}_b(B \cup H_2)$, then $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_b)$.

  Assume that there is such a conjunction c. Then $B \cup H_1$ has an answer set that extends $\langle\{\mathtt{i_1}, \ldots, \mathtt{i_m}\}, \{\mathtt{e_1}, \ldots, \mathtt{e_n}\}\rangle$ and $B \cup H_2$ does not. Hence $H_1 \in ILP_b(\langle B, \langle\{\mathtt{i_1}, \ldots, \mathtt{i_m}\}, \{\mathtt{e_1}, \ldots, \mathtt{e_n}\}\rangle\rangle)$ but $H_2 \notin ILP_b(\langle B, \langle\{\mathtt{i_1}, \ldots, \mathtt{i_m}\}, \{\mathtt{e_1}, \ldots, \mathtt{e_n}\}\rangle\rangle)$. So $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_b)$.

$\square$

Proposition 5.11. $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm})$.

Proof.

- First we show that $\mathcal{D}^1_1(ILP_b) \subseteq \mathcal{D}^1_1(ILP_{sm})$. Assume $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_b)$. Then there is a task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H_1 \in ILP_b(T_b)$ and $H_2 \not\in ILP_b(T_b)$. Let $T_{sm} = \langle B, \{\langle E^+, E^-\rangle\}\rangle$. $H_1 \in ILP_{sm}(T_{sm})$ but $H_2 \not\in ILP_{sm}(T_{sm})$. Hence, $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_{sm})$.
- Next we show that $\mathcal{D}^1_1(ILP_b) \supseteq \mathcal{D}^1_1(ILP_{sm})$. Assume $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_{sm})$. There must be a task $T_{sm} = \langle B, \{\langle E^+_1, E^-_1\rangle, \ldots, \langle E^+_n, E^-_n\rangle\}\rangle$ such that $H_1 \in ILP_{sm}(T_{sm})$ and $H_2 \not\in ILP_{sm}(T_{sm})$. There must be at least one partial interpretation $\langle E^+_i, E^-_i\rangle$ such that there is an answer set A of $B \cup H_1$ such that $E^+_i \subseteq A$ and $E^-_i \cap A = \emptyset$ and there is no such answer set of $B \cup H_2$. Hence, letting $T_b = \langle B, \langle E^+_i, E^-_i\rangle\rangle$, $H_1 \in ILP_b(T_b)$ but $H_2 \not\in ILP_b(T_b)$. So $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_b)$.

$\square$

Proposition 5.13.

$$\mathcal{D}^1_1(ILP_c) = \left\{ \langle B, H_1, H_2\rangle \middle| \begin{array}{c} AS(B \cup H_1) \neq \emptyset \wedge \\ (AS(B \cup H_2) = \emptyset \vee \mathcal{E}_c(B \cup H_2) \not\subseteq \mathcal{E}_c(B \cup H_1)) \end{array} \right\}$$

Proof.

- First we show that for any $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_c)$, $AS(B \cup H_1) \neq \emptyset$ and either $AS(B \cup H_2) = \emptyset$ or $\mathcal{E}_c(B \cup H_1) \not\subseteq \mathcal{E}_c(B \cup H_2)$.

  Let $\langle B, H_1, H_2\rangle$ be an arbitrary element of $\mathcal{D}^1_1(ILP_c)$. As $H_1 \in ILP_c(T_c)$, $AS(B \cup H_1) \neq \emptyset$. Assume that $\mathcal{E}_c(B \cup H_1) \subseteq \mathcal{E}_c(B \cup H_2)$. We must show that $AS(B \cup H_2) = \emptyset$. As $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_c)$, $\exists T_c = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H_1 \in ILP_c(T_c)$ and $H_2 \not\in ILP_c(T_c)$.

  As $H_1 \in ILP_c(T_c), \forall A \in AS(B \cup H_1) : E^+ \subseteq A$ and $E^- \cap A \neq \emptyset$, hence the conjunction $E^+ \wedge \{\, \texttt{not}\ e^- \mid e^- \in E^-\} \in \mathcal{E}_c(B \cup H_1)$; hence by our initial assumption that $\mathcal{E}_c(B \cup H_1) \subseteq \mathcal{E}_c(B \cup H_2)$, the conjunction is also in $\mathcal{E}_c(B \cup H_2)$; hence, $\forall A \in AS(B \cup H_2), E^+ \subseteq A$ and $E^- \cap A = \emptyset$. But as $H_2 \not\in ILP_c(T_c)$ this means that $AS(B \cup H_2) = \emptyset$.
- We now show that for any B, H_1 and H_2, if $AS(B \cup H_1) \neq \emptyset \wedge (AS(B \cup H_2) = \emptyset \vee \mathcal{E}_c(B \cup H_1) \not\subseteq \mathcal{E}_c(B \cup H_2)$, then $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_c)$.

  Case 1: $AS(B \cup H_1) \neq \emptyset \wedge AS(B \cup H_2) = \emptyset$.

  Consider the task $T_c = \langle B, \langle \emptyset, \emptyset\rangle\rangle$. $H_1 \in ILP_c(T_c)$ as $AS(B \cup H_1) \neq \emptyset$ and $\forall A \in AS(B \cup H_1), \emptyset \subseteq A$ and $A \cap \emptyset = \emptyset$. $H_2 \not\in ILP_c(T_c)$ as $AS(B \cup H_2) = \emptyset$.

  Case 2: $AS(B \cup H_1) \neq \emptyset \wedge \exists c = (\texttt{i}_1 \wedge \ldots \wedge \texttt{i}_\texttt{m} \wedge\ \texttt{not}\ \texttt{e}_1 \wedge \ldots \wedge \texttt{e}_\texttt{n}) \in \mathcal{E}_c(B \cup H_2)$ such that $c \not\in \mathcal{E}_c(B \cup H_1)$.

  Consider the task $T_c = \langle B, \langle \{\texttt{i}_1, \ldots, \texttt{i}_\texttt{m}\}, \{\texttt{e}_1, \ldots, \texttt{e}_\texttt{n}\}\rangle\rangle$. $H_1 \in ILP_c(T_c)$, but $H_2 \not\in ILP_c(T_c)$.

$\square$

Proposition 5.14. $\mathcal{D}^1_1(ILP_{LAS}) = \{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \neq AS(B \cup H_2)\}$

Proof.

- We first show that $\mathcal{D}^1_1(ILP_{LAS}) \subseteq \{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \neq AS(B \cup H_2)\}$. For any $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_{LAS})$ there is an ILP_{LAS} task $T = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H_1 \in ILP_{LAS}(T)$ and $H_2 \notin ILP_{LAS}(T)$.

  Case 1: $\exists e \in E^+$ such that $\forall A \in AS(B \cup H_2)$, A does not extend e.

  As $H_1 \in ILP_{LAS}(T), \exists A' \in AS(B \cup H_1)$ such that A' extends e. Hence as A' cannot be in $AS(B \cup H_2)$, $AS(B \cup H_1) \neq AS(B \cup H_2)$.

  Case 2: $\exists e \in E^-$, $\exists A \in AS(B \cup H_2)$ such that A extends e.

  As $H_1 \in ILP_{LAS}(T), \forall A' \in AS(B \cup H_1)$, A' does not extend e. Hence A cannot be in $AS(B \cup H_1)$ and so $AS(B \cup H_1) \neq AS(B \cup H_2)$.

- It remains to show that $\mathcal{D}^1_1(ILP_{LAS}) \supseteq \{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \neq AS(B \cup H_2)\}$. Take B, H_1, H_2 to be any ASP programs such that $AS(B \cup H_1) \neq AS(B \cup H_2)$

  Let L be the set of atoms which appear in answer sets of $B \cup H_1$ and $B \cup H_2$.

  Case 1: $\exists A \in AS(B \cup H_1)$ such that $A \notin AS(B \cup H_2)$

  Let $e_A = \langle A, L\backslash A\rangle$. A is the only interpretation in $AS(B \cup H_1)$ or $AS(B \cup H_2)$ which extends e_A (as e_A is completely defined over the atoms in L). Hence, there is an answer set of $B \cup H_1$ which extends e_A, but no such answer set of $B \cup H_2$.

  Hence, $H_1 \in ILP_{LAS}(\langle B, \langle\{e_A\}, \emptyset\rangle\rangle)$, but $H_2 \notin ILP_{LAS}(\langle B, \langle\{e_A\}, \emptyset\rangle\rangle)$.

  Hence $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_{LAS})$.

  Case 2: $\exists A \in AS(B \cup H_2)$ such that $A \notin AS(B \cup H_1)$

  Let $e_A = \langle A, L\backslash A\rangle$. A is the only interpretation in $AS(B \cup H_1)$ or $AS(B \cup H_2)$ which extends e_A. Hence, there is no answer set of $B \cup H_1$ which extends e_A, but there is such an answer set of $B \cup H_2$.

  Hence, $H_1 \in ILP_{LAS}(\langle B, \langle\emptyset, \{e_A\}\rangle\rangle)$, but $H_2 \notin ILP_{LAS}(\langle B, \langle\emptyset, \{e_A\}\rangle\rangle)$.

  Hence $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_{LAS})$.

□

Proposition 5.15.

$$\mathcal{D}^1_1(ILP_{LOAS}) = \left\{ \langle B, H_1, H_2\rangle \;\middle|\; \begin{array}{l} AS(B \cup H_1) \neq AS(B \cup H_2) \text{ or} \\ ord(B \cup H_1) \neq ord(B \cup H_2) \end{array} \right\}$$

Proof.

- We first show that $\mathcal{D}^1_1(ILP_{LOAS}) \subseteq \left\{ \langle B, H_1, H_2\rangle \;\middle|\; \begin{array}{c} AS(B \cup H_1) \neq AS(B \cup H_2) \text{ or} \\ ord(B \cup H_1) \neq ord(B \cup H_2) \end{array} \right\}$.

  For any $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_{LOAS})$ there is an ILP_{LOAS} task $T = \langle B, \langle E^+, E^-, O^b, O^c\rangle\rangle$ such that $H_1 \in ILP_{LOAS}(T)$ and $H_2 \notin ILP_{LOAS}(T)$.

  Case 1: $\exists e \in E^+$ such that $\forall A \in AS(B \cup H_2)$, A does not extend e.

  As $H_1 \in ILP_{LOAS}(T), \exists A' \in AS(B \cup H_1)$ such that A' extends e. Hence as A' cannot be in $AS(B \cup H_2)$, $AS(B \cup H_1) \neq AS(B \cup H_2)$.

Case 2: $\exists e \in E^-$, $\exists A \in AS(B \cup H_2)$ such that A extends e.

As $H_1 \in ILP_{LOAS}(T), \forall A' \in AS(B \cup H_1)$, A' does not extend e. Hence, A cannot be in $AS(B \cup H_1)$ and so $AS(B \cup H_1) \neq AS(B \cup H_2)$.

Case 3: $\exists \langle e_1, e_2, op \rangle \in O^b$ which is covered by H_1 but not H_2.

Assume $AS(B \cup H_1) = AS(B \cup H_2)$. $\exists A_1, A_2 \in AS(B \cup H_1)$ such that A_1 extends e_1, A_2 extends e_2 and $\langle A_1, A_2, op \rangle \in ord(B \cup H)$ as H_1 covers the ordering example. $\langle A_1, A_2, op \rangle \notin ord(B \cup H_2)$ as H_2 does not cover the ordering example; and hence, $ord(B \cup H_1) \neq ord(B \cup H_2)$.

Case 4: $\exists \langle e_1, e_2, op \rangle \in O^c$ which is covered by H_1 but not H_2.

Assume $AS(B \cup H_1) = AS(B \cup H_2)$. $\exists A_1, A_2 \in AS(B \cup H_2)$ such that A_1 extends e_1, A_2 extends e_2 and $\langle A_1, A_2, op \rangle \notin ord(B \cup H_2)$ as H_2 does not cover the ordering example. $\langle A_1, A_2, op \rangle \in ord(B \cup H_1)$ as H_1 does cover the ordering example; and hence, $ord(B \cup H_1) \neq ord(B \cup H_2)$.

Hence, in all cases, either $AS(B \cup H_1) \neq AS(B \cup H_2$ or $ord(B \cup H_1) \neq ord(B \cup H_2)$.

- It remains to show that $\mathcal{D}^1_1(ILP_{LOAS}) \supseteq \left\{ \langle B, H_1, H_2 \rangle \middle| \begin{array}{c} AS(B \cup H_1) \neq AS(B \cup H_2) \text{ or} \\ ord(B \cup H_1) \neq ord(B \cup H_2) \end{array} \right\}$. Take B, H_1, H_2 to be any ASP programs such that $AS(B \cup H_1) \neq AS(B \cup H_2)$ or $ord(B \cup H_1) \neq ord(B \cup H_2)$.

  Let L be the set of literals which appear in answer sets of $B \cup H_1$ and $B \cup H_2$.

  Case 1: $AS(B \cup H_1) \neq AS(B \cup H_2)$

  $\langle B, H_1, H_2 \rangle \in \mathcal{D}^1_1(ILP_{LAS})$ (by Proposition 5.14). Hence, there is an ILP_{LAS} task $T_{LAS} = \langle B, \langle E^+, E^- \rangle \rangle$ such that $H_1 \in ILP_{LAS}(T_{LAS})$ and $H_2 \notin ILP_{LAS}(T_{LAS})$. Let $T_{LOAS} = \langle B, \langle E^+, E^-, \emptyset, \emptyset \rangle \rangle$. $H_1 \in ILP_{LOAS}(T_{LOAS})$ and $H_2 \notin ILP_{LOAS}(T_{LOAS})$.

  Case 2: $AS(B \cup H_1) = AS(B \cup H_2)$ but $ord(B \cup H_1) \neq ord(B \cup H_2)$

  $\exists A_1, A_2 \in AS(B \cup H_1)$ (which is equal to $AS(B \cup H_2)$) such that there is a binary operator op such that $\langle A_1, A_2, op \rangle \in ord(B \cup H_1)$ but $\langle A_1, A_2, op \rangle \notin ord(B \cup H_2)$. Let $e_1 = \langle A_1, L \backslash A_1 \rangle$ and $e_2 = \langle A_2, L \backslash A_2 \rangle$ (where L is the set of atoms in the answer sets of $B \cup H_1$). Consider the ILP_{LOAS} task $T_{LOAS} = \langle B, \langle \{e_1, e_2\}, \emptyset, \{\langle e_1, e_2, op \rangle\}, \emptyset \rangle \rangle$. $H_1 \in ILP_{LOAS}(T_{LOAS})$ and $H_2 \notin ILP_{LOAS}(T_{LOAS})$.

  Hence, in both cases $\langle B, H_1, H_2 \rangle \in \mathcal{D}^1_1(ILP_{LOAS})$.

□

Proposition 5.17.

$$\mathcal{D}^1_1(ILP^{context}_{LOAS}) = \left\{ \langle B, H_1, H_2 \rangle \middle| \begin{array}{c} B \cup H_1 \not\equiv^s B \cup H_2 \text{ or} \\ \exists C \in \mathcal{ASP}^{ch} \textit{ such that } ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C) \end{array} \right\}$$

Proof.

- We first show $\mathcal{D}^1_1(ILP^{context}_{LOAS}) \subseteq \left\{ \langle B, H_1, H_2 \rangle \middle| \begin{array}{c} B \cup H_1 \not\equiv^s B \cup H_2 \text{ or} \\ \exists C \in \mathcal{ASP}^{ch} \text{ st } ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C) \end{array} \right\}$.

  For any $\langle B, H_1, H_2 \rangle \in \mathcal{D}^1_1(ILP^{context}_{LOAS})$ there is an $ILP^{context}_{LOAS}$ task $T = \langle B, \langle E^+, E^-, O^b, O^c \rangle \rangle$ such that $H_1 \in ILP^{context}_{LOAS}(T)$ and $H_2 \notin ILP^{context}_{LOAS}(T)$.

Case 1: $\exists\langle e, C\rangle \in E^{+}$ such that $\forall A \in AS(B \cup H_2 \cup C)$, A does not extend e.

As $H_1 \in ILP^{context}_{LOAS}(T), \exists A' \in AS(B \cup H_1 \cup C)$ such that A' extends e. Hence, as A' cannot be in $AS(B \cup H_2 \cup C)$, $AS(B \cup H_1 \cup C) \neq AS(B \cup H_2 \cup C)$. Hence, $B \cup H_1 \not\equiv_s B \cup H_2$.

Case 2: $\exists\langle e, C\rangle \in E^{-}$, $\exists A \in AS(B \cup H_2 \cup C)$ such that A extends e.

As $H_1 \in ILP^{context}_{LOAS}(T), \forall A' \in AS(B \cup H_1 \cup C)$, A' does not extend e. Hence, A cannot be in $AS(B \cup H_1 \cup C)$ and so $AS(B \cup H_1 \cup C) \neq AS(B \cup H_2 \cup C)$. Hence, $B \cup H_1 \not\equiv_s B \cup H_2$.

Case 3: $\exists\langle\langle e_1, C_1\rangle, \langle e_2, C_2\rangle, op\rangle \in O^{b}$ which is covered by H_1 but not H_2.

Assume that $B \cup H_1 \equiv_s B \cup H_2$.

Let S be the set $AS(B \cup H_1 \cup C_1) \cup AS(B \cup H_1 \cup C_2)$ (which is equal to the set $AS(B \cup H_2 \cup C_1) \cup AS(B \cup H_2 \cup C_2)$ as $B \cup H_1 \equiv_s B \cup H_2$). $\exists A_1 \in AS(B \cup H_1 \cup C_1)$, $A_2 \in AS(B \cup H_1 \cup C_2)$ such that A_1 extends e_1, A_2 extends e_2 and $\langle A_1, A_2, op\rangle \in ord(B \cup H_1, S)$ as H_1 covers the ordering example. $\langle A_1, A_2, op\rangle \notin ord(B \cup H_2, S)$ as H_2 does not cover the ordering example.

Let C be the $\mathcal{ASP}^{ch}$ program $append(C_1, \mathtt{a_1}) \cup append(C_2, \mathtt{a_2}) \cup \{\mathtt{1\{a_1, a_2\}1.}\}$ (where $\mathtt{a_1}$ and $\mathtt{a_2}$ are new atoms and $append(P, \mathtt{a})$ appends the atom $\mathtt{a}$ to the body of each rule in P). $AS(B \cup H_1 \cup C) = \{A \cup \{\mathtt{a_1}\} \mid A \in AS(B \cup H_1 \cup C_1)\} \cup \{A \cup \{\mathtt{a_2}\} \mid A \in AS(B \cup H_1 \cup C_2)\}$, and hence, $t = \langle A_1 \cup \{\mathtt{a_1}\}, A_2 \cup \{\mathtt{a_2}\}, op\rangle \in ord(B \cup H_1 \cup C)$, but $t \notin ord(B \cup H_2 \cup C)$.

Hence, $\exists C \in \mathcal{ASP}^{ch}$ such that $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C)$.

Case 4: $\exists\langle e_1, e_2, op\rangle \in O^{c}$ which is covered by H_1 but not H_2.

Assume that $B \cup H_1 \equiv_s B \cup H_2$.

Let S be the set $AS(B \cup H_1 \cup C_1) \cup AS(B \cup H_1 \cup C_2)$ (which is equal to the set $AS(B \cup H_2 \cup C_1) \cup AS(B \cup H_2 \cup C_2)$ as $B \cup H_1 \equiv_s B \cup H_2$). $\exists A_1 \in AS(B \cup H_1 \cup C_1)$, $A_2 \in AS(B \cup H_1 \cup C_2)$ such that A_1 extends e_1, A_2 extends e_2 and $\langle A_1, A_2, op\rangle \notin ord(B \cup H_2, S)$ as H_2 does not cover the ordering example. $\langle A_1, A_2, op\rangle \in ord(B \cup H_1, S)$ as H_1 does cover the ordering example.

Let C be the $\mathcal{ASP}^{ch}$ program $append(C_1, \mathtt{a_1}) \cup append(C_2, \mathtt{a_2}) \cup \{\mathtt{1\{a_1, a_2\}1.}\}$ (where $\mathtt{a_1}$ and $\mathtt{a_2}$ are new atoms and $append(P, \mathtt{a})$ appends the atom $\mathtt{a}$ to the body of each rule in P). $AS(B \cup H_1 \cup C) = \{A \cup \{\mathtt{a_1}\} \mid A \in AS(B \cup H_1 \cup C_1)\} \cup \{A \cup \{\mathtt{a_2}\} \mid A \in AS(B \cup H_1 \cup C_2)\}$, and hence, $t = \langle A_1 \cup \{\mathtt{a_1}\}, A_2 \cup \{\mathtt{a_2}\}, op\rangle \in ord(B \cup H_1 \cup C)$, but $t \notin ord(B \cup H_2 \cup C)$.

Hence, $\exists C \in \mathcal{ASP}^{ch}$ such that $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C)$.

Hence, in all cases, either $B \cup H_1 \equiv_s B \cup H_2$ or $\exists C \in \mathcal{ASP}^{ch}$ such that $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C)$.

- It remains to show that:

$$\mathcal{D}^{1}_{1}(ILP^{context}_{LOAS}) \supseteq \left\{ \langle B, H_1, H_2\rangle \;\middle|\; \begin{array}{c} B \cup H_1 \not\equiv^{s} B \cup H_2 \text{ or} \\ \exists C \in \mathcal{ASP}^{ch} \text{ st } ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C) \end{array} \right\}.$$

Take B, H_1, H_2 to be any ASP programs such that $B \cup H_1 \not\equiv^{s} B \cup H_2$ or $\exists C \in \mathcal{ASP}^{ch}$ st $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C)$.

Case 1: $B \cup H_1 \not\equiv_s B \cup H_2$

There must be a program C such that $AS(B \cup H_1 \cup C) \neq AS(B \cup H_2 \cup C)$.

Case i: $\exists A \in AS(B \cup H_1 \cup C)$ such that $A \notin AS(B \cup H_2 \cup C)$.

Let L be the set of atoms in the answer sets of $B \cup H_1 \cup C$ and $B \cup H_2 \cup C$ and let e_A be the partial interpretation $\langle A, L\backslash A\rangle$. Then $B \cup H_1 \cup C$ has an answer set that extends e_A, but $B \cup H_2 \cup C$ does not, and hence, $H_1 \in ILP^{context}_{LOAS}(\langle B, S_M, \langle\{\langle e_A, C\rangle\}, \emptyset, \emptyset, \emptyset\rangle\rangle)$ but H_2 is not.

Case ii: $\exists A \in AS(B \cup H_2 \cup C)$ such that $A \not\in AS(B \cup H_1 \cup C)$.

Let L be the set of atoms in the answer sets of $B \cup H_1 \cup C$ and $B \cup H_2 \cup C$ and let e_A be the partial interpretation $\langle A, L\backslash A\rangle$. Then $B \cup H_2 \cup C$ has an answer set that extends e_A, but $B \cup H_1 \cup C$ does not, and hence, $H_1 \in ILP^{context}_{LOAS}(\langle B, S_M, \langle \emptyset, \{\langle e_A, C\rangle\}, \emptyset, \emptyset\rangle\rangle)$ but H_2 is not.

Case 2: $B \cup H_1 \equiv_s B \cup H_2$ but $\exists C \in \mathcal{ASP}^{ch}$ such that $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C)$

$\exists A_1, A_2 \in AS(B \cup H_1 \cup C)$ (which is equal to $AS(B \cup H_2 \cup C)$) such that there is a binary operator op such that $\langle A_1, A_2, op\rangle \in ord(B \cup H_1 \cup C)$ but $\langle A_1, A_2, op\rangle \not\in ord(B \cup H_2 \cup C)$. Let $e_1 = \langle\langle A_1, L\backslash A_1\rangle, C\rangle$ and $e_2 = \langle\langle A_2, L\backslash A_2\rangle, C\rangle$ (where L is the set of atoms in the answer sets of $B \cup H_1 \cup C$. Consider the $ILP^{context}_{LOAS}$ task $T^{context}_{LOAS} = \langle B, \langle\{e_1, e_2\}, \emptyset, \{\langle e_1, e_2, op\rangle\}, \emptyset\rangle\rangle$. $H_1 \in ILP^{context}_{LOAS}(T^{context}_{LOAS})$ and $H_2 \not\in ILP^{context}_{LOAS}(T^{context}_{LOAS})$.

Hence, in both cases $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP^{context}_{LOAS})$.

□

Proposition 5.22. For any framework $\mathcal{F}$:

$$\overline{\mathcal{D}^1_m(\mathcal{F})} = \left\{ \langle B, H, \{H_1, \ldots, H_n\}\rangle \;\middle|\; \begin{array}{c} \langle B, H, H_1\rangle \in \mathcal{D}^1_1(\mathcal{F}), \\ \ldots, \\ \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F}) \end{array} \right\}$$

Proof.

- We first show that $\overline{\mathcal{D}^1_m(\mathcal{F})} \subseteq \{\langle B, H, \{H_1, \ldots, H_n\}\rangle | \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\}$. Take an arbitrary $\langle B, H, S\rangle \in \overline{\mathcal{D}^1_m(\mathcal{F})}$. We must show that $\langle B, H, S\rangle \in \{\langle B, H, \{H_1, \ldots, H_n\}\rangle | \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\}$. To do this, we need to show that $\forall H' \in S$, $\langle B, H, H'\rangle \in \mathcal{D}^1_1(\mathcal{F})$.

  Take an arbitrary $H' \in S$. It remains to show that $\langle B, H, H'\rangle \in \mathcal{D}^1_1(\mathcal{F})$. By definition of $\overline{\mathcal{D}^1_m(\mathcal{F})}$, there must be some subset $S' \subseteq S$ such that $H' \in S'$ and $\langle B, H, S'\rangle \in \mathcal{D}^1_m(\mathcal{F})$. Hence, $\exists T_{\mathcal{F}}$ such that $H \in ILP_{\mathcal{F}}(T_{\mathcal{F}})$ and $S' \cap ILP_{\mathcal{F}}(T_{\mathcal{F}}) = \emptyset$. Hence, as $H' \in S'$, $\langle B, H, H'\rangle \in \mathcal{D}^1_1(\mathcal{F})$.

- We now show that $\overline{\mathcal{D}^1_m(\mathcal{F})} \supseteq \{\langle B, H, \{H_1, \ldots, H_n\}\rangle | \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\}$. Take an arbitrary $\langle B, H, \{H_1, \ldots, H_n\}\rangle \in \{\langle B, H, \{H_1, \ldots, H_n\}\rangle | \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\}$. For each $i \in [1..n]$, $\langle B, H, H_i\rangle \in \mathcal{D}^1_1(\mathcal{F})$, and hence, $\langle B, H, \{H_i\}\rangle \in \mathcal{D}^1_m(\mathcal{F})$. Hence, by definition of $\overline{\mathcal{D}^1_m(\mathcal{F})}$, $\langle B, H, \{H_1, \ldots, H_n\}\rangle \in \overline{\mathcal{D}^1_m(\mathcal{F})}$.

□

Proposition 5.27. *ILP_c, ILP_{sm}, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$ all have closed one-to-many-distinguishability.*

Proof.

1. Consider any two ILP_c tasks, $T_c^1 = \langle B, \langle E_1^+, E_1^- \rangle\rangle$ and $T_c^2 = \langle B, \langle E_2^+, E_2^- \rangle\rangle$. Let $T_c^3 = \langle B, \langle E_1^+ \cup E_2^+, E_1^- \cup E_2^- \rangle\rangle$.

   $H \in ILP_c(T_c^1) \cap ILP_c(T_c^2)$ if and only if $AS(B \cup H)$ is non-empty and $\forall A \in AS(B \cup H)$: $E_1^+ \subseteq A$, $E_2^+ \subseteq A$, $E_1^- \cap A = \emptyset$ and $E_2^- \cap A = \emptyset$. This is the case if and only if $(E_1^+ \cup E_2^+) \subseteq A$, and $(E_1^- \cup E_2^-) \cap A = \emptyset$ which holds if an only if $H \in ILP_c(T_c^3)$.

   Hence, by Lemma 5.26, ILP_c has closed one-to-many-distinguishability.

2. For any tasks $T_{sm}^1 = \langle B, \langle \{e_1^1, \ldots, e_n^1\}\rangle\rangle$ and $T_{sm}^2 = \langle B, \langle \{e_1^2, \ldots, e_m^2\}\rangle\rangle$, let $T_{sm}^3 = \langle B, \langle \{e_1^1, \ldots, e_n^1, e_1^2, \ldots, e_m^2\}\rangle\rangle$. $ILP_{sm}(T_{sm}^3) = ILP_{sm}(T_{sm}^1) \cap ILP_{sm}(T_{sm}^2)$.

   Hence, by Lemma 5.26, ILP_{sm} has closed one-to-many-distinguishability.

3. For any tasks $T_{LAS}^1 = \langle B, \langle E_1^+, E_1^- \rangle\rangle$ and $T_{LAS}^2 = \langle B, \langle E_2^+, E_2^- \rangle\rangle$, let $T_{LAS}^3 = \langle B, \langle E_1^+ \cup E_2^+, E_1^- \cup E_2^- \rangle\rangle$. $ILP_{LAS}(T_{LAS}^3) = ILP_{LAS}(T_{LAS}^1) \cap ILP_{LAS}(T_{LAS}^2)$.

   Hence, by Lemma 5.26, ILP_{LAS} has closed one-to-many-distinguishability.

4. For any tasks $T_{LOAS}^1 = \langle B, \langle E_1^+, E_1^-, O_1^b, O_1^c \rangle\rangle$ and $T_{LOAS}^2 = \langle B, \langle E_2^+, E_2^-, O_2^b, O_2^c \rangle\rangle$, let $T_{LOAS}^3 = \langle B, \langle E_1^+ \cup E_2^+, E_1^- \cup E_2^-, O_1^b \cup O_2^b, O_1^c \cup O_2^c \rangle\rangle$. $ILP_{LOAS}(T_{LOAS}^3) = ILP_{LOAS}(T_{LOAS}^1) \cap ILP_{LOAS}(T_{LOAS}^2)$.

   Hence, by Lemma 5.26, ILP_{LOAS} has closed one-to-many-distinguishability.

5. For any tasks $T1_{LOAS}^{context} = \langle B, \langle E_1^+, E_1^-, O_1^b, O_1^c \rangle\rangle$ and $T2_{LOAS}^{context} = \langle B, \langle E_2^+, E_2^-, O_2^b, O_2^c \rangle\rangle$, let $T3_{LOAS}^{context} = \langle B, \langle E_1^+ \cup E_2^+, E_1^- \cup E_2^-, O_1^b \cup O_2^b, O_1^c \cup O_2^c \rangle\rangle$.

   $ILP_{LOAS}^{context}(T3_{LOAS}^{context}) = ILP_{LOAS}^{context}(T1_{LOAS}^{context}) \cap ILP_{LOAS}^{context}(T2_{LOAS}^{context})$.

   Hence, by Lemma 5.26, $ILP_{LOAS}^{context}$ has closed one-to-many-distinguishability.

□

Proofs from Chapter 6

Theorem 6.3. Let T be the $ILP_{LOAS}^{context}$ task $\langle B, S_M, \langle E^+, E^-, O^b, O^c \rangle\rangle$, $AS_{ids} = \{\mathtt{t_1}, \ldots, \mathtt{t_n}\}$ be a set of terms and let $H \subseteq S_M$. Consider the program $P = \mathcal{M}_1(T) \cup \{\mathtt{as(t).} \mid \mathtt{t} \in AS_{ids}\} \cup \{\mathtt{in\_h(h_{id}).} \mid h \in H\}$. For any list $[\langle I^1, e^1 \rangle, \ldots, \langle I^n, e^n \rangle]$ (of length $|AS_{ids}|$), where each I^i is an interpretation and each e^i is selected from $E^+ \cup E^-$, $\exists A \in AS(P)$ such that $\forall i \in [1, n]$, $\mathtt{ctx(e^i_{id}, t_i)} \in A$ and $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i}) = I^i$ if and only if $\forall i \in [1, n]$, $I^i \in AS(B \cup H \cup e^i_{ctx})$.

Proof. Let $[\langle I^1, e^1 \rangle, \ldots, \langle I^n, e^n \rangle]$ be a list of pairs, where each I^i is an interpretation and each e^i is an example in $E^+ \cup E^-$.

Assume $\exists A \in AS(P)$ such that $\forall i \in [1, n]$, $\mathtt{ctx(e^i_{id}, t_i)} \in A$ and $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i}) = I_i$

$\Leftrightarrow$ there is an answer set A of the program:

$\mathcal{A}(\mathcal{R}(non\_weak(B \cup H), \mathtt{in\_as}, \mathtt{AS\_ID}), \mathtt{as(AS\_ID)})$

$\quad \cup \{\mathcal{A}(\mathcal{A}(\mathcal{R}(e_{ctx}, \mathtt{in\_as}, \mathtt{AS\_ID}), \mathtt{as(AS\_ID)}), \mathtt{ctx(e_{id}, AS\_ID)}) | e \in E^+ \cup E^-\}$

$\quad \cup \{\mathtt{1\{ctx(ex^1_{id}, AS\_ID), \ldots, ctx(ex^m_{id}, AS\_ID)\}1 \text{:-} as(AS\_ID).}\}$

$\quad \cup \{\mathtt{as(t).} \mid \mathtt{t} \in AS_{ids}\}$

such that $\forall i \in [1, n]$, $\mathtt{ctx(e^i_{id}, t_i)} \in A$ and $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i}) = I^i$

(where $\{ex^1, \ldots, ex^m\} = E^+ \cup E^-$)
by the splitting set theorem (using the $\texttt{in\_h}$ atoms as a splitting set).

$\Leftrightarrow \forall i \in [1, n]$, there is an answer set A of the program:
$\mathcal{A}(\mathcal{R}(non\_weak(B \cup H), \texttt{in\_as}, \texttt{t}_\texttt{i}), \texttt{as}(\texttt{t}_\texttt{i}))$
$\cup \{\mathcal{A}(\mathcal{A}(\mathcal{R}(e_{ctx}, \texttt{in\_as}, \texttt{t}_\texttt{i}), \texttt{as}(\texttt{t}_\texttt{i})), \texttt{ctx}(\texttt{e}_\texttt{id}, \texttt{t}_\texttt{i})) | e \in E^+ \cup E^-\}$
$\cup \{\texttt{1}\{\texttt{ctx}(\texttt{ex}^\texttt{1}_\texttt{id}, \texttt{t}_\texttt{i}), \ldots, \texttt{ctx}(\texttt{ex}^\texttt{m}_\texttt{id}, \texttt{t}_\texttt{i})\}\texttt{1} \texttt{:-} \texttt{as}(\texttt{t}_\texttt{i}).\}$
$\cup \{\texttt{as}(\texttt{t}_\texttt{i}).\}$
such that $\texttt{ctx}(\texttt{e}^\texttt{i}_\texttt{id}, \texttt{t}_\texttt{i}) \in A$ and $\mathcal{M}^{-1}_{as}(A, \texttt{t}_\texttt{i}) = I^i$

$\Leftrightarrow \forall i \in [1, n]$, there is an $A \in AS(\mathcal{R}(non\_weak(B \cup H \cup e^i_{ctx}), \texttt{in\_as}, \texttt{t}_\texttt{i}))$ such that $\mathcal{M}^{-1}_{as}(A, \texttt{t}_\texttt{i}) = I^i$
by the splitting set theorem (using the $\texttt{ctx}$ and $\texttt{as}$ atoms as a splitting set).

$\Leftrightarrow \forall i \in [1, n]$, $I^i \in AS(non\_weak(B \cup H \cup e^i_{ctx}))$

$\Leftrightarrow \forall i \in [1, n]$, $I^i \in AS(B \cup H \cup e^i_{ctx})$

□

Theorem 6.4. Let T be an $ILP^{context}_{LOAS}$ task with background knowledge B and hypothesis space S_M, and let $H \subseteq S_M$. Let $AS_{ids} = \{\texttt{t}_\texttt{1}, \ldots, \texttt{t}_\texttt{n}\}$ be a set of terms. For each $\texttt{t} \in AS_{ids}$ let $E_\texttt{t}$ be a set of CDPIs.

$$\text{Consider the program } P = \mathcal{M}_1(T) \begin{array}{ll} \cup & \{\texttt{in\_h}(\texttt{h}_\texttt{id}). \mid h \in H\} \\ \cup & \{\texttt{as}(\texttt{t}). \mid \texttt{t} \in AS_{ids}\} \\ \cup & \{check(e, \texttt{t}) \mid \texttt{t} \in AS_{ids}, e \in E_\texttt{t}\} \end{array}.$$

1. For any list $[\langle I^1, e^1\rangle, \ldots, \langle I^n, e^n\rangle]$ (of length $|AS_{ids}|$) such that each e^i is selected from $E^+ \cup E^-$ and each I^i is an interpretation: $\exists A \in AS(P)$ such that $\forall i \in [1, n]$, $\texttt{ctx}(\texttt{e}^\texttt{i}_\texttt{id}, \texttt{t}_\texttt{i}) \in A$ and $\mathcal{M}^{-1}_{as}(A, \texttt{t}_\texttt{i}) = I^i$ if and only if $\forall i \in [1, n]$, $I^i \in AS(B \cup H \cup e^i_{ctx})$.

2. For any answer set $A \in AS(P)$, $\forall i \in [1, n]$, $\forall e \in E^+ \cup E^-$, $\texttt{cov}(\texttt{e}_\texttt{id}, \texttt{t}_\texttt{i}) \in A$ if and only if $\texttt{ctx}(\texttt{e}_\texttt{id}, \texttt{t}_\texttt{i}) \in A$, $e \in E_{\texttt{t}_\texttt{i}}$ and $\mathcal{M}^{-1}_{as}(A, \texttt{t}_\texttt{i})$ is an accepting answer set of e wrt $B \cup H$.

Proof. Note that the set of atoms in HB_P that do not have the predicate name $\texttt{cov}$ form a splitting set of P. Hence, we can use the splitting set theorem, using the answer sets of $P' = \mathcal{M}_1(T) \cup \{\texttt{in\_h}(\texttt{h}_\texttt{id}). \mid h \in H\} \cup \{\texttt{as}(\texttt{t}). \mid \texttt{t} \in AS_{ids}\}$ to compute the answer sets of P. Specifically, $AS(P) = \{A \mid A' \in AS(P'), A \in AS(\{\texttt{a}. \mid \texttt{a} \in A'\} \cup \{check(e, \texttt{t}) \mid \texttt{t} \in AS_{ids}, e \in E_\texttt{t}\})\}$.

1. Consider a list $[\langle I^1, e^1\rangle, \ldots, \langle I^n, e^n\rangle]$ and such that $\forall i \in [1, n]$, $e^i \in E^+ \cup E^-$ and each I^i is an interpretation.

   Assume $\exists A \in AS(P)$ such that $\forall i \in [1, n]$, $\texttt{ctx}(\texttt{e}^\texttt{i}_\texttt{id}, \texttt{t}_\texttt{i}) \in A$ and $\mathcal{M}^{-1}_{as}(A, \texttt{t}_\texttt{i}) = I^i$

   $\Leftrightarrow \exists A' \in AS(P')$ st $\forall i \in [1, n]$, $\texttt{ctx}(\texttt{e}^\texttt{i}_\texttt{id}, \texttt{t}_\texttt{i}) \in A$ and $\mathcal{M}^{-1}_{as}(A', \texttt{t}_\texttt{i}) = I^i$. This can be seen using the Splitting Set Theorem, with all but the $\texttt{cov}$ atoms in the splitting set.

   $\Leftrightarrow \forall i \in [1, n]$, $I^i \in AS(B \cup H \cup e^i_{ctx})$ (by Theorem 6.3)

2. Let $A \in AS(P)$, $i \in [1, n]$ and $e \in E^{+} \cup E^{-}$.

   Assume that $\mathtt{cov(e_{id}, t_i)} \in A$

   $\Leftrightarrow$ there is a rule in $ground(P)$ with $\mathtt{cov(e_{id}, t_i)}$ in the head whose body is satisfied by A

   $\Leftrightarrow$ $check(e, \mathtt{t_i}) \subseteq P$ and the body of the only rule in $check(e, \mathtt{t_i})$ is satisfied by A

   $\Leftrightarrow$ $e \in E_{\mathtt{t_i}}$, $\mathtt{ctx(e_{id}, t_i)} \in A$ and $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i})$ extends e_{pi}

   $\Leftrightarrow$ $e \in E_{\mathtt{t_i}}$, $\mathtt{ctx(e_{id}, t_i)} \in A$, $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i}) \in AS(B \cup H \cup e_{ctx})$ and $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i})$ extends e_{pi} (by Theorem 6.3)

   $\Leftrightarrow$ $e \in E_{\mathtt{t_i}}$, $\mathtt{ctx(e_{id}, t_i)} \in A$, $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i})$ is an accepting answer set of e wrt $B \cup H$.

$\square$

Theorem 6.6. Let T be an $ILP_{LOAS}^{context}$ task with background knowledge B and hypothesis space S_M, and let $H \subseteq S_M$. Let $AS_{ids} = \{\mathtt{t_1}, \ldots, \mathtt{t_n}\}$ be a set of terms and $Pair_{ids}$ be a set of pairs $\langle \mathtt{t_i}, \mathtt{t_j} \rangle$, where $\mathtt{t_i}$ and $\mathtt{t_j}$ are terms in AS_{ids} such that no term occurs more than once in $Pair_{ids}$.

For each $\mathtt{t} \in AS_{ids}$ let $E_{\mathtt{t}}$ be a set of CDPIs and for each tuple $p \in Pair_{ids}$ let O_p be a set of CDOEs.

$$
\begin{aligned}
\text{Let } P = \mathcal{M}_1(T) \;&\cup\; \{\mathtt{in\_h(h_{id})} \mid h \in H\} \\
&\cup\; \{\mathtt{as(t).} \mid \mathtt{t} \in AS_{ids}\} \\
&\cup\; \{check(e, \mathtt{t}) \mid \langle \mathtt{t}, e \rangle \in E_{\mathtt{t}}, \mathtt{t} \in AS_{ids}\} \\
&\cup\; \{check\_ord(T, o, \mathtt{t_i}, \mathtt{t_j}) \mid p = \langle \mathtt{t_i}, \mathtt{t_j} \rangle \in Pair_{ids}, o \in O_p\}
\end{aligned}
$$

For each term $\mathtt{t} \in AS_{ids}$, let $E(\mathtt{t}) = E_{\mathtt{t}} \cup \{e \mid p = \langle \mathtt{t}, \_ \rangle \in Pair_{ids}, \langle e, \_, \_ \rangle \in O_p\} \cup \{e \mid p = \langle \_, \mathtt{t} \rangle \in Pair_{ids}, \langle \_, e, \_ \rangle \in O_p\}$.

1. For any list $[\langle I^1, e^1 \rangle, \ldots, \langle I^n, e^n \rangle]$ (of length $|AS_{ids}|$) such that each e^i is selected from $E^{+} \cup E^{-}$ and each I^i is an interpretation: $\exists A \in AS(P)$ such that $\forall i \in [1, n]$, $\mathtt{ctx(e^i_{id}, t_i)} \in A$ and $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i}) = I^i$ if and only if $\forall i \in [1, n]$, $I^i \in AS(B \cup H \cup e^i_{ctx})$.

2. For any answer set $A \in AS(P)$, $\forall i \in [1, n]$, $\forall e \in E^{+} \cup E^{-}$, $\mathtt{cov(e_{id}, t_i)} \in A$ if and only if $\mathtt{ctx(e_{id}, t_i)} \in A$, $e \in E(\mathtt{t_i})$ and $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i})$ is an accepting answer set of e wrt $B \cup H$.

3. For any $A \in AS(P)$, for any ordering example $o = \langle oe^1, oe^2, op \rangle$ and for any $i, j \in [1, n]$, $\mathtt{ord\_respected(o_{id}, t_i, t_j)} \in A$ if and only if $p = \langle \mathtt{t_i}, \mathtt{t_j} \rangle \in Pair_{ids}$, $o \in O_p$, $\mathtt{cov(oe^1_{id}, t_i)} \in A$, $\mathtt{cov(oe^2_{id}, t_j)} \in A$ and $\langle \mathcal{M}_{as}^{-1}(A, \mathtt{t_i}), \mathcal{M}_{as}^{-1}(A, \mathtt{t_j}) \rangle$ is an accepting pair of answer sets of o wrt $B \cup H$.

Proof. In this proof, we use the notation $check\_ord^1(T, o, \mathtt{t_1}, \mathtt{t_2}), \ldots, check\_ord^6(T, o, \mathtt{t_1}, \mathtt{t_2})$ to refer to the 6 components of $check\_ord(T, o, \mathtt{t_1}, \mathtt{t_2})$ (in Meta-program 6.3).

Let PTs be the set of terms that occur in $Pair_{ids}$ and Ls be the set of priority levels in $B \cup S_M$.

$$\begin{aligned}
P &= \mathcal{M}_1(T) & \cup\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h\in H\}\\
&& \cup\{\texttt{as}(\texttt{t}).\mid \texttt{t}\in AS_{ids}\}\\
&& \cup\{check(e,\texttt{t})\mid \texttt{t}\in AS_{ids}, e\in E_{\texttt{t}}\}\\
&& \cup\{check\_ord^i(T,o,\texttt{t}_\texttt{i},\texttt{t}_\texttt{j})\mid p=\langle \texttt{t}_\texttt{i},\texttt{t}_\texttt{j}\rangle\in Pair_{ids}, o\in O_p, i\in[1,6]\}\\
\\
&= \mathcal{M}_1(T) & \cup\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h\in H\}\\
&& \cup\{\texttt{as}(\texttt{t}).\mid \texttt{t}\in AS_{ids}\}\\
&& \cup\{check(e,\texttt{t})\mid \texttt{t}\in AS_{ids}, e\in E_{\texttt{t}}\}\\
&& \cup\{check(e_1,\texttt{t}_\texttt{i})\cup check(e_2,\texttt{t}_\texttt{j})\mid p=\langle \texttt{t}_\texttt{i},\texttt{t}_\texttt{j}\rangle\in Pair_{ids}, \langle e_1,e_2,op\rangle\in O_p\}\\
&& \cup\{check\_ord^i(T,o,\texttt{t}_\texttt{i},\texttt{t}_\texttt{j})\mid p=\langle \texttt{t}_\texttt{i},\texttt{t}_\texttt{j}\rangle\in Pair_{ids}, o\in O_p, i\in[2,6]\}\\
\\
&= \mathcal{M}_1(T) & \cup\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h\in H\}\\
&& \cup\{\texttt{as}(\texttt{t}).\mid \texttt{t}\in AS_{ids}\}\\
&& \cup\{check(e,\texttt{t})\mid \texttt{t}\in AS_{ids}, e\in E_{\texttt{t}}\}\\
&& \cup\{check(e,\texttt{t})\mid p=\langle \texttt{t},\_\rangle\in Pair_{ids}, \langle e,\_,op\rangle\in O_p\}\\
&& \cup\{check(e,\texttt{t})\mid p=\langle \_,\texttt{t}\rangle\in Pair_{ids}, \langle \_,e,op\rangle\in O_p\}\\
&& \cup\{check\_ord^i(T,o,\texttt{t}_\texttt{i},\texttt{t}_\texttt{j})\mid p=\langle \texttt{t}_\texttt{i},\texttt{t}_\texttt{j}\rangle\in Pair_{ids}, o\in O_p, i\in[2,6]\}\\
\\
&= \mathcal{M}_1(T) & \cup\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h\in H\}\\
&& \cup\{\texttt{as}(\texttt{t}).\mid \texttt{t}\in AS_{ids}\}\\
&& \cup\{check(e,\texttt{t})\mid \texttt{t}\in AS_{ids}, e\in E(\texttt{t})\}\\
&& \cup\{check\_ord^i(T,o,\texttt{t}_\texttt{i},\texttt{t}_\texttt{j})\mid p=\langle \texttt{t}_\texttt{i},\texttt{t}_\texttt{j}\rangle\in Pair_{ids}, o\in O_p, i\in[2,6]\}
\end{aligned}$$

We partition the program P as follows:

$P_1 = \mathcal{M}_1(T)\cup\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h\in H\}\cup\{check(e,\texttt{t})\mid \texttt{t}\in AS_{ids}, e\in E(\texttt{t})\}\cup\{\texttt{as}(\texttt{t}).\mid \texttt{t}\in AS_{ids}\}$
$P_2 = check\_ord^i(T,o,\texttt{t}_\texttt{i},\texttt{t}_\texttt{j})\mid p=\langle \texttt{t}_\texttt{i},\texttt{t}_\texttt{j}\rangle\in Pair_{ids}, o\in O_p, i\in[2,6]\}$

1. First note that the atoms that occur in the heads of P_2 do not occur in the program P_1. Hence, (by Corollary 2.14), $AS(P)=\{A\in AS(\{\texttt{a}.\mid \texttt{a}\in A'\}\cup P_2)\mid A'\in AS(P_1)\}$. As there is no recursion in P_2, there is a unique answer set of $\{\texttt{a}.\mid \texttt{a}\in A'\}\cup P_2$ for each $A'\in AS(P_1)$. Hence, $\{A\cap HB_{P_1}\mid A\in AS(P)\}=AS(P_1)$.

   Consider a list $[\langle I^1,e^1\rangle,\ldots,\langle I^n,e^n\rangle]$ such that each $e^i\in E^+\cup E^-$ and each I^i is an interpretation.

   Assume $\exists A\in AS(P)$ such that $\forall i\in[1,n]$, $\texttt{ctx}(\texttt{e}^\texttt{i}_{\texttt{id}},\texttt{t}_\texttt{i})\in A$ and $\mathcal{M}_{as}^{-1}(A,\texttt{t}_\texttt{i})=I^i$.

   $\Leftrightarrow$ $\exists A\in AS(P_1)$ such that $\forall i\in[1,n]$, $\texttt{ctx}(\texttt{e}^\texttt{i}_{\texttt{id}},\texttt{t}_\texttt{i})\in A$ and $\mathcal{M}_{as}^{-1}(A,\texttt{t}_\texttt{i})=I^i$.

   $\Leftrightarrow$ $\forall i\in[1,n]$ and $I^i\in AS(B\cup H\cup e^i_{ctx})$ (by Theorem 6.4, part (1)).

2. Recall from (1) that $\{A\cap HB_{P_1}\mid A\in AS(P)\}=AS(P_1)$.

   For any answer set $A\in AS(P)$, for any example e and any $i\in[1,n]$, assume that $\texttt{cov}(\texttt{e}_{\texttt{id}},\texttt{t}_\texttt{i})\in A$

   $\Leftrightarrow$ $\exists A'\in AS(P_1)$ such that $A\cap HB_{P_1}=A'$ and $\texttt{cov}(\texttt{e}_{\texttt{id}},\texttt{t}_\texttt{i})\in A'$

   $\Leftrightarrow$ $\exists A'\in AS(P_1)$ such that $A\cap HB_{P_1}=A'$, $\texttt{ctx}(\texttt{e}_{\texttt{id}},\texttt{t}_\texttt{i})\in A'$, $e\in E(\texttt{t})$ and $\mathcal{M}_{as}^{-1}(A',\texttt{t}_\texttt{i})$ is an accepting answer set of e wrt $B\cup H$ (by Theorem 6.4, part (2))

   $\Leftrightarrow$ $\texttt{ctx}(\texttt{e}_{\texttt{id}},\texttt{t}_\texttt{i})\in A$, $e\in E(\texttt{t})$ and $\mathcal{M}_{as}^{-1}(A,\texttt{t}_\texttt{i})$ is an accepting answer set of e wrt $B\cup H$

3. Recall from (1) that $\{A \cap HB_{P_1} \mid A \in AS(P)\} = AS(P_1)$.

   Let $A \in AS(P)$, and $o = \langle oe^1, oe^2, op\rangle$. Let $i, j \in [1, n]$.

   Note that for each $k \in [1, n], \forall \mathtt{w}(\mathtt{wt}, \mathtt{lev}, \mathtt{args}(\mathtt{a_1}, \ldots, \mathtt{a_m}), \mathtt{t_k}) \in HB_P$, $\mathtt{w}(\mathtt{wt}, \mathtt{lev}, \mathtt{args}(\mathtt{a_1}, \ldots, \mathtt{a_m}), \mathtt{t_k}) \in A$ if and only if $\exists W \in \mathit{weak}(\mathit{ground}(B \cup H))$ such that $\mathcal{M}_{as}^{-1}(A, \mathtt{t_k})$ satisfies $\mathit{body}(W)$ and $\mathit{tail}(W) = [\mathtt{wt@lev}, \mathtt{a_1}, \ldots, \mathtt{a_m}]$; i.e., if and only if $(\mathtt{wt}, \mathtt{lev}, \mathtt{a_1}, \ldots, \mathtt{a_m}) \in \mathit{Weak}(B \cup H, \mathcal{M}_{as}^{-1}(A, \mathtt{t_k}))$.

   Hence, for each priority level $\mathtt{lev}$ that occurs in $B \cup S_M$, $\mathtt{dom\_at\_lev}(\mathtt{t_i}, \mathtt{t_j}, \mathtt{lev}) \in A$ if and only if $(B \cup H)^{\mathcal{M}_{as}^{-1}(A,\mathtt{t_i})}_{\mathtt{lev}} - (B \cup H)^{\mathcal{M}_{as}^{-1}(A,\mathtt{t_j})}_{\mathtt{lev}} < 0$, which is true if and only if $(B \cup H)^{\mathcal{M}_{as}^{-1}(A,\mathtt{t_i})}_{\mathtt{lev}} < (B \cup H)^{\mathcal{M}_{as}^{-1}(A,\mathtt{t_j})}_{\mathtt{lev}}$. Similarly, $\mathtt{dom\_at\_lev}(\mathtt{t_j}, \mathtt{t_i}, \mathtt{lev}) \in A$ if and only if $(B \cup H)^{\mathcal{M}_{as}^{-1}(A,\mathtt{t_i})}_{\mathtt{lev}} > (B \cup H)^{\mathcal{M}_{as}^{-1}(A,\mathtt{t_j})}_{\mathtt{lev}}$.

   Hence, $\mathtt{dom}(\mathtt{t_i}, \mathtt{t_j}) \in A$ if and only if there is a level $\mathtt{lev}$ in $B \cup S_M$ such that $(B \cup H)^{\mathcal{M}_{as}^{-1}(A,\mathtt{t_i})}_{\mathtt{lev}} > (B \cup H)^{\mathcal{M}_{as}^{-1}(A,\mathtt{t_j})}_{\mathtt{lev}}$ and for all levels $\mathtt{lev'}$ in $B \cup S_M$ such that $\mathtt{lev} < \mathtt{lev'}$, $(B \cup H)^{\mathcal{M}_{as}^{-1}(A,\mathtt{t_i})}_{\mathtt{lev'}} = (B \cup H)^{\mathcal{M}_{as}^{-1}(A,\mathtt{t_j})}_{\mathtt{lev'}}$. This is the case if and only if $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i}) \succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \mathtt{t_j})$. Similarly $\mathtt{dom}(\mathtt{t_j}, \mathtt{t_i}) \in A$ if and only if $\mathcal{M}_{as}^{-1}(A, \mathtt{t_j}) \succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \mathtt{t_i})$.

   - We first show that $\mathtt{ord\_respected}(\mathtt{o_{id}}, \mathtt{t_i}, \mathtt{t_j}) \in A \Rightarrow p = \langle \mathtt{t_i}, \mathtt{t_j}\rangle \in \mathit{Pair}_{ids}$, $o \in O_p$, $\mathtt{cov}(\mathtt{oe^1_{id}}, \mathtt{t_i})$ and $\mathtt{cov}(\mathtt{oe^2_{id}}, \mathtt{t_j})$ are both in A and $\langle \mathcal{M}_{as}^{-1}(A, \mathtt{t_i}), \mathcal{M}_{as}^{-1}(A, \mathtt{t_j})\rangle$ is an accepting pair of answer sets of o wrt $B \cup H$.

     Assume that $\mathtt{ord\_respected}(\mathtt{o_{id}}, \mathtt{t_i}, \mathtt{t_j}) \in A$.

     Assume for contradiction that either $p \notin \mathit{Pair}_{ids}$, $o \notin O_p$, $\mathtt{cov}(\mathtt{oe^1_{id}}, \mathtt{t_i}) \notin A$ or $\mathtt{cov}(\mathtt{oe^2_{id}}, \mathtt{t_j}) \notin A$. Then either there are no rules in P with $\mathtt{ord\_respected}(\mathtt{o_{id}}, \mathtt{t_i}, \mathtt{t_j})$ in the head, or the bodies of each such rule is not satisfied by A. This contradicts A being an answer set of P ($\{\mathtt{ord\_respected}(\mathtt{o_{id}}, \mathtt{t_i}, \mathtt{t_j})\}$ would be a non-empty unfounded subset of A wrt P).

     It remains to show that $\langle \mathcal{M}_{as}^{-1}(A, \mathtt{t_i}), \mathcal{M}_{as}^{-1}(A, \mathtt{t_j})\rangle$ is an accepting pair of answer sets of o wrt $B \cup H$.

     Firstly, as $\mathtt{cov}(\mathtt{oe^1_{id}}, \mathtt{t_i})$ and $\mathtt{cov}(\mathtt{oe^2_{id}}, \mathtt{t_j})$ are both in A, $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i})$ and $\mathcal{M}_{as}^{-1}(A, \mathtt{t_j})$ must be accepting answer sets of oe^1 and oe^2 (respectively) wrt $B \cup H$ (by part (2)). So it remains to show that $\langle \mathcal{M}_{as}^{-1}(A, \mathtt{t_i}), \mathcal{M}_{as}^{-1}(A, \mathtt{t_j}), op\rangle \in \mathit{ord}(B \cup H, AS(B \cup H \cup oe^1_{ctx}) \cup AS(B \cup H \cup oe^2_{ctx})$.

     Case 1: op is $<$

     Assume for contradiction that $\mathtt{dom}(\mathtt{t_i}, \mathtt{t_j}) \notin A$. Then there would be no rule in $\mathit{ground}(P)$ with $\mathtt{ord\_respected}(\mathtt{o_{id}}, \mathtt{t_i}, \mathtt{t_j})$ in the head whose body is satisfied by A (which would contradict A being an answer set of P). Hence, $\mathtt{dom}(\mathtt{t_i}, \mathtt{t_j}) \in A$. Hence, $\mathcal{M}_{as}^{-1}(A, \mathtt{t_i}) \succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \mathtt{t_j})$. This means that $\langle \mathcal{M}_{as}^{-1}(A, \mathtt{t_i}), \mathcal{M}_{as}^{-1}(A, \mathtt{t_j}), op\rangle \in \mathit{ord}(B \cup H, AS(B \cup H \cup oe^1_{ctx}) \cup AS(B \cup H \cup oe^2_{ctx})$.

     Case 2: op is $>$

     Assume for contradiction that $\mathtt{dom}(\mathtt{t_j}, \mathtt{t_i}) \notin A$. Then there would be no rule in $\mathit{ground}(P)$ with $\mathtt{ord\_respected}(\mathtt{o_{id}}, \mathtt{t_i}, \mathtt{t_j})$ in the head whose body is satisfied by A (which would contradict A being an answer set of P). Hence, $\mathtt{dom}(\mathtt{t_j}, \mathtt{t_i}) \in A$. Hence, $\mathcal{M}_{as}^{-1}(A, \mathtt{t_j}) \succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \mathtt{t_i})$. This means that $\langle \mathcal{M}_{as}^{-1}(A, \mathtt{t_i}), \mathcal{M}_{as}^{-1}(A, \mathtt{t_j}), op\rangle \in \mathit{ord}(B \cup H, AS(B \cup H \cup oe^1_{ctx}) \cup AS(B \cup H \cup oe^2_{ctx})$.

Case 3: *op* is $\neq$

Assume for contradiction that $\texttt{dom}(\texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}) \notin A$ and $\texttt{dom}(\texttt{t}_\texttt{j}, \texttt{t}_\texttt{i}) \notin A$. Then there would be no rule in *ground*(P) with $\texttt{ord\_respected}(\texttt{o}_\texttt{id}, \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j})$ in the head whose body is satisfied by A (which would contradict A being an answer set of P). Hence, either $\texttt{dom}(\texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}) \in A$ or $\texttt{dom}(\texttt{t}_\texttt{j}, \texttt{t}_\texttt{i}) \in A$. Hence, either $\mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}) \succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j})$ or $\mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j}) \succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i})$. This means that $\langle \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}), \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j}), op\rangle \in ord(B \cup H, AS(B \cup H \cup oe^1_{ctx}) \cup AS(B \cup H \cup oe^2_{ctx})$.

Case 4: *op* is $=$

Assume for contradiction that either $\texttt{dom}(\texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}) \in A$ or $\texttt{dom}(\texttt{t}_\texttt{j}, \texttt{t}_\texttt{i}) \in A$. Then there would be no rule in *ground*(P) with $\texttt{ord\_respected}(\texttt{o}_\texttt{id}, \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j})$ in the head whose body is satisfied by A (which would contradict A being an answer set of P). Hence, $\texttt{dom}(\texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}) \notin A$ and $\texttt{dom}(\texttt{t}_\texttt{j}, \texttt{t}_\texttt{i}) \notin A$. Hence, $\mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}) \not\succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j})$ and $\mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j}) \not\succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i})$. This means that $\langle \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}), \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j}), op\rangle \in ord(B \cup H, AS(B \cup H \cup oe^1_{ctx}) \cup AS(B \cup H \cup oe^2_{ctx})$.

Case 5: *op* is $\leq$

Assume for contradiction that $\texttt{dom}(\texttt{t}_\texttt{j}, \texttt{t}_\texttt{i}) \in A$. Then there would be no rule in *ground*(P) with $\texttt{ord\_respected}(\texttt{o}_\texttt{id}, \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j})$ in the head whose body is satisfied by A (which would contradict A being an answer set of P). Hence, $\texttt{dom}(\texttt{t}_\texttt{j}, \texttt{t}_\texttt{i}) \notin A$. Hence, $\mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j}) \not\succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i})$. This means that $\langle \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}), \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j}), op\rangle \in ord(B \cup H, AS(B \cup H \cup oe^1_{ctx}) \cup AS(B \cup H \cup oe^2_{ctx})$.

Case 6: *op* is $\geq$

Assume for contradiction that $\texttt{dom}(\texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}) \in A$. Then there would be no rule in *ground*(P) with $\texttt{ord\_respected}(\texttt{o}_\texttt{id}, \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j})$ in the head whose body is satisfied by A (which would contradict A being an answer set of P). Hence, $\texttt{dom}(\texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}) \notin A$. Hence, $\mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}) \not\succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j})$. This means that $\langle \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}), \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j}), op\rangle \in ord(B \cup H, AS(B \cup H \cup oe^1_{ctx}) \cup AS(B \cup H \cup oe^2_{ctx})$.

- We now show that if $p = \langle \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}\rangle \in Pair_{ids}$, $o \in O_p$, $\texttt{cov}(\texttt{oe}^1_\texttt{id}, \texttt{t}_\texttt{i}) \in A, \texttt{cov}(\texttt{oe}^2_\texttt{id}, \texttt{t}_\texttt{j}) \in A$ and $\langle \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}), \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j})\rangle$ is an accepting pair of answer sets of o wrt $B \cup H$ then $\texttt{ord\_respected}(\texttt{o}_\texttt{id}, \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}) \in A$

  Assume $p \in Pair_{ids}$, $o \in O_p$, $\texttt{cov}(\texttt{oe}^1_\texttt{id}, \texttt{t}_\texttt{i}) \in A, \texttt{cov}(\texttt{oe}^2_\texttt{id}, \texttt{t}_\texttt{j}) \in A$ and $\langle \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}), \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j})\rangle$ is an accepting pair of answer sets of o wrt $B \cup H$. We need to show that there is at least one rule in *ground*(P) whose head is $\texttt{ord\_respected}(\texttt{o}_\texttt{id}, \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j})$ and whose body is satisfied by A.

  Case 1: *op* is $<$

  $\mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}) \succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j})$ (as the two interpretations form an accepting pair of answer sets of $\langle oe^1, oe^2, op\rangle$). Hence, $\texttt{dom}(\texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}) \in A$.

  Hence, the body of the rule: $\texttt{ord\_respected}(\texttt{o}_\texttt{id}, \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j})\texttt{:-}\texttt{dom}(\texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}), \texttt{cov}(\texttt{oe}^1_\texttt{id}, \texttt{t}_\texttt{i}), \texttt{cov}(\texttt{oe}^2_\texttt{id}, \texttt{t}_\texttt{j})$ is satisfied.

  Case 2: *op* is $>$

  $\mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j}) \succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i})$. Hence, $\texttt{dom}(\texttt{t}_\texttt{j}, \texttt{t}_\texttt{i}) \in A$.

  Hence, the body of the rule: $\texttt{ord\_respected}(\texttt{o}_\texttt{id}, \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j})\texttt{:-}\texttt{dom}(\texttt{t}_\texttt{j}, \texttt{t}_\texttt{i}), \texttt{cov}(\texttt{oe}^1_\texttt{id}, \texttt{t}_\texttt{i}), \texttt{cov}(\texttt{oe}^2_\texttt{id}, \texttt{t}_\texttt{j})$ is satisfied.

  Case 3: *op* is $\neq$

  Either $\mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i}) \succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j})$ or $\mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{j}) \succ_{B \cup H} \mathcal{M}_{as}^{-1}(A, \texttt{t}_\texttt{i})$. Hence, either $\texttt{dom}(\texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}) \in A$ or $\texttt{dom}(\texttt{t}_\texttt{j}, \texttt{t}_\texttt{i}) \in A$. Hence, the body of one of: $\texttt{ord\_respected}(\texttt{o}_\texttt{id}, \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j})\texttt{:-}\texttt{dom}(\texttt{t}_\texttt{i}, \texttt{t}_\texttt{j}), \texttt{cov}(\texttt{oe}^1_\texttt{id}, \texttt{t}_\texttt{i}), \texttt{cov}(\texttt{oe}^2_\texttt{id}, \texttt{t}_\texttt{j})$ or $\texttt{ord\_respected}(\texttt{o}_\texttt{id}, \texttt{t}_\texttt{i}, \texttt{t}_\texttt{j})\texttt{:-}\texttt{dom}(\texttt{t}_\texttt{j}, \texttt{t}_\texttt{i}), \texttt{cov}(\texttt{oe}^1_\texttt{id}, \texttt{t}_\texttt{i}),$ $\texttt{cov}(\texttt{oe}^2_\texttt{id}, \texttt{t}_\texttt{j})$ is satisfied.

Case 4: *op* is $=$

$\mathcal{M}_{as}^{-1}(A,\mathtt{t_i}) \not\succ_{B\cup H} \mathcal{M}_{as}^{-1}(A,\mathtt{t_j})$ and $\mathcal{M}_{as}^{-1}(A,\mathtt{t_j}) \not\succ_{B\cup H} \mathcal{M}_{as}^{-1}(A,\mathtt{t_i})$. Hence, $\mathtt{dom(t_i,t_j)} \notin A$ and $\mathtt{dom(t_j,t_i)} \notin A$. Hence, the body of the rule: $\mathtt{ord\_respected(o_{id},t_i,t_j)}$ `:- not` $\mathtt{dom(t_i,t_j)}$, `not` $\mathtt{dom(t_j,t_i)}, \mathtt{cov(oe^1_{id},t_i)}, \mathtt{cov(oe^2_{id},t_j)}$ is satisfied.

Case 5: *op* is $\leq$

$\mathcal{M}_{as}^{-1}(A,\mathtt{t_j}) \not\succ_{B\cup H} \mathcal{M}_{as}^{-1}(A,\mathtt{t_i})$. Hence, $\mathtt{dom(t_j,t_i)} \notin A$. Hence, the body of the rule: $\mathtt{ord\_respected(o_{id},t_i,t_j)}$ `:- not` $\mathtt{dom(t_j,t_i)}, \mathtt{cov(oe^1_{id},t_i)}, \mathtt{cov(oe^2_{id},t_j)}$ is satisfied.

Case 6: *op* is $\geq$

$\mathcal{M}_{as}^{-1}(A,\mathtt{t_i}) \not\succ_{B\cup H} \mathcal{M}_{as}^{-1}(A,\mathtt{t_j})$. Hence, $\mathtt{dom(t_i,t_j)} \notin A$. Hence, the body of the rule: $\mathtt{ord\_respected(o_{id},t_i,t_j)}$ `:- not` $\mathtt{dom(t_i,t_j)}, \mathtt{cov(oe^1_{id},t_i)}, \mathtt{cov(oe^2_{id},t_j)}$ is satisfied.

□

Theorem 6.10. Let T be an $ILP^{context}_{LOAS}$ task, and H be a hypothesis.

1. $\exists A \in AS(\mathcal{M}(T))$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ if and only if $H \in \mathcal{P}(T)$.
2. $\forall e \in E^-$, for any $A \in AS(\mathcal{M}(T) \cup \{\mathtt{check\_violating.}\})$ such that $\mathtt{v\_i(e_{id})} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$, $\mathcal{M}^{-1}_{as}(A,\mathtt{v1})$ is an accepting answer set of e wrt $B \cup H$.
3. $\forall o \in O^c$, for any $A \in AS(\mathcal{M}(T) \cup \{\mathtt{check\_violating.}\})$ such that $\mathtt{v\_p(o_{id})} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$, $\langle \mathcal{M}^{-1}_{as}(A,\mathtt{v1}), \mathcal{M}^{-1}_{as}(A,\mathtt{v2})\rangle$ is an accepting pair of answer sets of *inverse*(o) wrt $B \cup H$.
4. $\exists A \in AS(\mathcal{M}(T) \cup \{\mathtt{check\_violating.}\})$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ if and only if $H \in \mathcal{V}(T)$.

Proof.

1. Let $[cdpi^1, \ldots, cdpi^{m+2n}]$ be the list $[e^1, \ldots, e^m, o^1_{eg1}, o^1_{eg2}, \ldots, o^n_{eg1}, o^n_{eg2}]$, where $\{e^1, \ldots, e^m\} = E^+$ and $\{o^1, \ldots, o^n\} = O^b$, and each o^1_{eg1} and o^2_{eg2} is assumed to be a copy of the CDPI example, with id o_{id1} and o_{id2}, respectively.

   Assume $H \in \mathcal{P}(T)$.

   $\Leftrightarrow$ there is a list $[\langle I^1, cdpi^1\rangle, \ldots, \langle I^{m+2n}, cdpi^{m+2n}\rangle]$ such that for each $i \in [1, m{+}2n]$, I^i is an accepting answer set of $cdpi^i$ wrt $B \cup H$ and for each $i \in [1,n]$, $\langle I^{m+2i}, I^{m+2i+1}\rangle$ is an accepting pair of answer sets of o^i.

   $$\Leftrightarrow \exists A \in AS\left(\begin{array}{lll} \mathcal{M}_1(T) & \cup & \{\mathtt{in\_h(h_{id}).} \mid h \in H\} \\ & \cup & \{\mathtt{as(cdpi^i_{id}).} \mid i \in [1, m+2n]\} \\ & \cup & \{check(e^i, e^i_{id}) \mid i \in [1,m]\} \\ & \cup & \{check\_ord(T, o^i, o^i_{id1}, o^i_{id2}) \mid i \in [1,n]\} \end{array}\right)$$

   such that for each $i \in [1, m+2n]$, $\mathtt{ctx(cdpi^i_{id}, cdpi^i_{id})} \in A$ and $\mathcal{M}^{-1}_{as}(A, \mathtt{cdpi^i_{id}})$ is an accepting answer set of $cdpi^i$ wrt $B \cup H$ and for each $i \in [1,n]$, $\langle \mathcal{M}^{-1}_{as}(A, \mathtt{cdpi^{m+2i}_{id}}), \mathcal{M}^{-1}_{as}(A, \mathtt{cdpi^{m+2i+1}_{id}})\rangle$ is an accepting pair of answer sets of o^i (by Theorem 6.6 (1))

$$\Leftrightarrow \exists A \in AS\left(\begin{array}{lll}\mathcal{M}_1(T) & \cup & \{\mathtt{in\_h(h_{id})}.\mid h \in H\}\\ & \cup & \{\mathtt{as(cdpi^{i}_{id})}.\mid i \in [1, m+2n]\}\\ & \cup & \{\mathit{check}(e^i, e^i_{id}) \mid i \in [1,m]\}\\ & \cup & \{\mathit{check\_ord}(T, o^i, o^i_{id1}, o^i_{id2}) \mid i \in [1,n]\}\end{array}\right)$$

such that for each $i \in [1,m]$, $\mathtt{cov(cdpi^{i}_{id}, cdpi^{i}_{id})} \in A$ and for each $i \in [1,n]$, $\langle \mathcal{M}^{-1}_{as}(A, \mathtt{cdpi^{m+2i}_{id}}), \mathcal{M}^{-1}_{as}(A, \mathtt{cdpi^{m+2i+1}_{id}})\rangle$ is an accepting pair of answer sets of o^i (by Theorem 6.6 (2))

$$\Leftrightarrow \exists A \in AS\left(\begin{array}{lll}\mathcal{M}_1(T) & \cup & \{\mathtt{in\_h(h_{id})}.\mid h \in H\}\\ & \cup & \{\mathtt{as(cdpi^{i}_{id})}.\mid i \in [1, m+2n]\}\\ & \cup & \{\mathit{check}(e^i, e^i_{id}) \mid i \in [1,m]\}\\ & \cup & \{\mathit{check\_ord}(T, o^i, o^i_{id1}, o^i_{id2}) \mid i \in [1,n]\}\end{array}\right)$$

such that for each $i \in [1, m+2n]$, $\mathtt{cov(cdpi^{i}_{id}, cdpi^{i}_{id})} \in A$ and for each $i \in [1,n]$, $\mathtt{ord\_respected(o^{i}_{id}, o^{i}_{id1}, o^{i}_{id2})} \in A$ (by Theorem 6.6 (3))

$$\Leftrightarrow \left(\begin{array}{lll}\mathcal{M}_1(T) & \cup & \{\mathtt{in\_h(h_{id})}.\mid h \in H\}\\ & \cup & \{\mathtt{as(cdpi^{i}_{id})}.\mid i \in [1, m+2n]\}\\ & \cup & \left\{\begin{array}{l}\mathit{check}(e^i, e^i_{id})\\ \mathtt{:\text{-}\ not\ cov(e^{i}_{id}, e^{i}_{id}).}\end{array}\middle| i \in [1,m]\right\}\\ & \cup & \left\{\begin{array}{l}\mathit{check\_ord}(T, o^i, o^i_{id1}, o^i_{id2})\\ \mathtt{:\text{-}\ not\ ord\_respected(o^{i}_{id}, o^{i}_{id1}, o^{i}_{id2}).}\end{array}\middle| i \in [1,n]\right\}\end{array}\right)$$

is satisfiable.

$$\Leftrightarrow \exists A \in AS\left(\begin{array}{lll}\mathcal{M}_1(T) & \cup & \{\mathtt{0\{in\_h(h_{id})\}1}.\mid h \in H\}\\ & \cup & \{\mathtt{as(cdpi^{i}_{id})}.\mid i \in [1, m+2n]\}\\ & \cup & \left\{\begin{array}{l}\mathit{check}(e^i, e^i_{id})\\ \mathtt{:\text{-}\ not\ cov(e^{i}_{id}, e^{i}_{id}).}\end{array}\middle| i \in [1,m]\right\}\\ & \cup & \left\{\begin{array}{l}\mathit{check\_ord}(T, o^i, o^i_{id1}, o^i_{id2})\\ \mathtt{:\text{-}\ not\ ord\_respected(o^{i}_{id}, o^{i}_{id1}, o^{i}_{id2}).}\end{array}\middle| i \in [1,n]\right\}\end{array}\right)$$

such that $\mathcal{M}^{-1}_{in\_h}(A) = H$

$\Leftrightarrow \exists A \in AS(\mathcal{M}(T))$ such that $\mathcal{M}^{-1}_{in\_h}(A) = H$ (the other rules in the program depend on `check_violating` which does not occur in the head of any rule in the program).

2. Let $e^* \in E^-$ and $A \in AS(\mathcal{M}(T) \cup \{\mathtt{check\_violating.}\})$ such that $\mathtt{v\_i(e^{*}_{id})} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$

   $\Rightarrow A \cap HB^{rel}_{P_1} \in AS(P_1)$, where P_1 is the program:

   $$\{\mathtt{0\{in\_h(h_{id})\}1.}|h \in S_M\} \cup \mathcal{M}_1(T) \cup \{\mathtt{check\_violating.}\}$$
   $$\cup\left\{\begin{array}{l}\mathtt{as(v1)\ {:}\text{-}\ check\_violating.}\\ \mathtt{v\_i(e_{id})\ {:}\text{-}\ cov(e_{id}, v1).}\\ \mathit{check}(e, \mathtt{v1})\end{array}\middle| e \in E^-\right\}$$

   (as $HB^{rel}_{P_1}$ is a splitting set of $\mathcal{M}(T) \cup \{\mathtt{check\_violating.}\}$).

   $\Rightarrow A \cap HB^{rel}_{P_2} \in AS(P_2)$, where P_2 is the program:

   $$\{\mathtt{in\_h(h_{id}).}|h \in H\} \cup \mathcal{M}_1(T)$$
   $$\cup\left\{\begin{array}{l}\mathtt{as(v1).}\\ \mathtt{v\_i(e_{id})\ {:}\text{-}\ cov(e_{id}, v1).}\\ \mathit{check}(e, \mathtt{v1})\end{array}\middle| e \in E^-\right\}$$

   $\Rightarrow A \cap HB^{rel}_{P_3} \in AS(P_3)$, where P_3 is the program:

$\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).|h \in H\} \cup \mathcal{M}_1(T)$
$$\cup \left\{ \begin{array}{l} \texttt{as(v1).} \\ \texttt{v\_i}(\texttt{e}^*_{\texttt{id}})\texttt{:-}\,\texttt{cov}(\texttt{e}^*_{\texttt{id}}, \texttt{v1}). \\ \mathit{check}(e^*, \texttt{v1}) \end{array} \right\}$$
(as $HB^{rel}_{P_3}$ is a splitting set of P_2).

$\Rightarrow A \cap HB^{rel}_{P_4} \in AS(P_4)$, where P_4 is the program:
$\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).|h \in H\} \cup \mathcal{M}_1(T)$
$$\cup \left\{ \begin{array}{l} \texttt{as(v1).} \\ \mathit{check}(e^*, \texttt{v1}) \end{array} \right\}$$
and $\texttt{cov}(\texttt{e}^*_{\texttt{id}}, \texttt{v1}) \in A \cap HB^{rel}_{P_4}$

$\Rightarrow \mathcal{M}^{-1}_{as}(A \cap HB^{rel}_{P_4}, \texttt{v1})$ is an accepting answer set of e^* wrt $B \cup H$ (by Theorem 6.4 part (2)).

$\Rightarrow \mathcal{M}^{-1}_{as}(A, \texttt{v1})$ is an accepting answer set of e^* wrt $B \cup H$.

3. Let $o^* \in O^c$ and $A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$ such that $\texttt{v\_p}(\texttt{o}^*_{\texttt{id}}) \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$

$\Rightarrow A \cap HB^{rel}_{P_1} \in AS(P_1)$, where P_1 is the program:
$\{\texttt{0}\{\texttt{in\_h}(\texttt{h}_{\texttt{id}})\}\texttt{1}|h \in S_M\} \cup \mathcal{M}_1(T) \cup \{\texttt{check\_violating.}\}$
$$\cup \left\{ \begin{array}{l|l} \texttt{as(v1)\,:-\,check\_violating.} & \\ \texttt{as(v2)\,:-\,check\_violating.} & \\ \texttt{v\_p}(\texttt{o}_{\texttt{id}})\texttt{:-}\,\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{v1}, \texttt{v2}). & o \in O^c \\ \mathit{check\_ord}(\mathit{inverse}(o), \texttt{v1}, \texttt{v2}) & \end{array} \right\}$$
(as $HB^{rel}_{P_1}$ is a splitting set of $\mathcal{M}(T) \cup \{\texttt{check\_violating.}\}$).

$\Rightarrow A \cap HB^{rel}_{P_2} \in AS(P_2)$, where P_2 is the program:
$\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).|h \in H\} \cup \mathcal{M}_1(T)$
$$\cup \left\{ \begin{array}{l|l} \texttt{as(v1).} & \\ \texttt{as(v2).} & \\ \texttt{v\_p}(\texttt{o}_{\texttt{id}})\texttt{:-}\,\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{v1}, \texttt{v2}). & o \in O^c \\ \mathit{check\_ord}(\mathit{inverse}(o), \texttt{v1}, \texttt{v2}) & \end{array} \right\}$$

$\Rightarrow A \cap HB^{rel}_{P_3} \in AS(P_3)$, where P_3 is the program:
$\{\texttt{in\_h}(\texttt{h}_{\texttt{id}})|h \in H\} \cup \mathcal{M}_1(T)$
$$\cup \left\{ \begin{array}{l|l} \texttt{as(v1).} & \\ \texttt{as(v2).} & o \in O^c \\ \mathit{check\_ord}(\mathit{inverse}(o), \texttt{v1}, \texttt{v2}) & \end{array} \right\} \text{ and } \texttt{ord\_respected}(\texttt{o}^*_{\texttt{id}}, \texttt{v1}, \texttt{v2}) \in A \cap HB^{rel}_{P_3}$$

$\Rightarrow A \cap HB^{rel}_{P_4} \in AS(P_4)$, where P_4 is the program:
$\{\texttt{in\_h}(\texttt{h}_{\texttt{id}})|h \in H\} \cup \mathcal{M}_1(T)$
$$\cup \left\{ \begin{array}{l|l} \texttt{as(v1).} & \\ \texttt{as(v2).} & \\ \mathit{check}(o_{eg1}, \texttt{v1}) & o \in O^c \\ \mathit{check}(o_{eg2}, \texttt{v2}) & \\ \mathit{check\_ord}(\mathit{inverse}(o), \texttt{v1}, \texttt{v2}) & \end{array} \right\} \text{ and } \texttt{ord\_respected}(\texttt{o}^*_{\texttt{id}}, \texttt{v1}, \texttt{v2}) \in A \cap HB^{rel}_{P_4}$$
(the extra program fragments are contained in the *check_ord* program).

$\Rightarrow \langle \mathcal{M}^{-1}_{as}(A, \texttt{v1}), \mathcal{M}^{-1}_{as}(A, \texttt{v2}) \rangle$ is an accepting pair of answer sets of $\mathit{inverse}(o)$ wrt $B \cup H$ (by Theorem 6.6, part (3))

4. Let $[\mathit{cdpi}^1, \ldots, \mathit{cdpi}^{m+2n}]$ be the list $[e^1, \ldots, e^m, o^1_{eg1}, o^1_{eg2}, \ldots, o^n_{eg1}, o^n_{eg2}]$, where $\{e^1, \ldots, e^m\} = E^+$ and

$\{o^1, \ldots, o^n\} = O^b$, and each o^1_{eg1} and o^2_{eg2} is assumed to be a copy of the CDPI example, with id o_{id1} and o_{id2}, respectively.

Assume that $\exists A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$

$$\Leftrightarrow \exists A_1 \in AS\left(\begin{array}{rcl} \mathcal{M}_1(T) & \cup & \{\texttt{0\{in\_h(h}_{\texttt{id}}\texttt{)\}1.} \mid h \in H\} \\ & \cup & \{\texttt{as(cdpi}^{\texttt{i}}_{\texttt{id}}\texttt{).} \mid i \in [1, m+2n]\} \\ & \cup & \left\{\begin{array}{l} \mathit{check}(e^i, e^i_{id}) \\ \texttt{:- not cov(e}^{\texttt{i}}_{\texttt{id}}\texttt{, e}^{\texttt{i}}_{\texttt{id}}\texttt{).} \end{array}\middle| i \in [1, m]\right\} \\ & \cup & \left\{\begin{array}{l} \mathit{check\_ord}(T, o^i, o^i_{id1}, o^i_{id2}) \\ \texttt{:- not ord\_respected(o}^{\texttt{i}}_{\texttt{id}}\texttt{, o}^{\texttt{i}}_{\texttt{id1}}\texttt{, o}^{\texttt{i}}_{\texttt{id2}}\texttt{).} \end{array}\middle| i \in [1, n]\right\} \end{array}\right)$$

such that $\mathcal{M}^{-1}_{in\_h}(A_1) = H$ and:

$$\left(\begin{array}{rcl} \{\texttt{a.} \mid \texttt{a} \in A\} & \cup & \mathcal{M}_1(T) \\ & \cup & \left\{\begin{array}{l} \texttt{check\_violating.} \\ \texttt{as(v1) :- check\_violating.} \\ \texttt{as(v2) :- check\_violating.} \\ \texttt{:- check\_violating, not violating.} \\ \texttt{violating :- v\_i(\_), check\_violating.} \\ \texttt{violating :- v\_p(\_), check\_violating.} \end{array}\right\} \\ & \cup & \left\{\begin{array}{l} \texttt{v\_i(e}_{\texttt{id}}\texttt{) :- cov(e}_{\texttt{id}}\texttt{, v1).} \\ \mathit{check}(e, \texttt{v1}) \end{array}\middle| e \in E^-\right\} \\ & \cup & \left\{\begin{array}{l} \texttt{v\_p(o}_{\texttt{id}}\texttt{) :- ord\_respected(o}_{\texttt{id}}\texttt{, v1, v2).} \\ \mathit{check\_ord}(\mathit{inverse}(o), \texttt{v1}, \texttt{v2}) \end{array}\middle| o \in O^c\right\} \end{array}\right)$$

is satisfiable (by Corollary 2.14)

$$\Leftrightarrow \exists A_1 \in AS\left(\begin{array}{rcl} \mathcal{M}_1(T) & \cup & \{\texttt{0\{in\_h(h}_{\texttt{id}}\texttt{)\}1.} \mid h \in H\} \\ & \cup & \{\texttt{as(cdpi}^{\texttt{i}}_{\texttt{id}}\texttt{).} \mid i \in [1, m+2n]\} \\ & \cup & \left\{\begin{array}{l} \mathit{check}(e^i, e^i_{id}) \\ \texttt{:- not cov(e}^{\texttt{i}}_{\texttt{id}}\texttt{, e}^{\texttt{i}}_{\texttt{id}}\texttt{).} \end{array}\middle| i \in [1, m]\right\} \\ & \cup & \left\{\begin{array}{l} \mathit{check\_ord}(T, o^i, o^i_{id1}, o^i_{id2}) \\ \texttt{:- not ord\_respected(o}^{\texttt{i}}_{\texttt{id}}\texttt{, o}^{\texttt{i}}_{\texttt{id1}}\texttt{, o}^{\texttt{i}}_{\texttt{id2}}\texttt{).} \end{array}\middle| i \in [1, n]\right\} \end{array}\right)$$

such that $\mathcal{M}^{-1}_{in\_h}(A_1) = H$ and:

$$\left(\begin{array}{rcl} \{\texttt{in\_h(h}_{\texttt{id}}\texttt{).} \mid h \in H\} & \cup & \mathcal{M}_1(T) \\ & \cup & \left\{\begin{array}{l} \texttt{check\_violating.} \\ \texttt{as(v1) :- check\_violating.} \\ \texttt{as(v2) :- check\_violating.} \\ \texttt{:- check\_violating, not violating.} \\ \texttt{violating :- v\_i(\_), check\_violating.} \\ \texttt{violating :- v\_p(\_), check\_violating.} \end{array}\right\} \\ & \cup & \left\{\begin{array}{l} \texttt{v\_i(e}_{\texttt{id}}\texttt{) :- cov(e}_{\texttt{id}}\texttt{, v1).} \\ \mathit{check}(e, \texttt{v1}) \end{array}\middle| e \in E^-\right\} \\ & \cup & \left\{\begin{array}{l} \texttt{v\_p(o}_{\texttt{id}}\texttt{) :- ord\_respected(o}_{\texttt{id}}\texttt{, v1, v2).} \\ \mathit{check\_ord}(\mathit{inverse}(o), \texttt{v1}, \texttt{v2}) \end{array}\middle| o \in O^c\right\} \end{array}\right)$$

is satisfiable (the other atoms in A_1 do not occur in the second program).

$$\Leftrightarrow H \in \mathcal{P}(T) \text{ and } \left(\begin{array}{lll} \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h\in H\} & \cup & \mathcal{M}_1(T)\\ & \cup & \left\{\begin{array}{l} \texttt{check\_violating.}\\ \texttt{as(v1) :- check\_violating.}\\ \texttt{as(v2) :- check\_violating.}\\ \texttt{:- check\_violating, not violating.}\\ \texttt{violating :- v\_i(\_), check\_violating.}\\ \texttt{violating :- v\_p(\_), check\_violating.} \end{array}\right\} \\ & \cup & \left\{\begin{array}{l} \texttt{v\_i}(\texttt{e}_{\texttt{id}})\texttt{ :- cov}(\texttt{e}_{\texttt{id}},\texttt{v1}).\\ \mathit{check}(e,\texttt{v1})\end{array}\middle| e\in E^-\right\}\\ & \cup & \left\{\begin{array}{l} \texttt{v\_p}(\texttt{o}_{\texttt{id}})\texttt{ :- ord\_respected}(\texttt{o}_{\texttt{id}},\texttt{v1},\texttt{v2}).\\ \mathit{check\_ord}(\mathit{inverse}(o),\texttt{v1},\texttt{v2})\end{array}\middle| o\in O^c\right\} \end{array}\right)$$

is satisfiable (by part (1))

$$\Leftrightarrow H \in \mathcal{P}(T) \text{ and } \left(\begin{array}{lll} \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h\in H\} & \cup & \mathcal{M}_1(T)\\ & \cup & \left\{\begin{array}{l} \texttt{as(v1).}\\ \texttt{as(v2).}\\ \texttt{:- not violating.}\\ \texttt{violating :- v\_i(\_).}\\ \texttt{violating :- v\_p(\_).} \end{array}\right\} \\ & \cup & \left\{\begin{array}{l} \texttt{v\_i}(\texttt{e}_{\texttt{id}})\texttt{ :- cov}(\texttt{e}_{\texttt{id}},\texttt{v1}).\\ \mathit{check}(e,\texttt{v1})\end{array}\middle| e\in E^-\right\}\\ & \cup & \left\{\begin{array}{l} \texttt{v\_p}(\texttt{o}_{\texttt{id}})\texttt{ :- ord\_respected}(\texttt{o}_{\texttt{id}},\texttt{v1},\texttt{v2}).\\ \mathit{check\_ord}(\mathit{inverse}(o),\texttt{v1},\texttt{v2})\end{array}\middle| o\in O^c\right\} \end{array}\right)$$

is satisfiable

$\Leftrightarrow H \in \mathcal{P}(T)$ and:

$$\exists A \in AS \left(\begin{array}{lll} \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h\in H\} & \cup & \mathcal{M}_1(T)\\ & \cup & \left\{\begin{array}{l} \texttt{as(v1).}\\ \texttt{as(v2).}\\ \texttt{violating :- v\_i(\_).}\\ \texttt{violating :- v\_p(\_).} \end{array}\right\} \\ & \cup & \left\{\begin{array}{l} \texttt{v\_i}(\texttt{e}_{\texttt{id}})\texttt{ :- cov}(\texttt{e}_{\texttt{id}},\texttt{v1}).\\ \mathit{check}(e,\texttt{v1})\end{array}\middle| e\in E^-\right\}\\ & \cup & \left\{\begin{array}{l} \texttt{v\_p}(\texttt{o}_{\texttt{id}})\texttt{ :- ord\_respected}(\texttt{o}_{\texttt{id}},\texttt{v1},\texttt{v2}).\\ \mathit{check\_ord}(\mathit{inverse}(o),\texttt{v1},\texttt{v2})\end{array}\middle| o\in O^c\right\} \end{array}\right)$$

such that $\texttt{violating} \in A$

$\Leftrightarrow H \in \mathcal{P}(T)$ and:

$$\exists A \in AS \left(\begin{array}{lll} \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h\in H\} & \cup & \mathcal{M}_1(T)\\ & \cup & \left\{\begin{array}{l} \texttt{as(v1).}\\ \texttt{as(v2).} \end{array}\right\} \\ & \cup & \left\{\begin{array}{l} \texttt{v\_i}(\texttt{e}_{\texttt{id}})\texttt{ :- cov}(\texttt{e}_{\texttt{id}},\texttt{v1}).\\ \mathit{check}(e,\texttt{v1})\end{array}\middle| e\in E^-\right\}\\ & \cup & \left\{\begin{array}{l} \texttt{v\_p}(\texttt{o}_{\texttt{id}})\texttt{ :- ord\_respected}(\texttt{o}_{\texttt{id}},\texttt{v1},\texttt{v2}).\\ \mathit{check\_ord}(\mathit{inverse}(o),\texttt{v1},\texttt{v2})\end{array}\middle| o\in O^c\right\} \end{array}\right)$$

such that $\exists e \in E^-$ such that $\texttt{v\_i}(\texttt{e}_{\texttt{id}}) \in A$ or $\exists o \in O^c$ such that $\texttt{v\_p}(\texttt{o}_{\texttt{id}}) \in A$

$\Leftrightarrow H \in \mathcal{P}(T)$ and:

$$\exists A \in AS\left(\begin{array}{lll} \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h \in H\} & \cup & \mathcal{M}_1(T) \\ & \cup & \left\{\begin{array}{l}\texttt{as(v1).}\\ \texttt{as(v2).}\end{array}\right\} \\ & \cup & \left\{ \mathit{check}(e, \texttt{v1}) \;\middle|\; e \in E^-\right\} \\ & \cup & \left\{ \mathit{check\_ord}(\mathit{inverse}(o), \texttt{v1}, \texttt{v2}) \;\middle|\; o \in O^c\right\} \end{array}\right)$$

such that $\exists e \in E^-$ such that $\texttt{cov}(\texttt{e}_{\texttt{id}}, \texttt{v1}) \in A$ or $\exists o \in O^c$ such that $\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{v2}, \texttt{v2}) \in A$

$\Leftrightarrow H \in \mathcal{P}(T)$ and either $\exists e \in E^-$ such that e has at least one accepting answer set wrt $B \cup H$ or $\exists o \in O^c$ such that there is at least one accepting pair of answer sets of $\mathit{inverse}(o)$ wrt $B \cup H$ (by Theorem 6.6)

$\Leftrightarrow H \in \mathcal{P}(T)$ and $H \in \mathcal{V}(T)$

$\Leftrightarrow H \in \mathcal{V}(T)$

$\square$

Proposition 6.11. Let T be any $ILP_{LOAS}^{context}$ task and $n \in \mathbb{N}$. $AS(\mathcal{M}^n(T)) = \{A \in \mathcal{M}(T) \mid |\mathcal{M}_{in\_h}^{-1}(A)| = n\}$.

Proof. Let $T = \langle B, S_M, E\rangle$, where $S_M = \{R^1, \ldots, R^m\}$.

Assume that $A \in AS(\mathcal{M}^n(T))$

$\Leftrightarrow A \in AS(\mathcal{M}(T))$ and $\#\texttt{sum}\{\texttt{in\_h}(\texttt{R}^1_{\texttt{id}}) = |\texttt{R}^1|, \ldots, \texttt{in\_h}(\texttt{R}^{\texttt{m}}_{\texttt{id}}) = |\texttt{R}^{\texttt{m}}|\} \neq \texttt{n}$ is not satisfied by A. (By Lemma 2.12)

$\Leftrightarrow A \in AS(\mathcal{M}(T))$ and $\left(\sum_{h \in \mathcal{M}_{in\_h}^{-1}(A)} |h|\right) = n$

$\Leftrightarrow A \in AS(\mathcal{M}(T))$ and $|\mathcal{M}_{in\_h}^{-1}(A)| = n$

$\square$

Proposition 6.12. Let T be an $ILP_{LOAS}^{context}$ task, $n \in \mathbb{N}$ and V be a set of hypotheses of length n. Consider the program $P = \mathcal{M}^n(T) \cup \{\mathit{constraint}(v) \mid v \in V\}$. Then $\mathcal{P}^n(T)\backslash V = \left\{\mathcal{M}_{in\_h}^{-1}(A) \middle| A \in AS(P)\right\}$.

Proof. Assume that $H \in \mathcal{P}^n(T)\backslash V$

$\Leftrightarrow H \in \mathcal{P}(T)$, $|H| = n$ and $H \notin \mathcal{V}(T)$

$\Leftrightarrow \exists A \in AS(\mathcal{M}(T))$ such that $\mathcal{M}_{in\_h}^{-1}(A) = H$, $|H| = n$ and $H \notin \mathcal{V}(T)$, by Theorem 6.10 part(1).

$\Leftrightarrow \exists A \in AS(\mathcal{M}^n(T))$ such that $H = \mathcal{M}_{in\_h}^{-1}(A)$ and $H \notin V$, by Proposition 6.11

$\Leftrightarrow \exists A \in AS(\mathcal{M}^n(T))$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ and $\forall v \in V$, $\exists r \in v$ such that $r \notin H$

(the $\Rightarrow$ holds as because H is not in V, it cannot be equal to v, and it cannot be a strict subset of V as both have the length n, and no rule in the hypothesis space can have length 0).

$\Leftrightarrow \exists A \in AS(\mathcal{M}^n(T))$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ and $\forall v \in V$, A does not satisfy the body of $constraint(v)$

$\Leftrightarrow \exists A \in AS(P)$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$
(by Lemma 2.12)

$\Leftrightarrow H \in AS\left(\left\{\mathcal{M}^{-1}_{in\_h}(A) \middle| A \in AS(P)\right\}\right)$

□

Proposition 6.13. Let T be any well-defined $ILP^{context}_{LOAS}$ task, $n \in \mathbb{N}$, and HS be any finite set of hypotheses.

1. $ground^{rel}(\mathcal{M}^n(T) \cup \{\texttt{check\_violating.}\})$ is finite.
2. $ground^{rel}(\mathcal{M}^n(T) \cup \{constraint(H) \mid H \in HS\})$ is finite.

Proof.

1. To prove that $\mathcal{M}^n(T) \cup \{\texttt{check\_violating}\}$ has a finite relevant grounding, it is sufficient to prove that $|HB^{rel}_{\mathcal{M}^n(T)\cup\{\texttt{check\_violating}\}}|$ is finite. Note that as the task is well defined, $|HB^{rel}_{B\cup S_M\cup CS}|$ is finite. Let GHS be the set of ground atoms which occur in the head of a rule in $\mathcal{M}^n(T)\cup\{\texttt{check\_violating}\}$. Let AS_IDs be the set of all terms $\texttt{as\_id}$ for which the fact $\texttt{as(as\_id)}$ occurs in $\mathcal{M}^n(T)\cup\{\texttt{check\_violating}\}$.

   We define a set $HB^{max} = GHS \cup \{\texttt{ctx}(\texttt{e}_{\texttt{id}}, \texttt{as\_id}) \mid e \in E^+ \cup E^-, \texttt{as\_id} \in AS\_IDs\} \cup \{\texttt{in\_as}(\texttt{a}, \texttt{as\_id}) \mid \texttt{a} \in HB^{rel}_{B\cup S_M\cup CS}, \texttt{as\_id} \in AS\_IDs\} \cup \{\texttt{w}(\texttt{wt}, \texttt{lev}, \texttt{args}(\texttt{t}_1, \ldots, \texttt{t}_\texttt{k}), \texttt{as\_id}) \mid {:\sim}\ \texttt{body}.[\texttt{wt@lev}, \texttt{t}_1, \ldots, \texttt{t}_\texttt{k}] \in ground^{rel}(B \cup S_M \cup CS), \texttt{as\_id} \in AS\_IDs\}$.

   Assume for contradiction that $\exists \texttt{atom} \in HB^{rel}_{\mathcal{M}^n(T)\cup\{\texttt{check\_violating}\}}$ such that $\texttt{atom} \notin HB^{max}$. As there are only three types of rules in $\mathcal{M}^n(T) \cup \{\texttt{check\_violating}\}$ that contain variables, there are only three possible cases:

   (a) $\exists R \in non\_weak(B \cup S_M \cup CS)$ such that there is a ground instance g of $\mathcal{R}_{\texttt{in\_as}}(R)$ such that $\texttt{atom} \in heads(g)$ and $body^+(g) \subseteq HB^{max}$.

   $\Rightarrow \exists R \in B \cup S_M \cup CS$ such that there is a ground instance g of R for which $\exists \texttt{a} \in heads(g)$ such that $\texttt{a} \notin HB^{rel}_{B\cup S_M\cup CS}$ and $body^+(g) \subseteq \{\texttt{a}_1 \mid \texttt{in\_as}(\texttt{a}_1, \texttt{a}_2) \in HB^{max}\}$.

   $\Rightarrow \exists R \in B \cup S_M \cup CS$ such that there is a ground instance g of R for which $\exists \texttt{a} \in heads(g)$ such that $\texttt{a} \notin HB^{rel}_{B\cup S_M\cup CS}$ and $body^+(g) \subseteq HB^{rel}_{B\cup S_M\cup CS}$. Contradiction, by the definition of $HB^{rel}_{B\cup S_M\cup CS}$.

(b) $\texttt{atom} \in \mathit{heads}(\texttt{1\{ctx(e}^{\texttt{1}}_{\texttt{id}}\texttt{, as\_id)}, \ldots, \texttt{ctx(e}^{\texttt{n}}_{\texttt{id}}\texttt{, as\_id)\}1 :- as(as\_id).})$, for some term $\texttt{as\_id}$ such that $\texttt{as(as\_id)} \in HB^{max}$ (where $\{e^1, \ldots, e^n\} = E^+ \cup E^-$). This cannot be the case, as HB^{max} was assumed to contain $\texttt{ctx(e}_{\texttt{id}}\texttt{, as\_id)}$ for each $e \in E^+ \cup E^-$ and each $\texttt{as\_id} \in AS\_IDs$.

(c) $\exists W \in \mathit{weak}(B \cup S_M)$ such that $\mathcal{R}_{weak}(W, \texttt{as\_ids})$ has a ground instance g such that $\texttt{atom} \in \mathit{heads}(g)$ and $\mathit{body}^+(g) \subseteq HB^{max}$.

$\Rightarrow \texttt{atom} = \texttt{w(wt, lev, args(t}_{\texttt{1}}, \ldots, \texttt{t}_{\texttt{k}}\texttt{), as\_id)}$ and $\exists W \in \mathit{weak}(B \cup S_M)$ such that W has a ground instance g such that $\mathit{body}^+(g) \subseteq HB^{rel}_{B \cup S_M \cup CS}$ and $\mathit{tail}(g) = [\texttt{wt@lev}, \texttt{t}_{\texttt{1}}, \ldots, \texttt{t}_{\texttt{k}}]$.

$\Rightarrow \texttt{atom} = \texttt{w(wt, lev, args(t}_{\texttt{1}}, \ldots, \texttt{t}_{\texttt{k}}\texttt{), as\_id)}$ and $\exists g \in \mathit{weak}(\mathit{ground}^{rel}(B \cup S_M \cup CS))$ such that and $\mathit{tail}(g) = [\texttt{wt@lev}, \texttt{t}_{\texttt{1}}, \ldots, \texttt{t}_{\texttt{k}}]$. Contradiction, as HB^{max} would contain such an $\texttt{atom}$.

Hence, there are no atoms in $HB^{rel}_{\mathcal{M}^n(T) \cup \{\texttt{check\_violating}\}} \subseteq HB^{max}$. So as HB^{max} is finite, the program $\mathcal{M}^n(T) \cup \{\texttt{check\_violating}\}$ must have a finite relevant grounding.

2. By part 1, $|HB^{rel}_{\mathcal{M}^n(T) \cup \{\texttt{check\_violating}\}}|$ is finite. Hence, $|HB^{rel}_{\mathcal{M}^n(T)}|$ is finite. Hence $|HB^{rel}_{\mathcal{M}^n(T) \cup \{constraint(H) | H \in HS\}}|$ is also finite (the extra constraints cannot possibly add anything to the fixpoint, as they have no head atoms).

□

Proofs from Chapter 7

Theorem 7.4. Let T be an $ILP^{context}_{LOAS}$ task, and H be a hypothesis. Consider the program $P = \{\texttt{0\{check\_violating\}1.}\} \cup \mathcal{M}(T)$.

1. $\exists A \in AS(P)$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ if and only if $H \in \mathcal{P}(T)$.

2. For any $A \in AS(P)$ and any $e \in E^-$, if $\texttt{v\_i(e}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$ then $\mathcal{M}^{-1}_{vi}(A)$ is a violating interpretation of H with respect to e.

3. For any $A \in AS(P)$ and any $o \in O^c$, if $\texttt{v\_p(o}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$ then $\mathcal{M}^{-1}_{vp}(A)$ is a violating pair of interpretations of H with respect to o.

4. $\exists A \in AS(P)$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ and $\texttt{violating} \in A$ if and only if $H \in \mathcal{V}(T)$.

Proof.

1. By Theorem 6.10, we know that $\exists A \in AS(\mathcal{M}(T))$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ if and only if $H \in \mathcal{P}(T)$ and that $\exists A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ if and only if $H \in \mathcal{V}(T)$.

   Hence, $\exists A \in AS(\mathcal{M}(T) \cup \{\texttt{0\{check\_violating\}1.}\})$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ if and only if $H \in \mathcal{P}(T)$ or $H \in \mathcal{V}(T)$.

   Hence, $\exists A \in AS(\mathcal{M}(T) \cup \{\texttt{0\{check\_violating\}1.}\})$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ if and only if $H \in \mathcal{P}(T)$ (as $\mathcal{V}(T) \subseteq \mathcal{P}(T)$).

2. By Theorem 6.10 (part (2)), we know that $\forall e \in E^-$, $\forall A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$, if $\texttt{v\_i(e}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$ then $\mathcal{M}^{-1}_{as}(A, \texttt{v1})$ is an accepting answer set of e wrt $B \cup H$ (i.e. it is a violating interpretation of H wrt e).

   Hence, $\forall e \in E^-$, $\forall A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$, if $\texttt{v\_i(e}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$ then $\mathcal{M}^{-1}_{vi}(A)$ is a violating interpretation of H wrt e.

   As the answer sets of $\mathcal{M}(T) \cup \{\texttt{0\{check\_violating\}1.}\}$ are the answer sets of $\mathcal{M}(T)$ and $\mathcal{M}(T) \cup \{\texttt{check\_violating.}\}$ combined, it remains to show that $\forall e \in E^-$, $\forall A \in AS(\mathcal{M}(T))$, if $\texttt{v\_i(e}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$ then $\mathcal{M}^{-1}_{as}(A, \texttt{v1})$ is an accepting answer set of e wrt $B \cup H$. Hence, it suffices to show that no answer set of $\mathcal{M}(T)$ can contain $\texttt{v\_i(e}_{\texttt{id}}\texttt{)}$. This is the case, because the only rule for $\texttt{v\_i(e}_{\texttt{id}}\texttt{)}$ (in $ground(\mathcal{M}(T))$) contains $\texttt{cov(e}_{\texttt{id}}\texttt{, v1)}$ as a positive body literal, the only rule for $\texttt{cov(e}_{\texttt{id}}\texttt{, v1)}$ contains $\texttt{ctx(e}_{\texttt{id}}\texttt{, v1)}$ as a positive body literal, the only rule for $\texttt{ctx(e}_{\texttt{id}}\texttt{, v1)}$ contains $\texttt{as(v1)}$ as a positive body literal, and the only rule for $\texttt{as(v1)}$ contains $\texttt{check\_violating}$ as a positive body literal. As there is no rule for $\texttt{check\_violating}$ in $ground(\mathcal{M}(T))$, A cannot contain $\texttt{v\_i(e}_{\texttt{id}}\texttt{)}$ without having a non-empty unfounded subset. Hence, no answer set of $\mathcal{M}(T)$ contains $\texttt{v\_i(e}_{\texttt{id}}\texttt{)}$.

3. By Theorem 6.10 (part 3), we know that $\forall o \in O^c$, $\forall A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$, if $\texttt{v\_p(o}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$ then $\langle \mathcal{M}^{-1}_{as}(A, \texttt{v1}), \mathcal{M}^{-1}_{as}(A, \texttt{v2})\rangle$ is an accepting pair of answer sets of $inverse(o)$ wrt $B \cup H$.

   Hence, $\forall o \in O^c$, $\forall A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$, if $\texttt{v\_p(o}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$ then $\mathcal{M}^{-1}_{vp}(A)$ is a violating pair of interpretations of H wrt o.

   As the answer sets of $\mathcal{M}(T) \cup \{\texttt{0\{check\_violating\}1.}\}$ are the answer sets of $\mathcal{M}(T)$ and $\mathcal{M}(T) \cup \{\texttt{check\_violating.}\}$ combined, it remains to show that $\forall o \in O^c$, $\forall A \in AS(\mathcal{M}(T))$, if $\texttt{v\_p(o}_{\texttt{id}}\texttt{)} \in A$ and $H = \mathcal{M}^{-1}_{in\_h}(A)$ then $\langle \mathcal{M}^{-1}_{as}(A, \texttt{v1}), \mathcal{M}^{-1}_{as}(A, \texttt{v2})\rangle$ is an accepting pair of answer sets of $inverse(o)$ wrt $B \cup H$. Hence, it suffices to show that no answer set of $\mathcal{M}(T)$ can contain $\texttt{v\_p(o}_{\texttt{id}}\texttt{)}$. This is the case, because the only rule for $\texttt{v\_p(o}_{\texttt{id}}\texttt{)}$ (in $ground(\mathcal{M}(T))$) contains $\texttt{ord\_respected(o}_{\texttt{id}}\texttt{, v1, v2)}$ as a positive body literal, the only rule for $\texttt{ord\_respected(o}_{\texttt{id}}\texttt{, v1, v2)}$ contains an atom $\texttt{cov(e}^1_{\texttt{id}}\texttt{, v1)}$ as a positive body literal, the only rule for $\texttt{cov(e}^1_{\texttt{id}}\texttt{, v1)}$ contains $\texttt{ctx(e}^1_{\texttt{id}}\texttt{, v1)}$ as a positive body literal, the only rule for $\texttt{ctx(e}^1_{\texttt{id}}\texttt{, v1)}$ contains $\texttt{as(v1)}$ as a positive body literal, and the only rule for $\texttt{as(v1)}$ contains $\texttt{check\_violating}$ as a positive body literal. As there is no rule for $\texttt{check\_violating}$ in $ground(\mathcal{M}(T))$, A cannot contain $\texttt{v\_p(o}_{\texttt{id}}\texttt{)}$ without having a non-empty unfounded subset. Hence, no answer set of $\mathcal{M}(T)$ contains $\texttt{v\_p(o}_{\texttt{id}}\texttt{)}$.

4. Assume $\exists A \in AS(P)$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ and $\texttt{violating} \in A$

   $\Leftrightarrow \exists A \in AS(P)$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ and $\texttt{check\_violating} \in A$ (there is a constraint in $\mathcal{M}(T)$ that means that if $\texttt{check\_violating}$ is in A then $\texttt{violating}$ must be in A too, and every rule for $\texttt{violating}$ depends on $\texttt{check\_violating}$, hence, every answer set either contains both or neither of the two atoms).

   $\Leftrightarrow \exists A \in AS(\mathcal{M}(T) \cup \{\texttt{check\_violating.}\})$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$

   $\Leftrightarrow H \in \mathcal{V}(T)$ (by Theorem 6.10 part(4))

□

Theorem 7.7.

Let T be an $ILP^{context}_{LOAS}$ task with hypothesis space S_M and let $H \subseteq S_M$. Let S be a set of tuples of the form $\langle I, e, \texttt{int\_id}\rangle$, where I is an interpretation, e is a CDPI and $\texttt{int\_id}$ is a unique ground term.

$\mathcal{M}_{vio}(T, S) \cup \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}) \mid h \in H\}$ has exactly one answer set, which consists of:

- The atom $\texttt{in\_h}(\texttt{h}_{\texttt{id}})$ for each $h \in H$
- For each $\langle I, e, \texttt{int\_id}\rangle \in S$, the atoms:
  - $\texttt{int}(\texttt{int\_id})$
  - $\texttt{in\_int}(\texttt{a}, \texttt{int\_id})$ for each $\texttt{a} \in I$
  - $\texttt{mmr}(\texttt{a}, \texttt{int\_id})$ for each $\texttt{a} \in M(ground^{rel}(B \cup H \cup e_{ctx})^I)$
  - If $I \notin AS(B \cup H \cup e_{ctx})$, the atom $\texttt{not\_as}(\texttt{int\_id})$

Proof. Assume that $A \in AS(\mathcal{M}_{vio}(T, S) \cup \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}) \mid h \in H\})$.

$\Leftrightarrow A = M(ground^{rel}(\mathcal{M}_{vio}(T, S) \cup \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}) \mid h \in H\})^A)$

$\Leftrightarrow A = M(ground^{rel}(P_1)^A)$, where:

$$\begin{array}{rl}
P_1 = & \{\texttt{int}(\texttt{int\_id}).\mid \langle I, e, \texttt{int\_id}\rangle \in S\}\\
& \cup\{\texttt{in\_int}(\texttt{atom}, \texttt{int\_id}).\mid \texttt{atom} \in I, \langle I, e, \texttt{int\_id}\rangle \in S\}\\
& \cup\{\mathcal{M}_{reduct}(e_{ctx}, \texttt{int\_id}) \mid \langle I, e, \texttt{int\_id}\rangle \in S\}\\
& \cup\mathcal{M}_{reduct}(B, \texttt{INT\_ID})\\
& \cup\{\mathcal{A}(\mathcal{M}_{reduct}(R, \texttt{INT\_ID}), \texttt{in\_h}(\texttt{R}_{\texttt{id}})) \mid R \in S_M\}\\
& \cup\left\{\begin{array}{l}\texttt{not\_as(INT\_ID):-int(INT\_ID),in\_int(ATOM,INT\_ID), not mmr(ATOM,INT\_ID).}\\ \texttt{not\_as(INT\_ID):-int(INT\_ID), not in\_int(ATOM,INT\_ID),mmr(ATOM,INT\_ID).}\end{array}\right\}\\
& \cup\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h \in H\}
\end{array}$$

$\Leftrightarrow A = M(ground^{rel}(P_2)^A)$, where:

$$\begin{array}{rl}
P_2 = & \{\texttt{int}(\texttt{int\_id}).\mid \langle I, e, \texttt{int\_id}\rangle \in S\}\\
& \cup\{\texttt{in\_int}(\texttt{atom}, \texttt{int\_id}).\mid \texttt{atom} \in I, \langle I, e, \texttt{int\_id}\rangle \in S\}\\
& \cup\{\mathcal{M}_{reduct}(e_{ctx}, \texttt{int\_id}) \mid \langle I, e, \texttt{int\_id}\rangle \in S\}\\
& \cup\mathcal{M}_{reduct}(B \cup H, \texttt{INT\_ID})\\
& \cup\left\{\begin{array}{l}\texttt{not\_as(INT\_ID):-int(INT\_ID),in\_int(ATOM,INT\_ID), not mmr(ATOM,INT\_ID).}\\ \texttt{not\_as(INT\_ID):-int(INT\_ID), not in\_int(ATOM,INT\_ID),mmr(ATOM,INT\_ID).}\end{array}\right\}\\
& \cup\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h \in H\}
\end{array}$$

$\Leftrightarrow A = M(ground^{rel}(P_3)^A)$, where:

$$\begin{aligned}P_3 = \;& \{\mathtt{int(int\_id).} \mid \langle I, e, \mathtt{int\_id}\rangle \in S\}\\ & \cup\{\mathtt{in\_int(atom, int\_id).} \mid \mathtt{atom} \in I, \langle I, e, \mathtt{int\_id}\rangle \in S\}\\ & \cup\{\mathcal{M}_{reduct}(B \cup H \cup e_{ctx}, \mathtt{int\_id}) \mid \langle I, e, \mathtt{int\_id}\rangle \in S\}\\ & \cup \left\{\begin{array}{l}\mathtt{not\_as(INT\_ID)\text{:-}\,int(INT\_ID), in\_int(ATOM, INT\_ID),\ not\ mmr(ATOM, INT\_ID).}\\ \mathtt{not\_as(INT\_ID)\text{:-}\,int(INT\_ID),\ not\ in\_int(ATOM, INT\_ID), mmr(ATOM, INT\_ID).}\end{array}\right\}\\ & \cup\{\mathtt{in\_h(h_{id}).} \mid h \in H\}\end{aligned}$$

$\Leftrightarrow A \cap HB_{P_4} = M(ground^{rel}(P_4)^{(A \cap HB_{P_4})})$, where:

$$\begin{aligned}P_4 = \;& \{\mathtt{int(int\_id).} \mid \langle I, e, \mathtt{int\_id}\rangle \in S\}\\ & \cup\{\mathtt{in\_int(atom, int\_id).} \mid \mathtt{atom} \in I, \langle I, e, \mathtt{int\_id}\rangle \in S\}\\ & \cup\{\mathcal{M}_{reduct}(B \cup H \cup e_{ctx}, \mathtt{int\_id}) \mid \langle I, e, \mathtt{int\_id}\rangle \in S\}\\ & \cup\{\mathtt{in\_h(h_{id}).} \mid h \in H\}\end{aligned}$$

$$\text{and } A \backslash HB_{P_4} = \left\{ \mathtt{not\_as(int\_id)} \;\middle|\; \begin{array}{c}\langle I, e, \mathtt{int\_id}\rangle \in S,\\ \{\mathtt{atom} \mid \mathtt{in\_int(atom, int\_id)} \in A\}\\ \neq\\ \{\mathtt{atom} \mid \mathtt{mmr(atom, int\_id)} \in A\}\end{array}\right\}$$

$\Leftrightarrow A \cap HB_{P_5} = M(P_5^{(A \cap HB_{P_5})})$, where:

$$\begin{aligned}P_5 = \;& \{\mathtt{int(int\_id).} \mid \langle I, e, \mathtt{int\_id}\rangle \in S\}\\ & \cup\{\mathtt{in\_int(atom, int\_id).} \mid \mathtt{atom} \in I, \langle I, e, \mathtt{int\_id}\rangle \in S\}\\ & \cup\{\mathcal{M}_{reduct}(ground^{rel}(B \cup H \cup e_{ctx}), \mathtt{int\_id}) \mid \langle I, e, \mathtt{int\_id}\rangle \in S\}\\ & \cup\{\mathtt{in\_h(h_{id}).} \mid h \in H\}\end{aligned}$$

$$\text{and } A \backslash HB_{P_5} = \left\{ \mathtt{not\_as(int\_id)} \;\middle|\; \begin{array}{c}\langle I, e, \mathtt{int\_id}\rangle \in S,\\ \{\mathtt{atom} \mid \mathtt{in\_int(atom, int\_id)} \in A\}\\ \neq\\ \{\mathtt{atom} \mid \mathtt{mmr(atom, int\_id)} \in A\}\end{array}\right\}$$

$\Leftrightarrow A = A^* \cup \{\mathtt{a} \in A_{\mathtt{int\_id}} \mid \langle I, e, \mathtt{int\_id}\rangle \in S\}$, where:

$$A^* = \{\mathtt{in\_h(h_{id}).} \mid h \in H\} \cup \left\{ \mathtt{not\_as(int\_id)} \;\middle|\; \begin{array}{c}\langle I, e, \mathtt{int\_id}\rangle \in S,\\ \{\mathtt{atom} \mid \mathtt{in\_int(atom, int\_id)} \in A\}\\ \neq\\ \{\mathtt{atom} \mid \mathtt{mmr(atom, int\_id)} \in A\}\end{array}\right\}$$

and for each $\langle I, e, \mathtt{int\_id}\rangle \in S$:

$$A_{\mathtt{int\_id}} = M\left(\begin{array}{l}\{\mathtt{int(int\_id).}\}\\ \quad \cup \{\mathtt{in\_int(atom, int\_id).} \mid \mathtt{atom} \in I\}\\ \quad \cup \mathcal{M}_{reduct}(ground^{rel}(B \cup H \cup e_{ctx}), \mathtt{int\_id})\end{array}\right)$$

$$
= M\left(\begin{array}{l}
\{\texttt{int(int\_id).}\}\\
\quad\cup\{\texttt{in\_int(atom,int\_id).}\mid \texttt{atom}\in I\}\\
\quad\cup\left\{\begin{array}{l}\texttt{mmr(head(R),int\_id):-}\\ \quad\texttt{int(int\_id)},\mathcal{R}(\texttt{body}^{+}\texttt{(R)},\texttt{mmr},\texttt{int\_id}),\\ \quad\mathcal{NR}(\texttt{body}^{-}\texttt{(R)},\texttt{in\_int},\texttt{int\_id}).\end{array}\middle|\begin{array}{c}R\in ground^{rel}(B\cup H\cup e_{ctx}),\\ R\text{ is normal}\end{array}\right\}\\
\quad\cup\left\{\begin{array}{l}\texttt{mmr}(\bot,\texttt{int\_id})\texttt{:-}\\ \quad\texttt{int(int\_id)},\mathcal{R}(\texttt{body}^{+}\texttt{(R)},\texttt{mmr},\texttt{int\_id}),\\ \quad\mathcal{NR}(\texttt{body}^{-}\texttt{(R)},\texttt{in\_int},\texttt{int\_id}).\end{array}\middle|\begin{array}{c}R\in ground^{rel}(B\cup H\cup e_{ctx}),\\ R\text{ is a hard constraint}\end{array}\right\}\\
\quad\cup\left\{\begin{array}{l}\texttt{mmr}(\bot,\texttt{int\_id})\texttt{:-}\\ \quad\texttt{int(int\_id)},\mathcal{R}(\texttt{body}^{+}\texttt{(R)},\texttt{mmr},\texttt{int\_id}),\\ \quad\mathcal{NR}(\texttt{body}^{-}\texttt{(R)},\texttt{in\_int},\texttt{int\_id}),\\ \quad\texttt{u}+\texttt{1}\{\texttt{in\_int}(\texttt{h}_1,\texttt{int\_id}),\ldots,\texttt{in\_int}(\texttt{h}_\texttt{n},\texttt{int\_id})\}.\\ \texttt{mmr}(\bot,\texttt{int\_id})\texttt{:-}\\ \quad\texttt{int(int\_id)},\mathcal{R}(\texttt{body}^{+}\texttt{(R)},\texttt{mmr},\texttt{int\_id}),\\ \quad\mathcal{NR}(\texttt{body}^{-}\texttt{(R)},\texttt{in\_int},\texttt{int\_id}),\\ \quad\{\texttt{in\_int}(\texttt{h}_1,\texttt{int\_id}),\ldots,\texttt{in\_int}(\texttt{h}_\texttt{n},\texttt{int\_id})\}\texttt{l}-\texttt{1}.\end{array}\middle|\begin{array}{c}R\in ground^{rel}(B\cup H\cup e_{ctx}),\\ head(R)=\texttt{l}\{\texttt{h}_1,\ldots,\texttt{h}_\texttt{n}\}\texttt{u}\end{array}\right\}\\
\quad\cup\left\{\begin{array}{l}\texttt{mmr}(\texttt{h}_\texttt{i},\texttt{int\_id})\texttt{:-}\\ \quad\texttt{int(int\_id)},\mathcal{R}(\texttt{body}^{+}\texttt{(R)},\texttt{mmr},\texttt{int\_id}),\\ \quad\mathcal{NR}(\texttt{body}^{-}\texttt{(R)},\texttt{in\_int},\texttt{int\_id}),\\ \quad\texttt{l}\{\texttt{in\_int}(\texttt{h}_1,\texttt{int\_id}),\ldots,\texttt{in\_int}(\texttt{h}_\texttt{n},\texttt{int\_id})\}\texttt{u}.\end{array}\middle|\begin{array}{c}R\in ground^{rel}(B\cup H\cup e_{ctx}),\\ head(R)=\texttt{l}\{\texttt{h}_1,\ldots,\texttt{h}_\texttt{n}\}\texttt{u},\\ i\in[1,n],\\ \texttt{h}_\texttt{i}\in I\end{array}\right\}
\end{array}\right)
$$

$$
= M\left(\begin{array}{l}
\left\{\begin{array}{l}\texttt{mmr(head(R),int\_id):-}\\ \quad\mathcal{R}(\texttt{body}^{+}\texttt{(R)},\texttt{mmr},\texttt{int\_id}).\end{array}\middle|\begin{array}{c}R\in ground^{rel}(B\cup H\cup e_{ctx}),\\ body^{-}(R)\cap I=\emptyset,\\ R\text{ is normal}\end{array}\right\}\\
\quad\cup\left\{\begin{array}{l}\texttt{mmr}(\bot,\texttt{int\_id})\texttt{:-}\\ \quad\mathcal{R}(\texttt{body}^{+}\texttt{(R)},\texttt{mmr},\texttt{int\_id}).\end{array}\middle|\begin{array}{c}R\in ground^{rel}(B\cup H\cup e_{ctx}),\\ body^{-}(R)\cap I=\emptyset,\\ R\text{ is a hard constraint}\end{array}\right\}\\
\quad\cup\left\{\begin{array}{l}\texttt{mmr}(\bot,\texttt{int\_id})\texttt{:-}\\ \quad\mathcal{R}(\texttt{body}^{+}\texttt{(R)},\texttt{mmr},\texttt{int\_id}).\end{array}\middle|\begin{array}{c}R\in ground^{rel}(B\cup H\cup e_{ctx}),\\ body^{-}(R)\cap I=\emptyset,\\ head(R)=\texttt{l}\{\texttt{h}_1,\ldots,\texttt{h}_\texttt{n}\}\texttt{u},\\ |heads(R)\cap I|>u\text{ or }|heads(R)\cap I|<l\end{array}\right\}\\
\quad\cup\left\{\begin{array}{l}\texttt{mmr}(\texttt{h}_\texttt{i},\texttt{int\_id})\texttt{:-}\\ \quad\mathcal{R}(\texttt{body}^{+}\texttt{(R)},\texttt{mmr},\texttt{int\_id})\end{array}\middle|\begin{array}{c}R\in ground^{rel}(B\cup H\cup e_{ctx}),\\ body^{-}(R)\cap I=\emptyset,\\ head(R)=\texttt{l}\{\texttt{h}_1,\ldots,\texttt{h}_\texttt{n}\}\texttt{u},\\ l<|heads(R)\cap I|<u,\\ i\in[1,n],\texttt{h}_\texttt{i}\in I\end{array}\right\}
\end{array}\right)
$$

$$
\begin{array}{l}
\quad\cup\{\texttt{int(int\_id)}\}\cup\{\texttt{in\_int(atom,int\_id)}\mid\texttt{atom}\in I\}\\
=M(\mathcal{R}(ground^{rel}(B\cup H\cup e^{ctx})^{I},\texttt{mmr},\texttt{int\_id}))\cup\{\texttt{int(int\_id)}\}\cup\{\texttt{in\_int(atom,int\_id)}\mid\texttt{atom}\in I\}\\
=\{\texttt{mmr(a,int\_id)}\mid\texttt{a}\in M(ground^{rel}(B\cup H\cup e^{ctx})^{I})\}\cup\{\texttt{int(int\_id)}\}\cup\{\texttt{in\_int(atom,int\_id)}\mid\texttt{atom}\in I\}
\end{array}
$$

Hence, $\mathcal{M}_{vio}(T,S)\cup\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}).\mid h\in S_M\}$ has exactly one answer set: $\{\texttt{in\_h}(\texttt{h}_{\texttt{id}})\mid h\in H\}\cup\{\texttt{not\_as(int\_id)}|I\notin AS(B\cup H\cup e_{ctx}),\langle I,e,\texttt{int\_id}\rangle\in S\}\cup\{\texttt{mmr(a,int\_id)}\mid\langle I,e,\texttt{int\_id}\rangle\in S,\texttt{a}\in M(ground^{rel}(B\cup H\cup e^{ctx})^{I})\}\cup\{\texttt{int(int\_id)}\mid\langle I,e,\texttt{int\_id}\rangle\in S\}\cup\{\texttt{in\_int(a,int\_id)}\mid\langle I,e,\texttt{int\_id}\rangle\in S,\texttt{a}\in I\}$.

□

Theorem 7.8.

Let T be the task $\langle B, S_M, E\rangle$, let S_1 be a list of tuples $[\langle I^1, e^1, \texttt{int\_id}^1\rangle, \ldots, \langle I^n, e^n, \texttt{int\_id}^n\rangle]$, where each I^i is an interpretation, each e^i is a CDPI st I^i extends e^i and each $\texttt{int\_id}^i$ is a unique ground term, and let S_2 be a set of tuples of the form $\langle \texttt{p}_{\texttt{id}}, \texttt{int\_id}^i, \texttt{int\_id}^j, op\rangle$, where $\texttt{p}_{\texttt{id}}$ is a unique ground term, $i, j \in [1, n]$ and $op \in \{<, \leq, >, \geq, =, \neq\}$. Let $H \subseteq S_M$.

The program $\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}). \mid h \in H\} \cup \mathcal{M}_{vio}(T, S_1) \cup \mathcal{M}_{vp}(T, S_2)$ has exactly one answer set, consisting of:

- All atoms in the unique answer set of $\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}). \mid h \in H\} \cup \mathcal{M}_{vio}(T, S_1)$.
- For each $\langle \texttt{p}_{\texttt{id}}, \texttt{int\_id}^i, \texttt{int\_id}^j, op\rangle \in S_2$:
  - $\texttt{vio\_w}(\texttt{wt}, \texttt{lv}, \texttt{args}(\texttt{t}_1, \ldots, \texttt{t}_n), \texttt{int\_id}^i)$ (resp. $\texttt{vio\_w}(\texttt{wt}, \texttt{lv}, \texttt{args}(\texttt{t}_1, \ldots, \texttt{t}_n), \texttt{int\_id}^j)$) for each weak constraint $W \in ground(B \cup H)$, with tail $[\texttt{wt@lv}, \texttt{t}_1, \ldots, \texttt{t}_n]$ such that $body(W)$ is satisfied by I^i (resp. I^j).
  - $\texttt{v\_dom\_lv}(\texttt{int\_id}^i, \texttt{int\_id}^j, \texttt{lev})$ (resp. $\texttt{v\_dom\_lv}(\texttt{int\_id}^j, \texttt{int\_id}^i, \texttt{lev})$) for each level $\texttt{lev}$ that occurs in $B \cup S_M$ such that $(B \cup H)^{I^i}_{lev} < (B \cup H)^{I^j}_{lev}$ (resp. $(B \cup H)^{I^i}_{lev} > (B \cup H)^{I^j}_{lev}$).
  - $\texttt{v\_dom}(\texttt{int\_id}^i, \texttt{int\_id}^j)$ (resp. $\texttt{v\_dom}(\texttt{int\_id}^j, \texttt{int\_id}^i)$) if $I^i \succ_{B \cup H} I^j$ (resp. $I^j \succ_{B \cup H} I^i$).
  - $\texttt{vp\_not\_resp}(\texttt{p}_{\texttt{id}})$ if I^i is an accepting answer set of e^i wrt $B \cup H$, I^j is an accepting answer set of e^j wrt $B \cup H$ and $\langle I^i, I^j, op\rangle \notin ord(B \cup H, AS(B \cup H \cup e^i_{ctx}) \cup AS(B \cup H \cup e^j_{ctx}))$

Proof.

Assume $A \in AS\left(\begin{array}{c}\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}). \mid h \in H\}\\ \cup\, \mathcal{M}_{vio}(T, S_1)\\ \cup\, \mathcal{M}_{vp}(T, S_2)\end{array}\right)$

$\Leftrightarrow$ $\exists A'$ such that $A \in AS\left(\begin{array}{c}\{\texttt{a}. \mid \texttt{a} \in A'\}\\ \cup\, \mathcal{M}_{vp}(T, S_2)\end{array}\right)$, and $A' \in AS\left(\begin{array}{c}\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}). \mid h \in H\}\\ \cup\, \mathcal{M}_{vio}(T, S_1)\end{array}\right)$ (by Corollary 2.14).

$\Leftrightarrow A \in AS\left(\begin{array}{c}\{\texttt{a}. \mid \texttt{a} \in A_1\}\\ \cup\, \mathcal{M}_{vp}(T, S_2)\end{array}\right)$, where A_1 is the unique answer set of $\{\texttt{in\_h}(\texttt{h}_{\texttt{id}}). \mid h \in H\} \cup \mathcal{M}_{vio}(T, S_1)$ (by Theorem 7.7).

$\Leftrightarrow \exists A' \in AS\left(\begin{array}{c}\{\texttt{a}. \mid \texttt{a} \in A_1\}\\ \cup \{\mathcal{M}^i_{vp}(T, S_2) \mid i \in [1, 2]\}\end{array}\right)$,

such that $A \in AS\left(\begin{array}{c}\{\texttt{a}. \mid \texttt{a} \in A'\}\\ \cup \{\mathcal{M}^i_{vp}(T, S_2) \mid i \in [3, 5]\}\end{array}\right)$ (by Corollary 2.14), where the $\mathcal{M}^i_{vp}$'s represent the (ordered) components of $\mathcal{M}_{vp}$ (in Meta-program 7.3)

$\Leftrightarrow \exists A' \in AS\left(\begin{array}{c}\{\texttt{a}. \mid \texttt{a} \in A_1\}\\ \cup \left\{\begin{array}{c|c}\mathcal{R}^{vio}_{weak}(W, \texttt{int\_id}^i) & \langle \texttt{p}_{\texttt{id}}, \texttt{int\_id}^i, \texttt{int\_id}^j, op\rangle \in S_2\\ \mathcal{R}^{vio}_{weak}(W, \texttt{int\_id}^j) & W \in weak(B \cup H)\end{array}\right\}\end{array}\right)$, such that $A \in AS\left(\begin{array}{l}\{\mathtt{a.} \mid \mathtt{a} \in A'\}\\ \quad \cup \{\mathcal{M}^i_{vp}(T, S_2) \mid i \in [3,5]\}\end{array}\right)$ (the removed rules contain an atom $\mathtt{in\_h}(\mathtt{h_{id}})$ that does not occur in the head of any rule in the program).

$\Leftrightarrow A \in AS\left(\begin{array}{l}\{\mathtt{a.} \mid \mathtt{a} \in A_1 \cup A_2\}\\ \quad \cup \{\mathcal{M}^i_{vp}(T, S_2) \mid i \in [3,5]\}\end{array}\right)$,

$$\text{where } A_2 = \left\{\mathtt{vio\_w}(\mathtt{wt}, \mathtt{lv}, \mathtt{args}(\mathtt{t_1} \ldots, \mathtt{t_m}), \mathtt{int\_id^i}) \;\middle|\; \begin{array}{c}\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2,\\ W \in weak(B \cup H),\\ I^i \text{ satisfies } body(W),\\ tail(W) = [\mathtt{wt@lv}, \mathtt{t_1}, \ldots, \mathtt{t_m}]\end{array}\right\}$$
$$\cup \left\{\mathtt{vio\_w}(\mathtt{wt}, \mathtt{lv}, \mathtt{args}(\mathtt{t_1} \ldots, \mathtt{t_m}), \mathtt{int\_id^j}) \;\middle|\; \begin{array}{c}\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2,\\ W \in weak(B \cup H),\\ I^j \text{ satisfies } body(W),\\ tail(W) = [\mathtt{wt@lv}, \mathtt{t_1}, \ldots, \mathtt{t_m}]\end{array}\right\}$$

$\Leftrightarrow A \in AS\left(\begin{array}{l}\{\mathtt{a.} \mid \mathtt{a} \in A_1 \cup A_2 \cup A_3\}\\ \quad \cup \{\mathcal{M}^i_{vp}(T, S_2) \mid i \in [4,5]\}\end{array}\right)$, where A_3 is the set of atoms $\mathtt{v\_dom\_lv}(\mathtt{int\_id^i}, \mathtt{int\_id^j}, \mathtt{lev})$ (resp. $\mathtt{v\_dom\_lv}(\mathtt{int\_id^j}, \mathtt{int\_id^i}, \mathtt{lev})$) for each level $\mathtt{lev}$ that occurs in $B \cup S_M$ such that $(B \cup H)^{I^i}_{lev} < (B \cup H)^{I^j}_{lev}$ (resp. $(B \cup H)^{I^i}_{lev} > (B \cup H)^{I^j}_{lev}$), where $\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2$

$\Leftrightarrow A \in AS\left(\begin{array}{l}\{\mathtt{a.} \mid \mathtt{a} \in A_1 \cup A_2 \cup A_3 \cup A_4\}\\ \quad \cup \mathcal{M}^5_{vp}(T, S_2)\end{array}\right)$,

$$\begin{aligned}
\text{where } A_4 &= \left\{\mathtt{v\_dom}(\mathtt{int\_id^i}, \mathtt{int\_id^j}) \;\middle|\; \begin{array}{c}\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2,\\ \mathtt{v\_dom\_lv}(\mathtt{int\_id^i}, \mathtt{int\_id^j}, \mathtt{lev}) \in A_3,\\ \forall \mathtt{lev'} > \mathtt{lev}, \mathtt{v\_dom\_lv}(\mathtt{int\_id^i}, \mathtt{int\_id^j}, \mathtt{lev'}) \notin A_3,\\ \forall \mathtt{lev'} > \mathtt{lev}, \mathtt{v\_dom\_lv}(\mathtt{int\_id^j}, \mathtt{int\_id^i}, \mathtt{lev'}) \notin A_3\end{array}\right\}\\
&\quad \cup \left\{\mathtt{v\_dom}(\mathtt{int\_id^j}, \mathtt{int\_id^i}) \;\middle|\; \begin{array}{c}\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2,\\ \mathtt{v\_dom\_lv}(\mathtt{int\_id^j}, \mathtt{int\_id^i}, \mathtt{lev}) \in A_3,\\ \forall \mathtt{lev'} > \mathtt{lev}, \mathtt{v\_dom\_lv}(\mathtt{int\_id^i}, \mathtt{int\_id^j}, \mathtt{lev'}) \notin A_3,\\ \forall \mathtt{lev'} > \mathtt{lev}, \mathtt{v\_dom\_lv}(\mathtt{int\_id^j}, \mathtt{int\_id^i}, \mathtt{lev'}) \notin A_3\end{array}\right\}\\
&= \left\{\mathtt{v\_dom}(\mathtt{int\_id^i}, \mathtt{int\_id^j}) \;\middle|\; \begin{array}{c}\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2,\\ (B \cup H)^{I^i}_{lev} < (B \cup H)^{I^j}_{lev},\\ \forall \mathtt{lev'} > \mathtt{lev}, (B \cup H)^{I^i}_{lev} = (B \cup H)^{I^j}_{lev}\end{array}\right\}\\
&\quad \cup \left\{\mathtt{v\_dom}(\mathtt{int\_id^j}, \mathtt{int\_id^i}) \;\middle|\; \begin{array}{c}\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2,\\ (B \cup H)^{I^j}_{lev} < (B \cup H)^{I^i}_{lev},\\ \forall \mathtt{lev'} > \mathtt{lev}, (B \cup H)^{I^i}_{lev} = (B \cup H)^{I^j}_{lev}\end{array}\right\}\\
&= \left\{\mathtt{v\_dom}(\mathtt{int\_id^i}, \mathtt{int\_id^j}) \;\middle|\; \begin{array}{c}\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2,\\ I^i \succ_{B \cup H} I^j\end{array}\right\}\\
&\quad \cup \left\{\mathtt{v\_dom}(\mathtt{int\_id^j}, \mathtt{int\_id^i}) \;\middle|\; \begin{array}{c}\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2,\\ I^j \succ_{B \cup H} I^i\end{array}\right\}
\end{aligned}$$

$\Leftrightarrow$ $A = A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5$, where A_5 is the set of atoms $\mathtt{vp\_not\_resp}(\mathtt{p_{id}})$ for each $\langle \mathtt{p_{id}}, \mathtt{int\_id^i}, \mathtt{int\_id^j}, op\rangle \in S_2$ such that I^i is an accepting answer set of e^i wrt $B \cup H$, I^j is an accepting answer set of e^j wrt $B \cup H$ and $\langle I^i, I^j, op\rangle \notin ord(B \cup H, AS(B \cup H \cup e^i_{ctx}) \cup AS(B \cup H \cup e^j_{ctx}))$ (the rules in $\mathcal{M}^5_{vp}$ depend on the operator op and are satisfied if and only if I^i is an accepting answer set of e^i wrt $B \cup H$ (guaranteed by the $\mathtt{not\_as}$ atoms in A_1), I^j is an accepting answer set of e^j wrt $B \cup H$ (again guaranteed by the $\mathtt{not\_as}$ atoms in A_1), and $\langle I^i, I^j, op\rangle \notin ord(B \cup H, AS(B \cup H \cup e^i_{ctx})$ (guaranteed by the $\mathtt{v\_dom}$ atoms in A_4)).

□

Theorem 7.9. Given an $ILP_{LOAS}^{context}$ task and a set of violating reasons VR, let AS be the set of optimal answer sets of $\mathcal{M}_{ILASP2}(T, VR)$

1. For any hypothesis H, $\exists A \in AS(\mathcal{M}_{ILASP2}(T, VR))$ such that $H = \mathcal{M}_{in\_h}^{-1}(A)$ if and only if $H \in \mathcal{P}(T)$ and $\forall vr \in VR$, vr is not a violating reason of H.
2. For any $A \in AS$ such that `violating` $\in A$, $extractVR(A)$ is a violating reason of $\mathcal{M}_{in\_h}^{-1}(A)$.
3. If no $A \in AS$ contains `violating`, then the set of optimal remaining hypotheses (none of which is violating) is exactly equal to the set $\{\mathcal{M}_{in\_h}^{-1}(A) \mid A \in AS\}$.

Proof.

1. We partition the $\mathcal{M}_{ILASP2}(T, VR)$ program such that P_1 is the program $\mathcal{M}(T) \cup \{$`0{check_violating}1.`$\}$ and P_2 is the remaining program.

   Let $H \subseteq S_M$. Assume that $\exists A \in AS(\mathcal{M}_{ILASP2}(T, VR))$ such that $\mathcal{M}_{in\_h}^{-1}(A) = H$.

   $\Leftrightarrow \exists A' \in AS(P_1)$ such that $\mathcal{M}_{in\_h}^{-1}(A) = H$ and the program $\{$`a.` $\mid$ `a` $\in A'\} \cup P_2$ is satisfiable (by Corollary 2.14).

   $\Leftrightarrow \exists A' \in AS(P_1)$ such that $\mathcal{M}_{in\_h}^{-1}(A) = H$ and the program $\{$`in_h(h_id).` $\mid h \in H\} \cup P_2$ is satisfiable (as the other atoms in A' do not occur in the bodies of ground instances of rules in P_2)

   $\Leftrightarrow H \in \mathcal{P}(T)$ and the program $\{$`in_h(h_id).` $\mid h \in H\} \cup P_2$ is satisfiable (by Theorem 7.4 part (1))

   $\Leftrightarrow H \in \mathcal{P}(T)$ and the unique answer set of $\{$`in_h(h_id).` $\mid h \in H\} \cup \mathcal{M}_{vio}(T, S_1) \cup \mathcal{M}_{vp}(T, S_2)$ contains `not_as(I_id)` for each $\langle I, e\rangle \in VR$ and does not contain `vp_not_resp(o_id)` for any $\langle\langle I_1, I_2\rangle, o\rangle \in VR$

   $\Leftrightarrow H \in \mathcal{P}(T)$ and for each $\langle I, e\rangle \in VR$, I is not an accepting answer set of e wrt $B \cup H$ and for each $\langle\langle I_1, I_2\rangle, o\rangle \in VR$, $\langle I_1, I_2\rangle$ is not an accepting pair of answer sets of $inverse(o)$ (by Theorem 7.7 and Theorem 7.8)

   $\Leftrightarrow H \in \mathcal{P}(T)$ and $\forall vr \in VR$, vr is not a violating reason of H.

2. Let $A \in AS$ st `violating` $\in A$.

   Given the rules in $\mathcal{M}(T)$, as `violating` $\in A$, either $\exists e \in E^-$ such that `v_i(e_id)` $\in A$ or $\exists o \in O^c$ such that `v_p(o_id)` $\in A$. Hence, by Theorem 7.4 (points (2) and (3)), $extractVR(A)$ can extract a violating reason of $\mathcal{M}_{in\_h}^{-1}(A)$ from A.

3. Assume no $A \in AS$ contains `violating`.

   Case 1: $AS = \emptyset$

   $\Rightarrow \{H \in \mathcal{P}(T) \mid \forall vr \in VR,\ vr$ is not a violating reason of $H\} = \emptyset$

   $\Rightarrow$ the set of optimal remaining hypotheses is empty (and hence, is equal to $\{\mathcal{M}_{in\_h}^{-1}(A) \mid A \in AS\}$)

   Case 2: $AS \neq \emptyset$

$\Rightarrow \{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS\} = \{H \in \mathcal{P}^n(T) \mid \forall vr \in VR$, vr is not a violating reason of $H\}$ (where n is the smallest n for which this set is non-empty). This is the set of remaining hypotheses of length n (and no shorter remaining hypotheses exist).

$\Rightarrow \{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS\}$ is the set of optimal remaining hypotheses.

It remains to show that none of the optimal remaining hypotheses is violating. For any optimal remaining hypothesis H, $\exists A \in AS$ such that $\mathcal{M}^{-1}_{in\_h}(A) = H$. Hence, $\forall vr \in VR$, vr is not a violating reason of H. Hence the program $\{\texttt{in\_h(h}_{\texttt{id}}\texttt{).} \mid h \in H\} \cup P_2$ is satisfiable.

Assume for contradiction that $H \in \mathcal{V}(T)$. Then $\exists A' \in AS(P_1)$ such that $\texttt{violating} \in A'$ and $\mathcal{M}^{-1}_{in\_h}(A) = H$ (by Theorem 7.4 (4)). Hence, there is an answer set A'' of $P_1 \cup P_2$ such that $\texttt{violating} \in A''$ and $\mathcal{M}^{-1}_{in\_h}(A) = H$. But as this answer set contains `violating` and represents the same hypothesis H as the answer sets in AS, it must be more optimal than the answer sets in AS. This contradicts AS being the set of optimal answer sets of $\mathcal{M}_{ILASP2}(T, VR)$.

□

Proposition 7.10. Let T be any well-defined $ILP^{context}_{LOAS}$ task, and VR be a finite set of violating reasons. $ground^{rel}(\mathcal{M}_{ILASP2}(T, VR))$ is finite.

Proof.

To prove that $\mathcal{M}_{ILASP2}(T, VR)$ has a finite relevant grounding, it is sufficient to prove that $|HB^{rel}_{\mathcal{M}_{ILASP2}(T,VR)}|$ is finite.

Firstly note that the relevant Herbrand base of $\mathcal{M}_{ILASP2}(T, VR)$ is equal to the relevant Herbrand base of $\mathcal{M}(T) \cup \{\texttt{check\_violating.}\} \cup \mathcal{M}_{vio}(T, S_1) \cup \mathcal{M}_{vp}(T, S_2)$ (as the remaining rules are hard or weak constraints, and thus do not contribute to the relevant Herbrand base).

Consider the following partition of the program:

$P_1 = \mathcal{M}(T) \cup \{\texttt{check\_violating.}\}$

$P_2 = \{\texttt{in\_h(h}_{\texttt{id}}\texttt{).} \mid h \in S_M\} \cup \mathcal{M}_{vio}(T, S_1) \cup \mathcal{M}_{vp}(T, S_2)$

Due to the division in the language of the two programs (other than the `in_h` atoms, which are in the relevant Herbrand base of both programs), $HB^{rel}_{\mathcal{M}_{ILASP2}(T,VR)} = HB^{rel}_{P_1} \cup HB^{rel}_{P_2}$.

Note that as the task is well defined, $|HB^{rel}_{B \cup S_M \cup CS}|$ is finite, by Proposition 6.13, $HB^{rel}_{P_1}$ is finite, so it remains to show that $HB^{rel}_{P_2}$ is finite. There are three predicates which appear unground in the head of a rule in P_2: `not_as`, `mmr` and `vio_w`. As `vio_w` and `not_as` do not appear in the bodies of any rule with an unground head, it remains to show that there are a finite number of ground instances of `mmr` in $HB^{rel}_{P_2}$. For every rule R such that $\texttt{mmr(a, int\_id)}$ is a ground instance of $head(R)$, $\exists R' \in B \cup S_M \cup CS$ such that $body^+(R) = \mathcal{R}(body^+(R'), \texttt{mmr}, \texttt{int\_id})$ for some `int_id`. Hence, `a` must be in $HB^{rel}_{B \cup S_M \cup CS}$ (which is finite as the task is well defined). As the `int_id`'s are a finite set of ground terms, this means that $HB^{rel}_{P_2}$ has a finite relevant Herbrand base. □

Theorem 7.11. Let T be any well-defined $ILP^{context}_{LOAS}$ task.

1. $ILASP2(T)$ terminates in a finite time.

2. $ILASP2(T) = {}^{*}ILP^{context}_{LOAS}(T)$

Proof.

1. By Proposition 7.10, we know that every call to *solve* must terminate (as the groundings of the programs being solved are always finite). Hence to prove termination, it is sufficient to show that there can only be a finite number of iterations of the while loop in the ILASP2 algorithm. We show this by proving that in every iteration $|\{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS(\mathcal{M}_{ILASP2}(T, VR_0))\}| > |\{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS(\mathcal{M}_{ILASP2}(T, VR_1))\}|$ (where VR_0 is the value of VR at the start of the loop iteration and VR_1 is the value of VR at the end of the loop iteration). As the initial value of $|\{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS(\mathcal{M}_{ILASP2}(T, VR))\}|$ is at most $2^{|S_M|}$ (and is hence finite), this means that there must be a finite number of iterations.

   - We first show that $\{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS(\mathcal{M}_{ILASP2}(T, VR_1))\}$ is a subset of $|\{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS(\mathcal{M}_{ILASP2}(T, VR_0))\}$.

     Let H be a hypothesis and assume that $\exists A \in AS(\mathcal{M}_{ILASP2}(T, VR_1))$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$

     $\Rightarrow H \in \mathcal{P}(T)$ and $\forall vr \in VR_1$: vr is not a violating reason of H (by Theorem 7.9 (1))

     $\Rightarrow H \in \mathcal{P}(T)$ and $\forall vr \in VR_0$: vr is not a violating reason of H (as $VR_0 \subset VR_1$)

     $\Rightarrow \exists A \in AS(\mathcal{M}_{ILASP2}(T, VR_0))$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ (by Theorem 7.9 (1))

   - It remains to show that there is at least one H such that $\exists A \in AS(\mathcal{M}_{ILASP2}(T, VR_0))$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$ and $\nexists A \in AS(\mathcal{M}_{ILASP2}(T, VR_1))$ such that $H = \mathcal{M}^{-1}_{in\_h}(A)$.

     Let $H = \mathcal{M}^{-1}_{in\_h}(A)$, where A is the answer set used in ILASP2. Then $extractVR(A)$ is a violating reason of H (by Theorem 7.9 (2)), and hence VR_1 contains a vr that is a violating reason of H. Hence, $H \notin \{\mathcal{M}^{-1}_{in\_h}(A) \mid A \in AS(\mathcal{M}_{ILASP2}(T, VR_1))\}$ (by Theorem 7.9 (1)).

2. We first show that at every step through the while loop, VR is a set violating reasons of T.

   Base Case: Before the loop has been entered, $VR = \emptyset$

   Inductive Hypothesis: Let VR_0 be a set of violating reasons. If $VR = VR_0$ at the start of an interation through the loop, then VR_1, the value of VR after one iteration of the loop, is still a set of violating reasons of T.

   Proof of Inductive Hypothesis: `violating` $\in A$. Hence, by Theorem 7.9 (2), $extractVR(A)$ is a violating reason. Hence as $VR_1 = VR_0 \cup \{extractVR(A)\}$, VR_1 is a set of violating reasons.

   Hence at every step through the loop, VR is a set of violating reasons of T.

   When $ILASP2(T)$ terminates, `violating` $\notin A$ and hence no answer set in $AS^{*}(\mathcal{M}_{ILASP2}(T, VR))$ can contain `violating`. Hence by Theorem 7.9 (3), $\{\mathcal{M}^{-1}_{in\_h}(as) \mid as \in AS^{*}(\mathcal{M}_{ILASP2}(T, VR))\}$ is the set of optimal remaining hypotheses, none of which is violating. This means that they are the optimal inductive solutions of T.

□

Proposition 7.13. Let T be any well-defined $ILP^{context}_{LOAS}$ task with a hypothesis space S_M, and let H be any hypothesis $H \subseteq S_M$. $ground^{rel}(\mathcal{M}_{fre}(T, H))$ is finite.

Proof. Consider the set $S = \mathcal{R}(HB^{rel}_{B\cup S_M\cup CS}, \texttt{in\_as}, \texttt{1}) \cup \mathcal{R}(HB^{rel}_{B\cup S_M\cup CS}, \texttt{in\_as}, \texttt{2}) \cup \{\texttt{ctx}(\texttt{e}_{\texttt{id}}, \texttt{i}) \mid e \in E^+ \cup E^-, i \in [1,2]\} \cup \{\texttt{w}(\texttt{wt}, \texttt{lv}, \texttt{args}(\texttt{t}_1, \ldots, \texttt{t}_n), \texttt{i}) \mid W \in \mathit{ground}^{rel}(B \cup S_M \cup CS), \mathit{tail}(W) = [\texttt{wt@lv}, \texttt{t}_1, \ldots, \texttt{t}_n], i \in [1,2]\} \cup G$ (where G is the set of all ground atoms which appear in the heads of rules in $\mathcal{M}_{fre}(T,H)$).

This set is finite, as the task is well defined. Hence, it suffices to show that $HB^{rel}_{\mathcal{M}_{fre}(T,H)} \subseteq S$. But this is the case as there is no ground instance of any rule R in $\mathcal{M}_{fre}(T,H)$ such that $\mathit{body}^+(R) \subseteq S$ and $\mathit{heads}(R) \not\subseteq S$. $\square$

Theorem 7.15. Let T be an $ILP^{context}_{LOAS}$ task, and let H be a hypothesis.

1. If $H \in ILP^{context}_{LOAS}(T)$ then $findRelevantExample(T,H)$ returns `nil`
2. If $H \notin ILP^{context}_{LOAS}(T)$ then $findRelevantExample(T,H)$ returns an example that is relevant to H given T.

Proof. Let T be the task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$

We first show for each of the four types of examples, an example e is covered by H if and only if $e \notin findRelevantExamples(T,H)$.

Note that the program $\mathcal{M}_{fre}(T,H)$ is equal to the program P in Theorem 6.6 (where $AS_{ids} = \{\texttt{1}, \texttt{2}\}$, $Pair_{ids} = \{\langle \texttt{1}, \texttt{2}\rangle\}$, $E_{\texttt{1}} = E_{\texttt{2}} = E^+ \cup E^-$ and $O_{\langle \texttt{1},\texttt{2}\rangle} = O^b \cup \{inverse(o) \mid i \in O^c\}$).

- Let $e \in E^+ \cup E^-$

  Firstly, assume that e is accepted by $B \cup H$.

  $\Rightarrow \exists I$ such that I is an accepting answer set of e wrt $B \cup H$

  $\Rightarrow \exists A \in AS(\mathcal{M}_{fre}(T,H))$ such that $\exists I$ such that $\texttt{ctx}(\texttt{e}_{\texttt{id}}, \texttt{1}), \texttt{ctx}(\texttt{e}_{\texttt{id}}, \texttt{2}) \in A$, $\mathcal{M}^{-1}_{as}(A, \texttt{1}) = \mathcal{M}^{-1}_{as}(A, \texttt{2}) = I$, and I is an accepting answer set of e wrt $B \cup H$ (by Theorem 6.6 (1), with the list $[\langle I, e\rangle, \langle I, e\rangle]$)

  $\Rightarrow \exists A \in AS(\mathcal{M}_{fre}(T,H))$ such that $\texttt{cov}(\texttt{e}_{\texttt{id}}, \texttt{1}) \in A$ (by Theorem 6.6 (2))

  $\Rightarrow \mathcal{M}_{fre}(T,H) \models_b \texttt{cov}(\texttt{e}_{\texttt{id}}, \texttt{1})$

  On the other hand, assume $\mathcal{M}_{fre}(T,H) \models_b \texttt{cov}(\texttt{e}_{\texttt{id}}, \texttt{1})$

  $\Rightarrow \exists A \in AS(\mathcal{M}_{fre}(T,H))$ such that $\texttt{cov}(\texttt{e}_{\texttt{id}}, \texttt{1}) \in A$

  $\Rightarrow \exists A \in AS(\mathcal{M}_{fre}(T,H))$ such that $\mathcal{M}^{-1}_{as}(A)$ is an accepting answer set of e wrt $B \cup H$ (by Theorem 6.6 (2))

  $\Rightarrow B \cup H$ accepts e

- Let $o = \langle e^1, e^2, op\rangle \in O^b \cup \{inverse(o) \mid o \in O^c\}$

  Assume that $B \cup H$ accepts o

  $\Leftrightarrow \exists p = \langle I_1, I_2\rangle$ such that p is an accepting pair of answer sets of o wrt $B \cup H$

  $\Leftrightarrow \exists p = \langle I_1, I_2\rangle$ such that I_1 is an accepting answer set of e^1 wrt $B \cup H$, I_2 is an accepting answer set of e^2 wrt $B \cup H$, $\langle I_1, I_2, op\rangle \in \mathit{ord}(B \cup H, AS(B \cup H \cup e^1_{ctx}) \cup AS(B \cup H \cup e^2_{ctx}))$

  $\Leftrightarrow \exists A \in AS(\mathcal{M}_{fre}(T,H))$ such that $\texttt{cov}(\texttt{e}^1_{\texttt{id}}, \texttt{1}), \texttt{cov}(\texttt{e}^2_{\texttt{id}}, \texttt{2}) \in A$ and $\langle \mathcal{M}^{-1}_{as}(A, \texttt{1}), \mathcal{M}^{-1}_{as}(A, \texttt{2})\rangle$ is an accepting answer set of o wrt $B \cup H$ (by Theorem 6.6 (1) and (2))

  $\Leftrightarrow \exists A \in AS(\mathcal{M}_{fre}(T,H))$ such that $\texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{1}, \texttt{2}) \in A$ (by Theorem 6.6 (3))

  $\Leftrightarrow \mathcal{M}_{fre}(T,H) \models_b \texttt{ord\_respected}(\texttt{o}_{\texttt{id}}, \texttt{1}, \texttt{2})$

Hence, $CDPIs$ is the set of all CDPI examples e such that $e \in E^{+} \cup E^{-}$ and e is accepted by $B \cup H$. Similarly, $CDOEs$ is the set of all CDOE examples o such that either $o \in O^{b}$ and o is accepted by $B \cup H$ or $o \in O^{c}$ and $inverse(o)$ is accepted by $B \cup H$.

Hence the returned set is the set of all examples in $E^{+} \cup E^{-} \cup O^{b} \cup O^{c}$ that are not covered by H.

We can now prove the two properties of the theorem (about the $findRelevantExample$ method).

1. Let H be any hypothesis in $ILP_{LOAS}^{context}(T)$. Then H covers all examples in T, and hence, $findRelevantExamples(T, H)$ will return $\emptyset$. Hence, $findRelevantExample(T, H)$ will return `nil`.
2. Let $H \subseteq S_M$ such that $H \notin ILP_{LOAS}^{context}(T)$. Then there is at least one example e in T that H does not cover. Hence, the set returned by $findRelevantExamples(T, H)$ will be non-empty (and by the above will be a set of examples that H does not cover). Hence, $findRelevantExample(T, H)$ will return an example in T that H does not cover (i.e. an example that is relevant to H given T).

□

Theorem 7.16. ILASP2i terminates for any well-defined $ILP_{LOAS}^{context}$ task.

Proof. Assume that the task $T = \langle B, S_M, E\rangle$ is well-defined. Note that this also means that $\langle B, S_M, Relevant\rangle$ is well-defined. The soundness of ILASP2 (Theorem 7.11) can be used to show that H will always cover every example in *Relevant*; hence, at each step re must be an example which is in E but not in *Relevant*. As there are a finite number of examples in E, this means there can only be a finite number of iterations; hence, it remains to show that each iteration terminates. This is the case because, as $\langle B, S_M, Relevant\rangle$ is well defined, the call to ILASP2 terminates (Theorem 7.11) and $findRelevantExample$ terminates (Corollary 7.14). □

Theorem 7.17. Let T be a well-defined $ILP_{LOAS}^{context}$ task.

1. If T is satisfiable, then $ILASP2i(T)$ returns an optimal inductive solution of T.
2. If T is unsatisfiable, then $ILASP2i(T)$ returns `UNSATISFIABLE`.

Proof.

1. Assume that T is satisfiable. Then no matter which examples are added to *Relevant*, the call to $ILASP2$ will always return a hypothesis. So $ILASP2i$ cannot return `UNSATISFIABLE`. Hence, it must return a hypothesis (as it terminates by Theorem 7.16). This means that the while loop must terminate. For this to happen $findRelevantExample$ must return `nil`. This means that $H \in ILP_{LOAS}^{context}(T)$ (by Theorem 7.15 part(2)). It remains to show that H is optimal. Assume for contradiction that there is a hypothesis $H' \in ILP_{LOAS}^{context}(T)$ such that $|H'| < |H|$. Then as $Relevant \subseteq E$, $H' \in ILP_{LOAS}^{context}(\langle B, S_M, Relevant\rangle)$. Hence, by *Theorem* 7.11, $ILASP2(\langle B, S_M, Relevant\rangle))$ will not return H (contradiction!). Hence, H is an optimal inductive solution of T.
2. Assume that T is unsatisfiable. In this case, $findRelevantExample$ will never return `nil` (Theorem 7.15). Hence, the while condition will never be satisfied. As ILASP2i is guaranteed to terminate (Theorem 7.16), it must return `UNSATISFIABLE` (eventually, $\langle B, S_M, Relevant\rangle$ will become unsatisfiable).

□

B.1 Proofs and Meta-programs from Chapter 10

Lemma 10.2.

Let P be any ASP program whose weak constraints are independent, l be any integer and I_1 and I_2 be any interpretations. Then $\Delta^P_l(I_1, I_2) = \sum\limits_{W \in weak(P)} \Delta^W_l(I_1, I_2)$.

Proof.

$$\begin{array}{lcl}
\Delta^P_l(I_1, I_2) & = & P^{I_1}_l - P^{I_2}_l \\
\Delta^P_l(I_1, I_2) & = & \sum\limits_{(\mathtt{wt}, \mathtt{l}, \mathtt{t_1}, \ldots, \mathtt{t_n}) \in weak(P, I_1)} wt - \sum\limits_{(\mathtt{wt}, \mathtt{l}, \mathtt{t_1}, \ldots, \mathtt{t_n}) \in weak(P, I_2)} wt \\
\Delta^P_l(I_1, I_2) & = & \sum \left(\left\{ \sum\limits_{(\mathtt{wt}, \mathtt{l}, \mathtt{t_1}, \ldots, \mathtt{t_n}) \in weak(\{W\}, I_1)} wt - \sum\limits_{(\mathtt{wt}, \mathtt{l}, \mathtt{t_1}, \ldots, \mathtt{t_n}) \in weak(\{W\}, I_2)} wt \,\middle|\, W \in weak(P) \right\} \right) \\
& & \text{(as the weak constraints are independent)} \\
\Delta^P_l(I_1, I_2) & = & \sum \left(\left\{ \{W\}^{I_1}_l - \{W\}^{I_2}_l \,\middle|\, W \in weak(P) \right\} \right) \\
\Delta^P_l(I_1, I_2) & = & \sum\limits_{W \in weak(P)} \Delta^W_l(I_1, I_2)
\end{array}$$

□

Lemma 10.3.

Let S be a set of interpretations and P be an ASP program. Given any pair of interpretations I_1 and I_2 in S and any binary operator $op \in \{<, >, \leq, \geq, =, \neq\}$, $\langle I_1, I_2, op \rangle \in ord(P, S)$ if and only if $\Delta^P(I_1, I_2)$ op 0.

Proof. **Case 1**: op is $<$

Assume that $\langle I_1, I_2, < \rangle \in ord(P, S)$

$\Leftrightarrow I_1 \succ_P I_2$

$\Leftrightarrow$ there is a priority level l such that $P^{I_1}_l < P^{I_2}_l$ and $\forall l' > l$ $P^{I_1}_l = P^{I_2}_l$

$\Leftrightarrow$ there is a priority level l such that $\Delta^P_l(I_1, I_2) < 0$ and $\forall l' > l$ $\Delta^P_{l'}(I_1, I_2) = 0$

$\Leftrightarrow \Delta^P(I_1, I_2) < 0$

Case 2: op is $>$

Assume that $\langle I_1, I_2, > \rangle \in ord(P, S)$

$\Leftrightarrow I_2 \succ_P I_1$

$\Leftrightarrow$ there is a priority level l such that $P^{I_1}_l > P^{I_2}_l$ and $\forall l' > l$ $P^{I_1}_l = P^{I_2}_l$

$\Leftrightarrow$ there is a priority level l such that $\Delta^P_l(I_1, I_2) > 0$ and $\forall l' > l$ $\Delta^P_{l'}(I_1, I_2) = 0$

$\Leftrightarrow \Delta^P(I_1, I_2) > 0$

Case 3: op is $=$

Assume that $\langle I_1, I_2, = \rangle \in ord(P, S)$

$\Leftrightarrow I_1 \not\succ_P I_2$ and $I_2 \not\succ_P I_1$

$\Leftrightarrow$ for every priority level l, $P_l^{I_1} = P_l^{I_2}$

$\Leftrightarrow$ for every priority level l, $\Delta_l^P(I_1, I_2) = 0$

$\Leftrightarrow \Delta^P(I_1, I_2) = 0$

Case 4: *op* is $\neq$

Assume that $\langle I_1, I_2, \neq\rangle \in ord(P, S)$

$\Leftrightarrow I_1 \succ_P I_2$ or $I_2 \succ_P I_1$

$\Leftrightarrow$ there is a priority level l such that $P_l^{I_1} \neq P_l^{I_2}$

$\Leftrightarrow$ there is a priority level l such that $\Delta_l^P(I_1, I_2) \neq 0$

$\Leftrightarrow \Delta_l^P(I_1, I_2) \neq 0$

Case 5: *op* is $\leq$

Assume that $\langle I_1, I_2, \leq\rangle \in ord(P, S)$

$\Leftrightarrow I_2 \not\succ_P I_1$

$\Leftrightarrow$ either there is a priority level l such that $P_l^{I_1} < P_l^{I_2}$ and $\forall l' > l$ $P_l^{I_1} = P_l^{I_2}$, or for every priority level l, $P_l^{I_1} = P_l^{I_2}$

$\Leftrightarrow$ either there is a priority level l such that $\Delta_l^P(I_1, I_2) < 0$ and $\forall l' > l$ $\Delta_{l'}^P(I_1, I_2) = 0$, or for every priority level l, $\Delta_l^P(I_1, I_2) = 0$

$\Leftrightarrow \Delta^P(I_1, I_2) < 0$ or $\Delta^P(I_1, I_2) = 0$

$\Leftrightarrow \Delta^P(I_1, I_2) \leq 0$

Case 6: *op* is $\geq$

Assume that $\langle I_1, I_2, \geq\rangle \in ord(P, S)$

$\Leftrightarrow I_1 \not\succ_P I_2$

$\Leftrightarrow$ either there is a priority level l such that $P_l^{I_1} > P_l^{I_2}$ and $\forall l' > l$ $P_l^{I_1} = P_l^{I_2}$, or for every priority level l, $P_l^{I_1} = P_l^{I_2}$

$\Leftrightarrow$ either there is a priority level l such that $\Delta_l^P(I_1, I_2) > 0$ and $\forall l' > l$ $\Delta_{l'}^P(I_1, I_2) = 0$, or for every priority level l, $\Delta_l^P(I_1, I_2) = 0$

$\Leftrightarrow \Delta^P(I_1, I_2) > 0$ or $\Delta^P(I_1, I_2) = 0$

$\Leftrightarrow \Delta^P(I_1, I_2) \geq 0$

$\square$

Lemma 10.6.

Consider a background knowledge B, hypothesis space S_M and a context-dependent ordering example o. Then for any hypothesis $H \subseteq S_M$, $B \cup H$ cautiously respects o if and only if $B \cup H$ does not bravely respect $inverse(o)$.

Proof. Assume that $B \cup H$ cautiously respects o

$\Leftrightarrow$ there is no accepting pair of answer sets of $inverse(o)$

$\Leftrightarrow$ $B \cup H$ does not bravely respect $inverse(o)$

$\square$

It is useful to present the following two theorems earlier than they were presented in the chapter (as their results are used by the proofs of some of the meta-level encodings which were omitted from the main chapter).

Theorem 10.16. Let SC be a set of hypothesis schemas and S_M be a hypothesis space. For each $H \subseteq S_M$, there is exactly one answer set A_H of $\mathcal{M}_{sc}(SC, S_M)$ such that $H = \mathcal{M}^{-1}_{in\_h}(A_H)$. Furthermore, $\forall sc \in SC$, $\texttt{conforms(sc}_{\texttt{id}}\texttt{)} \in A_H$ if and only if H conforms to sc.

Proof. We prove this by using the splitting set theorem [LT94]. $e_U(P, X)$ is the partial evaluation of P with respect to X (over the atoms in U), which is described in [LT94].

To aid the proof, we partition the program $\mathcal{M}_{sc}(SC, S_M)$ into three subparts:

- $\mathcal{M}^1_{sc} = \left\{ \texttt{not\_disj(d}_{\texttt{id}}\texttt{):-}\ \ \texttt{not in\_h(d}_1\texttt{)}, \ldots, \texttt{not in\_h(d}_{\texttt{n}}\texttt{).} \middle| d = \{\texttt{d}_1, \ldots, \texttt{d}_{\texttt{n}}\} \in DISJ(SC) \right\}$
- $\mathcal{M}^2_{sc} = \left\{ \begin{array}{c} \texttt{conforms(sc}_{\texttt{id}}\texttt{):- not not\_disj(D}^1_{\texttt{id}}\texttt{)}, \ldots, \\ \texttt{not not\_disj(D}^{\texttt{n}}_{\texttt{id}}\texttt{),not\_disj(V}_{\texttt{id}}\texttt{).} \end{array} \middle| \ sc \in SC, sc = \langle \{D^1, \ldots, D^n\}, V \rangle \right\}$
- $\mathcal{M}^3_{sc} = \{\texttt{0\{in\_h(h}_{\texttt{id}}\texttt{)\}1.} | h \in S_M\}$

Let $H \subseteq S_M$ and assume $\exists A_H \in AS(\mathcal{M}_{sc}(SC, S_M))$ st $H = \mathcal{M}^{-1}_{in\_h}(A_H)$

$\Leftrightarrow$ $A_1 = \{\texttt{in\_h(h}_{\texttt{id}}\texttt{)} \mid h \in H\} \in AS(\mathcal{M}^3_{sc})$ and $\exists A_2 \in AS(e_U(\mathcal{M}^1_{sc} \cup \mathcal{M}^2_{sc}, A_1))\}$
such that $A_H = A_1 \cup A_2$
(by the splitting set theorem, using the `in_h` atoms as the splitting set U).

$\Leftrightarrow$ $\exists A_2 \in AS(\{\texttt{not\_disj(d}_{\texttt{id}}\texttt{).} \mid d \in DISJ(SC), ids(H) \cap d = \emptyset\} \cup \mathcal{M}^2_{sc}))\}$
such that $A_H = \{\texttt{in\_h(h}_{\texttt{id}}\texttt{)} \mid h \in H\} \cup A_2$
(by evaluating the function e_U)

$\Leftrightarrow$ $A_H = \{\texttt{in\_h(h}_{\texttt{id}}\texttt{)} \mid h \in H\} \cup A_2 \cup A_3$, where:

$$A_2 = \{\texttt{not\_disj(d}_{\texttt{id}}\texttt{)} \mid d \in DISJ(SC), ids(H) \cap d = \emptyset\} \text{ and}$$

$$A_3 = \left\{ \texttt{conforms(sc}_{\texttt{id}}\texttt{)} \middle| \begin{array}{c} sc \in SC, sc = \langle D, V \rangle, \\ \forall d \in D, \texttt{not\_disj(d}_{\texttt{id}}\texttt{)} \notin A_2, \texttt{not\_disj(V}_{\texttt{id}}\texttt{)} \in A_2 \end{array} \right\}$$

by the splitting set theorem (with the `not_disj` atoms as a splitting set)

$\Leftrightarrow$ $A_H = \{\texttt{in\_h(h}_{\texttt{id}}\texttt{)} \mid h \in H\} \cup A_2 \cup A_3$, where:

$$A_2 = \{\texttt{not\_disj(d}_{\texttt{id}}\texttt{)} \mid d \in DISJ(SC), ids(H) \cap d = \emptyset\} \text{ and}$$

$$A_3 = \left\{ \texttt{conforms}(\texttt{sc}_{\texttt{id}}) \;\middle|\; \begin{array}{l} sc \in SC, sc = \langle D, V\rangle, \\ \forall d \in D, ids(H) \cap d \neq \emptyset, \\ ids(H) \cap V = \emptyset \end{array} \right\}$$

$\Leftrightarrow A_H = \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}) \mid h \in H\} \cup A_2 \cup A_3$, where:
$A_2 = \{\texttt{not\_disj}(\texttt{d}_{\texttt{id}}) \mid d \in DISJ(SC), ids(H) \cap d = \emptyset\}$ and
$A_3 = \{\texttt{conforms}(\texttt{sc}_{\texttt{id}}) \mid sc \in SC, H \text{ conforms to } sc\}$

Hence, for each $H \in S_M$, there is exactly one answer set A_H of $\mathcal{M}_{sc}(SC, S_M)$ st $H = \mathcal{M}^{-1}_{in\_h}(A_H)$, and $\forall sc \in SC$, $\texttt{conforms}(\texttt{sc}_{\texttt{id}}) \in A_H$ if and only if H conforms to sc.

□

Theorem 10.18.

Let S_M be a hypothesis space, SC be a set of hypothesis schemas and OSC be a set of ordering schemas such that $\{sc \mid \langle sc, ws, op\rangle \in OSC\} \subseteq SC$. For each $H \subseteq S_M$, there is exactly one answer set A_H of $\mathcal{M}_{sc}(SC, S_M) \cup \mathcal{M}_{osc}(OSC, S_M)$ such that $\mathcal{M}^{-1}_{in\_h}(A_H) = H$. Furthermore:

1. $\forall sc \in SC$, $\texttt{conforms}(\texttt{sc}_{\texttt{id}}) \in A_H$ if and only if H conforms to sc

2. $\forall osc \in OSC$, $\texttt{osc\_conforms}(\texttt{osc}_{\texttt{id}}) \in A_H$ if and only if H conforms to osc

Proof. Let $H \subseteq S_M$. Assume A_H is an answer set of $\mathcal{M}_{sc}(SC, S_M) \cup \mathcal{M}_{osc}(OSC, S_M)$ and $\mathcal{M}^{-1}_{in\_h}(A_H) = H$.

$\Leftrightarrow A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)} \in AS(\mathcal{M}_{sc}(SC, S_M))$, $\mathcal{M}^{-1}_{in\_h}(A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)}) = H$ and $A_H \backslash HB_{\mathcal{M}_{sc}(SC,S_M)}$ is an answer set of following program:

$$\left\{ \texttt{diff}(\omega^{\texttt{id}}, 1) \;\middle|\; \begin{array}{c} \langle sc, ws, op\rangle \in OCS, \\ \omega \in ws, \\ \omega[H] > 0 \end{array} \right\} \cup \left\{ \texttt{diff}(\omega^{\texttt{id}}, -1) \;\middle|\; \begin{array}{c} \langle sc, ws, op\rangle \in OCS, \\ \omega \in ws, \\ \omega[H] < 0 \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} \texttt{osc\_diff}(\texttt{osc}_{\texttt{id}}, \texttt{D}) \texttt{:-} \\ \quad \texttt{diff}(\omega_{\texttt{i}}, \texttt{D}), \\ \quad \texttt{not diff}(\omega_1^{\texttt{id}}, -1), \texttt{not diff}(\omega_1^{\texttt{id}}, 1), \\ \quad \ldots, \\ \quad \texttt{not diff}(\omega_{\texttt{i}-1}^{\texttt{id}}, -1), \texttt{not diff}(\omega_{\texttt{i}-1}^{\texttt{id}}, 1). \end{array} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ i \in [1, m] \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} \texttt{osc\_diff}(\texttt{osc}_{\texttt{id}}, 0) \texttt{:-} \\ \quad \texttt{not osc\_diff}(\texttt{osc}_{\texttt{id}}, 1), \\ \quad \texttt{not osc\_diff}(\texttt{osc}_{\texttt{id}}, -1). \\ \texttt{osc\_conforms}(\texttt{osc}_{\texttt{id}}) \texttt{:-} \\ \quad \texttt{osc\_diff}(\texttt{osc}_{\texttt{id}}, \texttt{D}), \texttt{D op 0}. \end{array} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ \texttt{conforms}(\texttt{sc}_{\texttt{id}}) \in A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)} \end{array} \right\}$$

$\Leftrightarrow A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)} \in AS(\mathcal{M}_{sc}(SC, S_M))$, $\mathcal{M}^{-1}_{in\_h}(A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)}) = H$ and $A_H \backslash HB_{\mathcal{M}_{sc}(SC,S_M)}$ is an answer set of following program:

$$\left\{ \texttt{diff}(\omega^{\texttt{id}}, 1) \;\middle|\; \begin{array}{c} \langle sc, ws, op\rangle \in OCS, \\ \omega \in ws, \\ \omega[H] > 0 \end{array} \right\} \cup \left\{ \texttt{diff}(\omega^{\texttt{id}}, -1) \;\middle|\; \begin{array}{c} \langle sc, ws, op\rangle \in OCS, \\ \omega \in ws, \\ \omega[H] < 0 \end{array} \right\}$$

$$\cup \left\{ \texttt{osc\_diff(osc}_{\texttt{id}}\texttt{,D):-diff(}\omega_{\texttt{i}}\texttt{,D).} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ i \in [1, m], \\ \forall j \in [1, i-1], \omega_j[H] = 0 \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} \texttt{osc\_diff(osc}_{\texttt{id}}\texttt{,0):-} \\ \qquad \texttt{not osc\_diff(osc}_{\texttt{id}}\texttt{,1),} \\ \qquad \texttt{not osc\_diff(osc}_{\texttt{id}}\texttt{,}-1\texttt{).} \\ \texttt{osc\_conforms(osc}_{\texttt{id}}\texttt{):-} \\ \qquad \texttt{osc\_diff(osc}_{\texttt{id}}\texttt{,D),D op 0.} \end{array} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ \texttt{conforms(sc}_{\texttt{id}}\texttt{)} \in A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)} \end{array} \right\}$$

$\Leftrightarrow A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)} \in AS(\mathcal{M}_{sc}(SC, S_M))$, $\mathcal{M}^{-1}_{in\_h}(A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)}) = H$ and $A_H \backslash HB_{\mathcal{M}_{sc}(SC,S_M)}$ is an answer set of following program:

$$\left\{ \texttt{diff(}\omega^{\texttt{id}}\texttt{,1)} \;\middle|\; \begin{array}{c} \langle sc, ws, op\rangle \in OCS, \\ \omega \in ws, \\ \omega[H] > 0 \end{array} \right\} \cup \left\{ \texttt{diff(}\omega^{\texttt{id}}\texttt{,}-1\texttt{)} \;\middle|\; \begin{array}{c} \langle sc, ws, op\rangle \in OCS, \\ \omega \in ws, \\ \omega[H] < 0 \end{array} \right\}$$
$$\cup \left\{ \texttt{osc\_diff(osc}_{\texttt{id}}\texttt{,1).} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ \uparrow([\omega_1[H], \ldots, \omega_m[H]]) > 0 \end{array} \right\}$$
$$\cup \left\{ \texttt{osc\_diff(osc}_{\texttt{id}}\texttt{,0).} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ \uparrow([\omega_1[H], \ldots, \omega_m[H]]) = 0 \end{array} \right\}$$
$$\cup \left\{ \texttt{osc\_diff(osc}_{\texttt{id}}\texttt{,}-1\texttt{).} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ \uparrow([\omega_1[H], \ldots, \omega_m[H]]) < 0 \end{array} \right\}$$
$$\cup \left\{ \texttt{osc\_conforms(osc}_{\texttt{id}}\texttt{).} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ \texttt{conforms(sc}_{\texttt{id}}\texttt{)} \in A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)}, \\ \uparrow([\omega_1[H], \ldots, \omega_m[H]])\, op\, 0 \end{array} \right\}$$

Hence, for each $H \subseteq S_M$ there is a unique answer set A_H of $\mathcal{M}_{sc}(SC, S_M) \cup \mathcal{M}_{osc}(OSC, S_M)$ such that $\mathcal{M}^{-1}_{in\_h}(A_H) = H$. Furthermore, $A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)} \in AS(\mathcal{M}_{sc}(SC, S_M))$ and $A_H \backslash HB_{\mathcal{M}_{sc}(SC,S_M)}$ is the unique answer set of following program:

$$\left\{ \texttt{diff(}\omega^{\texttt{id}}\texttt{,1)} \;\middle|\; \begin{array}{c} \langle sc, ws, op\rangle \in OCS, \\ \omega \in ws, \\ \omega[H] > 0 \end{array} \right\} \cup \left\{ \texttt{diff(}\omega^{\texttt{id}}\texttt{,}-1\texttt{)} \;\middle|\; \begin{array}{c} \langle sc, ws, op\rangle \in OCS, \\ \omega \in ws, \\ \omega[H] < 0 \end{array} \right\}$$
$$\cup \left\{ \texttt{osc\_diff(osc}_{\texttt{id}}\texttt{,1).} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ \uparrow([\omega_1[H], \ldots, \omega_m[H]]) > 0 \end{array} \right\}$$
$$\cup \left\{ \texttt{osc\_diff(osc}_{\texttt{id}}\texttt{,0).} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ \uparrow([\omega_1[H], \ldots, \omega_m[H]]) = 0 \end{array} \right\}$$
$$\cup \left\{ \texttt{osc\_diff(osc}_{\texttt{id}}\texttt{,}-1\texttt{).} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ \uparrow([\omega_1[H], \ldots, \omega_m[H]]) < 0 \end{array} \right\}$$
$$\cup \left\{ \texttt{osc\_conforms(osc}_{\texttt{id}}\texttt{).} \;\middle|\; \begin{array}{c} osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC, \\ \texttt{conforms(sc}_{\texttt{id}}\texttt{)} \in A_H \cap HB_{\mathcal{M}_{sc}(SC,S_M)}, \\ \uparrow([\omega_1[H], \ldots, \omega_m[H]])\, op\, 0 \end{array} \right\}$$

Hence, the unique answer set A_H of $\mathcal{M}_{sc}(SC, S_M) \cup \mathcal{M}_{osc}(OSC, S_M)$ such that $\mathcal{M}^{-1}_{in\_h}(A_H) = H$ has the following properties: (i) for each $sc \in SC$, $\texttt{conforms(sc}_{\texttt{id}}\texttt{)} \in A_H$ if and only if H conforms to sc (by Theorem 10.16); and (ii) for each $osc = \langle sc, \{\omega_1, \ldots, \omega_m\}, op\rangle \in OSC$, $\texttt{osc\_conforms(osc}_{\texttt{id}}\texttt{)} \in A_H$ if and only if H conforms to sc and $\uparrow([\omega_1[H], \ldots, \omega_m[H]])\, op\, 0$ (this is the case if and only if H conforms to osc).

□

Our method for computing a translation of an interpretation into a hypothesis schema (Algorithm 10.1) makes use of a meta-level program that we omitted from the main thesis. This meta-level program is used to compute a potential unfounded subset for a partial hypothesis schema. Given an interpretation I and a hypothesis schema sc, this meta-level program finds a hypothesis that conforms to sc, but for which I has at least one unfounded set with respect to $B \cup H$.

The first two components in $\mathcal{M}_u$ state which atoms are in the object-level interpretation, and use a set of choice rules to "guess" at an unfounded subset U of this interpretation (represented by the `in_u` atoms). The next two components define what it means for an atom to be supported by a rule (without using the atoms in U in the positive body literals of the rule). If a background rule gives support to an atom, then it is definitely supported; on the other hand, if a rule in the hypothesis space gives support to the atom, then it may or may not be supported, depending on whether the rule is part of a hypothesis. We use the predicate `supports` to represent which rules in the hypothesis space represent which atoms. This program is used iteratively to find the rule-disjunctions D of a hypothesis schema. Given any already computed rule-disjunction d in D, we know that any hypothesis must contain at least one of the rules whose ids are in d. This is ensured by the fifth component of the program. The final component computes the atoms that are supported, and constrains away any meta-level answer set in which an element of U has support (i.e. those answer sets where U is not an unfounded set). For any remaining answer set, U must therefore be an unfounded set.

Meta-program B.1 ($\mathcal{M}_u(P, S_M, I, sc)$)**.** Let P be a program, S_M be a hypothesis space, I be the interpretation $\{\mathtt{a_1}, \ldots \mathtt{a_m}\}$ and sc be the hypothesis schema $\langle D, V\rangle$. $\mathcal{M}_u(P, S_M, I, sc)$ is the ASP program including the following rules:

1. $\mathtt{in\_as(a_i)}$, for each atom $\mathtt{a_i} \in I$.
2. $\mathtt{1\{in\_u(a_1), \ldots, in\_u(a_m)\}m}$.
3. $\mathtt{supported(rh)}$:- $\mathcal{R}(body(r), \mathtt{in\_as}), \mathcal{NR}(body^{+}(r), \mathtt{in\_u})$.
   for each rule $r \in P$ and for each atom $\mathtt{rh} \in heads(r)$
4. $\mathtt{supports(rh, r_{id})}$:- $\mathcal{R}(body(r), \mathtt{in\_as}), \mathcal{NR}(body^{+}(r), \mathtt{in\_u})$.
   for each rule $r \in S_M$ and for each atom $\mathtt{rh} \in heads(r)$
5. $\mathtt{1\{in\_h(d_1), \ldots, in\_h(d_n)\}n}$.
   for each $\{\mathtt{d_1}, \ldots, \mathtt{d_n}\} \in D$
6. $\left\{ \begin{array}{l} \mathtt{supported(Atom)\text{ :- }supports(Atom, Rid), in\_h(Rid).} \\ \mathtt{\text{:- }in\_u(Atom), supported(Atom).} \end{array} \right\}$

Proposition B.2. *(proof on page 315)*

Let P be a program, S_M be a hypothesis space, I be an interpretation, sc be a hypothesis schema and U be any set of atoms. $\exists A \in AS(\mathcal{M}_u(P, S_M, I, sc))$ such that $U = \{\mathtt{a} \mid \mathtt{in\_u(a)} \in A\}$ if and only if U is a potential unfounded subset of I wrt sc.

Proof. Assumes that D is always disjoint from V (which is the case).

Assume $\exists A \in AS(\mathcal{M}_u(P, S_M, I, sc))$ such that $U = \{\mathtt{a} \mid \mathtt{in\_u(a)} \in A\}$

$\Leftrightarrow$ U is non empty, and the program:

$$\left\{\ \mathtt{in\_as(a).}\ \middle|\mathtt{a} \in I\right\}$$
$$\cup \left\{\ \mathtt{in\_u(a).}\ \middle|\mathtt{a} \in U\right\}$$
$$\cup \left\{\ \mathtt{supported(rh)}\,\mathtt{:\text{-}}\mathcal{R}(\mathit{body}(r), \mathtt{in\_as}), \mathcal{NR}(\mathit{body}^+(r), \mathtt{in\_u})\ \middle| r \in P, \mathtt{rh} \in \mathit{heads}(r)\right\}$$
$$\cup \left\{\ \mathtt{supports(rh, r_{id})}\,\mathtt{:\text{-}}\mathcal{R}(\mathit{body}(r), \mathtt{in\_as}), \mathcal{NR}(\mathit{body}^+(r), \mathtt{in\_u})\ \middle| r \in S_M, \mathtt{rh} \in \mathit{heads}(r)\right\}$$
$$\cup \{\mathtt{1\{in\_h(d_1), \ldots, in\_h(d_n)\}.} | \{d_1, \ldots, d_n\} \in D\}$$
$$\cup \left\{\begin{array}{l}\mathtt{supported(ATOM)\ \text{:-}\ supports(ATOM, RID), in\_h(RID).} \\ \mathtt{\text{:-}\ in\_u(ATOM), supported(ATOM).}\end{array}\right\} \text{ is satisfiable.}$$

$\Leftrightarrow$ U is non-empty and $\exists H \subseteq S_M$ st H conforms to $\langle D, \{\mathtt{h_{id}} \mid h \in S_M, \forall d \in D : \mathtt{h_{id}} \notin d\}\rangle$ and the program:

$$\left\{\ \mathtt{in\_as(a).}\ \middle|\mathtt{a} \in I\right\}$$
$$\cup \left\{\ \mathtt{in\_u(a).}\ \middle|\mathtt{a} \in U\right\}$$
$$\cup \left\{\ \mathtt{supported(rh)}\,\mathtt{:\text{-}}\mathcal{R}(\mathit{body}(r), \mathtt{in\_as}), \mathcal{NR}(\mathit{body}^+(r), \mathtt{in\_u})\ \middle| r \in P, \mathtt{rh} \in \mathit{heads}(r)\right\}$$
$$\cup \left\{\ \mathtt{supports(rh, r_{id})}\,\mathtt{:\text{-}}\mathcal{R}(\mathit{body}(r), \mathtt{in\_as}), \mathcal{NR}(\mathit{body}^+(r), \mathtt{in\_u})\ \middle| r \in S_M, \mathtt{rh} \in \mathit{heads}(r)\right\}$$
$$\cup \{\mathtt{in\_h(h_{id}).} | h \in H\}$$
$$\cup \left\{\begin{array}{l}\mathtt{supported(ATOM)\ \text{:-}\ supports(ATOM, RID), in\_h(RID).} \\ \mathtt{\text{:-}\ in\_u(ATOM), supported(ATOM).}\end{array}\right\} \text{ is satisfiable.}$$

$\Leftrightarrow$ U is non-empty and $\exists H \subseteq S_M$ st H conforms to $\langle D, \{\mathtt{h_{id}} \mid h \in S_M, \forall d \in D : \mathtt{h_{id}} \notin d\}\rangle$ and the program:

$$\left\{\ \mathtt{supported(rh).}\ \middle| \begin{array}{l} r \in \mathit{ground}(P), \mathtt{rh} \in \mathit{heads}(r), \mathit{body}^+(r) \subseteq I, \\ \mathit{body}^-(r) \cap I = \emptyset, \mathit{body}^+(r) \cap U = \emptyset \end{array}\right\}$$
$$\cup \left\{\ \mathtt{supports(rh, r_{id}).}\ \middle| \begin{array}{l} r \in S_M, gr \in \mathit{ground}(r), \mathtt{rh} \in \mathit{heads}(gr), \mathit{body}^+(gr) \subseteq I, \\ \mathit{body}^-(gr) \cap I = \emptyset, \mathit{body}^+(gr) \cap U = \emptyset \end{array}\right\}$$
$$\cup \left\{\ \mathtt{supported(ATOM)\ \text{:-}\ supports(ATOM, h_{id}).}\ \middle| h \in H\right\}$$

has at least one answer set that does not contain $\mathtt{supported(atom)}$ for any $\mathtt{atom} \in U$.

$\Leftrightarrow$ U is non-empty and $\exists H \subseteq S_M$ st H conforms to $\langle D, \{\mathtt{h_{id}} \mid h \in S_M, \forall d \in D : \mathtt{h_{id}} \notin d\}\rangle$ and the unique answer set of the program:

$$\left\{\ \mathtt{supported(rh).}\ \middle| \begin{array}{l} r \in \mathit{ground}(P), \mathtt{rh} \in \mathit{heads}(r), \mathit{body}^+(r) \subseteq I, \\ \mathit{body}^-(r) \cap I = \emptyset, \mathit{body}^+(r) \cap U = \emptyset \end{array}\right\}$$
$$\cup \left\{\ \mathtt{supports(rh, r_{id}).}\ \middle| \begin{array}{l} r \in S_M, gr \in \mathit{ground}(r), \mathtt{rh} \in \mathit{heads}(gr), \mathit{body}^+(gr) \subseteq I, \\ \mathit{body}^-(gr) \cap I = \emptyset, \mathit{body}^+(gr) \cap U = \emptyset \end{array}\right\}$$
$$\cup \left\{\ \mathtt{supported(ATOM)\ \text{:-}\ supports(ATOM, h_{id}).}\ \middle| h \in H\right\}$$

does not contain $\mathtt{supported(atom)}$ for any $\mathtt{atom} \in U$

(the program is definite, and so has exactly one answer set).

$\Leftrightarrow$ U is non-empty and $\exists H \subseteq S_M$ st H conforms to $\langle D, \{\mathtt{h_{id}} \mid h \in S_M, \forall d \in D : \mathtt{h_{id}} \notin d\}\rangle$ and the unique answer set of the program:

$$\left\{ \mathtt{supported(rh).} \;\middle|\; \begin{array}{l} r \in \mathit{ground}(P), \mathtt{rh} \in \mathit{heads}(r), \mathit{body}^+(r) \subseteq I, \\ \mathit{body}^-(r) \cap I = \emptyset, \mathit{body}^+(r) \cap U = \emptyset \end{array} \right\}$$
$$\cup \left\{ \mathtt{supports(rh, r_{id}).} \;\middle|\; \begin{array}{l} r \in H, gr \in \mathit{ground}(r), \mathtt{rh} \in \mathit{heads}(gr), \mathit{body}^+(gr) \subseteq I, \\ \mathit{body}^-(gr) \cap I = \emptyset, \mathit{body}^+(gr) \cap U = \emptyset \end{array} \right\}$$
$$\cup \left\{ \mathtt{supported(ATOM)\text{:-}supports(ATOM, h_{id}).} \;\middle|\; h \in H \right\}$$

does not contain $\mathtt{supported(atom)}$ for any $\mathtt{atom} \in U$
(the removed rules have no effect).

$\Leftrightarrow$ U is non-empty and $\exists H \subseteq S_M$ st H conforms to $\langle D, \{\mathtt{h_{id}} \mid h \in S_M, \forall d \in D : \mathtt{h_{id}} \notin d\}\rangle$ and $\nexists r \in \mathit{ground}(P \cup H)$ such that $\mathit{heads}(r) \cap U \neq \emptyset$, $\mathit{body}^+(r) \subseteq I \backslash U$ and $\mathit{body}^-(r) \cap I = \emptyset$.

$\Leftrightarrow$ $\exists H \subseteq S_M$ st H conforms to $\langle D, \{\mathtt{h_{id}} \mid h \in S_M, \forall d \in D : \mathtt{h_{id}} \notin d\}\rangle$ and U is a non-empty unfounded subset of I wrt $P \cup H$.

$\Leftrightarrow$ $\exists H \subseteq S_M$ st H conforms to $\langle D, V\rangle$ and U is a non-empty unfounded subset of I wrt $P \cup H$. Note that for every hypothesis H_1 that conforms to $\langle D, V\rangle$ there exists a hypothesis H_2 that conforms to $\langle D, \{\mathtt{h_{id}} \mid h \in S_M, \forall d \in D : \mathtt{h_{id}} \notin d\}\rangle$ and $H_2 \subseteq H_1$ (and if U is an unfounded subset of I wrt $P \cup H_1$, then it is also an unfounded subset of I wrt $P \cup H_2$).

$\Leftrightarrow$ U is a potential unfounded subset of I wrt sc and P.

$\square$

Theorem 10.8.
Let P be a program, S_M be a hypothesis space and I be an interpretation. The procedure $translateAS(P, S_M, I)$ terminates and returns a translation of I.

Proof.

1. We first show that if $translateAS(P, S_M, I)$ terminates, then it returns a translation of I. First note that V is the set of rule ids $h_{id} \in \mathit{ids}(S_M)$ such that the corresponding rule h is not a model of I. Hence, $\forall H \subseteq S_M$ such that I is a model of H, H conforms to $\langle \emptyset, V\rangle$. So at line 4, just before the first iteration of the loop, $\forall H \subseteq S_M$ such that $I \in AS(P \cup H)$, H conforms to $\langle D, V\rangle$. We now show that this property is an invariant of the while loop. Assume for contradiction that at the start of an arbitrary loop iteration $\forall H \subseteq S_M$ such that $I \in AS(P \cup H)$, H conforms to $\langle D, V\rangle$ but that at the end, $\exists H^* \subseteq S_M$ such that $I \in AS(P \cup H^*)$ but H^* does not conform to $\langle D, V\rangle$. This means that for the set U in the loop iteration $\mathit{ids}(H^*) \cap \{\mathtt{h_{id}} \mid \mathtt{h_{id}} \in \mathit{ids}(S_M) \backslash V, I \models \mathit{body}(h), U \cap \mathit{body}^+(h) = \emptyset, \mathit{heads}(h) \cap U \neq \emptyset\} = \emptyset$. Hence, there is a U such that U is a non-empty unfounded subset of I wrt $P \cup H^*$ (P cannot contain any rule preventing

U from being an unfounded subset of I, or U would not be a potential unfounded subset of I wrt $\langle D, V\rangle$ and P). This contradicts that I is an answer set of $P \cup H^*$. Hence, when the while loop terminates $\forall H \subseteq S_M$ such that $I \in AS(P \cup H)$, H conforms to $\langle D, V\rangle$.

It remains to show that when the while loop terminates, $\forall H \subseteq S_M$ such that H conforms to $\langle D, V\rangle$, $I \in AS(P \cup H)$. Let H^* be a hypothesis that conforms to $\langle D, V\rangle$ and assume for contradiction that $I \notin AS(P \cup H^*)$. By the definition of V, I must be a model of $AS(P \cup H^*)$; hence there must be a non-empty unfounded subset U of I wrt $P \cup H^*$. As H^* conforms to $\langle D, V\rangle$, this means that U is a potential unfounded subset of I wrt $\langle D, V\rangle$ and P. This contradicts that the loop terminates.

2. Note that in each iteration, by Proposition B.2, $\langle D, V\rangle$ is conformed to by a fewer number of hypotheses than in the iteration before – each potential unfounded subset U is no longer a potential unfounded subset by the end of the iteration, meaning that at least one hypothesis that conformed to the schema at the start of the iteration does not conform to the schema at the end of the iteration. Hence, as $\langle \emptyset, V\rangle$ is conformed to by a finite number of hypotheses, and it is impossible for a schema to be conformed to by a negative number of hypotheses, there must be a finite number of iterations.

□

Algorithm 10.2 uses a meta-level program (omitted from the main thesis) that searches for a hypothesis H that accepts a CDPI e but does not conform to any schema in a set SC. This program is formalised in Meta-program B.2.

Meta-program B.2 ($\mathcal{M}_{trans}(T, e, SC)$)**.** Let T be the ILP^{noise}_{LOAS} task $\langle B, S_M, E\rangle$, e be an example, where $e_{pi} = \langle\{e^{inc}_1, \ldots, e^{inc}_m\}, \{e^{exc}_1, \ldots, e^{exc}_n\}\rangle$, and SC be a partial translation of e. $\mathcal{M}_{trans}(T, e, SC)$ is the program consisting of the following components:

- $\mathcal{R}(B \cup e_{ctx}, \texttt{in\_as}) \cup \{\mathcal{A}(\mathcal{R}(h, \texttt{in\_as}), \texttt{in\_h}(\texttt{h}_{\texttt{id}})) \mid h \in S_M\}$
- $\texttt{cov:-}\,\texttt{in\_as}(\texttt{e}^{\texttt{inc}}_{\texttt{1}}), \ldots, \texttt{in\_as}(\texttt{e}^{\texttt{inc}}_{\texttt{m}}), \texttt{ not in\_as}(\texttt{e}^{\texttt{exc}}_{\texttt{1}}), \ldots, \texttt{ not in\_as}(\texttt{e}^{\texttt{exc}}_{\texttt{n}}).$
- `:- not cov.`
- $\mathcal{M}_{sc}(SC, S_M) \cup \{\texttt{:-}\,\texttt{conforms}(\texttt{sc}_{\texttt{id}}). \mid sc \in SC\}$

The first component of the meta-level program $\mathcal{M}_{trans}(T, e, SC)$ is the reification of $B \cup S_M$, where the rules $h \in S_M$ are appended with an extra condition, ensuring that the rule takes effect if and only if the atom $\texttt{in\_h}(\texttt{h}_{\texttt{id}})$ is in an answer set. Each answer set of this first component therefore corresponds to an answer set A of $B \cup e_{ctx} \cup H$, for some $H \subseteq S_M$. The particular H for which A is an answer set can be extracted from the `in_h` atoms in the meta-level answer set. The second and third components in the meta-level program ensures that only answer sets that extend e_{pi} are returned. The final component makes use of the previous meta-level program $\mathcal{M}_{sc}$ to ensure that the hypothesis represented by the meta-level answer set does not conform to any previously computed schema. This prevents us from computing the same schema twice.

Proposition B.3. *(proof on page 318)*

Let T be an ILP^{noise}_{LOAS} task, e be an example and SC be a partial translation of e. For every interpretation I, $\exists A \in AS(\mathcal{M}_{trans}(T, e, SC))$ such that $I = \{\mathtt{a} \mid \mathtt{in\_as(a)} \in A\}$ if and only if $\exists H \subseteq S_M$ such that H does not conform to any schema in SC and I is an accepting answer set of e wrt $B \cup H$.

Proof. In the proof below, let COV be the program:

$$\left\{\begin{array}{l} \mathtt{cov\text{:-}}\ \ \mathtt{in\_as(e^{inc}_1)}, \ldots, \mathtt{in\_as(e^{inc}_m)}, \\ \qquad\quad \mathtt{not\ in\_as(e^{exc}_1)}, \ldots, \mathtt{not\ in\_as(e^{exc}_n)}. \\ \mathtt{\text{:-}\ not\ cov.} \end{array}\right\}$$

Let I be an interpretation. Assume $\exists A \in AS(\mathcal{M}_{trans}(T, e, SC))$ st $I = \{\mathtt{a} \mid \mathtt{in\_as(a)} \in A\}$.

$$\Leftrightarrow \exists A \in \left\{ A_2 \left| \begin{array}{l} A_1 \in AS(\mathcal{M}_{sc}(SC, S_M) \cup \{\text{:-}\,\mathtt{conforms(sc_{id})}. \mid sc \in SC\}), \\ A_2 \in AS(e_U(\mathcal{R}(B \cup e_{ctx}, \mathtt{in\_as}) \cup \{\mathcal{A}(\mathcal{R}(h, \mathtt{in\_as}), \mathtt{in\_h(h_{id})}) \mid h \in S_M\} \cup COV, A_1)) \end{array} \right. \right\}$$

such that $I = \{\mathtt{a} \mid \mathtt{in\_as(a)} \in A\}$
(as the atoms in $\mathcal{M}_{sc}$ form a splitting set U of $\mathcal{M}_{trans}$).

$$\Leftrightarrow \exists A \in \left\{ A_2 \left| \begin{array}{l} A_1 \in AS(\mathcal{M}_{sc}(SC, S_M) \cup \{\text{:-}\,\mathtt{conforms(sc_{id})}. \mid sc \in SC\}), \\ A_2 \in AS(\mathcal{R}(B \cup e_{ctx}, \mathtt{in\_as}) \cup \{\mathcal{R}(h, \mathtt{in\_as}) \mid h \in S_M, \mathtt{in\_h(h_{id})} \in A_1\} \cup COV) \end{array} \right. \right\}$$

such that $I = \{\mathtt{a} \mid \mathtt{in\_as(a)} \in A\}$

$$\Leftrightarrow \exists A \in \left\{ A_2 \left| \begin{array}{l} H \subseteq S_M \text{ st } \forall sc \in SC, H \text{ does not conform to } sc, \\ A_2 \in AS(\mathcal{R}(B \cup e_{ctx} \cup H, \mathtt{in\_as}) \cup COV) \end{array} \right. \right\}$$

such that $I = \{\mathtt{a} \mid \mathtt{in\_as(a)} \in A\}$
(by Theorem 10.16)

$$\Leftrightarrow I \in \left\{ \{\mathtt{a} \mid \mathtt{in\_as(a)} \in A_2\} \left| \begin{array}{l} H \subseteq S_M \text{ st } \forall sc \in SC, H \text{ does not conform to } sc, \\ A_2 \in AS(\mathcal{R}(B \cup e_{ctx} \cup H, \mathtt{in\_as}) \cup COV) \end{array} \right. \right\}$$

$$\Leftrightarrow I \in \left\{ \{\mathtt{a} \mid \mathtt{in\_as(a)} \in A_2\} \left| \begin{array}{l} H \subseteq S_M \text{ st } \forall sc \in SC, H \text{ does not conform to } sc, \\ A_2 \in AS(\mathcal{R}(B \cup e_{ctx} \cup H, \mathtt{in\_as})) \end{array} \right. \right\}$$

and I extends e_{pi}

$$\Leftrightarrow I \in \left\{ A \left| \begin{array}{l} H \subseteq S_M \text{ st } \forall sc \in SC, H \text{ does not conform to } sc, \\ A \in AS(B \cup e_{ctx} \cup H) \end{array} \right. \right\}$$

and I extends e_{pi}

$\Leftrightarrow \exists H \subseteq S_M$ st $\forall sc \in SC$, H does not conform to sc and I is an accepting answer set of e wrt $B \cup H$.

□

Theorem 10.11.

Let T be an ILP^{noise}_{LOAS} task and e be an example. $translateExample(T, e)$ terminates and returns a pair $\langle e, SC\rangle$ where SC is a complete translation of e.

Proof. Before the first iteration of the while loop in *translateExample*, $SC = \emptyset$. Hence, at this stage SC is (vacuously) a partial translation of e.

We now show that for an arbitrary iteration of the loop, if SC is a partial translation of e at the beginning of the iteration, then SC is still a partial translation of e at the end of the iteration.

Note that I is an interpretation for which $\exists H \subseteq S_M$ such that $\forall sc \in SC$, H does not conform to sc and I is an accepting answer set of e wrt $B \cup H$. Hence, the call to *translateAS* returns a schema sc such that any hypothesis H that conforms to sc must accept e (as sc is the translation of I, by Theorem 10.8). Hence $SC \cup \{sc\}$ is a partial translation of e. Therefore, at the end of the iteration, SC is still a partial translation of e.

If *translateExample* terminates, then there is no interpretation I for which $\exists H \subseteq S_M$ such that $\forall sc \in SC$, H does not conform to sc and I is an accepting answer set of e wrt $B \cup H$. Hence, there is no hypothesis that accepts e but does not conform to at least one schema in SC. Hence, as SC must be a partial translation of e from the above, SC is a complete translation of e.

It remains to show that *translateExample* always terminates. Note that the number of hypotheses that conform to at least one schema in SC increases with every iteration – as the hypothesis H that causes the while condition to be met does not conform to any schema in SC at the start of the iteration, but conforms to the new schema that is added to SC during the iteration. As there are a finite number of hypotheses, this means that there must be a finite number of iterations. □

Theorem 10.13.

Let T be a ILP^{noise}_{LOAS} task and let A_1 and A_2 be interpretations. Let $[\omega_1, \ldots, \omega_n] = \omega(T, A_1, A_2)$. For any $H \subseteq S_M$, $\Delta^{B\cup H}(A_1, A_2) = \uparrow([\omega_1[H], \ldots, \omega_n[H]])$

Proof.

Let $[l_1, \ldots, l_n]$ be the list of priority levels in $B \cup S_M$ (in descending order).

$$\begin{array}{rcl}
\Delta^{B\cup H}(A_1, A_2) & = & \uparrow([\Delta^{B\cup H}_{l_1}, \ldots, \Delta^{B\cup H}_{l_n}]) \\
& = & \uparrow\left(\left[\left(\sum\limits_{W \in weak(B\cup H)} \Delta^{W}_{l_1}\right), \ldots, \left(\sum\limits_{W \in weak(B\cup H)} \Delta^{W}_{l_n}\right)\right]\right) \\
& = & \uparrow([\omega(T, A_1, A_2, l_1)[H], \ldots, \omega(T, A_1, A_2, l_n)[H]]) \\
& = & \uparrow([\omega_1[H], \ldots, \omega_n[H]])
\end{array}$$

□

Meta-program B.3 formalises the meta-level program, $\mathcal{M}_{pair}$ that searches for a hypothesis that does not conform to any of a set of ordering schemas, but that accepts a given ordering example. For any such hypothesis, there must be a pair of interpretations that form an accepting pair of answer sets of the ordering example. $\mathcal{M}_{pair}$ can be used to find these pairs of interpretations.

Meta-program B.3 ($\mathcal{M}_{pair}(T, OSC, o)$)**.** Let T be a ILP^{noise}_{LOAS} task, OSC be a set of hypothesis schemas, and o be an ordering example, where $o_{eg1} = \langle e^1_{ctx}, e^1_{pi}\rangle$ and $o_{eg2} = \langle e^2_{ctx}, e^2_{pi}\rangle$. $\mathcal{M}_{pair}(T, OSC, o)$ is the program consisting of the following components:

- $\mathcal{M}_1(T)$
- $\{\texttt{as(1). as(2).}\}$
- $check\_ord(T, o, 1, 2)$
- $\texttt{:- not ord\_respected}(\texttt{o}_{\texttt{id}}, 1, 2)$
- $\mathcal{M}_{sc}(SC, S_M)$ (where SC is the set of all hypothesis schemas that occur in OSC).
- $\mathcal{M}_{osc}(OSC, S_M) \cup \{\texttt{:-osc\_conforms}(\texttt{osc}_{\texttt{id}}). \mid osc \in OSC\}$

Proposition B.4. *(proof on page 320)*

Let T be the ILP^{noise}_{LOAS} task $\langle B, S_M, E\rangle$ and let OSC be a partial translation of the CDOE o. For every pair of interpretations I_1, I_2, $\exists A \in \mathcal{M}_{pair}(T, OSC, o)$ such that $I_1 = \{\texttt{a} \mid \texttt{in\_as}(\texttt{a}, 1) \in A\}$ and $I_2 = \{\texttt{a} \mid \texttt{in\_as}(\texttt{a}, 2) \in A\}$ if and only if $\exists H \subseteq S_M$ that does not conform to any schema in OSC and $\langle I_1, I_2\rangle$ is an accepting pair of answer sets of o wrt $B \cup H$.

Proof. Let I_1 and I_2 be interpretations and assume $\exists A \in \mathcal{M}_{pair}(T, OSC, o)$ such that $I_1 = \{\texttt{a} \mid \texttt{in\_as}(\texttt{a}, 1) \in A\}$ and $I_2 = \{\texttt{a} \mid \texttt{in\_as}(\texttt{a}, 2) \in A\}$.

$$\Leftrightarrow \exists A \in AS\left(\begin{array}{l} \mathcal{M}_1(T) \\ \quad \cup \{\texttt{as(1). as(2).}\} \\ \quad \cup\, check\_ord(T, o, 1, 2) \\ \quad \cup \{\texttt{:- not ord\_respected}(\texttt{o}_{\texttt{id}}, 1, 2).\} \\ \quad \cup\, \mathcal{M}_{sc}(\{sc \mid \langle sc, ws, op\rangle \in OSC\}, S_M) \\ \quad \cup\, \mathcal{M}_{osc}(OSC, S_M) \cup \{\texttt{:-osc\_conforms}(\texttt{osc}_{\texttt{id}}). \mid osc \in OSC\} \end{array}\right)$$
such that $I_1 = \{\texttt{a} \mid \texttt{in\_as}(\texttt{a}, 1) \in A\}$ and $I_2 = \{\texttt{a} \mid \texttt{in\_as}(\texttt{a}, 2) \in A\}$.

$$\Leftrightarrow \exists H \subseteq S_M \text{ such that } \exists A \in AS\left(\begin{array}{l} \mathcal{M}_1(T) \\ \quad \cup \{\texttt{as(1). as(2).}\} \\ \quad \cup\, check\_ord(T, o, 1, 2) \\ \quad \cup \{\texttt{:- not ord\_respected}(\texttt{o}_{\texttt{id}}, 1, 2).\} \\ \quad \cup \{\texttt{in\_h}(\texttt{h}_{\texttt{id}}). \mid h \in H\} \end{array}\right)$$
such that $I_1 = \{\texttt{a} \mid \texttt{in\_as}(\texttt{a}, 1) \in A\}$ and $I_2 = \{\texttt{a} \mid \texttt{in\_as}(\texttt{a}, 2) \in A\}$ and
$$\exists A' \in AS\left(\begin{array}{l} \mathcal{M}_{sc}(\{sc \mid \langle sc, ws, op\rangle \in OSC\}, S_M) \\ \quad \cup\, \mathcal{M}_{osc}(OSC, S_M) \cup \{\texttt{:-osc\_conforms}(\texttt{osc}_{\texttt{id}}). \mid osc \in OSC\} \end{array}\right)$$
such that $\mathcal{M}^{-1}_{in\_h}(A') = H$.

$$\Leftrightarrow \exists H \subseteq S_M \text{ such that } \exists A \in AS\left(\begin{array}{l}\mathcal{M}_1(T)\\ \quad\cup\{\texttt{as(1).}\ \texttt{as(2).}\}\\ \quad\cup\, check\_ord(T, o, 1, 2)\\ \quad\cup\{\texttt{:- not ord\_respected(o}_{\texttt{id}}\texttt{, 1, 2).}\}\\ \quad\cup\{\texttt{in\_h(h}_{\texttt{id}}\texttt{).} \mid h \in H\}\end{array}\right)$$

such that $I_1 = \{\texttt{a} \mid \texttt{in\_as(a, 1)} \in A\}$ and $I_2 = \{\texttt{a} \mid \texttt{in\_as(a, 2)} \in A\}$ and H does not conform to any ordering schema in OSC (by Theorem 10.18).

$\Leftrightarrow \exists H \subseteq S_M$ such that H does not conform to any ordering schema in OSC and $\langle I_1, I_2\rangle$ is an accepting pair of answer sets of o wrt $B \cup H$ (by Theorem 6.6).

□

Theorem 10.14.

Let T be the ILP^{noise}_{LOAS} task $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$. Also let o be a CDOE and A_1 and A_2 be interpretations that extend $(o_{eg1})_{pi}$ and $(o_{eg2})_{pi}$, respectively. $translatePairAS(T, o, A_1, A_2)$ terminates and returns an ordering schema osc such that $\forall H \subseteq S_M$, H conforms to osc if and only if $\langle A_1, A_2\rangle$ is an accepting pair of answer sets of o wrt $B \cup H$.

Proof.

The procedure $translatePairAS$ terminates as the two calls to $translateAS$ terminate (by Theorem 10.8). It remains to show that $\forall H \subseteq S_M$, H conforms to the returned schema $\langle sc_1 \cup sc_1, \omega(T, A_1, A_2), o_{op}\rangle$ if and only if $\langle A_1, A_2\rangle$ is an accepting pair of answer sets of o wrt $B \cup H$.

Let H be an arbitrary subset of S_M and assume that H conforms to $\langle sc_1 \cup sc_1, \omega(T, A_1, A_2), o_{op}\rangle$.

$\Leftrightarrow H$ conforms to $sc_1 \cup sc_2$ and $\uparrow([\omega_i[H] \mid \omega_i \in \omega(T, A_1, A_2)])\ o_{op}\ 0$.

$\Leftrightarrow H$ conforms to sc_1, H conforms to sc_2 and $\uparrow([\omega_i[H] \mid \omega_i \in \omega(T, A_1, A_2)])\ o_{op}\ 0$.

$\Leftrightarrow A_1$ is an accepting answer set of o_{eg1} wrt $B \cup H$, A_2 is an accepting answer set of o_{eg2} wrt $B \cup H$ and $\uparrow([\omega_i[H] \mid \omega_i \in \omega(T, A_1, A_2)])\ o_{op}\ 0$.
(by Theorem 10.8)

$\Leftrightarrow A_1$ is an accepting answer set of o_{eg1} wrt $B \cup H$, A_2 is an accepting answer set of o_{eg2} wrt $B \cup H$ and $\Delta^{B\cup H}(A_1, A_2)\ o_{op}\ 0$.
(by Theorem 10.13)

$\Leftrightarrow A_1$ is an accepting answer set of o_{eg1} wrt $B \cup H$, A_2 is an accepting answer set of o_{eg2} wrt $B \cup H$ and $\langle A_1, A_2, o_{op}\rangle \in ord(B \cup H, AS(B \cup H \cup (o_{eg1})_{ctx}) \cup AS(B \cup H \cup (o_{eg2})_{ctx}))$.
(by Lemma 10.3)

$\Leftrightarrow \langle A_1, A_2\rangle$ is an accepting pair of answer sets of o wrt $B \cup H$.

□

Theorem 10.15.

Let T be any ILP_{LOAS}^{noise} task and o be any CDOE. $translateOrderingExample(T, o)$ terminates and returns a pair $\langle o, OSC \rangle$, where OSC is a complete translation of o.

Proof. (Similar to the proof of Theorem 10.11)

Before the first iteration of the while loop in $translateOrderingExample$, $OSC = \emptyset$. Hence, at this stage OSC is (vacuously) a partial translation of o (wrt T).

We now show that for an arbitrary iteration of the while loop, if OSC is a partial translation of o at the beginning of the iteration, then OSC is still a partial translation of o at the end of the iteration.

Note that $\langle I_1, I_2 \rangle$ is a pair of interpretations for which $\exists H \subseteq S_M$ such that $\forall osc \in OSC$, H does not conform to osc and $\langle I_1, I_2 \rangle$ is an accepting pair of answer sets of o wrt $B \cup H$. This means that I_1 extends $(o_{eg1})_{pi}$ and I_2 extends $(o_{eg2})_{pi}$. Hence, the call to $translatePairAS$ returns an ordering schema osc such that any hypothesis H that conforms to osc must accept o, by Theorem 10.14. Hence $OSC \cup \{osc\}$ is a partial translation of o. Therefore, at the end of the iteration, OSC is still a partial translation of o.

If $translateOrderingExample$ terminates, then there is no pair of interpretations $\langle I_1, I_2 \rangle$ for which $\exists H \subseteq S_M$ such that $\forall osc \in OSC$, H does not conform to osc and $\langle I_1, I_2 \rangle$ is an accepting pair of answer sets of o wrt $B \cup H$. Hence, there is no hypothesis that accepts o but does not conform to at least one schema in OSC. Hence, as OSC must be a partial translation of o from the above, OSC is a complete translation of o.

It remains to show that $translateOrderingExample$ always terminates. Note that the number of hypotheses that conform to at least one schema in OSC increases with every iteration – as the hypothesis H that causes the while condition to be met does not conform to any schema in OSC at the start of the iteration, but conforms to the new schema that is added to OSC during the iteration. As there are a finite number of hypotheses, this means that there must be a finite number of iterations. □

Theorem 10.20.

For any set CC of coverage constraints, and any ILP_{LOAS}^{noise} task T:

1. $solve\_current\_task(CC, T)$ terminates.

2. If there is no hypothesis $H \subseteq S_M$ such that $\mathcal{S}_{lb}(H, T, CC)$ is finite, then $solve\_current\_task(CC, T)$ returns $\langle \texttt{nil}, \texttt{nil} \rangle$.

3. If there is a hypothesis $H \subseteq S_M$ such that $\mathcal{S}_{lb}(H, T, CC)$ is finite, then $solve\_current\_task(CC, T)$ returns a pair $\langle H^*, ApproxCov \rangle$, where H^* is optimal with respect to CC and $ApproxCov = ApproxCoverage(H, T, CC)$.

Proof. Let T be the ILP_{LOAS}^{noise} task $\langle B, S_M, \langle E^+, E^-, O^b, O^c \rangle \rangle$.

1. As the program $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$ has a finite grounding (the program contains no function symbols), the call to *solve* will terminate in a finite time. Hence *SolveCurrentTask* will also terminate in a finite time.

2. Assume that there is no hypothesis $H \subseteq S_M$ such that $\mathcal{S}_{lb}(H, T, CC)$ is finite. It is sufficient to show that $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$ is unsatisfiable. Assume for contradiction that there is an answer set A of $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$. Then there must be an answer set A' of $\mathcal{M}_{sc}(ALL\_SC, S_M) \cup \mathcal{M}_{osc}(OSC, S_M)$ such that the following four properties all hold:

   (a) $\forall\langle e, SC\rangle \in CC$ such that $e \in E^+$ and $e_{pen} = \infty$, $\exists sc \in SC$ such that $\texttt{conforms}(\texttt{sc}_{\texttt{id}}) \in A'$

   (b) $\forall\langle e, SC\rangle \in CC$ such that $e \in E^-$ and $e_{pen} = \infty$, $\forall sc \in SC$, $\texttt{conforms}(\texttt{sc}_{\texttt{id}}) \notin A'$

   (c) $\forall\langle o, OSC\rangle \in CC$ such that $o \in O^b$ and $o_{pen} = \infty$, $\exists osc \in OSC$ such that $\texttt{osc\_conforms}(\texttt{osc}_{\texttt{id}}) \in A'$

   (d) $\forall\langle o, OSC\rangle \in CC$ such that $inverse(o) \in O^c$ and $o_{pen} = \infty$, $\forall osc \in OSC$, $\texttt{osc\_conforms}(\texttt{osc}_{\texttt{id}}) \notin A'$

   Hence, by Theorem 10.18, there is a hypothesis H such that the following four properties all hold:

   (a) $\forall\langle e, SC\rangle \in CC$ such that $e \in E^+$ and $e_{pen} = \infty$, $\exists sc \in SC$ such that H conforms to sc

   (b) $\forall\langle e, SC\rangle \in CC$ such that $e \in E^-$ and $e_{pen} = \infty$, $\forall sc \in SC$, H does not conform to sc.

   (c) $\forall\langle o, OSC\rangle \in CC$ such that $o \in O^b$ and $o_{pen} = \infty$, $\exists osc \in OSC$ such that H conforms to osc

   (d) $\forall\langle o, OSC\rangle \in CC$ such that $inverse(o) \in O^c$ and $o_{pen} = \infty$, $\forall osc \in OSC$, H does not conform to H

   Hence $\mathcal{S}_{lb}(H, T, CC)$ is finite. Contradiction!

   Hence, $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$ is unsatisfiable, and hence *solve_current_task*(CC, T) returns $\langle\texttt{nil}, \texttt{nil}\rangle$.

3. We first show that for any H such that $\mathcal{S}_{lb}(H, T, CC)$ is finite, there is a unique answer set A_H of the program $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$ and the score of A_H at level 0 is equal to $\mathcal{S}_{lb}(H, T, CC)$.

   First note that there is a unique answer set of A_H of the program $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$ with the final three components omitted, and that for each $e \in (E^+ \cup E^- \cup O^b \cup O^c)$, $\texttt{uncov}(\texttt{e}_{\texttt{id}}) \in A_H$ if and only if $e \notin ApproxCoverage(H, T, CC)$ (this follows from Theorem 10.18 and the definition of the `uncov` rules). Hence, the hard constraints in $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$ are not violated, and the score of A_H at level 0 is equal to $\mathcal{S}_{lb}(H, T, CC)$ (this follows from the weak constraints in $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$).

   Hence if there is at least one hypothesis $H \subseteq S_M$ such that $\mathcal{S}_{lb}(H, T, CC)$ is finite, then the program $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$ must be satisfiable. Hence, *solve_current_task*(CC, T) returns $\langle\mathcal{M}^{-1}_{in\_h}(A^*), \{e \in E^+ \cup E^- \cup O^b \cup O^c \mid \texttt{uncov}(\texttt{e}_{\texttt{id}}) \notin A^*\}\rangle$, where A^* is the optimal answer set of $\mathcal{M}_{solve}(CC, S_M, \langle E^+, E^-, O^b, O^c\rangle)$. By the above, this is a pair $\langle H^*, ApproxCoverage(H, T, CC)\rangle$, where H^* is optimal wrt CC.

□

Theorem 10.21.

Let T be a well-defined ILP^{noise}_{LOAS} task.

1. $ILASP3(T)$ terminates.

2. If T is satisfiable, then $ILASP3(T)$ returns an optimal inductive solution of T.
3. If T is unsatisfiable, then $ILASP3(T)$ returns `UNSATISFIABLE`.

Proof. Let T be the ILP^{noise}_{LOAS} task $\langle B, S_M, E\rangle$, where $E = \langle E^+, E^-, O^b, O^c\rangle$.

By Theorem 10.11 and Theorem 10.15, at each stage in the ILASP3 algorithm, CC is a set of coverage constraints such that each element in CC is of one of the following three forms:

1. $\langle e, SC\rangle$, where $e \in E^+ \cup E^-$ and SC is a complete translation of e.
2. $\langle o, OSC\rangle$, where $o \in O^b$ and OSC is a complete translation of o.
3. $\langle o, OSC\rangle$, where $inverse(o) \in O^c$ and OSC is a complete translation of o.

1. Firstly, we show that the calls $findRelevantExample$ will never return the same example twice. This is because at any time that $findRelevantExample$ is called, if it has returned an example e in a previous iteration, then $\langle e, SC\rangle$ will be in the set CC, where SC is a complete translation of e. For $findRelevantExample$ to return e again, e would have to be in the set $ApproxCoverage$, but if this is the case, then by Theorem 10.20, e must be covered by H, and therefore will not be returned by $findRelevantExample$. Hence, there must be a finite number of iterations. As each of these iterations is guaranteed to terminate (as the individual function calls have been proven to terminate, and there are no loops inside each iteration), ILASP3 must terminate.

2. Assume that T is satisfiable.

   At each stage of the ILASP3 algorithm CC is a set of coverage constraints such that for each element $\langle e, SC\rangle \in CC$, SC is a complete translation of e. Hence, $\forall\langle e, SC\rangle \in CC$, for any hypothesis $H \subseteq S_M$ such that H covers e, H must satisfy $\langle e, SC\rangle$. Hence, for any hypothesis H, $\mathcal{S}_{lb}(H, T, CC) \leq \mathcal{S}(H, T)$.

   As the task is satisfiable, there is at least one hypothesis with a finite score, and hence, in each iteration, there is always at least one hypothesis with a finite lower bound score. Hence, by Theorem 10.20 (part (3)), *solve_current_task* will never return $\langle$`nil`$,$`nil`$\rangle$. Hence, as ILASP3 is guaranteed to terminate, it must return a hypothesis.

   Let H^* be the hypothesis returned by ILASP3. $findRelevantExample(\langle B, S_M, ApproxCoverage\rangle) =$ `nil`. And hence, $ApproxCoverage(H^*, T, CC) \subseteq Coverage(H^*, T)$. Hence, $\mathcal{S}_{lb}(H^*, T, CC) \geq \mathcal{S}(H^*, T)$. Hence, $\mathcal{S}_{lb}(H^*, T, CC) = \mathcal{S}(H^*, T)$.

   Assume for contradiction that there is a hypothesis H' such that $\mathcal{S}(H', T) < \mathcal{S}(H^*, T)$. Then $\mathcal{S}_{lb}(H', T, CC) < \mathcal{S}_{lb}(H^*, T, CC)$. This contradicts the fact that H^* is optimal wrt CC (which is the case due to Theorem 10.20 part (3)). Hence, $H^* \in {}^*ILP^{noise}_{LOAS}(T)$.

3. Assume that T is unsatisfiable. Then no hypothesis H has a finite score; and hence, every hypothesis H fails to cover at least one example with an infinite penalty. This means that *solve_current_task* will never return a pair $\langle H, ApproxCov\rangle$ such that H covers every example in $ApproxCov$, as every example with infinite penalty must be in $ApproxCov$. This is the case because $\mathcal{S}_{lb}(H, T, CC)$ is finite, and $ApproxCov = ApproxCoverage(H, T, CC)$.

This means that the condition of the while loop will never be falsified. As $findRelevantExample$ is guaranteed to return a different example every time it is called (shown in the proof of (1)), there will eventually be an iteration where CC is such that $\mathcal{S}_{lb}(H, T, CC)$ is infinite for every hypothesis H. In this iteration, $solve\_current\_task(CC, T)$ will return $\langle \texttt{nil}, \texttt{nil} \rangle$ (by Theorem 10.20 part(2)), and hence, ILASP3 will return `UNSATISFIABLE`.

$\square$