Lparse Programs Revisited: Semantics and Representation of Aggregates

Guohua Liu and Jia-Huai You

Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
{guohua,you}@cs.ualberta.ca

Abstract. Lparse programs are logic programs with weight constraints as implemented in the SMODELS system, which constitute an important class of logic programs with constraint atoms. To effectively apply lparse programs to problem solving, a clear understanding of its semantics and representation power is indispensable. In this paper, we study the semantics of lparse programs, called the *lparse semantics*. We show that for a large class of programs, called *strongly satisfiable programs*, the lparse semantics agrees with the semantics based on conditional satisfaction. However, when the two semantics disagree, a stable model admitted by the lparse semantics may be circularly justified. We then present a transformation, by which an lparse program can be transformed to a strongly satisfiable one, so that no circular models may be generated under the current implementation of SMODELS. This leads to an investigation of a methodological issue, namely the possibility of compact representation of aggregate programs by lparse programs. We present some experimental results to compare this approach with the ones where aggregates are more explicitly handled.

1 Introduction

Lparse programs are logic programs with weight constraints implemented in the SMODELS system [15], which have been considered one of the most important recent extensions to answer set programming (ASP). Since weight constraints can be nonmonotone, lparse programs constitute a class of logic programs with constraints beyond monotone constraints.

In a related development, ASP has been extended to support abstract constraint atoms, called *c-atoms* for short, for representation and reasoning with constraints on sets of atoms [11,12]. Many constraints studied in the past, such as weight constraints, aggregates, and what are called global constraints in the Constraint Satisfaction Problem (CSP) [1,19] can be represented by c-atoms. In this sense, logic programs with c-atoms subsume lparse programs and logic programs with aggregates and global constraints. One focus in the study of logic programs with c-atoms is on the semantics, with a number of proposals for programs with various kinds of c-atoms, such as [10] for monotone c-atoms, [5,7,14,17] for aggregates, and [9,12,16] for arbitrary constraint atoms. All of these semantics agree on programs with monotone c-atoms. When nonmonotone c-atoms are present, they may admit different sets of stable models. The relationships between these semantics have been investigated in [16].

M. Garcia de la Banda and E. Pontelli (Eds.): ICLP 2008, LNCS 5366, pp. 347–361, 2008.

Despite of being one of the most popular systems, the semantics of lparse programs has not been fully studied. In [11], it is shown that lparse programs can be transformed to logic programs with monotone weight constraints while preserving the lparse semantics. Based on this result, in [10] weight constraints are translated to pseudo-boolean constraints. We do not know of any study that addresses the lparse semantics itself. For instance, since lparse programs are logic programs with constraint atoms, question arises as how the lparse semantics is related to the semantics for logic programs with constraint atoms. If differences exist, what are the nature of the differences and their potential implications in applications. These questions are important since SMODELS has been used for benchmarks and serious applications involving cardinality and weight constraints (e.g. [8,20]), and will likely be adopted in further endeavors in applying the ASP technology to real world applications.

In this paper, we study the semantics of lparse programs by investigating the relationship between it and the semantics proposed for logic programs with c-atoms. It turns out that the lparse semantics differs from all the other semantics. However, we show that for a large class of lparse programs, called *strongly satisfiable programs*, the lparse semantics agrees with the semantics based on conditional satisfaction [16]. For example, lparse 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 latter semantics also hold for these programs. One important property is that any answer set is a *well-supported model* [16], ensuring that any conclusion must be supported by a non-circular justification in the sense of Fages [6].

Our study further reveals that for lparse programs where the two semantics disagree, lparse-stable models may be circularly justified, based on a formal notion of circular justification. We then show that there exists a transformation from lparse programs to strongly satisfiable programs, which provides a way to run lparse programs under the current implementation of SMODELS without generating circular models.

The SMODELS system has been used to run logic programs with aggregates (or called aggregate programs). A methodological question of interest is whether the standard aggregates can be effectively encoded by weight constraints[1], and if so, what are the advantages. It turns out that most aggregates proposed for ASP can be encoded by weight constraints in linear size. Most of these encodings are straightforward and already widely used in SMODELS' applications (e.g., [8]), with the exception of the MAX/MIN constraints, which are more involved. In this sense, SMODELS can be seen as a system for aggregate programs already.

To evaluate this approach to representing and computing with aggregates, we have conducted a preliminary round of experiments. We compare SMODELS with SMODELS^A and DLV^A on logic programs with standard aggregates. Our experiments show that lparse programs often run faster, sometimes substantially faster, than the two aggregate systems for the benchmarks tested. This suggests 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 SMODELS by a simple front end to essentially

[1] In this paper, by *effective* or *compact* encoding of a constraint by weight constraints, we mean the collective size of the encoded weight constraints is linear in the size of constraint's *domain*.

transform 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 a more explicit implementation.

The next section gives some preliminary definitions. Section 3 shows that the lparse semantics is closely related to the semantics based on conditional satisfaction. The differences between these two semantics are studied in Section 4, whereas Section 5 presents a transformation which closes the gap between the two semantics. In section 6, encodings of standard aggregates by weight constraints are presented, which provide necessary information for the experiments that are reported in Section 7. Section 8 concludes the paper and proposes issues that require further investigation.

2 Preliminaries

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

2.1 Lparse Semantics

A *weight constraint* is of the form

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

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 (only integers are supported by smodels). 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) $l \leq w(W, M) \leq u$, 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 $S \models W$ and $R \models W$, then for any I such that $R \subseteq I \subseteq S$, we have $I \models W$.

An *lparse program* is a finite set of rules of the form

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

where each W_i is a weight constraint.

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

If every weight constraint is of the form $1\ [l = 1]\ 1$ where l is a literal, then an lparse program is essentially a normal program. The weight constraint $1\ [l\ =\ 1]\ 1$ will be simply written as l.

As pointed out in [15], 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: For 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

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

From now on, we assume that weights are non-negative if not said otherwise.

The stable models of lparse 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 constraint

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

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

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

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

Definition 1. [15] *Let P be an lparse program and $M \subseteq At(P)$. M is an* lparse-stable model *of P iff the following two conditions hold:*

1. *$M \models P$,*
2. *M is the least model of P^M.*

Note that P^M is an lparse program where all constraints are monotone and the head of each rule is an atom. Thus its least model can be computed by a fixpoint construction.

2.2 Semantics of Logic Programs with Constraint Atoms

An *abstract constraint atom* (or *c-atom*) is of the form (D, C), where D is a set of atoms called the *domain* of the c-atom, and C a collection of the subsets from 2^D, consisting of the *admissible solutions* to the constraint. Given a c-atom $A = (D, C)$, We use A_d and A_c to refer to D and C, respectively.

A logic program with c-atoms is a collection of rules of the form: $C_0 \leftarrow C_1, ..., C_k$, where each C_i is a c-atom.

For a rule r of the above form, the *head* of r, denoted by $hd(r)$ is C_0, and the *body*, denoted by $bd(r)$ is the set $\{C_1, ..., C_k\}$.

A set of atoms M satisfies a c-atom A, written $M \models A$, if $M \cap A_d \in A_c$. M is a *model* of a program P if for every rule $r \in P$, either $M \models hd(r)$ or $M \not\models bd(r)$. If

a c-atom is not satisfied by any set of atoms (such as c-atoms of the form $(D, \emptyset)$) and appears in the head of a rule, we may write a special symbol $\bot$ instead. A c-atom A is said to be *elementary* if it is of the form $(\{a\}, \{\{a\}\})$, which is just written as a. A rule is said to be *basic* if its head is either elementary or $\bot$. A program is *basic* if every rule in it is basic.

Abstract constraint atoms are introduced to represent constraints on sets. In practical constraint systems however, constraints are concrete and *language supported*, such as weight constraints, aggregates, and global constraints. The satisfaction of such a concrete constraint is pre-defined in the given language. We can associate a concrete constraint C in a given language with a c-atom C' such that C'_d is the same as the domain of C, and for any set of atoms M, $M \models C$ if and only if $M \models C'$. In this case, we call C' a *c-atom representation* of C.

For instance, for a weight constraint W of the form (1), a c-atom representation of W is a c-atom A whose domain A_d is the set of atoms appearing in W and A_c consists of those $S \subseteq A_d$ such that $S \models W$. Under this setting, lparse programs constitute a class of logic program with c-atoms, where c-atoms are just weight constraints.

The definition of answer sets in [16] is based on the abstract form of constraint atoms. It is notationally important to lift this definition to cover all constraint atoms, be they in the abstract form or in a language supported form.

In the sequel, a c-atom refers to a constraint atom, either in the abstract form or in a language supported form. We use $dom(C)$ to denote the domain of a c-atom, particularly, if A is a weight constraint, $dom(C) = \{a \mid a \in lit(A)$ *or* $\texttt{not}\ a \in lit(A)\}$.

Answer sets for logic programs with c-atoms are defined in two steps. In the first, answer sets for basic programs are defined, based on *conditional satisfaction*.

Definition 2. *Let M and S be sets of atoms and W be a c-atom. The set S* conditionally satisfies *W w.r.t. M, denoted by $S \models_M W$, if $S \models W$ and for every I such that $S \cap dom(W) \subseteq I \subseteq M$, we have $I \models W$.*

Given sets R and S, and a basic program P, the operator $T_P(R, S)$ is defined as:

$$T_P(R, S) = \{a : \exists r \in P, hd(r) = a \neq \bot,\ R \models_S bd(r)\}.$$

T_P is monotone w.r.t. its first argument, given that the second argument is fixed.

Definition 3. *Let M be a model of a basic program P. M is an* answer set *for P iff $M = T_P^\infty(\emptyset, M)$, where $T_P^0(\emptyset, M) = \emptyset$ and $T_P^{i+1}(\emptyset, M) = T_P(T_P^i(\emptyset, M), M)$, for all $i \geq 0$.*

In the second step, a logic program with c-atoms is represented by its instances in the form of basic programs, and the answer sets of the former are defined in terms of the ones of the latter.

Let P be a program with c-atoms and $r \in P$. By an abuse of notation, assume $hd(r)$ is a c-atom representation of the constraint in the head of rule r. Then, for each $\pi \in hd(r)_c$, the *instance* of r w.r.t. π is the set of rules consisting of

1. $b \leftarrow bd(r)$, for each $b \in \pi$, and
2. $\bot \leftarrow d, bd(r)$, for each $d \in hd(r)_d \backslash \pi$.

An *instance of* P is a basic program obtained by replacing each rule of P with one of its instances.

Definition 4. [16] *Let P be a logic program with c-atoms and $M \subseteq At(P)$. M is an answer set for P iff M is an answer set for one of its instances.*

From now on, *answer sets* of logic programs with c-atoms or c-atom representation of weight constraints refer to Definition 4 if not said otherwise.

3 Coincidence between Semantics

We show that, for a large class of lparse programs, the lparse semantics coincides with that of [16].

Notation: 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.

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

Strongly satisfiable lparse programs constitute an interesting 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;
- $\sum_1^n w_{a_i} + \sum_1^m w_{b_i} \leq u$.

For example, the following constraints are all strongly satisfiable: $1\ [a = 1, b = 2]\ 2$, $1\ [a = 1, \texttt{not}\ b = 2]\ 3$, and $1\ [a = 1, \texttt{not}\ b = 2]$. But $1\ [a = 1, \texttt{not}\ b = 2]\ 2$ is not, since it is satisfied by $\{a, b\}$ but not by $\{a\}$.

Strongly satisfiable constraints are not necessarily convex or monotone.

Example 1. Let $A = 2[a = 1, b = 1, \texttt{not}\ c = 1]$ be a weight constraint. Since A is upper bound free, it is strongly satisfiable. But A is neither monotone nor convex, since $\{a\} \models A$, $\{a, c\} \not\models A$, and $\{a, b, c\} \models A$. □

Theorem 1. *Let P be an lparse 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 an lparse-stable model of P iff M is an answer set for P.*

The theorem can be proved as follows. Let M and S be two sets of atoms such that $S \subseteq M$, and P be a program in which the weight constraints that appear in the bodies of rules in P are strongly satisfiable by M. We will prove a key lemma below which relates $S \models_M W$ with $S \models W^M$. The goal is to ensure a one-to-one correspondence between the derivations based on conditional satisfaction (Definition 3) and the derivations in the construction of the least model (Definition 1). Then it can be shown, by induction on the length of derivations, that an lparse-stable model of P is an answer set for an instance of P, and vice versa.

Lemma 1. *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 $w(W, M) \leq u$.*
(ii) *If $S \models W^M$ and W is strongly satisfiable by M, then $S \models_M W$.*

The proof of (i) is routine, but the proof of (ii) involves some subtleties.

Proof. (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 $\exists I$, $S \cap dom(W) \subset I \subseteq M$, 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 $V \subseteq M_b$, $w(W, R) \leq u$. Assume $M \models W$. Let R be such that $R_b = I_b$ and $I_a \subseteq R_a$. It's clear that $w(W, R) \leq u$ leads to $w(W, I) \leq u$. 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, $I \not\models W$ leads to $I' \not\models W$, which is also due to the violation of the lower bound, as $w(W, I') \leq w(W, I)$, 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 $S \not\models W^M$. □

By Theorem 1 and the definition of strongly satisfiable programs, we can show the following.

Theorem 2. *Let P be a strongly satisfiable lparse program, and $M \subseteq At(P)$ be a set of atoms. M is an lparse-stable model of P iff M is an answer set for P.*

4 When the Semantics Disagree

The following theorem can be proved using Lemma 1 and Example 2 below.

Theorem 3. *Every answer set of an lparse program P is an lparse-stable model of P, but the converse does not hold.*

Question: What happens to the lparse programs that are not strongly satisfiable.

Example 2. Let P be a program consisting of a single rule: $a \leftarrow [\texttt{not}\ a = 1]\ 0$ and $M_1 = \emptyset$ and $M_2 = \{a\}$ be two sets. 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. By Definition 1, P has two lparse stable models: M_1 and M_2. But, by Definition 4, M_1 is an answer set for P and M_2 is not. Note that M_2 is not a minimal model.

The reason that M_2 is not an answer set for 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 clearly below.

– The weight constraint is substituted with an equivalent aggregate:
$$a \leftarrow COUNT(\{X \mid X \in D\}) = 1, \text{ where } D = \{a\}.$$

- The weight constraint is substituted with its c-atom representation:
  $$a \leftarrow (\{a\}, \{\{a\}\}).$$
- The weight constraint is transformed to an equivalent one without negative literal, but with a negative weight, according to [15]:[2]
  $$a \leftarrow [a = -1]{-}1.$$

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

The type of circular justification observed here is similar to "answer sets by reduct" in dealing with nonmonotone c-atoms [16]. But 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 lparse-stable models that are also minimal models may still be circularly justified.

Example 3. Consider the following lparse program P (which is obtained from the one in Example 2 by adding the second rule):

$$a \leftarrow [\texttt{not}\ a = 1]\ 0 \qquad f \leftarrow \texttt{not}\ f, \texttt{not}\ a$$

Now, $M = \{a\}$ is a minimal model of P, and also an lparse-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 lparse-stable models, borrowing the idea of *unfounded sets* previously used for normal programs [18] and logic programs with monotone and antimonotone aggregates [3].

Definition 5. *Let P be an lparse program and M an lparse-stable model of P. M is said to be* circularly justified*, or simply* circular*, if there exists a non-empty set $U \subseteq M$ such that $\forall \phi \in U$, $M \setminus U$ does not satisfy the body of any rule in P where ϕ is in the literal set of the head of the rule.*

Theorem 4. *Let P be an lparse program and M an lparse-stable model of P. If M is an answer set for P, then M is not circular.*

Example 2 shows that extra lparse-stable models (lparse-stable models that are not answer sets) of a program may be circular. However, not all extra lparse-stable models are necessarily circular.

Example 4. Consider an lparse program P that consists of three rules.

$$a \leftarrow \qquad b \leftarrow 2[a = 1, \texttt{not}\ b = 1] \qquad b \leftarrow [a = 1, \texttt{not}\ b = 1]1$$

$M = \{a, b\}$ is an lparse-stable model but not an answer set for P. However, it can be verified that M is not circular under our definition (Definition 5).[3] □

[2] Caution: Due to an internal bug, SMODELS produces $\emptyset$ as the only stable model, which is inconsistent with the lparse semantics defined in [15].

[3] It appears that the notion of circular justification is still an open issue; there could be different intuitions and definitions.

5 Transformation to Strongly Satisfiable Programs

In this section we show that all lparse programs can be transformed to strongly satisfiable programs. This is achieved by replacing each weight constraint of form (1) in a given program by two upper bound-free weight constraints.

Let W be a weight constraint of form (1). The *strongly satisfiable encoding* of W, denoted by (W_1, W_2) consists of the following constraints:

$$W_1 : l[a_1 = w_{a_1}, ..., a_n = w_{a_n}, \texttt{not}\ b_1 = w_{b_1}, \texttt{not}\ b_1 = w_{b_m}]$$
$$W_2 : -u + \sum_{i=1}^{n} w_{a_i} + \sum_{i=1}^{m} w_{b_i}[\texttt{not}\, a_1 = w_{a_1}, ..., \texttt{not}\, a_n = w_{a_n}, b_1 = w_{b_1}, ..., b_m = w_{b_m}]$$

Intuitively, W_1 and W_2 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 in the following lemma.

Lemma 2. *Let W be a weight constraint, (W_1, W_2) be its strongly satisfiable encoding, and M be a set of atoms. $M \models W$ iff $M \models W_1$ and $M \models W_2$.*

By Lemmas 1 and 2, the following result can be established.

Theorem 5. *Let W be a weight constraint, (W_1, W_2) 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 $S \models W_1^M$ and $S \models W_2^M$.*

Theorem 5 guarantees the one-to-one correspondence between the derivations based on conditional satisfaction (Definition 3) and the derivations in the construction of the least model (Definition 1).

Theorem 6. *Let P be an lparse program, $Tr(P)$ be the program obtained by replacing each W in the body of rules in P by the strongly satisfiable encoding of W, and M be a set of atoms. M is an answer set for P iff M is an lparse stable model of $Tr(P)$.*

Example 5. Consider a program P with a single rule: $a \leftarrow 0[\texttt{not}\ a = 3]2$. Then, $Tr(P)$ 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 lparse-stable models of P, but $\emptyset$ is the only lparse-stable model of $Tr(P)$, which is also the only answer set for P. □

6 Logic Programs with Aggregates

Aggregates as weight constraints:

Aggregates are constraints on sets taking the form

$$aggr(\{X \mid p(X)\})\ \texttt{op}\ Result \tag{6}$$

where $aggr$ is an *aggregate function*, p is a predicate symbol, and X is a variable which takes value from a set $D(X) = \{a_1, ..., a_n\}$, called the *variable domain*. The standard aggregate functions are those in {SUM, COUNT, AVG, MAX, MIN}. The relational operator op is from $\{=, \neq, <, >, \leq, \geq\}$ and $Result$ is 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 $|Dom(A)|$. Let $M \subseteq Dom(A)$. M is a *model* of an aggregate A, denoted $M \models A$, if $aggr(\{a \mid p(a) \in M\})$ op $Result$ holds.

Let A be an aggregate in the form (6). A set of weight constraints $\{W_1, ..., W_n\}$ is an *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, where $M'_{|S}$ denotes $M' \cap S$.

We show the encodings of aggregates of the form (6), where the operator op is $\geq$. The encodings can be easily extended to other relational operator except for $\neq$ (more on $\neq$ later in this section). For example, aggregate $SUM(\{X \mid p(X)\}) > k$ can be encoded as $SUM(\{Y \mid p(Y)\}) \geq k + 1$.

The encodings work 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 $Result$ also needs to be processed correspondingly.

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, $SUM(\{X \mid p(X)\}) \geq k$ can be represented by

$$k\ [p(a_1) = a_1, ..., p(a_n) = a_n]. \tag{7}$$

We note that aggregates $COUNT(\{X \mid p(X)\}) \geq k$ and $AVG(\{X \mid p(X)\}) \geq k$ can be encoded simply by substituting the weights in (7) with 1 and $a_i - k$ (for AVG the lower bound k is also replaced by zero), respectively.

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

$$\sum_{i=1}^{n} (a_i - k + 1) > -\sum_{i=1}^{n} |a_i - k + 1|. \tag{8}$$

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.

$$0\ [p(a_i) = -1, p^+(a_i) = 1, p^-(a_i) = 1]\ 0,\ 1 \leq i \leq n \tag{9}$$

$$0\ [p(a_i) = -d_i, p^+(a_i) = d_i],\ 1 \leq i \leq n \tag{10}$$

$$0\ [p(a_i) = d_i, p^-(a_i) = -d_i],\ 1 \leq i \leq n \tag{11}$$

$$1\ [p(a_1) = d_1, p^+(a_1) = d_1, p^-(a_1) = -d_1, \\ ..., p(a_n) = d_n, p^+(a_n) = d_n, p^-(a_n) = -d_n] \tag{12}$$

$$1\ [p(a_1) = 1, ..., p(a_n) = 1] \tag{13}$$

where $d_i = a_i - k + 1$.

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

The constraints (9), (10) and (11) 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 (12) encodes the relation (8) and the constraint (13) guarantees that a model of $e(A)$ is not an empty set.

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

$$\sum_{i=1}^{n} (a_i - k) = \sum_{i=1}^{n} |a_i - k|. \tag{14}$$

Similar to MAX, the constraint in (14) can be encoded by weight constraints.

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 newly introduced literals $p^+(a_i)$, $p^-(a_i)$ and the literal $p(a_i)$. 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 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 the encodings.

For a given aggregate A, the constraints in $e(A)$ can be transformed to strongly satisfiable weight constraints. In the sequel, we assume $e(A)$ contains only strongly satisfiable weight constraints.

Programs with Aggregates to lparse Programs

A logic program with aggregates is a set of rules of the form $h \leftarrow A_1, ..., A_n$, where h is an atom and A_i are aggregates from {SUM, COUNT, AVG, MIN, MAX}.

We will represent a logic program with aggregates P by an lparse program, denoted $\tau(P)$, as follows:

1. For each rule of the above form in P, we have an lparse rule of the form

$$h \leftarrow r(A_1), ..., r(A_n) \tag{15}$$

   in $\tau(P)$. In the formula (15), we use $r(A_i)$ to denote the conjunction of all the weight constraints in $r(A_i)$, and
2. If there are newly introduced literals in the encoding of aggregates, the *auxiliary rule* of the form

$$W \leftarrow p(a_i) \tag{16}$$

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

Theorem 7. *Let P be a logic program with aggregates where the relational operator is not $\neq$. For any lparse-stable model M of $Tr(\tau(P))$, $M_{|At(P)}$ is an answer set for P (as defined in Definition 4). For any answer set M for P, there is an lparse-stable model M' of $Tr(\tau(P))$ such that $M'_{|At(P)} = M$.*

When an aggregate is encoded by a conjunction of weight constraints, logic equivalence leads to equivalence under conditional satisfaction. This is why in the encodings so far we only need to ensure that an encoding is satisfaction-preserving. But this is not the case when disjunction is involved, which causes problem in dealing with the relational operator $\neq$.

Example 6. Let $A = SUM(\{X\ |p(X)\}) \neq -1$, $A_1 = SUM(\{X\ |p(X)\}) > -1$ and $A_2 = SUM(\{X\ |p(X)\}) < -1$. 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. □

Since the answer set existence problem under the lparse semantics is *NP*-complete, and the transformation to strongly satisfiable programs is a polynomial time reduction, our transformation enables a computational mechanism for a large class of logic programs with aggregates whose complexity falls into *NP*-completeness.[4]

7 Experiments

We code logic programs with aggregates as lparse programs and use SMODELS 2.32 for the stable model computation. If a benchmark program is not already strongly satisfiable, it will be transformed into one, thus we can use the current implementation of SMODELS for our experiments.

We compare our approach with two systems, SMODELS^A and DLV^A. The lparse programs are run on Linux AS release 4 with 1GHz CPU and 512MB RAM. The reported execution time of SMODELS consists of the transformation time (from aggregates to weight constraints), the grounding time (calling to lparse), and the search (by SMODELS) time. The execution time of smodels^A consists of grounding time, search time and unfolding time (computing the solutions to aggregates). The execution time of DLV^A, includes the grounding time and search time (the grounding phase is not separated from the search in DLV^A).

We list the results reported in [4] and [2] for comparison. Thus, the comparison of the execution time is only indicative, since the platforms are similar but not the same.

Comparison with Smodels^A

We compare our approach to the unfolding approach implemented in the system SMODELS^A [4].[5]

[4] This is closely related to a result of [17], which shows that although the existence problem is in general in NP^{co-NP}, the same problem is in *NP* if neither $SUM(.) \neq k$ nor $AVG(.) \neq k$ is present in the program.

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

Table 1. Benchmarks used by SMODELS^A

Program	Sample Size	smodels	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	2.40	12.14
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	0.61	0.21
NM1	150	0.75	0.29
NM2	125	0.65	2.24
NM2	150	1.08	3.36

The aggregates used in the first and second set of problems (the company control and employee raise problems) are SUM; the third set of problems (the party invitation problems) are $COUNT$, and the fourth and fifth set 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 execution times are the average of one hundred 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 $COUNT(\{a|a \in S\}) \geq k$, SMODELS^A would list all *aggregate solutions* 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, SMODELS takes little time to compute the corresponding weight constraints.

Comparison with DLV^A

In [2] the seating problem was chosen to evaluate the performance of DLV^A[6]. 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 [2]. The results are shown in Table 2. The instance size is the number of atom occurrences in the ground programs. We report the result of the average over one hundred randomly generated instances for each problem size.

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

[6] The program contains disjunctive head, but it can be easily transformed to a non-disjunctive program.

Table 2. Seating

C	T	Execution Time		Instance Size	
		smodels	DLV^A	smodels	DLV^A
4	3	0.1	0.01	293	248
4	4	0.2	0.01	544	490
5	5	0.58	0.02	1213	1346
5	10	0.35	0.31	6500	7559
5	15	1.24	1.88	18549	22049
5	20	3.35	7.08	40080	47946
5	25	8.19	64.29	73765	88781
5	30	16.42	152.45	12230	147567

Table 3. Pigeon-hole

p	h	Execution Time		Instance Size	
		Lparse	Normal	Lparse	Normal
5	4	0.00	0.01	98	345
6	5	0.01	0.01	142	636
7	6	0.01	0.06	194	1057
8	7	0.09	0.49	254	1632
9	8	0.74	4.38	322	2385
10	9	6.89	43.66	398	3340
11	10	71.92	480.19	482	4521
12	11	827.85	5439.09	574	5952

Lparse Programs vs. Normal Programs for Global Constraints
Some global constraints can be encoded by weight constraints compactly. We have experimented with the pigeon-hole problem modeled by the *AllDifferent* constraint. The lparse program that encodes *AllDifferent* is about one order of magnitude smaller than the normal program encoding [13] in the size and the execution of the lparse program is 6-7 times faster than its normal program counterpart for hard unsatisfiable instances (where the number of holes is one less than the number of pigeons), which are run on the same machine under the default setting. See Table 3 for the results.

8 Conclusions and Future Work

We have shown that for a large class of lparse programs the lparse semantics coincides with the semantics based on conditional satisfaction. In general, answer sets admitted by the latter are all lparse-stable models. When an lparse-stable model is not an answer set, it may be circularly justified. We have proposed a transformation, by which an lparse program can be translated to another one, such that all lparse-stable models are answer sets and thus well-supported models.

As an issue of methodology, we have shown that most standard aggregates can be encoded by weight constraints and SMODELS can be applied to efficiently compute the answer sets of aggregate programs with almost all standard aggregates.

Our work has left some aspects unexplored. As we have shown, lparse-stable models that are not sanctioned by the semantics based on conditional satisfaction may or may not be circular under our definition of circular justification. This left the question of what would be the desired semantics for lparse programs unanswered. It seems that the notion of unfounded sets can serve as a basis for a new semantics for lparse programs, since it appears to separate the desired lparse-stable models from the undesired ones. Then, a question is whether a transformation exists that eliminates only circular models.

Among the types of aggregates proposed in the literature, the only ones that cannot be encoded compactly by weight constraints are the *product constraints*, such as $TIMES(\{X \mid p(X)\}) \geq k$, due to the non-linear nature of the expressions involved.

Plus the difficulty in encoding aggregates involving the relational operator $\neq$, this shows the limit of using SMODELS to run aggregate programs.

References

1. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex scheduling and placement problems. J. Mathematical and Computer Modelling 17(7), 57–73 (1993)
2. Armi, D., Faber, W., Ielpa, G.: Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in DLV*. In: IJCAI 2003, pp. 847–852 (2003)
3. Calimeri, F., Faber, W., Leone, N., Perri, S.: Declarative and computational properties of logic programs with aggregates. In: IJCAI 2005, pp. 406–411 (2005)
4. Elkabani, I., Pontelli, E., Son, T.C.: $Smodels^A$ – a system for computing answer sets of logic programs with aggregates. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS, vol. 3662, pp. 427–431. Springer, Heidelberg (2005)
5. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS, vol. 3229, pp. 200–212. Springer, Heidelberg (2004)
6. Fages, F.: Consistency of Clark's completion and existence of stable models. J. Methods of Logic in Computer Science 1, 51–60 (1994)
7. Ferraris, P.: Answer sets for propositional theories. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS, vol. 3662, pp. 119–131. Springer, Heidelberg (2005)
8. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 1–17. Springer, Heidelberg (2007)
9. Liu, L., Pontelli, E., Son, T.C., Truszczyński, M.: Logic programs with abstract constraint atoms: The role of computations. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 286–301. Springer, Heidelberg (2007)
10. Liu, L., Truszczyński, M.: Properties and applications of programs with monotone and convex constraints. J. Artificial Intelligence Research 7, 299–334 (2006)
11. Marek, V., Niemelä, I., Truszczyński, M.: Logic programs with monotone abstract constraint atoms. J. Theory and Practice of Logic Programming 8(2), 167–199 (2008)
12. Marek, V.W., Remmel, J.B.: Set constraints in logic programming. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS, vol. 2923, pp. 167–179. Springer, Heidelberg (2003)
13. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Math. and Artificial Intelligence 25(3-4), 241–273 (1999)
14. Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and stable semantics of logic programs with aggregates. J. Theory and Practice of Logic Programming 7, 301–353 (2007)
15. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence 138(1-2), 181–234 (2002)
16. Son, T.C., Pontelli, E., Tu, P.H.: Answer sets for logic programs with arbitrary abstract constraint atoms. J. Artificial Intelligence Research 29, 353–389 (2007)
17. Son, T.C., Pontelli, E.: A constructive semantic characterization of aggregates in answer set programming. J. Theory and Practice of Logic Programming 7, 355–375 (2006)
18. van Gelder, A., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. J. ACM 38(3), 620–650 (1991)
19. van Hoeve, W.-J., Katriel, I.: Global constraints. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, Ch. 7. Elsevier, Amsterdam (2006)
20. Wu, G., You, J., Lin, G.: Quartet based phylogeny reconstruction with answer set programming. IEEE/ACM Transactions on Computational Biology and Bioinformatics 4(1), 139–152 (2007)