Ann Math Artif Intell (2010) 60:341–380
DOI 10.1007/s10472-011-9237-y

Weight constraint programs with evaluable functions

Yisong Wang · Jia-Huai You · Fangzhen Lin · Li Yan Yuan · Mingyi Zhang

Published online: 1 June 2011

Abstract In the current practice of Answer Set Programming (ASP), evaluable functions are represented as special kinds of relations. This often makes the resulting program unnecessarily large when instantiated over a large domain. The extra constraints needed to enforce the relation as a function also make the logic program less transparent. In this paper, we consider adding evaluable functions to answer set logic programs. The class of logic programs that we consider here is that of weight constraint programs, which are widely used in ASP. We propose an answer set semantics to these extended weight constraint programs and define loop completion to characterize the semantics. Computationally, we provide a translation from loop completions of these programs to instances of the Constraint Satisfaction Problem (CSP) and use the off-the-shelf CSP solvers to compute the answer sets of these programs. A main advantage of this approach is that global constraints implemented in such CSP solvers become available to ASP. The approach also provides a new encoding for CSP problems in the style of weight constraint programs. We have implemented a prototype system based on these results, and our experiments show that this prototype system competes well with the state-of-the-art ASP solvers. In addition, we illustrate the utilities of global constraints in the ASP context.

This is an extension of a preliminary version that appeared in the Proceedings of LPNMR'09 [43] drawing some results from [26].

Y. Wang (✉)
Department of Computer Science, Guizhou University, Guiyang, China
e-mail: yswang168@gmail.com

J.-H. You · L. Y. Yuan
Department of Computing Science, University of Alberta, Edmonton, AB, Canada

F. Lin
Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong

M. Zhang
Guizhou Academy of Sciences, Guiyang, China

Keywords Answer set programming · Weight constraints · Evaluable functions · Constraint satisfaction problem

Mathematics Subject Classifications (2010) 68T20 · 68T27 · 68T30

1 Introduction

Logic programming based on the stable model/answer set semantics, commonly called *answer set programming* (ASP), has proved to be a promising paradigm for declarative knowledge representation. The general idea is to encode a problem by a logic program such that the answer sets of the program correspond to the solutions of the problem [1, 20, 31, 34]. One particular class of such logic programs, called weight constraint programs (sometimes also called LPARSE programs) [40], has been developed, and found applications in a number of application areas such as diagnosis, scheduling, planning and so on (see, e.g., [15]). Some efficient ASP solvers, such as SMODELS,[1] CMODELS,[2] and CLASP,[3] have been developed for these programs.

In the current practice of ASP, mappings from their given domains to ranges are represented by special kinds of relations. For instance, to encode the graph coloring problem, instead of a unary function, say $color(x)$ that maps vertices to colors, one uses a binary relation, say $color(x, c)$, to represent that the vertex x is assigned the color c. For this to work, one needs to add some axioms saying that the predicate $color(x, c)$ is in fact functional. In general, the approach of encoding functions by relations tends to increase the sizes of instantiated programs, both in terms of the number of atoms and the number of rules. For lack of a standard terminology, let us call these functions *evaluable functions*.

Syntactically, functions have been allowed in logic programming from the very beginning [30]. However, they are normally interpreted under the Herbrand universe of a given language. For instance, in Prolog, one cannot declare a fact like "$f(a) = a$". In fact, the query "$f(a) = a$" will always receive a "no" answer. In other words, functions have a fixed interpretation and their values cannot be changed by the user. The same holds for most of the work in ASP where function symbols are allowed (see, e.g. [2, 4, 9, 14]). These approaches aim at increasing the expressive power of ASP by adding the capacity of defining recursive data structures over infinite Herbrand domains. While the language of LPARSE allows function symbols, terms constructed of these functions are just names standing for constants. One noticeable exception is the pure functional language studied in [6, 7], where programs are built from evaluable functions, excluding predicates.

In this paper, we consider adding evaluable functions into weight constraint programs, called *weight constraint programs with evaluable functions*, along the line of extending normal logic programs with evaluable functions [26]. For these programs we propose an answer set semantics and the notion of completion and loop formulas that are a generalization to that of weight constraint programs respectively,

[1] http://www.tcs.hut.fi/Software/smodels/

[2] http://userweb.cs.utexas.edu/users/tag/cmodels.html

[3] http://www.cs.uni-potsdam.de/clasp/

and show that answer sets of a weight constraint program with evaluable functions can be characterized by the models of its loop completion. Thus, a weight constraint program with evaluable functions can be translated to an instance of the Constraint Satisfaction Problem (CSP) [39], whose solutions correspond exactly to the answer sets of the original program.

The above results provide a basis for extending FASP,[4] a prototypical implementation for normal logic programs with evaluable functions, to compute answer sets of weight constraint programs with evaluable functions. This approach possesses two main advantages. One is that the CSP facilities such as *global constraints* can be readily brought into the ASP language, since programs in this language are translated into CSP instances. In this sense, this approach can be seen as another attempt to integrate CSP with ASP (cf. [12, 19, 21, 33]). Another advantage of this approach is that the sizes of grounded programs sometimes can be substantially smaller than those of the standard ASP encodings.

We conducted experiments on five benchmarks to evaluate the computational effectiveness of our approach. First, we tested FASP on the magic square problem encoded by a weight constraint program with evaluable functions. We use a top ranked CSP solver from the 3rd CSP Solver Competition: *Mistral*, as a black box to FASP. The experimental results show that FASP substantially outperforms the popular state-of-the-art ASP solvers. The results may be contributed by the grounding sizes and the use of the global constraint "allDifferent". We also tested FASP with other four benchmarks: the traveling salesperson problem, the weighted N-queens problem, the weighted Latin square problem, and the weight-bounded dominating set problem. The first three above are obtained from the website of PBMODELS and the last one is adopted from the 2nd ASP competition. For these benchmarks, FASP is competitive with the current ASP solvers.

The paper is organized as follows. In the next section we recall the basic notations of weight constraint programs, for which we prove the splitting-set theorem and propose the notion of a loop formula. In Section 3 we add evaluable functions to weight constraint programs, define their semantics, and formulate loop formulas for these programs. In Section 4 we briefly describe the implementation of FASP, followed by a report of experimental results. Section 5 discusses related work, and Section 6 concludes the paper with final remarks.

2 Weight constraint programs

In this section, we first recall the basic notations of weight constraint programs (without functions) [35]. We then present two characterizations of these programs, one being the splitting-set theorem and the other the notions of loops and loop formulas. The former is an extension of *splitting a logic program* by Lifschitz and Turner [25] for normal logic programs, while the latter improves the result of Liu and Truszczyński [29] for weight constraint programs, which only allow monotone and convex weight constraints. The loop formulas given here will serve as the basis for the extension of adding functions into these programs in a later section.

[4] http://www.cse.ust.hk/fasp/

2.1 Preliminary definitions

We assume an underlying propositional language $\mathcal{L}^P$. A *rule element* is an atom (positive rule element) or an atom prefixed with *not* (negative rule element). A *weight constraint* is an expression of the form

$$l \leq \{c_1 : w_1, \ldots, c_n : w_n\} \leq u \tag{1}$$

where

- each of l and u is a real number or one of the symbols $+\infty$, $-\infty$; if l (resp. u) is $-\infty$ (resp. $+\infty$) then "$l \leq$" (resp. "$\leq u$") can be omitted.
- $c_1, \ldots, c_n$ are rule elements, and
- $w_1, \ldots, w_n$ are nonnegative integers, called *weights*.[5]

We denote the weight constraint (1) by $l \leq S \leq u$, where l or u may be omitted as defined above and $S = \{c_1 : w_1, \ldots, c_n : w_n\}$.

Let $C = (l \leq S \leq u)$ be a weight constraint. We define the notation, $\mathcal{A}toms^+(C) = \{a \mid (a : w_a) \in S\}$ and $\mathcal{A}toms(C) = \mathcal{A}toms^+(C) \cup \{b \mid (not\, b : w_b) \in S\}$. Let $\mathcal{K}$ be a set of weight constraints. By $\mathcal{A}toms(\mathcal{K})$, we mean the set $\bigcup_{C \in \mathcal{K}} \mathcal{A}toms(C)$ and by $\mathcal{A}toms^+(\mathcal{K})$ we mean the set $\bigcup_{C \in \mathcal{K}} \mathcal{A}toms^+(C)$. A *weight constraint program* (or simply *a program*) is a finite set of *weight constraint rules* (simply *weight rules* or *rules*) of the form

$$C_0 \leftarrow C_1, \ldots, C_n \tag{2}$$

where C_i $(0 \leq i \leq n)$ are weight constraints. Let r be a rule of the form (2). Then C_0 and $\{C_1, \ldots, C_n\}$ are called its *Head* and *Body*, respectively. Alternatively, for convenience we may write r in the form of $Head \leftarrow Body$. A weight rule is a *Horn rule* if its head is a positive rule element and every member of its body has the form $l \leq S$, where S does not mention any negative rule elements. We denote by $\mathcal{A}toms(P)$ the union of $\mathcal{A}toms(C)$ where C is a weight constraint occurring in P.

Let Z be a set of atoms, a an atom and C a weight constraint of the form (1). We say that Z *satisfies* a, written $Z \models a$, if $a \in Z$; Z *satisfies not* a if $a \notin Z$; and Z *satisfies* C, written $Z \models C$, if the sum of the weights w_j for all j such that Z satisfies c_j is not less than l and not greater than u; and Z *satisfies* a logic program P if, for every rule of the form (2) in P, Z satisfies C_0 whenever Z satisfies $C_1, \ldots, C_n$.

Given a weight constraint $C = l \leq S \leq u$ and a set of atoms Z, the *reduct* of C with respect to Z, denoted C^Z, is the weight constraint $l' \leq S'$, where

- S' is obtained from S by dropping all pairs $(not\, b : w)$, and
- l' is l minus the sum of the weights w for all pairs $(not\, b : w)$ in S such that $b \notin Z$.

Since the reduct of C no longer mentions u, we will simply write $(l \leq S)^Z$ instead of $(l \leq S \leq u)^Z$. The *reduct* of a weight rule

$$C_0 \leftarrow l_1 \leq S_1 \leq u_1, \ldots, l_n \leq S_n \leq u_n \tag{3}$$

[5] We consider only nonnegative weights in this paper, as negative weights can be replaced by negative rule elements [32, 40]. It is also known that negative weights may cause some unintuitive results [17].

with respect to Z is the set of rules

$$\{a \leftarrow (l_1 \leq S_1)^Z, \ldots, (l_n \leq S_n)^Z \mid a \in \mathcal{A}toms^+(C_0) \cap Z \text{ and } Z \models S_i \leq u_i \text{ for all } i\ (0 \leq i \leq n)\}$$

The *reduct* P^Z of a weight constraint program P with respect to Z is the union of the reducts of the rules of P with respect to Z. Clearly, P^Z consists of Horn rules only.

If a weight constraint program P consists of Horn rules then it has a unique minimal set S of atoms such that $S \models P$. The set is called the *deductive closure* of P and denoted by $cl(P)$. Given a weight constraint program P and a set of atoms S, we say that S is an *answer set* of P if and only if $S \models P$ and $cl(P^S) = S$. In particular, if C_0 in a rule of the form (2) is equivalent to $\bot$ then the rule is called a *constraint*. In this case, we usually omit C_0 in (2). With constraints, the definition of answer set above can be extended to: A set of atoms M is an answer set of a weight constraint program P if M is an answer set of P' and M satisfies the constraints in P, where P' is obtained from P by eliminating all constraints of P.

Example 1 Consider the weight constraint program $P = \{a \leftarrow 1 \leq \{a : 1, not\ a : 1\}\}$. Suppose $S_1 = \emptyset$ and $S_2 = \{a\}$. It is clear that $S_1 \not\models P$, so that S_1 is not an answer set of P. As an exercise, note that $P^{S_1} = \{a \leftarrow 0 \leq \{a : 1\}\}$ and $cl(P^{S_1}) = \{a\}$. Meanwhile, since $S_2 \models P$, $P^{S_2} = \{a \leftarrow 1 \leq \{a : 1\}\}$, and $cl(P^{S_2}) = \emptyset\ (\neq S_2)$, S_2 is not an answer set of P either. It follows that P has no answer set.[6]

2.2 The splitting-set theorem for weight constraint programs

The general idea of splitting is that a logic program may be partitioned into two layers, the "bottom" part and the "top" part, for the computation and characterization of answer sets. We generalize this idea from normal logic programs [25] to weight constraint programs. We define the notion of a splitting set and prove the splitting-set theorem. The result will be used later in the proof of Theorem 3, but it is apparently of independent interest.

Let M be a set of atoms and $S = \{c_1 : w_1, \ldots, c_n : w_n\}$, where c_i $(1 \leq i \leq n)$ are rule elements and w_i $(1 \leq i \leq n)$ are weights. We denote $\Sigma(S, M) = \sum_{(\alpha:w) \in S \text{ and } M \models \alpha} w$.

Let V and M be two sets of atoms such that $M \subseteq V$, and $C = l \leq S \leq u$ a weight constraint. We use $e_V(C, M)$ to denote the following weight constraint

$$l - \Sigma(\Gamma_1(S, V), M) \leq \Gamma_2(S, V) \leq u - \Sigma(\Gamma_1(S, V), M)$$

where $\Gamma_1(S, V) = \{c : w \mid (c : w) \in S \text{ and } c \text{ mentions an atom in } V\}$ and $\Gamma_2(S, V) = S \setminus \Gamma_1(S, V)$. In another words, $e_V(C, M)$ is obtained from C by

- removing every pair $(c : w)$ from S if c mentions an atom in V; and
- replacing l and u with $l - W$ and $u - W$ respectively, where $W = \sum_i w_i$ such that $(c_i : w_i) \in S$, $M \models c_i$ and c_i mentions an atom in V.

[6] Another way to look at the semantics of program P is via semantics-preserving transformations. In [32] P is transformed to a weight constraint program $P' = \{\bar{a} \leftarrow 0 \leq \{a : 1\} \leq 0;\ \ a \leftarrow 1 \leq \{a : 1, \bar{a} : 1\}\}$, and in [17] to a logic program with nested expressions $P'' = \{a \leftarrow a; not\ a\}$. In terms of normal logic programs, the corresponding program is $P''' = \{a \leftarrow a;\ \ a \leftarrow not\ a\}$.

For example, consider the weight constraint C

$$2 \leq \{a : 1, b : 2, not\, c : 3, not\, d : 4\} \leq 7.$$

Suppose $V = \{a, c, d\}$ and $M = \{a, d\}$. Then we have that $e_V(C, M)$ is the weight constraint, $-2 \leq \{b : 2\} \leq 3$, by seeing that $\Gamma_1(S, V) = \{a : 1, not\, c : 3, not\, d : 4\}$ and $\Sigma(\Gamma_1(S, V), M) = 4$, where $S = \{a : 1, b : 2, not\, c : 3, not\, d : 4\}$.

Let P be a weight constraint program. A set V of atoms is a *splitting set* for P if, for every rule r in P, $\mathcal{A}toms(Head(r)) \cap V \neq \emptyset$ implies $\mathcal{A}toms(r) \subseteq V$. The set of rules $r \in P$ such that $\mathcal{A}toms(r) \subseteq V$ is called the *bottom* of P relative to V and denoted by $b_V(P)$.

Let P be a weight constraint program P, V a splitting set of P and M a set of atoms. We denote by $e_V(P, M)$ the weight constraint program obtained from P by replacing each rule in P of the form (2) with

$$e_V(C_0, M) \leftarrow e_V(C_1, M), \ldots, e_V(C_n, M).$$

Theorem 1 *Let V be a splitting set of a weight constraint program P. A set of atoms M is an answer set of P if and only if $M = M_1 \cup M_2$ where M_1 and M_2 are answer sets of $b_V(P)$ and $e_V(P \setminus b_V(P), M_1)$, respectively.*

We note that a notion of modularization has been proposed for the class of programs composed of basic constraint rules [36], which can be viewed as a generalization of splitting set. A *basic constraint rule* is either a choice rule whose head is a choice constraint and whose body is a conjunction of literals, or a weight constraint rule whose head is an atom and whose body is a weight constraint free of upper bound. It is argued in [36] that these basic rules provide a reasonable coverage of weight constraint programs, since they are the internal representations of weight constraint programs in the SMODELS system. In other words, weight constraint rules can be translated to basic constraint rules. However, such a translation introduces new symbols. A module theorem, which is a generalization of the splitting set theorem for the same class of programs, has been proved [36]. Apparently, Theorem 1 above does not follow from this module theorem, due to the more general form of weight constraint programs in our case and a direct characterization without using extra symbols.

2.3 Completion and loop formulas

Following [29], to define the completion we first introduce an extension $\mathcal{L}^{Pwc}$ of the language $\mathcal{L}^P$. A *formula* in $\mathcal{L}^{Pwc}$ is an expression built from weight constraints by means of boolean connectives $\wedge$, $\vee$, $\supset$, and $\neg$. The notion of a model of a formula extends in a standard way to the class of formulas in $\mathcal{L}^{Pwc}$.

Given a weight constraint program P, the *completion* of P, denoted $COMP(P)$, is defined as follows:

1. For every rule $Head \leftarrow Body$ in P we include in $COMP(P)$ an $\mathcal{L}^{Pwc}$ formula

$$\left(\bigwedge Body\right) \supset Head \tag{4}$$

2. For every atom $a \in \mathcal{A}toms(P)$, we include in $COMP(P)$ an $\mathcal{L}^{Pwc}$ formula

$$a \supset \bigvee_{1 \le i \le n} \left(\bigwedge Body_i \right) \tag{5}$$

where $(Head_1 \leftarrow Body_1), \ldots, (Head_n \leftarrow Body_n)$ are all the rules of P such that $a \in \mathcal{A}toms^+(Head_i)$, for all $1 \le i \le n$. Please note that $\bigvee \emptyset = \bot$ ("false") and $\bigwedge \emptyset = \top$("*true*").

Let P be a weight constraint program. The *positive dependency graph* of P, written G_P, is the directed graph (V, E), where

- $V = \mathcal{A}toms(P)$,
- $(a, b) \in E$ if there a rule of the form (2) in P such that $a \in \mathcal{A}toms^+(C_0)$ and $b \in \mathcal{A}toms^+(C_i)$ for some i $(1 \le i \le n)$.

Let L be a nonempty subset of $\mathcal{A}toms(P)$. We say that L is a *loop* of P if there is a non-zero length cycle in G_P that goes through only and all the nodes in L. A loop L of P is *maximal* if there is no loop L' of P such that $L \subset L'$. A maximal loop L of P is *terminating* if there is no other maximal loop L' of P such that G_P has a path from some node in L to some one in L'.

Let $C = l \le S \le u$ be a weight constraint and L a set of atoms. The *restriction* of C to L, written $C_{|L}$, is the following $\mathcal{L}^{Pwc}$ formula

$$(l \le S') \wedge (S \le u) \tag{6}$$

where S' is obtained from S by removing every pair $(c_i : w_i)$ from S if $c_i \in L$. The intended meaning is that $l \le S$ is independent of L, similar to the notion of *external support* of [22]. The *loop formula* for a loop L of P, written $LF(L, P)$, is the following $\mathcal{L}^{Pwc}$ formula

$$\bigvee L \supset \left(\bigvee_{1 \le i \le n} \bigwedge_{C \in Body_i} C_{|L} \right)$$

where $(Head_1 \leftarrow Body_1), \ldots, (Head_n \leftarrow Body_n)$ are all the rules of P such that, for each $1 \le i \le n$, $\mathcal{A}toms^+(Head_i) \cap L \neq \emptyset$.

Example 2 Let $P = \{a \leftarrow 1 \le \{a : 1\} \le 1\}$ and $Z = \{a\}$. We have that $COMP(P)$ consists of

$$a \supset 1 \le \{a : 1\} \le 1 \quad \text{and} \quad 1 \le \{a : 1\} \le 1 \supset a.$$

Clearly, $Z \models COMP(P)$. But Z does not satisfy the loop formula for the loop $L = \{a\}$ of P, since $LF(L, P)$ is the formula

$$a \supset (1 \le \{\}) \wedge (\{a : 1\} \le 1)$$

which is not satisfied by Z due to $Z \not\models 1 \le \{\}$. The interested readers can check that $\emptyset$ is the unique answer set of P.

Example 3 (Continued from Example 1) The completion of P, $COMP(P)$, consists of the following formulas

$$a \supset (1 \le \{a : 1, not\, a : 1\}) \quad \text{and} \quad (1 \le \{a : 1, not\, a : 1\}) \supset a.$$

It is clear that $\emptyset \not\models COMP(P)$ but $\{a\} \models COMP(P)$. Note that P has exactly one loop $L = \{a\}$ whose loop formula $LF(L, P)$ is

$$a \supset (1 \leq \{not\, a : 1\})$$

which is obviously not satisfied by $\{a\}$.

Theorem 2 *Let P be a weight constraint program. A set M is an answer set of P if and only if M is a model of $COMP(P) \cup LF(P)$, where $LF(P)$ is the set of loop formulas of P.*

Since a literal l can be regarded as the weight constraint $1 \leq \{l = 1\} \leq 1$, normal logic programs can be seen as special cases of weight constraint programs. Note further that, given a set L of atoms and a literal l, if l is an atom in L then $l_{|L}$ is $1 \leq \{\} \leq 1$ which is equivalent to $\bot$, and l itself otherwise. In this way, the above definitions of loops and loop formulas can be regarded as a generalization of those for normal logic programs [27].

It is known that a weight constraint program can be transformed into a logic program with nested expressions [17]. The rules of logic programs with nested expressions have a normal form of

$$H \leftarrow B, F$$

where H is a disjunction of atoms, B a conjunction of atoms, and F a negative formula. Lee and Lifschitz proposed the notion of loop completion for such logic programs [23].

Liu and Truszczyński formulated the loop completion of logic programs with monotone and convex constraints and showed that, given such a program, the models of its loop completion are precisely its answer sets. Since a weight constraint program can be easily transformed into one that mentions only positive rule elements by introducing new propositional variables, a weight constraint program can be regarded as a convex constraint program that can be transformed into a monotone-constraint program. Accordingly, by translating the completion and loop formulas into pseudo-Boolean constraints and using solvers of pseudo-boolean constraints, they implemented a system, called PBMODELS, to compute the answer sets of weight constraint programs [29]. Later, You and Liu generalized these concepts for logic programs with arbitrary abstract constraint atoms [44].

For the approaches proposed in [23, 29], loop formulas can be obtained for (arbitrary) weight constraint programs, indirectly via some transformations, i.e., from weight constraint programs to logic programs with nested expressions for the former, and to monotone-convex-constraint programs for the latter. In the latter case, new atoms are introduced. Here in this paper, the notions of completion and loop formulas do not depend on such a transformation. Another definition of loop formulas for weight constraint programs is formulated independently by Liu and You [28], which does not introduce new atoms as well.

3 Weight constraint programs with evaluable functions

In this section, we introduce evaluable functions into weight constraint programs, simply called *weight programs*, give their syntax and semantics, and show how such evaluable functions in weight programs may be replaced with predicates. We then present the completion, loops and loop formulas for weight programs. Finally, we present a translation from weight programs to instances of the Constraint Satisfaction Problem.

3.1 Syntax and semantics

We assume a many-sorted first-order language $\mathcal{L}$. In such a language, every predicate has an arity that specifies the number of arguments the predicate has and the type (sort) of each argument, and similarly for constants and functions. Variables also have types associated with them, and when they are used in a formula, their types are normally clear from the context. A function of the type $\tau_1 \times ... \times \tau_n \rightarrow \tau$ is *evaluable* if the domains for the types $\tau_1, \ldots, \tau_n$ and τ are finite and fixed. The language $\mathcal{L}$ may have pre-interpreted symbols such as the standard arithmetic functions " $+$ ", " $-$ ", ..., as well as the standard arithmetic relations "$>$", "$\geq$", "$<$", "$\leq$", etc. Note that the symbol "$\leq$" has been used in weight constraints. When it is used as a standard arithmetic relation, it is in the form $t \leq t'$ where both t and t' are numeric terms. Informally, numeric terms contain numbers and functions of the form $f(\mathbf{t})$ where f is an evaluable function whose range is a set of numbers and $\mathbf{t}$ is a tuple of terms. Thus the role of "$\leq$" should be clear from its context. *Arithmetic atoms* are expressions of the form $t_1 \diamond t_2$ where t_1 and t_2 are numeric terms and $\diamond$ is a standard arithmetic relation.

By predicates we mean *proper predicates* that do not mention equality and the standard arithmetic relations. By an *atom* we mean an atomic formula that mentions a proper predicate, an *equality atom* is a formula of the form $t = t'$, where t and t' are terms of $\mathcal{L}$. Unless stated otherwise, by functions we mean *proper functions* (whose arities are greater than 0). *Extended rule elements* are defined to be atoms, arithmetic atoms and equality atoms (positive rule elements), and their negations (negative rule elements). For convenience, we write inequality $t \neq t'$ for *not* $t = t'$.

An *extended weight constraint* is an expression of the form (1), that is,

$$l \leq \{c_1 : w_1, \ldots, c_n : w_n\} \leq u$$

in which each c_i is an extended rule element where $1 \leq i \leq n$. A *weight rule* (or simply *a rule*) is an expression of the form (2) in which each C_i $(0 \leq i \leq n)$ is an extended weight constraint. A *(weight) program* P is a finite set of weight rules together with a set of *type definitions*, one for each type τ used in the rules of P. A type definition takes the form

$$\tau : D \tag{7}$$

where D is a finite nonempty set of objects, called the *domain* of τ. The notations about sets of atoms labeled by the symbols $\mathcal{A}toms^+(.)$ and $\mathcal{A}toms(.)$ are extended to the case of weight programs (note that they denote sets of non-equality atoms and non-arithmetic atoms).

Informally, a type definition defines a domain for a type (sort). This is like in a first-order structure for a many-sorted language, there is a domain for each type. Here we require that if a constant c of type τ occurs in the rules of P, then the domain D of τ as specified in the type definitions of P must contain c (but is called an object in D).

Let P be a weight program. An atom $p(c_1, \ldots, c_k)$ is said to *reside in* P if p is a predicate of type $\tau_1 \times \ldots \times \tau_k$ in P, and $c_i \in D_i$, where D_i is the domain of the type τ_i, for each i. We denote by $\mathcal{A}t(P)$ the set of atoms residing in P.

Borrowing the familiar notation of *conditional literals* [40], the traveling salesperson problem and the weighted N-queen problems can be encoded as follows (as ":" is already used as part of a weight rule, below we will use "|" instead of ":" for conditional literals).

Example 4 (The TSP problem (TSP)) Recall that the traveling salesperson problem (also called the weighted Hamiltonian circuit problem) is to find a path in a graph such that every vertex of the graph occurs exactly once in the path. In addition, the sum of the weights on the arcs in the path must be no more than a bound. The problem can be formalized by the following weight rules:

$$\leftarrow not\, reached(x), \qquad reached(hc(x)) \leftarrow initial(x),$$

$$\leftarrow not\, arc(x, hc(x)), \qquad reached(hc(x)) \leftarrow reached(x),$$

$$\{hc(x) = y : wt(arc(x, y)) \mid vertex(x; y)\} \leq weight.$$

Here, *reached* and *initial* are unary predicates over the domain *vertex*, *arc* is a binary predicate of the type $vertex \times vertex$, *hc* is a unary function of the type $vertex \rightarrow vertex$, $wt(arc(x, y))$ is the predefined weight of $arc(x, y)$, and *weight* is a given upper bound of a solution. An instance of the problem is specified by a domain for *vertex*, a set of facts for $arc(x, y)$ and their weights (given by $wt(arc(x, y))$), a fact for $initial(x)$, and an upper bound *weight*. Apparently, since *hc* is a function and we start with exactly one vertex, there is no need to constrain that, for each vertex, there is no more than one incoming arc nor more than one outgoing arc (which we would otherwise have to specify if represented by a relation). This program is more or less a direct "functionalization" of Niemelä's encoding of the weighted version of Hamiltonian circuit problem [34], and is also similar to Cabalar's encoding of the problem in his functional action language.[7]

Example 5 (The weighted N-queens problem (WNQ)) The weighted N-queens problem is a variant of the N-queens problem by assigning a weight to each cell and requiring the sum of weights where queens were placed is no more than a bound. This problem can be formalized by the following weight rules:

$$\leftarrow q(x) = q(y), x \neq y,$$

$$\leftarrow |q(x) - q(y)| = |x - y|, x \neq y,$$

$$\{q(x) = y : wt(position(x, y)) \mid pos(x; y)\} \leq weight.$$

[7] http://www.dc.fi.udc.es/~cabalar/fal/

Here, q is a function of the type $pos \rightarrow pos$, $q(i)$ is the column where the ith-queen (the one in row i) is to be placed, *position* is of the type $pos \times pos$, $wt(position(x, y))$ is the predefined weight of $position(x, y)$, and *weight* is the upper bound of a solution. The expression $x \neq y$ in a rule stands for $not\, x = y$. There are actually two types here: *pos*, whose domain is $1..N$, and *int*, whose domain is $-N..N$, for the N-queens problem. The pre-interpreted function "$-$" is of the type $pos \times pos \rightarrow int$ and "$|.|$" of $int \rightarrow int$. These two functions have their standard meanings. The first two rules in the program consist of the encoding of the queens problem that are essentially the same as Cabalar's encoding of 8-queens problem in his functional action language. An instance of the problem is specified by a domain of *pos*, the facts $position(x, y)$ and their weights (given by $wt(position(x, y))$), and an upper bound *weight*.

We now proceed to define an answer set semantics for weight programs.

Let P be a weight program. The *grounding* of P consists of type definitions in P and the rules that are obtained by (1) replacing variables in the rules of P with elements in their respective domains, and then (2) replacing each weight constraint $l \leq S \leq u$ with $l' \leq S' \leq u'$, where

(i) S' is obtained from S by removing all of the pairs $([not]t = t' : w)$ where t and t' are two constants, and

(ii) $l' = l - T$, $u' = u - T$ where $T = \sum_{(c=c:w)\in S} w + \sum_{(c\neq c':w')\in S} w'$, and c, c' are two distinct constants.

Generally speaking, the grounding of a program may cause an exponential blow-up [41]. However, if the maximal number of distinct variables in a rule is fixed to some constant d then grounding results in a polynomial increase in size. In other words, grounding such a rule results in at most $O(n^d)$ number of instantiated rules where n is the number of domain elements.

Please note that a ground rule may have symbols not in the original language $\mathcal{L}$. We let $\mathcal{L}_P$ be the language that extends $\mathcal{L}$ by introducing a new constant for each element in the domain of a type (the introduced new constants must be distinct from each other and from all existing constants in $\mathcal{L}$). Each of these new constants will have the same type as the type of its corresponding element. In this case, the fully instantiated rules will be in the language $\mathcal{L}_P$. In the following, unless otherwise stated, we shall equate a weight program with its grounding in the extended language $\mathcal{L}_P$. Note that the language $\mathcal{L}_P$ is related to the given program P.

Given a weight program P, the intended domains of interpretations for P are exactly those specified in the type definitions of P. Thus, an *interpretation* I of P is a first-order structure that defines a mapping such that all of the following conditions are satisfied

- The domains of I are those specified in the type definitions of P.
- A constant is mapped to itself.
- If R is a relation of type $\tau_1 \times \ldots \times \tau_n$ and the type definitions $\tau_i : D_i$, $1 \leq i \leq n$, are in P, then $R^I \subseteq D_1 \times \ldots \times D_n$. A standard arithmetic relation should follow its standard interpretation.
- If f is a function of type $\tau_1 \times \ldots \times \tau_n \rightarrow \tau_{n+1}$, $n \geq 1$, and the type definitions $\tau_i : D_i$,$1 \leq i \leq n+1$, are in P, then f^I is a function from $D_1 \times \ldots \times D_n$ to D_{n+1}. A pre-interpreted function should follow its standard interpretation.

By I^a, we denote the set of atoms that are true under I, i.e., $\{p(\mathbf{c}) \mid I \models p(\mathbf{c})\}$, where $\mathbf{c}$ is a tuple of constants matching the arity of relation symbol p. Note that an interpretation is always associated with a weight program. The *valuation* of a term t under an interpretation I, denoted by t^I, is a constant defined as:

- $t^I = c$ if $t = c$;
- $t^I = c'$ if $t = f(\mathbf{s})$ and $f^I(\mathbf{s}^I) = c'$, where $\mathbf{s}$ is a tuple of terms matching the type of f, say $\mathbf{s} = (t_1, \ldots, t_n)$, and $\mathbf{s}^I$ stands for $\left(t_1^I, \ldots, t_n^I\right)$.

Let I be an interpretation. We say that I *satisfies* an atom $p(\mathbf{t})$ if $\mathbf{t}^I \in p^I$, i.e., $p(\mathbf{t}^I) \in I^a$. The interpretation I *satisfies* an equality atom $t_1 = t_2$ if $t_1^I = t_2^I$. The *satisfaction* (and *model*), written by $\models$, for literals, extended weight constraints and weight programs can be defined accordingly in a straightforward way.

Let P be a weight program, I an interpretation for P, and $l \leq S \leq u$ an extended weight constraint occurring in P. The *reduct* of $l \leq S \leq u$ with respect to I, written $[l \leq S]^I$ (again, since the reduct does not mention u, we will omit it), is the weight constraint $l^{(I,S)} \leq S^I$, where

- S^I is obtained from S by
  - replacing each functional term $f(\mathbf{t})$ in a rule by d if $f^I(\mathbf{t}^I) = d$;
  - removing all pairs $(c : w)$ whenever c is an equality atom or an arithmetic atom or a negative extended rule element.
- $l^{(I,S)} = l - \sum_{(c:w_c) \in S \text{ and } I \models c} w_c$, where c is a negative rule element or an equality atom or an arithmetic atom. In particular, equality is interpreted as an identity relation, i.e., for any interpretation I, $I \models (c = c)$, $I \not\models (c \neq c)$, $I \models (c \neq c')$, and $I \not\models (c = c')$ where c and c' are two distinct constants.

The *reduct* of a rule of the form

$$l_0 \leq S_0 \leq u_0 \leftarrow l_1 \leq S_1 \leq u_1, \ldots, l_n \leq S_n \leq u_n$$

in P, with respect to an interpretation I, is the set of rules

$$\{p(\mathbf{c}) \leftarrow [l_1 \leq S_1]^I, \ldots, [l_n \leq S_n]^I \mid (p(\mathbf{c}) : w) \in S_0^I, I \models p(\mathbf{c}) \text{ and } I \models S_i \leq u_i \text{ for all } i\ (0 \leq i \leq n)\}.$$

The *reduct* of P under the interpretation I, written P^I, is the union of the reducts of the rules in P with respect to I. Clearly, P^I is a weight constraint program consisting of Horn rules only. Thus the least model $cl(P^I)$ of P^I exists. We call the interpretation I for P an *answer set* of P if $I \models P$ and $I^a = cl(P^I)$.

Example 6 Consider the following weight program P:

$$f : \tau \to \tau, \qquad p : \tau, \qquad \tau : \{0, 1\},$$

$$1 \leq \{f(0) \neq f(1) : 1, p(0) : 2\} \leq 2 \leftarrow 1 \leq \{not\ p(f(1)) : 2, p(f(0)) : 1\} \leq 2.$$

Let us consider the interpretation I for P such that $f^I(0) = f^I(1) = 0$, $I \models p(0)$ and $I \not\models p(1)$. The reduct P^I consists of a single rule: $p(0) \leftarrow 1 \leq \{p(0) : 1\}$. Since $\emptyset$ is the unique answer set of P^I which is different from $I^a = \{p(0)\}$, it follows that I is not an answer set of P.

Note that any term t will be evaluated to a constant under an interpretation. Thus if a weight program P does not mention predicate symbols then an interpretation I for P is an answer set of P if and only if I satisfies every weight rule of P.

Example 7 Consider the weight program P consisting of

$$\begin{aligned} f : \tau \to \tau, \qquad \tau : \{0, 1\}, \\ f(0) > f(1) \leftarrow f(0) \neq f(1), \\ \leftarrow f(0) = f(1). \end{aligned}$$

It is easy to see that P has a unique answer set I such that $f^I(0) = 1$ and $f^I(1) = 0$. In fact, the program P is a normal one as defined in [26].

3.2 Eliminating functions

As elluded to in the introduction, functions are not necessary theoretically speaking. They can be eliminated by using relations. We now make this precise.

Let P be a weight program. For each function $f : \tau_1 \times \cdots \times \tau_n \to \tau$ in P, we introduce two corresponding relations f_r and $\overline{f}_r$. They both have the type $\tau_1 \times \cdots \times \tau_n \times \tau$, and informally speaking $f_r(x_1, ..., x_n, y)$ stands for $f(x_1, ..., x_n) = y$ and $\overline{f}_r(x_1, ..., x_n, y)$ for $f(x_1, ..., x_n) \neq y$. Now let $\mathbb{F}(P)$ be the union of the rules obtained by grounding the following rules for each function f in P using the domains in the type definitions of P:

$$\begin{aligned} &\leftarrow f_r(x_1, \ldots, x_n, y_1),\ f_r(x_1, \ldots, x_n, y_2),\ y_1 \neq y_2, \\ &f_r(x_1, \ldots, x_n, y) \leftarrow \mathit{not}\ \overline{f}_r(x_1, \ldots, x_n, y), \\ &\overline{f}_r(x_1, \ldots, x_n, y) \leftarrow f_r(x_1, \ldots, x_n, z),\ y \neq z. \end{aligned}$$

Let $\mathbb{R}(P)$ be the set of rules obtained from the rules in P by the following transformation:

- Repeatedly replace each functional term $f(u_1, \ldots, u_n)$, where each u_i is a simple term in that it does not mention a function symbol, by a new variable x and add $f_r(u_1, \ldots, u_n, x)$ to the body of the rule where the term appears.
- Ground all the variables introduced in the previous step.

For example, suppose the domain for the range of the function f is $\{c_1, \ldots, c_m\}$. The following rule in a weight program P:

$$l \leq \{p(a) : w_1,\ f(a) = c_1 : w_2\} \leq u \leftarrow 1 \leq \{p(b) : w_2\}$$

will be transformed into the following rules in $\mathbb{R}(P)$

$$\begin{aligned} &l - w_2 \leq \{p(a) : w_1\} \leq u - w_2 \leftarrow 1 \leq \{p(b) : w_2\},\ f_r(a, c_1), \\ &l \leq \{p(a) : w_1\} \leq u \leftarrow 1 \leq \{p(b) : w_2\},\ f_r(a, c_i)\ (2 \leq i \leq m). \end{aligned}$$

Clearly $\mathbb{F}(P) \cup \mathbb{R}(P)$ is a weight constraint program which is equivalent to P:

Theorem 3 *Let P be a weight program. An interpretation I is an answer set of P iff $\mathbb{R}(I)$ is an answer set of $\mathbb{F}(P) \cup \mathbb{R}(P)$, where*

$$\mathbb{R}(I) = I^a \cup \{ f_r(\mathbf{c}, a) \mid f_r(\mathbf{c}, a) \in \mathcal{A}\text{toms}(\mathbb{F}(P)) \text{ and } f^I(\mathbf{c}) = a\}$$
$$\cup \{\overline{f}_r(\mathbf{c}, a) \mid \overline{f}_r(\mathbf{c}, a) \in \mathcal{A}\text{toms}(\mathbb{F}(P)) \text{ and } f^I(\mathbf{c}) \neq a\}.$$

It is evident that, given a weight program P, if the arities of both predicates and functions are bounded by a constant then the size of of $\mathbb{F}(P) \cup \mathbb{R}(P)$ is polynomial in the size of P. Recall that the problem of deciding if a set of a ground weight constraint rules has an answer set if NP-complete (cf. Theorem 3.2 of [40]). Readers may referr to [37] for the notations on complexity theory. Thus the following proposition follows.

Proposition 1 *If the arities of predicates and functions are bounded by a constant, then the problem of deciding whether a weight program has an answer set is NP-complete.*

Proof Membership: If the arities of both predicates and functions of a weight program P are bounded by some constants then $\mathcal{A}t(P)$ is in polynomial size of P and the set of functional terms in which no functional term appears as their arguments is also in polynomial size of P. It follows that we can guess an interpretation I of P and check whether I is an answer set of P in polynomial time. Thus the problem is in NP.

Hardness: Since the class of ground weight constraint programs is a special case of ground weight programs, and the problem of deciding if a ground weight constraint program has an answer set is NP-hard (Theorem 3.2 of [40]), it implies that the same problem for a ground weight program is NP-hard. □

The following corollary directly follows from the above proposition.

Corollary 1 *Give a weight program P in which the arities of both predicates and functions are fixed by some constant, and α an atom in $\mathcal{A}t(P)$. Then we have that*

- *the problem if P has an answer set I such that $\alpha \in I^a$ is NP-complete, and*
- *the problem if $\alpha \in I^a$ for any answer set I of P is co-NP-complete.*

3.3 Completion and loop formulas

Similar to Section 2.3, to accommodate logic formulas that may contain extended weight constraints, we extend the language $\mathcal{L}$ to $\mathcal{L}^{wc}$, in which an extended weight constraint is regarded as an atomic formula of $\mathcal{L}^{wc}$, and formulas of $\mathcal{L}^{wc}$ are constructed in the normal way.

Let P be a weight program. The *completion* of P, written $COMP_f(P)$, consists of the following formulas of the language $\mathcal{L}^{wc}$:

1. For every rule $(Head \leftarrow Body)$ in P, we include the following formula of $\mathcal{L}^{wc}$ in $COMP_f(P)$

$$\left(\bigwedge Body\right) \supset Head. \tag{8}$$

2. For each atom $p(\mathbf{c}) \in \mathcal{A}t(P)$, we include the following formula of $\mathcal{L}^{wc}$ in $COMP_f(P)$

$$p(\mathbf{c}) \supset \bigvee_{1 \le i \le n} \left[\bigwedge Body_i \wedge \left(\bigvee_{p(\mathbf{t}) \in \mathcal{A}toms^+(Head_i)} \mathbf{t} = \mathbf{c} \right) \right] \tag{9}$$

where $Head_i \leftarrow Body_i (1 \le i \le n)$ are the rules of P, $\mathbf{t} = \mathbf{c}$ stands for the formula $t_1 = c_1 \wedge \ldots \wedge t_n = c_n$ whenever $\mathbf{t} = (t_1, \ldots, t_n)$ and $\mathbf{c} = (c_1, \ldots, c_n)$.

Let P be a weight program. The *positive dependency graph* of P, written G_P, is the directed graph (V, E), where

- $V = \mathcal{A}t(P)$ and
- $(p(\mathbf{c}), q(\mathbf{d})) \in E$ iff P has a weight rule of the form (2) such that, $p(\mathbf{t}) \in \mathcal{A}toms^+(C_0)$, $q(\mathbf{s}) \in \mathcal{A}toms^+(C_i)$, for some i $(1 \le i \le n)$, and there exists an interpretation I for P with $\mathbf{t}^I = \mathbf{c}$ and $\mathbf{s}^I = \mathbf{d}$.

Now, loops, maximal loops, and terminating loops are similarly defined. Formally, a non-empty subset L of $\mathcal{A}t(P)$ is a *loop* of P if there is a non-zero length cycle in G_P that goes through only and all the nodes in L. A loop L of P is *maximal* if there is no loop L' of P such that $L \subset L'$. A maximal loop L of P is *terminating* if there is no other maximal loop L' of P such that G_P has a path from some node in L to some one in L'.

In the following, to define loop formulas, we further extend the language $\mathcal{L}^{wc}$ to $\mathcal{L}_o^{wc}$ such that, for any predicate p and a non-empty subset L of $\mathcal{A}t(P)$, $\mathcal{L}_o^{wc}$ contains a predicate p_L with the same arity as that of p. The interpretations for P can be similarly extended. Let $p(\mathbf{t})$ be an atom and $L \subseteq \mathcal{A}t(P)$. The *L-irrelevant formula* of $p(\mathbf{t})$, written $IL(p(\mathbf{t}), L)$, is defined as

$$p_L(\mathbf{t}) \equiv \left(p(\mathbf{t}) \wedge \bigwedge_{p(\mathbf{c}) \in L} \neg(\mathbf{c} = \mathbf{t}) \right) \tag{10}$$

Intuitively, given an interpretation I of $\mathcal{L}^{wc}$, we can talk about the truth of $p_L(\mathbf{t})$ indirectly - "$p_L(\mathbf{t})$ is true under I" if and only if $I \models p(\mathbf{t})$ and $p(\mathbf{t}^I) \notin L$. Formally, the truth of $p_L(\mathbf{t})$ has to be treated in the language of $\mathcal{L}_o^{wc}$. In what follows, given an interpretation I of $\mathcal{L}^{wc}$, by $o(I)$ we mean the interpretation of $\mathcal{L}_o^{wc}$ which is the extension of I satisfying (10). In other words, $o(I)$ is the same interpretation as I for the signatures of $\mathcal{L}^{wc}$, and it satisfies (10) for each predicate p_L. It is clear that any interpretation of $\mathcal{L}_o^{wc}$ that satisfies (10) corresponds to a unique interpretation of $\mathcal{L}^{wc}$.

Let P be a weight program, $L \subseteq \mathcal{A}t(P)$ and $C = l \le S \le u$ be an extended weight constraint. The *restriction* of C to L, written $C_{\|L}$, is the following formula of $\mathcal{L}_o^{wc}$:

$$(l \le S') \wedge (S \le u) \wedge \left(\bigwedge_{p(\mathbf{t}) \in \mathcal{A}toms^+(C)} IL(p(\mathbf{t}), L) \right) \tag{11}$$

where S' is obtained from S by replacing every pair $(p(\mathbf{t}) : w)$ in S with $(p_L(\mathbf{t}) : w)$, and p is a proper predicate. In particular, if C mentions only literals that contain no function symbols, then $C_{\|L}$ coincides with $C_{|L}$ in the language $\mathcal{L}^{wc}$ since $p_L(\mathbf{t}) \equiv p(\mathbf{t})$ if $p(\mathbf{t}) \notin L$, and $p_L(\mathbf{t}) \equiv \bot$ otherwise.

Let L be a loop of a weight program P. The *loop formula* of L with respect to P, written $LF_f(L, P)$, is the following $\mathcal{L}_o^{wc}$ formula:

$$\bigvee L \supset \bigvee_{1 \le i \le n} \left[\left(\bigvee_{\substack{p(\mathbf{c}) \in L \\ p(\mathbf{t}) \in \mathcal{A}toms^+(Head_i)}} \mathbf{t} = \mathbf{c} \right) \wedge \left(\bigwedge_{C \in Body_i} C_{\| L} \right) \right] \tag{12}$$

where $Head_i \leftarrow Body_i$ $(1 \le i \le n)$ are all the rules of P. One should note that, if L and $\mathcal{A}toms^+(Head_i)$ share no common predicate then it is unnecessary to compute $\bigwedge_{C \in Body_i} C_{\| L}$ since

$$\left(\bigvee_{\substack{p(\mathbf{c}) \in L \\ p(\mathbf{t}) \in \mathcal{A}toms^+(Head_i)}} \mathbf{t} = \mathbf{c} \right) \equiv \bot.$$

Example 8 (Continued from Example 6) The completion of P, $COMP_f(P)$, consists of the following formulas:

$(1 \le \{not\ p(f(1)) : 2, p(f(0)) : 1\} \le 2) \supset (1 \le \{f(0) \ne f(1) : 1, p(0) : 2\} \le 2)$,
$p(0) \supset (1 \le \{not\ p(f(1)) : 2, p(f(0)) : 1\} \le 2) \wedge (0 = 0)$,
$p(1) \supset \bot$.

It is easy to see that the interpretation I, i.e., $f^I(0) = f^I(1) = 0$ and $I \models p(0)$ but $I \not\models p(1)$, does satisfy $COMP_f(P)$. Note further that $L = \{p(0)\}$ is a unique loop of P. The loop formula $LF_f(L, P)$ is the following formula

$$p(0) \supset \left[\begin{array}{l} (0 = 0) \wedge (1 \le \{not\ p(f(1)) : 2, p_L(f(0)) : 1\}) \wedge \\ (\{not\ p(f(1)) : 2, p(f(0)) : 1\} \le 2) \wedge IL(p(f(0)), L) \end{array} \right]$$

where $IL(p(f(0)), L)$ is the formula

$$p_L(f(0)) \equiv p(f(0)) \wedge (0 \ne f(0)).$$

It is not difficult to verify that $o(I)$ does not satisfy the loop formula $LF_f(L, P)$, since $p_L(f(0)) \equiv \bot$.

The below theorem formally shows that answer sets of a weight program can be characterized by the models of its loop completion.

Theorem 4 *Let P be a weight program. An interpretation I for P is an answer set of P if and only if $o(I)$ is a model of $COMP_f(P) \cup LF_f(P)$, where $LF_f(P)$ is the set of loop formulas of P and $o(I)$ is the extension of I satisfying* (10).

3.4 Translation to CSP

Formally, a CSP is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X}$ is a set of variables, $\mathcal{D}$ a set of domains, one for each variable in $\mathcal{X}$, and $\mathcal{C}$ a set of constraints about the variables in $\mathcal{X}$. A solution to a CSP is an assignment that maps each variable in $\mathcal{X}$ to an element in its domain such that under the assignment all constraints in $\mathcal{C}$ are satisfied. Abstractly, a constraint can be thought of as a pair $(\mathbf{x}, S)$, where $\mathbf{x}$ is a tuple of variables, and S a set of tuples of values in the domains of the variables in $\mathbf{x}$. Thus an assignment

satisfies a constraint $(\mathbf{x}, S)$ if under the assignment, the tuple of values taken by the variables in $\mathbf{x}$ is in S [39].

In order to translate a weight program into a CSP in terms of its completion and loop formulas, we need to assume a certain "normal form" for functional terms. Let P be a weight program. We say that P is *free of functions in arguments* if all terms that can be evaluated independently of interpretations have been replaced by constants in $\mathcal{L}_P$, and none of the predicates or functions that are not pre-interpreted have a functional term in their arguments. Given a weight program P, we can translate it into one that is free of functions in arguments using the following procedure:

(1) evaluate all terms that mention only constants and pre-interpreted functions to constants;
(2) for each rule in P, repeatedly replace every occurrence of a term $f(u_1, \ldots, u_n)$ in any argument of a predicate or a function that is not pre-interpreted in the rule by a new fresh variable v of the same type as the range of f, and add $f(u_1, \ldots, u_n) = v$ to the body of the rule, where u_i is a simple term; and
(3) ground the rules obtained in the above step.

It is evident that the original program and the transformed one are equivalent in the sense that they have the same answer sets. In the following, without loss of generality, we assume that weight programs are free of functions in arguments.

Accordingly, using completion and loop formulas, we can translate a weight program to a CSP, denoted $\mathcal{R}(P) = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, as follows. The set $\mathcal{X}$ of variables and their domains are

– for each atom in $\mathcal{A}t(P)$, there is a variable for it whose domain is $\{0, 1\}$, and
– for each functional term $f(u_1, \ldots, u_n)$ occurring in P such that f is not pre-interpreted, there is a variable for it whose domain is the range of f.

The set $\mathcal{C}$ of constraints is: for each formula ϕ in $COMP_f(P) \cup LF_f(P)$, there is a constraint $c(\phi) = \langle S, R \rangle$ in $\mathcal{C}$, where R is the constraint obtained from ϕ by replacing atoms and functional terms in it by their corresponding variables, and S is the set of variables occurring in R. In particular, for an extended weight constraint C of the form (1), we transform C to the following constraint:

$$l \leq \left(\sum_{i=1}^{n} if(\overline{c_i}, w_i, 0) \right) \leq u$$

where $\overline{c}_i = c$ if c is an atom or an equality atom or an arithmetic atom, and $\neg c$ if $c_i = not\, c$, $if(\psi, t_1, t_2)$ is an *if-then-else term* which means that the value of $if(\psi, t, t')$ is t if ψ is true and t' otherwise.

Under this formulation, the answer sets of a weight program P correspond to the solutions to its corresponding CSP $\mathcal{R}(P)$ under the following mapping: let I be an interpretation of P, the variable assignment corresponding to I, written $v(I)$, is defined as follows:

– if $x \in \mathcal{X}$ corresponds to an atom p, then $v(I)$ assigns x the value 1 if and only if p is true in I; and
– if $x \in \mathcal{X}$ corresponds to the term $f(u_1, \ldots, u_n)$, then $v(I)$ assigns x the value u if and only if $f^I(u_1, \ldots, u_n) = u$.

Similarly, given a variable assignment of $\mathcal{R}(P)$, a corresponding interpretation of P can be easily computed.

Theorem 5 *Let P be a weight program that is free of functions in arguments and I an interpretation of P. Then I is an answer set of P if and only if $v(I)$ is a solution of $\mathcal{R}(P)$.*

4 Implementation and experimentation

Due to Theorem 5, the answer sets of a weight program can be computed using a CSP search engine. In this section, we briefly introduce the implementation of FASP for this purpose and report some experimental results.

4.1 Implementation

As we know, the current CSP encoding formalism under the name of XCSP 2.1,[8] which is designed for the third CSP solvers' competition, allows global constraints, such as *weightSum*, *allDifferent*, *among*, *atleast*, *atmost*, *cumulative*, etc. Our approach of using CSP solvers to compute answer sets allows us to make use of these facilities. Therefore, we can incorporate global constraints into the ASP language for weight constraint programs with functions, by introducing appropriate notations. For example, we use the following notation

$$f : \tau_1 \times \ldots \times \tau_n \rightarrow \tau[\mathit{allDifferent}]$$

to express not only the type of the function symbol f but also the requirement that the function $f(\mathbf{t})$ should produce different values for different $\mathbf{t}$'s. In the current version of FASP, invocation to the CSP global constraint *allDifferent* has been implemented.

Given our translation above from weight constraint programs with functions to CSPs, we can compute the answer sets of such logic programs using an algorithm that is similar to the one used by ASSAT [27], except that we now use a CSP solver instead of a SAT solver.

First of all, notice that our translation from logic programs to CSPs actually consists of two steps: it first transforms a logic program to a set of quantifier-free sentences in the form of completions and loop formulas, and then from these sentences to a CSP. The second part is actually quite general in that it works for any quantifier-free sentences that do not have any functional terms in the arguments of predicates and functions, provided the domain of each type is given and finite.

As noticed in [27], a logic program may have an exponential number of loops and thus an exponential number of loop formulas. It is generally infeasible to add all of the loop formulas of a weight program P at the beginning and call a CSP solver with $COMP_f(P) \cup LF_f(P)$. The proposition below shows that if a model of $COMP_f(P)$ is not an answer set of P then there must be a loop formula for some terminating loop that is not satisfied by the model. Note that terminating loops are maximal ones

[8] www.cril.univ-artois.fr/CPAI08/XCSP2_1.pdf

and it is known that finding maximal loops (strongly connected components) of a given graph is tractable [42]. This motivates us to incrementally add loop formulas of some maximal loops to $COMP_f(P)$ when calling a CSP solver with $COMP_f(P)$. The process is very similar to that of ASSAT [27]. Though in the worst case, it is generally believed that an exponential number of loop formulas are theoretically necessary [24], many practical benchmarks involve only a small number of maximal loops or no loops at all.

Proposition 2 *Let P be a weight program, I a set of atoms such that $I \models COMP_f(P)$. If I is not an answer set of P then $G_P[I^-]$ has at least one terminating loop and $I \not\models LF_f(L, P)$ for any terminating loop L of $G_P[I^-]$ where $I^- = I^a \setminus cl(P^I)$ and $G_P[I^-]$ is the induced subgraph of G_P on I^-.*

Algorithm 1 FASP(X, P) - X stands for a CSP solver

Input: A weight program P
Output: An answer set of P if it has one, and report no otherwise.
Begin

(1) $\Sigma \leftarrow COMP_f(P)$.
(2) $\mathcal{R}(\Sigma) \leftarrow$ convert Σ to the format of X.
(3) Find a solution S of $\mathcal{R}(\Sigma)$ by X.
(4) If no solution, **return** no answer set.
(5) Map S to an interpretation I of P.
(6) Compute $M^- = I^a \setminus cl(P^I)$, where $cl(P^I)$ is the least model of P^I.
(7) If $M^- = \emptyset$, **return** I as an answer set.
(8) Compute all the maximal loops under M^-, add their loop formulas to Σ, and **goto** step (2).

End

Proposition 3 *Algorithm* 1 *is sound and complete if X is a complete CSP solver, i.e., given a weight program P, P has an answer set I if and only if* **FASP**(X, P) *returns I as an answer set of P.*

We can modify the algorithm **FASP**(X, P) slightly for the purpose of computing all answer sets by resorting to a complete CSP solver X in two ways. One is to compute all models of $COMP_f(P)$ in the first step, and then check if each model of $COMP_f(P)$ is an answer set of P. The other is to call the algorithm iteratively: once we get an answer set I of P, we add a constraint into P to prevent I from being a model of $COMP_f(P)$ gain. In the case that P has an exponential number of answer sets, the two approaches will exponentially blow up in space. We note that an algorithm of answer set enumeration running in polynomial space was proposed and implemented in CLASP [18].[9]

[9] It is a nontrivial extension to adopt similar techniques for CLASP, since we will likely have to modify the structure of a CSP solver, and as such, treat it as a white box, instead of a black box.

4.2 Experimental results

In what follows we report our experimental results for the benchmarks of magic square program, TSP problem, weighted N-queens problem, weighted latin square problem, and weight-bounded dominating set problem. The ASP encodings of first four problems are obtained from the website of PBMODELS while the last one is from the website of the second ASP competition.[10] For further details regarding these benchmarks we refer the reader to [29]. All the experiments were done on a Dell PowerEdge 50 server with a quad-core Intel Xeon E5506 (2.13GHz) CPUs, 2GB RAM running Linux with kernel 2.6. We compare the time consumption in seconds with a variety of the state-of-the-art ASP solvers: SMODELS 2.34, CMODELS 3.79 (with *MiniSat 2.0 beta*), CLASP 1.3.5 and PBMODELS 0.2 (with *satzoo* 1.02). The grounder for these ASP solvers is *lparse* 1.1.1. We tested FASP with the CSP solver *Mistral* 1.331. It should be noted that our current prototypical implementation accepts ground weight programs only. Thus the instances being tested are grounded by ourselves.

4.2.1 The magic square problem

We first tested the magic N-square problem. The problem is to construct an $N \times N$ array using each integer in $\{1, \ldots, N^2\}$ as an entry in the array exactly once in such a way that entries in each row, each column, and either of two main diagonals sum up to $N(N^2+1)/2$. The weight program for the problem contains the rules:

$$w \leq \{square(x, y) \mid num(y)\} \leq w \leftarrow num(x), \tag{13}$$

$$w \leq \{square(x, y) \mid num(x)\} \leq w \leftarrow num(y), \tag{14}$$

$$w \leq \{square(x, x) \mid num(x)\} \leq w \leftarrow, \tag{15}$$

$$w \leq \{square(x, N - x + 1) \mid num(x)\} \leq w \leftarrow \tag{16}$$

where *square* is a function of the type $num \times num \rightarrow value[allDifferent]$, the domain of *num* is $\{1, 2, \ldots, N\}$, the domain of *value* is $\{1, 2, \ldots, N^2\}$, $w = N(N^2+1)/2$, $\{square(x, y) \mid num(y)\}$ stands for $\sum_{num(y)} square(x, y)$, and the rule $l \leq t \leq w \leftarrow Body$ stands for the two rules $l \leq t \leftarrow Body$ and $t \leq w \leftarrow Body$, where l, w are real numbers and t is a numeric term. The intended meaning of lines (13)–(16) is that the sum of the numbers in each column, each row and each diagonal are exactly $N(N^2+1)/2$.

Note that a predicate $sqr/3$ is used in the encoding of the problem [29], where $sqr(x, y, v)$ means that the value at the entry (x, y) is v. In [29] the following weight rules are needed to ensure that all entries have distinct values from $1..N^2$ and each entry has exactly one value from $1..N^2$, respectively

$$1\{sqr(I, J, D) : num(I; J)\}1 \leftarrow data(D).$$

$$1\{sqr(I, J, D) : data(D)\}1 \leftarrow num(I), num(J).$$

[10] http://dtai.cs.kuleuven.be/events/ASP-competition/index.shtml

This is an explicit encoding of the *allDifferent* constraint. In our encoding, if we do not use *AllDifferent* in the declaration of the function *square*, we will then need the following rules

$$\leftarrow square(x, y) = square(x', y'), x \neq x',$$
$$\leftarrow square(x, y) = square(x', y'), y \neq y'$$

to guarantee that the values at different entries be distinct.

We compare the time cost in seconds (excluding the grounding cost) with the ASP solvers: SMODELS, CMODELS using *MiniSat*, CLASP, and PBMODELS using *satzoo*. In order to investigate the effectiveness of global constraints in FASP, we also tested the magic square problem for which the weight program encoding does not use the global constraint *allDifferent*.

We also tested the encodings in constraint answer program for the problem [12]. While translating these encodings into traditional logic programs, the global constraint *permutation* is translated using directed encoding, supported encoding and range encoding respectively. The experimental results show that the supported encoding outperforms the other two, thus only the performance of the supported encoding approach is reported. All the experimental results are summarized in Table 1.

It is clear that FASP outperforms the other ASP solvers under consideration for this benchmark. It also outperforms the best translational approach for the problem [12] as illustrated in Table 1. One should also notice that using the global constraint *allDifferent* leads to better efficiency for the problem. The only one exception is for $N = 10$. For grounding sizes, we also report the sizes of these ground programs in *lparse* and in our FASP. The ground programs in *lparse* are generated using *lparse* with the "-t" option. The result in Table 1 shows that using evaluable functions leads to smaller ground program size than that of *lparse*.

As we noted earlier, grounding could be a bottleneck for ASP solvers. For instance, using the well-known encoding for the N-queens problem, CLASP outperforms any known CSP solvers (to the best of our knowledge) for the problem size $N \leq 50$; when the ground program size becomes greater than one Gigabyte, CLASP degrades its performance severely [26]. However, for the magic square problem, it is clear that grounding is not a serious problem for CLASP as illustrated by the small sizes of the

Table 1 The magic N-square problem

N	smodels	cmodels	clasp	pbmodels	inca-s	FASP		Size	
						allDiff	No-allDiff	lparse	FASP
4	0.19	7.70	0.01	1.86	0.02	0.02	0.08	55K	832B
5	–	300.48	0.88	24.77	0.45	0.01	0.17	129K	1.2K
6	–	–	15.09	87.55	101.07	0.03	0.38	259K	1.7K
7	–	–	450.58	316.20	49.21	1.63	6.62	472K	2.2K
8	–	–	–	–	613.93	1.50	150.59	796K	2.8K
9	–	–	–	–	–	0.30	66.30	1.3M	3.4K
10	–	–	–	–	–	–	6.93	1.9M	4.2K

Legends: –: no result in 0.5 h; *inca-s*: CLASP 1.35 for the translated constraint answer set programs where global constraints are in support encoding; *allDiff*: with global constraint *AllDifferent*; *No-allDiff*: without global constraint *AllDifferent*; *lparse*: the size in LPARSE encoding; *FASP*: the size in FASP (for allDiff)

ground programs in Table 1 (even though they are still substantially larger than those of FASP). This makes us believe that it is the use of the global constraint *allDifferent* that contributed to the huge performance gap, especially for FASP with *Mistral*. Note that a typical implementation of a global constraint in CSP involves some special data structures and efficient and dedicated propagators. This is an advantage compared with ASP encodings and solvers without global constraints. For the magic square problem, *allDifferent* is constrained on a much larger number than N, i.e., N^2, which appears to have made search space pruning very effective by the implementation of the global constrain *allDifferent* in the CSP solver *Mistral*.

We notice that, instead of relying on existing implementations of global constraints, one can decompose some global constraints into simple arithmetic constraints on which bound or range consistency can be achieved; in some cases even greater pruning is possible [3]. This is also true for ASP as illustrated by Drescher and Walsh [12]. Note that our approach does not depend on whether a global constraint can be effectively decomposed to simple arithmetic constraints so that some standard consistency techniques can be achieved; it however does depend on an existing implementation of CSP. This is precisely what we are proposing in this paper—using the off-the-shelf CSP solvers to compute answer sets.

4.2.2 The other problems

We report the experimental results on the benchmarks of TSP, weighted N-queens, weighted Latin square, and weight-bounded dominating set problems below. The weight programs for TSP and weighted N-queens were given earlier in Section 3.

Weighted Latin square problem (WLSQ) The problem is to place one number from $\{1, \ldots, n\}$ to an n by n weighted board such that each number in $\{1, \ldots, n\}$ occurs exactly once in each row and each column, and the weight of the placement in each row (the sum of the weight of the cell times the number in that cell) is no more than a bound. We can encode the problem by the following rules:

$$\leftarrow latin(x, y) = latin(x, z), y \neq z,$$

$$\leftarrow latin(x, y) = latin(z, y), x \neq x,$$

$$\{latin(x, y) \times wt(position(x, y)) \,|num(y)\} \leq w \leftarrow num(x).$$

where *latin* is a function of the type $num \times num \rightarrow num$, *position* is a predicate of the type $num \times num$, the domain of *num* is $\{1, 2, \ldots, N\}$, $wt(position(x, y))$ is the predefined weight for $position(x, y)$, and w is the upper bound for a solution. The intuitive meanings of the above rules should be clear.

Weight-bounded dominating set problem (WBDS) The problem deals with directed graphs $G = (V, E)$, where V is the set of vertices and E the set of edges, such that every edge (x, y) in E is associated with a weight $wt(edge(x, y))$. Furthermore, we consider a certain cardinality k and a minimum weight w. The problem is to find a subset D of V such that $|D| \leq k$ and, for each vertex v in V, at least one of the following conditions holds:

(i) $v \in D$,
(ii) $\sum_{x \in D \text{ and } (x,v) \in E} wt(edge(x, v)) \geq w$, or

Table 2 Summary of instances

Instance	smodels	cmodels	clasp	pbmodels	FASP
tsp–e	(47,0)/153.94	(50,0)/30.36	(50,0)/1.75	(32,0)/514.61	(50,0)/4.77
tsp–h	(17,0)/755.36	(35,14)/33.98	(36,14)/72.87	(6,1)/608.12	(35,12)/208.78
wnq–e	(13,0)/585.88	(36,0)/403.04	(49,0)/232.28	(32,0)/732.49	(25,0)/543.17
wnq–h	(2,0)/1613.34	(4,0)/686.71	(21,3)/816.64	(0,0)/	(3,0)/262.91
wlsq–e	(23,0)/24.02	(45,5)/50.38	(45,5)/1.10	(45,5)/60.80	(45,5)/1.30
wlsq–h	(0,0)/	(8,42)/135.18	(8,42)/5.27	(8,42)/306.66	(8,42)/34.18
wbds	(5,0)/91.75	(16,0)/138.23	(19,0)/54.51	(15,0)/60.62	(11,0)/207.79
Total/avg	(108,0)/298.42	(194,61)/125.21	(228,64)/123.51	(138,48)/431.11	(177,59)/120.67

Legends: (#1,#2)/#3: the number of solved SAT instances is #1, the number of solved UNSAT instances is #2, the average time consumption for the total solved instances is #3

(iii) $\sum_{x \in D \text{ and } (v,x) \in E} wt(edge(v, x)) \geq w$.

We encode the problem by a weight program as below:

$$\{in(x)|vtx(x)\} \leq k \leftarrow, \tag{17}$$

$$\leftarrow \{in(x)\}0, \{in(y) \times wt(edge(x, y))|edge(x, y)\} \leq w - 1,$$
$$\{in(y) \times wt(edge(y, x))|edge(y, x)\} \leq w - 1, vtx(x). \tag{18}$$

where *in* is a function of the type $vtx \rightarrow dom$, the domain of *vtx* consists of the vertices of G, the domain of *dom* is $\{0, 1\}$, $in(v) = 1$ stands for $v \in D$, while $in(v) = 0$ for $v \notin D$. An instance of the problem is specified by a domain for *vtx*, a set of facts for $edge(x, y)$ whose weight is predefined by $wt(edge(x, y))$, and two numbers k and w for the cardinality of a dominating set and a minimal weight respectively. Intuitively, the rule (17) requires the number of vertices in a set D to be less than or equal to k, and the conjunction of the three extended weight constraints in (18) captures the inverse of the conditions (i)–(iii) in the problem description, respectively.

The benchmark for each of the first three problems includes 50 easy instances and 50 hard instances, and 20 instances for the last one. For the instances in Table 2, we use "–e" and "–h" to indicate they are easy and hard respectively. For each tested solver, we report the number of solved satisfiable and unsatisfiable instances in 30 minutes, besides the average running time for those solved instances. We also summarize the total number of solved instances and the average running time for each of the tested solvers. These results are summarized in Table 2.

It seems that the performance ordering over these solvers should look like CLASP > CMODELS(*MiniSat 2.0 beta*) > FASP(*Mistral*) > PBMODELS(*satzoo*) > SMODELS in terms of the number of solved instances, and FASP(*Mistral*) > CLASP > CMODELS(*MiniSat 2.0 beta*) > PBMODELS(satzoo) > SMODELS in terms of the average time cost for the solved instances.

5 Related work

For adding functions into ASP, a majority of recent work support functions over the Herbrand interpretations [2, 9, 14]. Quantified Equilibrium Logic (QEL) [38] and the General Theory of Stable Models [16] allow non-Herbrand interpretations. In

QEL, the equilibrium models are Kripke structures while the answer set semantics for the General Theory of Stable Models is defined by translating a sentence into a second-order one.

One noticeable exception is the work of [6, 7]. Cabalar and Lorenzo [7] introduced a pure functional logic programming language. Relations are considered as functions with only two possible values, true or false. There is no negation-as-failure operator in the language. Instead, functions can take on default values. This language is extended by Cabalar [6], and used as an action language.

A major difference between Cabalar and Lorenzo's formalism and ours is that functions can be partial in theirs but must be total in ours. For instance, consider the following program

$$f : \{1\} \to \{a, b\},$$
$$\leftarrow f(1) = a.$$

According to our semantics, the unique answer set of this program is $\{f(1) = b\}$. However, the unique model is the empty set according to theirs.

In a sense, one can see the language proposed here as a middle ground between traditional logic programming languages, which encode functions as relations, and the languages of [6, 7], which encode relations as functions.

Since we translate weight programs into CSPs based loop completion, the CSP facilities such as global constraints can be readily brought into the ASP language for weight programs. In this sense, weight programs can be seen as an attempt to integrate CSP with ASP. Gebser et al. proposed *constraint answer solving* by integrating Constraint Processing techniques into ASP in which function symbols are also permitted [19]. But the functions are uninterpreted. Mellarkod *et al.* proposed a knowledge representation language $\mathcal{AC}(\mathcal{C})$ which integrates answer set programming and constraint logic programming, in which the purpose of constraints is to avoid full grounding of logic programs [33]. In our approach, a program is translated in its entirety into a CSP. We should also mention that Dovier et al. experimentally compared ASP with constraint logic programming over finite domains on various classes of combinational search problems [11].

6 Concluding remarks and future work

In this paper, we proved the splitting set theorem and formulated completion and loop formulas for weight constraint programs, and then embedded evaluable functions into weight constraint programs, called weight programs, for which we defined answer set semantics, completion and loop formulas, and showed that answer sets of such a weight program can be characterized by models of its loop completion. This enables us to extend FASP to compute answer sets of a weight program. We implemented such a prototype and tested it with the benchmarks for the magic N-square problem, the TSP problem, the weighted N-queens problem, the weighted Latin-square problem and the weight-bounded dominating set problem. Comparing with the state-of-the-art ASP solvers, including SMODELS, CMODELS, CLASP and PBMODELS, FASP has a clear cut advantage over all of these ASP solvers for the magic N-square problem, and performs competitively for the remaining benchmarks.

There are two main benefits by allowing evaluable functions in ASP. One is the natural yet compact encoding of some problems in ASP, which sometimes leads to substantial size reduction in grounded programs. The other is the embedding of predefined constraints into functions. This becomes especially convenient if a CSP solver with global constraints is used to compute the answer sets of this type of programs. Our past experience shows that, typically for scheduling benchmarks, CSP-based solvers tend to be more efficient, sometimes they can be orders of magnitude faster than the traditional ASP solvers. Although in this paper we have shown how to use a CSP solver to compute answer sets, there are situations where the CSP approach to constraint solving may not be the best choice [5]. Hence, integration of ASP with CSP deserves more attention.

A number of theoretical and practical questions deserve further attention. First, it is of theoretical interest to investigate the complexity and expressive power of (non-ground) weight programs over finite structures, along the line of [8]. Another question of interest is a generalization of the module theorem of [36] from a special class of weight programs to the class of all weight programs directly, without relying on a transformation. In this case, the splitting theorem given in this paper becomes a special case. The third is from the perspective of practice. As we have mentioned, our current implementation works only for ground weight programs. It is interesting to design a grounder, like LPARSE for ASP solvers, or alternatively, to design a first-order solver for weight programs with variables, like the effort on first-order logic programs [10, 13]. This task is nontrivial. In addition, the problem of enumerating answer sets for FASP should also be considered, following the approach given in [18] to avoid an exponential blow-up. The challenge is whether we are still able to treat a CSP solver as a black box, and if not, what are the minimum changes on a CSP solver that must be done.

Acknowledgements We thank the reviewers for their detailed comments, which helped improve the presentation of this paper. We also appreciate Dr. Jianyong Pi for providing us a server to run our benchmarks. Yisong Wang, Fangzhen Lin and Mingyi Zhang were partially supported by the Natural Science Foundation of China under grant 60963009, the Fund of Guizhou Science and Technology: 2008[2119]. Yisong was also partially supported by the Fund of Education Department of Guizhou Province: 2008[011]. Jia-Huai You and Li Yan Yuan were supported in part by NSERC discovery grants, and by the 863 Project of China under grant 2009AA01Z150.

Appendix: Proofs

Given a weight constraint $l \leq S \leq u$ and a set M of atoms, where $S = \{c_1 : w_1, \ldots, c_n : w_n\}$, we define the following notations:

- $\Gamma_1^+(S, M) = \{p : w | p \in M \text{ and } (p : w) \in S\}$;
- $\Gamma_1^-(S, M) = \{not\ p : w | p \in M \text{ and } (not\ p : w) \in S\}$;
- $\Gamma_1(S, M) = \Gamma_1^+(S, M) \cup \Gamma_1^-(S, M)$;
- $\Gamma_2^+(S, M) = \{p : w | p \notin M \text{ and } (p : w) \in S\}$;
- $\Gamma_2^-(S, M) = \{not\ p : w | p \notin M \text{ and } (not\ p : w) \in S\}$;
- $\Gamma_2(S, M) = \Gamma_2^+(S, M) \cup \Gamma_2^-(S, M)$;
- $\Gamma^-(S) = \{not\ p : w | (not\ p : w) \in S\}$.
- $\Gamma^+(S) = S \setminus \Gamma^-(S)$;

It is clear that, for any set M of atoms, $\Gamma^+(S) = \Gamma_1^+(S, M) \cup \Gamma_2^+(S, M)$ and $\Gamma^-(S) = \Gamma_1^-(S, M) \cup \Gamma_2^-(S, M)$; and $S = \Gamma_1^+(S, M) \cup \Gamma_1^-(S, M) \cup \Gamma_2^+(S, M) \cup \Gamma_2^-(S, M)$. Intuitively, we use $\Gamma_1(S, M)$ and $\Gamma_2(S, M)$ to divide S into two parts such that rule elements in one part mentions some atom in M, while the other part consists of the remaining pairs in S. The following two lemmas are evident and we prove the third one.

Lemma 1 *Let $l \leq S$ be a weight constraint that mentions no negative rule element, M and M' be two sets of atoms with $M' \subseteq M$. Then $M' \models l \leq S$ implies $M \models l \leq S$.*

Lemma 2 *Let $l \leq S$ be a weight constraint and M a set of atoms. We have that $M \models l \leq S$ iff $M \models (l \leq S)^M$.*

Lemma 3 *Let C be a weight constraint, M_1, M_2 and V sets of atoms such that $M_2 \cap V = \emptyset$ and $M_1 \subseteq V$. Then $M_2 \models e_V(C, M_1)$ iff $M_1 \cup M_2 \models C$.*

Proof Let $C = l \leq S \leq u$.

$$
\begin{aligned}
M_2 \models e_V(C, M_1) & \\
& \Leftrightarrow M_2 \models l - \Sigma(\Gamma_1(S, V), M_1) \leq \Gamma_2(S, V) \leq u - \Sigma(\Gamma_1(S, V), M_1) \\
& \Leftrightarrow l - \Sigma(\Gamma_1(S, V), M_1) \leq \Sigma(\Gamma_2(S, V), M_2) \leq u - \Sigma(\Gamma_1(S, V), M_1) \\
& \Leftrightarrow l \leq \Sigma(\Gamma_1(S, V), M_1) + \Sigma(\Gamma_2(S, V), M_2) \leq u \\
& \Leftrightarrow l \leq \Sigma(\Gamma_1(S, V), M_1 \cup M_2) + \Sigma(\Gamma_2(S, V), M_2 \cup M_1) \leq u \\
& \Leftrightarrow l \leq \Sigma(\Gamma_1(S, V) \cup \Gamma_2(S, V), M_1 \cup M_2) \leq u \\
& \Leftrightarrow l \leq \Sigma(S, M_1 \cup M_2) \leq u \\
& \Leftrightarrow M_1 \cup M_2 \models C.
\end{aligned}
$$

□

Lemma 4 *Let $C = l \leq S$ be a weight constraint, M_1, M_2 and V be sets of atoms such that $M_1 \subseteq V$ and $M_2 \cap V = \emptyset$. Then for any set M^* with $M_1 \subseteq M^* \subseteq M_1 \cup M_2$, we have that $M^* \models C^{M_1 \cup M_2}$ iff $M^* \models [e_V(C, M_1)]^{M_2}$.*

Proof Let $M = M_1 \cup M_2$.

$$
\begin{aligned}
M^* \models [e_V(C, M_1)]^{M_2} & \\
& \Leftrightarrow M^* \models [l - \Sigma(\Gamma_1(S, V), M_1) \leq \Gamma_2(S, V)]^{M_2} \\
& \Leftrightarrow M^* \models l - \Sigma(\Gamma_1^+(S, V) \cup \Gamma_1^-(S, V), M_1) - \Sigma(\Gamma^-(\Gamma_2(S, V)), M_2) \leq \Gamma_2^+(S, V) \\
& \Leftrightarrow l - \Sigma(\Gamma_1^+(S, V), M_1) - \Sigma(\Gamma_1^-(S, V), M_1) - \Sigma(\Gamma_2^-(S, V), M_2) \\
& \quad \leq \Sigma(\Gamma_2^+(S, V), M^*) \\
& \Leftrightarrow l - \Sigma(\Gamma_1^-(S, V) \cup \Gamma_2^-(S, V), M_1 \cup M_2) \leq \Sigma(\Gamma_1^+(S, V), M_1) \\
& \quad + \Sigma(\Gamma_2^+(S, V), M^*) \text{ (due to } M_1 \subseteq V, M_2 \cap V = \emptyset)
\end{aligned}
$$

$$\Leftrightarrow l - \Sigma(\Gamma^-(S), M) \leq \Sigma(\Gamma_1^+(S, V), M_1) + \Sigma(\Gamma_2^+(S, V), M^*)$$
$$\Leftrightarrow l - \Sigma(\Gamma^-(S), M) \leq \Sigma(\Gamma_1^+(S, V), M^*) + \Sigma(\Gamma_2^+(S, V), M^*)$$
(due to $M_1 \subseteq M^*$ and $(M^* \setminus M_1) \cap V = \emptyset$)
$$\Leftrightarrow l - \Sigma(\Gamma^-(S), M) \leq \Sigma(\Gamma_1^+(S, V) \cup \Gamma_2^+(S, V), M^*)$$
$$\Leftrightarrow l - \Sigma(\Gamma^-(S), M) \leq \Sigma(\Gamma^+(S), M^*)$$
$$\Leftrightarrow M^* \models l - \Sigma(\Gamma^-(S), M_1 \cup M_2) \leq \Gamma^+(S)$$
$$\Leftrightarrow M^* \models (l \leq S)^{M_1 \cup M_2}$$
$$\Leftrightarrow M^* \models C^{M_1 \cup M_2}.$$

□

Lemma 5 *Let P be a weight constraint program and the weight constraint program P' is obtained from P by replacing each rule r of the form (2) by the following rules:*

$$C_0 \leftarrow n \leq \{p_1' : 1, \ldots, p_n' : 1\},$$
$$p_i' \leftarrow C_i, \qquad (1 \leq i \leq n)$$

where $p_1', \ldots, p_n'$ are distinct fresh propositional atoms. We have that a set Z of atoms in $\mathcal{A}toms(P)$ is an answer set of P if and only if Z' is an answer set of P' where

$$Z' = Z \cup \{p_i' \mid p_i' \text{ is the introduced fresh atom in } P' \text{ for } C_i \text{ such that } Z \models C_i\}.$$

Proof Firstly, it is obvious that $Z \models P$ if and only if $Z' \models P'$.

Secondly, for any rule $(r : C_0 \leftarrow l_1 \leq S_1 \leq u_1, \ldots, l_n \leq S_n \leq S_n)$ of P and any atom $a \in \mathcal{A}toms^+(C_0) \cap Z$, the rule $\big(a \leftarrow (l_1 \leq S_1)^Z, \ldots, (l_n \leq S_n)^Z\big)$ obtained from r by the reduct with respect to Z belongs to P^Z if and only if $P'^{Z'}$ contains the following rules:

$$p_i' \leftarrow (l_i \leq S_i)^{Z'}, \qquad (1 \leq i \leq n)$$
$$a \leftarrow n \leq \{p_1' : 1, \ldots, p_n' : 1\}.$$

It is not difficult to see that $cl(P^Z) = \mathcal{A}toms(P) \cap cl\big(P'^{Z'}\big)$. Consequently, Z is an answer set of P if and only if Z' is an answer set of P'. □

In the following, for the sake of clarity, we assume that every rule of a weight program has at most one weight constraint in its body, unless stated otherwise.

Given a set of atoms M and a weight constraint program P that consists of Horn rules only, we define the *immediate consequence operator* T_P as follows:

$$T_P(M) = \{q | (q \leftarrow l_1 \leq S_1, \ldots, l_n \leq S_n) \text{ in } P \text{ and } M \models l_1 \leq S_1, \ldots, M \models l_n \leq S_n\}.$$

Evidently, T_P is monotonic and it has the least fixpoint, written as $lfp(T_P)$, which is the least model of P, i.e., the deductive closure of P. Clearly $lfp(T_P) = T_P^\infty$, where

- $T_P^0 = \emptyset$;
- $T_P^{k+1} = T_P\big(T_P^k\big)$ for $k \geq 0$.

Proof of Theorem 1

($\Rightarrow$) Let $M_1 = V \cap M$ and $M_2 = M \setminus M_1$. It is evident that $M_1 \subseteq V$ and $M_2 \cap V = \emptyset$. Since $b_V(P)$ mentions no atoms in M_2, we have that $M_1 \cup M_2 \models P$
$\Rightarrow M_1 \cup M_2 \models b_V(P)$
$\Rightarrow M_1 \models b_V(P)$.
Note that $M_1 \cup M_2 = cl(P^{M_1 \cup M_2}) = cl(b_V(P)^{M_1 \cup M_2} \cup (P \setminus b_V(P))^{M_1 \cup M_2})$, and there is no rule in $P \setminus b_V(P)$ whose head mentions an atom in V. It follows that $[b_V(P)]^{M_1 \cup M_2} = [b_V(P)]^{M_1}$ and $M_1 = cl([b_V(P)]^{M_1})$. Thus M_1 is an answer set of $b_V(P)$.
Let $(r : C_0 \leftarrow C_1)$ be an arbitrary rule in $P \setminus b_V(P)$. By $M_1 \cup M_2 \models r$, we have that $M_1 \cup M_2 \models C_1$ implies $M_1 \cup M_2 \models C_0$. By Lemma 3, it is clear that $M_2 \models e_V(C_1, M_1)$ implies $M_2 \models e_V(C_0, M_1)$, i.e., $M_2 \models e_V(\{r\}, M_1)$. Thus $M_2 \models e_V(P \setminus b_V(P), M_1)$.
Let us consider the above rule r again. Suppose $C_1 = l_1 \leq S_1 \leq u_1$. Note that the rule

$$r' : q \leftarrow (l_1 \leq S_1)^{M_1 \cup M_2} \tag{19}$$

belongs to $\{r\}^{M_1 \cup M_2}$ if and only if the rule

$$r'' : q \leftarrow \left[e_V(l_1 \leq S_1, M_1)\right]^{M_2} \tag{20}$$

is in $[e_V(\{r\}, M_1)]^{M_2}$, since $M_1 \cup M_2 \models S_1 \leq u_1$ iff $M_2 \models e_V(S_1 \leq u_1, M_1)$ by Lemma 3. We prove $M_2 = cl([e_V(P \setminus b_V(P), M_1)]^{M_2})$ by showing that, for any number $k \geq 0$,

$$M_2 \cap T^k_{(M_1 \cup P \setminus b_V(P))^{M_1 \cup M_2}} = T^k_{e_V(P \setminus b_V(P), M_1)^{M_2}} \tag{21}$$

Please note that in (21), a set of atoms is understood as a set of facts.
Base: It is clear for $k = 0$.
Step: Suppose it holds for $k \leq n$. For any atom q,

$$q \in M_2 \cap T^{n+1}_{(M_1 \cup P \setminus b_V(P))^{M_1 \cup M_2}}$$

iff there is a rule of the form (19) in $(P \setminus b_V(P))^{M_1 \cup M_2}$ such that

$$T^n_{(M_1 \cup P \setminus b_V(P))^{M_1 \cup M_2}} \models (l_1 \leq S_1)^{M_1 \cup M_2}$$

iff $T^n_{[e_V(P \setminus b_V(P), M_1)]^{M_2}} \models (l_1 \leq S_1)^{M_1 \cup M_2}$ by the inductive assumption
iff there is a rule of the form (20) in $\left[e_V(P \setminus b_V(P), M_1)\right]^{M_2}$ such that

$$T^n_{[e_V(P \setminus b_V(P), M_1)]^{M_2}} \models \left[e_V(l_1 \leq S_1, M_1)\right]^{M_2}$$

by Lemma 4
iff $q \in T^{n+1}_{[e_V(P \setminus b_V(P), M_1)]^{M_2}}$.
Note that $M_1 \cup M_2 = T^{\infty}_{(M_1 \cup P \setminus b_V(P))^{M_1 \cup M_2}}$, thus $cl([e_V(P \setminus b_V(P), M_1)]^{M_2}) = M_2$. Then M_2 is an answer set of $e_V(P \setminus b_V(P), M_1)$.

($\Leftarrow$) Let $M = M_1 \cup M_2$, where M_1 and M_2 are answer sets of $b_V(P)$ and $e_V(P \setminus b_V(P), M_1)$ respectively. Firstly, we have that $M_1 \models b_V(P)$ implies $M_1 \cup M_2 \models b_V(P)$, since $b_V(P)$ mentions no atom in M_2; and $M_2 \models e_V(P \setminus b_V(P), M_1)$ implies $M_1 \cup M_2 \models P \setminus b_V(P)$ by Lemma 3. Thus $M_1 \cup M_2 \models P$.

Secondly, for any rule in $e_V(P \setminus b_V(P), M_1)$, if its reduct relative to M_2 has a rule of the form (20) in $e_V(P \setminus b_V(P), M_1)^{M_2}$, then there is a corresponding rule of the form (19) in $(P \setminus b_V(P))^{M_1 \cup M_2}$, by Lemma 3 again. From (21), we have

$$\begin{aligned} M_2 \cap cl\big(M_1 \cup (P \setminus b_V(P))^{M_1 \cup M_2}\big) \\ = cl\big(e_V(P \setminus b_V(P), M_1)^{M_2}\big) \\ = M_2. \end{aligned}$$

Notice that there is no rule in $(P \setminus b_V(P))^{M_1 \cup M_2}$ whose head mentions an atom in M_1. Thus

$$\begin{aligned} M_1 \cup M_2 &= cl\big(M_1 \cup (P \setminus b_V(P))^{M_1 \cup M_2}\big) \\ &= cl\big(b_V(P)^{M_1} \cup (P \setminus b_V(P))^{M_1 \cup M_2}\big) \\ &= cl\big(b_V(P)^{M_1 \cup M_2} \cup (P \setminus b_V(P))^{M_1 \cup M_2}\big) \\ &= cl\big(P^{M_1 \cup M_2}\big). \end{aligned}$$

Consequently, $M_1 \cup M_2$ is an answer set of P. □

The following lemma is evident.

Lemma 6 *Let P be a weight constraint program and a set $M \subseteq \mathcal{A}\text{toms}(P)$ satisfying $COMP(P)$. Then for any $a \in M$, there is a rule $(a \leftarrow C^M)$ in P^M such that $M \models C^M$.*

Let M be a set of atoms and r a rule of the form (2). The rule r is *M-applicable* if M satisfies every weight constraints in the body of r. For a weight constraint program P, by $AP(P, M)$ we denote the set of all M-applicable rules in P. A model M of P is *supported* if $M \subseteq \bigcup_{r \in AP(P,M)} \mathcal{A}toms^+(Head(r))$.

Proposition 4 *Let P be a weight constraint program and M a set of atoms. Then M is a supported model of P if and only if M is a model of $COMP(P)$.*

Proof

(⇒) Note that M is model of P implies that M satisfies all formulas in $COMP(P)$ of the type (4). For any atom $a \in M$, we have $a \in \mathcal{A}toms^+(C')$ for some rule $(C' \leftarrow C)$ in $AP(P, M)$. Thus $M \models C$. It follows that M satisfies the second type (5) formulas in $COMP(P)$.

(⇐) It is clear that $M \models P$ since M satisfies the first type (4) formulas in $COMP(P)$. For any atom $a \in M$, since M satisfies the second type (5) formulas in $COMP(P)$, thus there is at least one rule $(C' \leftarrow C)$ in P such that $M \models C$. Consequently, $(C' \leftarrow C)$ is M-applicable and $a \in \bigcup_{r \in AP(P,M)} \mathcal{A}toms^+(C')$, i.e., M is a supported model of P. □

Lemma 7 *Let $l \leq S$ be a weight constraint, M and L be two sets of atoms. Then we have $\big[(l \leq S)^M\big]_{|L} = \big[(l \leq S)_{|L}\big]^M$.*

Proof

$$\begin{aligned}
\left[(l \leq S)_{|L}\right]^M &= \left[l \leq \Gamma_2(S, L) \cup \Gamma_1^-(S, L)\right]^M \\
&= l - \Sigma(\Gamma^-(\Gamma_1^-(S, L) \cup \Gamma_2(S, L)), M) \leq \Gamma_2^+(S, L) \\
&= l - \Sigma(\Gamma^-(\Gamma_1^-(S, L) \cup \Gamma_2^+(S, L) \cup \Gamma_2^-(S, L)), M) \leq \Gamma_2^+(S, L) \\
&= l - \Sigma(\Gamma_1^-(S, L) \cup \Gamma_2^-(S, L), M) \leq \Gamma_2^+(S, L) \\
&= l - \Sigma(\Gamma^-(S), M) \leq \Gamma_2^+(S, L) \\
&= \left[l - \Sigma(\Gamma^-(S), M) \leq \Gamma^+(S)\right]_{|L} \\
&= \left[(l \leq S)^M\right]_{|L}.
\end{aligned}$$

□

Proposition 5 *Let P be a weight constraint program and M an answer set of P. Then* (1) *M is a model of $COMP(P)$ and* (2) *M satisfies $LF(L, P)$ for any loop L of P.*

Proof

(1) It is clear that M satisfies the formulas of the type (4) in $COMP(P)$ since $M \models P$. Suppose $a \in M$ and

$$a \supset \bigvee_{1 \leq i \leq n} C_i \tag{22}$$

be the formula of the type (5) in $COMP(P)$ where $(C_i' \leftarrow C_i)$ are all the rules of P such that $a \in \mathcal{A}toms^+(C_i')$ for every $1 \leq i \leq n$. Note that $a \in cl(P^M)$. Thus there exists at least one rule $(a \leftarrow [l_1 \leq S_1]^M)$ in P^M such that $M \models [l_1 \leq S_1]^M$. It follows that there is at least one rule: $(C_0 \leftarrow l_1 \leq S_1 \leq u_1)$ in P such that $a \in \mathcal{A}toms^+(C_0)$ and $M \models S_1 \leq u_1$. By Lemma 2, $M \models l_1 \leq S_1 \leq u_1$. Thus M satisfies the formula (22). Consequently, $M \models COMP(P)$.

(2) Let $L = \{a_1, \ldots, a_m\}$ be an arbitrary loop of P and $LF(L, P)$ be the following formula

$$\bigvee L \supset \bigvee_{1 \leq i \leq n} C_{i_{|L}} \tag{23}$$

where $(C_i' \leftarrow C_i)$ are the rules of P such that $\mathcal{A}toms^+(C_i') \cap L \neq \emptyset$ for every $1 \leq i \leq n$. Suppose $M \models \bigvee L$ and $M \not\models C_{i|L}$ for each $1 \leq i \leq n$. The former implies that there exists an atom $a \in M \cap L$. Without loss of generality, let us assume that $a = a_1$.

Note that $M = cl(P^M)$. Thus there is the least number $k_1 (k_1 > 0)$ such that $a_1 \in T_{P^M}^{k_1}$, i.e., there is a rule $(r_1' : a_1 \leftarrow (l_1 \leq S_1)^M)$ in P^M such that $T_{P^M}^{k_1 - 1} \models (l_1 \leq S_1)^M$. It follows that $M \models (l_1 \leq S_1)^M$ by Lemma 1. Suppose the rule r_1' is obtained from the rule $(r_1 : C \leftarrow l_1 \leq S_1 \leq u_1)$ of P by reduction relative to M. By $M \models S_1 \leq u_1$, we have $M \models (S_1 \leq u_1)_{|L}$. Thus $M \not\models (l_1 \leq S_1)_{|L}$ by $M \not\models (l_1 \leq S_1 \leq u_1)_{|L}$. It implies that there exists an atom in $T_{P^M}^{k_1 - 1} \cap \mathcal{A}toms^+(l_1 \leq S_1) \cap L$ which is different from the atom a_1. Let us assume the atom is a_2. Similarly, there exists the least number k_2 such that $a_2 \in T_{P^M}^{k_2}$. Clearly, $a_2 \in L \setminus \{a_1\}$ and $k_2 < k_1$. By this iterative construction, we have the sequence

$a_1, a_2, \ldots, a_i, \ldots, a_m$ such that for any a_i where $1 \leq i \leq m$, there exists the least number k_i such that $a_i \in T_{P^M}^{k_i}$. Now let us consider the atom a_m, we have the least integer k_m such that

$$a_m \in T_{P^M}^{k_m}.$$

Thus there is a rule r'_m:

$$a_m \leftarrow (l_m \leq S_m)^M$$

in P^M such that

$$T_{P^M}^{k_m-1} \models (l_m \leq S_m)^M.$$

Since $M \not\models (l_m \leq S_m)_{|L}$, it implies that there is an atom b in

$$T_{P^M}^{k_m-1} \cap \mathcal{A}toms^+(l_m \leq S_m) \cap L.$$

However, $b \in L \setminus \{a_1, \ldots, a_m\}$. It is absurd. Consequently, M satisfies $LF(L, P)$. □

Proposition 6 *Let P be a weight constraint program and M a set of atoms such that $M \models COMP(P) \cup LF(P)$ where $LF(P)$ is the set of loop formulas of P. Then M is an answer set of P.*

Proof Let Γ be the set of rules in P^M such that both their head and body are satisfied by M. By Proposition 4 and Lemma 6, M is a supported model of P and for any atom $a \in M$ there exists a rule $(a \leftarrow [l \leq S]^M)$ in Γ such that $M \models [l \leq S]^M$.

Let $M^* = cl(P^M)$ and $M^- = M \setminus M^*$. It is clear that $M^* \subseteq M$ and $M^* = cl(\Gamma)$ since $M \models P^M$. If M is not an answer set of P then $M^- \neq \emptyset$. Let Γ_{M^-} be the set of rules of Γ whose head is an atom in M^-. We claim that the positive dependency graph of Γ_{M^-} has at least one loop.

Consider an arbitrary atom $a \in M^-$. Suppose $(r' : a \leftarrow l' \leq S')$ is an arbitrary rule in Γ_{M^-} such that r' is obtained from $(r : C \leftarrow l \leq S \leq u)$ of P by the reduction relative to M where $a \in \mathcal{A}toms^+(C)$, $M \models S \leq u$, $(l' \leq S') = [l \leq S]^M$, $l' = l - \Sigma(\Gamma^-(S), M)$, and $S' = \Gamma^+(S)$. Note that $M^* \not\models [l \leq S]^M$ and $M \models [l \leq S]^M$. Thus we have

$$l' \leq \Sigma(S', M^*) + \Sigma(S', M \setminus M^*) \tag{24}$$

and

$$l' > \Sigma(S', M^*) \tag{25}$$

It follows that $M^- \cap \mathcal{A}toms^+(S') \neq \emptyset$. In this way, we can construct a sequence $(a_0(= a), a_1, \ldots, a_i, \ldots)$ of atoms such that $a_i \in M^-$ and a_i depends on a_{i+1} in the sense that there is a rule $(a_i \leftarrow C)$ in Γ_{M^-} such that $a_{i+1} \in \mathcal{A}toms^+(C) \cap M^-$. Since Γ_{M^-} is finite, and the sequence mentions only finite number of atoms. Thus the above constructed sequence must contain a loop. It also implies that Γ_{M^-} has at least one terminating loop.

Now let L be a terminating loop of Γ_{M^-} and $a \in L$. It is evident that $L \subseteq M^-$. We claim that for such above rule $\left(r' : a \leftarrow l' \leq S'\right)$ of Γ_{M^-},

$$M^- \cap \mathcal{A}toms^+(l' \leq S') \subseteq L. \tag{26}$$

Otherwise, suppose $b \in M^- \cap \mathcal{A}toms^+(l' \leq S')$ but $b \notin L$. In terms of the above construction, we can construct a sequence $(b_0 = (b), \ldots, b_i, \ldots)$ of atoms in M^- such that no b_i is in L for any $i \geq 0$, otherwise it contradicts with L being a maximal loop. Again this sequence must contain a loop from which we can construct a maximal one of Γ_{M^-}. It contradicts with L is a terminating loop of Γ_{M^-} since there is a path $(a, b, \ldots)$ from L to the newly constructed maximal loop.

From (25), we have

$$\begin{aligned}
& l - \Sigma(\Gamma^-(S), M) > \Sigma(\Gamma^+(S), M^*) \\
& \Rightarrow l > \Sigma(\Gamma^-(S), M) + \Sigma(\Gamma^+(S), M \setminus M^-) \\
& \Rightarrow l > \Sigma(\Gamma^-(S), M) + \Sigma(\Gamma^+(S), M \setminus L) \ \text{(by (26))} \\
& \Rightarrow l > \Sigma(\Gamma^-(S), M) + \Sigma(\Gamma_2^+(\Gamma^+(S), L), M) \\
& \Rightarrow l > \Sigma(\Gamma^-(S), M) + \Sigma(\Gamma_2^+(S, L), M).
\end{aligned}$$

It follows that $M \not\models [l \leq S]_{|L}$, otherwise $l \leq \Sigma(\Gamma^-(S), M) + \Sigma(\Gamma_2^+(S, L), M)$. Thus $M \not\models [l \leq S \leq u]_{|L}$.

Note that for any rule $(C_0 \leftarrow C)$ of P with $a \in \mathcal{A}toms^+(C_0) \cap L$ but $(a \leftarrow C^M)$ does not belong to P^M, we have that $M \not\models C$ and then $M \not\models C_{|L}$. For any rule $(a \leftarrow C^M)$ in P^M, if $a \in L$ then this rule must belong to Γ_{M^-}. It implies that $M \not\models C_{|L}$. Consequently, $M \not\models LF(L, P)$, a contradiction. Thus $M = M^* = cl(\Gamma) = cl(P^M)$ and then M is an answer set of P. □

From Propositions 5 and 6, we immediately have a proof of Theorem 2.

Proof of Theorem 3 Let $I^f = \mathbb{R}(I) \setminus I^a$ and $V = \mathcal{A}toms(\mathbb{F}(P))$. It is evident that V splits $\mathbb{R}(P) \cup \mathbb{F}(P)$, $b_V(\mathbb{F}(P) \cup \mathbb{R}(P)) = \mathbb{F}(P)$ and I^f is an answer set of $\mathbb{F}(P)$.

In the following, we show that P^I is strongly equivalent to $\left[e_V(\mathbb{R}(P), I^f)\right]^{I^a}$. Recall that a program X is *strongly equivalent* to a program Y if and only if the two programs $X \cup Z$ and $Y \cup Z$ has the same answer sets for any program Z. For the sake of clarity, we consider the following simple cases for a rule r in P:

(1) r is of the form

$$l \leq \{p(f(a)) : w\} \leq u \leftarrow C$$

where C mentions no function symbols. Suppose the range of f is $\{c_1, \ldots, c_m\}$. It follows that the corresponding rules in $\mathbb{R}(P)$ obtained from r, denoted by $\mathbb{R}(\{r\})$, are

$$l \leq \{p(c_i) : w\} \leq u \leftarrow C, f_r(a, c_i) \qquad (1 \leq i \leq m)$$

Without loss of generality, suppose $f^I(a) = c_m$. Note that $e_V(\mathbb{R}(\{r\}), I^f)$ contains the following rules

$$l \leq \{p(c_i) : w\} \leq u \leftarrow C, 1 \leq \{\} \leq 1, \qquad (1 \leq i \leq m-1)$$

$$l \leq \{p(c_m) : w\} \leq u \leftarrow C, 0 \leq \{\} \leq 0.$$

It is clear that $\left[e_V(\mathbb{R}(\{r\}), I^f)\right]^{I^a}$ is strongly equivalent to $\{r\}^I$.

(2) r is of the form

$$l \leq \{f(a) = c_1 : w\} \leq u \leftarrow C$$

where C mentions no function symbols. Suppose the range of f is $\{c_1, \ldots, c_m\}$. It follows that the corresponding rules in $\mathbb{R}(P)$ are

$$l \leq \{\} \leq u \leftarrow C,\ f_r(a, c_i), \qquad (2 \leq i \leq m)$$
$$l - w \leq \{\} \leq u - w \leftarrow C,\ f_r(a, c_1)$$

It is easy to see that $\{r\}^I$ is strongly equivalent to $\left[e_V(\mathbb{R}(\{r\}), I^f)\right]^{I^a}$ as well.

(3) The other cases where a function symbol appears either in the body of rule r or in negative form can be proved similarly.

We can extend the above proof to the case where a weight constraint contains more than one element. Therefore, we conclude that P^I and $\left[e_V(\mathbb{R}(P), I^f)\right]^{I^a}$ are strongly equivalent to each other.

We are now in the position to prove the statement in the theorem:

I is an answer set of P

iff I^a is an answer set of P^I

iff I^a is an answer set of $[e_V(\mathbb{R}(P), I^f)]^{I^a}$

iff I^a is an answer set of $e_V(\mathbb{R}(P), I^f)$

iff $I^a \cup I^f$ is an answer set of $\mathbb{R}(P) \cup \mathbb{F}(P)$ (by Theorem 1)

iff $\mathbb{R}(I)$ is an answer set of $\mathbb{R}(P) \cup \mathbb{F}(P)$.

which completes the proof. □

In the following, we denote $\mathbb{F}(P) \cup \mathbb{R}(P)$ by $\Xi(P)$ for convenience. Since $\Xi(P)$ mentions no functions and equality, its loops and loop formulas are defined as usual in Section 2.3.

Lemma 8 *Let P be a weight constraint program with evaluable functions and I an interpretation for P. Then $\mathbb{R}(I) \models COMP(\Xi(P))$ if and only if $I \models COMP_f(P)$.*

Proof For any rule $(r : C_0 \leftarrow C_1)$ of P where $C_i = l_i \leq S_i \leq u_i (i = 0, 1)$, we firstly prove that

$$\mathbb{R}(I) \models \bigwedge_{(H \leftarrow Body) \in \mathbb{R}(\{r\})} \left(\bigwedge Body \supset H\right) \ \ iff\ \ I \models C_1 \supset C_0. \tag{27}$$

We assume that both S_0 and S_1 contain only one element for clarity, i.e., $S_i (i = 0, 1)$ are of the form $\{\alpha : w\}$ or $\{not\, \alpha : w\}$ where α is an atom or equality atom. Evidently, the case for the latter form can be similarly proved as for the former. Equation (27) holds trivially if S_0 and S_1 do not mention any function symbols at all.

Just for clarity, we consider only two simple cases for the above α, i.e., α is $p(f(c))$ and $f(c) = c_1$, where f is a function symbol and c and c_1 are two constants. Suppose f is of the type $\tau_0 \rightarrow \tau_1$, the domain of τ_1 is $\{c_1, \ldots, c_m\}$ according to P. Let us assume

that C_0 mentions a function symbol while C_1 does not. It is clear that $I \models C_1$ iff $\mathbb{R}(I) \models C_1$. Let us consider the following two cases:

(i) Let $S_0 = \{p(f(c)) : w\}$. Note that $\mathbb{R}(\{r\})$ consists of the rules

$$l_0 \leq \{p(c_i) : w\} \leq u_0 \leftarrow C_1,\ f_r(c, c_i) \qquad (1 \leq i \leq m).$$

To prove (27), it is sufficient to show

$$\mathbb{R}(I) \models \bigwedge_{1 \leq i \leq m} \left(f_r(c, c_i) \supset l_0 \leq \{p(c_i) : w\} \leq u_0\right) \mathit{iff}\ I \models l_0 \leq \{p(f(c)) : w\} \leq u_0.$$

Recall that I is an interpretation for P. Thus there exists a unique c' such that $f_r(c, c') \in \mathbb{R}(I)$ where $c' \in \{c_1, \ldots, c_m\}$. It follows that the above equation holds.

(ii) Let $S_0 = \{f(c) = c_1 : w\}$. We have that $\mathbb{R}(\{r\})$ consists of the rules:

$$l_0 \leq \{\} \leq u_0 \leftarrow C_1,\ f_r(c, c_i) \qquad (2 \leq i \leq m),$$
$$l_0 - w \leq \{\} \leq u_0 - w \leftarrow C_1,\ f_r(c, c_1).$$

To prove (27), it is sufficient to show $I \models l_0 \leq \{f(c) = c_1 : w\} \leq u_0$ iff

$$\mathbb{R}(I) \models (f_r(c, c_1) \supset l_0 \leq w \leq u_0) \wedge \bigwedge_{2 \leq i \leq m} (f_r(c, c_i) \supset l_0 \leq 0 \leq u_0).$$

The equation is easily proved by noticing that either $f^I(c) = c_1$ or $f^I(c) \neq c_1$.

Recall that $\mathcal{A}t(P)$ is the set of atoms that reside in P. It follows that $\mathcal{A}toms(\mathbb{R}(P)) \setminus \mathcal{A}toms(\mathbb{F}(P)) \subseteq \mathcal{A}t(P)$. It is clear that for any atom $p(\mathbf{c}) \in \mathcal{A}t(P) \setminus \mathcal{A}toms(\mathbb{R}(P))$, the formula (9) is equivalent to $\neg p(\mathbf{c})$. Thus we only need to show that, for any atom $p(\mathbf{c}) \in \mathcal{A}t(P) \cap \mathcal{A}toms(\mathbb{R}(P))$, the formula (5) is satisfied by $\mathbb{R}(I)$ if and only if the formula (9) is satisfied by I. Note that $\mathbb{R}(I) \models p(\mathbf{c})$ iff $I \models p(\mathbf{c})$. It suffices to prove that for any rule $(r : C_0 \leftarrow C_1)$ in P and any atom $p(\mathbf{c}) \in \mathcal{A}t(P) \cap \mathcal{A}toms(\mathbb{R}(P))$,

$$\mathbb{R}(I) \models \bigvee_{\substack{(H \leftarrow Body) \in \mathbb{R}(\{r\}) \\ and\ p(\mathbf{c}) \in \mathcal{A}toms^+(H)}} \left(\bigwedge Body\right)\ \mathit{iff}\ I \models C_1 \wedge \left(\bigvee_{p(\mathbf{t}) \in \mathcal{A}toms^+(C_0)} \mathbf{t} = \mathbf{c}\right). \tag{28}$$

Let $C_i = l_i \leq S_i \leq u_i (i = 0, 1)$. For clarity, we assume $\mathbf{t}$ is a single term t instead of a tuple of terms where t may be a function or a constant, and S_i $(i = 0, 1)$ contain only one element. Let f be a function of the type $\tau_0 \rightarrow \tau_1$ and the domain of τ_1 is $\{c_1, \ldots, c_m\}$ according to P. Let us consider the following cases:

(i) $S_0 = \{p(a) : w\}$ and $S_1 = \{q(f(b)) : w'\}$. If $a \neq c$ then (28) holds trivially. Suppose $a = c$. In this case, $\mathbb{R}(\{r\})$ consists of the following rules:

$$l_0 \leq \{p(a) : w\} \leq u_0 \leftarrow l_1 \leq \{q(c_i) : w'\} \leq u_1,\ f_r(b, c_i) \qquad (1 \leq i \leq m).$$

It is easy to see that

$$I \models l_1 \leq \{q(f(b)) : w'\} \leq u_1\ \mathit{iff}\ \mathbb{R}(I) \models \bigvee_{1 \leq i \leq m} l_1 \leq \{q(c_i) : w'\} \leq u_1 \wedge f_r(b, c_i).$$

It follows that (28) holds as well.

(ii) $S_0 = \{p(f(b)) : w\}$ and $S_1 = \{q(a) : w'\}$. Note that $\mathbb{R}(\{r\})$ consists of

$$l_0 \leq \{p(c_i) : w\} \leq u_0 \leftarrow l_1 \leq \{q(a) : w'\} \leq u_1,\ f_r(b, c_i) \qquad (1 \leq i \leq m).$$

Recall that $I \models C_1$ iff $\mathbb{R}(I) \models C_1$. If there is no c_i such that $c_i = c$ where $1 \leq i \leq m$ then (28) hold trivially. Suppose $c = c_1$. There is a unique rule r' in $\mathbb{R}(\{r\})$ such that $p(c) \in \mathcal{A}toms^+(Head(r'))$ whose body is $C_1 \wedge f_r(b, c_1)$. Evidently, $\mathbb{R}(I) \models f_r(b, c_1)$ iff $I \models f(b) = c$. It follows that (28) holds as well.

For the other cases that the element of S_1 mentions a negative literal, the proofs are quite similar to the above two cases. And we can similarly extend the above statements for the cases where a weight constraint contains more than one element. Thus we have completed the proof. □

Lemma 9 *Let P be a weight constraint program with evaluable functions and $L \subseteq \mathcal{A}\mathrm{t}(P)$. Then L is a loop of P if and only if L is a loop of $\mathbb{R}(P)$.*

Proof Recall that, when translating P into $\mathbb{R}(P)$, equality symbols were removed in the process of instantiation. But it is different from translating normal logic programs with functions, where we do not delete any rules at all. The reason is that we regard an equality literal $c = c'$ as the weight constraint $1 \leq \{c = c' : 1\} \leq 1$. Instantiating a rule containing $c = c'$ is just to replace $c = c'$ with $1 \leq \{\} \leq 1$ if $c \neq c'$, and $0 \leq \{\} \leq 0$ otherwise.

Note that $(p(\mathbf{c}), q(\mathbf{d}))$ is an edge of G_P iff there exists a rule $(C_0 \leftarrow C_1)$ in P such that, $p(\mathbf{t}) \in \mathcal{A}toms^+(C_0)$, $q(\mathbf{s}) \in \mathcal{A}toms^+(C_1)$ and there is a functional assignment θ with $\mathbf{t}\theta = \mathbf{c}$ and $\mathbf{s}\theta = \mathbf{d}$. It is obvious that there is at least one rule r' in $\mathbb{R}(P)$ such that $p(\mathbf{c}) \in Head(r')$ and $q(\mathbf{d}) \in Body(r')$. The other direction is obvious. Consequently, L is a loop of P if and only if L is a loop of $\mathbb{R}(P)$. □

Lemma 10 *Let P be a weight constraint program with evaluable functions and I an interpretation for P such that $I \models COMP_f(P)$ and L a loop of P. Then $o(I) \models LF_f(L, P)$ if and only if $\mathbb{R}(I) \models LF(L, \mathbb{R}(P))$.*

Proof It is sufficient to show that, for any rule $(r : C_0 \leftarrow C)$ of P,

$$o(I) \models \left(\bigvee_{\substack{p(\mathbf{c}) \in L \\ p(\mathbf{t}) \in \mathcal{A}toms^+(C_0)}} \mathbf{t} = \mathbf{c} \right) \wedge C_{\|L} \Leftrightarrow \mathbb{R}(I) \models \bigvee_{\substack{(H \leftarrow Body) \in \mathbb{R}(\{r\}) \\ L \cap H \neq \emptyset}} \left(\bigwedge_{C' \in Body} C'_{|L} \right). \tag{29}$$

Let $C_0 = l_0 \leq S_0 \leq u_0$ and $C = l \leq S \leq u$. Equation (29) holds trivially if there is no atom $p(\mathbf{t}) \in \mathcal{A}toms^+(C_0)$ and $p(\mathbf{c}) \in L$ such that $I \models (\mathbf{t} = \mathbf{c})$. It what follows, let us assume $p(c) \in L$ and $p(t) \in \mathcal{A}toms^+(C_0)$ such that $t^I = c$ for convenience. For clarity, we assume both S_0 and S contain only one element. Suppose f is a function of the

type $\tau_0 \to \tau_1$, domain of τ_1 is $\{c_1, \ldots, c_m\}$ in terms of P, and $f^I(a) = c_1$, $f^I(b) = c_m$. Let us consider the following four cases:

(i) $S_0 = \{p(c) : w\}$, $S = \{q(c') : w'\}$. It is clear that $\mathbb{R}(\{r\}) = \{r\}$. We have that $C_{\|L}$ is the following formula

$$l \leq \{q_L(c') : w'\} \wedge \{q(c') : w'\} \leq u \wedge \left[q_L(c') \equiv \left(q(c') \wedge \bigwedge_{q(c^*) \in L} c' \neq c^* \right) \right].$$

Note that $C_{|L}$ is the formula $l \leq \{\} \wedge \{q(c') : w'\} \leq u$ if $q(c') \in L$ and $l \leq \{q(c') : w'\} \wedge \{q(c') : w'\} \leq u$ otherwise. It is easy to see that $o(I) \models C_{\|L}$ iff $\mathbb{R}(I) \models C_{|L}$. Thus (29) holds.

(ii) $S_0 = \{p(f(a)) : w\}$ and $S = \{q(c') : w'\}$. We have that $\mathbb{R}(\{r\})$ consists of the rules

$$l_0 \leq \{p(c_i) : w\} \leq u_0 \leftarrow C,\ f_r(a, c_i) \qquad (1 \leq i \leq m).$$

In this case, $C_{\|L}$ and $C_{|L}$ are the same as those of the first case. It implies that $o(I) \models C_{\|L}$ iff $\mathbb{R}(I) \models C_{|L}$. We have that $\{f_r(a, c_i) | 1 \leq i \leq m\} \cap \mathbb{R}(I) = f_r(a, c_1)$ by $f^I(a) = c_1$. Recall that $p(c_1) \in L$. Consequently, (29) holds since $f_r(a, c_i)_{|L} \equiv f_r(a, c_i)$.

(iii) $S_0 = \{p(c) : w\}$ and $S = \{q(f(b)) : w'\}$. We have that $\mathbb{R}(\{r\})$ consists of the rules

$$C_0 \leftarrow l \leq \{q(c_i) : w'\} \leq u,\ f_r(b, c_i) \qquad (1 \leq i \leq m).$$

Note that $f^I(b) = c_m$. Under the interpretation $o(I)$, we have that $C_{\|L}$ is equivalent to

$$\begin{aligned} & l \leq \{q_L(c_m) : w'\} \wedge \{q(c_m) : w'\} \\ & \leq u \wedge \left[q_L(c_m) \equiv \left(q(c_m) \wedge \bigwedge_{q(c^*) \in L} c_m \neq c^* \right) \right] \end{aligned}$$

which is equivalent to, under the interpretation $o(I)$, $l \leq \{q(c_m) : w'\} \wedge \{q(c_m) : w'\}$ if $q(c_m) \notin L$ and $l \leq \{\} \wedge \{q(c_m) : w'\} \leq u$ otherwise.
Under the interpretation $\mathbb{R}(I)$, the formula $\bigvee_{1 \leq i \leq m} [l \leq \{q(c_i) : w'\} \leq u]_{|L} \wedge f_r(b, c_i)$ is equivalent to $l \leq \{q(c_m) : w'\} \leq u]_{|L} \wedge f_r(b, c_m)_{|L}$ since $\{f_r(b, c_i) | 1 \leq i \leq m\} \cap \mathbb{R}(I) = f_r(b, c_m)$, which is further equivalent to $l \leq \{q(c_m) : w'\} \wedge \{q(c_m) : w'\} \leq u$ if $q(c_m) \notin L$, and $l \leq \{\} \wedge \{q(c_m) : w'\} \leq u$ otherwise. It follows that (29) holds.

(iv) $S_0 = \{p(f(a)) : w\}$ and $S = \{q(f(b)) : w'\}$. We have that $\mathbb{R}(\{r\})$ consists of

$$\begin{aligned} & l_0 \leq \{p(c_i) : w\} \leq u_0 \leftarrow l \\ & \quad \leq \{q(c_j) : w'\} \leq u,\ f_r(a, c_i),\ f_r(b, c_j) \qquad (1 \leq i, j \leq m). \end{aligned}$$

Note that $C_{\|L}$ is the same as that of the third case. Since $\{f_r(a, c_i) | 1 \leq i \leq m\} \cap \mathbb{R}(I) = f_r(a, c_1)$ and $\{f_r(b, c_i) | 1 \leq i \leq m\} \cap \mathbb{R}(I) = f_r(b, c_m)$, it implies that, under the interpretation $\mathbb{R}(I)$, the right hand of (29) is equivalent to

$$\left[l \leq \{q(c_m) : w'\} \leq u \right]_{|L} \wedge f_r(a, c_1) \wedge f_r(b, c_m)$$

which is further equivalent to $l \leq \{q(c_m) : w'\} \wedge \{q(c_m) : w'\} \leq u$ if $q(c_m) \notin L$ and $l \leq \{\} \wedge \{q(c_m) : w'\} \leq u$ otherwise. It follows that (29) holds.

We can similarly prove the cases where S mentions a negative literal and we can similarly extend the above statements for the cases where S_0 and S_1 contains more than one elements. Thus the proof is completed. □

Proof of Theorem 4 I is an answer set of P

iff $\mathbb{R}(I)$ is an answer set of $\Xi(P)$ (Theorem 3)

iff $\mathbb{R}(I)$ is an answer set of $COMP(\Xi(P)) \cup LF(\Xi(P))$ (Theorem 2)

iff $o(I)$ is an answer set of $COMP_f(P) \cup LF_f(P)$ (Lemmas 8–10).

□

Lemma 11 *Let ψ be a ground formula without function symbols occurring in ψ as an argument and I an interpretation. Then $I \models \psi$ iff $v(I)$ is a solution of $c(\psi)$.*

Proof It is clear by induction on structures of formulas. □

Proof of Theorem 5 I is an answer set of P

iff $I \models COMP_f(P) \cup LF_f(P)$ (Theorem 4)

iff $v(I)$ is a solution of $c(COMP_f(P) \cup LF_f(P))$ (Lemma 11)

iff $v(I)$ is a solution of $\mathcal{R}(P)$.

□

Let us extend the following notations to extended weight constraints. Given an extended weight constraint $l \leq S \leq u$ and I an interpretation, where $S = \{c_1 : w_1, \ldots, c_n : w_n\}$, we define:

- $\Gamma_2^+(S, I) = \{p(\mathbf{t}) : w | I \not\models p(\mathbf{t})$ *and* $(p(\mathbf{t}) : w) \in S\}$;
- $\Gamma^+(S) = \{p(\mathbf{t}) : w | (p(\mathbf{t}) : w) \in S\}$;
- $\Gamma^-(S) = S \setminus \Gamma^+(S)$;

where p is a predicate symbol and $\mathbf{t}$ a tuple of terms.

Proof of Proposition 2 We assume P is free of functions in arguments (if not we transform it to be so). Let Γ be the set of rules in P^I such that both their head and body are satisfied by I. Note that I is a supported model of P by $I \models COMP_f(P)$. It implies that there exists a rule $(a \leftarrow [l \leq S]^I)$ in Γ such that $I \models [l \leq S]^I$ for any atom $a \in I^a$.

Let $I^* = cl(P^I)$. Clearly, $I^* \subseteq I$ and $I^* = cl(\Gamma)$ since $I \models P^I$. Clearly $I^- \neq \emptyset$. Let Γ_{I^-} be the set of rules of Γ whose head is an atom in I^-. We show that the positive dependency graph of Γ_{I^-} has at least one loop. Let a be an arbitrary atom in I^- and $(r' : a \leftarrow l' \leq S')$ an arbitrary rule in Γ_{I^-} such that r' is obtained from $(r : C \leftarrow l \leq S \leq u)$ of P by the reduction relative to I where $a \in \mathcal{A}toms^+(C) \cap I$, $I \models S \leq u$,

$l' = l - \Sigma(\Gamma^-(S), I)$, and $S' = \Gamma^+(S)$. Note that $a \notin I^*$ implies that $I^* \not\models [l \leq S]^I$. Together with $I \models [l \leq S]^I$, we have

$$l' \leq \Sigma(S', I^*) + \Sigma(S', I^-) \text{ and } l' > \Sigma(S', I^*).$$

It follows that $I^- \cap \mathcal{A}toms^+(S') \neq \emptyset$. Thus we can construct a sequence $(a_0, a_1, \ldots, a_i, \ldots)$ of atoms such that $a_i \in I^-$ and a_i depends on a_{i+1} in the sense that there is a rule $(a_i \leftarrow C)$ in Γ_{I^-} such that $a_{i+1} \in \mathcal{A}toms^+(C) \cap I^-$. Since Γ_{I^-} is finite and the sequence mentions only finite number of atoms, the above constructed sequence must contain a loop. It also implies that Γ_{I^-} has at least one terminating loop.

Suppose L is a terminating loop of Γ_{I^-} and $a \in L$ where a is the atom in the head of the above rule r'. We show that

$$I^- \cap \mathcal{A}toms^+(l' \leq S') \subseteq L. \tag{30}$$

Suppose this is not true. Then there is some $b \in I^- \cap \mathcal{A}toms^+(l' \leq S')$ but $b \notin L$. In terms of the above construction, we can construct a sequence $(b_0 = (b), \ldots, b_i, \ldots)$ of atoms in I^- such that no b_i is in L for any $i \geq 0$, otherwise it contradicts with L being a maximal loop. Again this sequence must contain a loop from which we can construct a terminating one of Γ_{M^-}. It contradicts the fact that L is a terminating loop of Γ_{M^-} since there is a path $(a, b, \ldots)$ from L to the newly constructed maximal loop.

Recall that $l' > \Sigma(S', I^*)$ and $S' = \Gamma^+(S)$. It follows that

$$\begin{aligned}
& l - \Sigma(\Gamma^-(S), I) > \Sigma(\Gamma^+(S), I^*) \\
& \quad \Rightarrow l > \Sigma(\Gamma^-(S), I) + \Sigma(\Gamma^+(S), I^a \setminus I^-) \\
& \quad \Rightarrow l > \Sigma(\Gamma^-(S), I) + \Sigma(\Gamma^+(S), I^a \setminus L) \text{ (by (30))} \\
& \quad \Rightarrow l > \Sigma(\Gamma^-(S), I) + \Sigma(\Gamma_2^+(\Gamma^+(S), L), I^a) \\
& \quad \Rightarrow l > \Sigma(\Gamma^-(S), I) + \Sigma(\Gamma_2^+(S, L), I^a).
\end{aligned}$$

It follows that $I \not\models [l \leq S]_{\|L}$, otherwise $l \leq \Sigma(\Gamma^-(S), I) + \Sigma(\Gamma_2^+(S, L), I^a)$. Thus $I \not\models [l \leq S \leq u]_{\|L}$.

Note that for any rule $(C_0 \leftarrow C)$ of P with $a \in \mathcal{A}toms^+(C_0) \cap L$ and $(a \leftarrow C^I)$ does not belong to P^I, we have that $I \not\models C$ and then $I \not\models C_{\|L}$. For any rule $(a \leftarrow C^I)$ in P^I, if $a \in L$ then this rule must belong to Γ_{I^-}. This implies that $I \not\models C_{\|L}$. Consequently, $I \not\models LF_f(L, P)$. □

Proof of Proposition 3 Algorithm 1 is clearly terminating due to the fact that P has only a finite number of loops. Its soundness is obvious, while its completeness follows from Theorem 4. □

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, New York (2003)
2. Baselice, S., Bonatti, P.A., Criscuolo, G.: On finitely recursive programs. In: Proceedings of the Twenty Third International Conference on Logic Programming, pp. 89–103, Porto, Portugal. Springer, New York (2007)

3. Bessiere, C., Katsirelos, G., Narodytska, N., Quimper, C.-G., Walsh, T.: Decompositions of all different, global cardinality and related constraints. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, pp. 419–424. Pasadena, California (2009)
4. Bonatti, P.A.: Reasoning with infinite stable models. Artif. Intell. **156**(1), 75–111 (2004)
5. Bordeaux, L., Hamadi, Y., Zhang, L.: Propositional satisfiability and constraint programming: a comparative survey. ACM Comput. Surv. **38**(4), 12:1–12:54 (2006)
6. Cabalar, P.: A functional action language front-end. In: The Third International Workshop on Answer Set Programming: Advances in Theory and Implementation. http://www.dc.fi.udc.es/~cabalar/asp05_C.pdf (2005)
7. Cabalar, P., Lorenzo, D.: Logic programs with functions and default values. In: Proceedings of the Ninth European Conference on Logics in Artificial Intelligence, pp. 294–306, Lisbon, Portugal. Springer, New York (2004)
8. Cadoli, M., Schaerf, M.: A survey of complexity results for nonmonotonic logics. Log. Program. **17**(2/3&4), 127–160 (1993)
9. Calimeri, F., Cozza, S., Ianni, G., Leone, N.: Computable functions in ASP: theory and implementation. In: Logic Programming, 24th International Conference, Udine, Italy, vol. 5366 of Lecture Notes in Computer Science, pp. 407–424. Springer, New York (2008)
10. Chen, Y., Lin, F., Wang, Y., Zhang, M.: First-order loop formulas for normal logic programs. In: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), Lake District of the United Kingdom, pp. 298–307. AAAI Press, Menlo Park (2006)
11. Dovier, A., Formisano, A., Pontelli, E.: An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems. J. Exp. Theoret. Artif. Intell. **21**(2), 79–121 (2009)
12. Drescher, C., Walsh, T.: A translational approach to constraint answer set solving. Theory Pract. Log. Program. **10**(4–6), 465–480 (2010)
13. Eiter, T., Faber, W., Mushthofa, M.: Space efficient evaluation of asp programs with bounded predicate arities. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, Atlanta, Georgia, USA, pp. 303–308. AAAI Press, Menlo Park (2010)
14. Eiter, T., Simkus, M.: FDNC: decidable nonmonotonic disjunctive logic programs with function symbols. ACM Trans. Comput. Log. **11**(2), 14:1–14:50 (2010)
15. Erdem, E., Lin, F., Schaub, T. (eds.): Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, 14–18 September 2009. Proceedings, vol. 5753 of Lecture Notes in Computer Science. Springer, New York (2009)
16. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 372–379. Hyderabad, India (2007)
17. Ferraris, P., Lifschitz, V.: Weight constraints as nested expressions. Theory Pract. Log. Program. **5**(1–2), 45–74 (2005)
18. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set enumeration. In: Logic Programming and Nonmonotonic Reasoning, 9th International Conference, Tempe, AZ, USA, LPNMR 2007, vol. 4483 of Lecture Notes in Computer Science, pp. 136–148. Springer, New York (2007)
19. Gebser, M., Ostrowski, M., Schaub, T.: Constraint answer set solving. In: Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, vol. 5649 of Lecture Notes in Computer Science, pp. 235–249. Springer, New York (2009)
20. Gelfond, M., Leone, N.: Logic programming and knowledge representation—the a-prolog perspective. Artif. Intell. **138**(1–2), 3–38 (2002)
21. Järvisalo, M., Oikarinen, E., Janhunen, T., Niemelä, I.: A module-based framework for multi-language constraint modeling. In: Logic Programming and Nonmonotonic Reasoning, 10th International Conference, vol. 5753 of Lecture Notes in Computer Science, pp. 155–168. Springer, New York (2009)
22. Lee, J.: A model-theoretic counterpart of loop formulas. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, pp. 503–508. Professional Book Center, London (2005)
23. Lee, J., Lifschitz, V.: Loop formulas for disjunctive logic programs. In: Proceedings of the Nineteenth International Conference on Logic Programming, Mumbai, India, vol. 2916 of Lecture Notes in Computer Science, pp. 451–465. Springer, New York (2003)
24. Lifschitz, V., Razborov, A.A.: Why are there so many loop formulas? ACM Trans. Comput. Log. **7**(2), 261–268 (2006)

25. Lifschitz, V., Turner, H.: Splitting a logic program. In: Proceedings of the Eleventh International Conference on Logic Programming, Santa Margherita Ligure, Italy, pp. 23–37. MIT Press, Cambridge (1994)
26. Lin, F., Wang, Y.: Answer set programming with functions. In: Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning, Sydney, Australia, pp. 454–464. AAAI Press, Menlo Park (2008)
27. Lin, F., Zhao, Y.: ASSAT: computing answer sets of a logic program by SAT solvers. Artif. Intell. **157**(1–2), 115–137 (2004)
28. Liu, G., You, J.-H.: Level mapping induced loop formulas for weight constraint and aggregate logic programs. Fundam. Inform. **101**(3), 237–255 (2010)
29. Liu, L., Truszczynski, M.: Properties and applications of programs with monotone and convex constraints. J. Artif. Intell. Res. **27**, 299–334 (2006)
30. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, New York (1987)
31. Wiktor Marek, V., Truszczynski, M.: Stable models and an alternative logic programming paradigm. In: Apt, K.R., Marek, V.W., Truszczynski, M., Warren, D.S. (eds.) The Logic Programming Paradigm: A 25-Year Perspective, pp. 375–398. Springer, Berlin (1999)
32. Marek, V., Niemelä, I., Truszczyński, M.: Logic programs with monotone abstract constraint atoms. Theory Pract. Log. Program. **8**(2), 167–199 (2008)
33. Mellarkod, V.S., Gelfond, M., Zhang, Y.: Integrating answer set programming and constraint logic programming. Ann. Math. Artif. Intell. **53**(1–4), 251–287 (2008)
34. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. **25**(3–4), 241–273 (1999)
35. Niemelä, I., Simons, P.: Extending the Smodels System with Cardinality and Weight Constraints, Chapter 21, pp. 491–521. Kluwer, Dordrecht (2000)
36. Oikarinen, E., Janhunen, T.: Achieving compositionality of the stable model semantics for smodels programs. Theory Pract. Log. Program. (TPLP) **8**(5–6), 717–761 (2008)
37. Papadimitriou, C.H.: Computatioinal Complexity. Addison Wesley, Reading (1994)
38. Pearce, D., Valverde, A.: Towards a first order equilibrium logic for nonmonotonic reasoning. In: Logics in Artificial Intelligence, 9th European Conference, Lisbon, Portugal, vol. 3229 of Lecture Notes in Computer Science, pp. 147–160. Springer, New York (2004)
39. Rina, D.: Constraint Processing. Morgan Kaufmann, San Mateo (2003)
40. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artif. Intell. **138**(1–2), 181–234 (2002)
41. Syrjänen, T.: Omega-restricted logic programs. In: Proceedings of the Sixth International Conference on Logic Programming and Nonmonotonic Reasoning, Vienna, Austria, pp. 267–279. Springer, New York (2001)
42. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)
43. Wang, Y., You, J.-H., Yuan, L.-Y., Zhang, M.: Weight constraint programs with functions. In: Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, vol. 5753 of Lecture Notes in Computer Science, pp. 329–341. Springer, New York (2009)
44. You, J.-H., Liu, G.: Loop formulas for logic programs with arbitrary constraint atoms. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008), Chicago, Illinois, USA, pp. 584–589. AAAI Press, Menlo Park (2008)