

A Modular, Polynomial Time Method for Eliminating Weight Constraints

Paolo Ferraris

University of Texas at Austin, Austin TX 78712, USA otto@cs.utexas.edu

Abstract. Weight constraints represent an extension of logic programs under the answer set semantics that is essential for many applications to knowledge representation. The relation of this extension to the original semantics is an important topic. It has been proved that there exists a modular algorithm for eliminating weight constraints that is guaranteed to terminate in polynomial time under the assumption of a constant upper bound on the weights occurring in the program. In this paper we show that the statement of the theorem is true even without this assumption.

1 Introduction

The answer set (stable model) semantics for logic programs [Gelfond and Lifschitz, 1988] is an important formalism for knowledge representation and reasoning.

Several extensions to the original semantics have been invented to make the language more expressive. Among those, a construct called weight constraint [Niemelä *et al.*, 1999] has been used to encode problems in domains such as product configuration [Soininen *et al.*, 2001] and multi-unit combinatorial auctions [Baral and Uyan, 2001].

The question whether programs with weight constraints can be reduced to programs of a simpler form has been investigated by several authors. Initially, [Marek and Remmel, 2002] showed that this is the case for weight constraints where all weights are equal to 1 (cardinality constraints). Later, [Ferraris and Lifschitz, 2005] extended the result to arbitrary weight constraints, and, in the case of weights that are bounded by a constant, showed that the reduction can be done in polynomial time. These translations are modular: they transform each rule of the program independently. Still, the question of the existence of a modular and polynomial translation for programs with arbitrary weight constraints didn't have an answer.

Properties of logic programs in complexity theory tell us that a polynomial method for eliminating generic weight constraints must exist. However, we didn't know if there exists a modular polynomial-time translation. As an example of a similar situation, there is a way of eliminating negation as failure from programs consisting of rules of the form

$$a_1; \dots; a_n \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_p$$

$(n, m, p \geq 0)$ based on generating first the program’s answer sets. On the other hand, no modular translation can exist in this case, because of the monotonicity property of rules with $p = 0$.

This paper will show that a modular, polynomial method for eliminating weight constraints does actually exist. To this end, we propose a transformation from programs with arbitrary weight constraints to programs where all weight are 1 (programs with cardinality constraints) that it is not only sound and modular, but that can also be computed in a time that is polynomial in the size of its input. This shows that it is possible to completely eliminate weight constraints from a program in polynomial time by applying first the translation proposed in this paper, and then the polynomial-time method for eliminating cardinality constraints given in [Ferraris and Lifschitz, 2005].

After a review of the syntax of programs with weight and cardinality constraints (Sect. 2), we give the definition of the translation in Sect. 3, along with the statement of our main theorem. The proof of the fact that the translation is polynomial is given in Sect. 4. In Sect. 5 we review some properties of programs with nested expressions that are used in the proof of the soundness of the translation (Sect. 6).

2 Programs with weight constraints

In this section we reproduce the definition of the syntax of rules with weight constraints from [Niemelä *et al.*, 1999] with minor modifications, convenient for our purposes. We limit attention to the case of integer, positive weights and bounds.

A *set expression* is an expression of the form

$$\{a_1 = w_1, \dots, a_m = w_m\}, \quad (1)$$

where a_1, \dots, a_m ($m \geq 0$) are atoms (the atoms of the set expression), and w_1, \dots, w_m are positive integers (weights). We will sometimes identify a set expression (1) with the multiset $\{a_i = w_i : 1 \leq i \leq m\}$. We also usually omit the “= 1” in pairs $a = 1$ that belong to a set expressions.

A *weight constraint* is an expression of the form $N \leq S$, where S is a set expression and N — the bound of the constraint — is a positive integer.

Finally, a *program with weight constraints* is a set of rules of the form

$$H \leftarrow C_1, \dots, C_m, \text{not } C_{m+1}, \dots, \text{not } C_n \quad (2)$$

where C_1, \dots, C_n ($0 \leq m \leq n$) are weight constraints, and H is a , $\{a\}$ or \perp , for some atom a . If $n = 0$, rule (2) is written as H . Expressions of the form $\text{not } C_i$ in the body of a rule (2) have the same meaning as “upper bound” constraints in [Niemelä *et al.*, 1999]: we treat $\text{not } (N \leq S)$ as synonymous with $S \leq (N - 1)$. (This convention is acceptable because we assume all weights and bounds to be integers.) Except for this detail, we adopt the same definition of an answer set for programs with weight constraints as in [Niemelä *et al.*, 1999].

The syntax introduced in that paper is more general than what is allowed by our definition because it allows the head H of a rule to be an arbitrary weight constraint. But these more general heads can be eliminated from a program by simple equivalent transformations (as done, for instance, in the process of operation of LPARSE). In fact, even the list of three possible forms of H in the definition above (a , $\{a\}$ and \perp) contains a redundancy: a rule of the form

$$a \leftarrow C_1, \dots, C_m, \text{not } C_{m+1}, \dots, \text{not } C_n$$

can be equivalently replaced with two rules

$$\begin{aligned} \{a\} &\leftarrow C_1, \dots, C_m, \text{not } C_{m+1}, \dots, C_n \\ \perp &\leftarrow C_1, \dots, C_m, \text{not } (1 \leq \{a\}), \text{not } C_{m+1}, \dots, C_n \end{aligned}$$

without changing the program's answer sets.

The syntax of rules in [Niemelä *et al.*, 1999] is more general also in the sense that it allows negated atoms in set expressions. Occurrences of negation inside set expressions can be eliminated using additional atoms, without a significant change in the size of the program.

Cardinality constraints are weight constraints where all weights are 1. Similarly, *programs with cardinality constraints* are programs with weight constraints where all weights are 1. A cardinality constraint of the form $1 \leq \{a\}$ in the body of a rule can be abbreviated as a . Given this convention, $p \leftarrow \text{not } q$ stands for $p \leftarrow \text{not } (1 \leq \{q\})$. This shows that programs with cardinality constraints are a generalization of programs as defined in [Gelfond and Lifschitz, 1988].

Intuitively, a weight constraint $N \leq S$ expresses the condition that the sum of the weights associated to the atoms that are “true” in S is not lower than N . A rule (2) “justifies” the truth of H if C_1, \dots, C_m are true and C_{m+1}, \dots, C_n are false. If H is an atom, then that atom is justified; if H has the form $\{a\}$, then a is free to be true or false; finally, if $H = \perp$, this indicates a contradiction. It can be shown, for instance, that the program

$$\begin{aligned} p &\leftarrow \text{not } (2 \leq \{r = 3, q = 1\}) \\ q & \\ r &\leftarrow \text{not } p \end{aligned} \tag{3}$$

has two answer sets: $\{p, q\}$ and $\{q, r\}$.

3 Removing the weights

The process of eliminating general weights constraints in favor of cardinality constraints consists in replacing each weight constraint C with a cardinality constraint — we call it $\text{card}(C)$ — that may include auxiliary atoms. New rules are then added to the program to “define” such new atoms.

Let σ be the signature of the original program (the set of atoms that occur in its rules). We define an (extended) signature σ^* recursively as follows:

- if $p \in \sigma$ then $p \in \sigma^*$, and
- for every set expression S over σ^* with all weights equal to 1, and every finite set of positive integers X , the (“auxiliary”) atom $X \preceq S$ is in σ^* .

The signature of the translation that we will define is actually smaller than σ^* : it consists only of atoms from σ and atoms of the form $\{N\} \preceq S$, so that atoms of the form $X \preceq S$ for nonsingleton X 's may occur inside other atoms only. The presence of such atoms allows us to keep the computation time of the translation small.

The idea behind $\text{card}(C)$ is to recursively “halve” the weights in C . Consider the following definitions. If S is a set expression 1, then by H_S (“half”) we denote the set expression

$$\{a_i = \lfloor w_i/2 \rfloor : 1 \leq i \leq m, w_i > 1\},$$

and by R_S (“remainder”) the set expression

$$\{a_i : 1 \leq i \leq m, w_i \text{ is odd}\}.$$

Note that $N \leq S$ is basically the same as $N \leq H_S \cup H_S \cup R_S$, which already has the maximum weight halved. We want to “simplify” this weight constraint by halving N , taking only one occurrence of H_S , and using some tricks to halve the contribution of R_S .

For any set expression S of the form (1), by $|S|$ we denote m , and by \overline{S} we denote S with all the elements of the form $(\{N_1, \dots, N_k\} \preceq S') = w$ replaced by the following k elements:

$$(\{N_1\} \preceq S') = w, \dots, (\{N_k\} \preceq S') = w.$$

For any positive integer N , by $X_{N \leq S}$ we denote the set of integers

$$\{i : 0 < i \leq \overline{R_S}, i + N \text{ is even}\}.$$

Now we can define the function card from weight constraints over σ^* to cardinality constraints:

$$\text{card}(N \leq S) = \begin{cases} N \leq \overline{S} & \text{if } H_S = \emptyset, \text{ and} \\ \text{card}(\lceil N/2 \rceil \leq (H_S \cup \{X_{N \leq S} \preceq R_S\})) & \text{otherwise.} \end{cases}$$

Assume, for instance, that $N \leq S$ is

$$2 \leq \{r = 3, q = 1\}.$$

Since $H_S = \{r\}$, $R_S = \{r, q\}$, $\overline{R_S} = 2$ and $X_{N \leq S} = \{2\}$,

$$\text{card}(2 \leq \{r = 3, q = 1\}) = \text{card}(1 \leq \{r, \{2\} \preceq \{r, q\}\}).$$

Finally, since each of the two weights in $1 \leq \{r, \{2\} \preceq \{r, q\}\}$ is 1, we can conclude that

$$\text{card}(2 \leq \{r = 3, q = 1\}) = 1 \leq \{r, \{2\} \preceq \{r, q\}\}. \quad (4)$$

For any program Ω , by $\text{card}(\Omega)$ we denote the program consisting of the rules

$$H \leftarrow \text{card}(C_1), \dots, \text{card}(C_m), \text{not card}(C_{m+1}), \dots, \text{not card}(C_n)$$

for all rules (2) in Ω . By S_Ω we denote the set expression consisting of the auxiliary atoms that occur (possibly as subexpressions of other atoms) in $\text{card}(\Omega)$.

We are now ready to define our translation. Given a program Ω with weight constraints, the corresponding program Ω_c is defined as the union of $\text{card}(\Omega)$ with the set of rules

$$\{N\} \preceq S \leftarrow N \leq \overline{S} \quad (5)$$

for all atoms $N \preceq S$ in $\overline{S_\Omega}$.

For instance, if Ω is (3) then $\text{card}(\Omega_c)$ is

$$\begin{aligned} p &\leftarrow \text{not card}(\{2\} \leq \{r = 2, q = 1\}), \\ q \\ r &\leftarrow \text{not card}(\{1\} \leq \{p\}) \end{aligned}$$

(recall that the last p in (3) stands for $1 \leq \{p\}$). By (4), we can rewrite these rules as

$$\begin{aligned} p &\leftarrow \text{not} (1 \leq \{r, \{2\} \preceq \{r, q\}\}), \\ q \\ r &\leftarrow \text{not} (1 \leq \{p\}) \end{aligned} \quad (6)$$

The set S_Ω consists of the only auxiliary atom $\{2\} \preceq \{r, q\}$ that occurs in (6). In this case, $\overline{S_\Omega} = S_\Omega$, so that we can conclude that Ω_c consists of rules (6) and the following rule of the form (5):

$$\{2\} \preceq \{r, q\} \leftarrow 2 \leq \{r, q\} \quad (7)$$

The following theorem is the main result of this paper.

Theorem 1. *For any logic program Ω with weight constraints over a signature σ ,*

- (a) *the mapping $Z \mapsto Z \cap \sigma$ is a 1-1 correspondence between the answer sets of Ω_c and the answer sets of Ω , and*
- (b) *Ω_c is the union of the programs $\{r\}_c$ for all rules $r \in \Omega$.*

Furthermore, if Ω is finite,

- (c) *Ω_c can be computed in time polynomial in the size of Ω .*

As an example illustrating part (a), if Ω is program (3) then Ω_c has the following two answer sets:

$$\{q, r, \{2\} \preceq \{r, q\}\} \text{ and } \{p, q\},$$

If we consider the atoms over σ of those two answer sets, we get the two answer sets $\{q, r\}$ and $\{p, q\}$ of (3).

Part (b) expresses that the translation is modular: to apply it to a program that consists of several rules, we apply it to each rule separately. This property is immediate from the definition of Ω_c . Part (c) is proven in next section. The proof for part (a), which refers to programs with nested expressions (Sect. 5), is given in Sect. 6.

4 Proof of Theorem 1(c)

The size of every atom, set expression, weight constraint, rule or program P as a string is denoted by $s(P)$. For any set expression S of the form (1), we denote by $w(S)$ the length of the binary representation of the largest among w_1, \dots, w_m , or 1 if $m = 0$. It is clear that $|S|$, $|\overline{S}|$ and $w(S)$ are smaller than $s(S)$. This fact allow us to use those parameters when we prove upper bounds polynomial in the size of S . We start with some properties that easily follow from definitions.

Lemma 1. *for any set expression S ,*

- (i) $s(R_S) \leq s(S)$,
- (ii) $w(H_S) = \max(w(S) - 1, 1)$,

We define the sequence $N_0 \leq S_0, N_1 \leq S_1, \dots$ recursively as follows:

$$N_0 = N, S_0 = S, \\ N_{i+1} = \lceil N_i/2 \rceil \text{ and } S_{i+1} = H_{S_i} \cup \{X_{N_i \leq S_i} \preceq R_{S_i}\}.$$

An important property of such sequence is stated in next lemma.

Lemma 2. $\text{card}(N \leq S) = N_{w(S)-1} \leq \overline{S_{w(S)-1}}$.

Lemma 3. *For each $i \geq 0$,*

- (i) $N_i \leq N$,
- (ii) $s(H_{S_i}) \leq s(S)$,
- (iii) $|\overline{R_{S_i}}| \leq 2|\overline{S}|$.

In the next two lemmas, by “small” we mean “bounded by a polynomial in the size of $N \leq S$.”

Lemma 4. *For every $i = 0, \dots, w(S)$, $s(N_i \leq S_i)$ is small.*

Proof. Clearly, each $s(N_i)$ is small by Lemma 3(i). It remains to show that each $s(S_i)$ is small. Since $S_0 = S$ and $i \leq w(S)$, it is sufficient to show that $s(S_{i+1}) - s(S_i)$ is small. The set expression S_{i+1} , as a string, consists of H_{S_i} , $X_{N_i \leq S_i}$ and R_{S_i} . The first term is small by Lemma 3(ii); the second is small because it consists of distinct integers between 1 and $|\overline{R_S}|$, and $|\overline{R_S}|$ is small by Lemma 3(iii). Consequently, each $s(S_{i+1}) - s(R_{S_i})$ is bounded by a polynomial in $s(S)$. In view of Lemma 1(i), the same polynomial bounds $s(S_{i+1}) - s(S_i)$. \square

Lemma 5. *The time needed to compute $\text{card}(N \leq S)$ is small.*

Proof. By Lemma 2, $\text{card}(N \leq S)$ can be obtained by computing each element of the sequence

$$N_1 \leq S_1, N_2 \leq S_2, \dots, N_{w(S)-1} \leq S_{w(S)-1}$$

from the previous one (in the case of $N_1 \leq S_1$, from $N_0 \leq S_0$). It remains to notice that the time needed to complete each of these $w(S) - 1$ steps is small, in view of Lemma 4 and the fact that $N_{i+1} \leq S_{i+1}$ is computed from $N_i \leq S_i$ in a time polynomial in the size of $N_i \leq S_i$. \square

Proof of Theorem 1(c). By “fast” we mean “computable in a time polynomial in the size of its input”. The translation consists of computing (i) $\text{card}(\Omega)$ from Ω , (ii) S_Ω from $\text{card}(\Omega)$, (iii) \overline{S}_Ω from S_Ω and (iv) the rules (5) of Ω_c from \overline{S}_Ω . It is sufficient to show that each of those four step is fast.

The first step is fast by Lemma 5. Also step (ii) is fast. The last two steps are both fast in view of the fact that, for every set expression S , \overline{S} can be computed in a time polynomial in $s(S)$. \square

5 Weight constraints as nested expressions

We assume that the reader is familiar with the syntax, the semantics, and the property of strong equivalence for programs with nested expressions, and also with the logic of here-and-there, as reviewed in Sects. 2.1, 2.2, 2.4 of [Ferraris and Lifschitz, 2005]. The same paper showed how programs with weight constraints can be expressed in terms of programs with nested expressions. In this section we review that translation in the form adapted to the syntax of weight constraints used in this paper. It is convenient not to require that the bounds in constraints be positive.

By $[N \leq \{a_1 = w_1, \dots, a_m = w_m\}]$ we denote the formula

$$\underset{X \subseteq \{1, \dots, m\} : N \leq W_X}{\dot{\vee}} \left(\underset{j \in X}{\dot{\wedge}} a_j \right), \quad (8)$$

where $W_X = \sum_{i \in X} w_i$, and the symbols $\dot{\vee}$ and $\dot{\wedge}$ denote a multiple disjunction and a multiple conjunction (an empty conjunction is \top , and an empty disjunction is \perp).

Finally, we define a program $[\Omega]$ with nested expressions, consisting, for each rule (2) in Ω , of rule

$$\underline{H} \leftarrow [C_1], \dots, [C_m], \text{not } [C_{m+1}], \dots, \text{not } [C_n] \quad (9)$$

where \underline{H} stands for a ; *not* a if H has the form $\{a\}$, and H otherwise.

Next Lemma follows from Theorem 1 of [Ferraris and Lifschitz, 2005].

Lemma 6. *$[\Omega]$ and Ω have the same answer sets.*

In the proof of Theorem 1(a), we will also use two properties of programs with nested expressions, established in [Erdoğan and Lifschitz, 2004] and [Ferraris and Lifschitz, 2005] respectively. The expression $F \leftrightarrow G$ denotes the pair of rules $F \leftarrow G, G \leftarrow F$; we also use it meta-mathematically, to express that F is equivalent to G in the logic of here-and-there.

Lemma 7. (*Lemma on Explicit Definitions*) *Let Π be a program with nested expressions, and let Q be a set of atoms that do not occur in Π . For every $q \in Q$, let $\text{Def}(q)$ be a formula with no occurrences of atoms from Q in the scope of negation. Then $Z \mapsto Z \setminus Q$ is a 1-1 correspondence between the answer sets for $\Pi \cup \{q \leftarrow \text{Def}(q) : q \in Q\}$ and the answer sets for Π .*

Lemma 8. (*Completion Lemma*) *Let Π be a program with nested expressions, and let Q be a set of atoms that do not occur in the heads of the rules of Π . For every $q \in Q$, let $\text{Def}(q)$ be a formula. Then the program*

$$\Pi \cup \{q \leftarrow \text{Def}(q) : q \in Q\}$$

has the same answer sets as the program

$$\Pi \cup \{q \leftrightarrow \text{Def}(q) : q \in Q\}.$$

Notice that atoms of the signature σ^* (Sect. 3) may occur in other atoms of the same signature. In application to signatures like this, the concept of an “occurrence” in the statements of Lemmas 7 and 8 should be understood, of course, as “maximal occurrence” (that is, an occurrence that is not a subexpression of another atom).

6 Proof of Theorem 1(a)

By σ^+ we denote the union of σ with the set of atoms of the form $\{N\} \preceq S$ of σ^* , so that $\sigma \subseteq \sigma^+ \subseteq \sigma^*$. Note that the signature of Ω_c is contained in σ^+ . By N we denote an integer, and, in this section, we usually write auxiliary atoms of the form $\{N\} \preceq S$ as $N \preceq S$.

We recursively define, for each formula F over σ^+ , F^* to be the formula obtained from F by replacing each maximal occurrence of an atom $N \preceq S$ by $[N \preceq \bar{S}]^*$. Since each recursive step removes one level of nesting in auxiliary atoms, the transformation $F \mapsto F^*$ is clearly total.

Now we state some properties of this transformation. The proofs are omitted for lack of space. Entailment, equivalence properties in this section are under the logic of here-and-there.

Lemma 9. *For any two integers N_1 and N_2 with $N_1 \geq N_2$, and any set expression S , $(N_1 \preceq S)^*$ entails $(N_2 \preceq S)^*$.*

Lemma 10. *For any weight constraint $N \leq S$ where N and all weights in S are even, $(N \preceq S)^* = ((N - 1) \preceq S)^*$.*

Lemma 11. Let N_1, \dots, N_n ($n \geq 1$) be integers such that $N_1 < N_2 < \dots < N_n$. For every integer $i = 1, \dots, n$ and any set expression S ,

$$(i \preceq \{\{N_1, \dots, N_n\} \preceq S\})^* \leftrightarrow (N_i \preceq S)^*.$$

For any set expression S , we denote by ΣS the sum of the weights in \bar{S} .

Lemma 12. For any set expressions S_1 and S_2 , and any integer N ,

$$(N \preceq (S_1 \cup S_2))^* \leftrightarrow \bigvee_{i=0, \dots, \Sigma S_1} ((i \preceq S_1)^*, (N - i \preceq S_2)^*).$$

Lemma 13. For every formula $(N \preceq S)^*$, replacing two elements $a = w_1$ and $a = w_2$ in S with a single element $a = w_1 + w_2$ preserves strong equivalence.

Now we are ready to state some properties related to the translation. We denote by $P_{N \leq S}$ the set expression $\{X_{N \leq S} \preceq R_S\}$ used in the recursive definition of $\text{card}(N \leq S)$.

Lemma 14. For every integer i and every weight constraint $N \leq S$,

$$(\lceil N/2 \rceil - i \preceq P_{N \leq S})^* \leftrightarrow (N - 2i \preceq R_S)^*. \quad (10)$$

Proof. The proof is by cases. For the second and third case, for simplicity, we consider N to be even (the proofs for N that is odd are similar).

Case 1: $\lceil N/2 \rceil - i \leq 0$. It follows that $2\lceil N/2 \rceil - 2i \leq 0$ and finally that $N - 2i \leq 0$. Consequently both $\lceil N/2 \rceil - i \preceq \overline{P_{N \leq S}}$ and $N - 2i \preceq \overline{R_S}$ equal \perp , and the same holds for both sides of (10).

Case 2: $\lceil N/2 \rceil - i > \overline{P_{N \leq S}}$. That means that

$$2(\lceil N/2 \rceil - i) > 2\overline{P_{N \leq S}} + 1,$$

so that

$$N - 2i = 2\lceil N/2 \rceil - 2i > 2\overline{P_{N \leq S}} + 1 \geq \overline{R_S}.$$

Consequently both $\lceil N/2 \rceil - i \preceq \overline{P_{N \leq S}}$ and $N - 2i \preceq \overline{R_S}$ are equivalent to \top , and the same holds for both sides of (10).

Case 3: $1 \leq \lceil N/2 \rceil - i \leq \overline{P_{N \leq S}}$. Since $P_{N \leq S}$ is

$$\{\{2, 4, \dots, 2\lfloor \overline{R_S} \rfloor / 2\} \preceq R_S\},$$

then, by Lemma 11,

$$\begin{aligned} (\lceil N/2 \rceil - i \preceq P_{N \leq S})^* &\leftrightarrow (2\lceil N/2 \rceil - 2i \preceq R_S)^* \\ &= (N - 2i \preceq R_S)^*. \end{aligned}$$

□

Lemma 15. For every weight constraint C over σ , $[\text{card}(C)]^* \leftrightarrow [C]$.

Proof. We shall prove the following more general claim: for every weight constraint $N \leq S$ over σ^* ,

$$[\text{card}(N \leq S)]^* \leftrightarrow [N \leq \overline{S}]^*.$$

(note that if S is over σ , $\overline{S} = S$). The proof is by strong induction on $w(S)$ (defined at the beginning of Sect. 4). If $w(S) = 1$ then $H_S = \emptyset$ and it is sufficient to notice that $\text{card}(N \leq S) = N \leq \overline{S}$. Otherwise, by induction hypothesis,

$$[\text{card}(\lceil N/2 \rceil \leq H_S \cup P_{N \leq S})]^* \leftrightarrow [\lceil N/2 \rceil \leq \overline{H_S \cup P_{N \leq S}}]^*$$

(we know that $w(H_S) = w(S) - 1 < w(S)$ by Lemma 1(ii) and that $w(P_{N \leq S}) = 1 < w(S)$). Since

$$\text{card}(\lceil N/2 \rceil \leq H_S \cup P_{N \leq S}) = \text{card}(N \leq S),$$

it remains to show that

$$[\lceil N/2 \rceil \leq \overline{H_S \cup P_{N \leq S}}]^* \leftrightarrow [N \leq \overline{S}]^*,$$

which can be rewritten as

$$(\lceil N/2 \rceil \preceq H_S \cup P_{N \leq S})^* \leftrightarrow (N \preceq S)^*.$$

Let F denote $(\lceil N/2 \rceil \preceq H_S \cup P_{N \leq S})^*$. By Lemmas 12 and 14,

$$\begin{aligned} F &\leftrightarrow \bigvee_{i=0, \dots, \Sigma H_S} ((i \preceq H_S)^*, (\lceil N/2 \rceil - i \preceq P_{N \leq S})^*) \\ &\leftrightarrow \bigvee_{i=0, \dots, \Sigma H_S} ((i \preceq H_S)^*, (N - 2i \preceq R_S)^*). \end{aligned}$$

Let T_S be H_S with all weights doubled. It is easy to see that $(i \preceq H_S)^* = (2i \preceq T_S)^*$. Then

$$\begin{aligned} F &\leftrightarrow \bigvee_{i=0, \dots, \Sigma H_S} ((2i \preceq T_S)^*, (N - 2i \preceq R_S)^*) \\ &\leftrightarrow \bigvee_{i=0, 2, \dots, \Sigma T_S - 2, \Sigma T_S} ((i \preceq T_S)^*, (N - i \preceq R_S)^*). \end{aligned}$$

The above disjunction does not include disjunctive terms for values of i that are odd. However, for each $i = 2, 4, \dots, \Sigma T_S$, the disjunctive term corresponding to $i - 1$ entails the one corresponding to i . Indeed, first of all, $((N - (i - 1)) \preceq R_S)^*$ entails $(N - i \preceq R_S)^*$ by Lemma 9; secondly,

$$(i - 1 \preceq T_S)^* = (i \preceq T_S)^*$$

by Lemma 10. By Lemma 12, it follows that

$$F \leftrightarrow \bigvee_{i=0,1,\dots,\Sigma T_S} ((i \preceq T_S)^*, (N - i \preceq R_S)^*) \\ \leftrightarrow (N \preceq T_S \cup R_S)^*.$$

It remains to notice that, by Lemma 13 and the definition of T_S and R_S ,

$$(N \preceq T_S \cup R_S)^* \leftrightarrow (N \preceq S)^*.$$

□

Let Γ be the set of rules

$$\{N \preceq S \leftrightarrow [N \leq \bar{S}] : (N \preceq S) \in \bar{S}_\Omega\}.$$

Lemma 16. *For every $d \in \bar{S}_\Omega$, Γ entails $d \leftrightarrow d^*$.*

Proof. Assume Γ . Let d be $N \preceq S$. First of all, note that all auxiliary atoms that belong to \bar{S} are elements of \bar{S}_Ω . Indeed, it is sufficient to notice that if $N \preceq S \in \bar{S}_\Omega$ then some $X \preceq S$ with $N \in X$ occurs in S_Ω , so that every auxiliary atom that occur in S is in S_Ω .

For this reason, we assume, as a structural induction hypothesis, that for each auxiliary atom $N' \preceq S'$ that belongs to \bar{S} , $N' \preceq S' \leftrightarrow (N' \preceq S')^*$. By replacing every of such atoms $N' \preceq S'$ by the equivalent formula $(N' \preceq S')^*$, $[N \leq \bar{S}]$ becomes $(N \preceq S)^*$. It remains to notice that $N \preceq S \leftrightarrow [N \leq \bar{S}]$ belongs to Γ . □

Lemma 17. *Γ entails $[\Omega] \leftrightarrow [card(\Omega)]$.*

Proof. Assume Γ . Note that $[card(\Omega)]$ can be seen as $[\Omega]$ with all the expressions of the form $[C]$ replaced by $[card(C)]$. It is then sufficient to show that, for each of those C , both $[C]$ and $[card(C)]$ are equivalent to $[card(C)]^*$. Indeed, $[C] \leftrightarrow [card(C)]^*$ by Lemma 15, and $[card(C)]^* \leftrightarrow [card(C)]$ by Lemma 16 since we assumed Γ and all auxiliary atoms in $card(C)$ are elements of \bar{S}_Ω . □

Proof of Theorem 1(a).

Program $[\Omega_e]$ can be written as

$$[card(\Omega)] \cup \{N \preceq S \leftrightarrow [N \leq \bar{S}] : (N \preceq S) \in \bar{S}_\Omega\}. \quad (11)$$

Since the heads of rules of $[card(\Omega)]$ do not contain auxiliary atoms, by Lemma 8 it follows that (11) has the same answer sets of

$$[card(\Omega)] \cup \Gamma,$$

which, by Lemma 17, is strongly equivalent to

$$[\Omega] \cup \Gamma.$$

Consequently, since $[\Omega]$ is over σ , by Lemma 8 we have that $[\Omega_c]$ has the same answer sets of

$$[\Omega] \cup \{N \preceq S \leftarrow [N \leq \bar{S}] : (N \preceq S) \in \overline{S_\Omega}\}.$$

Since $[N \leq \bar{S}]$ does not contain negation, we can apply Lemma 7, and we can conclude that $Z \mapsto Z \cap \sigma$ is a 1–1 correspondence between the answer sets of $[\Omega_c]$ and $[\Omega]$. By Lemma 6, the same correspondence is between the answer sets of Ω_c and Ω . \square

7 Conclusion

We showed that it is possible to turn a logic program with weight constraints and integer weights into a program with cardinality constraints in a modular way and in polynomial time. A simple modular translation consists in replacing each element of the form $a = w$ in every set expression by w copies of the term $a = 1$. Unfortunately, this translation is not polynomial. The fact that our translation and the one that removes cardinality constraints [Ferraris and Lifschitz, 2005] are polynomial in time shows that arbitrary weight constraints can be removed in polynomial time.

Our translation does not preserve the answer sets of the original program (as it happens in the modular translations discussed in [Ferraris, 2005]), but it “conservatively extends” them by introducing in the answer sets atoms that are not part of the original signature. This is also the case for the translation from [Ferraris and Lifschitz, 2005] mentioned above. Introducing auxiliary atoms in the process of eliminating cardinality constraints is inevitable even if we put aside the modularity conditions. Indeed, the answer sets of a program in the sense of [Gelfond and Lifschitz, 1988] have the anti-chain property: none of such programs can have the answer sets \emptyset and $\{p\}$ that program with cardinality constraints $\{p\}$ has.

The process of eliminating weight constraints proposed in [Ferraris and Lifschitz, 2005] is used in the design of the answer set solver CMODELS ¹ We hope that the ideas of this paper can be used to improve the performance of CMODELS on programs containing large weights. Experiments show, however, that such an improvement cannot be achieved by simply using the method described in this paper as a preprocessing step. Perhaps “merging” a modified version of this method with elimination of cardinality constraints can be useful.

Acknowledgments

I am grateful to Selim Erdoğan, Gene Moo Lee, Joohyung Lee, and Wanwan Ren for comments on this draft. Special thanks go to Vladimir Lifschitz for many comments and discussions on the topic, and his careful reading of this paper. This research was partially supported by the National Science Foundation under Grant IIS-0412907.

¹ <http://www.cs.utexas.edu/users/tag/cmodels.html> .

References

- [Baral and Uyan, 2001] Chitta Baral and Cenk Uyan. Declarative specification and solution of combinatorial auctions using logic programming. *Lecture Notes in Computer Science*, 2173:186–199, 2001.
- [Erdoğan and Lifschitz, 2004] Selim T. Erdoğan and Vladimir Lifschitz. Definitions in answer set programming. In Vladimir Lifschitz and Ilkka Niemelä, editors, *Proceedings of 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7)*, pages 114–126, 2004.
- [Ferraris and Lifschitz, 2005] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5:45–74, 2005.
- [Ferraris, 2005] Paolo Ferraris. On modular translations and strong equivalence.² Submitted to the same conference, 2005.
- [Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080, 1988.
- [Marek and Remmel, 2002] Victor Marek and Jeffrey Remmel. On logic programs with cardinality constraints. In *Proc. NMR-02*, pages 219–228, 2002.
- [Niemelä *et al.*, 1999] Ilkka Niemelä, Patrik Simons, and Timo Soininen. Stable model semantics for weight constraint rules. In *Logic Programming and Non-monotonic Reasoning: Proc. Fifth Int’l Conf. (Lecture Notes in Artificial Intelligence 1730)*, pages 317–331, 1999.
- [Soininen *et al.*, 2001] Timo Soininen, Ilkka Niemelä, Juha Tiihonen, and Reijo Sulonen. Representing configuration knowledge with weight constraint rules. In *Proc. AAAI Spring Symp. on Answer Set Programming: Towards Efficient and Scalable Knowledge*, 2001.

² <http://www.cs.utexas.edu/users/otto/papers/modular.ps> .