Loop Formulas for Logic Programs with Arbitrary Constraint Atoms

Jia-Huai You and Guohua Liu
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{you,guohua}@cs.ualberta.ca

Abstract
We formulate loop formulas for logic programs with arbitrary constraint atoms, for the semantics based on conditional satisfaction. This provides a method for answer set computation by computing models of completion. One particular attractive candidate for the latter task is pseudo-boolean constraint solvers. To strengthen this connection, we show examples of compact encoding of aggregates and global constraints by pseudo-boolean constraints.

Introduction
Answer set programming (ASP), namely logic programming with negation under the stable model semantics, has recently been extended to include constraint atoms (or just c-atoms) (Marek & Truszczyński 2004).

A constraint atom expresses a constraint over a set of atoms. Its abstract form is \((D, C)\), where \(D\) is a set of atoms called the domain, and \(C\) a collection of the subsets from the power set of \(D\) serving as the admissible solutions.

A weakness of ASP in the past had been the lack of an explicit means to represent and reason with collections of atoms. This weakness becomes apparent in comparison with constraint programming (CP) based on the constraint satisfaction problem (CSP), both in representation and efficient processing of constraints. For example, aggregates can be naturally expressed and efficiently processed in CP (by the technique called bounds-consistency), while their handling in ASP had been studied only recently. As another example, global constraints (van Hoeve & Katriel 2006) play an important role in modeling and efficient processing of constraint problems (typically by embedded constraint propagators), but the concept did not even exist in ASP.

Under these circumstances, the introduction of constraint atoms into ASP is clearly significant, both in theory and potential applications. Many types of constraints studied in the past can be represented by constraint atoms, including weight constraints, aggregates, and global constraints.

In this paper, we present a method for answer set computation for logic programs with arbitrary c-atoms, by formulating loop formulas for these programs, for the semantics based on conditional satisfaction (Son, Pontelli, & Tu 2007). This semantics is known to coincide with the ultimate stable semantics for logic programs with aggregates (Denecker, Pelov, & Bruynooghe 2001)). Our work follows the previous work on loop formulas for normal logic programs (Lin & Zhao 2004), and loop formulas for logic programs with monotone and convex constraints (Liu & Truszczyński 2006). We show that answer sets of a logic program with arbitrary c-atoms are precisely the models of completion that satisfy our loop formulas. Since the semantics based on conditional satisfaction agrees with the semantics for logic programs with monotone and convex c-atoms, our loop formulas are applicable to the latter. Actually, for this class of programs, our loop formulas are simpler (sometimes exponentially simpler) than those proposed in (Liu & Truszczyński 2006). Our results can be applied to any constraint solver, where models of completion can be computed. The next question is which constraint solvers are suitable for this task.

In (Liu & Truszczyński 2006), it is shown how logic programs with cardinality and weight constraints, called lparse programs, can be encoded by pseudo-boolean (PB) theories. Here, we advocate in addition that PB constraint solvers (e.g. (Chai & Kuehlmann 2005)) are attractive candidates for computing models of completion for programs with arbitrary c-atoms, especially for programs with aggregates and global constraints. This is due to the observation that typical aggregates in logic programs, as well as some global constraints, can be encoded as PB theories compactly. That is, the size of the PB encoding of such a constraint is linear in the size of the constraint’s domain. This is in contrast with the unfolding approach (Pelov, Denecker, & Bruynooghe 2003; Son & Pontelli 2007) - translating a logic program with aggregates to a normal program, where the resulting normal program could be of exponential size for the same constraint. We show some encodings to illustrate this connection. Our work opens the door, for the first time to the best of our knowledge, for the possibility of incorporating global constraints into ASP.

Basic Definitions
We assume a propositional language with a countable set of propositional atoms. Once a program \(P\) is defined, we will denote by \(At(P)\) the set of atoms appearing in \(P\).

A constraint atom (c-atom) is of the form \((D, C)\), where \(D\) is a finite set of atoms and \(C \subseteq 2^D\). For a c-atom
\( A = (D, C) \), we may use \( A_d \) and \( A_c \) to refer to \( D \) and \( C \), respectively.

Let \( A \) be a c-atom. \( A \) is said to be elementary if it is of the form \( \{a\}, \{\{a\}\} \), which may be just written as \( a \); \( A \) is monotone if for every \( X \subseteq Y \subseteq A_d \), \( X \in A \) implies that \( Y \in A \); \( A \) is nonmonotone if it is not monotone; \( A \) is antimonotone if \( A_c \) is closed under subset, i.e., for every \( X, Y \subseteq A_d \), if \( Y \in A \), and \( X \subseteq Y \) then \( X \in A \); \( A \) is convex if for every \( X, Y, Z \subseteq A_d \) such that \( X \subseteq Y \subseteq Z \) and \( X, Z \in A_c \), then \( Y \in A_c \).

A set of atoms \( M \) satisfies a c-atom \( A \), written \( M \models A \), if \( M \cap A_d \subseteq A_c \). Otherwise \( M \) does not satisfy \( A \), written \( M \nmid A \). \( M \models \neg A \) (or \( M \models \neg A \)) if \( M \nmid A \). Satisfiability naturally extends to conjunctions of c-atoms (sometimes written as a set) and disjunctions of c-atoms.

C-atoms of the form \( D, \emptyset \) are not satisfiable. When a c-atom of this type appears in the head of a rule, we write \( \bot \) instead, and call it a constraint. On the other hand, some c-atoms are tautologies. For example, all monotone c-atoms of the form \( (D, 2^D) \) are tautologies.

A logic program with c-atoms, also called a constraint program (or just program), is a finite set of rules of the form

\[
A \leftarrow A_1, \ldots, A_n, \not A_{k+1}, \ldots, \not A_n.
\]

where \( A \) and \( A_i \)'s are arbitrary c-atoms. The literals \( \not A_j \) are called negative c-atoms.

For a rule \( r \) of the form (1), we define \( \text{hd}(r) = A \) and \( \text{bd}(r) = \{A_1, \ldots, A_n, \not A_{k+1}, \ldots, \not A_n\} \), which are called the head and the body of \( r \), respectively. A rule \( r \) is said to be basic if every c-atom in it occurs positively, and either \( \text{hd}(r) \) is elementary or \( r \) is a constraint. A program \( P \) is basic if every rule in it is basic; \( P \) is a normal program if every c-atom in it is elementary; \( P \) is a monotone-constraint program if every c-atom in it is monotone.

A set of atoms \( M \subseteq \text{At}(P) \) is a model of \( P \) if and only if \( M \models \text{hd}(r) \) whenever \( M \models \text{bd}(r) \). \( M \) is a supported model of \( P \) if for any \( a \in M \), there is \( r \in P \) such that \( a \in \text{hd}(r) \) and \( M \models \text{bd}(r) \).

Following (Son, Pontelli, & Tu 2007), a negative c-atom \( \not A \) in a program is interpreted by its complement, and substituted by c-atom \( \overline{A} \), where \( \overline{A_d} = A_d \) and \( \overline{A_c} = 2^{A \setminus A_d} \). For example, the negative elementary c-atom \( \not \{a\} \), \( \{\{a\}\} \) will be replaced by c-atom \( \{\overline{a}\}, \{\emptyset\} \).

Due to this assumption, in the sequel, if not said otherwise, constraint programs contain only positive c-atoms.

Answer sets for constraint programs are defined in two steps. In the first step, answer sets for basic programs are defined, based on a notion called conditional satisfaction.

Definition 1. Let \( M \) and \( S \) be sets of atoms. The set \( S \) conditionally satisfies a c-atom \( A \), w.r.t. \( M \), denoted by \( S \models_M A \), if \( S \models A \) and for every \( I \subseteq A_d \) such that \( S \cap A_d \subseteq I \) and \( I \subseteq M \cap A_d \), we have that \( I \in A_c \).

An operator \( T_P \) is then defined. For any sets \( R, S, \) and a basic program \( P \), \( T_P(R, S) \) is defined as:

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

Definition 2. Let \( M \) be a model of a basic program \( P \). \( M \) is an answer set for \( P \) iff \( M = T_P^n(\emptyset, M) \), where \( T_P^0(\emptyset, M) = \emptyset \) and \( T_P^{i+1}(\emptyset, M) = T_P(T_P^i(\emptyset, M), M) \), for all \( i \geq 0 \).

Next, a constraint program is represented by its instances in the form of basic programs, and the answer sets of the former are defined in terms of the answer sets of the latter.

Let \( P \) be a constraint program and \( r \in P \). For each \( \pi \in \text{hd}(r) \), the instance of \( r \) w.r.t. \( \pi \) is the set of rules consisting of

1. \( b \leftarrow \text{bd}(r) \), for each \( b \in \pi \), and
2. \( \bot \leftarrow d, \text{bd}(r) \), for each \( d \in \text{hd}(r)_d \setminus \pi \).

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

Definition 3. Let \( P \) be a constraint program and \( M \) a set of atoms. \( M \) is an answer set for \( P \) iff \( M \) is an answer set for one of its instances.

For a c-atom \( A \), we may be interested only in the domain atoms that actually appear in some admissible solution of \( A \). We thus define \( \text{ASet}(A) = \{a : a \in \pi, \text{for some } \pi \in A_c\} \).

Completion

In the sequel, for a set of atoms \( S = \{a_1, \ldots, a_n\} \), we use \( S^\wedge \) to denote the conjunction \( a_1 \land \ldots \land a_n \), and \( \neg S \) to denote the conjunction \( \neg a_1 \land \ldots \land \neg a_n \).

Following (Liu & Truszczyński 2006), the completion of a constraint program \( P \), denoted by \( \text{Comp}(P) \), consists of the following formulas

- \( \{\text{bd}(r)^\wedge \} \leftarrow \text{hd}(r), \) for each \( r \in P \);
- \( x \rightarrow \bigvee \{\text{bd}(r)^\wedge : r \in P, x \in \text{ASet}(\text{hd}(r))\}, \) for each atom \( x \in \text{At}(P) \).

The first formula above captures the if definition in a rule, whereas the second completes the definition by adding the only if part.

The completion of a constraint program is a set of formulas with c-atoms. The notion of satisfaction and models for constraint programs extends in a natural way to formulas with c-atoms.

Note that in the definition of (Liu & Truszczyński 2006), a negative literal \( \not A \) in completion is interpreted as \( \neg A \), while in this paper it is interpreted by its complement \( \overline{A} \). The two are consistent since for any set of atoms \( I \subseteq \text{At}(P) \), \( I \models \neg A \) if and only if \( I \models \overline{A} \).

Theorem 1. (Liu & Truszczyński 2006) Let \( P \) be a constraint program. A set of atoms \( M \) is a supported model of \( P \) if and only if \( M \) is a model of \( \text{Comp}(P) \).

Loop Formulas

To define the loop formulas for a constraint program, we introduce the notion of local power set.

Definition 4. Let \( A \) be a c-atom. A pair of sets \( (B, T) \), where \( B \subseteq T \subseteq A_d \), is called a local power set (LPS) of \( A \), if \( B \in A_c \), \( T \in A_c \), and for any set \( I \) such that \( B \subseteq I \subseteq T \), we have \( I \in A_c \).

Intuitively, \( (B, T) \) represents an “extension” of the power set \( 2^T \setminus B \), where \( B \) is the “bottom” element and \( T \) is the “top” element.
A local power set \((B, T)\) of a c-atom \(A\) is said to be maximal if there is no other local power set \((B', T')\) of \(A\) such that \(B' \subseteq B\) and \(T \subseteq T'\).

The LPS representation of a c-atom is defined as follows.

**Definition 5.** Let \(A\) be a c-atom. The LPS representation of \(A\), denoted by \(A^*\), is \((A_d, A^*_c)\), where \(A^*_c = \{(B, T) : (B, T)\) is a maximal LPS of \(A\}\).

**Example 1.** Let \(A = \{(a, b, c), \{\emptyset, \{a\}, \{a, b, c\}\}\). Then, \(A^* = \{(a, b, c), \{(\emptyset, \{a\}, \{a, b, c\}\}\}\).

Let \(M\) be a set of atoms and \(A\) be a c-atom. We say \(M\) satisfies \(A^*\), denoted \(M \models A^*\), if \(M \models \bigvee \{B^n \land \neg (A_d \setminus T) : (B, T) \in A^*_c\}\).

The satisfaction of a c-atom can be characterized by the satisfaction of its LPS representation.

**Proposition 1.** Let \(A\) be a c-atom and \(M\) be a set of atoms. We have \(M \models A\) iff \(M \models A^*\).

\(A^*\) can be seen as a compact representation of \(A\). For example, given a monotone c-atom \(A = (D, 2^D - \{\emptyset\})\), \(A^* = (A_d, \{(B, A_d) : B \subseteq A_d, B\) is singleton\})

To capture the loops in constraint programs, we define the dependency graph for a constraint program.

**Definition 6.** Let \(P\) be a constraint program. The dependency graph of \(P\), denoted by \(G^*_p = (V, E)\) (a stands for arbitrary c-atoms), 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 \(r \in P\) such that \(u \in ASet(hd(r))\) and \(v \in B\), for some \((B, T) \in A^*_c\) and \(A \in bd(r)\).

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. A loop is maximal if it is not a proper subset of any other loop in \(G\). A maximal loop is terminating if there is no edge in \(G\) from \(L\) to any other maximal loop.

For any nonempty set \(X \subseteq At(P)\), we denote by \(G^*_p[X]\) the subgraph of \(G^*_p\) induced by \(X\).

The idea of loop formula is that for any atom in a loop to be in an answer set of a program, it must be supported by atoms that are not in the loop. To capture this idea, the restriction of the LPS representation of a c-atom is defined.

**Definition 7.** Let \(P\) be a constraint program. \(A\) is a c-atom, and \(L \subseteq At(P)\). The restriction of \(A^*\) to \(L\), denoted by \(A^*_c|_L\), is \((A_d, A^*_c|_L)\), where

\[A^*_c|_L = \{(B, T) : L \cap B = \emptyset\}\]

Let \(L\) be a set of atoms and \(r\) be a rule \(A \leftarrow A_1, \ldots, A_n\).

We define the formula

\[\alpha_L (r) = \pi_1 \land \ldots \land \pi_n\]

where \(\pi_i = \bigvee \{B^n \land \neg (A_i \setminus T) : (B, T) \in A^*_c|_L\}\), for each \(i\), \(1 \leq i \leq n\). If there is a c-atom \(A_i\) such that \(A^*_c|_L = \emptyset\), then \(\alpha_L (r) = \text{false}\).

We are ready to give our definition of loop formulas.

**Definition 8.** Let \(P\) be a constraint program and \(L\) be a loop in \(G^*_p\). The loop formula for \(L\), denoted by \(LP^a(L)\), is defined as

\[\forall L \rightarrow \forall \{\alpha_L (r) : r \in P, L \cap ASet(hd(r)) \neq \emptyset\}.

The loop completion of a constraint program \(P\), denoted \(LComp(P)\), combines the completion of \(P\) with the loop formulas for loops in \(G^*_p\), and is defined as

\[LComp(P) = \text{Comp}(P) \cup \{LP^a(L) : L\) is a loop in \(G^*_p\}\}.

**Example 2.** Consider the following program \(P\):

\[r_1 : p \leftarrow \{(a, b, c), \{\emptyset, \{a, b\}, \{a, b, c\}\}\}\]

\[r_2 : a \leftarrow p\]

\[r_3 : b \leftarrow p\]

The only model of \(\text{Comp}(P)\) is \(M = \{a, b, p\}\). We will see that \(M\) is not an answer set for \(P\). Let \(A = \{(a, b, c), \{\emptyset, \{a, b\}, \{a, b, c\}\}\}\).

Then,

\[A^* = \{(a, b, c), \{(\emptyset, \emptyset), \{(a, b), \{a, b, c\}\}\}\}\]

\[L = \{a, b, p\}\) is a loop in \(G^*_p\), and its loop formula is:

\[LP^a(L) = \forall a \lor b \lor p \rightarrow \alpha_L (r_1) \lor \alpha_L (r_2) \lor \alpha_L (r_3)\]

Since \(\alpha_L (r_1) = \neg a \land \neg b \land \neg c\) and \(\alpha_L (r_2) = \alpha_L (r_3) = \text{false}\), we get

\[LP^a(L) = \forall a \lor b \lor p \rightarrow \neg (a \land \neg b \land \neg c)\]

As \(M \not\models LP^a(L)\), we conclude that \(M\) is not an answer set for \(P\).

**Example 3.** Consider the program \(P\):

\[d \leftarrow \{(a, b, c, d), \{(a), \{a, b\}, \{a, c\}, \{a, b, c\}, \{d\}, \{d, a, d\}\}\}\]

\[a \leftarrow \]

Let \(A\) be the c-atom in the body of the first rule. We have

\[A^* = \{(a, b, c, d), \{(a), \{a, b\}, \{a, c\}, \{a, b, c\}, \{(d\}, \{d, a, d\}\}\}\]

The loop formula for the loop \(L = \{d\}\) is:

\[LP^a(L) = d \leftarrow (a \land \neg d) \lor (a \land \neg b \land \neg c)\]

As \(M = \{a, d\}\) is a model of \(\text{Comp}(P)\) and \(M \not\models LP^a(L)\), for the only loop \(L\) in \(G^*_p\), it is an answer set for \(P\).

Now, here comes the main theorem.

**Theorem 2.** Let \(P\) be a constraint program. A set \(M \subseteq At(P)\) is an answer set for \(P\) if and only if \(M\) is a model of \(LComp(P)\).

The theorem can be proved using the following lemma.

**Lemma 1.** Let \(P\) be a basic program. A set \(M \subseteq At(P)\) is an answer set for \(P\) if and only if \(M\) is a model of \(LComp(P)\).

Below, we give a detailed proof sketch for the if part. The only if part is slightly less involved.

**Proof.** Assume \(M\) is a model of \(LComp(P)\) and we show \(M = T^\infty (\emptyset, M)\). The proof of \(T^\infty (\emptyset, M) \subseteq M\) is routine by an induction on the construction of \(T^\infty (\emptyset, M)\). We show the other direction, \(M \subseteq T^\infty(\emptyset, M)\).

Suppose the statement doesn’t hold, i.e., for some \(x \in M\), \(x \notin T^\infty(\emptyset, M)\). Then, since \(T^\infty(\emptyset, M) \subseteq M\), it follows \(T^\infty(\emptyset, M) \subseteq M\). Let \(M^- = M \setminus T^\infty(\emptyset, M)\). In \(G^*_p[M^-]\), there are either loops or no loops. If \(G^*_p[M^-]\) doesn’t have a loop, it’s easy to show that \(x \in T^\infty(\emptyset, M)\), causing a contradiction. Suppose \(G^*_p[M^-]\) has a loop, then it has a terminating loop \(L \subseteq M^-\). Since \(M \models LComp(P)\), there exist \(x \in M^-\) and \(r \in P\), where \(hd(r) = x\), such that \(M \models \text{bd}(r)\), and for any \(A \in \text{bd}(r)\).

586
Local power set representation

A naive algorithm would examine each pair \( X, Y \in A \), such that \( X \subseteq Y \), in the partial order of inclusiveness – a pair \((X, Y)\) includes another \((X', Y')\) if \( X \subseteq X' \) and \( Y' \subseteq Y \), whereas at least one of the \( \subseteq \) is proper, to see if for any \( I \) such that \( X \subseteq I \subseteq Y \), \( I \in A \). Whenever such a pair \((X, Y)\) is identified, it is maximal, and thus all the included pairs are dropped from consideration.

For some special classes of c-atoms, the construction of \( A^* \) is much simpler. Below, we show this for the classes of monotone, antimonotone, and convex c-atoms.

Given a set \( S \) of sets, \( \pi \in S \) is said to be minimal in \( S \) if there is no \( \pi' \in S \) such that \( \pi' \subset \pi \); similarly for maximal sets in \( S \).

- \( A \) is monotone: \( A^*_c = \{(B, A_d) : B \text{ is minimal in } A_c\} \)
- \( A \) is antimonotone: \( A^*_c = \{\emptyset, T : T \text{ is maximal in } A_c\} \)
- \( A \) is convex: \( A^*_c = \{(B, T) : B \subseteq T, B \text{ is minimal, } T \text{ is maximal in } A_c\} \)

Relation to Previous Work

Similar constructions or definitions have been presented in the literature for different purposes. In (Pelov, Denecke, & Bruynooghe 2003), indexed pairs are used to translate aggregate programs to normal logic programs. In (Son, Ponrouch, Shen & You 2007), which are used to define a power sets general form of Gelfond-Lifchitz reduction.

A naive algorithm would examine each pair \( X, Y \in A \), such that \( X \subseteq Y \), in the partial order of inclusiveness – a pair \((X, Y)\) includes another \((X', Y')\) if \( X \subseteq X' \) and \( Y' \subseteq Y \), whereas at least one of the \( \subseteq \) is proper, to see if for any \( I \) such that \( X \subseteq I \subseteq Y \), \( I \in A \). Whenever such a pair \((X, Y)\) is identified, it is maximal, and thus all the included pairs are dropped from consideration.

For some special classes of c-atoms, the construction of \( A^* \) is much simpler. Below, we show this for the classes of monotone, antimonotone, and convex c-atoms.

Given a set \( S \) of sets, \( \pi \in S \) is said to be minimal in \( S \) if there is no \( \pi' \in S \) such that \( \pi' \subset \pi \); similarly for maximal sets in \( S \).

- \( A \) is monotone: \( A^*_c = \{(B, A_d) : B \text{ is minimal in } A_c\} \)
- \( A \) is antimonotone: \( A^*_c = \{\emptyset, T : T \text{ is maximal in } A_c\} \)
- \( A \) is convex: \( A^*_c = \{(B, T) : B \subseteq T, B \text{ is minimal, } T \text{ is maximal in } A_c\} \)

Example 4. Consider a program with nonmonotone c-atoms, denoted \( P \):

\[
a \leftarrow \text{not} \{(a, b), \emptyset, \{b\}\}.
\]

\( G^{p^a} \) contains no edges. So, it cannot be used to deny a model of completion to be an answer set. Indeed, \( M = \{a\} \) is a model of \( \text{Comp}(P) \), but not an answer set for \( P \).

Our dependency graph is based on the program where negative c-atoms are replaced by their complements. In this case, we have the following program, denoted \( P' \):

\[
a \leftarrow \{\{a, b\}, \{a\}, \{b\}\}\)

where \( L_1 = \{a\} \) and \( L_2 = \{b\} \) are loops in \( G^{p^a} \), but \( L_3 = \{a, b\} \) is not. Note that \( L_3 \) is also a loop in \( G^{p^a} \), in addition to \( L_1 \) and \( L_2 \).

The loop formula for \( L_1 \) in our case, for example, is \( LP^n(L_1) = a \rightarrow \text{false} \). \( M \neq LP^n(L_1) \), which shows that \( M \) is not an answer set for \( P \).

Proposition 2. Let \( P \) be a constraint program. Any loop in \( G^{p^a} \) is a loop in \( G^{p^b} \), but the converse does not hold.

For monotone-constraint programs \( P \), the definition of \( G^{p^a} \) is consistent with that of \( G^{p^m} \). By definition, the complement of a monotone c-atom is antimonotone. As shown earlier, if \( A \) is antimonotone, then \( A^* \) is such that for every \( (B, T) \in A^*_c \), we have \( B = \emptyset \). It follows from the definition of \( G^{p^a} \) that a head atom does not depend on any body atoms in an antimonotone c-atom.

However, since the extra loops in \( G^{p^a} \) are non-essential (e.g. the loop \( L_3 \) in Example 4), we can show that our loop formulas also work with the definition of loops in \( G^{p^m} \).

Theorem 3. Let \( P \) be a constraint program, and \( \text{LC}(P) = \text{Comp}(P) \cup \{LP^n(L) : L \text{ is a loop in } G^{p^m}\} \). A set \( M \subseteq \text{At}(P) \) is an answer set for \( P \) if \( M \) is a model of \( \text{LC}(P) \).

Loop formulas

The loop formulas in (Liu & Truszczynski 2006) are defined as follows. For a rule of the form

\[
A \leftarrow A_1, \ldots, A_k, \text{not } A_{k+1}, \ldots, \text{not } A_m,
\]

define

\[
\beta_L(r) = A_1|L \land \ldots \land A_k|L \land \neg A_{k+1} \land \ldots \land \neg A_m
\]

where \( A_i | L = (A_i \setminus L, \{Y : Y \in A_{c}, Y \cap L = \emptyset\}) \).

The loop formula for \( L \) denoted \( LP^n(L) \), is defined as

\[\text{Dependancy graph}

In (Liu & Truszczynski 2006), positive dependency graph is defined for programs with monotone c-atoms. Let \( P \) be a program with monotone c-atoms which consists of rules of the form (1). The positive dependency graph of \( P \) is the directed graph \( G^{p^m} = (V, E) \), where \( V = \text{At}(P) \) and \( \langle u, v \rangle \) is an edge in \( E \) if there exists a rule \( r \in P \) such that \( u \in bd(r) \) and \( v \in A_d \), for some positive c-atom \( A \in bd(r) \).

We make two remarks. First, the construction of \( G^{p^m} \) is not directly applicable to programs with nonmonotone c-atoms. For these programs, a head atom in a rule may also positively depend on the atoms appearing in a negative body c-atom of the same rule. Second, there are in general more loops in \( G^{p^m} \) than in \( G^{p^a} \).
For monotone-constraint programs (note that they do not contain negative c-atoms), our loop formula is equivalent to the one above.

**Theorem 4.** Let $P$ be a monotone-constraint program and $M \subseteq \text{At}(P)$. For any loop $L \in G^m_P$, $M \models \text{LP}^m(L)$ iff $M \models \text{LP}^m(L)$.

**Corollary 1.** Let $P$ be a monotone-constraint program. For any loop $L \in G^m_P$, no atom in $\text{LP}^m(L)$ occurs negatively.

Thus, for any monotone-constraint program $P$ (without negative c-atoms), for any loop $L$ in $G^m_P$ or in $G^p_P$, the size of $\text{LP}^m(L)$ is no larger than that of $\text{LP}^m(L)$. In many cases, the former is substantially smaller; in some cases, the former is exponentially smaller. For instance, consider the following monotone program $P$:
\[
c \leftarrow \{\{a, b, c\}, \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}\}.
\]
There is no loop in $G^p_P$, but one loop $L = \{c\}$ in $G^m_P$. Our loop formula for this loop is $\text{LP}^m(L) = c \rightarrow a$, while the loop formula of $\text{LP}^m(L)$ is:
\[
\text{LP}^m(L) = c \rightarrow \{\{a, b\}, \{\{a\}, \{a, b\}\}\}.
\]
To scale up, replace the rule above with $a_n \leftarrow A$, where $A = \{\{1, \ldots, a_n\}, \{\pi : \pi \subseteq A_d, a_1 \in \pi\}\}$.

**Encoding Constraint Atoms by PB Theories**

We show some example encodings of c-atoms by pseudo-boolea (PB) theories, to illustrate the possibility of computing models of completion by PB constraint solvers. We contrast this approach with the unfolding approach (Pelov, Denecker, & Bruynooghe 2003; Son & Pontelli 2007).

**Pseudo-boolean constraints**

PB constraints are integer programming constraints in which variables have 0-1 domains. They are generally written in the form of inequalities
\[
w_1 \times x_1 + \ldots + w_n \times x_n \geq w
\]
where $\mathcal{R} \in \{\leq, \geq, <, >, =\}$, $w_1$ and $w_2$ are integer coefficients, and $x_1$ and $x_2$ are PB variables taking values from the domain $\{0, 1\}$. A PB theory is a set of PB constraints. Let's denote the set of variables in a PB theory by $X$. An assignment $\lambda : X \rightarrow \{0, 1\}$ is a solution to a PB theory if it satisfies every constraint in the theory.

Given a program $P$, the scheme for the encoding of $\text{Comp}(P)$ by PB constraints is given in (Liu & Truszczynski 2006), where all c-atoms need to be encoded.

There is a straightforward encoding of any c-atom $A$ in terms of a PB theory by enumerating all admissible solutions in $A$. We are interested in the cases where the resulting PB theories are linear in the size of a c-atom’s domain.

Let $\lambda$ be a solution to the PB encoding of a c-atom $A$, and suppose the domain of $A$ is $S = \{a_1, \ldots, a_n\}$. In the PB encoding of $A$ below, we will introduce a PB variable $x_i$ for each $a_i$, and define the membership set of $\lambda$ to be $M_\lambda = \{a_i, a_i \in S : \lambda(x_i) = 1\}$.

To represent the satisfaction of a c-atom $A$, we introduce a PB variable $y$. The intention is that an assignment $\lambda$ is a solution with $y = 1$ (resp. $y = 0$) to the encoded PB theory if and only if the corresponding membership set satisfies (resp. does not satisfy) $A$. We denote by $\tau_{pb}(\lambda \equiv A)$ the PB theory that encodes $A$, and an assignment $\lambda$ like the one above is said to be a model of $\tau_{pb}(\lambda \equiv A)$.

**Aggregates**

An aggregate atom (Son & Pontelli 2007) is of the form
\[
\text{aggr}\{\{a : a \in S\}\} \text{ op Result (3)}
\]
where $\text{aggr}$ is an aggregate function. Typical aggregate functions are $\{\text{COUNT}, \text{SUM}, \text{MIN}, \text{MAX}\}$. The set $S$ in the argument of an aggregate function is a set of atoms, on which the aggregate imposes the constraint. We call $S$ the domain of the aggregate. The relational operator op is from the set $\{=, \neq, <, >, \leq, \geq\}$ and $\text{Result}$ is either a symbolic or numeric constant.

**Example**

Consider $A = \text{COUNT}\{\{a = a \in S\}\} \geq k$.

For any $M \subseteq S$, $M \models A$ if and only if the number of elements in $M$ is greater than or equal to $k$.

Let $S = \{a_1, \ldots, a_n\}$. We introduce a PB variable $x_i$ for each $a_i$. A PB theory for this aggregate can be coded by
\[
\begin{align*}
\sum_{i=1}^{n} x_i & \geq y \cdot k \\
\sum_{i=1}^{n} x_i & \leq (1 - y) \cdot (k - 1) + y \cdot n
\end{align*}
\]
It can be verified that $\tau_{pb}(\lambda \equiv A)$ is satisfiable, and measured by the number of occurrences of variables and other symbols, the size of the PB theory is linear in $|S|$.

To contrast with the unfolding approach, consider a program with a single rule:
\[
h \leftarrow \text{COUNT}\{\{a : a \in S = \{a_1, a_2, a_3, a_4\}\} \geq 3.
\]
The unfolded normal program would be:
\[
h \leftarrow a_1, a_2, a_3, a_4. \quad h \leftarrow a_1, a_2, a_3, a_4,
\]
For an aggregate $\text{COUNT}\{\{a : a \in S\}\} \geq k$, the number of rules in the unfolded normal program is $\mathcal{C}^{\mathcal{K}}_{|S|}$, which is in general exponential in $|S|$.

**Global constraints**

A global constraint is a constraint that specifies a relation between a non-fixed number of variables. One of the most extensively studied global constraints is $\text{AllDifferent}(X, D)$ (van Hoeve & Katriel 2006), which specifies that each variable in the variable set $X$ must take a value from the domain $D$ and different variables must take distinct values.

Let $X = \{x_1, \ldots, x_m\}$ be a set of variables and $D = \{d_1, \ldots, d_n\}$ be the domain of the variables. We use atom $a(x_i, d_j)$ to represent that variable $x_i$ is assigned the value $d_j$. The global constraint $\text{AllDifferent}(X, D)$ can be represented by a c-atom $\text{AllDifferent}(S, C)$, where the domain of the constraint is $S = \{a(x, d) : x \in X, d \in D\}$, and $C$ is the collection of sets satisfying, for each set $\pi$ in $C$, (1) for
each \( i \), exactly one \( a(x_i, d_j) \) is in \( \pi \), for some \( d_j \), and (2) if \( a(x, d) \) and \( a(x', d') \) are in \( \pi \) and \( x \neq x' \), then \( d \neq d' \).

Let \( G = \text{AllDiff}(S, C) \) and \( y \) be a PB variable. We introduce the following additional PB variables.

- \( x_{ij} \) for each atom \( a(x_i, d_j) \).
- \( s_{i0}, s_{i1}, \) and \( s_{i2} \) for each variable \( x_i \). Intuitively, \( s_{i0} = 1 \) indicates that \( x_i \) is not assigned to any value; \( s_{i1} = 1 \) indicates that \( x_i \) is assigned to exactly one value; and \( s_{i2} = 1 \) indicates that \( x_i \) is assigned to at least two values.
- \( t_j \) for each value \( d_j \). Intuitively, if \( t_j = 1 \) then \( d_j \) is not assigned to any variable (not used) or assigned to one variable; if \( t_j = 0 \) then \( d_j \) is assigned to at least two variables.

The PB theory \( \tau_{pb}(y \equiv G) \) consists of the following PB constraints.

\[
\begin{align*}
\sum_{j=1}^{n} x_{ij} &\leq (1 - s_{i0}) \cdot n, \text{ for each } i, 1 \leq i \leq m \quad (6) \\
\sum_{j=1}^{n} x_{ij} &\geq 2 \cdot s_{i2}, \text{ for each } i, 1 \leq i \leq m \quad (7) \\
\sum_{j=1}^{n} x_{ij} &\leq n \cdot (1 - s_{i1}) + s_{i1}, \text{ for each } i, 1 \leq i \leq m \quad (8) \\
\sum_{j=1}^{n} x_{ij} &\geq s_{i1}, \text{ for each } i, 1 \leq i \leq m \quad (9) \\
\sum_{i=1}^{m} x_{ij} &\leq m \cdot (1 - t_j) + t_j, \text{ for each } j, 1 \leq j \leq n \quad (10) \\
\sum_{i=1}^{m} x_{ij} &\geq 2 \cdot (1 - t_j), \text{ for each } j, 1 \leq j \leq n \quad (11) \\
\sum_{i=1}^{m} s_{i1} + \sum_{j=1}^{n} t_j &\geq (m + n) \cdot y \quad (12) \\
\sum_{i=1}^{m} s_{i1} + \sum_{j=1}^{n} t_j &\leq (m + n - 1) \cdot (1 - y) + (m + n) \cdot y \quad (13) \\
\end{align*}
\]

Let \( \lambda \) be a solution to \( \tau_{pb}(y \equiv G) \). If \( \lambda(y) = 1 \), then \( \forall i. \lambda(s_{i1}) = 1 \) and \( \exists j. \lambda(t_j) = 1 \), due to (13). Then each variable is assigned to exactly one value, due to (9) and (10), and each value is assigned to no more than one variable due to (11).

If \( \lambda(y) = 0 \), then \( \exists i. \lambda(s_{i1}) = 0 \) or \( \exists j. \lambda(t_j) = 0 \), due to (14). If \( \lambda(s_{i0}) = 1 \), we have \( \lambda(s_{i1}) = 1 \) or \( \lambda(s_{i2}) = 1 \) due to (6). If \( \lambda(s_{i0}) = 1 \), then \( x_i \) is not assigned due to (7). If \( \lambda(s_{i1}) = 1 \), then \( x_i \) is assigned to more than one value due to (8). If \( \lambda(t_j) = 0 \), we have \( d_j \) is assigned to more than one variable, due to (12).

**Proposition 3.** Let \( G = \text{AllDiff}(S, C) \) and \( y \) be a PB variable. The following statements hold for the encoding \( \tau_{pb}(y \equiv G) \).

- The size of \( \tau_{pb}(y \equiv G) \) is linear in \(|S|\).
- The PB theory \( \tau_{pb}(y \equiv G) \) has at least one model.

Let \( \lambda \) be a model of \( \tau_{pb}(y \equiv G) \). The membership set \( M_\lambda = \{a(x_i, d_j) \in S : \lambda(x_{ij}) = 1\} \).

Note that the membership set of \( \lambda \) in this case is: \( M_\lambda = \{a(x_i, d_j) \in S : \lambda(x_{ij}) = 1\} \).

Clearly, the unfolding approach would be undesirable, as in the unfolded normal program it would have to list all solutions in \( C \) for the constraint \( \text{AllDiff}(S, C) \), which is exponential in \(|S|\).

The all-different constraint has many applications, for instance, to represent the pigeon-hole problem. A typical encoding of this problem by a normal program (Niemelä 1999) takes exponential time to decide unsatisfiability by an ASP solver. When the problem is encoded as a PB theory, unsatisfiability can be decided in polynomial time by a PB constraint solver (Chai & Kuehlmann 2005). Efficiency can be further improved by embedding special constraint propagators (Loúpe-Ortiz et al. 2003).

**References**


