Fundamenta Informaticae 106 (2011) 25–43
DOI 10.3233/FI-2011-375
IOS Press

Level Mapping Induced Loop Formulas for Weight Constraint and Aggregate Logic Programs

Guohua Liu and Jia-Huai You*
Department of Computing Science
University of Alberta
Edmonton, Canada
guohua, you@cs.ualberta.ca

Abstract. Level mapping and loop formulas are two different means to justify and characterize answer sets for normal logic programs. Both of them specify conditions under which a supported model is an answer set. Though serving a similar purpose, in the past the two have been studied largely in isolation with each other. In this paper, we study level mapping and loop formulas for weight constraint and aggregate (logic) programs. We show that, for these classes of programs, loop formulas can be devised from level mapping characterizations. First, we formulate a level mapping characterization of stable models and show that it leads to a new formulation of loop formulas for arbitrary weight constraint programs, without using any new atoms. This extends a previous result on loop formulas for weight constraint programs, where weight constraints contain only positive literals. Second, since aggregate programs are closely related to weight constraint programs, we further use level mapping to characterize the underlying answer set semantics based on which we formulate loop formulas for aggregate programs. The main result is that for aggregate programs not involving the inequality comparison operator, the dependency graphs can be built in polynomial time. This compares to the previously known exponential time method.

Keywords: Logic programs, answer sets, weight constraints, aggregates, loop formulas.

1. Introduction

Logic programming under the stable model semantics [10] has been extended to incorporate a variety of constraints to facilitate knowledge representation and reasoning. These constraints include weight

*Address for correspondence: Department of Computing Science, University of Alberta, Edmonton, Canada

constraints [23, 26, 27], aggregates [2, 5, 24, 28], and abstract constraints [20, 21, 25, 29]. We refer to logic programs with these constraints as weight constraint, aggregate, and abstract constraint programs, respectively. Much research has been done on the application and implementations of weight constraint and aggregate programs [3, 8, 11, 14, 18, 26].

It is known that every stable model of a normal logic program is a supported model of the program, but the converse is not true in general. Loops are the reasons that supported models may not be stable models. The approach to computing stable models of a normal program, proposed by Lin and Zhao [15], is to compute supported models of the program, namely the models of the completion [1], and then eliminate those that are not stable models, where loop formulas are used to prune the search space. Loop formulas are propositional formulas that capture the logical conditions under which the atoms in a loop can be in a stable model. Given a program, the loop formulas are formulated in two steps: construct the dependency graph of the given program and establish a formula for each loop in the graph.

Liu and Truszczyński [18] extend the approach to weight constraint programs where the weight constraints contain only positive literals and weights. They define loop formulas for such programs. In general, an arbitrary weight constraint can be transformed to one that contains only positive literals and weights [18, 19]. However, such a transformation introduces new atoms. In theory, new atoms enlarge the search space for stable model computation. An interesting question is whether the loop formulas for arbitrary weight constraint programs can be formulated without using extra atoms.

The method of level mapping has been studied to characterize stable models [4, 6, 22, 32]. We observe that such a characterization is closely related to the formulation of loop formulas. From the level mapping point of view, a loop formula can be satisfied by a supported model of a program if an atom in the loop can be derived by the atoms that are not in the loop and have a strictly lower level. Despite this close relationship, in the past level mapping and loop formulas have been studied largely in isolation.

Our interest in this paper is to show how one can define loop formulas based on a formulation of level mapping, for weight constraint and aggregate programs. Since level mapping is only a characterization whereas the loop formula method constitutes a construction, the former is conceptually simpler than the latter. Therefore, it is in general relatively easier to define a level mapping, which can serve as a basis for the formulation of loop formulas. In this paper, we follow this clue by first presenting a level mapping characterization of the stable model semantics for weight constraint programs and show that this characterization leads to a new formulation of loop formulas for arbitrary weight constraint programs, without introducing extra atoms.

Aggregate programs are closely related to weight constraint program, since many aggregates can be encoded by weight constraints [17]. There are different semantics proposed for aggregate programs [5, 7, 24, 28]. Among them, the semantics based on the notion of *conditional satisfaction* is considered the most conservative [29], in the sense that any answer set under this semantics is an answer set under others, but the reverse may not hold. We are interested in the formulation of loop formulas for this semantics. To distinguish from the stable model semantics of weight constraint programs, we call the semantics the *answer set semantics*[1].

Loop formulas for the answer set semantics are presented in [31]. In the approach, given a program, the construction of the dependency graph requires computing what is called ”local power set” for the constraints in the program, to capture conditional satisfaction. In the worst case, the process takes an exponential time in the size of the program. The question that we address in this paper is whether the

[1]In the literature, stable model and answer set are usually interchangeable. But in this paper we use them for different semantics.

construction of dependency graph can be done in polynomial time for aggregate programs. Again, we tackle the problem by means of level mapping.

Son et al. [29] give a level mapping characterization of the answer set semantics for abstract constraint programs. To capture the conditional satisfaction of a constraint, the definition of level of a constraint needs to employ an exponential time checking process, in the size of the domain of the constraint.

We investigate the level mapping characterization for aggregate programs and find that, for aggregates, the conditional satisfaction checking can be reduced to polynomial time standard satisfaction checking. Based on this finding, we give a level mapping characterization to answer sets. The characterization induces a formulation of loop formulas, where local power sets are not needed and the exponential process to compute them is avoided.

To summarize, the main contributions of this paper are:

- We characterize the stable model semantics of weight constraint programs by means of level mapping. This result is new, as in the past no level mapping characterizations have been formulated for this class of programs.

- We show that the above level mapping characterization leads to a new formulation of loop formulas for arbitrary weight constraint programs without using extra atoms, thus improving the previous formulation which may need extra atoms and rules.

- We further use level mapping to characterize the answer set semantics of aggregate programs, based on which we formulate loop formulas for these programs, where dependency graphs can be built in polynomial time. In the previously known results, this process takes exponential time in the worst case.

This paper is organized as follows. In Sections 2 and 3, respectively, we present level mapping characterizations and loop formulas for weight constraint and aggregate programs, after a review of related definitions. We compare the level mapping characterizations and loop formulas for weight constraint and aggregate programs in Section 4. Section 5 relates the notions of level mapping and loop formulas to the recently introduced notions of *level ranking* and *ranking constraints* [13, 22]. Finally, conclusions and future work are given in Section 6.

The main results of this paper have been reported in a poster paper [16], without proofs. The main extensions here include: (1) Section 4, where we demonstrate how the level mappings and loop formulas reveal the difference between the stable model and the answer set semantics; (2) Section 5, where we extend the level ranking characterization [13, 22] from normal logic programs to weight constraint programs and show that level mapping and loop formulas coincide with level ranking and ranking constraints, respectively, for normal logic programs; and (3) the detailed proofs of all the theorems and lemmas.

2. Level mapping induced loop formulas for weight constraint programs

We assume a propositional language determined by a fixed countable set of propositional atoms.

2.1. Stable model semantics for weight constraint programs

A *weight constraint* is of the form

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

where each a_i, b_j is an atom. Atoms $a_i's$ and not-atoms $\texttt{not}\ b_i's$ are also called *literals* (*positive* and *negative* literals, respectively). We denote by $lit(W)$ the set of literals in a weight constraint. Each literal in a constraint is associated with a *weight*. The numbers l and u give the lower and upper bounds of the constraint, 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 ∞.

The weights of literals could be negative. In [26], it is pointed out that negative weights can be eliminated by the following transformation: 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

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

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

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 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. W_0 is the *head* of the rule and $\{W_1, ..., W_n\}$ is the *body* of the rule. A weight constraint like $1\ [l = 1]$ will be written as l. Given a rule r, we denote the head and body of r by $hd(r)$ and $bd(r)$, respectively. We use $At(P)$ to denote the set of the atoms appearing in program P.

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

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

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

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

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

Definition 2.1. ([26]) Let P be a weight constraint program and $M \subseteq At(P)$. M is a *stable model* of P iff M is a model of P and the deductive closure of P^M.

Note that the deductive closure of P^M can be computed by a fixpoint construction, using the operator T_P defined in [18]. Let P be a weight constraint program, where the head of each rule is an atom and the body contains no negative literals. The operator T_P is defined as:

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

The least fixpoint of T_P can be constructed by

$$T_P^0(\emptyset) = \emptyset, \qquad T_P^{i+1}(\emptyset) = T_P(T_P^i(\emptyset)), \qquad T_P^\infty(\emptyset) = \cup_{i=0}^\infty T_P^i(\emptyset).$$

Then, we have the following proposition.

Proposition 2.1. Given a weight constraint program P, a set of atoms M is a stable model of P iff $M \models P$ and $M = T_{P^M}^\infty(\emptyset)$.

Notation:

In the rest of this paper, we will use the following notations: 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.

In general, an atom may appear both positively and negatively in a weight constraint. We call such an atom a *dual* atom. For example, the atom a in the weight constraint $1[a = 1, \texttt{not } a = 2]1$ is a dual atom. Given a set of atoms M, if there are no dual atoms in a weight constraint W, then $M_a(W) \cap M_b(W) = \emptyset$, otherwise, $M_a(W) \cap M_b(W) \neq \emptyset$.

Following the notation in [29], for a set of atoms X and a mapping λ from X to positive integers, we define

$$H(X) = max(\{\lambda(a) \mid a \in X\}). \quad (7)$$

For the empty set $\emptyset$, we define $\max(\emptyset) = 0$ and $\min(\emptyset) = \infty$.

2.2. Level mapping characterization of stable models

Given a set of atoms X, a *level mapping* of X is a function λ from X to positive integers.

Definition 2.2. Let W be a weight constraint of the form (1), M a set of atoms and λ a level mapping of M. The *level* of W w.r.t. M, denoted $L(W, M)$, is defined as:

$$L(W, M) = min(\{H(X_a) \mid X \subseteq M \text{ and } w(W, X_a) \geq l + \sum_{b_i \in M \setminus X_a} w_{b_i}\}). \quad (8)$$

If there are no dual atoms in W, $M_b \cap X_a = \emptyset$. In this case, $\sum_{b_i \in M \setminus X_a} w_{b_i}$ in formula (8) becomes $\sum_{b_i \in M} w_{b_i}$.

Intuitively, the *level* of a weight constraint W, w.r.t. M, is determined by subsets $X \subseteq M$ satisfying $w(W, X_a) \geq l + \sum_{b_i \in M \setminus X_a} w_{b_i}$. The latter specifies a condition in which W is satisfied by X. Among all $x \in X$, since they represent a conjunction, we select the largest level, and among all $X \subseteq M$, since they represent a disjunction we select the smallest one.

Example 2.1. Let $W = 1[a = 1, \texttt{not } a = 1, \texttt{not } b = 1]$ be a weight constraint, $M = \{a\}$ a set, and λ a level mapping of M. Considering the subset of M, $X = \emptyset$, we have $w(W, X_a) = 2$ and $l + \sum_{b_i \in M \setminus X_b} w_{b_i} = 1 + w_a = 2$. $w(W, X_a) \geq l + \sum_{b_i \in M \setminus X_b} w_{b_i}$. Similarly, we can check that the other subset of M, $X' = M$ also satisfies the inequality in formula (8). Thus, $L(W, M) = min(\{H(\emptyset), H(\{a\})\})$. Note that the level of any atom is a positive number. Then we have $L(W, M) = 0$.

Definition 2.3. Let P be a weight constraint program and M a set of atoms. M is said to be *level mapping justified* by P if there is a level mapping λ of M, such that for each $b \in M$, there is a rule $r \in P$, such that $b \in lit(hd(r))$, $M \models bd(r)$, and for each $W \in bd(r)$, $\lambda(b) > L(W, M)$.

In this case, we say that the level mapping λ justifies M by P.

By a manipulation of inequalities, we prove

Lemma 2.1. Let W be a weight constraint of the form (1), M and X be two sets of atoms. $w(W^M, X_a) \geq l^M$ iff $w(W, X_a) \geq l + \sum_{b_i \in M \setminus X_a} w_{b_i}$.

Proof:
Note that $w(W, X_a) = w(W^M, X_a) + \sum_{b_i \notin X_a} w_{b_i}$. Then, we have $w(W^M, X_a) \geq l - \sum_{b_i \notin M} w_{b_i}$ iff $w(W, X_a) \geq l - \sum_{b_i \notin M} w_{b_i} + \sum_{b_i \notin X_a} w_{b_i}$, which is $w(W, X_a) \geq l + \sum_{b_i \in M \setminus X_a} w_{b_i}$. □

Using Lemma 2.1, we can prove the following theorem.

Theorem 2.1. Let P be a weight constraint program and M a set of atoms. M is a stable model of P iff M is a model of P and level mapping justified by P.

Proof:
($\Rightarrow$) Since M is a stable model of P, by Proposition 2.1, M is the deductive closure of P^M. The derivation of the least fixpoint of the operator T_{P^M} can be used to assign levels to the atoms in M: assign an atom b in M to k if $b \in T^k_{P^M}(\emptyset)$ and $b \notin T^{k-1}_{P^M}(\emptyset)$. It can be seen clearly that the resulting level mapping justifies M by P.

($\Leftarrow$) Suppose M is a model of P and level mapping justified by P. Let λ be a level mapping that justifies M by P. Then, for each $b \in M$, there is a rule r in P, such that $b \in lit(hd(r))$ and for each $W \in bd(r)$, $M \models W$ and $\lambda(b) > L(W, M)$. By the definition of $L(W, M)$, for each $W \in bd(r)$, there is a subset $X \subseteq M \setminus \{b\}$ such that $w(W, X_a) \geq l + \sum_{b_i \in M \setminus X_a} w_{b_i}$. From this we get $w(W^M, X_a) \geq l^M$, by Lemma 2.1. Clearly, the level mapping λ enables the construction of M as the least fixpoint of the operator T_{P^M}. By proposition 2.1, M is a stable model of P. □

Example 2.2. Let P_1 be the program $\{a \leftarrow 1[a = 1, \texttt{not } a = 1, \texttt{not } b = 1]\}$. Let $W = 1[a = 1, \texttt{not } a = 1, \texttt{not } b = 1]$, $M = \{a\}$ and λ be a level mapping of M. We have $L(W, M) = 0$ (see Example 2.1). Therefore $\lambda(a) > L(W, M)$ and M is level mapping justified by P_1. M is a stable model of P_1.

2.3. Loop formulas for weight constraint programs

2.4. Completion

To characterize stable models by loop formulas, we need the concept of *completion* of a program. The models of the completion of a program are the supported models of the program. Following [18], the completion of a weight constraint program P is defined as a set of formulas built from weight constraints by means of Boolean connectives $\wedge$, $\vee$ and $\rightarrow$.

Let $S = \{f_1, ..., f_n\}$ be a set of weight constraints or formulas built from weight constraints. We denote the conjunction $f_1 \wedge ... \wedge f_n$ by $\wedge S$ and the disjunction $f_1 \vee ... \vee f_n$ by $\vee S$.

Let P be a weight constraint program. The completion of P, denoted $Comp(P)$, consists of the following formulas.

(1) $\wedge bd(r) \rightarrow hd(r)$, for every rule $r \in P$, and

(2) $x \rightarrow \vee\{\wedge bd(r) \mid r \in P, x \in lit(hd(r))\}$, for every atom $x \in At(P)$.

2.4.1. Loop formulas

The formulation of loop formulas consists of two steps: constructing a dependency graph and then establishing a formula for each loop in the graph.

The dependency graph for a weight constraint program is defined as below.

Let P be a weight constraint program. The *dependency graph* of P, denoted $G_P = (V, E)$, is a directed graph, where

- $V = At(P)$, and
- (u, v) is a directed edge from u to v in E, if there is a rule of the form (3) in P, such that $u \in lit(W_0)$ and $v \in lit(W_i)$, for some i ($1 \leq i \leq n$).

Let $G = (V, E)$ be a directed graph. A set $L \subseteq V$ is a *loop* in G if the subgraph of G induced by L is strongly connected.

For a loop in the dependency graph, the level mapping induced loop formula is established to enforce that the atoms in the loop must be justified by the atoms that are not in the loop and have a strictly lower level. Considering the definition of $L(W, M)$ (see formula (8)), the condition requires that an atom in a loop be derivable by a subset X of M which contains no atoms in the loop and satisfies the inequality in formula (8).

To enforce the above requirements, we define the *restriction* of a weight constraint w.r.t. a loop.

Let W be a weight constraint and L be a set of atoms. The *restriction* of W w.r.t. L, denoted $W_{|L}$, is the conjunction of weight constraints $W_{l|L} \wedge W_{u|L}$, where

- $W_{l|L}$ is obtained by removing the upper bound, all positive literals that are in L and their weights from W, and

- $W_{u|L}$ is obtained by removing the lower bound from W.

Definition 2.4. Let P be a weight constraint program and L a loop in G_P. The *loop formula* for L, denoted $LF(P, L)$, is defined as[2]

$$LF(P,L) = \bigvee L \rightarrow \bigvee \{ \bigwedge_{W \in bd(r)} W_{|L} \mid r \in P \text{ and } L \cap lit(hd(r)) \neq \emptyset \} \tag{9}$$

Let P be a weight constraint program. The *loop completion* of P is defined as

$$LComp(P) = Comp(P) \cup \{LF(P,L) \mid L \text{ is a loop in } G_P\} \tag{10}$$

With the definition of loop completion, we can prove

Theorem 2.2. Let P be a weight constraint program and M a set of atoms. M is a stable model of P iff M is a model of $LComp(P)$.

Proof:
We consider rules of the form $c \leftarrow W$, where c is an atom and W a weight constraint of the form (1). The proof can be extended straightforwardly to rules of a conjunctive body and a weight constraint in the head (by the reduct defined in equation (5), given a set of atoms M, the head weight constraint of a rule is decomposed into rules each of which has a single atom in the head).

($\Rightarrow$) Since M is a stable model of P, by Theorem 2.1, M is a model of P and level mapping justified by P. Thus, there exists a level mapping λ from $At(P)$ to positive integers satisfying: for all $b \in M$, there is a rule $b \leftarrow W$ such that $M \models W$ and $\lambda(b) > L(W, M)$. By the definition of the level of a weight constraint (cf. equation 8), there exists a set $X \subseteq M \setminus L$ such that $w(W, X_a) \geq l + \sum_{b_i \in M \setminus X_a} w_{b_i}$.

We need to prove $M \models LComp(P)$. For this purpose, let L be a loop in G_P and $b \in L$. Since a stable model of P is a model of P, we only need to prove $M \models W_{l|L} \wedge W_{u|L}$. Since $M \models W$, by the definition of $W_{u|L}$, that $M \models W_{u|L}$ is immediate. We show $M \models W_{l|L}$. By definition, we have

$$w(W_{l|L}, X_a) \leq \sum_{a_i \in M \setminus L} w_{a_i} + \sum_{b_i \notin X_a} w_{b_i}$$
$$w(W_{l|L}, M) = \sum_{a_i \in M \setminus L} w_{a_i} + \sum_{b_i \notin M} w_{b_i}$$

It can then be verified that

$$\sum_{b_i \notin X_a} w_{b_i} - \sum_{b_i \notin M} w_{b_i} = \sum_{b_i \in M \setminus X_a} w_{b_i}$$

Hence

$$w(W_{l|L}, M) \geq w(W_{l|L}, X_a) - \sum_{b_i \in M \setminus X_a} w_{b_i}$$

We note that, by definition, we have $w(W_{l|L}, X_a) = w(W, X_a)$. Then, by substitution, we get

$$w(W_{l|L}, M) \geq w(W, X_a) - \sum_{b_i \in M \setminus X_a} w_{b_i}$$

[2]The same loop formula was also discovered by Yisong Wang independently and reported in [30]. The loop formula introduced here is induced by a level mapping characterization, whereas the one in [30] is defined directly.

Since $w(W, X_a) \geq l + \sum_{b_i \in M \setminus X_a} w_{b_i}$, it follows $w(W_{l|L}, M) \geq l$, i.e., $M \models W_{l|L}$. By the definition of loop formulas, we have $M \models LF(P, L)$. As M is a model of P, $M \models Comp(P)$. Since this holds for any loop, we conclude $M \models LComp(P)$.

($\Leftarrow$) Assume $M \models LComp(P)$ and we show that M is a stable model of P. As M is a model of P, we only need to show that there is a level mapping that justifies M by P. For this purpose, let L be a loop in G_P. From the assumption that $M \models LComp(P)$, we have $M \models LF(P, L)$. Then, $\forall b \in M \cap L$, there exists a rule $b \leftarrow W$ in P, such that $M \models W_{l|L} \wedge W_{u|L}$. We only need to show that there exists a subset $X \subseteq M$ such that $w(W, X_a) \geq l + \sum_{b_i \in M \setminus X_a} w_{b_i}$. Below, we construct such an X.

Let $X = M_a \setminus L_a$. Note that $w(W_{l|L}, M) = \sum_{a_i \in X_a} w_{a_i} + \sum_{b_i \notin M} w_{b_i}$. Since $M \models W_{l|L}$, we have $w(W_{l|L}, M) \geq l$. It follows that

$$\sum_{a_i \in X_a} w_{a_i} + \sum_{b_i \notin M} w_{b_i} \geq l$$
$$\sum_{a_i \in X_a} w_{a_i} \geq l - \sum_{b_i \notin M} w_{b_i}$$

From $w(W_{l|L}, X_a) = \sum_{a_i \in X_a} w_{a_i} + \sum_{b_i \notin X_a} w_{b_i}$, we get

$$w(W_{l|L}, X_a) \geq l - \sum_{b_i \notin M} w_{b_i} + \sum_{b_i \notin X_a} w_{b_i}$$
$$w(W_{l|L}, X_a) \geq l + \sum_{M \setminus X_a} w_{b_i}$$

Since $w(W_{l|L}, X_a) = w(W, X_a)$, we have $w(W, X_a) \geq l + \sum_{b_i \in M \setminus X_a} w_{b_i}$. Thus, by definition, there is a level mapping λ where $\lambda(b) \geq max(\{\lambda(a_i) \mid a_i \in X\})$ that justifies M by P. Since M is a model of P and level mapping justified by P, it follows from Theorem 2.1 that M is a stable model of P. □

Example 2.3. Consider the program P_1 in Example 2.2. There is a loop in $L = \{a\}$ in G_{P_1}. The loop formula is

$$a \rightarrow 1[\texttt{not}\ a = 1, \texttt{not}\ b = 1] \wedge [a = 1, \texttt{not}\ a = 1, \texttt{not}\ b = 1]$$

As the second weight constraint has no bounds, it is satisfied automatically (we will follow this simplification convention in later examples), the above formula simplifies to

$$a \rightarrow 1[\texttt{not}\ a = 1, \texttt{not}\ b = 1]$$

which is satisfied by the set $M = \{a\}$. So, M is a stable model of P_1.

3. Level mapping induced loop formulas for aggregate programs

3.1. Answer set semantics for aggregate programs

Following [28], we define the syntax and semantics of aggregate programs as follows.

An aggregate is a constraint on sets taking the form

$$aggr(\{X \mid p(X)\})\ \texttt{op}\ Result \tag{11}$$

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 *intentional set*, where p is a predicate, and X is a variable, which takes value from a set $D(X) = \{a_1, ..., a_n\}$, called *variable domain*. The relational

operator op is from $\{=, \neq, <, >, \leq, \geq\}$ and $Result$ is either a variable or a numeric constraint. The *domain* of an aggregate A, denoted $Dom(A)$, is the set of atoms $\{p(a) \mid a \in D(X)\}$.

Let M be a set of atoms. M is a *model* of an aggregate A, denoted $M \models A$, if $aggr(\{a \mid p(a) \in M \cap Dom(A)\})$ op $Result$ holds, e.g., $\{p(0), p(1)\}$ is a model of $SUM(\{X|p(X)\}) \geq 1$, where $D(X) = \{-1, 0, 1\}$.

An aggregate program is a set of rules of the form

$$h \leftarrow A_1, ..., A_n \tag{12}$$

where h is an atom and $A_1, ..., A_n$ are aggregates[3]. For a rule r of the form (12), $hd(r)$ and $bd(r)$ denote h and $\{A_1, ..., A_n\}$, respectively. We use $At(P)$ to denote the propositional atoms in P.

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

Definition 3.1. Let A be an aggregate, R and S 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 $R \cap Dom(A) \subseteq I \subseteq S \cap Dom(A)$, $I \models A$.

Conditional satisfaction is naturally extended to conjunctions of aggregates.

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

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

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

Definition 3.2. ([28]) Let P be an aggregate program and M a set of atoms. M is an *answer set* of P iff M is the least fixpoint of $K_P(\emptyset, M)$, i.e., $M = K_P^{\infty}(\emptyset, M)$, where $K_P^0(\emptyset, M) = \emptyset$ and $K_P^{i+1}(\emptyset, M) = K_P(K_P^i(\emptyset, M), M)$, for all $i \geq 0$.

It has been shown that most standard aggregates can be encoded by weight constraints [17][4]. Especially, the aggregate SUM can be encoded by a weight constraint as follows.

Let $A = SUM(\{X|p(X)\}) \leq k$ be an aggregate, where $D(X) = \{a_1, ...a_m, b_1, ..., b_n\}$, and $a_i's$ and $b_i's$ are positive and negative numbers, respectively. A is equivalent to the following weight constraint

$$W = [p(a_1) = a_1, ..., p(a_m) = a_m, p(b_1) = b_1, ..., p(b_n) = b_n]k \tag{13}$$

in the sense that for any set of atoms M, $M \models A$ iff $M \models W$. Using the transformation in Section 2.1, W can be translated to

$$W(A) = [p(a_1) = a_1, ..., p(a_m) = a_m, \texttt{not } p(b_1) = b_1, ..., \texttt{not } p(b_n) = b_n]k' \tag{14}$$

where $k' = k + \sum_{i=1}^{n} |b_i|$. So, for any set of atoms M, $M \models A$ iff $M \models W(A)$. We call $W(A)$ the *weight constraint encoding* of A.

Aggregate SUM with other operators (except "$\neq$") can be encoded similarly, e.g., for the operator "$\geq$", k' becomes the lower bound of $W(A)$.

[3] In general, A_i's can also be atoms or negative atoms, as the latter are special cases of aggregates.

[4] The only exception is aggregates of the form $SUM(.) \neq k$. For programs with $SUM(.) \neq k$, the complexity of the *answer set existence problem* is at the second level of the polynomial hierarchy, according to Son and Pontelli [28].

A nice property of aggregates is that its weight constraint encoding contains no dual atoms. This property plays an important role in the proof of the lemmas given later in this section.

In this section, we focus on programs with aggregate SUM only, because: (1) It is a representative aggregate in that the aggregates $COUNT$ and AVG are special cases of SUM [17]; (2) It is the most commonly used aggregate in current practice of answer set programming e.g., this is the case for the benchmarks in the answer set programming system competition [9]; and (3) There are no technical difficulties in extending the results to aggregate programs with aggregates MAX and MIN, since they can be encoded by the aggregate SUM.

In the rest of this section, by aggregates we mean standard aggregates without involving the $\neq$ operator in the SUM aggregate function.

Summarizing the above exploration, we state

Proposition 3.1. Let A be an aggregate and $W(A)$ be its weight constraint encoding. Then, for any set of atoms I, $I \models A$ iff $I \models W(A)$.

3.2. Level mapping characterization of answer sets

Definition 3.3. Let A be an aggregate, M a set of atoms and λ a level mapping of M. The *answer set level* of A w.r.t. M, denoted $L^*(A, M)$, is defined as:

$$L^*(A, M) = min(\{H(X) \mid X \subseteq M, w(W(A), X_a) \geq l + \sum_{b_i \in M} w_{b_i}, \text{ and } w(W(A), X_b) \leq u - \sum_{a_i \in M} w_{a_i}\}). \tag{15}$$

Example 3.1. Let $A = SUM(\{X \mid p(X)\}) \leq 0$ be an aggregate, where $D(X) = \{-1, 1\}$. Then $W(A) = [p(1) = 1, \texttt{not}\ p(-1) = 1]1$. Let $M = \{p(-1), p(1)\}$ and λ be a level mapping of M, where $\lambda(p(-1)) = 1$ and $\lambda(p(1)) = 2$. It can be seen that the subsets of M that satisfy the inequalities in formula (15) are $X_1 = \{p(-1)\}$ and $X_2 = M$. So, $L^*(A, M) = min(\{H(\{p(-1)\}), H(\{p(-1), p(1)\})\}) = \lambda(p(-1)) = 1$.

Definition 3.4. Let P be an aggregate program and M a set of atoms. M is said to be *strongly level mapping justified* by P if there is a level mapping λ of M satisfying that for each $b \in M$, there is a rule $r \in P$, such that $b = hd(r)$, $M \models bd(r)$, and for each $A \in bd(r)$, $\lambda(b) > L^*(A, M)$.

In this case, we say that the level mapping λ strongly justifies M by P.

The main result of this section follows from the following lemma, by which the conditional satisfaction of an aggregate is reduced to the standard satisfaction of its weight constraint encoding.

Lemma 3.1. Let A be an aggregate, and X and M be two sets of atoms such that $X \subseteq M$. $X \models_M A$ iff

- $w(W(A), X_a) \geq l + \sum_{b_i \in M} w_{b_i}$, and
- $w(W(A), X_b) \leq u - \sum_{a_i \in M} w_{a_i}$

where l and u are the lower and upper bounds of $W(A)$, respectively.

Proof:
($\Rightarrow$) Assume $X \models_M A$, and we show that the two inequalities above hold. Let $S = M_b \cup X_a$. Then $X \subseteq S \subseteq M$ and $S \models W(A)$. It follows that $w(W(A), S) = w(W(A), X_a) - \sum_{b_i \in M} w_{b_i}$ and $w(W(A), S) \geq l$. Then we have $w(W(A), X_a) \geq l + \sum_{b_i \in M} w_{b_i}$. This is the first inequality. Let $S' = M_a \cup X_b$. We then have $X \subseteq S' \subseteq M$ and $S' \models W(A)$. Then clearly $w(W(A), S') = \sum_{a_i \in M} w_{a_i} + \sum_{b_i \notin X_b} w_{b_i}$. Note that $\sum_{b_i \notin X_b} w_{b_i} = w(W(A), X_b)$, from which we derive $w(W(A), S') = \sum_{a_i \in M} w_{a_i} + w(W(A), X_b)$. According to the choice of S', we have $w(W(A), S') \leq u$, and so $w(W(A), X_b) \leq u - \sum_{a_i \in M} w_{a_i}$. This is the second inequality.

($\Leftarrow$) Assuming that the two inequalities in the lemma hold, we show $X \models_M A$, i.e., we need to prove that for any S such that $X \subseteq S \subseteq M$, $S \models A$. Let S be any set with $X \subseteq S \subseteq M$. Clearly, $X_a \subseteq S_a$ and $S_b \subseteq M_b$. Thus $w(W(A), S) \geq w(W(A), X_a) - \sum_{b_i \in M} w_{b_i}$. It follows $w(W(A), S) \geq l$. Since $X \subseteq S \subseteq M$, we also have $S_a \subseteq M_a$ and $X_b \subseteq S_b$, and it follows that $w(W(A), S) \leq w(W(A), X_b) + \sum_{a_i \in M} w_{a_i}$. Thus $w(W(A), S) \leq u$. As both lower and upper bounds are satisfied by S, we have $S \models W(A)$, and by Proposition 3.1 we get $S \models A$. We are done. □

Now we are ready to give the main theorem of this section.

Theorem 3.1. Let P be an aggregate program and M a set of atoms. M is an answer set of P iff M is a model of P and strongly level mapping justified by P.

Proof:
($\Rightarrow$) Since M is an answer set of P, then M is a model of P and $M = K_P^\infty(\emptyset, M)$. The derivation of the least fixpoint of the operator K_P can be used to assign levels to the atoms in M: assign an atom b in M to k if $b \in K_P^k(\emptyset, M)$ and $b \notin K_P^{k-1}(\emptyset, M)$. It can be verified that the resulting level mapping strongly justifies M by P.

($\Leftarrow$) By the definition of the answer set level of an aggregate (Definition 15) and Lemma 3.1, we have $\forall b \in M$, there is a rule $r \in P$ and a set $R \subseteq M \setminus \{b\}$ such that $b = hd(r)$ and $R \models_M bd(r)$. So, $b \in K_P^\infty(\emptyset, M)$. Thus $M \subseteq K_P^\infty(\emptyset, M)$. Note that $K_P^\infty(\emptyset, M) \subseteq M$, due to the monotonicity of K_P. Thus we have $M = K_P^\infty(\emptyset, M)$, that is, M is an answer set of P. □

Example 3.2. Let P_2 be the program

$$p(-1) \leftarrow \qquad p(1) \leftarrow SUM(\{X \mid p(X)\}) \leq 0.$$

Let $M = \{p(-1), p(1)\}$ and λ be a level mapping of M, where $\lambda(p(-1)) = 1$ and $\lambda(p(1)) = 2$. We have $L^*(A, M) = 1$ (cf. Example 3.1). Therefore, $\lambda(p(1)) > L^*(A, M)$ and M is strongly level mapping justified by P_2. Hence, M is an answer set of P_2.

3.3. Loop formulas for aggregate programs

3.3.1. Completion

The completion of an aggregate program consists of the same set of formulas as that of weight constraint programs, except that the weight constraints in the formulas are the weight constraint encoding of aggregates.

3.3.2. Loop formulas

Let P be an aggregate program. The *dependency graph* of P, denoted $G^*_P = (V, E)$, is a directed graph, where

- $V = At(P)$,
- (u, v) is a directed edge from u to v in E, if there is a rule of the form (12) in P, such that $u = hd(r)$, and either v or $\texttt{not}\ v \in lit(W(A_i))$, for some i ($1 \leq i \leq n$).

Now we give the *strong restriction* of an aggregate w.r.t. a loop by defining the strong restriction of a weight constraint. Let W be a weight constraint and L a set of atoms. The strong restriction of W, w.r.t. L, denoted $W^*_{|L}$, is a conjunction of weight constraints $W^*_{l|L} \wedge W^*_{u|L}$, where

- $W^*_{l|L}$ is obtained by removing from W the upper bound, all positive literals that are in L and their weights;
- $W^*_{u|L}$ is obtained by removing from W the lower bound, $\texttt{not}\ b_i = w_{b_i}$ for each $b_i \in L$, and changing the upper bound to be $u - \sum_{b_i \in L} w_{b_i}$.

The strong restriction of an aggregate A w.r.t. a loop L is defined as the strong restriction of its weight constraint encoding w.r.t. the loop $W^*_{|L}(A)$.

Definition 3.5. Let P be an aggregate program and L a loop in G^*_P. The loop formula for L, denoted $LF^*(P, L)$, is defined as

$$LF^*(P, L) = \bigvee L \rightarrow \bigvee \{ \bigwedge_{A \in bd(r)} W^*_{|L}(A) \mid r \in P \text{ and } hd(r) \in L \} \tag{16}$$

Let P be an aggregate program. The loop completion of P, denoted $LComp^*(P)$, is defined as

$$LComp^*(P) = Comp(P) \cup \{LF^*(P, L) \mid L \text{ is a loop in } G^*_P\} \tag{17}$$

We prove the following theorem.

Theorem 3.2. Let P be an aggregate program and M a set of atoms. M is an answer set of P iff it is a model of $LComp^*(P)$.

Proof:
Below, since the aggregate A in consideration is always clear by context, for notational convenience, we will use respectively W, $W^*_{l|L}$, and $W^*_{u|L}$ to represent $W(A)$, $W^*_{l|L}(A)$, and $W^*_{u|L}(A)$. We consider the rule of the form $h \leftarrow A$. The proof can be generalized easily to arbitrary rules.

($\Rightarrow$) Since M is an answer set of P, by Theorem 3.1, M is a model of P and strongly level mapping justified by P. Thus, there exists a level mapping λ from $At(P)$ to positive integers satisfying: $\forall b \in M, \exists$ a rule $b \leftarrow W$ in P, such that $M \models W$ and $\lambda(b) > L^*(A, M)$. By definition, the latter means there exists a subset $X \subseteq M$, such that $w(W, X_a) \geq l + \sum_{b_i \in M} w_{b_i}$ and $w(W, X_b) \leq u - \sum_{a_i \in M} w_{a_i}$.

We need to show that $M \models LComp(P)$. For this purpose, let L be a loop in G^*_P and $b \in L$. Since an answer set of P is a model of P, we only need to show the satisfaction of bounds defined in $W^*_{|L}$, i.e.,

$M \models W^*_{l|L} \wedge W^*_{u|L}$. The proof of $M \models W^*_{l|L}$ is similar to that of Theorem 2.2. We prove $M \models W^*_{u|L}$. First, by definition, we have

$$w(W, M_b \setminus L_b) \leq w(W, X_b)$$
$$w(W, M_b \setminus L_b) = w(W, M) - \sum_{a_i \in M} w_{a_i} + \sum_{b_i \in L_b \cap M_b} w_{b_i}$$

Since $w(W, X_b) \leq u - \sum_{a_i \in M} w_{a_i}$, it follows $w(W, M) + \sum_{b_i \in L_b \cap M_b} w_{b_i} \leq u$. Note that, by definition, $w(W^*_{u|L}, M) = w(W, M) - \sum_{b_i \in L} w_{b_i} + \sum_{b_i \in L_b \cap M_b} w_{b_i}$. Then, we have $w(W^*_{u|L}, M) \leq u - \sum_{b_i \in L} w_{b_i}$, that is $M \models W^*_{u|L}$. We are done.

($\Leftarrow$) Assume $M \models LComp(P)$, and we show that M is an answer set. Let L be a loop in G^*_P, hence $M \models LF^*(P, L)$. Then, $\forall b \in M \cap L$, there exists a rule $b \leftarrow W$ in P, such that $M \models W^*_{l|L} \wedge W^*_{u|L}$.

We need to show that there exists a set $X \subseteq M$ such that $w(W, X_a) \geq l + \sum_{b_i \in M} w_{b_i}$ and $w(W, X_b) \leq u - \sum_{a_i \in M} w_{a_i}$. Let $X = X_a \cup X_b$, where $X_a = M_a \setminus L_a$ and $X_b = M_b \setminus L_b$. The proof of the first inequality is similar to the proof of Theorem 2.2. We show the second inequality.

Since $M \models W^*_{u|L}$, we have $w(W^*_{u|L}, M) \leq u - \sum_{b_i \in L} w_{b_i}$. By definition, $w(W^*_{u|L}, M) = w(W, M) - \sum_{b_i \in L} w_{b_i} + \sum_{b_i \in L \cap M} w_{b_i}$. It follows $w(W, M) \leq u - \sum_{b_i \in L \cap M} w_{b_i}$. By definition, $w(W, X_b) = w(W, M) - \sum_{a_i \in M} w_{a_i} + \sum_{b_i \in L \cap M} w_{b_i}$. Since $w(W, M) \leq u - \sum_{b_i \in L \cap M} w_{b_i}$, we have $w(W, X_b) \leq u - \sum_{a_i \in M} w_{a_i}$. Thus, by definition, there is a level mapping where $\lambda(b) \geq max(\{\lambda(b_i) \mid b_i \in X_b\})$ that strongly justifies M by P. As M is a model of P, by Theorem 3.1, we conclude that M is an answer set of P. □

Example 3.3. Consider the program P_2 in Example 3.2. There is a loop $\{p(1)\}$ in $G^*_{P_2}$. The loop formula is $p(1) \rightarrow [p(1) = 1, \texttt{not}\ p(-1) = 1]1$, which is satisfied by the set $M = \{p(-1), p(1)\}$. So, M is an answer set of P_2.

Remark

In the approach presented in this section, given a program P, the dependency graph G^*_P can be constructed by going through each rule, where an edge is built in G^*_P from the head of the rule to each literal in the domain of aggregates in the body of the rule. The process takes time linear in the size of P (number of rules plus the number of atoms in P). The exponential time process required in the approach [31] is therefore avoided. The establishment of the formula for a loop is also linear in the size of P, since the restriction of an aggregate can be obtained in linear time according to the definition.

It can be shown that any loop defined in [31] is a loop defined here, but the reverse does not hold. This is because we count negative dependencies as edges. As a result, there are more loops under this definition. However, for many of these loops, their loop formulas can be satisfied by any supported model (need not to be constructed in practice). For example, given the following program:

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

$L = \{a, b\}$ is a loop by our definition. The loop formula of L is satisfied by any supported model of the program.

4. A comparison

Weight constraint programs and aggregate programs are both logic programs with constraints, but they have different semantics. Our level mapping characterizations and loop formulas reveal the difference

between the semantics. Comparing the definition of the level of weight constraints in formula (8) to the answer set level of aggregates in formula (15), one can see that the atoms in X_b that are negative in W and the upper bound of W are not considered in formula (8), while they are constrained by the second inequality in formula (15).

Naturally, the difference is also manifested by the loop formulas for weight constraint and aggregate programs, respectively. For the stable model semantics, negative dependencies between atoms do not contribute to loop formulas, while they do for the answer set semantics. The following example demonstrates how level mapping and loop formulas capture this difference.

Example 4.1. Let P_3 be the aggregate program $\{p(-1) \leftarrow SUM(\{X|p(X)\}) \leq -1\}$, where $D(X) = \{-1\}$. We denote the aggregate in P_3 by A. We have $W(A) = [\texttt{not}\ p(-1) = 1]0$. The weight constraint program counterpart of P_3 is $P_3' = \{p(-1) \leftarrow [\texttt{not}\ p(-1) = 1]0\}$.

Consider the set $M = \{p(-1)\}$. For program P_3 and any level mapping λ, we have $L^*(A, M) = \lambda(p(-1))$, since the only subset of M that satisfies the inequalities in formula (15) is M itself. Namely, $p(-1)$ depends on itself, even though it appears negatively in the body of the rule. There is a loop $\{p(-1)\}$ in $G^*_{P_3}$. The loop formula is: $p(-1) \rightarrow [\,] - 1$, which is not satisfied by M. Thus M is not an answer set of P_3.

For program P_3' and any level mapping α, we have $L(W(A), M) = 0$, since $\emptyset$ satisfies the inequality in formula (8). There is no loop in $G_{P_3'}$. It can be verified that M is a stable model of P_3'.

5. Relation to level ranking characterization

Recently in [22], stable models of normal logic programs are characterized using a notion called *level ranking*. This characterization enables an alternative mechanism for stable model computation for normal logic programs using *difference logic* [13]. Our interest in this section is to relate our work with this new development. The main observations are

- For normal logic programs, level mapping and level ranking coincide. As a result, our work on level mapping can be viewed as an extension of level ranking to weight constraint programs.
- For normal logic programs, loop formulas correspond to *ranking constraints* in difference logic encoding [13, 22]. We show a concrete establishment between the two.

Recall that a normal logic program P is a set of rules of the form

$$a \leftarrow b_1, ..., b_m, \texttt{not}\ c_1, ..., \texttt{not}\ c_n. \tag{18}$$

where a, b_i and c_j are atoms. Given a program P and a set of atoms M, a level ranking lr of M for P is a mapping from M to natural numbers, such that for each atom $a \in M$, there is a rule r in the program reduct P^M where $hd(r) = a$ and $lr(a) > lr(b)$ for each $b \in bd(r)$. A model M of a program P is a stable model of P if and only if there is a level ranking of M for P.

For normal logic programs, the level mapping characterization coincides with the level ranking characterization, which can be stated as: A model M of a program P is level mapping justified by P iff there is a level ranking of M for P. This can be easily seen as follows. Given an atom a and a set of atoms M, by the definition of $L(W, M)$, we have $L(a, M) = \lambda(a)$, if $a \in M$; $L(a, M) = \infty$, otherwise. Thus, for

normal logic programs the definition of level ranking is the same as that of level mapping justification (Definition 2.3). Therefore, the existence of a level ranking for M corresponds to the statement that M is level mapping justified.

The concept of level ranking can be extended to weight constraint programs, using the notion of the levels of weight constraints.

Definition 5.1. Let M be a set of atoms and P be a weight constraint program. A function λ: $M \to \mathbb{N}$ is a level ranking of M for P iff for each $a \in M$, there is a rule $r \in P^M$ such that $a \in lit(hd(r))$ and for every W in $bd(r)$, $\lambda(a) > L(W, M)$.

The level ranking characterization of stable models for weight constraint programs can be stated as follows.

Proposition 5.1. Let M be a model of weight constraint program P. M is a stable model of P iff there is a level ranking of M for P.

Proposition 5.1 follows from Theorem 2.1.

Based on the level ranking characterization, a translation is proposed in [13, 22], which maps a normal logic program P to a set of *difference logic* formulas $T_{\text{diff}}(P)$. $T_{\text{diff}}(P)$ consists of two parts: the standard Clark's completion $CC(P)$ and ranking constraints $R(P)$. It turns out that each satisfying valuation of $T_{\text{diff}}(P)$ corresponds to a stable model of P. Below, we show that ranking constraints of a program play the same role as our loop formulas.

Difference logic is an extension of propositional logic by allowing simple linear constraints of the form $x_i + k \geq x_j$ where x_i, x_j are integer variables and k is an integer constant.

A valuation τ consists of a pair of functions $\tau_{\mathcal{P}}$: $\mathcal{P} \to \{\top, \bot\}$ and $\tau_{\mathcal{X}} : \mathcal{X} \to \mathbb{Z}$, where $\mathcal{P}$ is a set of propositional atoms, $\top$ value $true$, $\bot$ value $false$, and $\mathcal{X}$ a set of integer variables. A valuation is extended to formulas by defining $\tau(x_i + k \geq x_j) = \top$ iff $\tau_{\mathcal{X}}(x_i) + k \geq \tau_{\mathcal{X}}(x_j)$ and applying the usual rule for the Boolean connectives. A formula f is satisfied by a valuation τ iff $\tau(f) = \top$.

The completion $CC(P)$ consists of the set of following formulas for each atom $a \in At(P)$: if a does not appear as a head of any rule in P, the formula $\neg a$ is included. Otherwise the formula $a \leftrightarrow bd_a^1 \vee \cdots \vee bd_a^k$ is included for each rule of the form (18) and, furthermore, for each such rule a formula $bd_a^1 \leftrightarrow b_1 \wedge \cdots b_m \wedge \neg c_1 \cdots \neg c_n$ is added, where bd_a^i is a new atom associated to the rule.

The ranking constraints $R(P)$ consists of a set of formulas

$$a \to \vee_{i=1}^{k}(bd_a^i \wedge (x_a - 1 \geq x_{b_1}) \wedge \cdots \wedge (x_a - 1 \geq x_{b_m})) \tag{19}$$

for each atom a which has $k \geq 1$ rules of the form (18) in P where x_{b_i}'s are integer variables associated to the atoms b_i's in the bodies of the rules.

It is easy to prove that a set of atoms is a supported model of P if and only if there is a satisfying valuation τ of $CC(P)$, such that $M = \{a \mid a \in At(P) \text{ and } \tau_{\mathcal{P}}(a) = \top\}$. Below, we show that, for normal logic programs, ranking constraints play the same role as that of loop formulas.

Theorem 5.1. Let P be a normal logic program and M a supported model of P. $M \models LF(P, L)$ for each loop L in G_p iff there is a satisfying valuation τ of $R(P)$ such that $M = \{a \mid \tau_{\mathcal{P}}(a) = \top\}$.

The statement follows from the fact that the notion of level ranking coincides with that of level mapping, and the notion of level mapping justification corresponds to models of completion plus loop formulas, for weight constraint programs (cf. Theorem 2.1), of which normal logic programs are a special case. However, for normal logic programs, a direct proof is more desirable, since it explicitly shows that loop formulas play the same role as that of ranking constraints in the characterization of stable models. We give such a proof below.

Proof:
($\Rightarrow$) Let f be a formula of the form (19) in $R(P)$. If a is not on any loop, the proof is trivial. Let L be a loop and $a \in L$. Since $M \models LF(P, L)$, there is a rule r in P, such that $hd(r) = a$ and $M \models \wedge_{b \in bd(r)} b_{|L}$, where $b_{|L}$ is the restriction of b w.r.t L. By the definition of restriction, we have $b_{|L} = 1[\,]$, if $b \in L$ and $b_{|L} = b$, if $b \notin L$. Since $M \models \wedge_{b \in bd(r)} b_{|L}$, $b \notin L$ for each $b \in bd(r)$. Suppose r is associated to bd_a^k in $R(P)$. Then a valuation τ can be constructed such that $\tau_{\mathcal{P}}(a) = \top$, $\tau_{\mathcal{P}}(bd_a^k) = \top$, $\tau_{\mathcal{P}}(b_i) = \top$ for all $b_i \in bd(r)$, $\tau_{\mathcal{P}}(c_i) = \bot$ for all `not` $c_i \in bd(r)$, and $\tau_{\mathcal{X}}(x_a) > \tau_{\mathcal{X}}(x_{b_1}), \cdots, \tau_{\mathcal{X}}(x_a) > \tau_{\mathcal{X}}(x_{b_m})$.

($\Leftarrow$) Let L be a loop. $LF(P, L) = \vee L \rightarrow \vee\{\wedge_{b \in bd(r)} b_{|L} \mid r \in P,\ a = hd(r),\ \text{and}\ a \in L\}$. Let $a \in L \cap M$. There is a formula in $R(P)$ of the form (19). Suppose bd_a^i in the formula is associated to the rule r. Then there is a satisfying valuation τ of $R(P)$, such that $M = \{a \mid \tau_{\mathcal{P}}(a) = \top\}$. We have $\tau_{\mathcal{P}}(a) = \top$, $\tau_{\mathcal{P}}(bd_a^i) = \top$. Since M is a supported model, $\tau_{\mathcal{P}}(b_i) = \top$ for each $b_i \in bd(r)$, $\tau\mathcal{P}(c_i) = \bot$ for each `not` $c_i \in bd(r)$. Thus $b_i \in M$ for each $1 \leq i \leq m$. Since $\tau_{\mathcal{X}}(x_a) > \tau_{\mathcal{X}}(x_{b_1}), \cdots, \tau_{\mathcal{X}}(x_a) > \tau_{\mathcal{X}}(x_{b_m})$, we have, for $1 \leq i \leq m$, $b_i \notin L$. Therefore, $M \models LF(P, L)$. □

From the computational point of view, loop formulas and ranking constraints have their own strengths and weaknesses, which can be summarized as follows.

- Loop formulas are constructed on the fly. Only when a model of completion is generated which fails to be a stable model, the loop formula for a loop that caused the failure is constructed. In many cases, loop formulas need not be constructed at all [12]. In contrast, ranking constraints are introduced for rules in a given program, no matter whether there actually exist loops or not, or whether a loop contributes to the generation of a model of completion which is not a stable model.

- The number of loops could be exponential in the size of a program in the worst case, but the size of ranking constraints is linear in the size of a program.

6. Conclusion and future work

We have presented level mapping characterizations of semantics for weight constraint programs and aggregate programs, respectively, based on which we have improved the formulation of loop formulas for these programs. For arbitrary weight constraint programs, we have proposed a formulation of loop formulas that does not require introducing any new atoms; For an aggregate program, we have shown that the dependency graph can be constructed in time polynomial in the size of the program.

A number of issues require further investigation. First, the level mapping and loop formulas are defined on aggregate SUM. For the aggregates MAX and MIN, an encoding of these aggregates in terms of the SUM aggregate is nontrivial [17], and direct definitions may be desired for intuitiveness. Secondly, as we have mentioned, there may be many redundant loops. Thus, our result may be more appropriately regarded as evidence of the existence of a polynomial time construction of dependency graph

in order to formulate loop formulas, rather than a practical proposal which is ready for implementation. The impact of the redundant loops on efficiency and the possible methods to remove them are worth further study.

Finally, though we are able to relate level mapping with level ranking and extend the latter definition to weight constraint programs, it remains open as whether there exists a polynomial encoding of level ranking in difference logic, for weight constraint programs. Essentially, the question is whether the approach of answer set computation by difference logic encoding and solvers can be suitably extended from normal logic programs to weight constraint programs.

Acknowledgment

The review comments and editor's notes helped improve the presentation of this paper. Both authors are partially supported by a discovery grant from Natural Sciences and Engineering Research Council of Canada.

References

[1] Clark, K. L.: Negation as failure, *Logics and Databases*, 1978, 293–322.

[2] Dell'Armi, T., Faber, W., Lelpa, G., Leone, N.: Aggregate functions in disjunctive logic programming: semantics, complexity, and implementation in DLV, *Proc. IJCAI'03*, 2003.

[3] Elkabani, I., Pontelli, E., Son, T.: $Smodels^A$ – A system for computing answer sets of logic programs with aggregates, *Proc. LPNMR'05*, 2005.

[4] Erdem, E., Lifschitz, V.: Tight Logic Programs, *Theory and Practice of Logic Programming*, **3**(4), 2003, 499–518.

[5] Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs, *Proc. JELIA'04*, 2004.

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

[7] Ferraris, P.: Answer sets for propositional theories, *Proc. LPNMR'05*, 2005.

[8] Gebser, M., Kaufmann, B., Schaub, T.: The conflict-driven answer set solver clasp: progress report, *Proc. LPNMR'09*, 2009.

[9] Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first snswer set programming system competition, *Proc. LPNMR'07*, 2007.

[10] Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming, *Proc. 5th ICLP*, 1988.

[11] Giunchiglia, L., Lierler, Y., Maratea, M.: Cmodles-2: SAT-based answer set programming, *Proc. AAAI'04*, 2004.

[12] Huang, G.-S., Jia, X., Liau, C.-J., You, J.-H.: Two-literal logic programs and satisfiability representation of stable models: A comparison, *Proc. Canadian Artificial Intelligence*, 2002.

[13] Janhunen, T., Niemelä, I., Sevalnev, M.: Computing stable models via reductions to difference logic, *Proc. LPNMR'09*, 2009.

[14] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning, *ACM Transactions on Computational Logic*, **7**(3), 2006, 1–57.

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

[16] Liu, G.: Level mapping induced loop formulas for weight constraint and aggregate programs, *Proc. LPNMR'09*, 2009.

[17] Liu, G., You, J.: Lparse programs revisited: semantics and representation of aggregates, *Proc. ICLP'08*, 2008.

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

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

[20] Marek, V., Remmel, J. B.: Set constraints in logic programming, *Proc. LPNMR'04*, 2004.

[21] Marek, V., Truszczyński, M.: Logic programs with abstract constraint atoms, *Proc. AAAI'04*, 2004.

[22] Niemelä, I.: Stable models and difference logic, *Ann. Math. Artif. Intell.*, **53**(1-4), 2008, 313–329.

[23] Niemelä, I., Simons, P., Soininen, T.: Stable model semantics of weight constraint rules, *Proc. LPNMR'99*, 1999.

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

[25] Shen, Y., You, J.: A generalized Gelfond-Lifschitz transformation for logic programs with abstract constraints, *Proc. AAAI'07*, 2007.

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

[27] Soininen, T., Niemelä, I.: Developing a declarative rule language for applications in product configuration, *Proc. PADL'99*, 1999.

[28] Son, T., Pontelli, E.: A constructive semantic characterization of aggregates in answer set programming, *Theory and Practice of Logic Programming*, **7**, 2006, 355–375.

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

[30] Wang, Y., You, J., Yuan, L., Zhang, M.: Weight constraint programs with functions, *Proc. LPNMR'09*, 2009.

[31] You, J., Liu, G.: Loop formulas for logic programs with arbitrary constraint atoms, *Proc. AAAI'08*, 2008.

[32] You, J., Yuan, L., Zhang, M.: On the equivalence between answer sets and models of competioln for nested logic programs, *Proc. IJCAI'03*, 2003.