Available online at www.sciencedirect.com
SCIENCE DIRECT®
Artificial Intelligence 170 (2006) 160–185
Artificial Intelligence
www.elsevier.com/locate/artint

Loop formulas for circumscription

Joohyung Lee [a], Fangzhen Lin [b,*]

[a] *Department of Computer Science and Engineering, Arizona State University, AZ, USA*
[b] *Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong*

Received 4 April 2005; received in revised form 13 September 2005; accepted 14 September 2005

Available online 2 November 2005

Abstract

Clark's completion is a simple nonmonotonic formalism and a special case of several 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, with Lifschitz's pointwise circumscription playing the role of completion. We also show how to embed propositional circumscription in logic programs and in causal logic, inspired by the uniform characterization of "completion + loop formulas".

Keywords: Nonmonotonic reasoning; Commonsense reasoning; Knowledge representation; Circumscription; Clark's completion; Loop formulas; Logic programming

1. Introduction

Clark's predicate completion [5] 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".

* Corresponding author.
E-mail address: flin@cs.ust.hk (F. Lin).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2005.09.003

Despite these limitations, surprisingly perhaps, predicate completion has been used to solve many problems that were thought to require more sophisticated nonmonotonic logics. For instance, Reiter [34] 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 [25] 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 [25,27].

For the "definite" fragment of McCain–Turner causal logic [10,28], the problem of determining whether a theory is consistent can be reduced to the satisfiability problem for propositional logic by the process of "literal completion"—a translation similar to Clark's completion. This idea has led to the creation of the Causal Calculator (CCALC),[1] a system for representing commonsense knowledge about action and change. After turning a definite causal theory into a classical propositional theory, CCALC finds the models of the latter by invoking a satisfiability solver, such as CHAFF,[2] SATO[3] and RELSAT,[4] which in turn correspond to the models of the given causal theory. CCALC has been successfully applied to several challenge problems in the theory of commonsense knowledge [1,4,18,23] and to the formalization of multi-agent computational systems [2,3].

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 [9], have been studied quite extensively. First of all, it is well known that an answer set for a normal logic program is also a model of its completion, while the converse, generally, does not hold. Fages [8] showed that a certain syntactic condition, which is now called "tightness", is sufficient for establishing the equivalence between them. Erdem and Lifschitz [7] generalized Fages' theorem and extended it to programs with nested expressions (in the sense of [17]) 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 [24] considered how to strengthen completion to make it equivalent to the answer set semantics. The idea is that, since the presence of cycles is what causes the mismatch between the models of the completion and the answer sets for a program, one should address the problem raised by them directly. The completion semantics captures the intuition that for an atom to be true, one of the bodies of the rules with the atom as the head must be true. Similarly, Lin and Zhao associated with each loop a "loop formula" that captures the intuition that for the atoms in a loop to be true, there must be a rule whose head belongs to the loop, and whose body is true but its positive part does not have any atom in the loop. They showed that a set of atoms is an answer set for a nondisjunctive logic program iff it is a model of the completion and all loop formulas of the program. This idea allows SAT solvers to be used for finding

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

[2] http://www.ee.princeton.edu/~chaff/.

[3] http://www.cs.uiowa.edu/~hzhang/sato.html.

[4] http://www.almaden.ibm.com/cs/people/bayardo/resources.html.

answer sets and thus has led to the creation of SAT-based answer set solvers ASSAT [24] and CMODELS-2 [11].

As it turned out, program completion and loop formulas are not limited to nondisjunctive logic programs. Lee and Lifschitz [12] extended the Lin/Zhao theorem to disjunctive logic programs and, more generally, to arbitrary programs with nested expressions. Lee [13] showed that a similar result can be obtained for McCain and Turner causal logic and based on this, showed how to embed logic programs in causal logic.

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 [29,30] in the propositional case. Thus it is interesting to observe that these apparently different nonmonotonic formalisms have a uniform view of "completion + loop formulas". Using this idea, we show how to embed circumscription in logic programs and in McCain–Turner causal logic.

Hopefully, these results will lead to good implementations of propositional circumscription using SAT solvers and/or answer set solvers. This would be a significant progress in nonmonotonic reasoning as circumscription has found applications in commonsense reasoning, model-based diagnoses, discourse understanding, and others. While many of these applications in general make use of first-order circumscription, they can be solved using propositional circumscription when their domains are given and finite.

This paper is organized as follows. In Section 2, we introduce some notations that we will use in the rest of the paper, and recast the definition of circumscription in the propositional case. In Section 3, we discuss Clark's completion, and compare it with Lifschitz's pointwise circumscription [20], as the latter will serve as "completion" for our purpose. In Section 4, we introduce the notion of a loop via the notion of a dependency graph. Section 5 contains the main technical results of the paper, which shows that circumscription can be characterized by completion plus loop formulas. It also discusses some related work. Based on the results in Section 5, Section 6 shows how circumscription can be embedded in logic programs under the answer set semantics and in McCain–Turner causal logic. We conclude in Section 7.

2. 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.

We use variables that range over 0-place connectives $\top$ and $\bot$, and quantify over them. For instance, if $A(z, p_1, \ldots, p_k)$ is a propositional formula built with propositional variables $z, p_1, \ldots, p_k$, we write $\forall z A(z, p_1, \ldots, p_k)$ to denote the formula $A(\top, p_1, \ldots, p_k) \wedge A(\bot, p_1, \ldots, p_k)$, and similarly $\exists z A(z, p_1, \ldots, p_k)$ to denote the formula $A(\top, p_1, \ldots, p_k) \vee A(\bot, p_1, \ldots, p_k)$.

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 X is a tuple of variables and atoms of the same length as P, we use $A(X)$ or $A(X, Q)$ to denote the result of simultaneously replacing all elements

of P in A by the corresponding elements of X. We sometimes identify a tuple with the corresponding set when there is no confusion.

For $P = (p_1, \ldots, p_n)$, $Q = (q_1, \ldots, q_n)$,

$$P \leqslant Q \quad \text{stands for} \quad \bigwedge_{1 \leqslant i \leqslant n} (p_i \supset q_i),$$

$$P = Q \quad \text{stands for} \quad \bigwedge_{1 \leqslant i \leqslant n} (p_i \equiv q_i),$$

$$P < Q \quad \text{stands for} \quad (P \leqslant Q) \wedge \neg(P = Q).$$

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:

$$A(P, Z) \wedge \neg\exists XY\big(A(X, Y) \wedge X < P\big). \tag{1}$$

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

The second conjunct of formula (1) is actually a propositional formula as in the following example:

$$\begin{aligned}\mathrm{CIRC}[p \vee q; p] &= (p \vee q) \wedge \neg\exists x\big((x \vee q) \wedge (x \supset p) \wedge (x \not\equiv p)\big)\\ &= (p \vee q) \wedge \neg\big[\big((\top \vee q) \wedge (\top \supset p) \wedge (\top \not\equiv p)\big)\\ &\quad \vee \big((\bot \vee q) \wedge (\bot \supset p) \wedge (\bot \not\equiv p)\big)\big]\\ &\equiv (p \vee q) \wedge \neg(p \wedge q).\end{aligned} \tag{2}$$

The models of the circumscription are $\{p\}$ and $\{q\}$.[5]

There is a weaker notion of circumscription that will turn out to be important here. This is Lifschitz's *pointwise* circumscription [20]. In the propositional case, given an atom p and a tuple Z of atoms, the pointwise circumscription of p in $A(p, Z)$ with Z allowed to vary is

$$A(p, Z) \wedge \neg\exists xY\big(A(x, Y) \wedge x < p\big), \tag{3}$$

and the pointwise circumscription of a tuple P of atoms in A with Z allowed to vary is the conjunction of the pointwise circumscription of each $p \in P$ in A with Z allowed to vary. It can be seen that (3) is equivalent to $\mathrm{CIRC}[A; p; Z]$. Thus the pointwise circumscription of a tuple P of atoms in A with Z allowed to vary is $\bigwedge_{p \in P} \mathrm{CIRC}[A; p; Z]$.

For two interpretations (i.e., truth assignments) I, J of the same signature, we write $I \leqslant^{P;Z} J$ [6] 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 $I \leqslant^{P;Z} J$ but not $J \leqslant^{P;Z} I$.

[5] We identify an interpretation with the set of atoms that are true in it.

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

The following proposition (Proposition 1 from [19]) provides a model-theoretic account of circumscription.

Proposition 1. *An interpretation I is a model of* CIRC$[A; P; Z]$ *iff it is* minimal[7] on P with Z allowed to vary, *that is,*

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

For example, among the three models of $p \vee q$, $\{p, q\}$ is not a model of CIRC$[p \vee q; p]$ because $\{q\} <^{p} \{p, q\}$.

3. Completion and pointwise circumscription

Clark's completion turns "if" conditions into "if and only if" conditions. For instance, given the following rules about *Wet*,

$$\textit{Raining} \supset \textit{Wet}$$

$$\textit{SprinklerOn} \supset \textit{Wet},$$

Clark's completion, when applied to *Wet*, yields

$$\textit{Wet} \equiv \textit{Raining} \vee \textit{SprinklerOn}.$$

The underlying assumption here is what has been called the closed world assumption [32]: the given knowledge base contains complete knowledge about what can make *Wet* true. In particular, if there is no rule about a proposition, say p, then it is assumed to be false: $p \equiv \bot$.

In general, we have the following definition.

Definition 1. Let A be a set of formulas of the form $G \supset p$ where G is a formula and p is an atom, and suppose that the following are the only implications in A with the consequent q: $G_1 \supset q, \ldots, G_n \supset q$. Then Clark's completion of A on q is $q \equiv G_1 \vee \cdots \vee G_n$. Notice that when $n = 0$, this is $q \equiv \bot$. For a set P of atoms, Clark's completion of A on P is the conjunction of Clark's completions of A on p, for all $p \in P$.

Logically, Clark's completion on q is equivalent to adding to A the sentence $q \supset G_1 \vee \cdots \vee G_n$, i.e., making the *weakest* sufficient condition of q also its necessary condition.

Unfortunately, Clark's completion is not quite fit here for the following reasons. One problem is that it is defined for formulas of the form $G \supset p$, rather than for arbitrary formulas. Thus Clark's completion can be compared with circumscription only when formulas are given in this special form. Moreover, Clark's completion is sensitive to the syntactic form of the given knowledge base. For instance, while $\neg p \supset q$ and $\neg q \supset p$ are logically equivalent, their Clark's completions on $\{p, q\}$ are not.

[7] Recall Footnote 5.

Fortunately, there is another notion from the literature, Lifschitz's pointwise circumscription [20], that generalizes Clark's completion, and is syntax independent. To see this, notice first that, as we have mentioned above, Clark's completion on p essentially turns a "weakest" sufficient condition of p into its necessary condition. Formally, we can define the notion of *weakest sufficient conditions* as follows [26].

Given a propositional formula A and an atom q, a formula φ that does not mention q is called a *weakest sufficient condition* of q if

- $A \models \varphi \supset q$, and
- for any other formula ψ such that it does not mention q and $A \models \psi \supset q$, we have that $A \models \psi \supset \varphi$.

For any A and q, weakest sufficient conditions of q always exist and are unique up to logical equivalence under A. In the following, given a formula A and a set P of atoms, we use $A_{\bot}^{P}$ to denote the result of replacing all occurrences of atoms from P in A by $\bot$. In this section, P will always be a singleton.

Proposition 2 [26]. *For any formula A and any atom q, the formula $\neg A_{\bot}^{\{q\}}$ is a weakest sufficient condition of q.*

Thus we could extend Clark's completion to arbitrary formulas as follows: Given any formula A and any set P of atoms, the generalized Clark's completion of A on P is the conjunction of A and formulas $p \supset \neg A_{\bot}^{\{p\}}$ for all $p \in P$. As it turned out, this is exactly Lifschitz's pointwise circumscription of P in A.

Proposition 3. *For any formula A and any atom q,* CIRC$[A; q]$ *is equivalent to $A \wedge (q \supset \neg A_{\bot}^{\{q\}})$.*

Proof. $\neg\exists x(A(x) \wedge x < q)$ is equivalent to

$$\neg\big[\big(A(\top) \wedge \top < q\big) \vee \big(A(\bot) \wedge \bot < q\big)\big],$$

which is equivalent to

$$\neg\big[\big(A(\top) \wedge q \wedge \neg q\big) \vee \big(A(\bot) \wedge q\big)\big],$$

which is equivalent to $q \supset \neg A_{\bot}^{\{q\}}$. □

So in the following, we shall use the term "completion" and "pointwise circumscription" interchangeably, and for our purpose here, we also call the pointwise circumscription of P in A with Z allowed to vary, the *completion of A on P with Z allowed to vary*.

The following proposition shows that the completion of A is equivalent to Clark's completion of B for some B that is equivalent to A.

Proposition 4. *Let A be a formula, and P the set of atoms in it. There is a formula B of the required form in the definition of Clark's completion such that A is equivalent to B, and the completion of A on P is equivalent to Clark's completion of B on P.*

Proof. Let B_0 be a set of non-tautological clauses that is equivalent to A. Now for each $q \in P$, each clause that contains q, say $G \vee q$, generate the rule $\neg G \supset q$, and let B be the resulting theory. Clearly B is equivalent to B_0, thus to A as well. Clark's completion of B on P is the conjunction of B and $q \supset \neg B_{\bot}^{\{q\}}$ for all $q \in P$. But B is equivalent to A, and $B_{\bot}^{\{q\}}$ is equivalent to $A_{\bot}^{\{q\}}$. So the proposition follows. □

Lifschitz [20] showed that circumscription always entails pointwise circumscription (completion), but the converse does not hold in general. For instance, CIRC$[p \equiv q; p, q]$ yields $\neg p \wedge \neg q$. But the pointwise circumscription of (p, q) in $p \equiv q$ is equivalent to $p \equiv q$. The reason is that pointwise circumscription (completion), like Clark's completion, does not handle "loops", such as the one between p and q in the formula $p \equiv q$. The main purpose of this paper is to formally define what we mean by loops in a formula, associate a constraint with each loop, and show that propositional circumscription is equivalent to the conjunction of pointwise circumscription and the constraints for all loops. We begin by defining the notion of a loop. Like in logic programs, we appeal to the notion of a dependency graph.

4. 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 it can be rewritten as $\neg(C \backslash \{p\}) \supset p$.[8] This motivates the following definition of a dependency graph of a set of clauses.

Definition 2. Let A be a set of clauses, and P a set of atoms. The (*positive*) *dependency graph* of A on P is the directed graph such that

- the vertices are the atoms in P, and
- an edge goes from vertex p to vertex q if there is a clause C in A such that $p, \neg q \in C$.

A nonempty subset L of P is called a *loop* of A on P if, for every pair p, q of atoms in L, there exists a path of non-zero length from p to q in the dependency graph of A on P such that all vertices in this path belong to L. For example, consider the following set A_1 of clauses:

$$\{p \vee \neg q,\ \neg p \vee q,\ r \vee \neg s,\ \neg r \vee s,\ p \vee r\}.$$

This theory has two loops on $\{p, q, r, s\}$, $\{p, q\}$ and $\{r, s\}$, as shown by its dependency graph in Fig. 1.

We can make the definition of a loop slightly more general by dropping the requirement that the paths be of non-zero length. That is, a nonempty subset L of P is called a *generalized loop* of A on P if, for every pair p, q of atoms in L, there exists a path from p to q

[8] Recall that $\neg(C \backslash \{p\})$ is equal to $\bigwedge_{l \in C \backslash \{p\}} \bar{l}$ where $\bar{l}$ denotes a literal complementary to l, according to the convention of identifying a clause with the disjunction of its literals.

Fig. 1. The dependency graph of A_1 on $\{p, q, r, s\}$.

in the dependency graph of A on P such that all vertices in this path belong to L. In other words, a nonempty subset L of P is a generalized loop of A on P if the subgraph of the dependency graph of A on P induced by L is strongly connected. Note that a singleton subset $\{p\}$ of P is a loop if and only if there is an edge from p to itself in the dependency graph. On the other hand, every singleton subset of atoms is a generalized loop, regardless of the presence of such edges. For instance, in addition to the loops that we found before, A_1 has four other generalized loops on $\{p, q, r, s\}$: $\{p\}$, $\{q\}$, $\{r\}$, $\{s\}$. As we will see later, the notion of a generalized loop simplifies the statements of many of our results such as Theorem 1 below. It will also allow us to view loop formulas as a generalization of completion.

For an arbitrary formula A, its dependency graph can be defined by considering a set B of clauses that is equivalent to A:

Definition 3. Given a formula A, if B is a finite set of clauses that is equivalent to A, then the dependency graph of B on P is called the dependency graph of A on P under B. Similarly, the loops of B on P are called the loops of A on P under B, and the generalized loops of B on P are called the generalized loops of A on P under B.

5. Computing the models of propositional circumscription

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

5.1. The basic case

Recall that for any formula A and any set P of atoms, by $A_{\bot}^{P}$ we denote the result of replacing all occurrences of atoms from P in A by $\bot$.

Theorem 1. *Let A be a formula, and B a finite set of clauses that is equivalent to A. The following formulas are equivalent to each other.*

(a) CIRC$[A; P]$.
(b) *The conjunction of A and*

$$\bigvee K \supset \neg A_{\bot}^{K}$$ [9]

for all subsets K of P.

[9] When $\bigvee$ is applied to a set K as in the antecedent, it stands for the disjunction of all elements of K. $\bigwedge$ is similar.

(c) *The conjunction of* A *and*

$$\bigwedge L \supset \neg A^L_\bot \tag{4}$$

for all generalized loops L *of* A *on* P *under* B.

In view of the theorem, circumscribing a theory can be regarded as just adding to it formulas (4) for all generalized loops. When L is a loop, we call formula (4) the *conjunctive loop formula* of L for CIRC$[A; P]$. Notice that (c) can also be viewed as the conjunction of the completion of A on P and all conjunctive loop formulas.

Since conditions (b) and (c) are equivalent to each other, any intermediate condition between the two is also equivalent to (a)–(c):

Corollary 1. *The following formulas are equivalent to each of formulas* (a)–(c) *of Theorem* 1.

(d) *The conjunction of* A *and*

$$\bigwedge K \supset \neg A^K_\bot$$

for all nonempty subsets K *of* P.

(e) *The conjunction of* A *and*

$$\bigvee L \supset \neg A^L_\bot \tag{5}$$

for all generalized loops L *of* A *on* P *under* B.

Note that in (d), K should be nonempty. Otherwise, the formula is unsatisfiable. When L is a loop, we call formula (5) the *disjunctive loop formula* of L for CIRC$[A; P]$. Note also that (4) and (5) are the same when L is a singleton. The two formulas can be different only when L is a loop.

Another corollary of Theorem 1 is when the theory has no loops:

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

Corollary 2 is more general than Proposition 3 in that P is not required to be a singleton. We can still compute circumscription by completion if there are no loops. For instance, CIRC$[p \vee q; p, q]$ is equivalent to the completion of $p \vee q$ on $\{p, q\}$. On the other hand, Corollary 2 does not apply even when P is a singleton if B has a loop. However, such a loop can only come from a tautological clause, which can be dropped without affecting the models. After then Proposition 3 follows from Corollary 2.

For an example theory that has loops, consider the theory A_1 and the tuple $P = (p, q, r, s)$ in the previous section. This theory has three models, $\{p, q\}$, $\{r, s\}$ and $\{p, q, r, s\}$, among which the last is not a model of CIRC$[A_1; P]$ because $\{p, q\} <^P \{p, q, r, s\}$. Theorem 1 tells us that the models of the circumscription can be found by computing the models of the completion and loop formulas. As we have seen in the previous section, there are two loops of A_1 on P, $\{p, q\}$ and $\{r, s\}$. Their loop formulas are

$p \wedge q \supset \neg r$ and $r \wedge s \supset \neg p$, respectively. Among the models of the completion of A_1 on P, $\{p, q, r, s\}$ does not satisfy loop formula $p \wedge q \supset \neg r$ (or $r \wedge s \supset \neg p$), so that it is not a model of CIRC$[A_1; P]$. The other two satisfy all loop formulas, and thus are the models of the circumscription.

The view of circumscription in terms of "completion + loop formulas" sometimes helps us observe why two circumscriptions are equivalent to each other. For instance in the example above, even if we restrict atoms to be circumscribed in to be only $\{p, q\}$, the result remains the same. 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 $\{p, q\}$. Set $\{p, q, r, s\}$ still does not satisfy 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 with all other atoms fixed,

(1) one first converts A into a finite set B of clauses,
(2) constructs the dependency graph of B on P,
(3) finds the loops of the dependency graph, and then
(4) computes loop formulas.

In the first step, a formula may be equivalent to many different sets of clauses, which in turn may yield different dependency graphs. (For instance, formulas

$$\{p \vee q \vee \neg r,\ p \vee \neg q \vee \neg r,\ \neg p \vee \neg q \vee r,\ \neg p \vee \neg q \vee \neg r\}$$

and

$$\{p \vee \neg r,\ \neg p \vee \neg q\}$$

are equivalent to each other, but their dependency graphs on $\{p, q, r\}$ are different.) For our purpose, everything else 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 an equivalent set of clauses that would yield the smallest number of loops. But we do not have a proof, and it remains an open question.

A standard way of converting a formula A (assuming all connectives other than $\neg$, $\wedge$, $\vee$ are eliminated) into an equivalent set of clauses is first to distribute $\neg$ over $\wedge$ and $\vee$ until it applies to atoms only, and then to distribute $\vee$ over $\wedge$ until it applies to literals only. However, when distributing $\vee$ over $\wedge$, the size of the formula could grow exponentially in the number of atoms in A. To avoid the problem, one can introduce new atoms. We will see in Section 5.2 that these new atoms can be treated similarly to atoms that are allowed to vary.

Once converted into an equivalent set B of clauses (without introducing new atoms), its dependency graph can be constructed in time polynomial to the number of atoms in A. However, the number of loops can be exponential in the worst case. From the complexity point of view, we cannot do much better about this as it turned out inevitable assuming a conjecture from the theory of computational complexity which is widely believed to be true. Lifschitz and Razborov [16] showed that any equivalent translation from logic programs to propositional formulas involves a significant increase in size assuming the

conjecture. A modification of their theorem holds for circumscription as well (Vladimir Lifschitz, personal communication).

One can also construct a dependency graph directly without actually generating a set of clauses. Given a formula A, $NNF(A)$ denotes the *negation normal form* of A, that is, a formula obtained from A by distributing $\neg$ over $\wedge$ and $\vee$ until it applies to atoms only. Every formula A can be written in the form $C_1 \wedge \cdots \wedge C_n$ ($n \geqslant 1$) where each C_i is not a conjunction. We call each C_i *conjunctive component* of A.

Definition 4. Let A be a formula, and P a set of atoms. The *(positive) dependency graph* of A on P is the directed graph such that

- the vertices are the atoms in P, and
- an edge goes from vertex p to vertex q if there is a subformula $F \vee G$ of some conjunctive component of $NNF(A)$ such that p occurs in F and $\neg q$ occurs in G, or the other way around.

Theorem 1 still holds if we replace "for all generalized loops L of A on P under B" with "for all generalized loops L of A on P" under this definition. This is justified by the following lemma:[10]

Lemma 1. *Let A be a formula in negation normal form. There exists an equivalent set B of clauses such that two literals l_1, l_2 belong to the same clause in B iff there is a subformula $F \vee G$ of A such that $l_1 \in F$ and $l_2 \in G$, or the other way around.*

Proof. This can be proved by structural induction. □

When we are given a set of clauses rather than an arbitrary formula to circumscribe in, Theorem 1 may yield a shorter reformulation of circumscription:

Corollary 3. *For any finite set A of clauses,* CIRC$[A; P]$ *is equivalent to the conjunction of A and*

$$\bigwedge L \supset \bigvee_{\substack{C \in A \\ C \cap L \neq \emptyset,\ C \cap \overline{L} = \emptyset}} \neg(C \backslash L)$$

for all generalized loops L of A on P.

Proof. Note first that for every clause C such that $C \cap \overline{L} = \emptyset$, $C^L_\perp$ is equivalent to $C \backslash L$. By Theorem 1, it is sufficient to show that every model of A does not satisfy $\neg C^L_\perp$ such that $C \cap \overline{L} \neq \emptyset$ or $C \cap L = \emptyset$:

- For every clause C in A such that $C \cap \overline{L} \neq \emptyset$, $C^L_\perp$ is a tautology, so that $\neg C^L_\perp$ is unsatisfiable.

[10] The lemma is due to Paolo Ferraris.

- For every clause C in A such that $C \cap L = \emptyset$, C entails $C^L_\perp$, so that every model of A, which satisfies C, does not satisfy $\neg C^L_\perp$. □

5.2. Varying constants

The following proposition [19, Proposition 2] shows how to eliminate varied constants in general:

Proposition 5. CIRC$[A(P, Z); P; Z]$ *is equivalent to*

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

Thus circumscription with varied constants reduces to the basic case to which Theorem 1 applies: we consider the dependency graph of $\exists z A(P, z)$.

Alternatively, we can generalize the definitions of a dependency graph and a loop. Intuitively, paths are allowed to have varied atoms and their negations as intermediate vertices. Given a set X of literals, $\overline{X}$ is the set of literals complementary to literals in X.

Definition 5. Let A be a set of clauses, and P, Z be sets of atoms. The (*positive*) *dependency graph* of A on P with Z varied is the directed graph such that

- the vertices are the literals in $P \cup Z \cup \overline{Z}$, and
- an edge goes from vertex l_1 to vertex l_2 if there is a clause C in A such that $l_1, \bar{l}_2 \in C$.

The definition is a generalization of Definition 2 in that it reduces to Definition 2 when Z is empty. Similarly to Definition 3, we can also extend Definition 5 to an arbitrary formula by referring to an equivalent set of clauses.

Let L_0 be the set of literals such that, for every pair l_1, l_2 of literals in L_0, there exists a path of non-zero length from l_1 to l_2 in the dependency graph of A on P with Z allowed to vary such that all vertices in this path belong to L_0. A nonempty set $L = L_0 \backslash (Z \cup \overline{Z})$ is called a *loop* of A on P with Z allowed to vary. A *generalized loop* of A on P with Z allowed to vary is defined similarly to the case when no atoms are allowed to vary, by dropping the requirement that the paths be of non-zero length.

For example, A_2 is the set of clauses

$$\{p \supset \neg z,\ \neg z \supset q,\ q \supset p\},$$

and its dependency graph on $\{p, q\}$ with $\{z\}$ allowed to vary is shown in Fig. 2. There are three generalized loops: $\{p\}$, $\{q\}$, $\{p, q\}$, and only the last one is a loop.

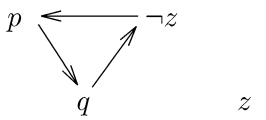

Fig. 2. The dependency graph of A_2 on $\{p, q\}$ with $\{z\}$ varied.

The definition of a loop above is intuitive, but here is another definition that is more economical in terms of the number of loops we get. Given a set A of clauses and a tuple of atoms $Z = (z_n, \ldots, z_1)$, A_k $(0 \leqslant k \leqslant n)$ is defined as follows:

- $A_0 = A$,
- $A_k = A_{k-1} \cup \{C_1 \cup C_2\colon\ C_1 \cup \{z_k\}, C_2 \cup \{\neg z_k\} \in A_{k-1}\}$.

The *Z-collapsed set of A* is obtained from A_n by removing clauses C such that C contains a pair of complementary literals, or there is a proper subset C' of C such that $C' \in A_n$.

Let A_Z be the Z-collapsed set of A, and A'_Z the set of clauses in A_Z that do not mention atoms in Z. We could then define the loops and the generalized loops of A on P with Z allowed to vary to be the loops and the generalized loops of A'_Z on P. A loop (generalized loop, respectively) according to this alternative definition is also a loop (generalized loop, respectively) according to the definition above, but not vice versa. The following result holds under either definition.

Theorem 2. *Let $A(P, Z)$ be a formula, and B a finite set of clauses equivalent to $A(P, Z)$. The following formulas are equivalent to each other.*

(a) CIRC$[A(P, Z); P; Z]$.
(b) *The conjunction of $A(P, Z)$ and*

$$\bigvee K \supset \forall z \neg A(P, z)^K_\perp$$

for all subsets K of P.
(c) *The conjunction of $A(P, Z)$ and*

$$\bigwedge L \supset \forall z \neg A(P, z)^L_\perp \tag{6}$$

for all generalized loops L of $A(P, Z)$ on P under B with Z varied.

When L is a loop, we call formula (6) the *conjunctive loop formula* of L for CIRC$[A(P, Z); P; Z]$.

Theorem 2 follows from Theorem 1, Proposition 5 and the following lemma.

Lemma 2. *Let $A(P, Z)$ be a finite set of clauses. Formula $\exists z A(P, z)$ is equivalent to the set of all clauses in the Z-collapsed set of A that do not mention atoms in Z.*

Similarly to Theorem 1, we have the following corollary to Theorem 2.

Corollary 4. *The following formulas are equivalent to each of formulas* (a)–(c) *of Theorem* 2.

(d) *The conjunction of $A(P, Z)$ and*

$$\bigwedge K \supset \forall z \neg A(P, z)^K_\perp$$

for all nonempty subsets K of P.

(e) *The conjunction of $A(P, Z)$ and*

$$\bigvee L \supset \forall z \neg A(P, z)^L_\bot$$

for all generalized loops L of A on P under B with Z varied.

When L is a loop, we call formula (6) the *disjunctive loop formula* of L for CIRC$[A(P, Z); P; Z]$. We can again divide each of (c) and (e) into two parts: completion and loop formulas. Recall that the completion of a formula $A(P, Z)$ on P with Z allowed to vary is defined to be the pointwise circumscription of P in A with Z allowed to vary, which is equivalent to the conjunction of $A(P, Z)$ and formulas

$$p \supset \forall z \neg A(P, z)^{\{p\}}_\bot$$

for all atoms p in P.

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

Corollary 5. *For any formula A, if there is an equivalent set B of clauses such that there are no loops of A on P under B with Z varied, then* CIRC$[A; P; Z]$ *is equivalent to the completion of A on P with Z varied.*

In Section 5.1 we mentioned the use of new atoms to avoid exponential blow-up in converting a formula into a conjunctive normal form. Here is one standard way to do so:

CLAUSIFY*(F,Γ)
  if F is a conjunction of clauses $C_1 \wedge \cdots \wedge C_k$, then exit with $\{C_1, \ldots, C_k\} \cup \Gamma$;
  $G :=$ a minimal non-literal subformula of F;
  $u :=$ a new atom;
  $F :=$ the result of replacing G in F by u;
  CLAUSIFY*$(F, \Gamma \cup$ CLAUSIFY$(u \equiv G))$.

(CLAUSIFY(F) returns a set of clauses equivalent to F by the method described in Section 5.1.)

Let $A(P, Z)$ be a formula and $A'(P, Z, S)$ the result of CLAUSIFY*$(A(P, Z), \emptyset)$ which introduces a set S of new atoms. Then $A(P, Z)$ is equivalent to $\exists s A'(P, Z, s)$. According to Proposition 5,

$$\begin{aligned}\text{CIRC}\big[A(P, Z); P; Z\big] &\equiv \text{CIRC}\big[\exists s A'(P, Z, s); P; Z\big] \\ &\equiv \exists s A'(P, Z, s) \wedge \text{CIRC}\big[\exists z s A'(P, z, s); P\big].\end{aligned}$$

So it is sufficient to find the loops of the dependency graph of $\exists z s A'(P, z, s)$. By Lemma 2, we can consider the dependency graph of $(Z \cup S)$-collapsed set of $A'(P, Z, S)$ that do not mention atoms in $Z \cup S$.

Similarly to Definition 4, we can also define a dependency graph and a loop of an arbitrary formula, without referring to an equivalent set of clauses.

Definition 6. Let A be a formula, and P, Z be sets of atoms. The (*positive*) *dependency graph* of A on P with Z varied is the directed graph such that

- the vertices are the literals in $P \cup Z \cup \overline{Z}$, and
- an edge goes from vertex l_1 to vertex l_2 if there is a subformula $F \vee G$ of some conjunctive component of $NNF(A)$ such that l_1 occurs in F and $\bar{l}_2$ occurs in G, or the other way around.

Once we define a dependency graph, the definition of a loop is the same as before. Theorem 2 still holds if we replace "for all generalized loops L of $A(P, Z)$ on P under B" with "for all generalized loops L of $A(P, Z)$ on P" under this definition.

Similarly to Corollary 3, when we are given a set of clauses rather than an arbitrary formula to circumscribe in, Theorem 2 may yield a shorter reformulation of circumscription:

Corollary 6. *For any finite set A of clauses,* CIRC$[A; P; Z]$ *is equivalent to the conjunction of A and*

$$\bigwedge L \supset \forall_Z \bigvee_{C \in A,\ C \cap \overline{L} = \emptyset} \neg(C \backslash L)$$

for all generalized loops L of A on P with Z varied.

Note that unlike Corollary 3 we do not restrict the disjunction to be over $C \cap L \neq \emptyset$. Indeed, if we did, the statement of the corollary would become incorrect. Consider a set A_3 of clauses

$$\{\neg p \vee q, \neg q \vee p, \neg r \vee s, \neg s \vee q\}.$$

Set $\{p, q, r, s\}$ is one of the models of CIRC$[A_3; p, q; s]$, but it does not satisfy the modified (wrong) loop formula, $p \wedge q \supset \bot$.

Proof of Corollary 6. Note first that for every clause C such that $C \cap \overline{L} = \emptyset$, $C^L_\bot$ is equivalent to $C \backslash L$. By Theorem 2, it is sufficient to show that every model of A does not satisfy $\neg C^L_\bot$ such that $C \cap \overline{L} \neq \emptyset$: for every clause C in A such that $C \cap \overline{L} \neq \emptyset$, $C^L_\bot$ is a tautology, so that $\neg C^L_\bot$ is unsatisfiable. □

5.3. Relating to some known results

Reiter [33] was the first one 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 the completion on P. In the propositional case, our new contributions are as follows. We showed that Lifschitz's pointwise circumscription extends Clark's completion to arbitrary theories, not just those that are Horn. With this extension of completion, we gave a general syntactic condition that guarantees the equivalence between circumscription and completion. Also, we defined notions of loops and loop formulas, and showed that circumscription can be reduced to "completion + loop

formulas" like answer sets in logic programming but with completion here defined to be pointwise circumscription.

Traditionally, computing circumscription means finding classes of first-order theories for which their circumscriptions are equivalent to first-order theories (e.g., [6,19,20]). 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 economically 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 a tight program are exactly the models of the completion. Corollaries 2 and 5 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 easily be explained by our corollaries.

We say that 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 is *positive* relative to P if all occurrences of P in it are positive, and *negative* if all occurrences of P are negative. We see that if A 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$[A; P]$ is equivalent to the completion of A on P. The result can also be extended when some atoms are allowed to vary. The following proposition, which is from the propositional case of Proposition 2a from [20], is easy to prove.

Proposition 6. *For any tuple of atoms $P = (p_1, \ldots, p_n)$ and any formula A that is positive relative to each p_i,* CIRC$[A; P; Z]$ *is equivalent to the completion of A on P with Z allowed to vary.*

Proof. This follows from the fact that there exists a set of clauses equivalent to A such that the dependency graph of A under it has no loops. □

Note that Corollary 5 provides a more general syntactic condition for the equivalence between circumscription and pointwise circumscription, than the one in Proposition 6. For example, consider CIRC$[(p \supset q) \wedge (q \supset r); q, r]$. The formula has no loops, and consequently the circumscription is equivalent to its corresponding pointwise circumscription by Corollary 5. 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 6 does not apply.

Sometimes it is easy to observe that the dependency graphs of some circumscriptions have the same loops. For instance, this observation provides an easy proof of the following proposition from [20].

Proposition 7. *For any formula A and B, if B is negative relative to a tuple P of atoms, then* CIRC$[A \wedge B; P]$ *is equivalent to* CIRC$[A; P] \wedge B$.

Proof. There exists a set B' of clauses which is equivalent to B and is negative relative to P. Notice that $A \wedge B'$ and A have the same dependency graph on P, hence have the

same loops. We see that loop formulas of $A \wedge B'$ on P and those of A on P are equivalent to each other when we notice that $(B')^L_\bot$ for any loop L is entailed by B'. □

However, if some atoms are allowed to vary, then $A \wedge B$ may have more loops than A ($A_2(P, z)$ in Section 5.2 with $B = \{p \supset \neg z\}$ for example), and B may not be "factored out".

6. Embedding circumscription in other nonmonotonic logics

As mentioned in the introduction, the idea of "completion + loop formulas" has been applied to logic programs [12,24] and to McCain–Turner causal logic [13]. The characterizations of these nonmonotonic logics in terms of propositional logic are useful tools for comparing these formalisms. Based on this idea, Lee [13] showed how to embed logic programs in causal logic.

Inspired by a similar characterization for circumscription, we show how to embed circumscription in logic programs and in causal logic. The propositions are proved in Appendix A by turning each formalism into equivalent propositional formulas, and then show that the translations are equivalent to each other (in propositional logic).

6.1. Embedding circumscription in disjunctive logic programs

For the semantics of disjunctive logic programs, we refer the reader to Section 5.1 of [21].

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

$$\bigvee_{p \in C \cap P} p \leftarrow \bigwedge_{q \in C \setminus P} q_{not}.$$

For example, $R_{(p,q,r)}(p \vee q \vee \neg r \vee s) = p\,;\, q \leftarrow r,\ not\ s$.

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

Proposition 8. *For any finite set A of clauses, a set of atoms is a model of* CIRC$[A; P]$ *iff it is an answer set for logic program*

$$\{R_P(C)\colon\ C \in A\} \cup \{a\,;\, not\ a\colon\ a \in \sigma_A \setminus P\}.$$

We can also embed circumscription with varied constants in logic programs. Given a finite set A of clauses, let A_Z be the Z-collapsed set of A. Program $\Pi_{A;P;Z}$ consists of the following rules:

- $R_P(C)$ for all clauses C in A_Z that do not mention atoms in Z,
- $\leftarrow \bigwedge_{p \in C} p_{not}$ for all other clauses C in A_Z,
- $a\,;\, not\ a$ for all atoms $a \in \sigma_A \setminus P$.

Proposition 9. *For any finite set A of clauses, a set of atoms is a model of* CIRC$[A; P; Z]$ *iff it is an answer set for logic program $\Pi_{A;P;Z}$.*

For example, CIRC$[A_1; p, q, r]$ can be turned into:

$$\begin{aligned} p &\leftarrow q \\ q &\leftarrow p \\ r &\leftarrow s \\ &\leftarrow r,\ \mathit{not}\ s \\ p\,;\, r &\leftarrow \\ s\,;\, \mathit{not}\ s &\leftarrow \end{aligned}$$

As in the example, the translation gives us disjunctive programs in general. Thus one could implement circumscription in logic programming systems like DLV[11] and GnT[12] using Propositions 8 and 9.[13]

6.2. Embedding circumscription in causal logic

The semantics of McCain–Turner causal logic is given in [28].

Let A be a finite set of clauses, and P a tuple of atoms. For each clause $C \in A$, the corresponding causal rule $CR_P(C)$ is

$$C \cap (P \cup \overline{P}) \Leftarrow \neg\big(C \backslash (P \cup \overline{P})\big).$$

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

Proposition 10. *For any finite set A of clauses, an interpretation is a model of* CIRC$[A; P]$ *iff it is a model of causal theory*

$$\big\{CR_P(C)\colon\ C \in A\big\} \cup \{\neg a \Leftarrow \neg a\colon\ a \in \sigma_A\} \cup \{a \Leftarrow a\colon\ a \in \sigma_A \backslash P\}.$$

whose signature is σ_A.

Proposition 10 is similar to the propositional case of Proposition 1 of [22].

Extending the result to circumscription with varied constants is similar to the case with logic programs. Given a finite set A of clauses, let A_Z be the Z-collapsed set of A. Causal theory $CT_{A;P;Z}$ consists of the following causal rules:

- $CR_P(C)$ for all clauses C in A_Z that do not mention atoms in Z,
- $\Leftarrow \neg C$ for all other clauses C in A_Z,

[11] http://www.dbai.tuwien.ac.at/proj/dlv/.

[12] http://www.tcs.hut.fi/Software/gnt/.

[13] Strictly speaking, the current versions of these 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.

- $\neg a \Leftarrow \neg a$ for all atoms $a \in \sigma_A$,
- $a \Leftarrow a$ for all atoms $a \in \sigma_A \setminus P$.

Proposition 11. *For any finite set A of clauses, an interpretation is a model of* CIRC$[A; P; Z]$ *iff it is a model of causal theory $CT_{A;P;Z}$ whose signature is σ_A.*

7. Conclusion

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

- Showed that pointwise circumscription is an extension of Clark's completion from Horn clauses to arbitrary formulas.
- Introduced a notion of a dependency graph for a finite set of clauses, and based on it, notions of loops and loop formulas.
- Showed that circumscription is equivalent to completion (pointwise circumscription) plus loop formulas, and based on this result, showed how to embed circumscription in other nonmonotonic logics which have similar characterizations.

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. According to Theorems 1 and 2, if a given formula has no loops 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".

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, Hudson Turner, and the anonymous referees who reviewed this paper or the short version of this paper presented in AAAI'04 for their useful comments. Special thanks to Vladimir Lifschitz who provided us with valuable advice and pointers to earlier work and helped us improve the presentation. Joohyung Lee was partially supported by the Texas Higher Education Coordinating Board under Grant 003658-0322-2001. Fangzhen Lin was partially supported by HK RGC under CERG HKUST6205/02E, CERG HKUST6170/04E, and by China NSFC under grant 60496322. Most of this work was done while Joohyung was at the University of Texas at Austin.

Appendix A. Proofs

The proof of the main theorem is based on the following fact and the main lemma below.

Fact 1. *For any formula A and any sets I, K of atoms,*

$$I \models A^K_\perp \quad \textit{iff} \quad I \setminus K \models A.$$

Proof. The proof is immediate by structural induction. □

Main Lemma. *Let A be a formula, B a finite set of clauses equivalent to A, I a model of A, P a set of atoms, and K a nonempty subset of P. If I does not satisfy $A^L_\perp$ for any generalized loop L of A on P under B such that $L \subseteq K$, then I does not satisfy $A^K_\perp$.*

The proof is given in Section A.2.

A.1. Proof of Theorem 1

Theorem 1. *Let A be a formula, and B a finite set of clauses that is equivalent to A. The following formulas are equivalent to each other.*

(a) CIRC$[A; P]$.
(b) *The conjunction of A and*

$$\bigvee K \supset \neg A^K_\perp$$

for all subsets K of P.
(c) *The conjunction of A and*

$$\bigwedge L \supset \neg A^L_\perp$$

for all generalized loops L of A on P under B.

Proof. *From* (b) *to* (c) is clear.

From (a) *to* (b): Let I be a model of CIRC$[A; P]$. Let K be any subset of P such that $I \cap K \neq \emptyset$. Since $I \setminus K <^P I$, it follows that $I \setminus K \not\models A$. By Fact 1, it follows that $I \not\models A^K_\perp$.

From (c) *to* (a): Let I be a model of the conjunction of A and

$$\bigwedge L \supset \neg A^L_\perp$$

for all generalized loops L of A on P under B. Let J be any set of atoms such that $J <^P I$. We will show that $J \not\models A$. Let $K = I \setminus J$. For every generalized loop L that is contained in K, since $I \models \bigwedge L$, we have that $I \not\models A^L_\perp$. Since K is nonempty and $I \not\models A^L_\perp$ for any generalized loop L that is contained in K, by the main lemma, it follows that $I \not\models A^K_\perp$, which is equivalent to $I \setminus K \not\models A$, i.e., $J \not\models A$ by Fact 1. Therefore I is a model of CIRC$[A; P]$. □

A.2. Proof of the Main Lemma

Lemma 3. *Let A be a formula, B a finite set of clauses equivalent to A, I a model of A, P a set of atoms, K a subset of P, and L a nonempty subset of K. Suppose that the dependency graph of B has no edge from an atom in L to an atom in $K \setminus L$. If $I \not\models A^L_\perp$, then $I \not\models A^K_\perp$.*

Proof. We prove it for the special case when $A = B$. The general case follows because when A and B are equivalent to each other, $A^L_\perp$ is equivalent to $B^L_\perp$ for any set L of atoms.

Assume that $I \not\models A^L_\perp$. There exists a clause C of A such that $I \not\models C^L_\perp$, or by Fact 1,

$$I \setminus L \not\models C. \tag{A.1}$$

For this C, we will show that $I \not\models C^K_\perp$, from which it follows that $I \not\models A^K_\perp$.

Since $I \models A$,

$$I \models C \tag{A.2}$$

also. From (A.1) and (A.2), it follows that C contains at least one positive occurrence of an atom from L. On the other hand, since the dependency graph of A has no edge from an atom in L to an atom in $K \setminus L$, C does not contain any negative occurrence of atoms from $K \setminus L$. Then it follows from (A.1),

$$I \setminus K \not\models C,$$

which is equivalent to $I \not\models C^K_\perp$ by Fact 1. □

Proof of the Main Lemma. In view of Lemma 3, it is sufficient to show that there exists a generalized loop L in K such that the dependency graph of B does not have any edge from an atom in L to an atom in $K \setminus L$. To see that there is indeed such a generalized loop, let G be the subgraph of the dependency graph of A which is induced by K, and let G' be the graph obtained from G by collapsing strongly connected components of G, i.e., the vertices of G' are the strongly connected components of G and G' has an edge from a vertex V to a vertex V' iff G has an edge from an atom in V to an atom in V'. Since K is nonempty, there is at least one generalized loop in K. Consequently, there is at least one vertex in G'. From the fact that the vertices of G' are the strongly connected components of G, it follows that there is a terminal vertex in G'. Let L be that vertex. It is clear that there is no edge from an atom in L to an atom in $K \setminus L$ in the dependency graph of B. □

A.3. Proof of Lemma 2

Lemma 2. *Let $A(P, Z)$ be a finite set of clauses. Formula $\exists z A(P, z)$ is equivalent to the set of all clauses in the Z-collapsed set of A that do not mention atoms in Z.*

Proof. Since the Z-collapsed set of A is equivalent to $A_{|Z|}$ (Section 5.2), it is sufficient to prove that $\exists z A(P, z)$ is equivalent to the set of all clauses in $A_{|Z|}$ that do not mention atoms in Z. The proof is by strong induction on the length of z. Assume that

$$\exists z_{k-1}, \ldots, z_1 A(P, z_k, z_{k-1}, \ldots, z_1)$$

is equivalent to the set of all clauses in $A_{k-1}(z_k)$ that do not mention atoms in $z_{k-1}, \ldots, z_1$. Let's denote the set by $A'_{k-1}(z_k)$. Then $\exists z_k, \ldots, z_1 A(P, z_k, \ldots, z_1)$ is equivalent to $A'_{k-1}(\top) \vee A'_{k-1}(\bot)$, from which a conjunctive normal form can be obtained by distributing $\vee$ over $\wedge$. Let's denote the resulting set of clauses by B_k. One can check that B_k is equivalent to the set of all clauses in A_k that do not mention atoms in $z_k, \ldots, z_1$:

- B_k contains all clauses in $A'_{k-1}(z_k)$ that do not mention z_k.
- Every other clause in B_k is entailed by a clause in $A'_{k-1}(z_k)$ that does not mention z_k except for clauses $C_1 \vee C_2$ that are obtained by taking a disjunction of $C_1 \vee \neg z_k$ from $A'_{k-1}(\top)$ and $C_2 \vee z_k$ from $A'_{k-1}(\bot)$. □

A.4. Proof of Theorem 2

Theorem 2. *Let $A(P, Z)$ be a formula, and B a finite set of clauses equivalent to $A(P, Z)$. The following formulas are equivalent to each other.*

(a) CIRC$[A(P, Z); P; Z]$.
(b) *The conjunction of $A(P, Z)$ and*

$$\bigvee K \supset \forall z \neg A(P, z)^K_\bot$$

for all subsets K of P.
(c) *The conjunction of $A(P, Z)$ and*

$$\bigwedge L \supset \forall z \neg A(P, z)^L_\bot$$

for all generalized loops L of $A(P, Z)$ on P under B with Z varied.

Proof. By Proposition 5, CIRC$[A(P, Z); P; Z]$ is equivalent to

$$A(P, Z) \wedge \text{CIRC}\big[\exists z A(P, z); P\big], \tag{A.3}$$

and by Theorem 1(b), (A.3) is equivalent to the conjunction of $A(P, Z)$ and

$$\bigvee K \supset \neg \exists z A(P, z)^K_\bot$$

for all subsets K of P. By Theorem 1(c), (A.3) is also equivalent to the conjunction of $A(P, Z)$ and

$$\bigwedge L \supset \neg \exists z A(P, z)^L_\bot \tag{A.4}$$

for all generalized loops L of $\exists z A(P, z)$ under (the clausal form of) $\exists z B(P, z)$. By Lemma 2, $\exists z B(P, z)$ is equivalent to the set of all clauses in the Z-collapsed set of B that do not mention atoms in Z, from which we get the generalized loops of A on P under B with Z varied. □

A.5. Proof of Proposition 9

Given a disjunctive logic program Π without classical negation, by $\Pi^K_\bot$ we denote the program obtained from Π by replacing all occurrences of atoms from K that are not in the scope of negation as failure with $\bot$.

When Π is finite, propositional theory $T(\Pi)$ is the conjunction of Π and

$$\bigwedge K \supset \neg \Pi_{\perp}^{K}$$

for all nonempty sets K of atoms that occur in Π.[14]

The following proposition is from Corollary 6 of [15].

Proposition 12. *For any finite disjunctive logic program Π without classical negation, a set of atoms is an answer set for Π iff it is a model of $T(\Pi)$.*

Proof of Proposition 9. According to Corollary 4, CIRC$[A; P; Z]$ is equivalent to the conjunction of A and

$$\bigwedge K \supset \neg \exists z A(P, z)_{\perp}^{K} \tag{A.5}$$

for all nonempty subsets K of P; according to Proposition 12, $T(\Pi_{A;P;Z})$ is equivalent to the conjunction of $\Pi_{A;P;Z}$ and

$$\bigwedge K \supset \neg (\Pi_{A;P;Z})_{\perp}^{K} \tag{A.6}$$

for all nonempty sets K of atoms from σ_A.

We will show that these two theories are equivalent to each other in propositional logic. First it is easy to check that $\Pi_{A;P;Z}$ is equivalent to A in propositional logic. One can also check that every formula (A.6) where K contains an atom from $\sigma_A \backslash P$ is a tautology due to the presence of rules a; *not* a for $a \in \sigma_A \backslash P$.

We will now show that under $\Pi_{A;P;Z}$ (or A) the set of formulas (A.5) for all nonempty subsets K of P is equivalent to the set of formulas (A.6) for all nonempty subsets K of P. It is sufficient to show that under the same assumption $\exists z A(P, z)_{\perp}^{K}$ is equivalent to $(\Pi_{A;P;Z})_{\perp}^{K}$ for every nonempty subset K of P. It follows from Lemma 2 that $\exists z A(P, z)_{\perp}^{K}$ is equivalent to the conjunction of $R_P(C)_{\perp}^{K}$ for all clauses C in A_Z that do not mention atoms in Z; for all rules r other than such $R_P(C)$'s in $\Pi_{A;P;Z}$, it is easy to check that $r_{\perp}^{K}$ is entailed by $\Pi_{A;P;Z}$, so that $\exists z A(P, z)_{\perp}^{K}$ is equivalent to $(\Pi_{A;P;Z})_{\perp}^{K}$ as well. □

A.6. Proof of Proposition 11

Given a formula F and a consistent set K of literals, by F_K we denote the formula obtained from formula F by replacing all occurrences of atoms a in F by

- $\perp$ if $a \in K$, and
- $\top$ if $\neg a \in K$.

Given a causal theory CT, by CT_K we denote the theory obtained from CT by replacing all rules $F \Leftarrow G$ in CT with $F_K \Leftarrow G$.

[14] We identify a logic program with a propositional theory by identifying '*not*' with '$\neg$', ',' with '$\wedge$', and ';' with '$\vee$'.

When CT is finite, propositional theory $T(CT)$ is the conjunction of CT and

$$\bigwedge K \supset \neg CT_K$$

for all nonempty sets K of literals from the signature of CT.[15]

Similarly to Proposition 12, the following proposition holds [14, Chapter 10.3, Theorem 3]:

Proposition 13. *Let CT be a finite causal theory whose signature is σ. An interpretation of σ is a model of CT iff it is a model of $T(CT)$.*

Proof of Proposition 11. According to Corollary 4, CIRC$[A; P; Z]$ is equivalent to the conjunction of A and

$$\bigwedge K \supset \neg \exists z A(P, z)^K_\perp \tag{A.7}$$

for all nonempty subsets K of P; according to Proposition 13, $T(CT_{A;P;Z})$ is equivalent to the conjunction of $CT_{A;P;Z}$ and

$$\bigwedge K \supset \neg (CT_{A;P;Z})_K \tag{A.8}$$

for all nonempty sets K of literals from σ_A.

We will show that these two theories are equivalent to each other in propositional logic. First $CT_{A;P;Z}$ is equivalent to A in propositional logic. One can also check that every formula (A.8) where K contains a negative literal l or a literal l from $\sigma_A \setminus P$ is a tautology due to the presence of rules $l \Leftarrow l$.

We will now show that under $CT_{A;P;Z}$ (or A) the set of formulas (A.7) for all nonempty subsets K of P is equivalent to the set of formulas (A.8) for all nonempty subsets K of P. It is sufficient to show that under the same assumption $\exists z A(P, z)^K_\perp$ is equivalent to $(CT_{A;P;Z})_K$ for every nonempty subset K of P. It follows from Lemma 2 that $\exists z A(P, z)^K_\perp$ is equivalent to the conjunction of $CR_P(C)_K$ for all clauses C in A_Z that do not mention atoms in Z; for all causal rules r other than such $CR_P(C)$'s in $CT_{A;P;Z}$, it is easy to check that r_K is entailed by $CT_{A;P;Z}$, so that $\exists z A(P, z)^K_\perp$ is equivalent to $(CT_{A;P;Z})_K$ as well. □

References

[1] V. Akman, S. Erdoğan, J. Lee, V. Lifschitz, H. Turner, Representing the Zoo World and the Traffic World in the language of the Causal Calculator, Artificial Intelligence 153 (1–2) (2004) 105–140.

[2] A. Artikis, M. Sergot, J. Pitt, An executable specification of an argumentation protocol, in: Proceedings of Conference on Artificial Intelligence and Law (ICAIL), ACM Press, New York, 2003, pp. 1–11.

[3] A. Artikis, M. Sergot, J. Pitt, Specifying electronic societies with the Causal Calculator, in: F. Giunchiglia, J. Odell, G. Weiss (Eds.), Proceedings of Workshop on Agent-Oriented Software Engineering III (AOSE), in: Lecture Notes in Comput. Sci., vol. 2585, Springer, Berlin, 2003.

[15] We identify a causal theory with a propositional theory by identifying '$\Leftarrow$' with material implication.

[4] J. Campbell, V. Lifschitz, Reinforcing a claim in commonsense reasoning, in: Working Notes of the AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning, 2003, http://www.cs.utexas.edu/users/vl/papers/sams.ps.
[5] K. Clark, Negation as failure, in: H. Gallaire, J. Minker (Eds.), Logic and Data Bases, Plenum Press, New York, 1978, pp. 293–322.
[6] P. Doherty, W. Łukaszewicz, A. Szałas, Computing circumscription revisited: A reduction algorithm, J. Automat. Reasoning 18 (3) (1997) 297–336.
[7] E. Erdem, V. Lifschitz, Tight logic programs, Theory and Practice of Logic Programming 3 (2003) 499–518.
[8] F. Fages, Consistency of Clark's completion and existence of stable of stable models, J. Methods of Logic in Computer Science 1 (1994) 51–60.
[9] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. Fifth International Conference and Symposium on Logic Programming, 1988, pp. 1070–1080.
[10] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, H. Turner, Nonmonotonic causal theories, Artificial Intelligence 153 (1–2) (2004) 49–104.
[11] E. Giunchiglia, Y. Lierler, M. Maratea, SAT-based answer set programming, in: Proc. AAAI-04, San Jose, CA, 2004, pp. 61–66.
[12] J. Lee, V. Lifschitz, Loop formulas for disjunctive logic programs, in: Proc. ICLP-03, 2003, pp. 451–465.
[13] J. Lee, Nondefinite vs. definite causal theories, in: Proc. 7th Internat. Conference on Logic Programming and Nonmonotonic Reasoning, 2004, pp. 141–153.
[14] J. Lee, Automated reasoning about actions, PhD thesis, University of Texas at Austin, 2005.
[15] J. Lee, A model-theoretic counterpart of loop formulas, in: Proc. IJCAI-05, Edinburgh, UK, 2005, pp. 503–508.
[16] V. Lifschitz, A. Razborov, Why are there so many loop formulas? ACM Trans. Comput. Logic (2005), submitted for publication.
[17] V. Lifschitz, L.R. Tang, H. Turner, Nested expressions in logic programs, Ann. Math. Artificial Intelligence 25 (1999) 369–389.
[18] V. Lifschitz, N. McCain, E. Remolina, A. Tacchella, Getting to the airport: The oldest planning problem in AI, in: J. Minker (Ed.), Logic-Based Artificial Intelligence, Kluwer, Dordrecht, 2000, pp. 147–165.
[19] V. Lifschitz, Computing circumscription, in: Proc. IJCAI-85, Los Angeles, CA, 1985, pp. 121–127.
[20] V. Lifschitz, Pointwise circumscription, in: M. Ginsberg (Ed.), Readings in Nonmonotonic Reasoning, Morgan Kaufmann, San Mateo, CA, 1987, pp. 179–193.
[21] V. Lifschitz, Foundations of logic programming, in: G. Brewka (Ed.), Principles of Knowledge Representation, CSLI Publications, 1996, pp. 69–128.
[22] V. Lifschitz, On the logic of causal explanation, Artificial Intelligence 96 (1997) 451–465.
[23] V. Lifschitz, Missionaries and cannibals in the Causal Calculator, in: Principles of Knowledge Representation and Reasoning: Proc. Seventh Internat. Conf., 2000, pp. 85–96.
[24] F. Lin, Y. Zhao, ASSAT: Computing answer sets of a logic program by SAT solvers. in: Proc. AAAI-02, Edmonton, AB, 2002, pp. 112–117.
[25] F. Lin, Embracing causality in specifying the indirect effects of actions, in: Proc. IJCAI-95, Montreal, Quebec, IJCAI Inc., 1995, pp. 1985–1993, Distributed by Morgan Kaufmann, San Mateo, CA.
[26] F. Lin, On strongest necessary and weakest sufficient conditions, Artificial Intelligence 128 (1–2) (2001) 143–159.
[27] F. Lin, Compiling causal theories to successor state axioms and STRIPS-like systems, J. Artificial Intelligence Res. 19 (2003) 279–314.
[28] N. McCain, H. Turner, Causal theories of action and change, in: Proc. AAAI-97, Providence, RI, 1997, pp. 460–465.
[29] J. McCarthy, Circumscription—a form of non-monotonic reasoning, Artificial Intelligence 13 (1980) 27–39, 171–172. Reproduced in [31].
[30] J. McCarthy, Applications of circumscription to formalizing common sense knowledge, Artificial Intelligence 26 (3) (1986) 89–116. Reproduced in [31].
[31] J. McCarthy, Formalizing Common Sense: Papers by John McCarthy, Ablex, Norwood, NJ, 1990.
[32] R. Reiter, On closed world data bases, in: H. Gallaire, J. Minker (Eds.), Logics and Data Bases, Plenum Press, New York, 1978, pp. 55–76.

[33] R. Reiter, Circumscription implies predicate completion (sometimes), in: Proc. AAAI-82, Pittsburg, PA, 1982, pp. 418–420.
[34] R. Reiter, The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression, in: V. Lifschitz (Ed.), Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, Academic Press, San Diego, CA, 1991, pp. 418–420.