Nineteenth National Conference on Artificial Intelligence (AAAI-04), pages 281-286, San Jose, CA, 2004

Loop Formulas for Circumscription

Joohyung Lee
Department of Computer Sciences
University of Texas, Austin, TX, USA
appsmurf@cs.utexas.edu

Fangzhen Lin
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
flin@cs.ust.hk

Abstract

Clark's completion is a simple nonmonotonic formalism and a special case of many nonmonotonic logics. Recently there has been work on extending completion with "loop formulas" so that general cases of nonmonotonic logics such as logic programs (under the answer set semantics) and McCain–Turner causal logic can be characterized by propositional logic in the form of "completion + loop formulas". In this paper, we show that the idea is applicable to McCarthy's circumscription in the propositional case. We also show how to embed propositional circumscription in logic programs and in causal logic, inspired by the uniform characterization of "completion + loop formulas".

Introduction

Clark's predicate completion (Clark 1978) is a simple and intuitive nonmonotonic formalism. Normally it is applicable when the knowledge base is given as a set of rules, and works when the rules do not yield a cycle.

Despite these limitations, surprisingly perhaps, predicate completion has been used to solve some problems that were thought to require more sophisticated nonmonotonic logics. For instance, Reiter (1991) showed that under certain reasonable assumptions, successor state axioms can be computed from action effect axioms by predicate completion, and thus solved the frame problem when there are no state constraints. For state constraints, Lin (1995) argued that they should be encoded using a notion of causality, and once they are encoded this way, successor state axioms can once again be computed using predicate completion for a class of causal rules that includes almost all of the benchmark planning domains (Lin 1995; 2003). The implementation of the fragment of McCain and Turner causal logic (McCain & Turner 1997) in the Causal Calculator (CCALC)[1] is also based on completion; CCALC has been successfully applied to several challenge problems in the theory of commonsense knowledge.

In logic programming where predicate completion is best known and commonly referred to as program completion semantics, its relationships with other semantics, especially the answer set semantics (also known as the stable model semantics) of Gelfond and Lifschitz (1988), have been studied quite extensively. First of all, it is well known that an answer set for a normal (non-disjunctive) logic program is also a model of its completion, while the converse, generally, does not hold. Fages (1994) showed that a certain syntactic condition, which is now called "tightness," is sufficient for establishing the equivalence between them. Erdem and Lifschitz (2003) generalized Fages' theorem and extended it to programs with nested expressions (in the sense of (Lifschitz, Tang, & Turner 1999)) in the bodies of rules.

Instead of looking for conditions that will guarantee the equivalence between the completion semantics and the answer set semantics, Lin and Zhao (2002) considered how to strengthen completion to make it equivalent to the answer set semantics. The idea is that, since cycles are what cause the mismatch between the models of the completion and the answer sets for a program, one should address the problems raised by them directly. Just like the completion semantics captures the intuition that for a predicate to be true, one of the bodies of the rules with the predicate as the head must be true, for each loop, Lin and Zhao associated with it a loop formula that captures the intuition that for the atoms in the loop to be true, there must be a rule whose head is in the loop, and whose body is true but its positive part does not have any atom in the loop. They showed that an interpretation is an answer set for a logic program iff it satisfies the completion and all loop formulas of the program. SAT-based answer set solvers ASSAT (Lin & Zhao 2002) and CMODELS-2 (Giunchiglia, Lierler, & Maratea 2004) are based on this idea.

As it turned out, program completion and loop formulas are not limited to non-disjunctive logic programs. Lee and Lifschitz (2003) extended the Lin/Zhao theorem to disjunctive logic programs and, more generally, to arbitrary programs with nested expressions. Lee (2004) showed that a similar result can be obtained for McCain and Turner causal logic and based on this, discussed the relationship between causal logic and logic programs.

Given these results, one wonders how far this idea of "completion + loop formulas" can go. Is it general enough to capture other nonmonotonic logics? In this paper, we answer this question positively for circumscription (McCarthy 1980; 1986) in the propositional case.

[1] http://www.cs.utexas.edu/users/tag/ccalc/

Logical Preliminaries

A *literal* is a (propositional) atom or the negation of an atom. A *(propositional) formula* is formed from literals using propositional connectives. A *clause* is a finite set of literals. We identify a clause C with the disjunction of its elements. It is well known that any formula can be transformed into an equivalent set of clauses. Given a set X of literals, $\overline{X}$ is the set of literals complementary to literals in X. In the following, given a language L, we identify an interpretation of L with a set of atoms in L.

We assume some variables ranging over 0-place connectives ($\top$, $\bot$), and quantify over them. For instance, we write $\forall z(z \vee p)$, where p is an atom, to denote $(\top \vee p) \wedge (\bot \vee p)$, and similarly $\exists z(z \vee p)$ to denote $(\top \vee p) \vee (\bot \vee p)$.

In the following, we sometimes write a formula A as $A(P)$ or $A(P, Q)$ for tuples P and Q of atoms. This way, when z is a tuple of variables and atoms of the same length as P, we use $A(z)$ or $A(z, Q)$ to denote the result of simultaneously replacing each element of P in A by the corresponding element of z. In this notation we sometimes allow a set in place of a tuple when there is no confusion. On the other hand, for convenience, we sometimes treat a tuple as the corresponding set.

For $P = (P_1, ...P_n)$, $Q = (Q_1, ...Q_n)$,

$P \leq Q$ stands for $\bigwedge_{1 \leq i \leq n}(P_i \supset Q_i)$,
$P = Q$ stands for $\bigwedge_{1 \leq i \leq n}(P_i \equiv Q_i)$,
$P < Q$ stands for $(P \leq Q) \wedge \neg(P = Q)$.

Let P and Z be tuples of atoms, and $A(P, Z)$ a formula. The circumscription of P in $A(P, Z)$ with atoms in Z allowed to vary, is the following formula:

$$A(P, Z) \wedge \neg \exists pz(A(p, z) \wedge p < P). \quad (1)$$

The formula is denoted by CIRC$[A(P, Z); P; Z]$, which may also be written as CIRC$[A(P); P]$ when Z is empty.

The following is a model-theoretic account of circumscription. For two interpretations I, J of the same signature, we write $I \leq^{P;Z} J$ [2] if

- I and J agree on all atoms that are not in P and Z, and
- for each $P_i \in P$, if $P_i \in I$ then $P_i \in J$.

We write $I <^{P;Z} J$ if $I \leq^{P;Z} J$ but not $J \leq^{P;Z} I$.

An interpretation I is a model of (1) iff it is *minimal on P with Z allowed to vary*, that is,

- I is a model of $A(P, Z)$, and
- there is no model J of $A(P, Z)$ such that $J <^{P;Z} I$.

Positive Dependency Graphs and Loops

A clause like $p \vee q \vee \neg r$ can be rewritten as $(r \wedge \neg q) \supset p$ or $(r \wedge \neg p) \supset q$. So if one wants to count ways an atom can be derived, this clause needs to be counted for both p and q. In general, if a clause C contains an atom p, then $\neg(C \setminus \{p\})$[3] implies p. This motivates the following definition of a (positive) dependency graph of a set of clauses.

[2] We may even write $I \leq^{P} J$ when Z is empty.

[3] $\neg(C \setminus \{p\})$ is equal to $\bigwedge_{l \in C \setminus \{p\}} \overline{l}$ according to our convention of identifying a clause with the disjunction of its literals.

Figure 1: The dependency graph of $A_1(P)$ on P

Definition 1 *The (positive) dependency graph of a set A of clauses on a set P of atoms is the directed graph G such that*

- *the vertices of G are the atoms in P, and*
- *for each clause C in A, G has an edge from each atom in $C \cap P$ to each atom in $\overline{C} \cap P$.*

A nonempty set L of atoms is called a *loop* of A on P if, for every pair p, p' of atoms in L, there exists a path of nonzero length from p to p' in the dependency graph of A on P such that all the vertices in this path belong to L.

For example, let $P = (p, q, r, s)$, and let $A_1(P)$ be the set of clauses: $\{p \vee \neg q,\ \neg p \vee q,\ r \vee \neg s,\ \neg r \vee s,\ p \vee r\}$. The dependency graph of $A_1(P)$ on P, shown in Figure 1, has two loops: $\{p, q\}$, $\{r, s\}$.

Definition 2 *Given a formula A, if B is a finite set of clauses that is logically equivalent to A, then the dependency graph of B on P is called the dependency graph of A on P under B. Similarly, loops of B on P are called loops of A on P under B.*

Computing Propositional Circumscription

We begin with the simple case when there are no constants (atoms) allowed to vary.

Fixed Constants

For any formula A and any tuple P of atoms, we denote by $A(\hat{P})$ the result of replacing every atom of P in A by $\bot$. The following proposition follows from the definition of circumscription straightforwardly:

Proposition 1 CIRC$[A(P); P]$ *is equivalent to the conjunction of $A(P)$ and*

$$\bigwedge K \ \supset \ \neg A(\hat{K}) \quad (2)$$

for each nonempty subset K of P, where $\bigwedge K$ stands for the conjunction of all elements in K.

In fact, including all subsets of P is unnecessary. We will now discuss how Proposition 1 can be improved. As mentioned above, a clause C that contains an atom p is equivalent to $\neg(C \setminus \{p\}) \supset p$. More generally, by Theorem 2 of (Lin 2001), for any formula A and any atom p, $A \models \neg A(\hat{p}) \supset p$. Furthermore, $\neg A(\hat{p})$ is the weakest such condition in the sense that for any other formula B that does not mention p, if $A \models B \supset p$, then $A \models B \supset \neg A(\hat{p})$. In other words, $\neg A(\hat{p})$ is the weakest sufficient condition for p in A. Now to form the completion of p, one just makes the weakest sufficient condition also its necessary condition.

Definition 3 *The completion of a formula $A(P)$ on P is the theory formed by taking the union of $A(P)$ and the set consisting of the formulas*

$$p \ \supset \ \neg A(\hat{p}) \quad (3)$$

for each $p \in P$.

It is easy to see that a model of CIRC$[A(P); P]$ is also a model of the completion of $A(P)$ on P, but the converse does not hold in general.

Theorem 1 *Let $A(P)$ be a formula, and B a finite set of clauses equivalent to A.* CIRC$[A(P); P]$ *is equivalent to the union of the completion of $A(P)$ on P and the set consisting of the formulas*

$$\bigwedge L \supset \neg A(\hat{L}) \tag{4}$$

for each loop L of $A(P)$ on P under B.

We call formula (4) the *(conjunctive) loop formula* of L for CIRC$[A(P); P]$ under B.

Note that compared with Proposition 1, Theorem 1 tells us that, for a model X of a formula $A(P)$ to be a model of CIRC$[A(P); P]$, it is sufficient to check whether every singleton subset of X and every loop of A in X satisfies formulas (2).

Clearly, a circumscription is equivalent to the completion when there are no loops:

Corollary 1 *For any formula $A(P)$, if there is an equivalent finite set B of clauses such that there are no loops of $A(P)$ on P under B, then* CIRC$[A(P); P]$ *is equivalent to the completion of $A(P)$ on P.*

For example, let $A_2(P)$ be the formula $p \vee q$ where $P = (p, q)$. $A_2(P)$ has three models, $\{p\}$, $\{q\}$ and $\{p, q\}$, among which $\{p, q\}$ is not a model of CIRC$[A_2(P); P]$ because $\{p\} <^P \{p, q\}$. Since there are no loops of $A_2(P)$ on P, Corollary 1 gives us another way of computing the models of the circumscription, by just computing the models of the completion, $\{p \vee q,\ p \supset \neg q,\ q \supset \neg p\}$.

$A_1(P)$ on $P = (p, q, r, s)$ in the previous section is an example that contains loops. $A_1(P)$ has three models, $\{p, q\}$, $\{r, s\}$ and $\{p, q, r, s\}$, among which the last is not a model of CIRC$[A_1(P); P]$ because $\{p, q\} <^P \{p, q, r, s\}$. Theorem 1 gives us another way of computing the models of the circumscription, by computing the models of the completion and loop formulas. First, the completion of $A_1(P)$ on P is the union of $A_1(P)$ and

$$\{p \supset q \vee \neg r \vee (r \not\equiv s),\ \ q \supset p \vee (r \not\equiv s) \vee \neg(p \vee r),\\ r \supset s \vee \neg p \vee (p \not\equiv q),\ \ s \supset r \vee (p \not\equiv q) \vee \neg(p \vee r)\}$$

from (3). We see that the models of the completion are the three models of $A_1(P)$ above. There are two loops of $A_1(P)$ on P, $\{p, q\}$ and $\{r, s\}$, and their loop formulas are $p \wedge q \supset \neg r$ and $r \wedge s \supset \neg p$ each. Among the models of the completion of $A_1(P)$ on P, $\{p, q, r, s\}$ does not satisfy any loop formula, so that it is not a model of CIRC$[A_1(P); P]$. The other two satisfy every loop formula, and are models of the circumscription.

Furthermore, it is easy to see that CIRC$[A_1; p, q]$ is equivalent to CIRC$[A_1; P]$. The completion of A_1 on (p, q) is a subset of the completion of A_1 on P, so that all three models of A_1 still satisfy the completion of A_1 on (p, q). $\{p, q, r, s\}$ still does not satisfy the only loop formula $p \wedge q \supset \neg r$ for CIRC$[A_1; p, q]$, which is also a loop formula for CIRC$[A_1; P]$.

According to Theorem 1, to compute the circumscription of P in $A(P)$ with all other atoms fixed, one first converts A to a finite set B of clauses, constructs the dependency graph of B on P, computes the loops of the dependency graph, and then computes the completion and loop formulas.

Computationally, there are a few problems here. First of all, without introducing new atoms, the size of B could be exponential in the number of atoms in A, although the size of the dependency graph of B is always polynomial. The second problem is that the number of loops can be exponential.

From the complexity point of view, we cannot do much about the second problem as SAT is in NP, and the problem of finding a model of circumscription is Σ_2^P-hard.

For the first problem, one can introduce new atoms when converting A to its clausal form to avoid combinatorial blow-up when distributing "$\vee$" over "$\wedge$." One can also try to construct a dependency graph directly without actually generating the set of clauses. However, a formula may be equivalent to many different sets of clauses, which in turn may yield different dependency graphs. Intuitively, for our purpose, everything being equal, the fewer loops that a dependency graph has the better. We believe that in general, given a formula A, it is computationally hard to find a set of clauses for A that would yield the smallest number of loops. But we do not have a proof, and it remains an open question.

Varied Constants

The following fact (Lifschitz 1985, Proposition 2) shows how to eliminate varied constants in general:

Fact 1 CIRC$[A(P, Z); P; Z]$ *is equivalent to*

$$A(P, Z) \wedge \text{CIRC}[\exists z A(P, z); P].$$

Thus circumscription with varied constants reduces to the basic case where Theorem 1 applies.

Alternatively, we have the following result by modifying the definitions of a dependency graph and a loop. Intuitively, paths are allowed to have varied atoms and their negations as intermediate vertices.

Definition 4 *The (positive) dependency graph of a set A of clauses on P with Z allowed to vary is the directed graph G such that*

- *the vertices of G are the literals in $P \cup Z \cup \overline{Z}$, and*
- *for each clause C in A, G has an edge from each literal l in $C \cap (P \cup Z \cup \overline{Z})$ to each literal in $\overline{C \setminus \{l\}} \cap (P \cup Z \cup \overline{Z})$.*

A nonempty set L_0 of literals is called an *extended loop* of A on P with Z allowed to vary if, for every pair p, p' of literals in L_0, there exists a path of nonzero length from p to p' in the dependency graph of A on P with Z allowed to vary such that all the vertices in this path belong to L_0. A nonempty set $L = L_0 \setminus (Z \cup \overline{Z})$ is called a *loop* of A on P with Z allowed to vary.

For example, let $P = \{p, q\}$, $Z = \{z\}$ and $A_3(P, Z)$ be the set of clauses $\{p \supset \neg z,\ \neg z \supset q,\ q \supset p\}$. The dependency graph of $A_3(P, Z)$ on P with Z allowed to vary is shown in Figure 2. It has only one extended loop, $\{p, q, \neg z\}$, and so has only one loop, $\{p, q\}$.

The above definition of a loop is intuitive, but here is an alternative, more economical definition in terms of the number of loops we get. For a set A of clauses, by $S(A)$ we

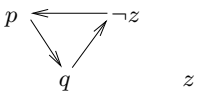

Figure 2: The dependency graph of A_2 on P with varied z

denote the set of clauses C in A such that C does not contain a pair of complementary literals and there is no proper subset C' of C such that $C' \in A$. Given a set A of clauses and a tuple of atoms $Z = (z_1, \ldots, z_n)$, B_Z is defined as follows.

- $B_{()} = A$,
- $B_{(z_1,\ldots,z_k)} = S(B_{(z_1,\ldots,z_{k-1})}) \cup \{C_1 \cup C_2 : C_1 \cup \{z_k\}, C_2 \cup \{\neg z_k\} \in S(B_{(z_1,\ldots,z_{k-1})})\}$.

We call $S(B_Z)$ the Z-collapsed set of A.

Let A_Z be the Z-collapsed set of A and let A'_Z be the set of clauses in A_Z that do not mention Z. Loops of the dependency graph of A'_Z on P are called loops of A on P with Z allowed to vary. A loop according to this alternative definition is also a loop according to the definition above, but not vice versa. The following result holds under either definition.

Here is a counterpart of Proposition 1, where we allow some atoms to vary.

Proposition 2 *CIRC$[A(P,Z); P; Z]$ is equivalent to the conjunction of $A(P,Z)$ and*

$$\bigwedge K \supset \forall z \neg A(\hat{K}, z)$$

for each nonempty subset K of P.

Definition 5 *The completion of a formula $A(P,Z)$ on P with Z allowed to vary is the theory formed by taking the union of $A(P,Z)$ and the set consisting of the formulas*

$$p \supset \forall z \neg A(\hat{p}, z)$$

for each $p \in P$.

Theorem 2 *Let $A(P,Z)$ be a formula, and B a finite set of clauses equivalent to $A(P,Z)$. CIRC$[A(P,Z); P; Z]$ is equivalent to the union of the completion of $A(P,Z)$ on P with Z allowed to vary and the set consisting of the formulas*

$$\bigwedge L \supset \forall z \neg A(\hat{L}, z) \tag{5}$$

for each loop L of $A(P,Z)$ on P with Z allowed to vary.

We call formula (5) the *(conjunctive) loop formula* of L for CIRC$[A(P,Z); P; Z]$ under B.

$A_3(P,z)$ above has two models, $\{p,q\}$ and $\{z\}$, but the first is not a model of CIRC$[A_3(P,z); P; z]$, though it satisfies the completion of $A_3(P,z)$ on P, $\{p \supset q,\ q \supset p\}$. There is only one loop of $A_3(P,z)$ on P with z allowed to vary, $\{p,q\}$, and its loop formula is $p \wedge q \supset \bot$. Only the interpretation $\{z\}$ satisfies both the completion and the loop formula.

Similarly to Corollary 1, we get the following corollary to Theorem 2:

Corollary 2 *For any formula $A(P,Z)$, if there is an equivalent finite set B of clauses such that there are no loops of $A(P,Z)$ on P with Z allowed to vary under B, then CIRC$[A(P,Z); P; Z]$ is equivalent to the completion of $A(P,Z)$ on P with Z allowed to vary.*

Relating to Some Known Results

Reiter (1982) was the first to show some relationships between Clark's predicate completion and circumscription. He proved that if a theory is Horn in a predicate P, then the circumscription of P logically entails P's completion. In the propositional case, our new contributions are as follows. First, we extended the notion of completion to arbitrary theories, not just those that are Horn. Second, we showed that it is always the case that circumscription of an atom p entails p's completion. Third, we gave a general syntactic condition that guarantees the equivalence between circumscription and completion. Lastly, and most importantly, we defined notions of loops and loop formulas, and showed that circumscription can be reduced to completion plus loop formulas.

Traditionally, computing circumscription means finding classes of first-order theories for which circumscription is equivalent to first-order theories (e.g. (Lifschitz 1985; 1987) and (Doherty, Łukaszewicz, & Szałas 1997)). In the propositional case, circumscription is always equivalent to a propositional theory. So the problem in the propositional case is not whether circumscription can be reduced to propositional logic, but how well this can be done.

In logic programming, when the positive dependency graph of a program has no loops, the program is called "tight", and the answer sets for such a tight program are exactly the models of the completion. Corollaries 1 and 2 in this paper address a similar syntactic condition for circumscription in the propositional case. In the following, we show that some of the known results about circumscription can be easily explained by our corollaries.

We say an occurrence of an atom in a formula is *positive* if it is in the range of an even number of negations, and *negative* otherwise (assuming that $\supset$ and $\equiv$ have been eliminated in favor of other connectives). A formula $A(P)$ is *positive* (relative to P) if all occurrences of P in it are positive, and *negative* if all occurrences of P are negative.[4] We see that if $A(P)$ is positive (or negative) relative to P, then there is an equivalent set of clauses whose dependency graph on P has no loops, so that CIRC$[A(P); P]$ is equivalent to the completion of $A(P)$ on P. The result can also be extended when some atoms are allowed to vary. The following is the propositional case of Proposition 2a from (Lifschitz 1987).

Proposition 3 *If $A(p,Z)$ is positive relative to an atom p, then CIRC$[A(p,Z); p; Z]$ is equivalent to*

$$A(p,Z) \wedge \neg\exists z(p \wedge A(\bot, z)). \tag{6}$$

Proof.

$$\begin{aligned} \text{CIRC}[A(p,Z); p; Z] &\equiv A(p,Z) \wedge (p \supset \forall z \neg A(\hat{p}, z)) \\ &\equiv A(p,Z) \wedge \neg\exists z(p \wedge A(\bot, z)). \end{aligned}$$

[4] A formula is both positive and negative relative to P if it doesn't contain P.

The first equivalence holds because there exists a set of clauses equivalent to $A(p, Z)$ such that the dependency graph of $A(p, Z)$ under it has no loops, so that Corollary 2 applies. ∎

The form of circumscription in (6) is the propositional case of what Lifschitz (1987) called *pointwise circumscription.*

Proposition 3 can be extended to parallel circumscription:

Proposition 4 *(Lifschitz 1987, Proposition 5) If P is a tuple of atoms $p_1, \ldots, p_n$ and $A(P, Z)$ is positive relative to each p_i, then* $\mathrm{CIRC}[A(P, Z); P; Z] \equiv \bigwedge_i \mathrm{CIRC}[A(P, Z); p_i; Z]$.

Proof. Again, this is easy to see from the facts that there exists a set of clauses equivalent to $A(P, Z)$ such that the dependency graph of $A(P, Z)$ under it has no loops, and that the completion of $A(P, Z)$ on P is equivalent to the conjunction of each completion of $A(p_i, Z)$ on p_i. ∎

In fact, Corollary 2 presents us a much more general condition than the one in Proposition 3 (and 4) that shows the equivalence between the circumscription of the form (1) and pointwise circumscription. For example, consider $\mathrm{CIRC}[(p \supset q) \wedge (q \supset r); q, r]$. There are no loops of the formula, and consequently the circumscription is equivalent to its corresponding pointwise circumscription according to Corollary 2. But the conjunction in the formula cannot be divided into two parts so that q,r are positive in one part and negative in the other, so Proposition 3 does not apply.

Sometimes it is easy to observe that the dependency graphs of some circumscriptions have the same loops.

Proposition 5 *(Lifschitz 1987) If $B(P)$ is negative relative to P, then* $\mathrm{CIRC}[A(P) \wedge B(P); P]$ *is equivalent to* $\mathrm{CIRC}[A(P); P] \wedge B(P)$.

Proof. There exists a set $B'(P)$ of clauses which is equivalent to $B(P)$ and is negative relative to P. Notice that $A(P) \wedge B'(P)$ and $A(P)$ have the same dependency graph on P, hence have the same loops. We see that the completion and loop formulas of $A(P) \wedge B'(P)$ on P and those of $A(P)$ on P are equivalent to each other when we notice that $B'(\hat{P})$ is entailed by $B'(P)$. ∎

However, if some atoms are allowed to vary, then $A(P) \wedge B(P)$ may have more loops than $A(P)$ ($A_3(P, z)$ for example), and $B(P)$ may not be easily "factored out".

Embedding Circumscription in Other Nonmonotonic Logics

As mentioned in the introduction, the idea of "completion + loop formulas" has been applied to logic programs in (Lin & Zhao 2002; Lee & Lifschitz 2003) and to McCain–Turner causal logic in (Lee 2004). The characterizations of these nonmonotonic logics in terms of propositional logic are useful tools for comparing the formalisms. Based on this idea, Lee (2004) showed how to embed logic programs in causal logic.

Inspired by a similar characterization for circumscription, we show how to relate circumscription to logic programs and to causal logic. The propositions below can be proved by turning each formalism into "completion+loop formulas" and then show that the translations are equivalent to each other.

Logic Programming

For a literal l, by l_{not} we denote *not* l if l is positive, and $\overline{l}$ otherwise. Let $A(P)$ be a finite set of clauses. For each clause $C \in A(P)$, the corresponding rule C_P is

$$\mathop{;}_{p \in C \cap P} p \leftarrow \mathop{,}_{q \in C \setminus P} q_{not}$$

By σ_A we denote the set of all atoms that occur in A.

Proposition 6 *For any finite set $A(P)$ of clauses, a set of atoms is a model of* $\mathrm{CIRC}[A(P); P]$ *iff it is an answer set for the logic program*

$$\{C_P \ : \ C \in A(P)\} \ \cup \ \{a \ ; \ \textit{not}\ a \ : \ a \in \sigma_A \setminus P\}.$$

We may even embed circumscription with varied constants in logic programs. Given a finite set $A(P, Z)$ of clauses, $\Pi_{A;P;Z}$ is the following logic program. Let A_Z be the Z-collapsed set of A.

- For each clause C in A_Z that does not mention Z, $\Pi_{A;P;Z}$ has the rule C_P,
- For every other clause C in A_Z, $\Pi_{A;P;Z}$ has the rule

$$\leftarrow \mathop{,}_{p \in C} p_{not},$$

- For each atom $a \in \sigma_A \setminus P$, $\Pi_{A;P;Z}$ has the rule $a \ ; \ \textit{not}\ a$.

Proposition 7 *For any finite set $A(P, Z)$ of clauses, a set of atoms is a model of* $\mathrm{CIRC}[A(P, Z); P; Z]$ *iff it is an answer set for the logic program* $\Pi_{A;P;Z}$.

McCain–Turner Causal Logic

McCain–Turner causal logic is a useful formalism for describing effects of actions, and is a basis of action language $\mathcal{C}$+ (Giunchiglia *et al.* 2004), which is a high level language for describing transition systems.

Proposition 8 *Let $A(P)$ be a finite set of clauses. An interpretation is a model of* $\mathrm{CIRC}[A(P); P]$ *iff it is a model of the causal theory*

$$\{C \cap (P \cup \overline{P}) \Leftarrow \neg(C \setminus (P \cup \overline{P})) : C \in A\}$$
$$\cup \{\neg r \Leftarrow \neg r : r \in \sigma_A\} \cup \{r \Leftarrow r : r \in \sigma_A \setminus P\}$$

whose signature is σ_A.

Proposition 8 is similar to the propositional case of Proposition 1 of (Lifschitz 1997). Extending the result to circumscription with varied constants is similar to the case with logic programs and is omitted here due to the lack of space.

Conclusion

To recast, the following are our main contributions (all in the propositional case):

- Extend completion to arbitrary theories.

- Introduce a notion of a positive dependency graph for a finite set of clauses, and based on it, notions of loops and loop formulas.
- Show that circumscription is equivalent to completion plus loop formulas, and based on this result, embed circumscription in other nonmonotonic logics.

These results are of both theoretical interest and practical importance. A major obstacle in implementing a reasoning system for propositional circumscription is that checking if an assignment is a model of a circumscription is NP-hard. In comparison, checking if an assignment is a model of a formula in propositional logic or an answer set for a nondisjunctive logic program can be done efficiently. By Theorems 1 and 2, if a given formula has no loop or has only a polynomial number of loops and these loops can be computed in polynomial time, then checking if an assignment is a model of circumscription can be done in polynomial time as well. Hopefully, many applications of circumscription will belong to this class, just as many logic programs for practical problems are "tight" or "tight on the models of completion". Alternatively, one could implement circumscription in logic programming systems like DLV and GnT using Propositions 6 and 7.[5]

Note that we could have defined a loop to allow a path of length 0, which corresponds to each atom. In this sense, the second part of completion is just a set of loop formulas, and circumscribing a theory can be regarded as just adding loop formulas to it.

For future work, there is a need to better understand how loops can be computed. More importantly, there is a need to extend the results of this paper to the first-order case.

Acknowledgements

We are grateful to Selim Erdoğan, Paolo Ferraris, Vladimir Lifschitz, Hudson Turner and the anonymous referees for useful comments. Joohyung Lee was partially supported by the Texas Higher Education Coordinating Board under Grant 003658-0322-2001. Fangzhen Lin's work has been supported in part by HK RGC under CERG HKUST6205/02E, and by the NSFC under its major research program "Basic Theory and Core Techniques of Non-Canonical Knowledge".

References

Clark, K. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. New York: Plenum Press. 293–322.

Doherty, P.; Łukaszewicz, W.; and Szałas, A. 1997. Computing circumscription revisited: A reduction algorithm. *Journal of Automated Reasoning* 18(3):297–336.

Erdem, E., and Lifschitz, V. 2003. Tight logic programs. *Theory and Practice of Logic Programming* 3:499–518.

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

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. Fifth International Conference and Symposium on Logic Programming*, 1070–1080.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153:49–104.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2004. SAT-based answer set programming. In *Proc. AAAI-04*. this proceeding.

Lee, J., and Lifschitz, V. 2003. Loop formulas for disjunctive logic programs. In *Proc. Nineteenth Int'l Conf. on Logic Programming*, 451–465.

Lee, J. 2004. Nondefinite vs. definite causal theories. In *Proc. 7th Int'l Conference on Logic Programming and Nonmonotonic Reasoning*, 141–153.

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

Lifschitz, V. 1985. Computing circumscription. In *Proc. IJCAI-85*, 121–127.

Lifschitz, V. 1987. Pointwise circumscription. In Ginsberg, M., ed., *Readings in nonmonotonic reasoning*. San Mateo, CA: Morgan Kaufmann. 179–193.

Lifschitz, V. 1997. On the logic of causal explanation. *Artificial Intelligence* 96:451–465.

Lin, F., and Zhao, Y. 2002. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proc. AAAI-02*, 112–117.

Lin, F. 1995. Embracing causality in specifying the indirect effects of actions. In *Proc. IJCAI-95*, 1985–1993.

Lin, F. 2001. On strongest necessary and weakest sufficient conditions. *Artificial Intelligence* 128(1-2):143–159.

Lin, F. 2003. Compiling causal theories to successor state axioms and STRIPS-like systems. *Journal of Artificial Intelligence Research* 19:279–314.

McCain, N., and Turner, H. 1997. Causal theories of action and change. In *Proc. AAAI-97*, 460–465.

McCarthy, J. 1980. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence* 13:27–39,171–172.

McCarthy, J. 1986. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence* 26(3):89–116.

Reiter, R. 1982. Circumscription implies predicate completion (sometimes). In *Proceedings of AAAI–82*, 418–420.

Reiter, R. 1991. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. San Diego, CA: Academic Press. 359–380.

[5] Strictly speaking, the current versions of those systems do not allow negation as failure in the head of a rule, so it cannot handle such a rule as a; *not* a. However there is a well-known technique to "simulate" rules of this kind using additional atoms.