TLP **13** (1): 1–31, 2013. © Cambridge University Press 2011
doi:10.1017/S147106841100038X First published online 30 June 2011

Relating weight constraint and aggregate programs: Semantics and representation

GUOHUA LIU and JIA-HUAI YOU
University of Alberta, Edmonton T6G 2R3, Canada
(*e-mail:* {guohua, you}@cs.ualberta.ca)

submitted 21 June 2010; revised 2 January 2011; accepted 9 May 2011

Abstract

Weight constraint and aggregate programs are among the most widely used logic programs with constraints. In this paper, we relate the semantics of these two classes of programs, namely, the stable model semantics for weight constraint programs and the answer set semantics based on conditional satisfaction for aggregate programs. Both classes of programs are instances of logic programs with constraints, and in particular, the answer set semantics for aggregate programs can be applied to weight constraint programs. We show that the two semantics are closely related. First, we show that for a broad class of weight constraint programs, called *strongly satisfiable programs*, the two semantics coincide. When they disagree, a stable model admitted by the stable model semantics may be circularly justified. We show that the gap between the two semantics can be closed by transforming a weight constraint program to a strongly satisfiable one so that no circular models may be generated under the current implementation of the stable model semantics. We further demonstrate the close relationship between the two semantics by formulating a transformation from weight constraint programs to logic programs with nested expressions, which preserves the answer set semantics. Our study on the semantics leads to an investigation of a methodological issue, namely, the possibility of compact representation of aggregate programs by weight constraint programs. We show that almost all standard aggregates can be encoded by weight constraints compactly. This makes it possible to compute the answer sets of aggregate programs using the answer set programming solvers for weight constraint programs. This approach is compared experimentally with the ones where aggregates are handled more explicitly, which show that the weight constraint encoding of aggregates enables a competitive approach to answer set computation for aggregate programs.

KEYWORDS: stable model, weight constraint, aggregates, logic programs with constraints

1 Introduction

Answer set programming (ASP), namely, logic programming under the answer set semantics (Gelfond and Lifschitz 1988; Niemelä 1999), is a constraint programming paradigm, which has been successfully deployed in many applications (Balduccini *et al.* 2001; Wu *et al.* 2007; Caldiran *et al.* 2009; Delgrande *et al.* 2009; Erdem *et al.* 2009; Ielpa *et al.* 2009; Oetsch *et al.* 2009). Recently, ASP was extended to include constraints to facilitate reasoning with sets of atoms. These constraints include

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

weight constraints (Simons *et al.* 2002), aggregates (Faber *et al.* 2004; Ferraris 2005; Pelov *et al.* 2007; Son and Pontelli 2007), and abstract constraints (Marek and Remmel 2004; Liu and Truszczyński 2006; Marek *et al.* 2007; Son *et al.* 2007; You *et al.* 2007; Shen *et al.* 2009; Liu *et al.* 2010). Among them, weight constraints and aggregates are the most widely used constraints in practice. In this paper, logic programs with weight constraints and aggregates will be referred to as *weight constraint* and *aggregate programs*, respectively.

The semantics of weight constraint programs, called the *stable model semantics*, is well established and implemented in a number of ASP solvers (Simons *et al.* 2002; Giunchiglia *et al.* 2006; Gebser *et al.* 2007a). Especially, the results of the ASP solver competitions (Gebser *et al.* 2007b; Denecker *et al.* 2009) show that CLASP is an efficient solver that implements this semantics.

For aggregate programs, various semantics have been proposed (Faber *et al.* 2004; Ferraris 2005; Pelov *et al.* 2007; Son and Pontelli 2007). The one proposed in Pelov *et al.* (2007) (previously in Denecker *et al.* 2001; Pelov *et al.* 2004), called the *ultimate stable semantics*, is based on an iterative construction on partial interpretations. The same semantics is reformulated by Son and Pontelli (2007) and Son *et al.* (2007) and extended to logic programs with arbitrary abstract constraint atoms, which embodies a key concept called *conditional satisfaction*. Since this reformulation is conceptually simpler, as it does not resort to three-valued logic, in this paper, we call this semantics *conditional satisfaction-based*. Among the semantics for aggregate programs, this semantics is known to be the most conservative, in the sense that any answer set under this semantics is an answer set under others, but the reverse may not hold. The relationships of these semantics have been studied in Son *et al.* (2007), Shen *et al.* (2009), and Liu *et al.* (2010). In this paper, we refer to the semantics based on conditional satisfaction as the *answer set semantics*.[1]

Despite the fact that weight constraint and aggregate programs are among the most popular classes of programs in practice, the relationship among them has not been fully studied, both in semantics and in representation.

In this paper, we study the relationship between the stable model semantics and the answer set semantics. We show that for a broad class of weight constraint programs, called *strongly satisfiable programs*, the stable model semantics agrees with the answer set semantics. For example, weight constraint programs where weight constraints are upper bound free are all strongly satisfiable. This result is useful in that we are now sure that the known properties of the answer sets also hold for these programs. One important property is that any answer set is a *well-supported model* (Son *et al.* 2007), ensuring that any conclusion must be supported by a noncircular justification in the sense of Fages (1994).

Our study further reveals that for weight constraint programs where the stable model and answer set semantics disagree, stable models may be circularly justified. We then show that the gap between the two can be closed by a transformation, which translates an arbitrary weight constraint program to a strongly satisfiable program

[1] In the literature, *stable model* and *answer set* are usually interchangeable for logic programs without the "classic negation" (see Gelfond 2008). In this paper, we use them to refer to different semantics.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

so that the answer sets of the original program are exactly the stable models of the translated program.

We further demonstrate the precise difference between the two semantics using a more general logic programming framework, logic programs with nested expressions. We propose yet another transformation from weight constraint programs to logic programs with nested expressions, which preserves the answer set semantics. We compare this transformation to the one given in Ferraris and Lifschitz (2005), which is faithful to the stable model semantics. Interestingly, the difference is small but subtle: given a weight constraint $l[S]u$, where l and u are lower and upper bounds, respectively, and $[S]$ expresses a collection of literals with weights, while in our transformation the satisfaction of the upper bound is interpreted directly as "less than or equal to u," in Ferraris and Liftschitz (2005), the interpretation is by negation-as-failure "not greater than u."

The observation that the gap between the answer set and the stable model semantics can be closed by a transformation leads to an approach for computing answer sets of aggregate programs using the ASP solvers that implement the stable models semantics of weight constraint programs. We propose such an approach where aggregate programs are encoded compactly as weight constraint programs and their answer sets are computed using a stable model solver. We conducted a series of experiments to evaluate this approach. The results suggest that representing aggregates by weight constraints is a promising alternative to the explicit handling of aggregates in logic programs.

Besides efficiency, another advantage is at the system level: an aggregate language can be built on top of a stable models solver with a simple front end that essentially transforms standard aggregates to weight constraints in linear time. This is in contrast with the state of the art in handling aggregates in ASP, which typically requires an explicit implementation for each aggregate.

The paper is organized as follows. The next section gives preliminary definitions. In Section 3, we relate the stable model semantics with the answer set semantics. We first establish a sufficient condition for the two to coincide and then discuss their differences. In Section 4, we present a transformation to close the gap between the two semantics, followed by Section 5 where we show how to represent aggregate programs by weight constraint programs. Further in Section 6, to pinpoint the precise difference between the stable model semantics and the answer set semantics for weight constraint programs, by proposing a transformation from weight constraint programs to logic programs with nested expressions, which preserves the answer set semantics, and comparing this with that of Ferraris and Lifschitz (2005). We implemented a prototype system called ALPARSE , and in Section 7, we report some experimental results. Section 8 concludes the paper.

A preliminary version of this paper has appeared as Liu and You (2008). The main extensions here include the following: (i) Section 6, where we propose a transformation from weight constraint programs to logic programs with nested expressions, which preserves the answer set semantics—this transformation shows exactly what makes the answer set semantics differ from the stable model semantics; (ii) Section 7, where experiments are expanded including the benchmarks for

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

aggregate programs used in the 2007 ASP Solver Competition (Gebser *et al.* 2007b); and (iii) the proofs of all the theorems and lemmas.

2 Preliminaries

Throughout the paper, we assume a fixed propositional language with a countable set of propositional atoms.

2.1 Stable model semantics for weight constraint programs

A *weight constraint* is of the form

$$l\,[a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathtt{not}\, b_1 = w_{b_1}, \ldots, \mathtt{not}\, b_m = w_{b_m}]\,u, \tag{1}$$

where each a_i, b_j is an atom, and each atom and not-atom (negated atom) is associated with a *weight*. Atoms and not-atoms are also called *literals* (the latter may be emphasized as *negative* literals).The literal set of a weight constraint W, denoted $lit(W)$, is the set of literals occurring in W. The numbers l and u are the *lower* and *upper bounds*, respectively. The weights and bounds are real numbers. Either of the bounds may be omitted in which case the missing lower bound is taken to be $-\infty$ and the missing upper bound by ∞.

A set of atoms M satisfies a weight constraint W of the form (1), denoted $M \models W$, if (and only if) $l \leqslant w(W, M) \leqslant u$, where

$$w(W, M) = \sum_{a_i \in M} w_{a_i} + \sum_{b_i \notin M} w_{b_i}. \tag{2}$$

M satisfies a set of weight constraints Π if $M \models W$ for every $W \in \Pi$.

A weight constraint W is *monotone* if for any two sets R and S, if $R \models W$ and $R \subseteq S$, then $S \models W$; otherwise, W is *nonmonotone*. There are some special classes of nonmonotone weight constraints. W is *antimonotone* if for any R and S, $S \models W$ and $R \subseteq S$ imply $R \models W$; W is *convex* if for any R and S such that $R \subseteq S$, if $R \models W$ and $S \models W$ then for any I such that $R \subseteq I \subseteq S$ we have $I \models W$.

A *weight constraint program* is a finite set of *weight rules* of the form

$$W_0 \leftarrow W_1, \ldots, W_n, \tag{3}$$

where each W_i is a weight constraint. Given a (weight) rule r of the above form, we will use $hd(r)$ to denote W_0 and $bd(r)$ the conjunction of the weight constraints in the body of the rule.

We will use $At(P)$ to denote the set of the atoms appearing in a program P.

Weight constraint programs are often called *lparse programs*, which generally refer to the kind of nonground, function-free logic programs one can write based on the LPARSE syntax. These programs are grounded before calling an ASP solver. In this paper, for the theoretical study, we always assume that a given weight constraint program is ground.

Given a weight constraint program P, if the head of each rule in P is of the form $1\ [a = 1]\ 1$, where a is an atom, then P is said to be *basic*. If, in addition, all the

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

weight constraints in the bodies of rules in P are of the form $1\ [l = 1]\ 1$, where l is a literal, then we have a normal program. We will simply write a weight constraint $1\ [l = 1]\ 1$ as l, since they are equivalent in terms of satisfaction.

As pointed out by Simons *et al.* (2002), negative weights and negative literals are closely related in that they can replace each other and that one is inessential when the other is available. Negative weights can be eliminated by applying the following transformation (Simons *et al.* 2002): given a weight constraint W of the form (1), if $w_{a_i} < 0$, then replace $a_i = w_{a_i}$ with $\texttt{not}a_i = |w_{a_i}|$ and increase the lower bound to $l + |w_{a_i}|$ and the upper bound to $u + |w_{a_i}|$; if $w_{b_i} < 0$, then replace $\texttt{not}b_i = w_{b_i}$ with $b_i = |w_{b_i}|$ and increase the lower bound to $l + |w_{b_i}|$ and the upper bound to $u + |w_{b_i}|$.

For instance, the weight constraint

$$-1\ [a_1 = -1, a_2 = 2, \texttt{not}b_1 = 1, \texttt{not}b_2 = -2]\ 1$$

can be transformed to

$$2\ [\texttt{not}a_1 = 1, a_2 = 2, \texttt{not}b_1 = 1, b_2 = 2]\ 4.$$

Note that this transformation is satisfaction-preserving, in the sense that for any weight constraint W and set of atoms M, $M \models W$ iff $M \models W'$, where W' is obtained by applying the transformation.

From now on, we assume that negative weights are always eliminated by the above transformation.

The stable models of weight constraint programs are defined using the reduct of weight constraints, which is defined as follows: the *reduct* of a weight constraint W of the form (1) w.r.t. a set of atoms M, denoted by W^M, is the weight constraint

$$l'\ [a_1 = w_{a_1}, \ldots, a_n = w_{a_n}], \tag{4}$$

where $l' = l - \sum_{b_i \notin M} w_{b_i}$.

Let P be a weight constraint program and M be a set of atoms. The reduct of P w.r.t. M, denoted P^M, is defined by

$$\begin{aligned} P^M \quad = \quad & \{p \leftarrow W_1^M, \ldots, W_n^M \mid W_0 \leftarrow W_1, \ldots W_n \in P, \\ & p \in lit(W_0) \cap M \text{ and } w(W_i, M) \leqslant u, \text{ for all } \ i \geqslant 1\}. \end{aligned} \tag{5}$$

Definition 1 (Simons *et al.* 2002)
Let P be a weight constraint program and $M \subseteq At(P)$. M is an *stable model* of P iff the following two conditions hold:

(1) $M \models P$,
(2) M is the deductive closure of P^M.

Note that P^M is a monotone basic weight constraint program. The deductive closure of such a program can be constructed using the operator T_P defined in Liu and Truszczyński (2006). Let P be a monotone basic weight constraint program. The operator T_P is defined as

$$T_P(S) = \{h \mid \exists r \in P \text{ of the form } h \leftarrow bd(r) \text{ and } S \models bd(r)\}. \tag{6}$$

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

We note that, for a monotone program P, the operator T_P is monotone with respect to S. Then, we have the lemma below.

Lemma 1
Given a weight constraint program P, a set of atoms M is a stable model of P iff $M \models P$ and $M = T^{\infty}_{P^M}(\emptyset)$.

Proof
Let P be a weight constraint program and M a set of atoms. P^M is a monotone program. The deductive closure of P^M is the least fixpoint of $T_{P^M}(\emptyset)$. Then, the lemma follows from Definition 1. □

2.2 Answer set semantics for aggregate programs

Following Son and Pontelli (2007), we define the syntax and semantics for aggregate programs below.

An aggregate is a constraint on a set of atoms taking the form

$$aggr(\{X \mid p(X)\}) \text{ op } Result, \tag{7}$$

where *aggr* is an *aggregate function*. The standard aggregate functions are those in {SUM, COUNT, AVG, MAX, MIN}. The set $\{X \mid p(X)\}$ is called an *intensional set*, where p is a predicate, and X is a variable, which takes value from a set $D(X) = \{a_1, \ldots, a_n\}$, called the *variable domain*. The relational operator *op* is from $\{=, \neq, <, >, \leqslant, \geqslant\}$ and *Result* is either a variable or a numeric constant.

The *domain* of an aggregate A, denoted $Dom(A)$, is the set of atoms $\{p(a) \mid a \in D(X)\}$. The size of an aggregate is $|Dom(A)|$.

For an aggregate A, the intensional set $\{X \mid p(X)\}$, the variable domain $D(X)$, and the domain of an aggregate $Dom(A)$ can also be a multiset, which may contain duplicate members.

Let M be a set or multiset of atoms. M is a *model* of (satisfies) an aggregate A, denoted $M \models A$, if $aggr(\{a \mid p(a) \in M \cap Dom(A)\})$ op *Result* holds; otherwise, M is not a model of (does not satisfy) A, denoted $M \not\models A$.

For instance, consider the aggregate $A = SUM(\{X|p(X)\}) \geqslant 2$, where $D(X) = \{-1, 1, 1, 2\}$. For the sets $M_1 = \{p(2)\}$ and $M_2 = \{p(-1), p(1)\}$, we have $M_1 \models A$ and $M_2 \not\models A$. For the multiset $M_3 = \{p(1), p(1)\}$, we have $M_3 \models A$.

An aggregate program is a set of rules of the form

$$h \leftarrow A_1, \ldots, A_n, \tag{8}$$

where h is an atom and $A_1, \ldots, A_n$ are aggregates.[2] For a rule r of the form (8), we use $hd(r)$ and $bd(r)$ to denote h and the set $\{A_1, \ldots, A_n\}$, respectively.

The definition of *answer set* of aggregate programs is based on the notion of *conditional satisfaction*.

[2] In general, the A_i's could also be atoms or negative atoms. Here, we focus on aggregates. The results can be extended to the general case, where the atoms and negative atoms are treated exactly the same as that in normal logic programs (Son and Pontelli 2007).

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Definition 2
Let A be an aggregate and R and S be two sets of atoms. R conditionally satisfies A, w.r.t. S, denoted $R \models_S A$, iff $R \models A$ and for every set I such that $R \cap Dom(A) \subseteq I \subseteq S \cap Dom(A)$, $I \models A$.

R conditionally satisfies a set of aggregates Π w.r.t. S, if $R \models_S A$ for every $A \in \Pi$.

Given two sets R and S, and an aggregate program P, the operator $K_P(R, S)$ is defined as

$$K_P(R, S) = \{hd(r) \mid \exists r \in P,\ R \models_S bd(r)\}.$$

K_P is monotone w.r.t. its first argument, given that the second argument is fixed. Following Son and Pontelli (2007), given a set of atoms M, the least fixpoint of K_P w.r.t M is defined as $K_P^\infty(\emptyset, M)$, where $K_P^0(\emptyset, M) = \emptyset$ and $K_P^{i+1}(\emptyset, M) = K_P(K_P^i(\emptyset, M), M)$, for all $i \geqslant 0$.

Definition 3 (Son and Pontelli 2007)
Let P be an aggregate program and M be a set of atoms. M is an *answer set* of P iff M is a model of P and $M = K_P^\infty(\emptyset, M)$.

2.3 Answer sets of weight constraint programs

To present our results, it is notationally important to lift the concepts of conditional satisfaction and answer set to weight constraints. Given a weight constraint W, the *domain of* W, denoted $Dom(W)$, is the set $\{a \mid a \in lit(W)$ or $\texttt{not}\, a \in lit(W)\}$. Let W be a weight constraint and R and S be two sets of atoms. R conditionally satisfies W, w.r.t. S, denoted $R \models_S W$, if for all I such that $R \cap Dom(W) \subseteq I \subset S \cap Dom(W)$, we have $I \models W$.

First, the answer sets of a basic weight constraint program are defined using the concept of conditional satisfaction.

Definition 4
Let P be a basic weight constraint program and M be a set of atoms. M is an answer set of P iff M is a model of P and $M = K_P^\infty(\emptyset, M)$.

Then, following Son *et al.* (2007), the answer sets of a general weight constraint program are defined as the answer sets of its *instances*.

Let P be a weight constraint program, r be a rule in P of the form (3), and M be a set of atoms. The *instance of* r *w.r.t.* M is

$$inst(r, M) = \begin{cases} \{a \leftarrow bd(r) \mid a \in M \cap lit(W_0)\} & \text{if } M \models W_0 \\ \emptyset & \text{otherwise.} \end{cases}$$

The *instance of* P *w.r.t.* M, denoted $inst(P, M)$, is the program

$$inst(P, M) = \cup_{r \in P} inst(r, M). \tag{9}$$

Note that an instance of a program is a basic program.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Definition 5
Let P be a weight constraint program and M be a set of atoms. M is an answer set of P iff M is an answer set of the instance of P w.r.t. M.

In the next section, we will show that, for some weight constraint programs, the stable model and the answer set semantics coincide, while, for some others, these semantics are different.

Before ending this section, we give a useful proposition, which shows a one-to-one correspondence between the stable models/answer sets of an arbitrary weight constraint program and those of its basic program counterparts. This result will be used later in this paper.

Proposition 1
Let P be a weight constraint program and M be a model of P. Define

$$P' = \{p \leftarrow W_1, \ldots, W_n \mid W_0 \leftarrow W_1, \ldots W_n \in P \text{ and } p \in lit(W_0) \cap M\}.$$

Then, M is a stable model (respectively, answer set) of P iff M is a stable model (respectively, answer set) of P'.

Proof
The correspondence between answer sets follows from Definition 5 above. For the correspondence between stable models, note that $P^M = P'^M$. □

3 Relating answer sets with stable models

In this section, we relate answer sets with stable models. First, we give a sufficient condition under which they agree with each other. Then, we show the difference between these semantics, that is, stable models that are not answer sets may be circular justified, based on a formal notion of circular justification. At the end, we discuss the related justifications in the literature.

3.1 When semantics agree

We show that for a broad class of weight constraint programs, the stable models are precisely answer sets, and vice versa.

Given a weight constraint W of the form (1) and a set of atoms M, we define $M_a(W) = \{a_i \in M \mid a_i \in lit(W)\}$ and $M_b(W) = \{b_i \in M \mid \texttt{not}\, b_i \in lit(W)\}$. Since W is always clear by context, we will simply write M_a and M_b.

Definition 6
Let M be a set of atoms and W be a weight constraint of the form (1). W is said to be *strongly satisfiable by* M if $M \models W$ implies that for any $V \subseteq M_b$, $w(W, M \setminus V) \leqslant u$. W is *strongly satisfiable* if for any set of atoms M, W is strongly satisfiable by M. A weight constraint program P is *strongly satisfiable* if every weight constraint that appears in the body of a rule in P is strongly satisfiable.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Intuitively, a strongly satisfiable weight constraint is a weight constraint whose upper bound is large enough to guarantee that, if a set of atoms satisfies the constraint, then any of its subset also satisfies the constraint.

Strongly satisfiable programs constitute a nontrivial class of programs. In particular, weight constraints W that possess one of the following syntactically checkable conditions are strongly satisfiable.

- $lit(W)$ contains only atoms;
- $\sum_{i=1}^{n} w_{a_i} + \sum_{i=1}^{m} w_{b_i} \leqslant u$.

Note that upper bound free weight constraints satisfy the second condition above.

Example 1
The following constraints are all strongly satisfiable:

$$1\ [a = 1, b = 2]\ 2,$$
$$1\ [a = 1, \mathtt{not} b = 2]\ 3,$$
$$1\ [a = 1, \mathtt{not} b = 2].$$

But the weight constraint

$$1\ [a = 1, \mathtt{not} b = 2]\ 2$$

is not, since it is satisfied by $\{a, b\}$ but not by $\{a\}$. □

Strongly satisfiable weight constraints are not necessarily convex or monotone.

Example 2
Let A be the following weight constraint:

$$2[a = 1, b = 1, \mathtt{not} c = 1].$$

Since A is upper bound free, it is strongly satisfiable. But A is neither monotone nor convex, as $\{a\} \models A$, $\{a, c\} \not\models A$, and $\{a, b, c\} \models A$. □

We show that the stable model semantics coincides with the answer set semantics for strongly satisfiable programs. We need a lemma.

Lemma 2
Let W be a weight constraint of the form (1), and S and M be sets of atoms such that $S \subseteq M$. Then,

(i) If $S \models_M W$ then $S \models W^M$ and $w(W, M) \leqslant u$.
(ii) If $S \models W^M$ and W is strongly satisfiable by M, then $S \models_M W$.

Proof
(i) We prove it by contraposition. That is, we show that if $w(W, M) > u$ or $S \not\models W^M$, then $S \not\models_M W$. The case of $w(W, M) > u$ is simple, which leads to $M \not\models W$ hence $S \not\models_M W$.

Assume $S \not\models W^M$. By definition, the lower bound is violated, i.e., $w(W^M, S) < l'$, where $l' = l - \sum_{b_i \notin M} w_{b_i}$. Let $I = I_a \cup I_b$, where $I_a = S_a$ and $I_b = M_b$. Since $w(W^M, S) = w(W^I, S)$ and $w(W^M, S) < l'$, we have $w(W^I, S) < l'$. Then, from $I_b = M_b$ and the assumption $S \not\models W^M$, we get $S \not\models W^I$. It then follows from $I_a = S_a$

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

that $I \not\models W$. By construction, we have $S \cap Dom(W) \subseteq I \subseteq M \cap Dom(W)$, and therefore, we conclude $S \not\models_M W$.

(ii) Assume $S \not\models_M W$ and W is strongly satisfiable by M. We show $S \not\models W^M$. We have either $S \models W$ or $S \not\models W$. If $S \not\models W$ then clearly $S \not\models W^M$. Assume $S \models W$. Then, from $S \not\models_M W$, we have $\exists I$, $S \cap Dom(W) \subset I \subseteq M \cap Dom(W)$ such that $I \not\models W$. Since W is strongly satisfiable by M, if $M \models W$ then for any $R = M \setminus V$, where $V \subseteq M_b$, $w(W,R) \leqslant u$. Assume $M \models W$. Let R be such that $R_b = I_b$ and $I_a \subseteq R_a$. It is clear that $w(W,R) \leqslant u$ leads to $w(W,I) \leqslant u$. Thus, since $M \models W$, that $I \not\models W$ is due to the violation of the lower bound, i.e., $w(W,I) < l$.

Now consider $I' = S_a \cup M_b$, i.e., we restrict I_a to S_a and expand I_b to M_b. Note that by construction, it still holds that $S \cap Dom(W) \subset I' \subseteq M$. Clearly that $I \not\models W$ leads to $I' \not\models W$, which is due to the violation of the lower bound, as $w(W,I') \leqslant w(W,I)$, i.e., we have $w(W,I') < l$. By definition, we have $w(W^{I'},I') < l'$, where $l' = l - \sum_{b_i \notin I'} w_{b_i}$. Note that since $I'_b = M_b$, we have $l' = l - \sum_{b_i \notin M} w_{b_i}$. Since $I'_a = S_a$, it follows that $w(W^{I'},S) < l'$. Now, since $W^{I'}$ is precisely the same constraint as W^M, we have $w(W^{I'},S) = w(W^M,S)$, and therefore, $w(W^M,S) < l'$. This shows $S \not\models W^M$. □

Theorem 1

Let P be a weight constraint program and $M \subseteq At(P)$. Suppose for any weight constraint W appearing in the body of a rule in P, W is strongly satisfiable by M. Then, M is a stable model of P iff M is an answer set of P.

Proof

Due to Proposition 1, we only need to prove the claim for basic weight constraint programs.

Assume P is a basic weight constraint program and M is a model of P such that all weight constraints in P are strongly satisfiable by M. It suffices to prove that for any positive integer k, $T^k_{P^M}(\emptyset) = K^k_P(\emptyset, M)$, by induction on k.

Base case: $k = 0$. We have $T^k_{P^M}(\emptyset) = K^k_P(\emptyset, M) = \emptyset$.

Induction Step: Assume, for any $k > 0$, $T^k_{P^M}(\emptyset) = K^k_P(\emptyset, M)$, and prove that $T^{k+1}_{P^M}(\emptyset) = K^{k+1}_P(\emptyset, M)$. Let a be an atom such that $a \notin T^k_{P^M}(\emptyset)$ and $a \in T^{k+1}_{P^M}(\emptyset)$. Then, there exists a rule $r \in P$ such that $a \in lit(hd(r))$ and $T^k_{P^M}(\emptyset) \models W^M$, for each $W \in bd(r)$. It then follows from part (ii) of Lemma 2 that $T^k_{P^M}(\emptyset) \models_M W$. Then, $K^k_P(\emptyset, M) \models_M W$ by the induction hypothesis. So, $a \in K^{k+1}_P(\emptyset, M)$. Thus, $T^{k+1}_{P^M}(\emptyset) \subseteq K^{k+1}_P(\emptyset, M)$. Similarly, we can show $K^{k+1}_P(\emptyset, M) \subseteq T^{k+1}_{P^M}(\emptyset)$ using part (i) of Lemma 2. Thus, $T^{k+1}_{P^M}(\emptyset) = K^{k+1}_P(\emptyset, M)$.

We therefore conclude $T^\infty_{P^M}(\emptyset) = K^\infty_P(\emptyset, M)$. □

The following theorem follows from Theorem 1 and the definition of strongly satisfiable programs.

Theorem 2

Let P be a strongly satisfiable weight constraint program and $M \subseteq At(P)$. M is a stable model of P iff M is an answer set of P.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

3.2 When semantics disagree

It has been shown that for logic programs with arbitrary abstract constraints the semantics based on conditional satisfaction are the most conservative in that the answer sets under this semantics are answer sets/stable models of a number of other semantics (Son *et al.* 2007). It is then expected that the same holds true for programs with concrete constraints, such as weight constraints.

Theorem 3
Let P be a weight constraint program. Every answer set of P is a stable model of P, but the converse does not hold.

Proof
Let M be a set of atoms. Using part (i) of Lemma 2, it is easy to show by induction that for any positive integer k, we have $K_P^k(\emptyset, M) \subseteq T_{P^M}^k(\emptyset)$. For the converse, see the counterexample in Example 3 below. □

Question arises as why some stable models are not answer sets. Later in Section 6, we will give a technical answer to this question. Here, we suggest that in these extra stable models there may exist circular justifications. Consider the following example.

Example 3
Let P be a single-rule program:

$$a \leftarrow [\texttt{not}\, a = 1]\ 0. \tag{10}$$

Let $M_1 = \emptyset$ and $M_2 = \{a\}$. The weight constraint $[\texttt{not}\, a = 1]\ 0$ in P is not strongly satisfiable, since although M_2 satisfies the upper bound, its subset M_1 does not. Both M_1 and M_2 are stable models by Definition 1. Note that this is because $P^{M_1} = \emptyset$ and $P^{M_2} = \{a \leftarrow\}$. But, M_1 is an answer set and M_2 is not, by Definition 5.

The reason that M_2 is not an answer set of P is due to the fact that a is derived by its being in M_2. This kind of circular justification can be seen more intuitively below using equivalence substitutions.

- The weight constraint is substituted with an equivalent aggregate:

$$a \leftarrow COUNT(\{X \mid X \in D\}) = 1,$$

  where $D = \{a\}$.
- The weight constraint is transformed to an equivalent one without negative literal, but with a negative weight, according to Simons *et al.* (2002):

$$a \leftarrow [a = -1]{-}1.$$

- The weight constraint is substituted with an equivalent *abstract constraint atom* (Marek and Truszczyński 2004)[3]:

$$a \leftarrow (\{a\}, \{\{a\}\}).$$

[3] An *abstract constraint atom* is a pair (D, C), where D is a finite set of ground atoms called the *domain*, and C is a collection of subsets of D called *admissible solutions*. In this example, the set $I = \{a\}$ satisfies the abstract constraint atom, since the admissible solution $\{a\}$ in it is satisfied by I.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

For the claim of equivalence, note that for any set of atoms M, we have: $M \models [\texttt{not}a = 1]0$ iff $M \models [a = -1]{-}1$ iff $M \models COUNT(\{X \mid X \in D\}) = 1$ iff $M \models (\{a\}, \{\{a\}\})$. □

For logic programs with abstract constraint atoms, it is often said that all of the major semantics coincide for programs with monotone constraints. For example, this is the case for the semantics proposed in Faber *et al.* (2004), Marek and Remmel (2004), Ferraris (2005), Liu and Truszczyński (2006), Marek *et al.* (2007), Son *et al.* (2007), Shen *et al.* (2009), and Liu *et al.* (2010). What is unexpected is that this is not the case for the stable model semantics for weight constraint programs. By the standard definition of monotonicity, the constraint $[\texttt{not}a = 1]\ 0$ is actually monotone!

One may think that the culprit for M_2 above is because it is not a minimal model. However, the following example shows that stable models that are minimal models may still be circularly justified.

Example 4
Consider the following weight constraint program P (obtained from the one in Example 3 by adding the second rule):

$$a \leftarrow [\texttt{not}a = 1]\ 0, \tag{11}$$

$$f \leftarrow \texttt{not}f, \texttt{not}a. \tag{12}$$

Now, $M = \{a\}$ is a minimal model of P, and also a stable model of P, but clearly a is justified by its being in M. □

We now give a more formal account of *circular justification* for stable models, borrowing the idea of *unfounded sets* previously used for normal programs (van Gelder *et al.* 1991) and logic programs with monotone and antimonotone aggregates (Calimeri *et al.* 2005).

Definition 7
Let P be an weight constraint program and M be a stable model of P. M is said to be *circularly justified*, or simply *circular*, if there exists a nonempty set $U \subseteq M$ such that $\forall \phi \in U$, $M \setminus U$ does not satisfy the body of any rule $r \in P$ such that $\phi \in lit(hd(r))$. Otherwise, M is said to be *noncircular*.

Proposition 2
Let P be a weight constraint program and M be a stable model of P. If M is an answer set of P, then M is not circular.

Proof
Let P be a weight constraint program and M be an answer set of P. Assume M is circular. Then, there exists a nonempty subset $U \subseteq M$ such that $\forall \phi \in U$, $M \setminus U$ does not satisfy the body of any rule $r \in P$ such that $\phi \in lit(hd(r))$. By a simple induction on the construction of $K_P^\infty(\emptyset, M)$, it can be shown that for each of such ϕ, we have $\phi \notin K_P^\infty(\emptyset, M)$. This contradicts the assumption that M is an answer set. □

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Example 3 shows that extra stable models (the stable models that are not answer sets) of a program may be circular. However, not all extra stable models are necessarily circular, according to Definition 7. Therefore, the notion of circularity given in Definition 7 only serves as a partial characterization of circular justification.

Example 5
Consider a weight constraint program P that consists of the following three rules:

$$b \leftarrow 1[\texttt{not}b = 1], \tag{13}$$

$$b \leftarrow [\texttt{not}b = 1]0. \tag{14}$$

$M = \{b\}$ is a stable model but not an answer set of P. However, it can be verified that M is not circular under Definition 7: b can be derived from the first rule if we don't have b, and by the second rule if we do. □

We shall comment that other forms of noncircular nature of answer sets have been formulated in different ways, e.g., by the existence of a level mapping (Son *et al.* 2007) and by a translation of a constraint to sets of solutions (Pelov *et al.* 2003).

3.3 Other justifications

A semantics is a formal account of intuitions of what justifications for atoms in a stable model ought to be. For weight constraint programs, there are different intuitions.

Consider the single-rule program P in Example 3 again

$$a \leftarrow [\texttt{not}a = 1]0. \tag{15}$$

One possible interpretation of how a is "justified" to be in a stable model is by using the transformation proposed in Marek *et al.* (2007). By the transformation, P is translated to P^*, which consists the following rules:

$$a \leftarrow [b = 1]0, \tag{16}$$

$$b \leftarrow 0[a = 1]0. \tag{17}$$

It can be verified that the set $M = \{a\}$ is a stable model of P^*. The justification of a is: given M, b cannot be in any stable model of P^* by the second rule, then a can be derived by the first rule. Note that the above transformation introduces the new atom b in the translated program P^*. It assumes that $\texttt{not}a$ implies something new to the original program. The justification of atom a depends on the truth status of the new atom b. Whether such a justification is intuitive seems arguable.

Another justification of a is by transforming the program P to a program with nested expressions (Ferraris and Lifschitz 2005). We will discuss this in more details later in Section 6.4.

For this example, we also note that the set $M = \{a\}$ is not an answer set under any semantics based on computations studied in Liu *et al.* (2010).

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

4 Transformation to strongly satisfiable programs

We show that the gap between the answer set semantics and the stable model semantics can be closed by a transformation, which translates a weight constraint program to a strongly satisfiable program whose stable models are free of circular justifications. In this way, we are able to apply current ASP systems that implement the stable model semantics to compute answer sets for weight constraint programs. In particular, later on we will use strongly satisfiable programs to represent aggregate programs so that an implementation of aggregate programs can be realized by an implementation of weight constraint programs.

4.1 Strongly satisfiable encoding

We present an encoding of a weight constraint, where a weight constraint is represented by two strongly satisfiable weight constraints. This encoding captures conditional satisfaction for weight constraints in terms of standard satisfaction. In other words, for weight constraints, the encoding allows conditional satisfaction to be checked by standard satisfaction.

Definition 8
Let W be a weight constraint of the form (1). The *strongly satisfiable encoding* of W, denoted (W_l, W_u), consists of the following constraints:

$$W_l : l[a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathtt{not} b_1 = w_{b_1}, \mathtt{not} b_1 = w_{b_m}],$$

$$W_u : -u + \textstyle\sum_{i=1}^{n} w_{a_i} + \sum_{i=1}^{m} w_{b_i}[\mathtt{not} a_1 = w_{a_1}, \ldots, \mathtt{not} a_n = w_{a_n}, b_1 = w_{b_1}, \ldots, b_m = w_{b_m}].$$

Intuitively, W_l and W_u are to code the lower and upper bound constraints of W, respectively. It is easy to verify that the encoding is satisfaction-preserving, as shown below.

Lemma 3
Let W be a weight constraint, (W_l, W_u) be its strongly satisfiable encoding, and M be a set of atoms. $M \models W$ iff $M \models W_l$ and $M \models W_u$.

Proof
The satisfaction of W_l is trivial, since W_l is just the lower bound part of W. We show that W_u is the upper bound part of W. Note that the upper bound part of W is

$$\textstyle\sum_{i=1}^{n} a_i \cdot w_{a_i} + \sum_{i=1}^{m} b_i \cdot (-w_{b_i}) \leqslant u - \sum_{i=1}^{m} w_{b_i}, \tag{18}$$

which is equivalent to

$$-u + \textstyle\sum_{i=1}^{m} w_{b_i} \leqslant \sum_{i=1}^{n} a_i \cdot (-w_{a_i}) + \sum_{i=1}^{m} b_i \cdot w_{b_i}. \tag{19}$$

By the transformation that eliminates the negative weights (introduced in Section 2.1), the constraint (19) is equivalent to the weight constraint W_u. □

Using Lemmas 2 and 3, we establish the following theorem.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Theorem 4
Let W be a weight constraint, (W_l, W_u) be the strongly satisfiable encoding of W, and S and M be two sets of atoms such that $S \subseteq M$. $S \models_M W$ iff $S \models W_l^M$ and $S \models W_u^M$.

Proof
($\Rightarrow$) Since W_l^M is the same as W^M, by part (i) of Lemma 2, we have $S \models W_l^M$. In the following, we show $S \models W_u^M$.

Assume $S \models_M W$ and $S \subseteq M$. Then, by definition, we have $S \models W$, and $\forall I$ such that $S \cap lit(W) \subseteq I$ and $I \subseteq M \cap lit(W)$, $I \models W$. Let $I = I_a \cup I_b$ such that $I_b = S_b$ and $I_a = M_a$. Under this notation, from the assumption $S \models_M W$ and $S \subseteq M$, we get $S \cap lit(W) \subseteq I$ and $I \subseteq M \cap lit(W)$. It follows that $w(W, I) \leqslant u$, that is, $\sum_{a_i \in I_a} w_{a_i} + \sum_{b_i \notin I_b} w_{b_i} \leqslant u$. This implies $\sum_{a_i \in M_a} w_{a_i} + \sum_{b_i \notin S_b} \leqslant u$, from which the following inequations can be derived:

$$\begin{aligned}
&\textstyle\sum_{i=1}^{n} w_{a_i} - \sum_{a_i \notin M_a} w_{a_i} + \sum_{i=1}^{m} w_{b_i} - \sum_{b_i \in S_b} w_{b_i} \leqslant u,\\
&\textstyle\sum_{b_i \in S_b} w_{b_i} \geqslant -u + \sum_{i=1}^{n} w_{a_i} + \sum_{i=1}^{m} w_{b_i} - \sum_{a_i \notin M_a} w_{a_i},\\
&\textstyle\sum_{b_i \in S} w_{b_i} \geqslant -u + \sum_{i=1}^{n} w_{a_i} + \sum_{i=1}^{m} w_{b_i} - \sum_{a_i \notin M} w_{a_i}.
\end{aligned}$$

The last one shows $S \models W_u^M$.

($\Leftarrow$) Assume $S \models W_l^M$ and $S \models W_u^M$. Since neither has an upper bound, both of them are strongly satisfiable. From part (ii) of Lemma 2, we have $S \models_M W_l$ and $S \models_M W_u$. It then follows $\forall I$ such that $S \cap lit(W) \subseteq I$ and $I \subseteq M \cap lit(W)$, $I \models W_l$ and $I \models W_u$. Then, by Theorem 3, we have $\forall I$ such that $S \cap lit(W) \subseteq I$ and $I \subseteq M \cap lit(W)$, $I \models W$. This shows $S \models_M W$. □

4.2 Transformation for the answer set semantics

Using the strongly satisfiable encoding of weight constraints, a weight constraint program can be translated to a strongly satisfiable program so that the answer sets of the original program are precisely the stable models of the translated program and vice versa.

Definition 9
Let P be a weight constraint program. The *strongly satisfiable translation* of P, denoted $Tr(P)$, is the program obtained by replacing each W in the body of rules in P by the strongly satisfiable encoding of W.

Theorem 5
Let P be a weight constraint program and M be a set of atoms. M is an answer set of P iff M is a stable model of $Tr(P)$.

Proof
Due to Proposition 1, we only need to prove the claim for basic weight constraint programs.

Using Theorem 4, we have a one-to-one correspondence between the derivations based on conditional satisfaction (Definition 4) and the derivations in the

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

construction of the least model (Definition 1), which can be shown by an easy induction on the length of these constructions. □

Example 6
Consider a program P with a single rule:

$$a \leftarrow 0[\texttt{not}a = 3]2.$$

$Tr(P)$ consists of

$$a \leftarrow 0[\texttt{not}a = 3],\ 1[a = 3].$$

The weight constraints in $Tr(P)$ are all upper bound free, hence $Tr(P)$ is strongly satisfiable. Both $\emptyset$ and $\{a\}$ are stable models of P, but $\emptyset$ is the only stable model of $Tr(P)$, which is also the only answer set of P.

5 Representing aggregate programs by weight constraint programs

In this section, we propose an approach to computing the answer sets of an aggregate program. For this, we translate an aggregate program to a strongly satisfiable weight constraint program and then compute its stable models as answer sets.

5.1 Aggregates as weight constraints

This section shows that the aggregates can be encoded as weight constraints. In the following, given sets M and S, it is convenient to express the set M, restricted to S, as $M_{|S}$ which is defined by $M \cap S$.

Definition 10
Let A be an aggregate in the form (7). A set of weight constraints $\{W_1, \ldots, W_n\}$ is an *weight constraint encoding* (or *encoding*) of A, denoted $e(A)$, if for any model M of A, there is a model M' of $e(A)$ such that $M'_{|Dom(A)} = M$, and for any model M' of $e(A)$, $M'_{|Dom(A)}$ is a model of A.

We show the encoding of aggregates of the form (7), where the operator op is $\geqslant$. The encoding can be easily extended to other relational operators except for the operator $\neq$ (more on $\neq$ later in this section). For example, aggregate $SUM(\{X \mid p(X)\}) > k$ can be expressed as $SUM(\{Y \mid p(Y)\}) \geqslant k + 1$.

The encoding works for the aggregates whose variable domain contains only integers. For the aggregates whose variable domain contains real numbers, each real number can be converted to an integer by multiplying a factor. In this case, the *Result* (in the formula (7)) also needs to be processed accordingly.

For convenience, below we may write negative weights in weight constraints. Recall that negative weights can be eliminated by a simple transformation.

$SUM, COUNT, AVG$
These aggregates can be encoded by weight constraints rather directly.

For instance, aggregate $SUM(\{X \mid p(X)\}) \geqslant k$ can be represented by

$$k\ [p(a_1) = a_1, \ldots, p(a_n) = a_n], \tag{20}$$

where the domain of the aggregate is $\{p(a_1), \ldots, p(a_n)\}$.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Note that a multiset in $SUM(\{X \mid p(X)\}) \geqslant k$ can be encoded directly by a weight constraint, since the latter does not require distinct literals in it. For example, for the aggregate $A = SUM(X \mid p(X)) > k$, where X is defined by the multiset $\{p(1), p(1), p(2)\}$, the aggregate can be encoded by the weight constraint $k[p(1) = 1, p(1) = 1, p(2) = 2]$.

We note that aggregates $COUNT(\{X \mid p(X)\}) \geqslant k$ and $AVG(\{X \mid p(X)\}) \geqslant k$ can be encoded simply by substituting the weights in (20) with 1 and $a_i - k$ (for AVG the lower bound k is also replaced by zero), respectively.

MAX

Let $A = MAX(\{X \mid p(X)\}) \geqslant k$ be an aggregate. The idea in the encoding of A is that for a set of numbers $S = \{a_1, \ldots, a_n\}$, the maximum number in S is greater than or equal to k if and only if

$$\sum_{i=1}^{n} (a_i - k + 1) > -\sum_{i=1}^{n} |a_i - k + 1|. \tag{21}$$

For each atom $p(a_i)$, two new literals $p^+(a_i)$ and $p^-(a_i)$ are introduced. The encoding $e(A)$ consists of the following constraints:

$$0\ [p(a_i) = -1, p^+(a_i) = 1, p^-(a_i) = 1]\ 0,\ 1 \leqslant i \leqslant n, \tag{22}$$

$$0\ [p(a_i) = -d_i, p^+(a_i) = d_i],\ 1 \leqslant i \leqslant n, \tag{23}$$

$$0\ [p(a_i) = d_i, p^-(a_i) = -d_i],\ 1 \leqslant i \leqslant n, \tag{24}$$

$$1\ [p(a_1) = d_1, p^+(a_1) = d_1, p^-(a_1) = -d_1,$$
$$\ldots, p(a_n) = d_n, p^+(a_n) = d_n, p^-(a_n) = -d_n], \tag{25}$$

$$1\ [p(a_1) = 1, \ldots, p(a_n) = 1], \tag{26}$$

where $d_i = a_i - k + 1$.

In the following, for any model M of such an encoding, $a = 1$ means $a \in M$ and $a = 0$ means $a \notin M$.

The constraints (22), (23), and (24) are used to encode $|a_i - k + 1|$. Clearly, if $a_i > k-1$, we have $p^+(a_i) = p(a_i)$ and $p^-(a_i) = 0$; if $a_i < k-1$, we have $p^-(a_i) = p(a_i)$ and $p^+(a_i) = 0$; and if $a_i = k - 1$, we have $p^+(a_i) = p(a_i)$ or $p^-(a_i) = p(a_i)$.

The constraint (25) encodes the relation (21) and the constraint (26) guarantees that a model of $e(A)$ is not an empty set.

MIN

Let $A = MIN(\{X \mid p(X)\}) \geqslant k$ be an aggregate. The idea in the encoding of A is that for a set of numbers $S = \{a_1, \ldots, a_n\}$, the minimal number in S is greater than or equal to k if and only if

$$\sum_{i=1}^{n} (a_i - k) = \sum_{i=1}^{n} |a_i - k|. \tag{27}$$

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Similar to MAX, the aggregate MIN can be encoded by the following weight constraints:

$$0\ [p^+(a_i) = 1, p^-(a_i) = 1, p(a_i) = -1]\ 0,\ 1 \leqslant i \leqslant n, \tag{28}$$

$$0\ [p^+(a_i) = d_i, p(a_i) = -d_i],\ 1 \leqslant i \leqslant n, \tag{29}$$

$$0\ [p^-(a_i) = -d_i, p(a_i) = d_i],\ 1 \leqslant i \leqslant n, \tag{30}$$

$$0\ [p(a_1) = d_1, p^+(a_1) = -d_1, p^-(a_1) = d_1, \ldots, p(a_n) = d_n, p^+(a_n) = -d_n, p^-(a_n) = d_n]\ 0, \tag{31}$$

$$1\ [p(a_1) = 1, \ldots, p(a_n) = 1], \tag{32}$$

where $d_i = a_i - k$.

The constraint (28), (29), and (30) are the same to the first three constraints in the encoding of MAX (except for the value of d_i), respectively. The constraint (31) encodes the relation (27) and the constraint (32) guarantees that a model of $e(A)$ is not an empty set.

We note that all the encodings above result in weight constraints whose collective size is linear in the size of the domain of the aggregate being encoded.

In the encoding of MAX (similarly for MIN), the first three constraints are the ones between the literal $p(a_i)$ and the newly introduced literals $p^+(a_i)$ and $p^-(a_i)$. We call them *auxiliary constraints*. The last two constraints code the relation between $p(a_i)$ and $p(a_j)$, where $i \neq j$. We call them *relation constraints*. Let A be an aggregate, we denote the set of auxiliary constraints in $e(A)$ by $a(A)$ and the set of relation constraints by $r(A)$. If A is aggregate SUM, $COUNT$, or AVG, we have that $r(A) = e(A)$, because no new literals are introduced in their encodings.

Theorem 6

The set of weight constraint (20), the set of weight constraints from (22) to (26), and the set of weight constraints from (28) to (32) are weight constraint encodings (Definition 10) of the aggregates SUM, MAX, and MIN, respectively.

Proof

The proof for the encoding of aggregate SUM is straightforward. The proof for the encoding of aggregate MIN is similar to that for MAX, which we show below.

Let M be a set of atoms and $M \models A$. Suppose $p(a_1) \in M$ and $a_1 \geqslant k$. Then, we can construct M' as follows:

(1) $p(a_i) \in M'$ and $p^+(a_i) \in M'$, if $p(a_i) \in M$ and $a_i \geqslant k$;
(2) $p(a_i) \in M'$ and $p^-(a_i) \in M'$, if $p(a_i) \in M$ and $a_i < k$.

We use W_1 to W_5 to denote the weight constraints in (22)–(26).

It is easy to check that the weight constraints W_1, W_2, and W_3 are satisfied by M'. Since $a_1 \geqslant k$, we have $p(a_1) \in M'$ and $p^+(a_1) \in M'$. Therefore, W_4 and W_5 are also satisfied by M'. So $M' \models e(A)$.

Let M' be a set of atoms and $M' \models e(A)$. Since M' satisfies W_1, W_2, and W_3, we have $p^+(a_i) = p(a_i)$ and $p^-(a_i) = 0$, for $a_i \geqslant k$; $p^-(a_i) = p(a_i)$ and $p^+(a_i) = 0$, for $a_i < k - 1$; and $p^+(a_i) = p(a_i)$ or $p^-(a_i) = p(a_i)$, if $a_i = k - 1$. Since $M' \models W_4$ and

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

$M' \models W_5$, there must be an i such that $a_i \geqslant k$ and $p(a_i) = 1$. That is, $p(a_i) \in M'_{|Dom(A)}$. Then, we have $M \models A$. □

5.2 Aggregate programs as weight constraint programs

We translate an aggregate program P to a weight constraint program, denoted $\tau(P)$, as follows:

(1) For each rule of the form (8) in P, we include in $\tau(P)$ a weight rule of the form

$$h \leftarrow r(A_1), \ldots, r(A_n), \tag{33}$$

where $r(A_i)$ is the conjunction of the weight constraints that encode the aggregate A_i; and

(2) If there are newly introduced literals in the encoding of aggregates, the *auxiliary rule* of the form

$$W \leftarrow p(a_i) \tag{34}$$

is included in $\tau(P)$, for each auxiliary constraint W of each atom $p(a_i)$ in the aggregates.

Note that a weight constraint program P can be translated to a strongly satisfiable program using the translation $Tr(P)$ given in Section 4.

We have the following theorem establishing the correctness of the transformation τ.

Theorem 7
Let P be an aggregate program where the relational operator is not $\neq$. For any stable model M of $Tr(\tau(P))$, $M_{|At(P)}$ is an answer set of P. For any answer set M for P, there is a stable model M' of $Tr(\tau(P))$ such that $M'_{|At(P)} = M$.

Proof
The rules of the form (33) are the translated counterpart of the rules in P. The auxiliary rules of the form (34) are added to enforce the auxiliary constraints.

Note that $Tr(\tau(P))$ is a strongly satisfiable program. Then, the theorem follows from Theorem 5 and Theorem 6. □

Remark For an aggregate where the relation operator is not '$\neq$', the aggregate can be encoded by a conjunction of weight constraints as we have shown in this section. In this case, logic equivalence leads to equivalence under conditional satisfaction. That is why we only need to ensure that an encoding is satisfaction-preserving.

For an aggregate where the relation operator is '$\neq$', two classes are distinguished. One consists of aggregates of the forms $COUNT(.) \neq k$. For these aggregates, the operator '$\neq$' can be treated as the disjunction of the operators '$>$' and '$<$'. Consider the aggregate $A = COUNT(.) \neq k$. A is logically equivalent to $A_1 \vee A_2$, where $A_1 = COUNT(.) > k$ and $A_2 = COUNT(.) < k$. Let R and S be two sets of atoms, it is easy to show that $R \models_S A$ iff $R \models_S A_1$ or $R \models_S A_2$. The other class consists of the aggregates of the forms $SUM(.) \neq k$, $AVG(.) \neq k$, $MAX(.) \neq k$, and $MIN(.) \neq k$.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

For these aggregates, the operator '$\neq$' cannot be treated as the disjunction of '$>$' and '$<$', since the conditional satisfaction may not be preserved. Below is an example.

Example 7
Consider the aggregates $A = SUM(\{X \mid p(X)\}) \neq -1$, $A_1 = SUM(\{X \mid p(X)\}) > -1$, and $A_2 = SUM(\{X \mid p(X)\}) < -1$. Note that A is logically equivalent to $A_1 \vee A_2$. Consider $S = \{p(1)\}$ and $M = \{p(1), p(2), p(-3)\}$. While S conditionally satisfies A w.r.t. M (i.e., $S \models_M A$), it is not the case that S conditionally satisfies A_1 w.r.t. M or S conditionally satisfies A_2 w.r.t. M.

6 Transformation to programs with nested expressions

In this section, we further relate answer sets with stable models in terms of logic programs with nested expressions. We formulate a transformation of weight constraint programs to programs with nested expressions and compare this transformation to the one in Ferraris and Lifschitz (2005). The comparison reveals that the difference of the semantics lies in the different interpretations of the constraint on the upper bounds of weight constraints in a program: while our transformation interprets it directly, namely, as "less than or equal to," the one in Ferraris and Lifschitz (2005) interprets it as "not greater than," which may create double negations (the atoms that are preceded by `not not`) in nested expressions. It is the semantics of these double negations that differentiates the two semantics.

6.1 Stable model semantics for programs with nested expressions

In the language of nested expressions, *elementary formulas* are atoms[4] and symbols $\bot$ (false) and $\top$ (true). *Formulas* are built from elementary formulas using the unary connective `not` and the binary connectives , (conjunction) and (disjunction).

A rule with nested expressions is of the form

$$Head \leftarrow Body, \tag{35}$$

where both *Body* and *Head* are formulas. For a rule r of the form (35), we use $hd(r)$ and $bd(r)$ to denote the *Head* and the *Body* of r, respectively.

A program with nested expressions is a set of rules with nested expressions.

The satisfaction of a formula by a set of atoms M is defined as follows:

- for a literal l, $M \models l$ if $l \in M$,
- $M \models \top$,
- $M \not\models \bot$,
- $M \models (F, G)$ if $M \models F$ and $M \models G$,
- $M \models (F; G)$ if $M \models F$ or $M \models G$,
- $M \models \texttt{not}F$ if $M \not\models F$.

[4] In the original syntax (Lifschits *et al.* 1999), elementary formulas can be atoms or atoms with classical negation $\neg$. The classical negation is irrelevant here.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

The reduct of a formula F with respect to a set of atoms M, denoted F^M, is defined recursively as follows:

- for an elementary formula F, $F^M = F$,
- $(F, G)^M = F^M, G^M$,
- $(F; G)^M = F^M; G^M$,
- $(\texttt{not}F)^M = \begin{cases} \bot, & \text{if } M \models F; \\ \top, & \text{otherwise.} \end{cases}$

The reduct of a program P with respect to a set of atoms M is the set of rules

$$Head^M \leftarrow Body^M, \tag{36}$$

for each rule of the form (35) in P.

The concept of a stable model is defined as follows.[5]

Definition 11 (Ferraris and Lifschitz 2005)
Let P be a logic program with nested expressions and M be a set of atoms. M is a stable model of P if M is a minimal model of P^M.

6.2 Direct nested expression encoding

We present a nested expression encoding, called the *direct nested expression encoding* of weight constraints. We show that conditional satisfaction of a weight constraint can be captured by the standard satisfaction of the reduct of the encoding of the weight constraint.

In the rest of the paper, we will use the following notation: For a set of literals X, we define $X^+ = \{a \mid a \in X\}$ and $X^- = \{a \mid \texttt{not}a \in X\}$.

Definition 12
Given a weight constraint W of the form (1), the *nested expression encoding* of W, denoted $NE(W)$, is the formula

$$;_{X \subseteq At(W) \text{ and } X \models W} [(,_{a_i \in X} a_i), (,_{b_i \in (At(W) \setminus X)} \texttt{not}b_i)], \tag{37}$$

where $At(W)$ is the set of atoms in W.[6]

Intuitively, $NE(W)$ is a nested expression representing the sets that satisfy W.

Example 8
Let $W = 1[a = 1, b = 1]1$ be a weight constraint. The subsets of $At(W)$ that satisfy W are $X_1 = \{a\}$ and $X_2 = \{b\}$. Thus, $NE(W) = a, \texttt{not}b; b, \texttt{not}a$.

[5] In the literature, the term *answer set* has been used. Here, we use *stable model* to avoid possible confusion with the answer sets defined in Son *et al.* (2007).

[6] For a more readable notation, let us use $(\Phi_{cond} Exp)$ for $(;_{cond} Exp)$ and $(\Psi_{cond} Exp)$ for $(,_{cond} Exp)$. Then, we can rewrite this formula as

$$\Phi_{X \subseteq At(W) \text{ and } X \models W} [(\Psi_{a_i \in X} a_i), (\Psi_{b_i \in (At(W) \setminus X)} \texttt{not}b_i)].$$

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

An interesting result of the directed encoding is that for the resulting nested expression, conditional satisfaction is precisely the satisfaction of the *reduct* of the expression. That is, given a weight constraint W and two sets of atoms $S \subseteq M$, $S \models_M W$ if and only if $S \models NE(W)^M$.

Before giving this result, we prove a lemma first.

Lemma 4

Let W be a weight constraint, $NE(W)$ its nested expression encoding, and (W_l, W_u) its strongly satisfiable encoding. Then, for any two sets of atoms S and M such that $S \subseteq M$, $S \models NE(W)^M$ iff $S \models W_l^M$ and $S \models W_u^M$.

Proof

In this proof, given a weight constraint W of the form (1), we denote $W^+ = \{a_1, \ldots, a_n\}$ and $W^- = \{b_1, \ldots, b_m\}$; we express $NE(W)$ as a disjunction of conjunctions X_i's, i.e., $NE(W) = ;_{1 \leqslant i \leqslant k} X_i$, for some $k \geqslant 0$. For notational convenience, such a conjunction may be referred to as a set as well, i.e., given a conjunction $X_i = x_1, \ldots, x_n$, we may use the same X_i to denote the set $\{x_1, \ldots, x_n\}$, and vice versa. Given a set S, we use the notation $\texttt{not}(S) = \{\texttt{not}a \mid a \in S\}$.

Since the proof is mainly about mathematical transformation, for convenience, we present it as a set of mechanical inferences.

($\Rightarrow$) We give a detailed proof for $S \models W_l^M$. The proof of $S \models W_u^M$ is similar.

(1) Assume $S \models NE(W)^M$ and $S \subseteq M$.
(2) $\exists X \in \{X_1, \ldots, X_k\}$ such that $S \models X^M$, due to (1) and the definition of $NE(W)$.
(3) $\forall b \in X^-$, we have $b \notin M$, that is, $X^- \subseteq {W_l}^+ \setminus M_b$, due to (2).
(4) $X^+ \subseteq S$, due to (2).
(5) $w(W_l, X) \geqslant l$, due to (2).
(6) $\sum_{a_i \in X^+} w_{a_i} + \sum_{b \in X^-} w_{b_i} \geqslant l$, due to (5).
(7) $\sum_{a \in X^+} w_{a_i} \leqslant \sum_{a_i \in S} w_{a_i}$, due to (4).
(8) $\sum_{b \in X^-} w_{b_i} \leqslant \sum_{b \notin M_b} w_{b_i}$ where $M_b = M \cap {W_l}^+$, due to (3).
(9) $\sum_{a_i \in S} w_{a_i} + \sum_{b \notin M} w_{b_i} \geqslant l$, due to (6)–(8).
(10) $\sum_{a_i \in S} w_{a_i} \geqslant l - \sum_{b \notin M} w_{b_i}$, due to (9).
(11) $w(W_l^M, S) \geqslant l - \sum_{b \notin M} w_{b_i}$, due to (10).
(12) $S \models W_l^M$, due to (11).

($\Leftarrow$)

(1) Assume $S \models W_l^M$, $S \models W_u^M$ and $S \subseteq M$.
(2) $\sum_{a_i \in S} w_{a_i} + \sum_{b_i \notin M} w_{b_i} \geqslant l$, due to (1).
(3) $-\sum_{b_i \in S} w_{b_i} \leqslant u - \sum_{i=1}^{n} w_{a_i} - \sum_{i=1}^{m} w_{b_i} + \sum_{a_i \notin M} w_{a_i}$, due to (1).
(4) $\sum_{a_i \in M} w_{a_i} + \sum_{b_i \notin S} w_{b_i} \leqslant u$, due to (3).
(5) Let $X = S \cup \texttt{not}((W^+ \cup W^-) \setminus M)$.
(6) $\sum_{a_i \in X} w_{a_i} + \sum_{b \notin X} w_{b_i} \leqslant \sum_{a_i \in M} w_{a_i} + \sum_{b_i \notin S} w_{b_i}$, due to (5) and (1).
(7) $\sum_{a_i \in X} w_{a_i} + \sum_{b \notin X} w_{b_i} \leqslant u$, due to (4) and (6).
(8) $l \leqslant \sum_{a_i \in X} w_{a_i} + \sum_{b \notin X} w_{b_i} \leqslant u$, due to (2) and (7).
(9) $X \models W$, due to (8).
(10) $X \in \{X_1, \ldots, X_k\}$, due to (9).

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

(11) $S \models X^M$, due to (5).
(12) $S \models NE(W)^M$, due to (11). □

Theorem 8
Let W be a weight constraint, and S and M be the two sets of atoms. Then, $S \models_M W$ iff $S \models NE(W)^M$.

Proof
This follows from Theorem 4 and Lemma 4. □

6.3 Transformation for the answer set semantics

Using the direct nested expression encoding of weight constraints, a weight constraint program can be translated to a program with nested expressions such that the answer sets of the original program are precisely the stable models of the translated program and vice versa.

Definition 13
Let P be a weight constraint program and r be a rule of the form (3) in P. The *nested expression translation of* r, denoted $NE(r)$, is the rule of the form

$$(l_1 ; \texttt{not} l_1), \ldots, (l_p ; \texttt{not} l_p), NE(W_0) \leftarrow NE(W_1), \ldots, NE(W_n), \tag{38}$$

where $l_1, \ldots, l_p$ are the positive literals in W_0.

Intuitively, the conjunctive term $(l_1 ; \texttt{not} l_1), \ldots, (l_p ; \texttt{not} l_p)$ represent that we are free to choose the atoms in the head of the rule to include in an answer set.

Definition 14
Let P be a weight constraint program. The *nested expression translation of* P, denoted $NE(P)$, is the program obtained by replacing each rule r in P by $NE(r)$.

Let P be a weight constraint program, r be a rule in P, and M be a set of atoms. By the definitions of $NE(r)$ and the reduct of a nested expression, we know that the reduct of $NE(r)$ w.r.t. M, denoted $NE(r)^M$, is the rule of the form

$$l_1, \ldots, l_{p'}, NE(W_0)^M \leftarrow NE(W_1)^M, \ldots, NE(W_n)^M, \tag{39}$$

where $\{l_1, \ldots l_{p'}\} = M \cap lit(W_0)$. We will use this fact in the proofs below.

Our main result is that the answer sets of a weight constraint program coincide with the stable models of its nested expression translation. To establish this, we need to show that a one-to-one correspondence between the least fixpoint of the operator K applied on a weight constraint program is the unique minimal model of the reduct of its nested expression translation.

To show the main theorem, we prove two lemmas first. One shows the coincidence of the satisfaction of a weight constraint, its nested expression encoding, and the reduct of the nested encoding. Then, using this lemma, we show the coincidence of the models of a weight constraint program, its nested expression translation, and the reduct of its nested expression translation. The later lemma helps to establish the correspondence between the least fixpoint of the operator K and the minimal model of the program reduct.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Lemma 5
Let W be a weight constraint of the form (1), $NE(W)$ be its nested expression encoding, and M be a set atoms. Then, $M \models W$ iff $M \models NE(W)$ iff $M \models NE(W)^M$.

Proof
That $M \models W$ iff $M \models NE(W)$ follows directly from the definition of $NE(W)$. We give a proof of the claim $M \models NE(W)$ iff $M \models NE(W)^M$. Suppose $NE(W) = X_1; \ldots; X_k$.

($\Rightarrow$) Since $M \models NE(W)$, there is an $X \in \{X_1, \ldots, X_k\}$ such that $X^+ \subseteq M$ and $X^- \cap M = \emptyset$. By the definition of the reduct of a nested expression, we have $M \models X^M$, and then, $M \models NE(W)^M$.

($\Leftarrow$) Since $M \models NE(W)^M$, there is an $X \in \{X_1, \ldots, X_k\}$ such that $M \models X^M$. By the definition of reduct, we have $X^+ \subseteq M$ and $X^- \cap M = \emptyset$. Therefore, $M \models X$, and then, $M \models NE(W)$. $\square$

Lemma 6
Let P be a weight constraint program and M be a set of atoms. $M \models P$ iff $M \models NE(P)$ iff $M \models NE(P)^M$.

Proof
The equivalence $M \models P$ iff $M \models NE(P)$ holds simply because $NE(P)$ is satisfaction-preserving. We give a proof of the statement $M \models NE(P)$ iff $M \models NE(P)^M$.

($\Rightarrow$) Suppose $M \models NE(P)$. Let $r^M \in NE(P)^M$ be a rule of the form (39). If $M \models NE(W_i)^M$ for all $1 \leqslant i \leqslant n$, then by Lemma 5, we have $M \models NE(W_i)$ for all $1 \leqslant i \leqslant n$. Since $M \models NE(P)$, then $M \models NE(W_0)$. Again by Lemma 5, we know $M \models NE(W_0)^M$. Since $\{l_1, \ldots l_{p'}\} = M \cap lit(W_0)$, we have $M \models hd(r^M)$. As r^M is arbitrary in $NE(P)^M$, we conclude $M \models NE(P)^M$.

($\Leftarrow$) Suppose $M \models NE(P)^M$. Let $r \in NE(P)$ be a rule of the form (38). If $M \models NE(W_i)$ for all $1 \leqslant i \leqslant n$, then by Lemma 5, we have $M \models NE(W_i)^M$ for all $1 \leqslant i \leqslant n$. Since $M \models NE(P)^M$, then $M \models NE(W_0)^M$. Again by Lemma 5, we know $M \models NE(W_0)$. It is obvious that $M \models_{,1 \leqslant i \leqslant n} (l_i; \mathtt{not} l_i)$. So $M \models hd(r)$. As r is any rule in $NE(P)$, we have $M \models NE(P)$. $\square$

Theorem 9
Let P be a weight constraint program, $NE(P)$ be the nested expression translation of P, and M be a set of atoms. Then, M is an answer set of P iff M is a stable model of $NE(P)$.

Proof
($\Rightarrow$) Let P be a weight constraint program and M be an answer set of P. Then, $M \models P$ and $M = K^{\infty}_{inst(P,M)}(\emptyset, M)$. By Lemma 6, $M \models NE(P)$ and $M \models NE(P)^M$. We prove by contradiction that M is a minimal model of $NE(P)^M$. Suppose for some $M' \subset M$ such that $M' \models NE(P)$. Note that by Lemma 6, we also have $M' \models NE(P)^M$. Let $a \in M \setminus M'$. Then, there exists a rule $r \in P$ of the form (3) satisfying that $a \in lit(hd(r))$ and $\exists k$ such that $K^k_{inst(P,M)}(\emptyset, M) \models_M W_i$, for all $W_i \in bd(r)$. Note that $K^k_{inst(P,M)}(\emptyset, M) \subseteq M$, hence Theorem 8 is applicable, from

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

which we know $K^k_{inst(P,M)}(\emptyset, M) \models NE(W_i)^M$, for all $W_i \in bd(r)$. Since $M' \models NE(P)^M$, we must have $a \in M'$, which contradicts to the assumption that $a \in M \setminus M'$. We therefore conclude that M is a minimal model of $NE(P)^M$, i.e., M is a stable model of $NE(P)$.

($\Leftarrow$) Let P be a weight constraint program and suppose M is a minimal model of $NE(P)^M$. By Lemma 6, M is a model of P. By the definitions of the operator K_P and $inst(P, M)$, we have $K^\infty_{inst(P,M)}(\emptyset, M) \subseteq M$. Let $\Delta = K^\infty_{inst(P,M)}(\emptyset, M)$. We will prove that $\Delta = M$. For this, let's assume it is not the case, i.e., $\Delta \subset M$. Since M is a minimal model of $NE(P)^M$, we have $\Delta \not\models NE(P)^M$. Then, there is a rule r of the form (3) in P and its corresponding rule $NE(r)$ of the form (39) in $NE(P)^M$ such that $\Delta \models NE(W_i)^M$ for all $W_i \in bd(NE(r))$, and $\Delta \not\models hd(NE(r))$. Since $M \models NE(P)^M$, for the rule $NE(r)$, we have $M \models NE(W_0)^M$. It follows that for some $L = \{l_1 \ldots l_{p'}\} \subseteq M \cap lit(W_0)$, $L \models NE(W_0)^M$. As $\Delta \not\models hd(NE(r))$, it must be the case that $\exists l \in L$ such that $l \notin \Delta$. By Theorem 8, however, that $\Delta \models NE(W_i)^M$ for all $W_i \in bd(NE(r))$ leads to $\Delta \models_M W_i$ for all $W_i \in bd(r)$. Then, by the definitions of the operator K_P and $inst(P, M)$, we must have $l \in \Delta$. This is a contradiction. Therefore, it must be the case that $\Delta = M$, and it follows that M is an answer set of P. □

Example 9

Consider the program P in Example 5. Its nested expression translation $NE(P)$ consists of the following rules:

$$a, \tag{40}$$

$$b \leftarrow a, \mathtt{not}\, b, \tag{41}$$

$$b \leftarrow \mathtt{not}\, a, \mathtt{not}\, b; b, \mathtt{not}\, a; a, b. \tag{42}$$

It can be verified that the only stable model of $NE(P)$ is $\{a\}$, which is also the only answer set of P.

6.4 Comparison to ferraris and lifschitz's translation

Ferraris and Lifschitz (2005) proposed a nested expression encoding of weight constraints. Using this encoding, a weight constraint program can be translated to a program with nested expressions such that there is a one-to-one correspondence between the stable models of the weight constraint program and the stable models of its translated program with nested expressions.

The difference between our nested expression translation and Ferraris and Lifschitz's translation (FL-translation) lies in the interpretation of the upper bound constraint of a weight constraint.

To illustrate this difference, let's denote a weight constraint W of the form (1) by $l[S]u$, where $[S] = [a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathtt{not}\, b_1 = w_{b_1}, \ldots, \mathtt{not}\, b_m = w_{b_m}]$. We call that $l[S]$ and $[S]u$ the lower bound constraint and upper bound constraint of W, respectively. Obviously, the lower bound and upper bound constraints are also weight constraints.

In our translation, the upper bound constraint $[S]u$ is *directly* encoded by the sets of atoms that satisfy it. In the FL-translation, $[S]u$ is encoded as $\mathtt{not}\, u + 1[S]$, where

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

$u + 1[S]$ is further encoded as the sets of atoms that satisfy the weight constraint $u + 1[S]$, possibly creating double negations.

This difference is the *only* reason that the stable models of our translated program are the answer sets of the original program while the stable models of the FL-translated program are the stable models of the original program. It should be clear that the extra stable models that are not answer sets are created by double negations generated by the indirect interpretation in the FL-translation.

We use the following example for an illustration.

Example 10
Consider the program P in Example 3, which consist of a single rule

$$a \leftarrow [\texttt{not}a = 1]0. \tag{43}$$

By our translation, $NE(P)$ consists of

$$a \leftarrow a. \tag{44}$$

The only stable model of $NE(P)$ is $\emptyset$, which is the unique answer set of P. By the FL-translation, the translated program P' is

$$a \leftarrow \texttt{not not}a. \tag{45}$$

The stable models of P' are $\emptyset$ and $\{a\}$. Among them, the set $\{a\}$ is not an answer set, but it is justified by the stable model semantics through the double negation `not not`a.

7 Experiments

The theoretical studies show that an aggregate program can be translated to a weight constraint program whose stable models are precisely the answer sets of the original program. This leads to a prototype implementation called ALPARSE[7] to compute the answer sets for aggregate programs. In ALPARSE, an aggregate program is first translated to a strongly satisfiable program using the translation given in Section 5, then the stable models of the translated strongly satisfiable program are computed using an ASP solver that implements the stable model semantics for weight constraint programs. In the next two subsections, we use SMODELS version 2.34 and CLASP version 2.0.3, respectively, as the underlying ASP solver of ALPARSE and compare ALPARSE with the implementations of aggregate programs SMODELS^A and DLV version 2007-10-11.[8]

The experiments are run on Scientific Linux release 5.1 with 3 GHz CPU and 1 GB RAM. The reported time of ALPARSE consists of the transformation time (from aggregate programs to strongly satisfiable programs), the grounding time (calling to LPARSE version 1.1.2 for SMODELS and GRINGO version 2.0.3 for CLASP), and the

[7] The name stands for computing aggregate programs by LPARSE program solvers. LPARSE is the synonym of weight constraint programs.

[8] We should note that DLV is a language that allows to express programs belonging to a higher complexity class.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Table 1. *Benchmarks used by* SMODELS^A

Program	Sample size	ALPARSE	SMODELS^A
Company contr.	20	0.03	0.09
Company contr.	40	0.18	0.36
Company contr.	80	0.87	2.88
Company contr.	120	1.40	8.14
Employee raise	15/5	0.01	0.69
Employee raise	21/15	0.05	4.65
Employee raise	24/20	0.05	5.55
Party invit.	80	0.02	0.05
Party invit.	160	0.07	0.1
NM1	125	0.21	0.1
NM1	150	0.25	0.1
NM2	125	0.30	1.24
NM2	150	0.68	2.36

search time (by SMODELS or CLASP). The time of SMODELS^A consists of grounding time, search time, and unfolding time (computing the solutions to aggregates). The time of DLV includes the grounding time and search time (the grounding phase is not separated from the search in DLV). All times are in seconds.

7.1 alparse *based on* smodels

In this section, we compare our approach with two systems, SMODELS^A and DLV.

Comparison with smodels^A

We compare the encoding approach proposed in last section to the unfolding approach implemented in the system SMODELS^A (Elkabani *et al.* 2005).[9] The aggregates used in the first and second sets of problems (the company control and employee raise problems) are SUM; the third set of problems (the party invitation problems) is $COUNT$, and the fourth and fifth sets of problems (the NM1 and NM2, respectively) are MAX and MIN, respectively.

The experimental results are reported in Table 1, where the "sample size" is measured by the argument used to generate the test cases. The times are the average of 100 randomly generated instances for each sample size. The results show that SMODELS is often faster than SMODELS^A, even though both use the same search engine.

Scale-up could be a problem for SMODELS^A, due to exponential blowup. For instance, for an aggregate like $COUNT(\{a|a \in S\}) \geqslant k$, SMODELS^A would list all *aggregate solutions* (Son and Pontelli 2007) in the unfolded program, whose number is $C^k_{|S|}$. For a large domain S and k being around $|S|/2$, this is a huge number. If one or a few solutions are needed, ALPARSE takes much less time to compute the corresponding weight constraints than SMODELS^A.

[9] The benchmarks and programs can be found at `www.cs.nmsu.edu/~ielkaban/asp-aggr.html`

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Table 2. *Seating*

Tables	Chairs	Time		Instance size	
		ALPARSE	DLV	ALPARSE	DLV
3	4	0.1	0.01	293	248
4	4	0.2	0.01	544	490
5	5	0.23	0.01	1,213	1,346
10	5	0.30	0.27	6,500	7,559
15	5	0.88	1.52	18,549	22,049
20	5	1.35	4.08	40,080	47,946
25	5	6.19	58.29	73,765	88,781
30	5	10.42	110.45	12,230	14,7567

Comparison with dlv

In Armi *et al.* (2003), the seating problem was chosen to evaluate the performance of DLV.[10] The problem is to generate a sitting arrangement for a number of guests, with m tables and n chairs per table. Guests who like each other should sit at the same table; guests who dislike each other should not sit at the same table. The aggregate used in the problem is $COUNT$.

We use the same setting to the problem instances as in Armi *et al.* (2003). The results are shown in Table 2. "Tables" and "Chairs" are the number of tables and the number of chairs at each table, respectively. The instance size is the number of atom occurrences in a ground program. We report the result of the average over 100 randomly generated instances for each problem size.

The experiments show that, by encoding logic programs with aggregates as weight constraint programs, ALPARSE solves the problem efficiently. For large instances, the running time of ALPARSE is about one order of magnitude lower than that of DLV and the sizes of the instances are also smaller than those in the language of DLV.

7.2 alparse *based on* clasp

We use the benchmarks reported in an ASP solver competition and run all instances for each benchmark.[11] In the experiments, we set the cutoff time to 600 s. The instances that are solved in the cutoff time are called "solvable," otherwise, "unsolvable". Table 3 is a summary of the results. In the table, the "Time" is the average running time in seconds for the solvable instances. It can be seen that ALPARSE constantly outperforms DLV by several orders of magnitude, except for the benchmark of Towers of Hanoi.

The system CLASP has progressed to support aggregates SUM, MIN, and MAX. The aggregates used in the benchmarks are SUM except for Towers of Hanoi where the aggregate MAX is used. The aggregate SUM is essentially the same as weight

[10] The program contains disjunctive head, but it can be easily transformed to a nondisjunctive program.
[11] We choose the benchmarks that have DLV programs available. The descriptions of benchmarks and programs can be found at `http://asparagus.cs.uni-potsdam.de/contest/`

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Table 3. *Benchmarks from ASP competition*

Benchmarks	Number of instances	Solved instances		Time	
		ALPARSE	DLV	ALPARSE	DLV
15 Puzzle	11	11	11	0.31	1.16
Schur number	5	5	4	0.10	0.62
Blocked N-queens	37	37	12	8.94	328.92
Wt. spanning tree	30	30	30	0.12	0.17
Bd. spanning tree	30	30	5	1.91	414.42
Hamiltonian cycle	29	29	29	0.84	29.22
Towers of Hanoi	29	29	21	21.61	18.35
Social golfer	168	129	107	1.52	14.69
Wt. latin square	35	35	18	0.03	105.01
Wt. dominating set	30	23	3	0.26	192.53
Traveling sales	24	24	23	0.11	12.74
Car sequencing	54	23	0	0.08	–

constraints. We compare the CLASP programs with the aggregate MAX and the corresponding translated weight constraint programs (note that the answer sets of this aggregate program correspond to those of the corresponding weight constraint program). The performances of CLASP on these two kinds of programs are similar.

As we have mentioned, the transformation approach indicates that it is important to focus on an efficient implementation of aggregate SUM rather than on the implementation of other aggregates one by one, since they can be encoded by SUM.[12]

8 Conclusion

We have shown that for a large class of programs, the stable model semantics coincides with the answer set semantics based on conditional satisfaction. In general, answer sets admitted by the latter are all stable models. When a stable model is not an answer set, it may be circularly justified. We have proposed a transformation, by which a weight constraint program can be translated to strong satisfiable program, such that all stable models are answer sets and thus well-supported models. We have also given another transformation from weight constraint programs to logic programs with nested expressions, which preserves the answer set semantics. In conjunction with the one given in Ferraris and Lifschitz (2005), their difference reveals precisely the relation between stable models and answer sets.

As an issue of methodology, we have shown that most standard aggregates can be encoded by weight constraints. Therefore, the ASP systems that support weight constraints can be applied to efficiently compute the answer sets of logic programs with aggregates. The experimental results demonstrate the effectiveness of this approach.

[12] The aggregate $TIMES$ can be translated to SUM, using a logarithm transformation, thanks to Tomi Janhunen for the comments during the presentation of Liu and You (2008).

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Currently, ALPARSE does not handle programs with aggregates like $SUM(.) \neq k$ or $AVG(.) \neq k$, due to the fact that the complexity of such programs is higher than NP. What is the best way to include this practically requires further investigation.

References

ARMI, D., FABER, W. AND IELPA, G. 2003. Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in DLV*. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI '03)*, 847–852.

BALDUCCINI, M., GELFOND, M., WATSON, R. AND NOGUEIRA, M. 2001. The USA-advisor: A case study in answer set planning. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR)*, 439–442.

CALDIRAN, O., HASPALAMUTGIL, K., OK, A., PALAZ, C., ERDEM, E. AND PATOGLU, V. 2009. Bridging the gap between high-level reasoning and low-level control. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR)*, 342–354.

CALIMERI, F., FABER, W., LEONE, N. AND PERRI, S. 2005. Declarative and computational properties of logic programs with aggregates. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI '05)*, 406–411.

DELGRANDE, J. P., GROTE, T. AND HUNTER, A. 2009. A general approach to the verification of cryptographic protocols using answer set programming. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR)*, 355–367.

DENECKER, M., PELOV, N. AND BRUYNOOGHE, M. 2001. Ultimate well-founded and stable semantics for logic programs with aggregates. In *Proc. of International Conference on Logic Programming (ICLP '01)*, 212–226.

DENECKER, M., VENNEKENS, J., BOND, S., GEBSER, M. AND TRUSZCZYNSKI, M. 2009. The second answer set programming competition. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR)*, 637–654.

ELKABANI, I., PONTELLI, E., AND SON, T. C. 2005. *Smodels*^{A}—A system for computing answer sets of logic programs with aggregates. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR '05)*, 427–431.

ERDEM, E., LIN, F. AND SCHAUB, T., Eds., 2009. Session 3. Original Application Papers. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR '09)*. Springer, 342–395.

FABER, W., LEONE, N. AND PFEIFER, G. 2004. Recursive aggregates in disjunctive logic programs. In *Proc. of European Conference on Logics in Artificial Intelligence (JELIA '04)*, 200–212.

FAGES, F. 1994. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science 1*, 51–60.

FERRARIS, P. 2005. Answer sets for propositional theories. In *Proc. Logic Programming and Non-Monotonic Reasoning (LPNMR '05)*, 119–131.

FERRARIS, P. AND LIFSCHITZ, V. 2005. Weight constraints as nested expressions. *Theory and Practice of Logic Programming 5*(1–2), 45–74.

GEBSER, M., KAUFMANN, B., NEUMANN, A. AND SCHAUB, T. 2007a. Conflict-driven answer set solving. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI '07)*, 386–392.

GEBSER, M., LIU, L., NAMASIVAYAM, G., NEUMANN, A., SCHAUB, T. AND TRUSZCZYŃSKI, M. 2007b. The first answer set programming system competition. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR '07)*, 1–17.

GELFOND, M. 2008. Answer sets. In *Handbook of Knowledge Representation*. Elsevier, Chapter 1, 285–316.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proc. of International Conference on Logic Programming (ICLP)*, 1070–1080.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press

Giunchiglia, E., Lierler, Y. and Maratea, M. 2006. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning 36*(4), 345–377.

Ielpa, S. M., Iiritano, S., Leone, N. and Ricca, F. 2009. An ASP-based system for e-tourism. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR)*, 368–381.

Lifschits, V., Tang, L. and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25*, 369–389.

Liu, L., Pontelli, E., Son, T. and Truszczynski, M. 2010. Logic programs with abstract constraint atoms: the role of computations. *Artificial Intelligence 174*(3–4), 295–315.

Liu, L. and Truszczyński, M. 2006. Properties and applications of programs with monotone and convex constraints. *Journal of Artificial Intelligence Research 7*, 299–334.

Liu, G. and You, J. 2008. Lparse programs revisited: semantics and representation of aggregates. In *Proc. of International Conference on Logic Programming (ICLP '08)*, 347–361.

Marek, V., Niemelä, I. and Truszczyński, M. 2007. Logic programs with monotone abstract constraint atoms. *Theory and Practice of Logic Programming 8*(2), 167–199.

Marek, V. and Truszczyński, M. 2004. Logic programs with abstract constraint atoms. In *Proc. of Association for the Advancement of Artificial Intelligence (AAAI '04)*, 86–91.

Marek, V. W. and Remmel, J. B. 2004. Set constraints in logic programming. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR '04)*, 167–179.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Math. and Artificial Intelligence 25*(3–4), 241–273.

Oetsch, J., Seidl, M., Tompits, H. and Woltran, S. 2009. cct on stage: Generalised uniform equivalence testing for verifying student assignment solutions. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR)*, 382–395.

Pelov, N., Denecker, M. and Bruynooghe, M. 2003. Translation of aggregate programs to normal logic programs. In *Proc. of Answer Set Programming (ASP '03)*, 29–42.

Pelov, N., Denecker, M. and Bruynooghe, M. 2004. Partial stable models for logic programs with aggregates. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR '04)*, 207–219.

Pelov, N., Denecker, M. and Bruynooghe, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming 7*, 301–353.

Shen, Y., You, J. and Yuan, L. 2009. Characterizations of stable model semantics for logic programs with arbitrary constraint atoms. *Theory and Practice of Logic Programming 9*(4), 529–564.

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

Son, T. C. and Pontelli, E. 2007. A constructive semantic characterization of aggregates in answer set programming. *Theory and Practice of Logic Programming 7*, 355–375.

Son, T. C., Pontelli, E. and Tu, P. H. 2007. Answer sets for logic programs with arbitrary abstract constraint atoms. *Journal of Artificial Intelligence Research 29*, 353–389.

van Gelder, A., Ross, K. and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38*(3), 620–650.

Wu, G., You, J. and Lin, G. 2007. Quartet based phylogeny reconstruction with answer set programming. *IEEE/ACM Transactions on Computational Biology and Bioinformatics 4*(1), 139–152.

You, J., Yuan, L. Y., Liu, G. and Shen, Y. 2007. Logic programs with abstract constraints: Representation, disjunction and complexities. In *Proc. of Logic Programming and Non-Monotonic Reasoning (LPNMR '07)*, 228–240.

https://doi.org/10.1017/S147106841100038X Published online by Cambridge University Press