Solver fast prototyping for reduct-based ELP semantics

Stefania Costantini[1,3], Andrea Formisano[2,3]

[1]Università dell'Aquila, via Vetoio, L'Aquila, Italy
[2]Università di Udine, via delle Scienze 206, Udine, Italy
[3]GNCS-INdAM, Gruppo Nazionale per il Calcolo Scientifico - INdAM, Roma, Italy

Abstract

Over time, various semantic frameworks have emerged for Epistemic Logic Programs (ELPs), which extend Answer Set Programming (ASP) by incorporating epistemic operators. These frameworks often define ELP semantics in terms of "world views," comprising sets of belief sets. Many of these approaches are "reduct-based," mirroring techniques used in ASP. They typically involve starting with a candidate world view, constructing the program's reduct based on this candidate, determining the stable models of the reduct, and verifying whether the candidate indeed forms a valid world view. Several solvers have been devised for these methods. However, ongoing debates over the "right" semantics continue, leading to the introduction of new variations. We recently proposed a rapid prototyping approach to facilitate experimentation with reduct-based semantics. This approach allows for testing on small to medium-sized programs before investing resources in developing dedicated solvers. In this paper, we refine this method in our paper and showcase its implementation in the ASP Chef System, applying it to various well-established seminal semantic approaches.

1. Introduction

The initial formulation of Epistemic Logic Programs (ELPs, referred to simply as "programs" unless specified otherwise) was introduced several years ago in [1] and [2]. ELPs aimed to extend Answer Set Programs (ASP programs), which are defined under the Answer Set Semantics [3], by incorporating "epistemic operators" to facilitate various forms of epistemic reasoning in a practical yet principled manner. Consequently, there was an early anticipation of extending ASP solvers to accommodate ELPs. However, devising the semantics for ELPs, which should serve as the foundation for solver implementations, proved to be more intricate than initially anticipated. The obstacles arise from the fact that the new epistemic operators have the capability to introspectively examine a program's own semantics, defined in terms of its "answer sets." In fact, $\mathbf{K}A$, where $\mathbf{K}$ represents "knowledge," is deemed true if the (ground) atom A holds true in every answer set of the program Π where $\mathbf{K}A$ is situated.

Semantics of ELPs is provided in terms of *world views*, which are sets of answer sets (instead of a unique set of answer sets like in ASP). Each world view, in accordance with a specified semantic approach, consistently fulfills the epistemic expressions found within a given program. Several semantic approaches for ELPs have been introduced beyond the seminal ones, among which those proposed in [4], [5], [6], [7], [8], [9], and [10]. Many of these approaches extend to ELPs what done for ASP, and thus are *reduct-based*, that is, to find world views of a given program they prescribe to:

- start with a candidate world view;
- build the *reduct* of the program with respect to this candidate world view, according to some specific definition of such reduct;
- compute the set of stable models of the reduct;
- check whether the candidate world view is indeed a world view, i.e., consists exactly of this set of stable models.

CILC24: Italian Conference on Computational Logics 2024, Rome 26-28th June.
[*]Research partially supported by Action COST CA17124 "DigForASP", by Interdepartmental Project on AI (Strategic Plan UniUD−22-25), by MaPSART-FAIR project, and by INdAM-GNCS project CUP E53C22001930001.
stefania.costantini@univaq.it (S. Costantini); andrea.formisano@uniud.it (A. Formisano)
0000-0002-5686-6124 (S. Costantini); 0000-0002-6755-9314 (A. Formisano)

While solvers have been devised for several of these approaches, the landscape of semantic proposals continues to evolve, with the likelihood of new ones being introduced in the future. This ongoing evolution is driven by the absence of consensus regarding the "right" semantics.

We have proposed in [11] a fast-prototyping methodology to obtain what we can call a "quick solver" (in the sense that it is quickly and easily obtained) for any reduct-based semantics, with no attention to performance, but with the advantage of being able to experiment with the approach on small/medium programs and not only on very small programs as done so far on paper. In this paper, we revise and simplify the method described in [11] by making non-trivial changes that increase effectiveness and usability of the approach. Consequently, we present its implementation in the ASP Chef system [12]. We then illustrate in some detail how the implementation works on some of the seminal semantic approaches, and show how it is easily customizable to other approaches. So, the demanding process of implementing a performant dedicated solver for a new semantic approach can be postponed after a testing phase is performed by using the quick solver.

The paper is organized as follows. In Section 2 we recall ASP. In Sections 3 and 4, we recall ELPs and their (envisaged) semantic properties, and we report the definition of some well-known semantic approaches. In Section 5, we introduce our proposal for constructing a solver for any given reduct-based semantics with no regard to efficiency but rather to fast prototyping. We present its implementation, developed via the ASP Chef system, and discuss how it works on some well-known seminal semantic approaches. Finally, in Section 6, we conclude. In the Appendix, for the sake of completeness, we report tables with the results that well-established semantics return on some tiny programs, and a list of available ELP solvers.

2. Answer Set Programming and Answer Set Semantics

In ASP one can see a program as a set of statements that specify a problem, where each answer set represents a solution compatible with this specification. Whenever an ASP program has no answer sets, it is said to be *inconsistent*, otherwise it is said to be *consistent*. Syntactically, an ASP program Π is a collection of *rules* of the form

$$A_1 \vee \ldots \vee A_g \leftarrow \; L_1, \ldots, L_n \tag{1}$$

where each A_i, $0 \leq i \leq g$, is an atom, $\vee$ denotes disjunction and the L_is, $0 \leq i \leq n$, are literals (i.e., atoms or negated atoms of the form $not\ A$). The left-hand side and the right-hand side of the rule are called *head* and *body*, respectively. A rule with empty body is called a *fact*. Disjunction can occur in rule heads only (and hence in facts). A rule with empty head (or, equivalently, with head $\bot$), of the form '$\leftarrow L_1, ..., L_n$.' or '$\bot \leftarrow L_1, ..., L_n$.', is a *constraint*, stating that $L_1, \ldots, L_n$ are not allowed to be simultaneously true in an answer set. The impossibility of fulfilling such requirements is one reason a program is inconsistent. Various extensions of the ASP language, including those employing aggregates and weak constraints, have been proposed. For a thorough explanation, we direct interested readers to the literature [13, 14]. In this study, we consider just *ground* programs, so implicitly assuming that all atoms are propositional.

The answer set (or "stable model") semantics can be defined in several ways [15, 16, 17]. However, the answer sets of a program Π, if any exists, are among the supported minimal classical models of the program interpreted as a first-order theory in the obvious way. The original definition [3] introduced for programs where rule heads were limited to be single atoms, was in terms of the 'GL-Operator'. Given set of atoms I and program Π, $GL_\Pi(I)$ is defined as the least Herbrand model of the program Π^I, namely, the (so-called) Gelfond-Lifschitz reduct of Π w.r.t. I. Π^I is obtained from Π by: (1) removing all rules which contain a negative literal $not\ A$, for $A \in I$; and by (2) removing all negative literals from the remaining rules. The fact that Π^I is a positive program ensures that a least Herbrand model exists. Then, I is an answer set whenever $GL_\Pi(I) = I$.

Well-developed freely available solvers exist that compute the answer sets of a given program [18, 19]. Solvers are normally provided within engines that feature auxiliary functionalities to aid programmers.

3. Epistemic Logic Programming

3.1. Introduction and Intuition

In Epistemic Logic Programming (ELP), one can postulate about what is known, meaning true in every answer set of a program in the program itself. This is done through literals of the form $\mathbf{K}L$, $\mathbf{K}$ intuitively meaning "knowledge", where $\mathbf{K}L$ is called a *subjective literal* in contrast to usual *objective literals.*

Note that mentioning L in some rules of a given program Π means that L is intended to be true in some of the answer sets of Π. In contrast, $\mathbf{K}L$ has a much stronger connotation, meaning that L is unequivocally true in any situation, i.e., true in all the answer sets. This makes Epistemic Logic Programming suitable for all those critical applications in fields such as law, cybersecurity, distributed computing, recommender systems, etc., where one must be able to refer to, so to say, *reliable truth.*

Still, as in ASP, uncertainty or conflict about the truth of some atom gives rise to several answer sets; in addition, in ELP, uncertainty or conflict about knowledge gives rise to radically alternative scenarios called *world views*, that are sets of answer sets, each one stemming from the given program and the assumptions on knowledge expressed therein. For example, the program Π_1

$$\begin{array}{ll} a \leftarrow \; not\, b & \quad e \leftarrow \; not\, \mathbf{K} f \\ b \leftarrow \; not\, a & \quad f \leftarrow \; not\, \mathbf{K} e \end{array} \tag{2}$$

has two world views, under every semantics: one is $[\{a, e\}, \{b, e\}]$, where $\mathbf{K}e$ is true (as e is, in fact, true in both answer sets) and $\mathbf{K}f$ is false, and the other one is $[\{a, f\}, \{b, f\}]$ where $\mathbf{K}f$ is true (as f is, in fact, true in both answer sets) and $\mathbf{K}e$ is false. Note that, according to a widely-used convention, each world view, which is a set of answer sets, is denoted by using brackets $[\,]$. The presence of two answer sets in each world view of Π_1 is due to the cycle on objective atoms a and b. In contrast, the presence of two world views is due to the cycle involving subjective literals (in general, the existence and number of world views are related to such kinds of cycles; see [20] for a detailed discussion).

As a concrete example, consider the following program, expressed in basic non-disjunctive ASP enriched via the $\mathbf{K}$ operator. The program states (in a hypersimplified way) under which conditions a patient should consult a doctor. In particular, in the following formulation, a patient will consult a specific doctor for some problem p on which the doctor is specialized only if the doctor is known to be reliable. Notice here the difference that this brings to the formulation: it is not sufficient that the doctor might be possibly reliable (in some answer set) but must be certainly reliable (in every answer set). Since the first rule must be grounded, in order to show concise world views we consider below only one doctor. The initial version of the program can be the following:

$$\begin{array}{l} consult(patient, X, p) \leftarrow \; doctor(X), specialized(X, p), good\_reputation(X), \mathbf{K}\, reliable(X). \\ doctor(d1). \\ specialized(d1, p). \end{array}$$

This embryonic program has only one world view, shown below, with just one answer set inside (the world view is enclosed in square brackets and the answer set in braces):

$$[\{doctor(d1), specialized(d1, p)\}]$$

There is no indication in the program, and thus, from the world view, that the doctor is reliable and has a good reputation and therefore can be consulted. One can then extend the program to outline radical uncertainty by stating that a doctor may be known to be either reliable or unreliable. This is expressed by a negative cycle on $not\, \mathbf{K}$, that generates two world views:

$$\begin{array}{rl} reliable(X) \leftarrow & doctor(X), not\, \mathbf{K}\, unreliable(X). \\ unreliable(X) \leftarrow & doctor(X), not\, \mathbf{K}\, reliable(X). \end{array}$$

Assume that there is also uncertainty about the doctor's good reputation, expressed in ASP as follows:

$$\begin{array}{rl} good\_reputation(d1) \leftarrow & not\, nogood\_reputation(d1). \\ nogood\_reputation(d1) \leftarrow & not\, good\_reputation(d1). \end{array}$$

The program outlined so far has, in fact, two world views, each including two answer sets, reflecting the uncertainty about the doctor's good reputation. The doctor is concluded reliable in one world view, W_1, and unreliable in the other one, W_2. Still, even in the former world view it is not concluded that (s)he can be consulted because of the uncertainty about her reputation.

$$\begin{aligned} W_1 = \ & [\{doctor(d1), specialized(d1,p), reliable(d1), good\_reputation(d1)\}, \\ & \{doctor(d1), specialized(d1,p), reliable(d1), nogood\_reputation(d1)\}] \\ W_2 = \ & [\{doctor(d1), specialized(d1,p), unreliable(d1), good\_reputation(d1)\}, \\ & \{doctor(d1), specialized(d1,p), unreliable(d1), nogood\_reputation(d1)\}] \end{aligned}$$

Assume now that a new rule will be established stating that a doctor has a good reputation if (s)he is known to have issued brilliant diagnoses in the past and that evidence about this will also be acquired concerning doctor $d1$.

$$\begin{aligned} & good\_reputation(X) \leftarrow \ doctor(X), \mathbf{K} past\_brilliant\_diagnoses(X). \\ & past\_brilliant\_diagnoses(d1). \end{aligned}$$

Note that $\mathbf{K} past\_brilliant\_diagnoses(d1)$ immediately follows as its argument is a fact, i.e., is unconditionally true. The conclusion $good\_reputation(X)$ that can be now drawn rules out the second answer set from both world views, which become:

$$\begin{aligned} W_1' = \ & [\{doctor(d1), specialized(d1,p), reliable(d1), good\_reputation(d1), \\ & past\_brilliant\_diagnoses(d1), consult(patient, d1, p)\}] \\ W_2' = \ & [\{doctor(d1), specialized(d1,p), unreliable(d1), good\_reputation(d1), \\ & past\_brilliant\_diagnoses(d1)\}] \end{aligned}$$

According to W_1, the patient can finally consult the doctor about her problem. In addition, since a brilliant doctor can hardly be considered unreliable, one might add the following constraint, stating, in fact, that it is impossible that a doctor issuing brilliant diagnoses in the past is known to be unreliable:

$$\leftarrow \ doctor(X), \mathbf{K} past\_brilliant\_diagnoses(X), \mathbf{K}\, unreliable(X).$$

This constraint rules out the second world view, so W_1' will become the unique world view of the program, and no uncertainty is left.

Notice that for this very simple program that we have incrementally constructed, we end up with a single world view, but this is not always the case. In general, there can be several world views, and so uncertainty exists about which scenario (outlined in a world view) is more reliable. Extensions to the basic ELP approach (that we do not consider here) provide epistemic operators ranging over the entire set of world views.

3.2. Syntax and Semantics

Epistemic Logic Programs (ELPs) allow one to express within ASP programs positive and negative *subjective literals* (in addition to *objective literals*, that are those that can occur in plain ASP programs, plus the truth constants $\top$ and $\bot$). A positive subjective literal is of the form $\mathbf{K}L$, where $\mathbf{K}$ is the *epistemic operator* of knowledge and L is an objective literal (so, epistemic operators cannot be nested), meaning that L is "known" as it is true in every answer set of the given program Π (namely, L is a *cautious consequence* of Π). The syntax of rules in an ELP program is analogous to ASP, save that literals in the body can now be either objective or subjective. An ELP program is called *objective* if no subjective literals occur therein; that is, it is an ASP program. A constraint involving (also) subjective literals is called a *subjective constraint*, whereas one involving objective literals only is an *objective constraint.* Let $Body_{subj}(r)$ be the (possibly empty) set of subjective literals occurring in the body of rule r. *Subjective rules* are those whose body is only composed of subjective literals.

Let a semantics $\mathcal{S}$ be a function mapping each program into sets of belief views, that is sets of sets of objective literals, where $\mathcal{S}$ has the property that, if Π is an objective program, then the unique member of $\mathcal{S}(\Pi)$ is the set of stable models of Π. Given a program Π, each member of $\mathcal{S}(\Pi)$ is called an $\mathcal{S}$*-world view* of Π. One usually writes "world view" instead of "$\mathcal{S}$-world view" whenever mentioning the specific semantics is irrelevant. As usual, for any world view W and any subjective literal $\mathbf{K}L$, we write $W \models \mathbf{K}L$ if and only if for all $I \in W$ the literal L is satisfied by I (i.e., if $L \in I$ for L atom, or $A \notin I$ if L is $not\ A$). W satisfies a rule r if each $I \in W$ satisfies r.

An interesting attempt to establish useful properties that ELP's semantics should obey can be found in the works by Cabalar et al. 2021, 2020. The authors state that properties analogous to the ones which have been defined over time for ASP might prove useful for ELPs as well. In ASP, in fact, the answer sets of a given program can be computed incrementally, starting from the answer sets of the so-called bottom part of the program, which are used to simplify the so-called top part, where the process can be iterated by further splitting the top and/or the bottom [22]. Hence, Cabalar et al. extend to ELPs this idea by proposing a notion of *epistemic splitting*, where top and bottom are defined w.r.t. the occurrence of epistemic operators. They also consider *subjective constraint monotonicity* and *foundedness*. The property of *subjective constraint monotonicity* states that, for any ELP program Π and any subjective constraint r, W is a world view of $\Pi \cup \{r\}$ iff both W is a world view of Π and W satisfies r. Thus, if this property is fulfilled by a semantic $\mathcal{S}$, a constraint can rule out world views but it cannot rule out some answer set from within a world view. Notice that epistemic splitting implies subjective constraint monotonicity. *Foundedness* implies that atoms occurring in sets within a world view cannot have been derived through cyclic positive dependencies, where, to define such dependencies, $\mathbf{K}A$ is seen as the same as A. Finally, they define the class of *epistemically stratified programs* that admit a unique world view (these programs are those where, intuitively, the use of epistemic operators is stratified). Foundedness appears to be a reasonable property to fulfill. The problem of unfounded world views was discovered a long time ago with G94. In fact, in the example:

$$a \leftarrow \mathbf{K}a.$$

one has that $[\{a\}]$ is a world view, in that it entails $\mathbf{K}a$, thus allowing a to be derived. This is due to the fact that, in many approaches, what is known is dictated by the world view, and there is no way (so far, though work is being done to overcome this problem) to impose that it should be consistently derived from the program as well.

4. Proposals for ELP Semantics

In this section we report some of the most relevant semantic definitions for ELPs, concentrating on the reduct-based ones. To compare the behaviour of the various semantics on practical tiny examples, the reader may refer to Tables 1 and 2 in the Appendix. At present, there is no consensus about which is the 'right' semantic approach, and of which results a semantics should return on controversial examples (for instance, on those in Table 2). A debate is quite lively about what the 'intuitive' outcome of these examples should be. This constitutes a strong motivation for our approach, that is, providing a way of quickly and easily constructing a solver so as to be able to make experiments with new semantic approaches or with variations of existing ones.

We start with the seminal definition of the first ELP semantics, introduced in [2], that we call for short G94. In what follows, let r be a rule in an ELP program Π.

Definition 4.1 (G94-world views). *The G94-reduct of* Π *w.r.t. a non-empty set of interpretations* W *is obtained by:*

(i) replacing by $\top$ *every subjective literal* $L \in Body_{subj}(r)$ *such that* L *is of the form* $\boldsymbol{K}G$ *and* $W \models G$*, and*

(ii) replacing all other occurrences of subjective literals of the form $\boldsymbol{K}G$ *by* $\bot$*.*

A non-empty set of interpretations W *is a G94-world view of* Π *iff* W *coincides with the set of all stable models of the G94-reduct of* Π *w.r.t.* W*.*

This definition was then extended in [4] introducing the semantics G11:

Definition 4.2 (G11-world views). *The G11-reduct of* Π *w.r.t. a non-empty set of interpretations* W *is obtained by:*

(i) replacing by $\bot$ *every subjective literal* $L \in Body_{subj}(r)$ *such that* $W \not\models L$,
(ii) removing all other occurrences of subjective literals of the form $not\, \boldsymbol{K}L$,
(iii) replacing all other occurrences of subjective literals of the form $\boldsymbol{K}L$ *by* L.

The set W *is a G11-world view of* Π *iff* W *coincides with the set of all stable models of the G11-reduct of* Π *w.r.t.* W.

Cabalar et al. [21] noticed that the semantics K15 introduced in [23] and recalled by the following definition, slightly generalizes the semantics G11:

Definition 4.3 (K15-world views). *The K15-reduct of* Π *w.r.t. a non-empty set of interpretations* W *is obtained by:*

(i) replacing by $\bot$ *every subjective literal* $L \in Body_{subj}(r)$ *such that* $W \not\models L$, *and*
(ii) replacing all other occurrences of subjective literals of the form $\boldsymbol{K}L$ *by* L.

The set W *is a K15-world view of* Π *iff* W *coincides the set of all stable models of the K15-reduct of* Π *w.r.t.* W.

Semantics G11 and K15, that are refinements of the original G94 semantics, have been proposed over time to cope with new examples that were discovered, on which existing semantic approaches produce unwanted or not intuitive world views.

K15 can be seen as a basis for the semantics proposed in [7], called S16 for short. In particular, S16 treats K15 world views as candidate solutions, to be pruned in a second step, where some world views are removed by applying the principle of keeping those which maximize what is not known. World views in S16 are obtained in particular as follows, where note however that [7] consider the operator **not**, that can be rephrased as $not\, \mathbf{K}A$ where not is ASP standard 'default negation' (meaning that A must be false in some answer set of a given world view).

Let $EP(\Pi)$ be the set of literals of the form **not** F occurring in given program Π.

Definition 4.4 (S16-world views). *Given* $\Phi \subseteq EP(\Pi)$, *the* epistemic reduct Π^{Φ} *of* Π *w.r.t.* Φ *is obtained by:*

(i) replacing every ***not*** $F \in \Phi$ *with* $\top$, *and*
(ii) replacing every ***not*** $F \notin \Phi$ *with* $not\, F$.

Then, the set $\mathcal{A}$ *of the answer sets of* Π^{Φ} *is a* candidate world view *if every* ***not*** $F \in \Phi$ *is true w.r.t.* $\mathcal{A}$ *(i.e.,* F *is false in some answer set* $J \in \mathcal{A}$*) and every* ***not*** $F \notin \Phi$ *is false (i.e.,* F *is true in every answer set* $J \in \mathcal{A}$*).*

We say that $\mathcal{A}$ *is obtained from* Φ *(or it is corresponding to* Φ, *or that it is a candidate world view w.r.t.* Φ*), where* Φ *is called a* candidate valid guess. *Then,* $\mathcal{A}$ *is an S16-world view if it is maximal, that is, if there exists no other candidate world view obtained from guess* Φ' *where* $\Phi \subset \Phi'$ *(so,* Φ *is called a* valid guess*).*

All the above semantics, in order to check whether a belief view $\mathcal{A}$ is indeed a world view, adopt some kind of reduct, reminiscent of that related to the stable model semantics, and $\mathcal{A}$ is a world view if it is *stable* w.r.t. this reduct.

The F15 semantics [6, 24] is based on very different principles. Namely, it is based on a combination of Equilibrium Logic [25, 26] with the modal logic S5. Differently from F15, FAAEL [10] is based on the modal logic KD45. As [10] showed, FAAEL satisfies epistemic splitting, subjective constraint monotonicity, and foundedness. G94 satisfies epistemic splitting, subjective constraint monotonicity, but not foundedness. In [10], it is proved that FAAEL world views coincide with *founded* G94 world views; this is an interesting connection, as a solver for G94, enhanced with a post-processing phase, works also for FAAEL.

For formal definitions of F15 and FAAEL, which, for lack of space, we cannot report here, we refer the reader to the aforementioned references. Also, we apologize to the readers and to the authors because, for lack of space, we do not consider other recent semantics, such as [9, 27].

Another reduct-based semantics has been proposed by the authors of this paper [28, 29] and is still under refinement; the last version is reported below. This approach considers subjective literals $\mathbf{K}G$ and $\mathbf{K}notG$ as new atoms, called *knowledge atoms*. Negation not in front of knowledge atoms is assumed to be the standard default negation. So, instead of ELPs proper, we here consider ASP programs possibly involving knowledge atoms.

An *epistemic interpretation* $\mathcal{W}$ is in our case a possibly empty set of sets of atoms, each such set composed of objective atoms occurring in the head of some (possibly unit) rule in Π. We make a difference with previous approaches where we allow $\mathcal{W}$ to be empty.

Definition 4.5 (CF24F-adaptation). *The CF24F-adaptation Π-$\mathcal{W}$ of a program Π with respect to an epistemic interpretation $\mathcal{W}$ is a new program, now obtained by modifying Π as follows:*

(i) whenever $\mathcal{W} \models G$, in all non-unit rules with head G substitute head G with $\mathbf{K}G$, and add new rule $G \leftarrow \mathbf{K}G$ and
(ii) whenever $\mathcal{W} \models not\, G$, add new rule $\mathbf{K}notG \leftarrow not\, G$

Let $F_{\Pi\text{-}\mathcal{W}}$ be the set of the newly added rule heads of the form $\mathbf{K}G$.

The CF24F Adaptation is nothing other than our definition of a reduct, though extended and more involved than previously known ones. Accordingly, we have the following notion of world view, given that for any epistemic interpretation $\mathcal{W}$ for Π, let $SM'(\Pi)$ be the set of sets obtained from the stable models $SM(\Pi)$ of the CF24F-adaptation Π-$\mathcal{W}$ of Π by cancelling knowledge atoms.

Definition 4.6 (CF24F world view). *An epistemic interpretation $\mathcal{W}$ is a CF24F world view of a program Π if $\mathcal{W} = SM'(\Pi\text{-}\mathcal{W})$.*

We may notice that, the S16 semantics is remarkable in the sense that it maximizes what is not known, which is equivalent to minimizing what is known. The proposers of S16 consider each potential world view (that, in their approach, is associated with a guess about what is not known) as a candidate world view and discard those for which there exists another one with a larger guess on what is not known (equivalently, a smaller guess on what is known) in terms of set inclusion. Rephrasing their criterion (referred to as S16C) in terms of our approach, we have:

Definition 4.7 (S16 Criterion - CF24F+S16C). *Each world view $\mathcal{W}$ as defined in Def. 4.6 is considered to be a* candidate world view*. A candidate world view $\mathcal{W}$ is indeed a world view under CF24F+S16C if no other candidate world view $\mathcal{W}$' exists, where $F_{\Pi\text{-}\mathcal{W}'} \subset F_{\Pi\text{-}\mathcal{W}}$.*

5. Fast Prototyping of a solver for reduct-based semantic approaches

An obstacle to developing a new solver for ELPs resides in the computational complexity of evaluating them [30]. The central decision problem, that is, checking whether an ELP has a world view, is Σ_P^3-complete [7, 31].

Table 3 in the Appendix shows that solvers for various semantic approaches have been developed.

The method that we propose improves the one introduced in [11] and allows a solver for any reduct-based semantics to be quickly obtained with little implementation effort. Thus, researchers proposing new semantic approaches (as it often happens) or extending/modifying an existing one can exploit the method in order to make experiments. It must be noticed, however, that the method provides correctness of the solver with respect to a given semantics but does not cope with efficiency issues. Thus, given the high complexity of evaluating ELPs, the prototypical solver obtained via our method can only be used to experiment with programs of small-medium size.

5.1. The Method

Notice that, for checking whether an epistemic interpretation $\mathcal{W}$ is a world view for program Π according to a semantics $\mathcal{S}$ based upon a definition $\mathcal{R}$ of a reduct, one has to apply $\mathcal{R}$ to Π, then find the set of answer sets of the reduced program, then possibly perform some post-processing thus obtaining a final set of answer sets, then perform a final comparison to see whether the result coincides with $\mathcal{W}$. In the case of CF24F, for instance, the post-processing cancels knowledge atoms. If applying a minimality criterion as defined in S16 and adopted in CF24F+S16C, the resulting 'candidate' world views must be filtered, and this, as discussed in [7], adds further complexity.

To find all the world views of Π, one has to devise and check all the epistemic interpretations. This, given the set At_Π of the atoms occurring in Π, one has to find all the subsets of At_Π, and all the sets of such subsets. Clearly, this process is highly complex. Basically, it 'absorbs' most of the complexity of the entire method. Then, all the epistemic interpretations must be checked (possibly with some simplifications, for instance, to exclude those epistemic interpretations containing two sets, one included in the other, etc.). Below we formalize the various steps of the proposed method.

Fast ELP-solver pipeline

The 'quick solver' for given $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{P}$, which returns all world views of any program Π, is obtained by running on Π the pipeline of the following modules:

1. A module $M_\mathcal{W}$ that computes all epistemic interpretations $\mathcal{W}_1, \dots, \mathcal{W}_k$ for Π (i.e., all sets of subsets of At_Π);
2. A module M_{red} that applies, for each $\mathcal{W}_i$, the reduct $\mathcal{R}$ to Π, and generates the reduct program Π_i (which is an ASP program);
3. A module M_{ASP} that computes the set SMs_i of answer sets Π_i, for $i = 1, \dots, k$;
4. In case a post-processing $\mathcal{P}$ is required, a module $M_\mathcal{P}$ applying $\mathcal{P}$ to select the desired candidate SMs_i's;
5. A module M_{chk} that checks each SMs_i produced by the previous step and selects those which are world views w.r.t. $\mathcal{S}$, as they coincide with the corresponding $\mathcal{W}_i$.

Correctness of the solver depends upon the correct implementation of the various modules, that, however, should not be difficult to ensure. In fact, each module copes with a single aspect and will thus be sufficiently transparent and reasonable in size.

Note that the method initially designed and described in [11] makes a single call to an ASP solver. Intuitively, this call processes a single ASP program that encodes the union (of *standardized-apart* versions) of all Π_is defined in step 2 of the above pipeline. The answer sets of such program are then filtered to extract the answer sets of the individual Π_is, which are subsequently combined into the desired candidate world views. A method that makes a single call to an ASP solver remains feasible in theory, but has strong limitations due to the excessive computational load required. Thus, one of the crucial differences between the pipeline above and the method in [11] is the processing of a single Π_i at a time (step 3). This reduces the computational load and makes the (revised) approach practical.

5.2. The ASP Chef system

ASP, as observed in [12], has been widely applied to complex search and optimization combinatorial problems in real-world and industrial applications [32, 33, 34], features linguistic high-level constructs typical of logic-based programming languages but is not formulated nor intended as a comprehensive programming language. Consequently, ASP is best utilized to tackle particular tasks within a more extensive pipeline. This implies that tasks managed by ASP engines are anticipated to receive input from other modules within the pipeline, potentially implemented using diverse paradigms. Likewise, these tasks are also expected to generate output for consumption by subsequent modules within the pipeline, which may also employ varied paradigms.

The ASP Chef system that we will use in our implementation aims to provide a framework, in which users can employ ASP directly by specifying a set of logic rules but also indirectly via mapping data from one format to the format accepted by ASP engines and mapping the output produced by ASP engines to some other format suitable to be presented to the end-user or further processed in a pipeline. ASP Chef is thus designed to ease the creation of operations pipelines rather than being just an IDE (Integrated Development Environment) or an editor for ASP.

In the ASP Chef system, there is a notion of ASP *recipe* as a chain of *ingredients* that are the instantiation of different operations, where an operation can be one of the "traditional" ASP computational tasks, or some data manipulation procedure or data visualization procedure. In order to enable the composition of linear chains of ingredients, sequences of interpretations (i.e., sequences of sets of atoms) have been adopted as a uniform format for the input and the output of all operations that can have parameters to customize their behaviour and have side output to enable inspection and visualization of intermediate states of the evaluation of ASP recipes. The adopted uniform format allows several operations implemented in ASP Chef to be combined in any order and new operations to be easily accommodated.

A web app (https://asp-chef.alviano.net/) is available to implement even long pipelines involving ASP as a core engine to perform several computational tasks, putting into practice the notion of ASP recipe. Several operations are already available; the default ASP engine exploited in ASP Chef is `clingo-wasm`, a web-accessible version of `clingo` [19]. Operations that have already been implemented and are available to future developers include searching whether an atom is in an answer set and filtering optimal answer sets according to a given objective function, sorting atoms within each model according to their lexicographical ordering, merging and splitting interpretations.

5.3. Implementation in ASP Chef

We implemented the pipeline of modules as an ASP Chef recipe, which we illustrate below. The complete recipe is accessible, usable, and modifiable through the ASP Chef web app, at this clickable link:

ELP in ASP Chef .

Let us remark that this recipe is not optimal because we chose to implement the pipeline for expository purposes rather than aiming for an optimized implementation. Also, we chose to implement each step of the pipeline separately, not necessarily using a minimal number of ingredients. Hence, a more compact recipe could be obtained, for instance, by merging different ingredients or by skipping some steps that perform some inessential processing (such as filtering of atoms that are useless for the subsequent ingredients to reduce the size of `clingo-wasm` input).

The Recipe Let us now describe the crucial parts of our recipe. Concerning the input format, since ASP Chef ingredients expect to be fed with a sequence of sets of atoms, we encoded the ELP program as a set of atoms: each rule of the form (1) is encoded as a fact `rule(head(`A_1`, ..., `A_g`), body(`L_1`, ..., `L_n`))`. Each subjective literal $\mathbf{K}G$ is encoded by a term `k(`G`)`, while negation is represented by the functor `neg`. So, we reserved the symbols `neg` and `k` and they cannot occur in input. For example, the program (2) of Section 3 is encoded as the set of four facts:

```
rule(head(a), body(neg(b))).
rule(head(b), body(neg(a))).
rule(head(e), body(neg(k(f)))).
rule(head(f), body(neg(k(e)))).
```

This single set of facts is the input sequence for the recipe. The following ASP program processes the input set to detect atoms and literals, both objective and subjective, occurring in the ELP rules. This ASP program is evaluated by a Search Models ingredient and uses some functions (i.e., `@functor`, `@arity`, and `@argument`) introduced by a specific Introspection Terms ingredient (not detailed here) exploited to introduce some Lua functionalities useful to decompose input facts and to access their sub-terms.

```
rule_head(rule(H,B), @argument(H,I)) :- rule(H,B), I = 1..@arity(H).
```

```
rule_body(rule(H,B), @argument(B,I)) :- rule(H,B), I = 1..@arity(B).
hlit(L) :- rule_head(_,L). % literals occurring in heads or in bodies
blit(L) :- rule_body(_,L). % literals occurring in bodies
atom(L) :- hlit(L).
atom(A) :- blit(neg(A)), @functor(A)!="neg", @functor(A)!="k".
atom(A) :- blit(k(A)), @functor(A)!="neg", @functor(A)!="k".
blit(L) :- blit(neg(L)).
blit(L) :- blit(k(L)).
klit(A) :- blit(k(A)). % literals/atoms in subj. literals
klit(L) :- klit(neg(L)).
```

In the output of this ingredient, one obtains the set At_Π encoded as a collection of facts of the form `atom(A)`. A Search Models ingredient evaluates such collection together with the simple ASP program:

```
{guess_true(A)} :- atom(A), hlit(A).
```

to generate a sequence of answer sets, each of them containing a possible subset of At_Π. Then, a Merge ingredient combines all these sets in a single set by distinguishing/indexing their elements by the predicate `__atomset__` (i.e., each atom $\langle atom \rangle$ occurring in the i-th set is encoded in output by an atom of the form `__atomset__`$(i, \langle atom \rangle)$). Now a Search Models ingredient processes such "indexed atoms" together with the ASP rules

```
numset(SetID) :- __atomset__(SetID,_).
{selectset(SetID)} :- numset(SetID).
% count the number of selected IDs
numOfSetsInW(Count) :- Count==#count{I:selectset(I)}.
setInW(SetID,A) :- __atomset__(SetID,A), selectset(SetID).
```

The first rule simply collects all sets indices `SetID` introduced by the previous ingredient. The second rule generates all possible selections of such indices/sets (i.e., the possible epistemic interpretations). Each of these epistemic interpretations $\mathcal{W}_i$ appears in one answer set of the ingredient output, and its members (the candidate stable models of Π_i) are encoded by facts of the form `setInW(SetID,A)`. This completes step 1 in the pipeline.

A further Search Models ingredient evaluates the following program for each $\mathcal{W}_i$ generated by the previous ingredient and infers which literals $\mathcal{W}_i$ models (by simply counting the modelling sets/IDs and comparing their number with the cardinality of $\mathcal{W}_i$):

```
modeledByW(A) :- atom(A), numOfSetsInW(N), N>0,
    N==#count{I,A:setInW(I,guess_true(A)),selectset(I)}.
modeledByW(neg(A)) :- atom(A), 0==#count{I,A:setInW(I,guess_true(A)),selectset(I)}.
```

At this point the reduct of Π can be computed, w.r.t. each of the epistemic interpretation $\mathcal{W}_i$. In the case of semantics G94, this can be achieved by a Search Models ingredient processing the following program (for each of the $\mathcal{W}_i$s):

```
red_blit(k(L),true) :- blit(k(L)), modeledByW(L).
red_blit(k(L),false) :- blit(k(L)), not modeledByW(L).
red_blit(neg(L),neg(L)) :- blit(neg(L)), @functor(L)!="k".
red_blit(L,L) :- blit(L), @functor(L)!="neg", @functor(L)!="k".
red_blit(neg(k(L)),false) :- blit(neg(k(L))), modeledByW(L).
red_blit(neg(k(L)),true) :- blit(neg(k(L))), not modeledByW(L).
% reduced rules head and body literals :
red_rule_head(rule(H,B), @argument(H,I)) :- rule(H,B), I = 1..@arity(H).
red_rule_body(rule(H,B), R) :- rule_body(rule(H,B), L), red_blit(L,R).
```

The first six rules assign to each literal in the input program Π a "substitute" (either the literal itself or one of the truth values `false` and `true`), according to Def. 4.1. Then, depending on which subjective literals are modeled by $\mathcal{W}_i$, facts of the form `red_rule_head(`$\langle rule \rangle$`,`$\langle lit \rangle$`)` and `red_rule_body(`$\langle rule \rangle$`,`$\langle lit \rangle$`)` are derived, representing the rules of the reduced program. Each reduced program is represented in a different set of atoms in the output sequence of the ingredient. This completes step 2 of the pipeline.

Each set so obtained is joined to the program below and the answer sets are computed. Each element of each SMs_i (for all i) is encoded in a distinct set of atoms in the output sequence of the ingredient by a collection of facts `true(`$\langle atom \rangle$`)`.

```
% detect falsified reduced rules bodies:
red_body_false(R) :- red_rule_body(R,false).
% infer true literals w.r.t. reduced rules
true(L) : red_rule_head(rule(H,B), L) :-
  rule(H,B), not red_body_false(rule(H,B));
  true(N):red_rule_body(rule(H,B),N), @functor(N)!="neg", @functor(N)!="true";
  not true(M):red_rule_body(rule(H,B),neg(M)), @functor(M)!="neg",
      @functor(M)!="false";
  not not true(M):red_rule_body(rule(H,B),neg(neg(M))), @functor(M)!="false".
```

Few ingredients are used to gather in a single set the collection SMs_i, for each i. This operation outputs the sequence of SMs_is and completes step 3 in the pipeline.

Step 5 of the pipeline is performed (note that step 4 is not needed for G94). For brevity, we omit the details: the remaining part of the recipe compares each SMs_i with $\mathcal{W}_i$ filtering out those that do not match. The output is then processed for better readability, and the world views of Π are listed by facts of the form `worldView_SMid_Atom(`$\langle SMid \rangle$`,`$\langle atom \rangle$`)`. For instance, this is the output for the ELP program (2):

```
worldView_SMid_Atom(101,a).
worldView_SMid_Atom(101,e).
worldView_SMid_Atom(102,b).
worldView_SMid_Atom(102,e).
§
worldView_SMid_Atom(157,a).
worldView_SMid_Atom(157,f).
worldView_SMid_Atom(158,b).
worldView_SMid_Atom(158,f).
```

The two sets in the output sequence (separated by §) represent two world views made of two answer sets (identified by different numeric IDs), each composed of two atoms.

The last three ingredient of the recipe,[1] provide a graphical representation of the result. Namely, a SELECT MODELS ingredient can be used to select one of the world views. Then, a SEARCH MODELS ingredient processes a simple ASP program joined with the selected world view and defines nodes and edges of a graph. The yellow colors is associated to nodes representing answer sets (the corresponding ID labels the node), while edges link each answer set to nodes representing its true atoms. Finally, a GRAPH ingredient displays the world view.

We conclude this section by observing that the pipeline we described and implemented in ASP Chef can be easily exploited to mechanize any reduct-based semantics for ELP (hence the “fast prototyping”). It simply suffices to modify a single ingredient. Namely, the one that implements the module M_{red} of the pipeline and, for each $\mathcal{W}_i$, computes the reduct program Π_i. This is equivalent to providing adequate alternative definitions of the predicates `red_blit/2` and (possibly) `red_rule_body/2` seen before, according to the specific notion of reduct at hand. The rest of the recipe remains unchanged.

The following is a possible encoding of the G11-reduct which involves a change in the definition of `red_blit/2` only.

```
red_blit(k(L),false) :- blit(k(L)), not modeledByW(L).
red_blit(neg(k(L)),false) :- blit(neg(k(L))), modeledByW(L).
red_blit(neg(k(L)),true) :- blit(neg(k(L))),
    not modeledByW(@argument(@argument(L,1),1)).
red_blit(k(L),L) :- blit(k(L)), modeledByW(L).
red_blit(neg(L),neg(L)) :- blit(neg(L)), @functor(L)!="k".
red_blit(L,L) :- blit(L), @functor(L)!="neg", @functor(L)!="k".
```

[1]We thank an anonymous reviewer for suggesting this final fragment of the recipe.

In an analogous way one can customize the solver for the K15 semantics by using the following definition of `red_blit/2`:

```
red_blit(k(L),false) :- blit(k(L)), not modeledByW(L).
red_blit(neg(k(L)),true) :- blit(neg(k(L))), not modeledByW(L).
red_blit(neg(k(neg(L))),neg(neg(L))) :- blit(neg(k(neg(L)))), modeledByW(neg(L)).
red_blit(k(L),L) :- blit(k(L)), modeledByW(L).
red_blit(neg(k(L)),neg(L)) :- blit(neg(k(L))), @functor(L)!="neg", modeledByW(L).
red_blit(neg(L),neg(L)) :- blit(neg(L)), @functor(L)!="k".
red_blit(L,L) :- blit(L), @functor(L)!="neg", @functor(L)!="k".
```

6. Conclusions

The quest for the "right" semantics of Epistemic Logic Programs continues and can be facilitated by the possibility of experimenting with new semantic approaches, going beyond the tiny programs that one may consider on paper. To allow for such experimentations, we devised a method for fast prototyping of solvers for reduct-based semantic approaches. We illustrated the method for the G94 semantics (which implies that the method is also indirectly applicable to FAAEL), and we showed that it is, however, easily customizable to every other reduct-based approach. This is due to the modular definition and the corresponding modular implementation in ASP Chef. As a matter of fact, we verified through the proposed tool all the examples occurring in the literature plus others. A current limitation is the limited scalability of the implementation, which still does not allow the application of the method to large programs.

Thus, future work will be concerned with implementing and experimenting with other semantics and improving the efficiency of the implementation. We will also investigate the possibility to extend the method to semantic approaches that are not reduct-based.

References

[1] M. Gelfond, H. Przymusinska, Definitions in epistemic specifications, in: A. Nerode, V. W. Marek, V. S. Subrahmanian (Eds.), Proc. of the 1st Intl. Workshop on Logic Programming and Non-monotonic Reasoning, The MIT Press, 1991, pp. 245–259.

[2] M. Gelfond, Logic programming and reasoning with incomplete information, Ann. Math. Artif. Intell. 12 (1994) 89–116. doi:10.1007/BF01530762.

[3] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. Kowalski, K. Bowen (Eds.), Proc. of the 5th Intl. Conf. and Symp. on Logic Programming, MIT Press, 1988, pp. 1070–1080.

[4] M. Gelfond, New semantics for epistemic specifications, in: J. P. Delgrande, W. Faber (Eds.), Proc. of LPNMR'11, volume 6645 of *LNCS*, Springer, 2011, pp. 260–265.

[5] M. Truszczynski, Revisiting epistemic specifications, in: M. Balduccini, T. C. Son (Eds.), Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning, volume 6565 of *LNCS*, Springer, 2011, pp. 315–333.

[6] L. Fariñas del Cerro, A. Herzig, E. I. Su, Epistemic equilibrium logic, in: Q. Yang, M. J. Wooldridge (Eds.), Procİ 2015, AAAI Press, 2015, pp. 2964–2970.

[7] Y. Shen, T. Eiter, Evaluating epistemic negation in answer set programming, Artificial Intelligence 237 (2016) 115–135.

[8] P. T. Kahl, A. P. Leclerc, Epistemic logic programs with world view constraints, in: A. D. Palù, P. Tarau, N. Saeedloei, P. Fodor (Eds.), Tech. Comm. of ICLP 2018, volume 64 of *OASIcs*, Schloss Dagstuhl, 2018, pp. 1:1–1:17.

[9] E. I. Su, Epistemic answer set programming, in: F. Calimeri, N. Leone, M. Manna (Eds.), Proc. of JELIA'19, volume 11468 of *LNCS*, Springer, 2019, pp. 608–626.

[10] P. Cabalar, J. Fandinno, L. Fariñas del Cerro, Autoepistemic answer set programming, Artif. Intell. 289 (2020) 103382.

[11] S. Costantini, A. Formisano, Fast prototyping of a solver for reduct-based ELP semantics, in: Proc. of CILC'23, volume 3428 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.

[12] M. Alviano, D. Cirimele, L. A. Rodriguez Reiners, Introducing ASP recipes and ASP Chef, in: J. Arias, et al. (Eds.), Proceedings of the ICLP'23 Workshops, volume 3437 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.

[13] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer Set Solving in Practice, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2012.

[14] V. Lifschitz, Answer Set Programming, Springer, 2019.

[15] V. Lifschitz, Thirteen definitions of a stable model, in: A. Blass, N. Dershowitz, W. Reisig (Eds.), Fields of Logic and Computation, volume 6300 of *LNCS*, Springer, 2010, pp. 488–503.

[16] S. Costantini, A. Formisano, RASP and ASP as a fragment of linear logic, Journal of Applied Non-Classical Logics (JANCL) 23 (2013) 49–74. doi:10.1080/11663081.2013.798997.

[17] S. Costantini, A. Formisano, Negation as a resource: a novel view on answer set semantics, Fundamenta Informaticae 140 (2015) 279–305. doi:10.3233/FI-2015-1255.

[18] B. Kaufmann, N. Leone, S. Perri, T. Schaub, Grounding and solving in answer set programming, AI Mag. 37 (2016) 25–32. doi:10.1609/AIMAG.V37I3.2672.

[19] ASP solvers, 2024. Cmodels: www.cs.utexas.edu/users/tag/cmodels; DLV: www.dlvsystem.it; Clingo: potassco.sourceforge.net; WASP: alviano.github.io/wasp.

[20] S. Costantini, About epistemic negation and world views in epistemic logic programs, Theory Pract. Log. Program. 19 (2019) 790–807. doi:10.1017/S147106841900019X.

[21] P. Cabalar, J. Fandinno, L. Fariñas del Cerro, Splitting epistemic logic programs, Theory Pract. Log. Program. 21 (2021) 296–316. doi:10.1017/S1471068420000058.

[22] V. Lifschitz, H. Turner, Splitting a logic program, in: Proc. of ICLP'94, MIT Press, 1994, pp. 23–37.

[23] P. Kahl, R. Watson, E. Balai, M. Gelfond, Y. Zhang, The language of epistemic specifications (refined) including a prototype solver, J. Log. Comp. 30 (2015) 953–989.

[24] E. I. Su, L. Fariñas del Cerro, A. Herzig, Autoepistemic equilibrium logic and epistemic specifications, Artif. Intell. 282 (2020) 103249. doi:10.1016/j.artint.2020.103249.

[25] D. Pearce, A new logical characterization of stable models and answer sets, in: Non-Monotonic Extensions of Logic Programming, number 1216 in LNAI, Springer, 1997, pp. 55–70.

[26] D. Pearce, A. Valverde, Synonymous theories in answer set programming and equilibrium logic, Proc. of ECAI'04 (2004) 388–390.

[27] E. I. Su, Refining the semantics of epistemic specifications, in: A. Formisano, et al. (Eds.), Tech. Comm. of ICLP'21, volume 345 of *EPTCS*, 2021, pp. 113–126. doi:10.4204/EPTCS.345.25.

[28] S. Costantini, A. Formisano, Epistemic logic programs: a novel perspective and some extensions, in: J. Arias, et al. (Eds.), Proceedings of the ICLP'22 Workshops, volume 3193 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022.

[29] S. Costantini, A. Formisano, Epistemic logic programs: A study of some properties, Theory and Practice of Logic Programming (2024). doi:10.1017/S1471068424000012.

[30] M. Hecher, M. Morak, S. Woltran, Structural decompositions of epistemic logic programs, in: Proc. of AAAI'20, IAAI'20, EAAI'20, AAAI Press, 2020, pp. 2830–2837.

[31] M. Morak, Epistemic logic programs: A different world view, in: B. Bogaerts, et al. (Eds.), Technical Communications of ICLP'19, volume 306 of *EPTCS*, 2019, pp. 52–64. doi:10.4204/EPTCS.306.11.

[32] A. Dal Palù, A. Dovier, A. Formisano, E. Pontelli, ASP applications in bio-informatics: A short tour, KI - Kunstliche Intelligenz 32 (2018) 157–164. doi:10.1007/s13218-018-0551-y.

[33] A. Dovier, A. Formisano, E. Pontelli, An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems, Journal of Experimental and Theoretical Artificial Intelligence 21 (2009) 79–121. doi:10.1080/09528130701538174.

[34] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (2016) 53–68. doi:10.1609/AIMAG.V37I3.2678.

Table 1
On the left, examples where G94, G11, K15, F15, S16, and FAEEL agree. On the right, examples where G94/G11/FAEEL differ from K15/F15/S16.

Program	World views
$a \vee b$	$[\{a\}, \{b\}]$
$a \vee b$ $a \leftarrow \mathbf{K}b$	$[\{a\}, \{b\}]$
$a \vee b$ $a \leftarrow \mathit{not}\, \mathbf{K}b$	$[\{a\}]$
$a \vee b$ $c \leftarrow \mathit{not}\, \mathbf{K}b$	$[\{a, c\}, \{b, c\}]$
$a \leftarrow \mathit{not}\, \mathbf{K}b$ $b \leftarrow \mathit{not}\, \mathbf{K}a$	$[\{a\}],\ [\{b\}]$
$a \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, a$ $a \leftarrow \mathit{not}\, \mathbf{K}a$	$[\{a\}]$

Program	G94/G11/FAEEL	K15/F15/S16
$a \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, a$	$[\emptyset],\ [\{a\}]$	$[\{a\}]$
$a \vee b$ $a \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, b$	none	$[\{a\}]$
$a \vee b$ $a \leftarrow \mathbf{K}\mathit{not}\, b$	$[\{a\}],\ [\{a\}, \{b\}]$	$[\{a\}, \{b\}]$
$a \leftarrow b$ $b \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, a$	$[\emptyset],\ [\{a, b\}]$	$[\{a, b\}]$
$a \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, b$ $b \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, a$	$[\emptyset],\ [\{a, b\}]$	$[\{a\}, \{b\}]$

Table 2
Examples showing differences among several semantics.

Program	World views			
	G94	G11/FAEEL	K15	F15/S16
$a \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, b \wedge \mathit{not}\, b$ $b \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, a \wedge \mathit{not}\, a$	$[\emptyset], [\{a\}, \{b\}]$			$[\{a\}, \{b\}]$
$a \leftarrow \mathbf{K}a$	$[\emptyset], [\{a\}]$	$[\emptyset]$		
$a \leftarrow \mathbf{K}a$ $a \leftarrow \mathit{not}\, \mathbf{K}a$	$[\{a\}]$	none		

A. Semantic Results for Interesting ELP Programs

In Tables 1 and 2, a summary is reported, taken from [10], of how the semantics presented in this paper behave on some examples which are considered to be significant of situations that can be found in practical programming.

B. Available ELP Solvers

Table 3 shows, to the best of our knowledge, a list of available solvers for the semantics reported in previous sections.

Table 3
List of solvers available for the semantics summarized in Section 4

Solver	Year	Semantics	Underlying ASP-solver	Impl. language	Availability
ELMO	1994	G94	dlv	Prolog	n/a
sismodels	1994	G94	claspD	C++	n/a
Wviews	2007	G94	clingo	C++	Windows binary
Esmodels	2013	G11	clingo	(unknown)	Windows binary
ELPS	2014	K15	clingo	Java	source+binary
GISolver	2015	K15	clingo	(unknown)	Windows binary
ELPsolve	2016	K15/S16	clingo	C++	binary only
Wviews2	2017	G94	clingo	Python	Windows binary
EP-ASP	2017	K15/S16	clingo	Python+ASP	Windows binary
PelpSolver	2017	S16	clingo	Java	Windows binary
ELPsolve2	2017	S16	clingo	C++	not public release
EHEX	2018	S16	clingo	Python	source
selp	2018	S16	clingo	Python	source
eclingo	2020	G94	clingo	Python	source