Answer Set Programming via Mixed Integer Programming

Guohua Liu and **Tomi Janhunen** and **Ilkka Niemelä**
Aalto University School of Science
Department of Information and Computer Science
{Guohua.Liu, Tomi.Janhunen, Ilkka.Niemela}@aalto.fi

Abstract

Answer set programming is a programming paradigm where a given problem is formalized as a logic program whose answer sets correspond to the solutions to the problem. In this paper, we link answer set programming with another widely applied paradigm, viz. mixed integer programming. As a theoretical result, we establish translations from non-disjunctive logic programs to linear constraints used in mixed integer programming so that the solutions to the constraints correspond to the answer sets of the programs. These translations create the basis for an extended answer set programming language that includes linear constraints as a primitive and enables more compact encodings of problems. On a practical level, we have implemented a prototype system that computes answer sets using a state-of-the-art mixed integer programming solver. The reported experiments demonstrate the effectiveness of this approach applied to a number of optimization problems and problems with variables ranging over large domains.

Introduction

Answer set programming (ASP) is a declarative programming paradigm where a given problem is solved by devising a logic program such that the answer sets of the program provide the answers to the problem, i.e., solving the problem is reduced to the computation of answer sets for the program. Answer set programming combines a simple yet intuitive modeling language and efficient solving tools and has been employed in a wide variety of applications as discussed by Brewka, Eiter, and Truszczyński (2011).

Although ASP has been very successful in a number of domains, there are classes of constraints which lack full treatment in the basic ASP modelling language. A prime example are linear constraints over integers and reals. These are challenging for the implementation techniques used in state-of-the-art ASP solvers which implement a two-phase computation of answer sets: *grounding* and *model search*. In the grounding phase the input program containing variables and complex cardinality, aggregate, and other constraints is transformed to a basically propositional representation which is the starting point of the model search phase. Although efficient grounding techniques have been developed for a range of constraints, for linear constraints and, in particular, for variables ranging over large or infinite domains, the grounding phase becomes very challenging. Depending on the constraints and the ranges of the variables involved, the grounding could be infinite or finite but very large leading to significant performance problems.

In this paper, we study the integration of linear constraints over integers and reals to ASP so that even infinite variable domains can be supported. To overcome the limitations of typical grounding-based implementation techniques we investigate an alternative translation-based approach. Here we exploit mixed integer programming (MIP), a widely applied mathematical programming paradigm, where (optimization) problems are modeled in terms of linear arithmetic constraints and the solutions to the constraints give (optimal) solutions to the problems.

As regards technical results, we (1) establish translations from an ASP program to a MIP program such that the answer sets of the ASP program correspond to the solutions of the MIP program; (2) formalize an extended ASP language ASP(LC) that integrates ASP and linear constraints so that problems involving infinite and large variable domains can be compactly encoded; (3) develop a translation from programs in the extended language to a set of linear constraints whose solutions capture answer sets; (4) implement an approach to answer set computation based on the translation in a prototype system. Our preliminary experiments show the effectiveness of this approach to optimization problems and problems with variables ranging over large domains.

There are pioneering works on integrating ASP and constraint processing techniques (Balduccini 2011; Mellarkod, Gelfond, and Zhang 2008; Gebser, Ostrowski, and Schaub 2009). In these approaches, answer sets are computed using an ASP solver and a constraint solver together: the former deals with ASP rules whereas the latter handles constraints only—implying that the searches of the two solvers have to be synchronized in a way or another. In particular, an efficient solver is implemented in (Gebser, Ostrowski, and Schaub 2009). In contrast, our approach is based on the translations from ASP to MIP programs, thus, treating ASP rules and constraints uniformly within a single formalism and fully exploiting the capabilities of MIP solvers.

The rest of the paper is organized as follows. After defin-

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

ing some preliminary concepts, translations from pure ASP programs to MIP programs are presented. Next, an extended ASP language ASP(LC) supporting linear constraints is introduced, discussed in terms of examples, and translated back to a set of linear constraints. Then we describe a prototype implementation of ASP(CL) and report experimental results obtained for a number of benchmark problems. The paper ends with a summary of related work and conclusions.

Preliminaries

In this section, we recall the main concepts of logic programs, linear constraints, and mixed integer programming.

Normal Logic Programs

A normal logic program (NLP) (or simply program) is a set of rules of the form

$$a \leftarrow b_1, \ldots, b_m, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_n. \quad (1)$$

where each a, b_i, and c_j is a propositional atom. Atoms and atoms preceded by 'not' are also called *literals* (the latter may be emphasized as *negative* literals). Given a rule r of the form (1), we introduce the following abbreviations. The *head* and the *body* of r are defined by $H(r) = a$ and $B(r) = \{b_1, \ldots, b_m, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_n\}$, respectively. We let $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{c_1, \ldots, c_n\}$ to distinguish the *positive* and the *negative* parts of $B(r)$, respectively. A rule without body is a *fact* whose head is true unconditionally. A rule of the form (1) without the head a is an *integrity constraint* enforcing the body of the rule to be false. We semantically treat such a rule as a shorthand for a normal rule $f \leftarrow b_1, \ldots, b_m, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_n, \mathrm{not}\, f$ where f is a new atom.

The *Herbrand base* of a program P, denoted $\mathrm{At}(P)$, is the set of atoms that appear in its rules. A set of atoms M satisfies an atom a if $a \in M$ and a negative literal $\mathrm{not}\, a$ if $a \notin M$, denoted $M \models a$ and $M \models \mathrm{not}\, a$, respectively; M satisfies a set of literals L, denoted $M \models L$, if it satisfies each literal in L; M satisfies a rule r of the form (1), denoted $M \models r$, if $M \models H(r)$ whenever $M \models B(r)$. A set of atoms M is a *(classical) model* of P, denoted $M \models P$, if M satisfies each rule of P.

An *answer set* of a program is in a sense "justified" model of the program which is captured by the concept of a *reduct*.

Definition 1. *Let P be a normal program and M a set of atoms. The* reduct *of P with respect to M is defined by $P^M = \{H(r) \leftarrow B^+(r) \mid r \in P \text{ and } B^-(r) \cap M = \emptyset\}$.*

The reduct P^M does not contain any negative literals and, hence, has a unique $\subseteq$-minimal model. If this model coincides with M, then M is an answer set of P.

Definition 2. *(Gelfond and Lifschitz 1988) A set $M \subseteq \mathrm{At}(P)$ is an* answer set *(originally, a* stable model*) of a normal logic program P iff M is the minimal model of P^M (under set inclusion).*

In general, the number of answer sets can vary, and we we use $AS(P)$ to denote the set of answer sets of a program P.

Example 1. Consider a logic program P consisting of:

$$\begin{array}{llll} r_1\colon & a \leftarrow b. & r_2\colon & a \leftarrow c. \\ r_3\colon & b \leftarrow a. & & \\ r_4\colon & c \leftarrow \mathrm{not}\, d. & r_5\colon & d \leftarrow \mathrm{not}\, c. \end{array}$$

Then $AS(P) = \{\{a, b, c\}, \{d\}\}$. To verify the set $\{d\}$, note that $P^{\{d\}}$ consists of r_1, r_2, r_3, and the fact $d \leftarrow$. The resulting unique minimal model of $P^{\{d\}}$ coincides with $\{d\}$. □

Niemelä (2008) characterizes answer sets using *level rankings*. Intuitively, a level ranking of a set of atoms gives an order in which the atoms in the set are derived and the set is an answer set only if there is such an order for the atoms in the set. The characterization is based on the concepts of a supported model (Apt, Blair, and Walker 1988) and applicable rules. A set of atoms M is a *supported model* of a program P iff $M \models P$ and for every atom $a \in M$ there is a rule $r \in P$ such that $H(r) = a$ and $M \models B(r)$. For a program P and an interpretation $M \subseteq \mathrm{At}(P)$ the set of *M-applicable rules* $P_M = \{r \in P \mid M \models B(r)\}$. Any answer set of a normal logic program is also a supported model of the program but the converse does not hold in general (Marek and Subrahmanian 1992). The level ranking characterization gives the condition under which a supported model is an answer set.

Definition 3. *Let M be a set of atoms and P a normal program. A function* $\mathrm{lr} : M \to \mathbb{Z}^+$ *is a* level ranking *of M for P iff for each $a \in M$, there is a rule $r \in P_M$ such that $H(r) = a$ and for every $b \in B^+(r)$, $\mathrm{lr}(a) > \mathrm{lr}(b)$.*

Theorem 1. *(Niemelä 2008) Let M be a supported model of a normal program P. Then M is an answer set of P iff there is a level ranking of M for P.*

Weight Constraint Programs

A *weight constraint program* (WCP) is a set of rules[1]

$$a \leftarrow W. \quad (2)$$

where W is a *weight constraint* of the form

$$l[b_1 = w_{b_1}, \ldots, b_m = w_{b_m}, \mathrm{not}\, c_1 = w_{c_1}, \ldots, \mathrm{not}\, c_n = w_{c_n}] \quad (3)$$

in which each b_i and each $\mathrm{not}\, c_j$ is a literal accompanied by an integer *weight*. The integer number l is a *lower* bound. For a weight constraint W of the form (3), we denote the set of positive and negative literals in W by $W^+ = \{b_1, ..., b_m\}$ and $W^- = \{c_1, ..., c_n\}$ respectively. For a rule r of the form (2), we use $W(r)$ to refer the weight constraint in the body of r and define $B(r) = W(r)^+ \cup \{\mathrm{not}\, c \mid c \in W(r)^-\}$, $B^+(r) = W(r)^+$, and $B^-(r) = W(r)^-$.

A set of atoms M *satisfies* a weight constraint W of the form (3), denoted $M \models W$, iff $w(W, M) \geq l$, where $w(W, M) = \sum_{b_i \in M} w_{b_i} + \sum_{c_j \notin M} w_{c_j}$. The concept of a model of a program is naturally extended to WCPs.

[1] Generally, weight constraint programs may contain rules of more complicated form. It has been shown sufficient to consider the rules of the given form only (Simons, Niemelä, and Soininen 2002; Marek, Niemelä, and Truszczyński 2008; Janhunen and Niemelä 2011).

Answer sets of weight constraint programs are defined using the *reduct* of weight constraints defined as: For a weight constraint W of the form (3) and a set of atoms M, the reduct of W w.r.t. M, denoted W^M, is the weight constraint $l^M\ [b_1 = w_{b_1}, ..., b_m = w_{b_m}]$ where $l^M = l - \sum_{c_j \notin M} w_{c_j}$.

Definition 4. *Let P be a weight constraint program and M a set of atoms. The reduct of P with respect to M, denoted P^M, is defined by $P^M = \{a \leftarrow W^M \mid a \leftarrow W \in P\}$.*

Definition 5. *A set $M \subseteq \mathrm{At}(P)$ is an* answer set *of a WCP P iff M is the minimal model of P^M.*

A level ranking characterization of answer sets for WCPs is given in (Liu and You 2010) generalizing the characterization for NLPs. We restate the results as below for our purposes, with the natural extensions of the concepts of a supported model and applicable rules to WCPs.

Definition 6. *Let M be a set of atoms and P a weight constraint program. A function $\mathrm{lr} : M \to \mathbb{Z}$ is a* level ranking *of M for P iff for each $a \in M$, there is a rule $r \in P_M$ such that $H(r) = a$ and there is a set $X \subseteq M \cap W(r)^+$, such that $X \models W(r)$ and for every $b \in X$, $\mathrm{lr}(a) > \mathrm{lr}(b)$.*

Theorem 2. *(Liu and You 2010) Let M be a supported model of a weight constraint program P. Then M is an answer set of P iff there is a level ranking of M for P.*

Components and Defining Rules

The *dependency graph* of a program P is a directed graph $G = \langle V, E\rangle$ where $V = \mathrm{At}(P)$ and E is a set of edges $\langle a, b\rangle$ for which there is a rule $r \in P$ such that $H(r) = a$ and $b \in B^+(r)$. A *strongly connected component* (SCC) of G is a maximal subset S of V such that there is a path from each vertex of S to every other vertex of S in the subgraph of G induced by S. When referring to the SCCs of a dependency graph in the sequel, the program will always be clear by the context. For an atom a, we use $\mathrm{SCC}(a)$ to denote the SCC containing a and $|\mathrm{SCC}(a)|$ gives it size.

For a program P and an atom $a \in \mathrm{At}(P)$, we define the respective sets of *defining rules*, *externally* defining rules, and *internally* defining rules by the following:

$$\mathrm{Def}_P(a) = \{r \in P \mid H(r) = a\} \quad (4)$$

$$\mathrm{Ext}_P(a) = \{r \in \mathrm{Def}_P(a) \mid B^+(r) \cap \mathrm{SCC}(a) = \emptyset\} \quad (5)$$

$$\mathrm{Int}_P(a) = \{r \in \mathrm{Def}_P(a) \mid B^+(r) \cap \mathrm{SCC}(a) \neq \emptyset\} \quad (6)$$

Intuitively, there is no loop between a and the atoms in the body of its externally defining rules. In contrast, there is a loop between a and some atoms in the body of its any internally defining rule $r \in \mathrm{Int}_P(a)$. Hence, we define the set of *internally supporting atoms* $\mathrm{IS}(a, r) = \mathrm{SCC}(a) \cap B^+(r)$.

Linear Constraints and Mixed Integer Programs

In mixed-integer programming (MIP), the goal is to minimize or maximize a linear function subject to a set of linear constraints. A MIP program is of the form

$$\text{minimize (or maximize) } \sum_{i=1}^{n} c_i x_i \text{ subject to}$$

$$\sum_{j=1}^{n} c_{1j} x_j \sim k_1,\ \ \ldots,\ \ \sum_{j=1}^{n} c_{mj} x_j \sim k_m$$

where each x_i is an integer or real variable and each c_i, c_{ij}, or k_i is an integer or real coefficient. The operator $\sim$ is one of $\leq$, $\geq$, and $=$. In the program, the function $\sum_{i=1}^{n} c_i x_i$ is the *objective function* and the linear constraints of the form

$$\sum_{i=1}^{n} c_i x_i \sim k \quad (7)$$

are called the *constraints* of the program.

A valuation ν from variables to numbers in the respective domains is a *solution* of (or *satisfies*) a linear constraint C of the form (7), denoted $\nu \models C$, iff $\sum_{i=1}^{n} c_i \nu(x_i) \sim k$. A valuation ν is a solution to a set of linear constraints $\mathcal{C} = \{C_1, ..., C_n\}$, denoted $\nu \models \mathcal{C}$, iff $\nu \models C_i$ for each C_i in $\mathcal{C}$. A set of linear constraints is *satisfiable* if it has a solution.

A valuation ν is a solution to a mixed integer program P, denoted $\nu \models P$, iff ν is a solution to the constraints of P that minimizes (or maximizes) the objective function. Note that the objective function could be empty which is trivially minimized (or maximized) by any valuation.

Translating Normal Programs

The goal of this section is to translate a NLP P to a MIP program, denoted $\tau^{\mathrm{N}}_{\mathrm{mip}}(P)$, so that the solutions of $\tau^{\mathrm{N}}_{\mathrm{mip}}(P)$ correspond to the answer sets of P. The translation $\tau^{\mathrm{N}}_{\mathrm{mip}}(P)$ consists of linear constraints developed below.

1. For each atom $a \in \mathrm{At}(P)$, introduce a synonymous *binary variable* a, i.e., an integer variable over the domain $\{0, 1\}$. In the sequel, the symbol a denotes the atom a when it occurs in a NLP and the binary variable a when it appears in a linear constraint.
2. For each atom $a \in \mathrm{At}(P)$, include $a = 0$ if a does not appear as the head of any rule in P, or $a = 1$ if a is a fact.
3. For each atom $a \in \mathrm{At}(P)$, include the constraint

$$\sum_{r \in \mathrm{Def}_P(a)} bd^r - |\mathrm{Def}_P(a)| \cdot a \leq 0 \quad (8)$$

   where bd^r is a binary variable for each $r \in \mathrm{Def}_P(a)$. Intuitively, bd^r represents the body of rule r and the constraint encodes that an atom a must be satisfied (valued to 1 as a binary variable) if the body of one of its defining rules is satisfied.
4. For each $r \in P$, include the following constraints[2]:

$$\sum_{b \in B^+(r)} b - \sum_{c \in B^-(r)} c - |B(r)| \cdot bd^r \geq -|B^-(r)| \quad (9)$$

$$\sum_{b \in B^+(r)} b - \sum_{c \in B^-(r)} c - bd^r \leq |B^+(r)| - 1 \quad (10)$$

   These constraints express that the body of a rule r is satisfied iff each positive and negative literal in it is satisfied.

[2]If r is an integrity constraint, the linear constraint $bd^r = 0$ is included in addition to the constraints developed here. This will also apply to the translation of WCPs to be presented in the sequel.

5. For each atom $a \in \mathrm{At}(P)$, include the constraint

$$\sum_{r \in \mathrm{Ext}_P(a)} bd^r + \sum_{r \in \mathrm{Int}_P(a)} s^r - a \geq 0 \qquad (11)$$

where s^r is a binary variable for each $r \in \mathrm{Int}_P(a)$. For intuition, the binary variable s^r represents the condition that the body of the rule r is satisfied and the respective level ranking constraints are satisfied. The constraint specifies that the atom a must be satisfied if the body of one of its externally defining rules is satisfied, or the body of one of its internally defining rules and the related level ranking constraints are both satisfied.

6. For each atom $a \in \mathrm{At}(P)$ and each $r \in \mathrm{Int}_P(a)$, include the constraints

$$bd^r - s^r \geq 0 \qquad (12)$$

$$\sum_{b \in \mathrm{IS}(a,r)} gt_{ab} - |\mathrm{IS}(a,r)| \cdot s^r \geq 0 \qquad (13)$$

where gt_{ab} is a binary variable for each $b \in \mathrm{IS}(a,r)$. The variable gt_{ab} captures the fact that the rank of a is greater than the rank of b. The two constraints above enforce that if s^r is satisfied then both bd^r and all the related ranking constraints are satisfied.

7. For each atom $a \in \mathrm{At}(P)$, each $r \in \mathrm{Int}_P(a)$, and each $b \in \mathrm{IS}(a,r)$, include the constraint

$$x_a - x_b - |\mathrm{SCC}(a)| \cdot gt_{ab} \geq 1 - |\mathrm{SCC}(a)| \qquad (14)$$

where x_a and x_b are unique integer (or real) variables introduced for a and b, respectively. The variables x_a and x_b hold the level ranks of atoms a and b, respectively. The constraint guarantees that if gt_{ab} is satisfied then $x_a > x_b$.

Then, let us illustrate how the translation $\tau^{\mathrm{N}}_{\mathrm{mip}}(P)$ captures the set $AS(P)$ using the program P from Example 1.

Example 2. The constraints resulting from (8) are:

$$bd^{r_1} + bd^{r_2} - 2a \leq 0 \qquad bd^{r_3} - b \leq 0$$
$$bd^{r_4} - c \leq 0 \qquad bd^{r_5} - d \leq 0$$

Constraints (9) and (10) yield the ones listed below

$$0 \leq b - bd^{r_1} \leq 0 \qquad (bd^{r_1} = b)$$
$$0 \leq c - bd^{r_2} \leq 0 \qquad (bd^{r_2} = c)$$
$$0 \leq a - bd^{r_3} \leq 0 \qquad (bd^{r_3} = a)$$
$$-1 \leq -d - bd^{r_4} \leq -1 \qquad (bd^{r_4} = 1 - d)$$
$$-1 \leq -c - bd^{r_4} \leq -1 \qquad (bd^{r_5} = 1 - c)$$

together with their simplified forms in parentheses. The substitution of these equalities to the inequalities above give us

$$b + c - 2a \leq 0 \qquad a - b \leq 0$$
$$1 - d - c \leq 0 \qquad 1 - c - d \leq 0$$

which have four solutions over the binary domain: $\nu_1(a) = \nu_1(b) = \nu_1(c) = \nu_1(d) = 1$; $\nu_2(d) = 0$ and $\nu_2(a) = \nu_2(b) = \nu_2(c) = 1$; $\nu_3(c) = 0$ and $\nu_3(a) = \nu_3(b) = \nu_3(d) = 1$; or $\nu_4(a) = \nu_4(b) = \nu_4(c) = 0$ and $\nu_4(d) = 1$. Actually, these solutions would correspond to the classical models $\{a,b,c,d\}$, $\{a,b,c\}$, $\{a,b,d\}$, and $\{d\}$ of P. Then, adding the constraints resulting from (11)

$$s^{r_1} + c - a \geq 0 \qquad 1 - d - c \geq 0$$
$$s^{r_3} - b \geq 0 \qquad 1 - c - d \geq 0$$

simplified by the equations concerning bd^{r_i}:s effectively excludes ν_1 which falsifies the last two inequalities. Finally, from (12), (13), and (14) we obtain in a similar way:

$$b - s^{r_1} \geq 0 \qquad a - s^{r_3} \geq 0$$
$$gt_{ab} - s^{r_1} \geq 0 \qquad gt_{ba} - s^{r_3} \geq 0$$
$$x_a - x_b - 2gt_{ab} \geq -1 \qquad x_b - x_a - 2gt_{ba} \geq -1$$

which together exclude ν_3 because it is not extendible to a solution as $\nu_3(s^{r_1}) = \nu_3(s^{r_3}) = \nu_3(gt_{ab}) = \nu_3(gt_{ba}) = 1$ would be necessary—making the residual inequalities $x_a - x_b \geq 1$ and $x_b - x_a \geq 1$ unsatisfiable. Such an extension is only possible for ν_2 and ν_4 which correspond to the answer sets of P. For ν_2, we use $\nu_2(x_a) = 1$, $\nu_2(x_b) = 2$, $\nu_2(gt_{ab}) = \nu_2(s^{r_1}) = 0$, and $\nu_2(gt_{ba}) = \nu_2(s^{r_3}) = 1$ to illustrate the feasibility of such an extension. □

Our next objective is to generalize the observations made in Example 2 and to prove that the translation $\tau^{\mathrm{N}}_{\mathrm{mip}}(P)$ is generally correct. While doing so, we extract an interpretation M^ν_P from a valuation ν by setting

$$M^\nu_P = \{a \in \mathrm{At}(P) \mid \nu(a) = 1\}. \qquad (15)$$

Theorem 3. *Let P be a normal logic program.*

(i) If there is a solution ν to $\tau^{\mathrm{N}}_{\mathrm{mip}}(P)$, then $M^\nu_P \in AS(P)$.

(ii) If $M \in AS(P)$, then there is a solution ν to $\tau^{\mathrm{N}}_{\mathrm{mip}}(P)$ such that $M = M^\nu_P$.

Proof. We prove (i). The proof of (ii) is similar.

Let ν be a solution to $\tau^{\mathrm{N}}_{\mathrm{mip}}(P)$. Let $r : a \leftarrow B(r)$ be a rule in P and $M^\nu_P \models B(r)$. Then $\nu(bd^r) = 1$ due to (9) and (10). Then $\nu(a) = 1$ due to (8). So, $a \in M^\nu_P$ and $M^\nu_P \models P$.

For any $a \in M^\nu_P$, we have $\nu(a) = 1$. Then either $\exists r \in \mathrm{Ext}_P(a)$ s.t. $\nu(bd^r) = 1$ or $\exists r \in \mathrm{Int}_P(a)$ s.t. $\nu(s^r) = 1$ due to (11). For the latter case, we have $\nu(bd^r) = 1$ due to (12). Note that $\nu(bd^r) = 1$ iff $M^\nu_P \models B(r)$ due to (9) and (10). Then we have either $\exists r \in \mathrm{Ext}_P(a)$ or $\exists r \in \mathrm{Int}_P(a)$, s.t. $M^\nu_P \models B(r)$. So, M^ν_P is a supported model of P.

Now we show that there is a level mapping of M^ν_P. For any $a \in M^\nu_P$, either $\exists r \in \mathrm{Ext}_P(a)$ such that $\nu(bd^r) = 1$ or $\exists r \in \mathrm{Int}_P(a)$ such that $\nu(s^r) = 1$, due to (11). The first case is trivial. We consider the second. Since $\nu(s^r) = 1$, then $\nu(bd^r) = 1$ due to (12). So $M^\nu_P \models B(r)$ due to (9) and (10). Then we have $r \in P_{M^\nu_P}$. Also, $\nu(x_a) > \nu(x_b)$ for each $b \in \mathrm{IS}(a,r)$ due to (13) and (14). Then, a mapping $\mathrm{lr}(\cdot)$ can be constructed component by component by setting $\mathrm{lr}(a) = m + \nu(x_a) + n$ where m is the maximum rank from the SCCs preceding $\mathrm{SCC}(a)$ and n is a constant which makes $\nu(x_b) + n$ positive for each $b \in \mathrm{SCC}(a)$. This is compatible with Definition 3 turning $\mathrm{lr}(\cdot)$ into a level ranking of M^ν_P.

Then by Theorem 1, M^ν_P is an answer set of P. □

Translating Weight Constraint Programs

Recall that we use $W(r)$ to denote the weight constraint in the body of a rule r in a WCP. The notions of defining rules, externally and internally defining rules, and the set of internally supporting atoms are naturally generalized. Moreover, for an atom a and a rule r we define the set of *externally supporting atoms* of a with respect to r by setting

$\mathrm{ES}(a,r) = W(r)^+ \setminus \mathrm{SCC}(a)$. Below, we may use W^+ and W^- to represent $W(r)^+$ and $W(r)^-$, respectively, when r is clear by the context.

The translation of a WCP P, denoted $\tau^{\mathrm{W}}_{\mathrm{mip}}(P)$, consists of linear constraints developed as follows.

1. For each atom $a \in \mathrm{At}(P)$, introduce a synonymous binary variable a.
2. For each atom $a \in \mathrm{At}(P)$, include $a = 0$ if a does not appear as the head of any rule in P, or $a = 1$ if a is a fact.
3. For each atom $a \in \mathrm{At}(P)$, include the constraint
$$\sum_{r \in \mathrm{Def}_P(a)} w^r - |\mathrm{Def}_P(a)| \cdot a \leq 0 \tag{16}$$
where w^r is a binary variable for each $r \in \mathrm{Def}_P(a)$.
4. For each rule $r \in P$, include the constraints
$$\sum_{b \in W^+} w_b \cdot b - \sum_{c \in W^-} w_c \cdot c - l \cdot w^r \geq - \sum_{c \in W^-} w_c \tag{17}$$
$$\sum_{b \in W^+} w_b \cdot b - \sum_{c \in W^-} w_c \cdot c - (\sum_{b \in W^+} w_b + \sum_{c \in W^-} w_c + 1 - l) \cdot w^r \leq l - \sum_{c \in W^-} w_c - 1 \tag{18}$$
5. For each atom $a \in \mathrm{At}(P)$, include the constraint
$$\sum_{r \in \mathrm{Ext}_P(a)} w^r + \sum_{r \in \mathrm{Int}_P(a)} w^r_s - a \geq 0 \tag{19}$$
where w^r_s is a binary variable for each $r \in \mathrm{Int}_P(a)$.
6. For each atom $a \in \mathrm{At}(P)$ and each $r \in \mathrm{Int}_P(a)$, include the constraints
$$\sum_{b \in \mathrm{ES}(a,r)} w_b \cdot b + \sum_{b \in \mathrm{IS}(a,r)} w_b \cdot s_b - \sum_{c \in W^-} w_c \cdot c - l \cdot w^r_s \geq - \sum_{c \in W^-} w_c \tag{20}$$
$$\sum_{b \in \mathrm{ES}(a,r)} w_b \cdot b + \sum_{b \in \mathrm{IS}(a,r)} w_b \cdot s_b - \sum_{c \in W^-} w_c \cdot c - (\sum_{b \in W^+} w_b + \sum_{c \in W^-} w_c + 1 - l) \cdot w^r_s \leq l - \sum_{c \in W^-} w_c - 1 \tag{21}$$
where s_b is a binary variable for each $b \in \mathrm{IS}(a,r)$.
7. For each atom $a \in \mathrm{At}(P)$, each $r \in \mathrm{Int}_P(a)$, and each atom $b \in \mathrm{IS}(a,r)$, include the constraints
$$s_b - b \leq 0 \tag{22}$$
$$x_a - x_b - |\mathrm{SCC}(a)| \cdot s_b \geq 1 - |\mathrm{SCC}(a)| \tag{23}$$
where x_a and x_b are unique integer variables introduced for a and b, respectively.

The idea of $\tau^{\mathrm{W}}_{\mathrm{mip}}(P)$ is very similar to that of $\tau^{\mathrm{N}}_{\mathrm{mip}}(P)$, where the binary variables w^r, w^r_s, and s_b are counterparts of bd^r, s^r, and gt_{ab}, respectively and the constraints (16)–(18) are counterparts of (8)–(10), respectively. The subtleties lie in the constraints (19)–(23) which encode the constraints (11)–(14) in the more complex case of WCPs as illustrated below.

For an atom a and its internally defining rule r, the difficulty in the translation is how to encode the relations between the rank of a and the ranks of the atoms in $W(r)^+$: the rank of a does not have to be greater than the rank of each atom in $W(r)^+$, but only the ranks of atoms in some subset of $W(r)^+$ that suffices to satisfy $W(r)$. This is captured by constraints (19)–(23).

Example 3. Consider a program P consisting of the rules:
$$a \leftarrow 1[b = 1, c = 1]. \quad b \leftarrow a. \quad c \leftarrow a. \quad b.$$
The essential aspect of the first rule, say r_1, is that if a is derived then b is derived earlier than a, **or** c is derived earlier than a. This implication is encoded by the constraint (19) in the translation $\tau^{\mathrm{W}}_{\mathrm{mip}}(P)$ which simplifies in this case to
$$w^{r_1}_s - a \geq 0 \tag{24}$$
It is easy to check that if $a = 1$ then $w^{r_1}_s = 1$.

The disjunctive aspect is encoded by (20) and (21), i.e.:
$$s_b + s_c - w^{r_1}_s \geq 0 \tag{25}$$
$$s_b + s_c - 2w^{r_1}_s \leq 0 \tag{26}$$
It can be seen that $w^{r_1}_s = 1$ iff either $s_b = 1$ or $s_c = 1$. The conjunctive aspects are encoded by (22) and (23):
$$s_b - b \leq 0 \qquad x_a - x_b - 3s_b \geq -2 \tag{27}$$
$$s_c - c \leq 0 \qquad x_a - x_c - 3s_c \geq -2 \tag{28}$$
It can be verified that if $s_b = 1$ then $b = 1$ and $x_a > x_b$ and the constraints for s_c act similarly.

The constraints from (24) to (28) together with the other constraints in $\tau^{\mathrm{W}}_{\mathrm{mip}}(P)$ have a solution ν with $\nu(a) = \nu(b) = \nu(c) = 1$ corresponding to the answer set $\{a, b, c\}$. □

Theorem 4. *Let P be a weight constraint program.*

(i) If there is a solution ν to $\tau^{\mathrm{W}}_{\mathrm{mip}}(P)$, then M^ν_P defined by (15) belongs to $AS(P)$.

(ii) If $M \in AS(P)$, then there is a solution ν to $\tau^{\mathrm{W}}_{\mathrm{mip}}(P)$ such that $M = M^\nu_P$ as defined by (15).

The theorem follows from the translation $\tau^{\mathrm{W}}_{\mathrm{mip}}(P)$, Definition 6, and Theorem 2. The proof of the theorem is similar to that of Theorem 3.

As a WCP can be translated to a NLP (Janhunen and Niemelä 2011), a straightforward way to translate a WCP to a MIP program is first translating it to a NLP and then translating the NLP to a MIP program using the translation in the previous section. But transforming a WCP to a NLP introduces extra atoms and rules that turn the straightforward transformation inefficient. The native translation of a WCP provided in this section avoids many extra atoms and rules.

NLPs are a special case of WCPs as formula (1) can be obtained from (2) by substituting l with $m + n$ and each weight by 1. We present the translations for NLPs and

WCPs respectively, since the integrated language ASP(LC), as to be introduced later, is based on the syntax of NLPs and hence we can exploit the translation of NLPs in a modular way. The translation of WCPs is devised orthogonally to the translation of ASP(LC) programs.

Translating Optimization Statements

Optimization statements in an ASP program take the form:

minimize (or maximize)

$$[a_1 = w_{a_1}, ..., a_m = w_{a_m}, \text{not } b_1 = w_{b_1}, ..., \text{not } b_n = w_{b_n}]$$

where each a_i and each $\text{not } b_j$ is a literal and w_{a_i}:s and w_{b_j}:s are weights associated with them, respectively.

An answer set is *optimal* if the sum of weights of literals that hold is minimal (or maximal) as required by the statement, amongst all answer sets of the given program. Multiple optimization statements can be reduced to a single one as shown in (Simons, Niemelä, and Soininen 2002). An optimization statement can be straightforwardly encoded as a MIP objective function

minimize (or maximize)

$$w_{a_1} \cdot a_1 + ... + w_{a_m} \cdot a_n - w_{b_1} \cdot b_1 - ... - w_{b_n} \cdot b_n \quad (29)$$

where each a_i and each b_j is a binary variable.

The Integrated Language

In this section, we extend normal logic programs to include linear constraints. Weight constraint programs can be extended in a similar way. However, for the sake of simplicity, we present the case of NLPs and linear constraints only.

A *logic program with linear constraints*, or an ASP(LC) program, is a set of rules of the form

$$a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n, t_1, \ldots, t_l \quad (30)$$

where each a, b_i, and c_j is a propositional atom. As before, we use $\text{At}(P)$ to denote the propositional atoms in a program P. Each t_k, called *theory atom*, is a linear constraint of the form (7). The variables in theory atoms, i.e., the MIP variables in linear constraints, such as x_i in the formula (7), are called *free constants*[3]. For a rule r of the form (30), the set of theory atoms $\{t_1, \ldots, t_l\}$ is called the *theory* part of the body of r and denoted by $B^t(r)$. Using our previous definitions from the section on preliminaries, we obtain $B(r) = B^+(r) \cup \{\text{not } c \mid c \in B^-(r)\} \cup B^t(r)$. Comparing (30) to (1), the integrated language extends the pure ASP language in terms of theory atoms used as additional conditions in rule bodies.

For a theory atom t, we use $\neg t$ to denote the linear constraint of the form (7) where $\sim$ is the complementary of the operator in t. For a ASP(LC) program P, an *interpretation* I is a pair $\langle M, T\rangle$ where $M \subseteq \text{At}(P)$ and T is a subset of theory atoms appearing in P such that $T \cup \bar{T}$ is satisfiable in linear arithmetics where $\bar{T}$ is the set of linear constraints containing $\neg t$ for each theory atom t appearing in P but not in T. An interpretation $I = \langle M, T\rangle$ satisfies a propositional atom, a literal, a theory atom, or a rule if and only if the set $M \cup T$ satisfies them as defined in the section of normal logic programs, respectively, and they will be denoted in the same way as in the previous section.

An interpretation I is a *model* of P, denoted $I \models P$, if and only if I satisfies all rules of P. Now, we generalize Definitions 1 and 2 as follows.

Definition 7. *Let P be a logic program with linear constraints and $\langle M, T\rangle$ an interpretation. The* reduct *of P with respect to $\langle M, T\rangle$ is defined by $P^{\langle M,T\rangle} = \{H(r) \leftarrow B^+(r) \mid r \in P,\ B^-(r) \cap M = \emptyset,\ \text{and } B^t(r) \subseteq T\}$.*

Definition 8. *An interpretation $\langle M, T\rangle$ of a program P is an* answer set *of P, iff $\langle M, T\rangle \models P$ and M is the minimal model of $P^{\langle M,T\rangle}$.*

Example 4. Let P be a program consisting of the rules:

$$\begin{array}{ll} a \leftarrow x > z. & \leftarrow \text{not } a. \\ b \leftarrow x \leq y. & b \leftarrow c. \\ c \leftarrow b, y \leq z. & \end{array}$$

Here a, b, and c are propositional atoms whereas the symbols x, y, and z are free constants. Consider the interpretation $I_1 = (\{a\}, \{x > z\})$. It is an answer set of P, since the set $\{(x > z), \neg(x \leq y), \neg(y \leq z)\}$ is satisfiable in linear arithmetics, $I_1 \models P$, and the set $\{a\}$ is the minimal model of the reduct $P^{I_1} = \{a \leftarrow;\ b \leftarrow c\}$. On the other hand, the interpretation $I_2 = (\{a, b, c\}, \{x > z, x \leq y, y \leq z\})$ is not an answer set, since $\{(x > z), (x \leq y), (y \leq z)\}$ is not satisfiable. Finally, consider $I_3 = (\{a, b, c\}, \{x > z, y \leq z\})$. It is not an answer set as $\{a, b, c\}$ is not the minimal model of the reduct $P^{I_3} = \{a \leftarrow;\ b \leftarrow c;\ c \leftarrow b\}$. □

It can be verified that the semantics given by Definition 8 coincides with Definition 2 if no theory atoms are present.

So far we have discussed only ground (propositional) logic programs. The treatment can generalized to *non-ground programs* by using Herbrand interpretations. We use the logic program convention to write ASP variables starting with a capital letter. For example,

$$\mathsf{r}(X) \leftarrow \mathsf{r}(Y), \mathsf{edge}(X, Y). \quad (31)$$

is a rule with two ASP variables X and Y. The idea is to treat such a rule with variables as a shorthand for its Herbrand instantiations where the variables in the rule are replaced by terms in the Herbrand universe of the program. The universe contains all ground terms constructible out of constant and function symbols appearing in the program. Now answer sets of a non-ground program are defined as the answer sets of the full grounding of the program, i.e., the set of all possible Herbrand instantiations of its rules. Notice that a function symbol in a program makes its Herbrand universe infinite and leads to undecidability. Hence, in order to preserve decidability, restrictions on the use of function symbols are needed. Here we assume for simplicity that function symbols are not allowed.

Note that the free constants in the theory atoms, e.g., x, y, and z in Example 4, are treated differently from ASP variables and ASP constants in the rules as they can vary over

[3]The term free constant is to distinguish MIP variables from ASP variables in the integrated language, borrowed from the SAT Modulo Theories literature (Nieuwenhuis, Oliveras, and Tinelli 2006). Further details are discussed at the end of this section.

a range of values when determining the satisfiability of the corresponding theory atoms. However, our non-ground integrated language supports interaction of free constants and ASP variables to enable uniform encodings where arrays of related free constants can be easily represented. The idea is to allow free constants *indexed* by ASP variables such as the free constants $x(Q_1)$ and $x(Q_2)$ in the rule

$$\leftarrow \mathsf{queen}(Q_1), \mathsf{queen}(Q_2), x(Q_1) - x(Q_2) = 0.$$

Once such a rule is grounded, the ASP variables Q_1 and Q_2 will be substituted by ground terms from the Herbrand universe of the program in question. The resulting ground terms of the form $x(t)$ will be treated as free constants when the satisfaction of theory atoms is considered.

Encoding Problems in ASP(LC)

Next we illustrate the use and advantages of our integrated language ASP(LC) by discussing a scheduling problem, namely the simplified *job shop* problem, where the goal is to find a schedule for a number of tasks. In the problem, the *time domain*, i.e., the range of values of possible starting and ending points of tasks, can be taken to be discrete but is usually very large compared to the number of tasks. Similar characteristics are typical in many real-life applications such as resource allocation, timetabling, and complex manufacturing with multiple production lines.

Example 5 *(Job shop problem).* We are given a number of tasks and a deadline. Assume that the problem instance is given as a set of facts of the form $\mathsf{tk}(i, e, d)$ denoting that the task i has the *earliest* starting time e and duration d that it takes to complete as well as a constant $deadline$ denoting the deadline in the problem. The execution of the tasks cannot overlap. The goal is to find a schedule to finish all the tasks before the deadline. □

We start by analyzing difficulties arising in such applications when using standard ASP. To encode the problem in pure ASP, a typical approach is to introduce predicates $\mathsf{time}(T)$, $\mathsf{st}(I, T)$, and $\mathsf{en}(I, T)$ to denote the domain of time T, the starting and ending times of task I, respectively. In addition, choice rules for selecting possible starting and ending times are needed plus a set of integrity constraints eliminating non-valid combinations of starting and ending times.

Although the constraints can be stated very intuitively and compactly using rules with variables, ASP solvers have difficulties in handling such constraints. This is because they are based on a two phase implementation technique where they first perform the grounding step, i.e., compute a relevant part of the full grounding preserving answer sets and then use a search algorithm for computing answer sets of the grounded program. However, even state-of-the-art grounding techniques produce very big grounded programs when the time domain is large. We illustrate this with an integrity constraint enforcing enough execution time for each task:

$$\leftarrow \mathsf{st}(I, T_1), \mathsf{en}(I, T_2), T_2 - T_1 < D, \mathsf{tk}(I, E, D), \mathsf{time}(T_1), \mathsf{time}(T_2). \quad (32)$$

Here we have two variables T_1 and T_2 ranging over the time domain and the number of rules after grounding is quadratic in the size of the time domain. Hence, if the size is in the thousands, the number of ground rules is in the millions.

Now we show how to solve the problem using our integrated language ASP(LC). In ASP(LC), we can treat the starting and ending times as free constants ranging over integers and index them by the tasks, i.e., we introduce terms $s(I)$ and $e(I)$ to denote the starting and ending times of task I, respectively. Then the problem can be encoded in ASP(LC) in a very compact way:

$$\leftarrow s(I) < E, \mathsf{tk}(I, E, D). \quad (33)$$

$$\leftarrow e(I) - s(I) < D, \mathsf{tk}(I, E, D). \quad (34)$$

$$\leftarrow s(I_1) \leq s(I_2), s(I_2) \leq e(I_1), \mathsf{tk}(I_1, E, D), \mathsf{tk}(I_2, E, D). \quad (35)$$

$$\leftarrow e(I) > deadline, \mathsf{tk}(I, E, D). \quad (36)$$

The rule (33) says that no task can begin before its earliest starting time; (34) enforces enough execution time for each task; (35) guarantees no overlap in task executions; and (36) requires tasks to be finished before the deadline.

When comparing with the standard ASP approach, the main difference is that the encoding does not contain any ASP variables ranging over the time domain but the starting and ending times are treated as free constants ranging over integers. In the next section we show how to implement such rules so that free constants need not to be grounded over integers. This is achieved by translating the rules completely to a set of linear constraints.

Translating ASP(LC) Programs

The translation of an ASP(LC) program consists of two steps. First, an ASP(LC) program is *normalized* to a normal logic program and a *naming program*. Second, the normal logic program and the naming program are translated to sets of linear constraints, respectively. The union of the sets of linear constraints is the translation of the ASP(LC) program.

Given an ASP(LC) program P, a rule of the form (30) is normalized as a rule

$$a \leftarrow d_1, ..., d_l, b_1, ..., b_m, \text{not } c_1, ..., \text{not } c_n. \quad (37)$$

and a set of *naming rules*

$$d_i \leftarrow t_i. \quad (38)$$

where i ranges from 1 to l and each d_i is a newly introduced propositional atom used as a name for t_i.

Given an ASP(LC) program P, the respective *normalized* program $\mathrm{NT}(P)$ consists of two parts: the *normal* part $\mathrm{NPart}(P)$, a set of normalized rules of the form (37), and the *theory part* $\mathrm{TPart}(P)$, a set of rules of the form (38). According to the semantics of ASP(LC) programs, we can establish the following.

Theorem 5. *Let P be a logic program with linear constraints and $\langle M, T\rangle$ an interpretation. Then $\langle M, T\rangle \in AS(P)$ iff there is a set $D \subseteq \mathrm{At}(\mathrm{TPart}(\mathrm{NT}(P)))$ such that $(M \cup D) \in AS(\mathrm{NPart}(\mathrm{NT}(P)) \cup D)$ and $\langle D, T\rangle \in AS(\mathrm{TPart}(\mathrm{NT}(P)))$.*

As discussed above, we will use naming programs to formalize the translation of the theory part of a normalized program. In general, naming rules are of the form

$$d \leftarrow t \tag{39}$$

where d is a propositional atom and t is a theory atom. Moreover, we assume that rules in a naming program do not share any propositional atoms. So, the atom d can be intuitively viewed as a unique *name* for the theory atom t.

We use *indicator constraints* for a convenient presentation of our translation. A indicator constraint is of the form

$$b = v \rightarrow C \tag{40}$$

where b is a boolean variable, v is a value from $\{0, 1\}$, and C is a linear constraint of the form (7). Intuitively, the value of b indicates whether or not the constraint C is satisfied. For example, $b = 1 \rightarrow x \geq 0$ says that if the value of b is 1 then x must be greater than or equal to 0. Note that indicator constraints can be encoded as linear constraints using the so-called *big-M formulations*.[4]

Let P be a naming program. The translation of P, denoted $\tau^{\mathrm{Na}}_{\mathrm{mip}}(P)$, consists of the following constraints for each rule r of the form (39) in P.

$$d = 1 \rightarrow t \tag{41}$$

$$d = 0 \rightarrow \neg t \tag{42}$$

The theorem below follows from the translation.

Theorem 6. *Let P be a naming program.*

(i) If there is a solution ν to $\tau^{\mathrm{Na}}_{\mathrm{mip}}(P)$, then $\langle D, T\rangle \in AS(P)$ where $\langle D, T\rangle$ is an interpretation, D equals to M^{ν}_{P} defined by (15), and $T = \{t \mid d \leftarrow t \in P \text{ and } \nu \models t\}$.

(ii) If $\langle D, T\rangle \in AS(P)$, then there is a solution ν to $\tau^{\mathrm{Na}}_{\mathrm{mip}}(P)$ such that $D = M^{\nu}_{P}$ as defined by (15) and

$$T = \{t \mid d \leftarrow t \in P \text{ and } \nu \models t\}.$$

Now, we are ready to define the translation $\tau_{\mathrm{mip}}(P)$ of an ASP(LC) program P as the union

$$\tau^{\mathrm{N}}_{\mathrm{mip}}(\mathrm{NPart}(\mathrm{NT}(P))) \cup \tau^{\mathrm{Na}}_{\mathrm{mip}}(\mathrm{TPart}(\mathrm{NT}(P)))$$

where $\tau^{\mathrm{N}}_{\mathrm{mip}}(\cdot)$ is applied as defined in the translation of NLPs except that naming atoms d are not assigned to 0. The following result is implied by Theorems 3, 5, and 6.

Theorem 7. *Let P be an ASP(LC) program.*

(i) If there is solution ν to $\tau_{\mathrm{mip}}(P)$, then $\langle M, T\rangle \in AS(P)$ where $\langle M, T\rangle$ is an interpretation, $M = M^{\nu}_{P}$ as defined by (15), and $T = \{t \mid d \leftarrow t \in \mathrm{NT}(P) \text{ and } \nu \models t\}$.

(ii) If $\langle M, T\rangle \in AS(P)$, then there is a solution ν to $\tau_{\mathrm{mip}}(P)$ such that $M = M^{\nu}_{P}$ as defined by (15) and

$$T = \{t \mid d \leftarrow t \in \mathrm{NT}(P) \text{ and } \nu \models t\}.$$

Given Theorem 7, the integrated language ASP(LC) can be implemented using a suitable back-end solver for linear constraints. There are many MIP solvers available for this purpose. However, notice that in the translation strict inequality is used in one constraint (42) while typical MIP implementations allow only non-strict inequalities. Hence, the integrated language with *linear constraints over integers* can be implemented using a MIP solver as for integers strict inequalities can be replaced by non-strict ones straightforwardly. This allows also the sophisticated MIP optimization techniques to be used for implementing the ASP optimization statements. The integrated language with *linear constraints over reals* can be handled, e.g., using an extended Simplex method (Dutertre and de Moura 2006) for deciding satisfiability of such constraints. However, the method does not directly support implementing the ASP optimization statements simultaneously.

As regards the space efficiency of our translations, it is easy to verify the following result where the size of logic program P means the number of atom occurrences in it.

Theorem 8. *Let P be a normal, weight constraint, or ASP(LC) program. The numbers of variables and constraints in the respective translations $\tau^{\mathrm{N}}_{\mathrm{mip}}(P)$, $\tau^{\mathrm{W}}_{\mathrm{mip}}(P)$, and $\tau_{\mathrm{mip}}(P)$ are linear in the size of P.*

Implementation

We implemented a system called MINGO that supports the extended ASP language ASP(LC), using existing off-the-shelf ASP and MIP tools, and other Unix/Linux tools. The architecture of the implementation is outlined as below.

(i) An input program is grounded by the existing ASP grounder GRINGO, where theory atoms are represented by special *relational* predicates with reserved names, e.g., $2a + 3b \leq 7$ is represented by $\mathsf{mleq}(2, a, 3, b, 7)$ and special *domain* predicates are reserved to declare the types of the arguments of the relational predicates, e.g., $\mathsf{int}(a)$ declares that a is a free integer constant and not an ASP constant.

(ii) The ground program is then translated into a MIP program using a tool LP2MIP that implements our translation presented in previous sections.

(iii) The relevant type information, e.g., the $\mathsf{int}(a)$, is extracted from the ground program and incorporated as type declarations to the MIP program.

(iv) The theory atoms are extracted from the ground program and expanded into respective linear constraints using a standard macro processor M4.

(v) The MIP solver CPLEX is then invoked to compute a solution to the MIP program (if any) and mapped back to an answer set of the original program. Note that CPLEX supports non-strict inequalities only and our system supports theory atoms involving free integer constants only.

This implementation is justified by Theorem 7.

Experiments

In this section, we compare MINGO with the state-of-the-art ASP solvers CLINGCON (Gebser, Ostrowski, and Schaub 2009) and CLINGO (Gebser, Kaufmann, and Schaub 2009) using a number of benchmarks. The results are reported in Tables 1 and 2. The system CLINGO supports pure ASP language only and CLINGCON is an extension of CLINGO that supports linear constraints with infinite domains. The experiments are run on a Linux node (in a shared cluster) with 2.4GHz CPUs and 2.7G RAM. The cut off time is set to 600

[4] Please refer to the User's Manual of CPLEX v12.2.

seconds and running times greater than that are indicated by a dash '–' in the tables.

The benchmarks[5] were designed to evaluate three aspects of the performance of MINGO: optimization, dealing with variables with large domains, and both. The first five benchmarks in Table 1 are optimization problems, the three in the middle are problems with large domains, and the last two involve both optimization and variables with large domains.

In Table 1, the running time and the number of solved instances are reported for each benchmark. The second column contains the total number of instances for each benchmark. The following columns present numbers of solved instances and the average time to solve them for each system. We can see that MINGO is more efficient than other systems for optimization problems. Specifically, it is orders of magnitude faster than CLINGCON and CLINGO for the *knapsack* and *subset sum* problems and it can solve a number of instances for the *dominating set* and *traveling sales person* problems while the others cannot. In these optimization benchmarks, all constraints are encoded in pure ASP. Regardless of that CPLEX is performing surprisingly well.

For problems involving large variable domains, CLINGCON is the best one and MINGO is more efficient than CLINGO. The lesson we learned here is that encoding benchmark problems in the integrated language and making use of linear constraints is preferable in answer set programming.

For problems involving both optimization and large variable domains, MINGO is the best for maximization but the worst for minimization. This may be due to the optimization algorithm used in CLINGO based systems. For this group, CLINGO is better than CLINGCON. Our conjecture is that CLINGCON incurs overhead due to interaction between the ASP solver and the constraint solver.

In Table 2, we present the size growth tendency for the ground programs of the job shop problem. Recall that the job shop problem involves a large time domain. In the encodings used for MINGO, the starting and ending times are free constants and in that for CLINGO, they are ASP variables. The first column of the table is the size of the time domain. The second and third columns show how the size of the ground programs increases. The size of the smallest ground program is set to 1 and the ratios of the bigger ground programs follow. It can be seen that the size increases linearly for the encoding in ASP(LC) and cubicly for the encoding in pure ASP language. Similar behavior can be observed for other problems such as the *newspaper* and *sorting* problems and we also expect similar advantage that can be achieved by using CLINGCON. This indicates the reason why the use of free constants of MINGO or the similar constructs of CLINGCON is preferable.

We also tried out 516 benchmark instances from the category of NP-complete problems used in the Second ASP Competition (Denecker et al. 2009). In the encodings of these benchmarks, primitives provided by MINGO are not used and the linear constraints are encoded in pure ASP language. It turns out that MINGO is less efficient than CLINGO: MINGO finishes 306 while CLINGO finishes 465 out of 516. It seems that for these kinds of problems involving only binary variables and constraints, the performance of CPLEX is not as competitive as state-of-the-art ASP solvers.

Selecting a solver for a particular answer set computation task is a non-trivial issue. Our experiences suggest the following: In transformation based approaches, it is highly important to choose a solver for which a concise, preferably linear transformation from ASP languages to the solver input language is available as achieved in this paper. Generally speaking, it is hard to specify the most suitable solver for a particular problem in advance. In practice, we recommend to experiment with a number of solvers on small problem instances before determining one for solving larger instances.

[5]The MINGO system and benchmarks can be found under `http://research.ics.aalto.fi/software/asp/`

Related Work

In (Bell et al. 1994), a translation from a logic program to a MIP program is presented. In the approach, the solutions to the MIP program do not correspond to the answer sets of the logic program but its supported models. To compute the answer sets, an extra check procedure is needed to filter out the supported models that are not answer sets. This is because the translation actually corresponds to the *completion* of a logic program and no ranking constraints are included.

There are translations from logic programs to clauses in classical propositional logic (Lin and Zhao 2004) and pseudo-Boolean constraints (Liu and Truszczyński 2006), where the solutions to the translated formulas correspond to the answer sets of the program. However, these translations are exponential, i.e., the number of formulas resulting from a program is exponential in the size of the program, since they are based on a notion of *loop formulas* the number of which is exponential in the size of a program. Janhunen (2004) presents a sub-quadratic translation which is basically linear but involves a logarithmic factor. In contrast, our translation is linear in the size of a program as the constraints enforced by the loop formulas are captured by the ranking constraints the number of which is linear in the size of a program.

Translations from normal logic programs to *difference logic* are studied in (Niemelä 2008) and (Janhunen, Niemelä, and Sevalnev 2009). Actually, the linear constraint encodings presented in this paper lead to a simpler translation from logic programs to difference logic theories. Given a normal program P, an analogous translation into difference logic takes place as follows.

1. For each $r \in P$, include a formula
$$bd^r \leftrightarrow B^+(r) \wedge \neg B^-(r) \tag{43}$$
where bd^r is a propositional atom introduced for $B(r)$.
2. For each atom $a \in \mathrm{At}(P)$, include formulas
$$\bigvee_{r \in \mathrm{Def}_P(a)} bd^r \rightarrow a \tag{44}$$
$$a \rightarrow \bigvee_{r \in \mathrm{Ext}_P(a)} bd^r \vee \bigvee_{r \in \mathrm{Int}_P(a)} (bd^r \wedge \bigwedge_{b \in \mathrm{IS}(a,r)} x_a > x_b) \tag{45}$$

Benchmark	Instance	MINGO		CLINGCON		CLINGO	
		solved	time	solved	time	solved	time
Dominating Set	29	6	370.42	0	–	0	–
Knapsack	100	100	8.92	50	109.98	80	140.61
Subset Sum	130	130	0.02	130	2.64	110	55.55
Traveling Sales	29	12	313.90	0	–	0	–
Warehouse	100	100	74.19	30	168.50	30	163.29
Job Shop	100	100	7.79	100	0.03	70	118.83
Newspaper	100	100	0.66	100	0.01	60	42.79
Sorting	100	100	22.40	100	0.42	50	52.02
Routing Max	100	100	12.25	100	265.12	100	162.28
Routing Min	100	80	101.61	90	0.82	100	0.01

Table 1: Performance Comparison

Job Shop		
Range	MINGO	CLINGO
50	1	1
100	1.97	8.14
150	2.97	28.48
200	3.98	68.53
250	4.98	135.04
300	5.98	235.22
350	6.98	375.06
400	7.98	562.33
450	8.98	801.19
500	9.98	1098.21

Table 2: Space Complexity

where x_a and x_b are integer variables in difference logic.

We denote the set of formulas $\{(43), (44), (45)\}$ by $\tau_{\mathrm{dl}}^{\mathrm{N}}(P)$. The difference between $\tau_{\mathrm{dl}}^{\mathrm{N}}(P)$ and the translation in (Niemelä 2008) goes back to the formula (45). In $\tau_{\mathrm{dl}}^{\mathrm{N}}(P)$, we distinguish the externally and internally defining rules and the ranking constraints are enforced only for the atoms in the bodies of internally defining rules, while in the previous translation, the external and internal defining rules are not distinguished and the ranking constraints are enforced for atoms in the bodies of both of them. The translation in (Janhunen, Niemelä, and Sevalnev 2009) also distinguishes the externally and internally defining rules. But, to formalize the difference, a number of extra propositional atoms and formulas are introduced which are not used in $\tau_{\mathrm{dl}}^{\mathrm{N}}(P)$.

We also note the work on translations between propositional logic and integer programming formulas (Li, Zhou, and Du 2004). We investigate transformations of ASP rules to MIP formulas that enable the integration of ASP and MIP.

Conclusion

In this paper, we relate ASP and MIP by establishing translations from ASP to MIP. These translations provide a promising way to extend ASP with linear constraints. We propose an extended language with linear constraints making infinite domain variables available for modeling in the ASP framework and facilitating compact encodings of problems involving large variable domains. We have implemented an answer set computation approach where answer sets of programs in the extended language are computed using a MIP solver back-end. The effectiveness of the approach is demonstrated in a number of experiments. These results create new opportunities to combine the expressive and computational capacities of ASP and MIP paradigms for knowledge representation and reasoning—opening up interesting possibilities for further investigations.

References

Apt, K.; Blair, H.; and Walker, A. 1988. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases*. Morgan Kaufmann. Chapter 2.

Balduccini, M. 2011. Industrial-size scheduling with ASP+CP. In *Proc. LPNMR'11*, volume 6645 of *LNCS*, 284–296.

Bell, C.; Nerode, A.; Ng, R. T.; and Subrahmanian, V. S. 1994. Mixed integer programming methods for computing nonmonotonic deductive databases. *J. ACM* 41(6):1178–1215.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Denecker, M.; Vennekens, J.; Bond, S.; Gebser, M.; and Truszczynski, M. 2009. The second answer set programming competition. In *Proc. LPNMR'09*, 637–654.

Dutertre, B., and de Moura, L. M. 2006. A fast linear-arithmetic solver for DPLL(T). In *Proc. CAV*, 81–94.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2009. Conflict-driven answer set solver *clasp*: Progress report. In *Proc. LPNMR'09*, 509–514.

Gebser, M.; Ostrowski, M.; and Schaub, T. 2009. Constraint answer set solving. In *Proc. ICLP'09*, 235–249.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. ICLP'88*, 1070–1080.

Janhunen, T., and Niemelä, T. 2011. Compact translations of non-disjunctive answer set programs to propositional clauses. In *Michael Gelfond's 65th Anniversary Festscrift*, volume 6565 of *LNCS*, 111–130.

Janhunen, T.; Niemelä, I.; and Sevalnev, M. 2009. Computing stable models via reductions to difference logic. In *Proc. LPNMR'09*, 142–154.

Janhunen, T. 2004. Representing normal programs with clauses. In *Proc. ECAI'04*, 358–362.

Li, R.; Zhou, D.; and Du, D. 2004. Satisfiability and integer programming as complementary tools. In *Proc. ASP-DAC'04*, 879–882.

Lin, F., and Zhao, Y. 2004. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence* 157(1-2):115–137.

Liu, L., and Truszczyński, M. 2006. Properties and appli-

cations of programs with monotone and convex constraints. *JAIR* 7:299–334.

Liu, G., and You, J.-H. 2010. Level mapping induced loop formulas for weight constraint and aggregate logic programs. *Fundam. Inform.* 101(3):237–255.

Marek, V., and Subrahmanian, V. 1992. The relationship between stable, supported, default and autoepistemic semantics for general logic programs. *TCS* 103:365–386.

Marek, V.; Niemelä, I.; and Truszczyński, M. 2008. Logic programs with monotone abstract constraint atoms. *TPLP* 8(2):167–199.

Mellarkod, V. S.; Gelfond, M.; and Zhang, Y. 2008. Integrating answer set programming and constraint logic programming. *Annals of Math. and AI* 53(1-4):251–287.

Niemelä, I. 2008. Stable models and difference logic. *Annals of Math. and AI* 53(1-4):313–329.

Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2006. Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *JACM* 53(6):937–977.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2):181–234.