Relating Circumscription and Event Calculus to First-Order Stable Model Semantics

Joohyung Lee and Ravi Palla
School of Computing, Informatics and Decision Systems Engineering
Arizona State University
Tempe, AZ 85287, USA
{joolee, Ravi.Palla}@asu.edu

Abstract

The first-order stable model semantics proposed by Ferraris, Lee and Lifschitz is closely related to circumscription by McCarthy, but neither semantics is stronger than the other. We present the class of formulas called *canonical* formulas, for which the two semantics coincide, and the larger class of *pseudo-canonical* formulas, which can be turned into canonical formulas by equivalent transformations in classical logic. We also present the class of *almost universal* sentences, which can be turned into a logic program while preserving the stable models. The system F2LP is an implementation of this reduction, which allows us to compute Herbrand stable models of first-order sentences using answer set solvers.

We use these results to reformulate circumscriptive event calculus in terms of the stable model semantics, and to apply answer set solvers for event calculus reasoning. Compared with the DEC reasoner, the answer set programming based event calculus reasoner handles more expressive reasoning and is more efficient on several benchmark problems.

1 Introduction

The stable model semantics [Gelfond and Lifschitz, 1988] is the mathematical basis of answer set programming (ASP) [Marek and Truszczyński, 1999; Niemelä, 1999; Lifschitz, 2008b]. The semantics was generalized by Ferraris, Lee and Lifschitz to arbitrary first-order sentences, under which logic programs are viewed as a special class of first-order sentences [Ferraris *et al.*, 2007; Ferraris *et al.*, 2010]. Unlike the other extensions of the stable model semantics [Lifschitz, 2008a], this definition refers neither to grounding nor to fixpoints. Instead, the stable models of a first-order sentence F are defined as the models of a second-order sentence that is obtained by applying the "stable model operator" SM to F. The definition of SM is close to the definition of the circumscription operator CIRC [McCarthy, 1980;

McCarthy, 1986]. However, neither the stable model semantics nor circumscription is stronger than the other. For example,

$$\mathrm{CIRC}[\forall x(p(x) \lor \neg p(x));\ p] \tag{1}$$

is equivalent to $\forall x \neg p(x)$, and

$$\mathrm{SM}[\forall x(p(x) \lor \neg p(x));\ p] \tag{2}$$

is equivalent to $\top$, so that (1) is stronger than (2). On the other hand,

$$\mathrm{CIRC}[\forall x(\neg p(x) \rightarrow q(x));\ p, q] \tag{3}$$

is equivalent to $\forall x(\neg p(x) \leftrightarrow q(x))$, and

$$\mathrm{SM}[\forall x(\neg p(x) \rightarrow q(x));\ p, q] \tag{4}$$

is equivalent to $\forall x(\neg p(x) \land q(x))$, so that (3) is weaker than (4).

In general, identifying a syntactic class of theories for which different semantics coincide is important in understanding the relationship between them. For instance, it is well known that, for *tight* logic programs [Erdem and Lifschitz, 2003], the stable model semantics coincides with the completion semantics [Clark, 1978]. The definition of a tight program was extended to the syntax of formulas in [Ferraris *et al.*, 2010]. In this paper, we identify the class of formulas called *canonical formulas*, for which the first-order stable model semantics coincides with circumscription, and the larger class of formulas called *pseudo-canonical formulas*, which can be turned into canonical formulas by equivalent transformations in classical logic.

Another main result in this paper is about turning a first-order theory under the stable model semantics into an answer set program, in order to compute the Herbrand stable models of a first-order theory using existing answer set solvers. One of the difficulties involved in the translation is that, while the first-order stable model semantics allows explicit quantifiers, answer set programs do not have them, and there appears no consensus about the general method of simulating quantifiers in answer set programs.[1] Unlike in resolution theorem proving, we show that Skolemization does not preserve the intended meaning under the stable model semantics, even when we do not impose the Herbrand assumption. On the other hand, it has been a well-established idiom in answer set programming to simulate negated existential quantifiers in the body of a rule by using auxiliary predicates. For instance, answer set program

$$\begin{array}{l} p' \leftarrow p(x), q(x) \\ s \leftarrow r, \neg p' \end{array} \tag{5}$$

[1]See the related discussion posted in Texas Action Group, "Do We Need Existential Quantifiers in Logic Programming?" `http://www.cs.utexas.edu/users/vl/tag/existential`.

can be used to represent

$$r \wedge \neg \exists x (p(x) \wedge q(x)) \rightarrow s \tag{6}$$

under the first-order stable model semantics. Extending this idea, we present the class of *almost universal* sentences, in which quantifiers can be eliminated using auxiliary predicates. The resulting quantifier-free formulas can be further turned into ASP rules by applying a straightforward extension of the translation from [Cabalar *et al.*, 2005] that turns arbitrary propositional formulas into ASP rules.

We developed system F2LP that implements this translation method, which allows the existing answer set solvers to be used for computing answer sets of a first-order sentence. Together with the result on the relationship between the stable model semantics and circumscription mentioned above, this provides a way to compute Herbrand models of circumscriptive theories using answer set solvers. Since circumscription has found many applications in commonsense reasoning, model-based diagnoses, discourse understanding, and others, our system may be useful in a wide range of applications. We note that the conditions on a canonical formula and an almost universal formula are satisfied for a wide range of circumscriptive theories and first-order theories under the stable model semantics. For instance, it is easy to see that the well-known action formalisms such as the circumscriptive event calculus [Shanahan, 1995], causal situation calculus [Lin, 1995], and temporal action logics [Doherty *et al.*, 1998] satisfy the conditions.

In this paper, we apply the results to provide a few reformulations of the circumscriptive event calculus in terms of the stable model semantics, and show how F2LP with answer set solvers can be used for reasoning about circumscriptive event calculus. Interestingly, the development of the event calculus has spanned over both classical logic and logic programming traditions. The original version of the event calculus [Kowalski and Sergot, 1986] was formulated as logic programs but not under the stable model semantics (that was the time before the invention of the stable model semantics). More extensive later developments of the event calculus have been carried out on the classical logic foundation via circumscription (e.g., [Shanahan, 1995; Shanahan, 1997; Shanahan, 1999; Miller and Shanahan, 1999]). Based on the reduction of circumscription to completion, Shanahan and Witkowski [2004] and Mueller [2004a] implemented SAT-based event calculus systems. The DEC reasoner[2] implemented by Mueller outperforms the system implemented by Shanahan and Witkowski due to a more efficient and general encoding method. In contrast to these systems, we show that the ASP-based event calculus system can compute the *full* version of the event calculus assuming that the domain is given and finite. Thanks to efficient ASP solvers, our experiments indicate that our system outperforms the DEC reasoner.

[2]`http://decreasoner.sourceforge.net/`.

The paper is organized as follows. In Section 2, we review circumscription and the first-order stable model semantics. In Section 3, we show the classes of formulas such that the two semantics coincide, and, based on this, a reduction of circumscription to the first-order stable model semantics, and, in Section 4, a further reduction to answer set programs. In Section 5, these results are applied to turn circumscriptive event calculus into the first-order stable model semantics and into answer set programs, which provides a method of computing event calculus using ASP solvers. In Section 6, we compare the performance of our ASP-based event calculus reasoner with the DEC reasoner. The complete proofs are given in Appendix.

This article is an extended version of the conference papers [Kim *et al.*, 2009] and [Lee and Palla, 2009].[3]

2 Review: Circumscription and Stable Model Semantics

We assume the following set of primitive propositional connectives and quantifiers:

$$\bot \text{ (falsity)}, \ \wedge, \ \vee, \ \rightarrow, \ \forall, \ \exists\ .$$

$\neg F$ is an abbreviation for $F \rightarrow \bot$, symbol $\top$ stands for $\bot \rightarrow \bot$, and $F \leftrightarrow G$ stands for $(F \rightarrow G) \wedge (G \rightarrow F)$.

2.1 Circumscription

Let $\mathbf{p}$ be a list of distinct predicate constants $p_1, \dots, p_n$, and let $\mathbf{u}$ be a list of distinct predicate variables $u_1, \dots, u_n$ of the same length as $\mathbf{p}$. By $\mathbf{u} \leq \mathbf{p}$ we denote the conjunction of the formulas $\forall \mathbf{x}(u_i(\mathbf{x}) \rightarrow p_i(\mathbf{x}))$ for all $i = 1, \dots n$ where $\mathbf{x}$ is a list of distinct object variables whose length is the same as the arity of p_i. Expression $\mathbf{u} < \mathbf{p}$ stands for $(\mathbf{u} \leq \mathbf{p}) \wedge \neg(\mathbf{p} \leq \mathbf{u})$. For instance, if p and q are unary predicate constants then $(u, v) < (p, q)$ is

$$\forall x(u(x) \rightarrow p(x)) \wedge \forall x(v(x) \rightarrow q(x)) \wedge \neg(\forall x(p(x) \rightarrow u(x)) \wedge \forall x(q(x) \rightarrow v(x))).$$

The models of circumscription are defined in terms of the CIRC operator with *minimized* predicates. For any first-order formula F, expression CIRC$[F; \mathbf{p}]$ stands for the second-order formula

$$F \wedge \neg \exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F(\mathbf{u})),$$

[3]This article generalizes the definition of a canonical formula in [Kim *et al.*, 2009] and the related theorems, and show how to use system F2LP for reasoning about circumscriptive theories, instead of system ECASP reported in [Kim *et al.*, 2009] that was tailored to the event calculus. [Lee and Palla, 2009] is a system description paper about F2LP, and we present the underlying theories in this article.

where $F(\mathbf{u})$ is the formula obtained from F by substituting u_i for p_i. When F is a sentence (i.e., a formula with no free variables), intuitively, the models of CIRC$[F; \mathbf{p}]$ are the models of F that are "minimal" on $\mathbf{p}$.

The definition can be easily extended to the case when F is many-sorted [Lifschitz, 1994, Section 2.4].

2.2 First-Order Stable Model Semantics

This review follows [Ferraris *et al.*, 2010], a journal version of [Ferraris *et al.*, 2007], which distinguishes between intensional and non-intensional predicates.

The stable models are defined in terms of the SM operator with *intensional* predicates, similar to the circumscription operator above: For any first-order formula F and any finite list $\mathbf{p}$ of intensional predicates, SM$[F; \mathbf{p}]$ is defined as [4]

$$F \wedge \neg\exists\mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})),$$

where $\mathbf{u}$ is defined the same as in CIRC$[F; \mathbf{p}]$ and $F^*(\mathbf{u})$ is defined recursively:

- $p_i(\mathbf{t})^* = u_i(\mathbf{t})$ for any list $\mathbf{t}$ of terms;
- $F^* = F$ for any atomic formula F (including $\bot$ and equality) that does not contain members of $\mathbf{p}$;
- $(F \wedge G)^* = F^* \wedge G^*$;
- $(F \vee G)^* = F^* \vee G^*$;
- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;
- $(\forall x F)^* = \forall x F^*$;
- $(\exists x F)^* = \exists x F^*$.

When F is a sentence, the models of SM$[F; \mathbf{p}]$ are called the $\mathbf{p}$-*stable* models of F. Intuitively they are the models of F that are "stable" on $\mathbf{p}$. In this paper we will often write SM$[F]$ instead of SM$[F; \mathbf{p}]$ when $\mathbf{p}$ is the list of all predicate constants occurring in F. In [Lee *et al.*, 2008] *answer sets* are defined as a special class of stable models as follows. By $\sigma(F)$ we denote the signature consisting of the object, function and predicate constants occurring in F. If F contains at least one object constant, an Herbrand interpretation of $\sigma(F)$ that satisfies SM$[F]$ is called an *answer set* of F. The answer sets of a logic program Π are defined as the answer sets of the FOL-representation of Π (i.e., the conjunction of the

[4]Here we use expression SM$[F; \mathbf{p}]$ in place of SM$_{\mathbf{p}}[F]$ used in [Ferraris *et al.*, 2010].

universal closures of implications corresponding to the rules). For example, the FOL-representation of the program

$$\begin{array}{l} p(a) \\ q(b) \\ r(x) \leftarrow p(x), \mathit{not}\ q(x) \end{array}$$

is

$$p(a) \wedge q(b) \wedge \forall x(p(x) \wedge \neg q(x) \rightarrow r(x)). \tag{7}$$

It is shown in [Ferraris *et al.*, 2007; Ferraris *et al.*, 2010] that this definition of an answer set, when applied to the syntax of logic programs, is equivalent to the traditional definition of an answer set that is based on grounding and fixpoints [Gelfond and Lifschitz, 1988].

Note that the definition of a stable model is more general than the definition of an answer set in the following ways: stable models are not restricted to Herbrand models, the underlying signature can be arbitrary, and the intensional predicates are not set to the list of predicate constants occurring in the formula. The last fact is not essential in view of the following proposition. By $pr(F)$ we denote the list of all predicate constants occurring in F; by $\mathit{Choice}(\mathbf{p})$ we denote the conjunction of "choice formulas" $\forall \mathbf{x}(p(\mathbf{x}) \vee \neg p(\mathbf{x}))$ for all predicate constants p in $\mathbf{p}$ where $\mathbf{x}$ is a list of distinct object variables whose length is the same as the arity of p; by $\mathit{False}(\mathbf{p})$ we denote the conjunction of $\forall \mathbf{x} \neg p(\mathbf{x})$ for all predicate constants p in $\mathbf{p}$. We sometimes identify a list with the corresponding set when there is no confusion.

Proposition 1 *Formula*

$$\mathrm{SM}[F; \mathbf{p}] \leftrightarrow \mathrm{SM}[F \wedge \mathit{Choice}(pr(F) \setminus \mathbf{p}) \wedge \mathit{False}(\mathbf{p} \setminus pr(F))]$$

is logically valid.

2.3 Extension to Many-Sorted First-Order Formulas and Many-Sorted Logic Programs

The definition of a stable model reviewed in the previous section can be easily extended to many-sorted first-order sentences, similar to the extension of circumscription to many-sorted first-order sentences. In order to facilitate describing the translation of the event calculus, which is based on many-sorted first-order logic, into the first-order stable model semantics, here we treat logic programs as a special class of many-sorted sentences under the stable model semantics. We assume that the underlying signature contains the integer sort and contains several built-in symbols, such as integer constants, built-in arithmetic functions, such as

$+$, $-$, and comparison operators, such as $<$, $\leq$, $>$, $\geq$. A *rule* is an expression of the form

$$\begin{aligned} A_1 \,;\ldots;\, A_k \leftarrow\ & A_{k+1},\ldots,A_m, not\ A_{m+1},\ldots, not\ A_n,\\ & not\ not\ A_{n+1},\ldots, not\ not\ A_p \end{aligned}$$

($0 \leq k \leq m \leq n \leq p$), where each A_i is an atomic formula (including $\bot$ and equality). A *choice rule* of the form $\{A\} \leftarrow Body$ stands for

$$A \leftarrow Body, not\ not\ A.$$

A *(logic) program* is a finite set of rules.

The semantics of a logic program is given by identifying the logic program with its many-sorted FOL-representation. The built-in symbols, such as integer constants, $+$, $-$, $\leq$, $\geq$, are evaluated in the standard way, and we consider only those standard interpretations throughout this paper.[5] The *stable models* of a program Π is defined as the models of $\mathrm{SM}[\Pi^{FOL}]$ where Π^{FOL} is the FOL-representation of Π.

Finite sorts can be represented in the input language of LPARSE[6] and GRINGO[7] using domain predicates.

3 Turning Circumscription into the Stable Model Semantics

We say that an occurrence of a predicate constant, or any other subexpression, in a formula F is *positive* if the number of implications containing that occurrence in the antecedent is even, and *negative* otherwise. (Recall that we treat $\neg G$ as shorthand for $G \rightarrow \bot$.) We say that the occurrence is *strictly positive* if the number of implications in F containing that occurrence in the antecedent is 0. For example, in (7), both occurrences of q are positive, but only the first one is strictly positive.

We say that a formula F is *canonical* relative to a list $\mathbf{p}$ of predicate constants if

- no occurrence of a predicate constant from $\mathbf{p}$ is in the antecedent of more than one implication in F, and
- if a predicate constant p from $\mathbf{p}$ occurs in the scope of a strictly positive occurrence of $\exists$ or $\vee$ in F, then the occurrence of p is strictly positive in F.

[5]This implies that built-in predicates such as $\leq$, $\geq$ are non-intensional.

[6]`http://www.tcs.hut.fi/Software/smodels` .

[7]`http://potassco.sourceforge.net` .

Example 1 *The formula*

$$\forall x(\neg p(x) \to q(x)) \tag{8}$$

that is shown in the introduction is not canonical relative to $\{p, q\}$ *since it does not satisfy the first clause of the definition. On the other hand, the formula is canonical relative to* $\{q\}$*. The formula*

$$\forall x(p(x) \lor \neg p(x)) \tag{9}$$

is not canonical relative to $\{p\}$ *since it does not satisfy the second clause; the formula*

$$p(a) \land (\exists x\, p(x) \to \exists x\, q(x)) \tag{10}$$

is canonical relative to $\{p, q\}$*, while*

$$p(a, a) \land \exists x(p(x, a) \to p(b, x)) \tag{11}$$

is not canonical relative to $\{p, q\}$ *since it does not satisfy the second clause.*

The following theorem shows that, for any canonical formula, circumscription coincides with the stable model semantics.

Theorem 1 *For any canonical formula* F *relative to* $\mathbf{p}$*,*

$$\text{CIRC}[F; \mathbf{p}] \leftrightarrow \text{SM}[F; \mathbf{p}] \tag{12}$$

is logically valid.

For instance, for formula (10), which is canonical relative to $\{p, q\}$, formulas CIRC[(10); p, q] and SM[(10); p, q] are equivalent to each other. Also, any sentence F is clearly canonical relative to $\emptyset$, so that CIRC[F; $\emptyset$] is equivalent to SM[F; $\emptyset$], which in turn is equivalent to F.

On the other hand, formula (12) may not necessarily be logically valid when F is not canonical relative to $\mathbf{p}$. For instance, in the introduction, we observed that for formula $\forall x(p(x) \lor \neg p(x))$ that is not canonical relative to $\{p\}$, formulas (1) and (2) are not equivalent to each other; for formula $\forall x(\neg p(x) \to q(x))$ that is not canonical relative to $\{p, q\}$, formulas (3) and (4) are not equivalent to each other. We also observe that CIRC[(11); p, q] is not equivalent to SM[(11); p, q]: the Herbrand interpretation $\{p(a, a), p(b, a)\}$ satisfies SM[(11); p, q], but does not satisfy CIRC[(11); p, q].

Note that some non-canonical formulas can be equivalently rewritten as canonical formulas. Since any equivalent transformation preserves the models of circumscription, Theorem 1 can be applied to such non-canonical formulas, by first rewriting them as canonical formulas. For example, formula (8) is equivalent to

$$\forall x(p(x) \lor q(x)), \tag{13}$$

which is canonical relative to $\{p, q\}$, so that CIRC[(8); p, q] is equivalent to SM[(13); p, q]. For another example, formula (9) is equivalent to

$$\forall x(p(x) \rightarrow p(x)), \tag{14}$$

which is canonical relative to $\{p\}$, so that CIRC[(9); p] is equivalent to SM[(14); p]. Below we define the class of "pseudo-canonical" formulas that can be equivalently rewritten as canonical formulas.

First we distinguish between the occurrences of a subformula QxG in a formula F ($Q \in \{\forall, \exists\}$). We say that an occurrence of QxG in F is *singular* if

- Q is $\exists$ and the occurrence of QxG is positive in F, or
- Q is $\forall$ and the occurrence of QxG is negative in F.

For example, the occurrence of $\exists x\, q(x)$ is singular in (10), but the occurrence of $\exists x\, p(x)$ is not.

We say that a formula F is *pseudo-canonical* relative to $\mathbf{p}$ if, for all singular occurrences of QxG in F, either

- all occurrences of members of $\mathbf{p}$ in G are positive in G, or
- all occurrences of members of $\mathbf{p}$ in G are negative in G.

For example, (8) is pseudo-canonical relative to $\{p, q\}$ and (9) is pseudo-canonical relative to $\{p\}$ because the formulas contain no singular occurrences. Formula (10) is pseudo-canonical relative to $\{p, q\}$, but (11) is not. Formula $\exists x \neg p(x)$ is pseudo-canonical relative to $\{p\}$.

We can check that every canonical formula is pseudo-canonical. Moreover, the following result holds.

Proposition 2 *Every pseudo-canonical formula relative to* $\mathbf{p}$ *is equivalent to a canonical formula relative to* $\mathbf{p}$.

Thus, in view of Theorem 1, circumscription of any pseudo-canonical formula can be characterized by the stable model semantics.

The following procedure gives one method for rewriting any pseudo-canonical formula into an equivalent canonical formula. We call a maximal singular occurrence of QxG in F *pseudo-atomic*.

Algorithm 1 (PSEUDO-CANONICAL TO CANONICAL) *(1) Given a pseudo-canonical formula, for every pseudo-atomic occurrence of a subformula of the form* $\exists xG$ *where every occurrence of every predicate constant from* $\mathbf{p}$ *is negative in* G*, rewrite the subformula as* $\neg\forall x\neg G$*. Similarly, for every pseudo-atomic* *occurrence of a subformula of the form $\forall x G$ where every occurrence of every predicate constant from* $\mathbf{p}$ *is negative in G, rewrite the subformula as $\neg\exists x \neg G$. (This results in a formula such that, for every pseudo-atomic occurrence of QxG in it, every occurrence of a predicate constant from* $\mathbf{p}$ *in G is positive in G.)*

(2) Apply the standard prenex conversion procedure to

- *every positive occurrence of $\forall$, and*
- *every negative occurrence of $\exists$*

that is not in the scope of any pseudo-atomic occurrence of a formula, so that the resulting formula is of the form $\forall \mathbf{x} M$ ($\mathbf{x}$ is a list of variables) where every occurrence of a quantifier in M belongs to a pseudo-atomic occurrence of some QxG.

(3) By treating every pseudo-atomic occurrence of a subformula like an atom, turn M into a conjunction of implications $H \rightarrow K$ where H is a conjunction of atoms and formulas of the form QxG, and K is a disjunction of atoms and formulas of the form QxG.

(4) Turn each pseudo-atomic occurrence of QxG in the resulting formula into a negation normal form.

The proof of Proposition 2 follows from the following lemma.

Lemma 1 *Algorithm 1 turns every pseudo-canonical formula relative to* $\mathbf{p}$ *into an equivalent canonical formula relative to* $\mathbf{p}$.

Example 2 *Consider the following formula that is pseudo-canonical relative to $\{p, q, r\}$:*

$$\exists x\, r(x) \wedge \forall x \neg p(x) \rightarrow \exists x\, q(x) \vee \exists x (r(x) \rightarrow s(x)).$$

In this formula, except the occurrence of $\exists x\, r(x)$, all other occurrences of formulas of the form QxG are pseudo-atomic.

Step (1) turns the above formula into

$$\exists x\, r(x) \wedge \neg\exists x \neg\neg p(x) \rightarrow \exists x\, q(x) \vee \neg\forall x \neg(r(x) \rightarrow s(x)),$$

which is turned into the following formula in Step (2):

$$\forall x (r(x) \wedge \neg\exists x \neg\neg p(x) \rightarrow \exists x\, q(x) \vee \neg\forall x \neg(r(x) \rightarrow s(x))).$$

After Step (3), this formula is turned into

$$\forall x (r(x) \wedge \forall x \neg(r(x) \rightarrow s(x)) \rightarrow \exists x\, q(x) \vee \exists x \neg\neg p(x)),$$

which is turned into the following formula that is canonical relative to $\{p, q, r\}$ after Step (4):

$$\forall x(r(x) \land \forall x(r(x) \land \neg s(x)) \rightarrow \exists x\, q(x) \lor \exists x\, p(x)).$$

Although pseudo-canonical formulas are syntactically more general than canonical formulas, canonical formulas themselves are general enough to cover most circumscriptive theories, such as the event calculus, causal situation calculus [Lin, 1995], and temporal action logics [Doherty *et al.*, 1998]. In fact, in Section 5 we use Theorem 1 to reformulate the circumscriptive event calculus in terms of the stable model semantics. Lee and Palla [2010] apply the same idea to turn causal situation calculus into answer set programs.

4 Computing Answer Sets of First-Order Formulas Using Answer Set Solvers

While the stable model semantics has been extended to arbitrary formulas, the input languages of current answer set solvers are restricted to logic program syntax. In this section, we show, under a certain syntactic condition, how to turn arbitrary first-order formulas into logic programs so that existing answer set solvers can be used for computing answer sets of first-order formulas. Our translation turns a first-order sentence into a universal sentence and then into a set of ASP rules.

Note that the notion of equivalence is "stronger" under the stable model semantics. Classically equivalent transformations do not necessarily preserve stable models. For instance, while $\neg\exists x \neg p(x)$ is classically equivalent to $\forall x\, p(x)$, their stable models are different. In this section, we show that the standard prenex form conversion method in first-order logic is "strong" enough, but Skolemization is not. In other words, the former preserves the stable models, but the latter does not.

4.1 Quantifier Elimination

Our translation is built upon recent theoretical results. One of them shows that every universal sentence (including every propositional formula) is strongly equivalent to a logic program [Cabalar and Ferraris, 2007; Cabalar *et al.*, 2005; Lee and Palla, 2007].[8] Furthermore, Proposition 5 from [Lee and Palla, 2007]

[8]According to [Ferraris *et al.*, 2010], we say that formula F is *strongly equivalent* to formula G if, for any formula H containing F as a subformula (and possibly containing object, function and predicate constants that do not occur in F, G), and for any list $\mathbf{p}$ of distinct predicate constants, $\text{SM}[H; \mathbf{p}]$ is equivalent to $\text{SM}[H'; \mathbf{p}]$, where H' is obtained from H by replacing an occurrence of F by G. Strong equivalence is important since it allows us to replace a subformula with another subformula without affecting stable models.

shows that the standard prenex form conversion rules in first-order logic preserve strong equivalence as well, so that the following holds.

Theorem 2 *[Lee and Palla, 2007, Proposition 5] Every first-order formula is strongly equivalent to a prenex formula.*

The standard prenex form conversion turns a non-singular occurrence of a quantifier into an outermost $\forall$ while preserving strong equivalence. Consequently, if a sentence contains no singular occurrence of a quantifier, then the above results can be used to turn the formula into a universal sentence and then into a set of ASP rules. However, the method does not apply when the formula contains a singular occurrence of a quantifier since the standard prenex form conversion turns it into an outermost $\exists$, which is not allowed in answer set programs. Below we consider how to deal with singular occurrences of quantifiers.

If the Herbrand universe is finite, and if we are interested in answer sets only,[9] quantifiers can be replaced with multiple disjunctions and conjunctions. We do not even need to consider turning the formula into a prenex form. For example, for formula (6) occurring in a theory whose signature contains $\{1,\dots,n\}$ as the only object constants (and no other function constants), if we replace $\exists x(p(x)\land q(x))$ with multiple disjunctions and then turn the result into a logic program as in [Lifschitz *et al.*, 1999], 2^n rules are generated. For instance, if $n=3$, the resulting logic program is

$$\begin{array}{l}
s \leftarrow r, \mathit{not}\ p(1), \mathit{not}\ p(2), \mathit{not}\ p(3)\\
s \leftarrow r, \mathit{not}\ p(1), \mathit{not}\ p(2), \mathit{not}\ q(3)\\
s \leftarrow r, \mathit{not}\ p(1), \mathit{not}\ q(2), \mathit{not}\ p(3)\\
s \leftarrow r, \mathit{not}\ p(1), \mathit{not}\ q(2), \mathit{not}\ q(3)\\
s \leftarrow r, \mathit{not}\ q(1), \mathit{not}\ p(2), \mathit{not}\ p(3)\\
s \leftarrow r, \mathit{not}\ q(1), \mathit{not}\ p(2), \mathit{not}\ q(3)\\
s \leftarrow r, \mathit{not}\ q(1), \mathit{not}\ q(2), \mathit{not}\ p(3)\\
s \leftarrow r, \mathit{not}\ q(1), \mathit{not}\ q(2), \mathit{not}\ q(3).
\end{array}$$

One may introduce new atoms to avoid exponential blowups, but the translation is still not modular as it depends on the underlying domain; the multiple disjunctions or conjunctions need to be updated when the domain changes. More importantly, this method is not applicable if the language contains function constants of positive arity, as its Herbrand universe is infinite. Furthermore the method does not work when we consider non-Herbrand stable models.

One may also consider using Skolemization to eliminate existential quantifiers as in first-order logic, presuming that, for any sentence F and its Skolem form F', $\mathrm{SM}[F;\mathbf{p}]$ is satisfiable iff $\mathrm{SM}[F';\mathbf{p}]$ is satisfiable. However, unlike prenex form conversion, *Skolemization does not preserve the stable model semantics,* as the following example shows.

[9]Recall that an answer set is defined as a Herbrand stable model.

Example 3 *For*

$$F = (\forall x\, p(x) \to q) \wedge \neg\neg\exists x(q \wedge \neg p(x)), \tag{15}$$

$\mathrm{SM}[F; q]$ *is equivalent to first-order sentence*

$$(q \leftrightarrow \forall x\, p(x)) \wedge \exists x(q \wedge \neg p(x)),$$

which is unsatisfiable (the equivalence can be established using Theorems 3 and 11 from [Ferraris et al.*, 2010]). On the other hand, a Skolem form of F is*

$$F' = (p(a) \to q) \wedge \neg\neg(q \wedge \neg p(b))$$

where a and b are Skolem constants. Formula $\mathrm{SM}[F'; q]$ *is equivalent to*

$$(q \leftrightarrow p(a)) \wedge (q \wedge \neg p(b)),$$

which is satisfiable.

Below we present a method for eliminating some quantifiers by introducing auxiliary predicates. As we described earlier, in order to eliminate $\exists$ in (6), we introduce a new predicate constant p', and turn (6) into

$$(r \wedge \neg p' \to s) \wedge \forall x(p(x) \wedge q(x) \to p'), \tag{16}$$

which is the FOL-representation of (5). Formula $\mathrm{SM}[(6);\ p, q, r, s]$ has the same models as

$$\mathrm{SM}[(r \wedge \neg p' \to s) \wedge \forall x(p(x) \wedge q(x) \to p');\ p, q, r, s, p'] \tag{17}$$

if we disregard p'. This method does not involve grounding, so that the translation does not depend on the domain and is not restricted to Herbrand models.

The idea can be generalized as follows. Following [Ferraris *et al.*, 2009], about a formula F we say that it is *negative* on a list $\mathbf{p}$ of predicate constants if members of $\mathbf{p}$ have no strictly positive occurrences in F. For any second-order sentences F and G of a signature and any subset σ of that signature, we say that F *is σ-equivalent to* G, denoted by $F \Leftrightarrow_\sigma G$, if the class of models of F restricted to σ is identical to the class of models of G restricted to σ. The following proposition tells us how to replace some subformulas with new atoms.

Theorem 3 *Let F be a sentence of a signature σ, let $\mathbf{p}$ be a finite list of distinct predicate constants and let q be a predicate constant that does not belong to σ. Consider any non-strictly positive occurrence of a subformula $G(\mathbf{x})$ in F which is contained in a subformula of F that is negative on $\mathbf{p}$, where $\mathbf{x}$ is the list of all free variables of $G(\mathbf{x})$. Let F' be the formula obtained from F by replacing that occurrence with $q(\mathbf{x})$. Then*

$$\begin{array}{rcl} \mathrm{SM}[F; \mathbf{p}] & \Leftrightarrow_\sigma & \mathrm{SM}[F'; \mathbf{p}] \wedge \mathrm{SM}[\forall \mathbf{x}(G(\mathbf{x}) \to q(\mathbf{x}));\ q] \\ & \Leftrightarrow & \mathrm{SM}[F' \wedge \forall \mathbf{x}(G(\mathbf{x}) \to q(\mathbf{x}));\ \mathbf{p}, q]. \end{array}$$

This theorem is in particular useful for eliminating some positive occurrences of existential quantifiers: if $G(\mathbf{x})$ above is $\exists y H(\mathbf{x}, y)$,

$$\mathrm{SM}[F' \wedge \forall \mathbf{x}(\exists y H(\mathbf{x}, y) \rightarrow q(\mathbf{x}));\ \mathbf{p}, q]$$

is equivalent to

$$\mathrm{SM}[F' \wedge \forall \mathbf{x} y(H(\mathbf{x}, y) \rightarrow q(\mathbf{x}));\ \mathbf{p}, q]$$

in view of prenex conversion.

Example 4 *In formula (6), $\exists x(p(x) \wedge q(x))$ is contained in a negative formula (relative to any set of intensional predicates). According to Theorem 3,* $\mathrm{SM}[(6);\ p, q, r, s]$ *has the same models as*

$$\mathrm{SM}[(r \wedge \neg p' \rightarrow s) \wedge (\exists x(p(x) \wedge q(x)) \rightarrow p');\ p, q, r, s, p'] \tag{18}$$

if we disregard p'. Formula (18) is equivalent to (17).

In view of the Theorem on Double Negations from [Ferraris *et al.*, 2009], also reviewed in Section B.1, any negative occurrence of a formula $\forall y H(\mathbf{x}, y)$ contained in a subformula of F that is negative on $\mathbf{p}$ can be eliminated by first rewriting $\forall y H(\mathbf{x}, y)$ as $\neg\exists y \neg H(\mathbf{x}, y)$, and then applying the transformation in Theorem 3.[10] For example, $\mathrm{SM}[(15);\ q]$ is equivalent to

$$\mathrm{SM}[(\neg\exists x \neg p(x) \rightarrow q) \wedge \neg\neg\exists x(q \wedge \neg p(x));\ q]$$

which is equivalent to

$$\mathrm{SM}[(\neg p' \rightarrow q) \wedge \neg\neg q' \wedge \forall x(\neg p(x) \rightarrow p') \wedge \forall x(q \wedge \neg p(x) \rightarrow q');\ q, p', q'],$$

if we disregard p', q'.

We say that a formula F is *almost universal relative to* $\mathbf{p}$ if every singular occurrence of QxG in F is contained in a subformula of F that is negative on $\mathbf{p}$. For example, (6) is almost universal relative to any set of predicates because the only singular occurrence of $\exists x(p(x) \wedge q(x))$ in (6) is contained in $\neg\exists x(p(x) \wedge q(x))$, which is negative on any list of predicates. Formula (15) is almost universal relative to $\{q\}$ because the singular occurrence of $\forall x\, p(x)$ is contained in the formula itself, which is negative on q, and the singular occurrence of $\exists x(q \wedge \neg p(x))$ is contained in $\neg\exists x(q \wedge \neg p(x))$.

The following procedure can be used to eliminate all (possibly nested) quantifiers in any almost universal sentence.

[10]Recall that we distinguish between formula being negative and an occurrence being negative.

Algorithm 2 (ELIM-QUANTIFIERS) *Given an almost universal formula F relative to $\mathbf{p}$, first prepend $\neg\neg$ to every maximal strictly positive occurrence of formulas of the form $\exists y H(\mathbf{x}, y)$, and then repeat the following until there are no occurrences of quantifiers remaining:*

Select a maximal occurrence of a formula of the form $QyG(\mathbf{x}, y)$ in F where Q is $\forall$ or $\exists$, and $\mathbf{x}$ is the list of all free variables in $QyG(\mathbf{x}, y)$.

(a) If Q is $\exists$ and the occurrence of $QyG(\mathbf{x}, y)$ in F is negative, or if Q is $\forall$ and the occurrence of $QyG(\mathbf{x}, y)$ in F is positive, then set F to be the formula obtained from F by replacing the occurrence of $QyG(\mathbf{x}, y)$ with $G(\mathbf{x}, z)$ where z is a new variable.

(b) If Q is $\exists$ and the occurrence of $QyG(\mathbf{x}, y)$ in F is positive, then set F to be

$$F' \wedge (G(\mathbf{x}, y) \rightarrow p_G(\mathbf{x}))$$

where p_G is a new predicate constant and F' is the formula obtained from F by replacing the occurrence of $QyG(\mathbf{x}, y)$ with $p_G(\mathbf{x})$.

(c) If Q is $\forall$ and the occurrence of $QyG(\mathbf{x}, y)$ in F is negative, then set F to be the formula obtained from F by replacing the occurrence of $QyG(\mathbf{x}, y)$ with $\neg\exists y \neg G(\mathbf{x}, y)$.

We assume that the new predicate constants introduced by the translation do not belong to the signature of F. The following theorem tells us that any almost universal sentence F can be turned into the form $\forall\mathbf{x}G$ where G is a quantifier-free formula.

Theorem 4 *Let F be a sentence of a signature σ that is almost universal relative to $\mathbf{p}$, let F' be the universal closure of the formula obtained from F by applying translation* ELIM-QUANTIFIERS, *and let $\mathbf{q}$ be the list of new predicate constants introduced by the translation. Then* $\mathrm{SM}[F; \mathbf{p}]$ *is σ-equivalent to* $\mathrm{SM}[F'; \mathbf{p} \cup \mathbf{q}]$.

The statement of the theorem becomes incorrect if F is not required to be almost universal relative to $\mathbf{p}$. For instance, if ELIM-QUANTIFIERS is applied to $\exists x\, p(x)$, it will result in $\forall x(\neg\neg q \wedge (p(x) \rightarrow q))$. However, $\mathrm{SM}[\exists x\, p(x);\ p]$ is not $\{p\}$-equivalent to $\mathrm{SM}[\forall x(\neg\neg q \wedge (p(x) \rightarrow q));\ p, q]$. The former means that p is a singleton, but the latter is inconsistent.

4.2 F2LP: Computing Answer Sets of First-Order Formulas

Using the translation ELIM-QUANTIFIERS defined in the previous section, we introduce the translation F2LP that turns a first-order formula into a logic program, which allows us to use existing answer set solvers for computing answer sets of the formula. We assume that the underlying signature contains finitely many predicate constants.

Algorithm 3 (Translation F2LP) *1. Given an almost universal formula F and a list of intensional predicates $\mathbf{p}$, apply translation* ELIM-QUANTIFIERS *(Algorithm 2) to F;*

2. *Add choice formulas $(q(\mathbf{x}) \vee \neg q(\mathbf{x}))$ for all non-intensional predicates q.*

3. *Turn the resulting quantifier-free formula into a logic program by applying the translation from [Cabalar* et al.*, 2005, Section 3].*[11]

The following theorem asserts the correctness of the translation.

Theorem 5 *Let F be a sentence of a signature σ that is almost universal relative to $\mathbf{p}$, let Π be the program obtained from F by applying translation* F2LP *with $\mathbf{p}$ as intensional predicates, and let Π^{FOL} be the FOL-representation of Π. Then* $\mathrm{SM}[F;\mathbf{p}]$ *is σ-equivalent to*

$$\mathrm{SM}[\Pi^{FOL} \wedge \mathit{False}(\mathbf{p} \setminus pr(\Pi^{FOL}))].$$

Example 5 *Consider one of the domain independent axioms in the discrete event calculus (DEC5 axiom):*

$$\begin{array}{l} \mathit{HoldsAt}(f,t) \wedge \neg \mathit{ReleasedAt}(f,t{+}1) \wedge \\ \qquad \neg \exists e(\mathit{Happens}(e,t) \wedge \mathit{Terminates}(e,f,t)) \rightarrow \mathit{HoldsAt}(f,t{+}1). \end{array} \tag{19}$$

Step 1 of translation F2LP *introduces the formula*

$$\mathit{Happens}(e,t) \wedge \mathit{Terminates}(e,f,t) \rightarrow q(f,t),$$

and replaces (19) with

$$\mathit{HoldsAt}(f,t) \wedge \neg \mathit{ReleasedAt}(f,t{+}1) \wedge \neg q(f,t) \rightarrow \mathit{HoldsAt}(f,t{+}1).$$

Step 3 turns these formulas into rules

$$q(f,t) \leftarrow \mathit{Happens}(e,t),\ \mathit{Terminates}(e,f,t)$$

$$\mathit{HoldsAt}(f,t{+}1) \leftarrow \mathit{HoldsAt}(f,t), \mathit{not}\ \mathit{ReleasedAt}(f,t{+}1),\ \mathit{not}\ q(f,t).$$

Turning the program obtained by applying the translation F2LP into the input languages of LPARSE and GRINGO requires minor rewriting, such as moving equality and negated atoms in the head to the body[12] and adding domain predicates in

[11][Cabalar *et al.*, 2005] presents two transformations. System F2LP uses *vocabulary-preserving* transformation.

[12]For instance, `(X=Y) | -q(X,Y) :- p(X,Y)` is turned into
`:- X!=Y, {not q(X,Y)}0, p(X,Y).`

the body for all variables occurring in the rule in order to reduce the many sorted signature into the non-sorted one.[13]

System F2LP is an implementation of the translation F2LP, which turns a formula into the languages of LPARSE and GRINGO. The system can be downloaded from its home page

```
http://reasoning.eas.asu.edu/f2lp/ .
```

Formulas can be encoded in the language of F2LP using the following ASCII characters.

Symbol	$\neg$	$\wedge$	$\vee$	$\rightarrow$	$\bot$	$\top$	$\forall xyz$	$\exists xyz$
ASCII	`-`	`&`	`|`	`->`	`false`	`true`	`![X,Y,Z]:`	`?[X,Y,Z]:`

The system also allows extended rule form $F \leftarrow G$ where F and G are formulas. In this case `not` is used to represent $\neg$, and `<-` is used to represent $\leftarrow$. Figure 2 in Section 6.2 shows an example in the input language of F2LP.

The usual LPARSE and GRINGO rules are also allowed in F2LP. They are simply copied to the output. The program returned by F2LP can be passed to ASP grounders and solvers that accept LPARSE and GRINGO languages.

5 Reformulating the Event Calculus in the Stable Model Semantics

5.1 Review: Circumscriptive Event Calculus

Recall that the equivalence between circumscription and the stable model semantics depends on the syntactic condition of canonical formulas. Here we review the syntax of circumscriptive event calculus described in [Mueller, 2006, Chapter 2], and note that it satisfies the condition on canonical formulas.

We assume a many-sorted first-order language, which contains an *event* sort, a *fluent* sort, and a *timepoint* sort. A *fluent term* is a term whose sort is a fluent; an *event term* and a *timepoint term* are defined similarly. A *condition* is defined recursively as follows:

- If τ_1 and τ_2 are terms, then comparisons $\tau_1 < \tau_2$, $\tau_1 \leq \tau_2$, $\tau_1 \geq \tau_2$, $\tau_1 > \tau_2$, $\tau_1 = \tau_2$, $\tau_1 \neq \tau_2$ are conditions;
- If f is a fluent term and t is a timepoint term, then $HoldsAt(f, t)$ and $\neg HoldsAt(f, t)$ are conditions;
- If γ_1 and γ_2 are conditions, then $\gamma_1 \wedge \gamma_2$ and $\gamma_1 \vee \gamma_2$ are conditions;

[13]Alternatively this can be done by declaring variables using the `#domain` directive in LPARSE and GRINGO languages.

- If v is a variable and γ is a condition, then $\exists v \gamma$ is a condition.

We will use e and e_i to denote event terms, f and f_i to denote fluent terms, t and t_i to denote timepoint terms, and γ and γ_i to denote conditions. We assume the same set of primitive connectives and quantifiers as in Section 2.

An event calculus domain description is defined as

$$\begin{aligned}&\mathrm{CIRC}[\Sigma\ ;\ \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{CIRC}[\Delta\ ;\ \mathit{Happens}]\\ &\wedge\ \mathrm{CIRC}[\Theta\ ;\ \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \wedge \Xi\end{aligned} \tag{20}$$

where

- Σ is a conjunction of universal closures of axioms of the form

  $$\begin{aligned}&\gamma \rightarrow \mathit{Initiates}(e, f, t)\\ &\gamma \rightarrow \mathit{Terminates}(e, f, t)\\ &\gamma \rightarrow \mathit{Releases}(e, f, t)\\ &\gamma \wedge \pi_1(e, f_1, t) \rightarrow \pi_2(e, f_2, t) \qquad (\text{“effect constraint”})\\ &\gamma \wedge [\neg]\mathit{Happens}(e_1, t) \wedge \cdots \wedge [\neg]\mathit{Happens}(e_n, t) \rightarrow \mathit{Initiates}(e, f, t)\\ &\gamma \wedge [\neg]\mathit{Happens}(e_1, t) \wedge \cdots \wedge [\neg]\mathit{Happens}(e_n, t) \rightarrow \mathit{Terminates}(e, f, t)\end{aligned}$$

  where each of π_1 and π_2 is either *Initiates* or *Terminates* ($[\neg]$ means that $\neg$ is optional);

- Δ is a conjunction of universal closures of *temporal ordering formulas* (comparisons between timepoint terms) and axioms of the form

  $$\begin{aligned}&\gamma \rightarrow \mathit{Happens}(e, t)\\ &\sigma(f, t) \wedge \pi_1(f_1, t) \wedge \cdots \wedge \pi_n(f_n, t) \rightarrow \mathit{Happens}(e, t) \qquad (\text{“causal constraints”})\\ &\mathit{Happens}(e, t) \rightarrow \mathit{Happens}(e_1, t) \vee \cdots \vee \mathit{Happens}(e_n, t) \quad (\text{“disjunctive event axiom”})\end{aligned}$$

  where σ is *Started* or *Stopped* and each π_j $(1 \leq j \leq n)$ is either *Initiated* or *Terminated*.

- Θ is a conjunction of universal closures of *cancellation* axioms of the form

  $$\gamma \rightarrow \mathit{Ab}_i(..., t);$$

- Ξ is a conjunction of first-order sentences (outside the scope of circumscription) including unique name axioms, state constraints, event occurrence constraints, and the set of domain-independent axioms in the event calculus, such as *EC* and *DEC* axioms [Mueller, 2006]. It also includes the definitions of *Initiated* and *Terminated* for causal constraints in Δ:

  $$\begin{aligned}&\mathit{Started}(f, t) \overset{def}{\leftrightarrow} (\mathit{HoldsAt}(f, t) \vee \exists e(\mathit{Happens}(e, t) \wedge \mathit{Initiates}(e, f, t))) \quad (CC_1)\\ &\mathit{Stopped}(f, t) \overset{def}{\leftrightarrow} (\neg\mathit{HoldsAt}(f, t) \vee \exists e(\mathit{Happens}(e, t) \wedge \mathit{Terminates}(e, f, t))) \quad (CC_2)\\ &\mathit{Initiated}(f, t) \overset{def}{\leftrightarrow} (\mathit{Started}(f, t) \wedge \neg\exists e(\mathit{Happens}(e, t) \wedge \mathit{Terminates}(e, f, t))) \quad (CC_3)\\ &\mathit{Terminated}(f, t) \overset{def}{\leftrightarrow} (\mathit{Stopped}(f, t) \wedge \neg\exists e(\mathit{Happens}(e, t) \wedge \mathit{Initiates}(e, f, t))) \quad (CC_4);\end{aligned}$$

It is clear that all axioms in Σ are canonical relative to $\{Initiates, Terminates, Releases\}$; all axioms in Δ are canonical relative to $\{Happens\}$; all axioms in Θ are canonical relative to $\{Ab_1, \dots, Ab_n\}$.

5.2 Reformulating the Event Calculus in the Stable Model Semantics

The following theorem shows a few equivalent reformulations of circumscriptive event calculus in terms of the stable model semantics. We assume that Ξ was already equivalently rewritten so that Ξ is negative on $\{Initiates, Terminates, Releases, Happens, Ab_1, \dots, Ab_n\}$. This can be done by prepending $\neg\neg$ to strictly positive occurrences of those predicates.

Theorem 6 *For any event calculus description (20), the following theories are equivalent to each other:*[14]

(a) $\text{CIRC}[\Sigma; I, T, R] \wedge \text{CIRC}[\Delta; H] \wedge \text{CIRC}[\Theta; Ab_1, \dots, Ab_n] \wedge \Xi$;

(b) $\text{SM}[\Sigma; I, T, R] \wedge \text{SM}[\Delta; H] \wedge \text{SM}[\Theta; Ab_1, \dots, Ab_n] \wedge \Xi$;

(c) $\text{SM}[\Sigma \wedge \Delta \wedge \Theta \wedge \Xi; I, T, R, H, Ab_1, \dots, Ab_n]$;

(d) $\text{SM}[\Sigma \wedge \Delta \wedge \Theta \wedge \Xi \wedge Choice(pr(\Sigma \wedge \Delta \wedge \Theta \wedge \Xi) \setminus \{I, T, R, H, Ab_1, \dots, Ab_n\})]$.

The assumption that Ξ is negative on the intensional predicates is essential. For example, consider the case when Σ, Θ are empty, and Δ contains just one axiom:

$$Happens(E, 0). \tag{21}$$

Assume that Ξ consists of the event occurrence constraint

$$Happens(E, t) \rightarrow Happens(E_1, t), \tag{22}$$

the UNA

$$E \neq E_1, \tag{23}$$

and the domain independent axioms EC and CC=CC_1-CC_4. It is clear that any interpretation that satisfies $\text{CIRC}[(21); Happens]$ will not satisfy $(22) \wedge (23)$. This implies that the event calculus description (a) is inconsistent. On the other hand, (c) is the formula

$$\text{SM}[(21) \wedge (22) \wedge (23) \wedge EC \wedge CC; I, T, R, H]. \tag{24}$$

One can check that there exists an interpretation J such that $E^J \neq E_1^J$, $Happens^J = \{(E^J, 0^J), (E_1^J, 0^J)\}$, $Initiates^J, Terminates^J, Releases^J$ are empty and $J \models$

[14]For brevity, we abbreviate the names of circumscribed predicates.

(21) $\wedge$ (22) $\wedge$ (23) $\wedge$ $EC \wedge CC$. This implies that J satisfies (24). However, if we write (22) as

$$Happens(E, t) \rightarrow \neg\neg Happens(E_1, t),$$

then the resulting theory (c) has no stable models.

Like answer set programs, the intensional predicates of formula (d) in Theorem 6 are all the predicates that occur in the event calculus description. On the other hand, $\Sigma \wedge \Delta \wedge \Theta \wedge \Xi$ contains formulas that may not be in the rule form, and may contain existential quantifiers.

5.3 Turning Event Calculus Descriptions to Answer Set Programs

The following procedure turns an event calculus description into an answer set program.

Algorithm 4 (Translation EC2ASP) *1. Given an event calculus description, rewrite all the definitional axioms of the form*

$$\forall \mathbf{x}(p(\mathbf{x}) \overset{def}{\leftrightarrow} G) \tag{25}$$

as $\forall \mathbf{x}(G' \rightarrow p(\mathbf{x}))$ where G' is obtained from G by prepending $\neg\neg$ to all occurrences of Initiates Terminates, Releases, Happens, $Ab_1, \dots, Ab_n$. Also prepend $\neg\neg$ to the strictly positive occurrences of the same predicates in the remaining axioms of Ξ.[15]*.*

2. Apply translation F2LP to the resulting theory with intensional predicates

$$\{Initiates, Terminates, Releases, Happens, Ab_1, \dots, Ab_n\} \cup \mathbf{p}$$

where $\mathbf{p}$ is the set of all predicate constants p considered in Step 1.

The following theorem states the correctness of the translation.

Theorem 7 *Let T be an event calculus domain description, let Π^{FOL} be the FOL-representation of the program Π obtained from T by applying the translation EC2ASP. Then T is $\sigma(T)$-equivalent to* $\mathrm{SM}[\Pi^{FOL}]$.

6 Experiments and Comparison

6.1 Comparison with the DEC Reasoner

Theorem 7 provides a computing method for the event calculus using the combination of F2LP and answer set solvers. In this section, we compare this method

[15]Theorem 7 becomes wrong if we do not prepend $\neg\neg$ to these predicates. The reason is similar to the explanation that follows the statement of Theorem 6.

with another event calculus system called the DEC reasoner, implemented by Erik Mueller [Mueller, 2004b]. The DEC reasoner is similar to our method in the sense that it is based on the reduction of event calculus reasoning to satisfiability checking and uses efficient SAT solvers for computation. On the other hand, the reduction is based on completion, so that the system does not handle recursive axioms or disjunctive axioms, such as effect constraints and disjunctive event axioms (Section 5.1). For example, the following are effect constraints that describe the indirect effects of an agent's walking on the objects that he is holding:

$$\begin{array}{l} \mathit{HoldsAt}(\mathit{Holding}(a,o),t) \wedge \mathit{Initiates}(e,\mathit{InRoom}(a,r),t) \\ \qquad\qquad\qquad\qquad \rightarrow \mathit{Initiates}(e,\mathit{InRoom}(o,r),t) \\ \mathit{HoldsAt}(\mathit{Holding}(a,o),t) \wedge \mathit{Terminates}(e,\mathit{InRoom}(a,r),t) \\ \qquad\qquad\qquad\qquad \rightarrow \mathit{Terminates}(e,\mathit{InRoom}(o,r),t) \end{array} \tag{26}$$

On the other hand, the logic program rules corresponding to (26) can be handled by answer set solvers. Indeed, since circumscription coincides with the stable model semantics for canonical formulas, the ASP approach can handle the *full* version of the event calculus, if the domain is given and finite.

Problem (max. step)	DEC reasoner	F2LP with LPARSE + CMODELS	F2LP with GRINGO + CMODELS	F2LP with GRINGO + CLASP(D)	F2LP with CLINGO
BusRide (15)	—	0.47 (0.43 + 0.04) A:902/R:7779 C:0	0.05 (0.03 + 0.02) A:683/R:3514 C:0	0.04 (0.03 + 0.01) A:778/R:3593	—
Commuter (15)	—	592.04 (536.45 + 55.59) A:32861/R:8734019 C:0	77.03 (40.54 + 36.49) A:5269/R:6020143 C:5344	49.22 (41.47 + 8.75) A:30092/R:6020143	31.32
Kitchen Sink (25)	71.10 (70.70+0.40) A:1014/C:12109	43.75 (37.69+6.06) A:121621/R:480187 C:0	8.67 (4.42 + 4.25) A:121043/R:474946 C:0	6.34 (4.40+1.94) A:121043/R:474946	4.75
Thielscher Circuit (20)	11.10 (10.50+0.6) A:1394/C:42454	0.52 (0.47+0.05) A:3932/R:10999 C:0	0.08 (0.05 + 0.03) A:1866/R:6634 C:0	0.07 (0.05+0.02) A:1866/R:6634	0.05
Walking Turkey (15)	—	0.03 (0.03+0.00) A:370/R:518 C:0	0.01 (0.01 + 0.00) A:367/R:507 C:0	0.01 (0.01+0.00) A:367/R:507	0.01
Falling w/ AntiTraj (15)	270.2 (269.3+0.9) A:416/C:3056	0.74 (0.66+0.08) A:4994/R:9717 C:0	0.13 (0.07 + 0.06) A:4125/R:7823 C:0	0.10 (0.07+0.03) A:4125/R:7823	0.08
Falling w/ Events (25)	107.70 (107.50+0.20) A:1092/C:12351	34.95 (31.12+3.83) A:1240/R:388282 C:1436	4.68 (2.01 + 2.67) A:1219/R:202202 C:1415	2.99 (2.01+0.98) A:146197/R:202202	2.35
HotAir Baloon (15)	61.10 (61.10+0.00) A:288/C:1163	0.18 (0.15+0.03) A:494/R:2451 C:689	0.05 (0.03 + 0.02) A:492/R:1912 C:681	0.03 (0.03+0.00) A:1140/R:1912	0.03
Telephone1 (40)	18.20 (17.70+0.50) A:5419/C:41590	1.69 (1.51+0.18) A:21414/R:27277 C:0	0.49 (0.26 + 0.23) A:1938/R:27104 C:1939	0.31 (0.26+0.05) A:21376/R:27104	0.24

A: number of atoms, C: number of clauses, R: number of ground rules

Figure 1: Comparing the DEC reasoner and F2LP with answer set solvers

We compared the performance of the DEC reasoner (v 1.0) running RELSAT (v 2.0) with the following:

- F2LP with LPARSE (v 1.1.1)+CMODELS (v 3.79) running RELSAT (v 2.0),
- F2LP with GRINGO (v 2.0.3)+CMODELS (v 3.79) running RELSAT (v 2.0),
- F2LP with GRINGO (v 2.0.3) +CLASP (v 1.2.1) (CLASPD (v 1.1) used instead for disjunctive programs), and
- F2LP with CLINGO (v 2.0.3).

F2LP turns an input theory into the languages of LPARSE and GRINGO, and LPARSE and GRINGO turn the result into a ground ASP program. CMODELS turns this ground program into a set of clauses and then invokes a SAT solver to compute answer sets, while CLASP computes answer sets using the techniques similar to those used in SAT solvers but without actually invoking them. CLINGO is a system that combines GRINGO and CLASP in a monolithic way.

The first five examples in Figure 1 are part of the benchmark problems from [Shanahan, 1997; Shanahan, 1999]. The next four are from [Mueller, 2006]. (We increased timepoints to see more notable differences.) More examples can be found from the F2LP homepage. All experiments were done on a Pentium machine with 3.00 GHz CPU and 2GB RAM running 64 bit Linux. The reported run times are in seconds and were obtained using the Linux `time` command, except for the DEC reasoner for which we recorded the times reported by the system. This was to avoid including the time spent by the DEC reasoner in producing output in a neat format, which may take some time. For the DEC reasoner, the times in parentheses are "(encoding time + SAT solving time)." For the others, they are the times spent by each of the grounder and the solver. CMODELS time includes the time spent in converting the ground program generated by LPARSE/GRINGO into a set of clauses, and calling the SAT solver. The time spent by F2LP in translating an event calculus description into a logic program (with variables) is negligible for these problems. '—' denotes that the system cannot solve the example due to the limited expressivity. For instance, `BusRide` includes disjunctive event axioms, which results in a disjunctive program that cannot be handled by CLINGO. Similarly, the DEC reasoner cannot handle `BusRide`, `Commuter` (compound event) and `Walking Turkey` (effect constraint). Overall, F2LP with CLINGO was the clear winner, followed by F2LP with GRINGO + CLASP. As is evident from the experiments, the main reason for the efficiency of the ASP-based approach is due to the efficient grounding mechanisms implemented in the ASP grounders. Though the DEC reasoner and CMODELS call the same SAT solver RELSAT,[16] the number of atoms produced by the DEC reasoner is in general much smaller. This is because

[16]The choice of the SAT solver is not critical since SAT solving times are negligible in the experiments.

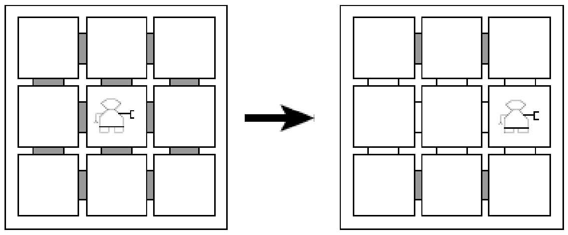

Figure 2: Robby's apartment in a 3×3 grid

the DEC reasoner adopts an optimized encoding method (that is based on predicate completion) which avoids a large number of ground instances of atoms such as $\mathit{Initiates}(e, f, t)$, $\mathit{Terminates}(e, f, t)$, and $\mathit{Releases}(e, f, t)$ [Mueller, 2004a, Section 4.4]. On the other hand, in several examples, the number of clauses generated by CMODELS is 0, which means that the answer sets were found without calling the SAT solver. This is because for these examples the unique answer set coincides with the well-founded model, which is efficiently computed by CMODELS without calling SAT solvers. Out of the 14 benchmark examples from [Shanahan, 1997; Shanahan, 1999], 10 of them belong to this case when LPARSE is used for grounding.

6.2 Reasoning Involving Transitive Closure

One of the advantages of ASP over completion is the ability to represent transitive closure. The reformulation of the event calculus in terms of ASP allows us to use ASP rules together with the event calculus axioms, which in turn allows us to enhance the event calculus reasoning. We illustrate this using the example from [Doğandağ *et al.*, 2004]. There are 9 rooms and 12 doors as shown in Figure 2. Initially the robot "Robby" is in the middle room and all the doors are closed. The goal of the robot is to make all rooms accessible from each other. Figure 3 (File 'robby') shows an encoding of the problem in the language of F2LP. Predicates `door(x, y)` denotes that there is a door between rooms `x` and `y`; `open(x, y)` denotes the event "Robby opening the door between rooms `x` and `y`"; `goto(x)` denotes the event "Robby going to room `x`"; `opened(x, y)` denotes that the door between `x` and `y` has been opened; `inRoom(x)` denotes that Robby is in room `x`; `accessible(x, y)` denotes that `y` is accessible from `x`. Note that the rules defining the relation `accessible` are not event calculus axioms (Section 5.1). This example illustrates an advantage of allowing ASP rules in event calculus descriptions.

The minimal number of steps to solve the given problem is 11. We can find such a plan using the combination of F2LP, GRINGO and CLASPD in the following way.

```
% File 'robby'

% objects
step(0..maxstep).
astep(0..maxstep-1).
room(1..9).

% variables
#domain step(T).
#domain room(R).
#domain room(R1).
#domain room(R2).

% position of the doors
door(R1,R2) <- R1 >= 1 & R2 >=1 & R1 < 4 & R2 < 4 & R2 = R1+1.
door(R1,R2) <- R1 >= 4 & R2 >= 4 & R1 < 7 & R2 < 7 & R2 = R1+1.
door(R1,R2) <- R1 >= 7 & R2 >= 7 & R1 < 10 & R2 < 10 & R2 = R1+1.
door(R1,R2) <- R2 < 10 & R2 = R1+3.

door(R1,R2) <- door(R2,R1).

% fluents
fluent(opened(R,R1)) <- door(R1,R2).
fluent(inRoom(R)).
% F ranges over the fluents
#domain fluent(F).

% events
event(open(R,R1)) <- door(R,R1).
event(goto(R)).
% E and E1 range over the events
#domain event(E).
#domain event(E1).

% effect axioms
initiates(open(R,R1),opened(R,R1),T).
initiates(open(R,R1),opened(R1,R),T).

initiates(goto(R2),inRoom(R2),T)
  <- holdsAt(opened(R1,R2),T) & holdsAt(inRoom(R1),T).

terminates(E,inRoom(R1),T)
  <- holdsAt(inRoom(R1),T) & initiates(E,inRoom(R2),T).

% action precondition axioms
holdsAt(inRoom(R1),T) <- happens(open(R1,R2),T).
```

```
% event occurrence constraint
not happens(E1,T) <- happens(E,T) & E != E1.

% state constraint
not holdsAt(inRoom(R2),T) <- holdsAt(inRoom(R1),T) & R1 != R2.

% accessibility
accessible(R,R1,T) <- holdsAt(opened(R,R1),T).
accessible(R,R2,T) <- accessible(R,R1,T) & accessible(R1,R2,T).

% initial state
not holdsAt(opened(R1,R2),0).
holdsAt(inRoom(5),0).

% goal state
not not accessible(R,R1,maxstep).

% happens is exempt from minimization in order to find a plan.
{happens(E,T)} <- T < maxstep.

% all fluents are inertial
not releasedAt(F,0).
```

Figure 3: Robby in F2LP

```
$ f2lp dec robby | gringo -c maxstep=11 | claspD
```

File `dec` is an F2LP encoding of the domain independent axioms in the Discrete Event Calculus (The file is given in Appendix A).[17] The following is one of the plans found:

```
happens(open(5,6),0) happens(open(5,8),1) happens(open(5,4),2)
happens(goto(6),3) happens(open(6,9),4) happens(open(6,3),5)
happens(goto(5),6) happens(open(5,2),7) happens(goto(4),8)
happens(open(4,7),9) happens(open(4,1),10)
```

6.3 Comparison with Mueller's ASP Approach

Our work on ASP-based event calculus was motivated by Erik Mueller's work that is available on the webpage

`http://decreasoner.sourceforge.net/csr/ecas/,`

[17]The file is also available at `http://reasoning.eas.asu.edu/f2lp`, along with F2LP encodings of the domain independent axioms in other versions of the event calculus.

where a few example answer set programs are given to illustrate that event calculus like reasoning can be done using ASP solvers. However, this is rather a "proof of concept"[18] and no formal justification is provided there. Still we observe a few differences.

First, these examples use strong negation, while our method does not. We do not need both negations—default negation (*not*) and strong negation ($\neg$)—to embed the two-valued event calculus into ASP. Second, no choice rules are used in Mueller's work, which results in limiting attention to temporal projection problems—to determine the states that result from performing a sequence of actions. On the other hand, our approach can handle not only temporal projection problems, but also planning and postdiction problems. To solve a planning problem *Happens* should not be minimized. Adding choice rules for *Happens* is a way to exempt it from minimization in answer set programs (see Figure 3 for example).

7 Related Work and Conclusion

The first-order stable model semantics is closely related to equilibrium logic [Pearce and Valverde, 2005], and indeed, Ferraris *et al.* [2010] showed that they are essentially equivalent. Based on the similarity between the definitions of the stable model semantics and circumscription, Ferraris *et al.* [2007] showed that the first-order stable model semantics can be characterized by circumscription, extending the work by Lin [1991], which was limited to the syntax of logic programs in the propositional case. Essentially the same characterization was independently given in [Lin and Zhou, 2007], via logic of knowledge and justified assumption from [Lin and Shoham, 1992]. The theorems on canonical and pseudo-canonical formulas in this paper provide a characterization in the other direction. The result is an extension of the characterization from [Lee and Lin, 2004] that was limited to the propositional case. Janhunen and Oikarinen [2004] showed another characterization of propositional circumscription in terms of the stable model semantics, but the translation uses auxiliary atoms and the idea appears very different.

Just as the concept of *tight formulas* [Ferraris *et al.*, 2010] helps us understand the relationship between the first-order stable model semantics and the completion semantics,[19] the concept of *canonical formulas* (and *pseudo-canonical formulas*) helps us understand the relationship between the first-order stable model semantics and circumscription, and, together with the reduction of the formulas to logic programs, allows us to compute Herbrand models of circumscriptive theories using answer set solvers.

We applied the results on canonical formulas and F2LP to reformulate circumscriptive event calculus in terms of the stable model semantics and to compute the

[18]Personal communication with Erik Mueller (June 28, 2008).

[19]See Section B.1 for the reviews of the definition and the related theorem.

event calculus using answer set solvers. Our experiments indicate that answer set programming based event calculus reasoning is a promising approach, overcoming certain limitations of the completion-based computation. The methods developed in this paper were also applied to turn the situation calculus into ASP in [Lee and Palla, 2010].

More synergies are expected from the relationship between the event calculus and ASP. In addition to action languages, circumscriptive event calculus can now be regarded as another useful high level action formalism for ASP, addressing some issues that are not well regarded in the context of action languages, such as describing continuous change and hierarchical planning. Embedding both the event calculus and action languages into ASP may be useful in comparing these high level formalisms and even in merging the descriptions written in each of them.

Acknowledgements

We are grateful to Yuliya Lierler, Vladimir Lifschitz and Erik Mueller for useful comments on the idea of this paper and several pointers to earlier work. We are also grateful to anonymous referees for their useful comments on the preliminary reports that this article extends. The authors were partially supported by the National Science Foundation under Grants IIS-0839821, IIS-0916116 and by the IARPA SCIL program.

A Appendix: File ‘`dec`’ in the Language of F2LP

```
% File ‘dec’

#domain fluent(F;F1;F2), event(E), time(T;T1;T2).

time(0..maxstep).

% DEC 1
stoppedIn(T1,F,T2) <- happens(E,T) & T1<T & T<T2 & terminates(E,F,T).

% DEC 2
startedIn(T1,F,T2) <- happens(E,T) & T1<T & T<T2 & initiates(E,F,T).

% DEC 3
holdsAt(F2,T1+T2) <- happens(E,T1) & initiates(E,F1,T1) & T2>0 &
   trajectory(F1,T1,F2,T2) & not stoppedIn(T1,F1,T1+T2) & T1+T2<=maxstep.

% DEC 4
holdsAt(F2,T1+T2) <- happens(E,T1) & terminates(E,F1,T1) & 0<T2 &
   antiTrajectory(F1,T1,F2,T2) & not startedIn(T1,F1,T1+T2) &
   T1+T2<=maxstep.

% DEC 5
holdsAt(F,T+1) <- holdsAt(F,T) & not releasedAt(F,T+1) &
   not ?[E]:(happens(E,T) & terminates(E,F,T)) & T<maxstep.

% DEC 6
not holdsAt(F,T+1) <- not holdsAt(F,T) & not releasedAt(F,T+1) &
   not ?[E]:(happens(E,T) & initiates(E,F,T)) & T<maxstep.

% DEC 7
releasedAt(F,T+1) <-
   releasedAt(F,T) & not ?[E]:(happens(E,T) &
   (initiates(E,F,T) | terminates(E,F,T))) & T<maxstep.

% DEC 8
not releasedAt(F,T+1) <- not releasedAt(F,T) &
   not ?[E]: (happens(E,T) & releases(E,F,T)) & T<maxstep.

% DEC 9
holdsAt(F,T+1) <- happens(E,T) & initiates(E,F,T) & T<maxstep.

% DEC 10
not holdsAt(F,T+1) <- happens(E,T) & terminates(E,F,T) & T<maxstep.

% DEC 11
releasedAt(F,T+1) <- happens(E,T) & releases(E,F,T) & T<maxstep.

% DEC 12
not releasedAt(F,T+1) <- happens(E,T) &
   (initiates(E,F,T) | terminates(E,F,T)) & T<maxstep.

{holdsAt(F,T)}.
{releasedAt(F,T)}.
```

B Appendix: Proofs

B.1 Review of Some Theorems

We review some theorems from [Ferraris *et al.*, 2010] and [Ferraris *et al.*, 2009] which will be used to prove our main results. In fact, we will provide a splitting theorem which is slightly more general than the one given in [Ferraris *et al.*, 2009], which will facilitate our proof efforts.

Lemma 2 *Formula*

$$\mathbf{u} \leq \mathbf{p} \to ((\neg F)^*(\mathbf{u}) \leftrightarrow \neg F)$$

is logically valid.

Theorem 8 *[Ferraris* et al.*, 2010, Theorem 2] For any first-order formula F and any disjoint lists $\mathbf{p}$, $\mathbf{q}$ of distinct predicate constants,*

$$\mathrm{SM}[F;\mathbf{p}] \leftrightarrow \mathrm{SM}[F \wedge \mathit{Choice}(\mathbf{q});\ \mathbf{p} \cup \mathbf{q}]$$

is logically valid.

Let F be a first-order formula. A *rule* of F is an implication that occurs strictly positively in F. The *predicate dependency graph* of F (relative to $\mathbf{p}$) is the directed graph that

- has all members of $\mathbf{p}$ as its vertices, and
- has an edge from p to q if, for some rule $G \rightarrow H$ of F,
  - p has a strictly positive occurrence in H, and
  - q has a positive occurrence in G that does not belong to any subformula of G that is negative on $\mathbf{p}$.

Theorem 9 *[Ferraris* et al.*, 2009, Splitting Theorem] Let F, G be first-order sentences, and let $\mathbf{p}$, $\mathbf{q}$ be disjoint lists of distinct predicate constants. If*

(a) each strongly connected component of the predicate dependency graph of $F \wedge G$ relative to $\mathbf{p}$, $\mathbf{q}$ is either a subset of $\mathbf{p}$ or a subset of $\mathbf{q}$,

(b) F is negative on $\mathbf{q}$, and

(c) G is negative on $\mathbf{p}$

then

$$\mathrm{SM}[F \wedge G;\ \mathbf{p} \cup \mathbf{q}] \leftrightarrow \mathrm{SM}[F;\ \mathbf{p}] \wedge \mathrm{SM}[G;\ \mathbf{q}]$$

is logically valid.

The theorem is slightly more general than the one from [Ferraris *et al.*, 2009] in that the notion of a dependency graph above yields less edges than the one given in [Ferraris *et al.*, 2009]. Instead of

- q has a positive occurrence in G that does not belong to any subformula of G that is negative on $\mathbf{p}$,

the definition from [Ferraris *et al.*, 2009] has

- q has a positive occurrence in G that does not belong to any subformula of the form $\neg K$.

For instance, according to [Ferraris *et al.*, 2009], the dependency graph of

$$((p \to q) \to r) \to p \tag{27}$$

relative to p has an edge that goes from p to itself, while according to our definition, the dependency graph has no edges.

However, the generalization is not essential in view of the following theorem.

Theorem 10 *[Ferraris* et al.*, 2009, Theorem on Double Negations] Let H be a sentence, F a subformula of H, and H^- the sentence obtained from H by inserting $\neg\neg$ in front of F. If F is contained in a subformula G of H that is negative on $\mathbf{p}$, then* $\text{SM}[H; \mathbf{p}]$ *is equivalent to* $\text{SM}[H^-; \mathbf{p}]$*.*

For instance, $\text{SM}[(27); p]$ is equivalent to $\text{SM}[\neg\neg((p \to q) \to r) \to p;\ p]$. The dependency graph of $\neg\neg((p \to q) \to r) \to p$ relative to p according to the definition from [Ferraris *et al.*, 2009] is identical to the dependency graph of (27) relative to p according to our definition.

Next, we say that a formula F is in *Clark normal form* (relative to the list $\mathbf{p}$ of intensional predicates) if it is a conjunction of formulas of the form

$$\forall \mathbf{x}(G \to p(\mathbf{x})), \tag{28}$$

one for each intensional predicate p, where $\mathbf{x}$ is a list of distinct object variables. The *completion* (relative to $\mathbf{p}$) of a formula F in Clark normal form is obtained by replacing each conjunctive term (28) with

$$\forall \mathbf{x}(p(\mathbf{x}) \leftrightarrow G).$$

The following theorem from [Ferraris *et al.*, 2010] relates SM to completion. We say that F is *tight* on $\mathbf{p}$ if the predicate dependency graph of F relative to $\mathbf{p}$ is acyclic.

Theorem 11 *[Ferraris* et al.*, 2010] For any formula F in Clark normal form that is tight on $\mathbf{p}$, formula* $\text{SM}[F; \mathbf{p}]$ *is equivalent to the completion of F relative to $\mathbf{p}$.*

B.2 Proof of Proposition 1

By Theorem 8 and Theorem 9,

$$\begin{array}{rcl}
\text{SM}[F; \mathbf{p}] & \Leftrightarrow & \text{SM}[F;\ \mathbf{p} \cap pr(F)] \wedge \text{SM}[\top;\ \mathbf{p} \backslash pr(F)] \\
& \Leftrightarrow & \text{SM}[F;\ \mathbf{p} \cap pr(F)] \wedge \mathit{False}(\mathbf{p} \backslash pr(F)) \\
& \Leftrightarrow & \text{SM}[F \wedge \mathit{Choice}(pr(F) \backslash \mathbf{p})] \wedge \mathit{False}(\mathbf{p} \backslash pr(F)) \\
& \Leftrightarrow & \text{SM}[F \wedge \mathit{Choice}(pr(F) \backslash \mathbf{p}) \wedge \mathit{False}(\mathbf{p} \backslash pr(F))].
\end{array}$$

∎

B.3 Proof of Theorem 1

In the following, F is a formula, $\mathbf{p}$ is a list of distinct predicate constants $p_1, \dots, p_n$, and $\mathbf{u}$ is a list of distinct predicate variables $u_1, \dots, u_n$ of the same length as $\mathbf{p}$.

Lemma 3 *[Ferraris* et al.*, 2010, Lemma 5] Formula*

$$\mathbf{u} \leq \mathbf{p} \rightarrow (F^*(\mathbf{u}) \rightarrow F)$$

is logically valid.

Lemma 4 *If every occurrence of every predicate constant from* $\mathbf{p}$ *is strictly positive in* F*,*

$$(\mathbf{u} \leq \mathbf{p}) \rightarrow (F^*(\mathbf{u}) \leftrightarrow F(\mathbf{u}))$$

is logically valid.

Proof. By induction. We will show only the case when F is $G \rightarrow H$. The other cases are straightforward. Consider

$$F^*(\mathbf{u}) = (G^*(\mathbf{u}) \rightarrow H^*(\mathbf{u})) \wedge (G \rightarrow H).$$

Since every occurrence of predicate constants from $\mathbf{p}$ in F is strictly positive, G contains no predicate constants from $\mathbf{p}$, so that $G^*(\mathbf{u})$ is equivalent to $G(\mathbf{u})$, which is the same as G. Also by I.H., $H^*(\mathbf{u}) \leftrightarrow H(\mathbf{u})$ is logically valid. Therefore it is sufficient to prove that under the assumption $\mathbf{u} \leq \mathbf{p}$,

$$(G \rightarrow H(\mathbf{u})) \wedge (G \rightarrow H) \leftrightarrow (G \rightarrow H(\mathbf{u}))$$

is logically valid. From left to right is clear. Assume $(\mathbf{u} \leq \mathbf{p})$, $G \rightarrow H(\mathbf{u})$, and G. We get $H(\mathbf{u})$, which is equivalent to $H^*(\mathbf{u})$ by I.H. By Lemma 3, we conclude H. ▮

The proof of Theorem 1 is immediate from the following lemma, which can be proved by induction.

Lemma 5 *If* F *is canonical relative to* $\mathbf{p}$*, then formula*

$$(\mathbf{u} \leq \mathbf{p}) \wedge F \rightarrow (F^*(\mathbf{u}) \leftrightarrow F(\mathbf{u}))$$

is logically valid.

Proof.

- F is an atomic formula. Trivial.
- $F = G \wedge H$. Follows from I.H.

- $F = G \vee H$. Assume $(\mathbf{u} \leq \mathbf{p}) \wedge (G \vee H)$. Since $G \vee H$ is canonical relative to $\mathbf{p}$, every occurrence of every predicate constant from $\mathbf{p}$ is strictly positive in G or in H, so that, by Lemma 4, $G^*(\mathbf{u})$ is equivalent to $G(\mathbf{u})$, and $H^*(\mathbf{u})$ is equivalent to $H(\mathbf{u})$.

- $F = G \rightarrow H$. Assume $(\mathbf{u} \leq \mathbf{p}) \wedge (G \rightarrow H)$. It is sufficient to show

$$((G^*(\mathbf{u}) \rightarrow H^*(\mathbf{u})) \leftrightarrow (G(\mathbf{u}) \rightarrow H(\mathbf{u})). \tag{29}$$

  Since $G \rightarrow H$ is canonical relative to $\mathbf{p}$, every occurrence of every predicate constant from $\mathbf{p}$ in G is strictly positive in G, so that, by Lemma 4, $G^*(\mathbf{u})$ is equivalent to $G(\mathbf{u})$.

  - *Case 1:* $\neg G$. By Lemma 3, $\neg G^*(\mathbf{u})$. The claim follows since $\neg G^*(\mathbf{u})$ is equivalent to $\neg G(\mathbf{u})$.
  - *Case 2:* H. By I.H. $H^*(\mathbf{u})$ is equivalent to $H(\mathbf{u})$. The claim follows since $G^*(\mathbf{u})$ is equivalent to $G(\mathbf{u})$.

- $F = \forall x G$. Follows from I.H.

- $F = \exists x G$. Since every occurrence of every predicate constant from $\mathbf{p}$ in G is strictly positive in G, the claim follows from Lemma 4.

∎

B.4 Proof of Lemma 1

Step (1) turns the formula into a form such that, for every pseudo-atomic occurrence of QxG in it, every occurrence of a predicate constant p from $\mathbf{p}$ in G is positive in G.

Step (2) moves all positive occurrences of $\forall$ and all negative occurrences of $\exists$ not in the scope of pseudo-atomic occurrences to the front, so that the remaining quantifiers in M are in the scope of some pseudo-atomic occurrences.

Step (3) yields the formula of the form $\forall \mathbf{x} M$ where M is a conjunction of implications $H \rightarrow K$ such that

- H is a conjunction of atoms and formulas of the form QxG such that every occurrence of every predicate constant from $\mathbf{p}$ in G is positive in G, and

- K is a disjunction of atoms and formulas of the form QxG such that every occurrence of every predicate constant from $\mathbf{p}$ in G is positive in G.

Finally, Step (4) ensures that the final formula is of the form $\forall \mathbf{x} M$ where M is a conjunction of implications $H \rightarrow K$ such that

- H is a conjunction of atoms and formulas of the form QxG such that every occurrence of every predicate constant from $\mathbf{p}$ in G is strictly positive in G, and
- K is a disjunction of atoms and formulas of the form QxG such that every occurrence of every predicate constant from $\mathbf{p}$ in G is strictly positive in G.

Consequently, the resulting formula satisfies both conditions of canonical formulas. ∎

B.5 Proof of Theorem 3

Lemma 6 *Let F be a formula, let $\mathbf{p}$ be a list of distinct predicate constants, let G be a subformula of F and let G' be any formula that is equivalent to G. Let F' be the formula obtained from F by substituting G' for G. If the occurrence of G is in a subformula of F that is negative on $\mathbf{p}$ and the occurrence of G' is in a subformula of F' that is negative on $\mathbf{p}$, then*

$$\mathrm{SM}[F;\mathbf{p}] \leftrightarrow \mathrm{SM}[F';\mathbf{p}]$$

is logically valid.

Proof. Let F^- be the formula obtained from F by prepending $\neg\neg$ to G, and let F'^- be the formula obtained from F' by prepending $\neg\neg$ to G'. By the Theorem on Double Negations (Theorem 10), the following formulas are logically valid.

$$\begin{array}{l}\mathrm{SM}[F;\mathbf{p}] \leftrightarrow \mathrm{SM}[F^-;\mathbf{p}],\\ \mathrm{SM}[F';\mathbf{p}] \leftrightarrow \mathrm{SM}[F'^-;\mathbf{p}].\end{array}$$

From Lemma 2, it follows that

$$(G \leftrightarrow G') \rightarrow ((F^-)^*(\mathbf{u}) \leftrightarrow (F'^-)^*(\mathbf{u}))$$

is logically valid, where $\mathbf{u}$ is a list of predicate variables corresponding to $\mathbf{p}$. Consequently,

$$\mathrm{SM}[F^-;\mathbf{p}] \leftrightarrow \mathrm{SM}[F'^-;\mathbf{p}]$$

is logically valid. ∎

Proof of Theorem 3. In formula

$$\mathrm{SM}[F' \wedge \forall\mathbf{x}(G(\mathbf{x}) \rightarrow q(\mathbf{x}));\mathbf{p},q], \tag{30}$$

clearly, F' is negative on q and $\forall\mathbf{x}(G(\mathbf{x}) \rightarrow q(\mathbf{x}))$ is negative on $\mathbf{p}$. Let H be any subformula of F that is negative on $\mathbf{p}$ and contains the occurrence of $G(\mathbf{x})$. Consider two cases.

- Case 1: the occurrence of $G(\mathbf{x})$ in H is not strictly positive. Thus the dependency graph of $F' \wedge \forall\mathbf{x}(G(\mathbf{x}) \rightarrow q(x))$ relative to $\{\mathbf{p}, q\}$ has no incoming edges into q.
- Case 2: the occurrence of $G(\mathbf{x})$ in H is strictly positive. Since H is negative on $\mathbf{p}$, $G(\mathbf{x})$ is negative on $\mathbf{p}$ as well, so that the dependency graph of $F' \wedge \forall\mathbf{x}(G(\mathbf{x}) \rightarrow q(\mathbf{x}))$ relative to $\{\mathbf{p}, q\}$ has no outgoing edges from q.

Therefore, every strongly connected component in the dependency graph belongs to either $\mathbf{p}$ or $\{q\}$. Consequently, by Theorem 9, (30) is equivalent to

$$\mathrm{SM}[F'; \mathbf{p}] \wedge \mathrm{SM}[\forall\mathbf{x}(G(\mathbf{x}) \rightarrow q(\mathbf{x})); q] \tag{31}$$

Since $G(\mathbf{x})$ is negative on q, formula $\forall\mathbf{x}(G(\mathbf{x}) \rightarrow q(\mathbf{x}))$ is tight on $\{q\}$. By Theorem 11, (31) is equivalent to

$$\mathrm{SM}[F'; \mathbf{p}] \wedge \forall\mathbf{x}(G(\mathbf{x}) \leftrightarrow q(\mathbf{x})). \tag{32}$$

From Lemma 6, it follows that (32) is equivalent to

$$\mathrm{SM}[F; \mathbf{p}] \wedge \forall\mathbf{x}(G(\mathbf{x}) \leftrightarrow q(\mathbf{x})).$$

Consequently, the claim follows. ∎

B.6 Proof of Theorem 4

It is clear that the algorithm terminates and yields a quantifier-free formula K. We will prove that $\mathrm{SM}[F; \mathbf{p}] \Leftrightarrow_\sigma \mathrm{SM}[\forall\mathbf{x}K; \mathbf{p} \cup \mathbf{q}]$ where $\mathbf{x}$ is the list of all (free) variables of K.

Let F^- be the formula obtained from the initial formula F by prepending double negations in front of every maximal strictly positive occurrence of formulas of the form $\exists y G(\mathbf{x}, y)$. Since F is almost universal relative to $\mathbf{p}$, such an occurrence is in a subformula of F that is negative on $\mathbf{p}$. Thus by the Theorem on Double Negations (Theorem 10), $\mathrm{SM}[F; \mathbf{p}]$ is equivalent to $\mathrm{SM}[F^-; \mathbf{p}]$. Note that F^- contains no strictly positive occurrence of formulas of the form $\exists y G(\mathbf{x}, y)$.

For each iteration, let us assume that the formula before the iteration is

$$H_0 \wedge \cdots \wedge H_n$$

where H_0 is transformed from F^- by the previous iterations, and each H_i $(i > 0)$ is a formula of the form $G(\mathbf{x}, y) \rightarrow p_G(\mathbf{x})$ that is introduced by Step (b). Initially H_0 is F^- and $n = 0$. Let $\mathbf{r}_0$ be $\mathbf{p}$, and let $\mathbf{r}_i$ be each p_G for H_i $(i > 0)$. By induction we can prove that

(i) every positive occurrence of formulas of the form $\exists y G(\mathbf{x}, y)$ in H_i is not strictly positive, and is in a subformula of H_i that is negative on $\mathbf{r}_i$;

(ii) every negative occurrence of formulas of the form $\forall y G(\mathbf{x}, y)$ in H_i is in a subformula of H_i that is negative on $\mathbf{r}_i$.

We will prove that if Step (a) or Step (c) is applied to turn H_k into H'_k, then

$$\mathrm{SM}[\forall \mathbf{x}_0 H_0; \mathbf{r}_0] \wedge \cdots \wedge \mathrm{SM}[\forall \mathbf{x}_n H_n; \mathbf{r}_n] \tag{33}$$

is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}'_0 H'_0; \mathbf{r}_0] \wedge \cdots \wedge \mathrm{SM}[\forall \mathbf{x}'_n H'_n; \mathbf{r}_n] \tag{34}$$

where $H'_j = H_j$ for all j different from k, and $\mathbf{x}_i$ ($i \geq 0$) is the list of all free variables of H_i, and $\mathbf{x}'_i$ ($i \geq 0$) is the list of all free variables of H'_i.

Indeed, Step (a) is a part of prenex form conversion, which preserves strong equivalence (Theorem 2). So it is clear that (33) is equivalent to (34).

When Step (c) is applied to turn (33) into (34), since $\forall y H(\mathbf{x}, y)$ is in a subformula of H_k that is negative on $\mathbf{r}_k$, the equivalence between (33) and (34) follows from Lemma 6.

When Step (b) is applied to turn H_k into H'_k and introduces a new conjunctive term H'_{n+1}, formula (33) is $(\sigma, \mathbf{r}_1, \ldots, \mathbf{r}_n)$-equivalent to

$$\mathrm{SM}[\forall \mathbf{x}'_0 H'_0; \mathbf{r}_0] \wedge \cdots \wedge \mathrm{SM}[\forall \mathbf{x}'_n H'_n; \mathbf{r}_n] \wedge \mathrm{SM}[\forall \mathbf{x}'_{n+1} H'_{n+1}; \mathbf{r}_{n+1}] \tag{35}$$

by Theorem 3 due to condition (i).

Let

$$H''_0 \wedge \cdots \wedge H''_m \tag{36}$$

be the final quantifier-free formula where H''_0 is transformed from F^-. By the induction, it follows that $\mathrm{SM}[F; \mathbf{p}]$ is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}''_0 H''_0;\ \mathbf{r}_0] \wedge \cdots \wedge \mathrm{SM}[\forall \mathbf{x}''_m H''_m;\ \mathbf{r}_m]. \tag{37}$$

modulo σ, where each $\mathbf{x}''_i$ ($0 \leq i \leq m$) is the list of all free variables of H''_i.

Since every non-strictly positive occurrence of new predicate $\mathbf{r}_i$ ($i > 0$) in any H''_j ($0 \leq j \leq m$) is positive, there is no incoming edge into $\mathbf{r}_i$ in the dependency graph of (36) relative to $\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_m$. Consequently, every strongly connected component of the dependency graph belongs to one of $\mathbf{r}_i$ ($i \geq 0$). Moreover, it is clear that each H''_i ($i \geq 0$) is negative on every $\mathbf{r}_j$ for $j \neq i$. (In the case of H''_0, recall that the occurrence of $\mathbf{r}_j$ for any $j > 0$ is not strictly positive since F^-, from which H''_0 is obtained, contains no strictly positive occurrence of formulas of the form $\exists y G(\mathbf{x}, y)$. Thus by the splitting theorem (Theorem 9), formula (37) is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}''_0 H''_0 \wedge \cdots \wedge \forall \mathbf{x}''_m H''_m;\ \mathbf{r}_0 \cup \cdots \cup \mathbf{r}_m]. \tag{38}$$

∎

B.7 Proof of Theorem 5

We use the notations introduced in the proof of Theorem 4. By Theorem 4, $\mathrm{SM}[F; \mathbf{p}]$ is σ-equivalent to (38) and, by Theorem 8, (38) is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}_0'' H_0'' \wedge \cdots \wedge \forall \mathbf{x}_m'' H_m'' \wedge \mathit{Choice}(\sigma^{pred} \setminus \mathbf{p});\ \sigma^{pred} \cup \mathbf{r}_1 \cup \cdots \cup \mathbf{r}_m] \tag{39}$$

($\mathbf{r}_0$ is $\mathbf{p}$) where σ^{pred} is the set of all predicate constants in signature σ. It follows from Proposition 3 from [Cabalar *et al.*, 2005] that (39) is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}_0'' H_0''' \wedge \cdots \wedge \forall \mathbf{x}_m'' H_m''' \wedge \mathit{Choice}(\sigma^{pred} \setminus \mathbf{p});\ \sigma^{pred} \cup \mathbf{r}_1 \cup \cdots \cup \mathbf{r}_m] \tag{40}$$

where H_i''' is obtained from H_i'' by applying the translation from [Cabalar *et al.*, 2005, Section 3] that turns a quantifier-free formula into a set of rules. It is easy to see that Π^{FOL} is the same as the formula

$$\forall \mathbf{x}_0'' H_0''' \wedge \cdots \wedge \forall \mathbf{x}_m'' H_m''' \wedge \mathit{Choice}(\sigma^{pred} \setminus \mathbf{p})$$

and $\sigma^{pred} \cup \mathbf{r}_1 \cup \cdots \cup \mathbf{r}_m$ is the same as $\mathbf{p} \cup \mathit{pr}(\Pi^{FOL})$, so that (40) can be written as

$$\mathrm{SM}[\Pi^{FOL};\ \mathbf{p} \cup \mathit{pr}(\Pi^{FOL})],$$

which is equivalent to

$$\mathrm{SM}[\Pi^{FOL} \wedge \mathit{False}(\mathbf{p} \setminus \mathit{pr}(\Pi^{FOL}))].$$

by Proposition 1. ∎

B.8 Proof of Theorem 6

Proof. *Between (a) and (b):* Follows immediately from Theorem 1.

Between (b) and (c): Note first that Ξ is equivalent to $\mathrm{SM}[\Xi; \emptyset]$. Since

- every strongly connected component in the dependency graph of $\Sigma \wedge \Delta$ relative to $\{I, T, R, H\}$ either belongs to $\{I, T, R\}$ or $\{H\}$,
- Σ is negative on $\{H\}$, and
- Δ is negative on $\{I, T, R\}$,

it follows from Theorem 9 that (b) is equivalent to

$$\mathrm{SM}[\Sigma \wedge \Delta;\ I, T, R, H] \wedge \mathrm{SM}[\Theta;\ \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \wedge \mathrm{SM}[\Xi;\ \emptyset]$$

Similarly, applying the Theorem 9 repeatedly, we can show that the above formula is equivalent to (c).

Between (c) and (d): By Theorem 8. ∎

B.9 Proof of Theorem 7

Assume that T is

$$\begin{aligned}&\mathrm{CIRC}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{CIRC}[\Delta; \mathit{Happens}]\\ &\wedge\ \mathrm{CIRC}[\Theta; \mathit{Ab}_1, \dots, \mathit{Ab}_n] \wedge \Xi,\end{aligned}$$

which is equivalent to

$$\begin{aligned}&\mathrm{SM}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{SM}[\Delta; \mathit{Happens}]\\ &\wedge\ \mathrm{SM}[\Theta; \mathit{Ab}_1, \dots, \mathit{Ab}_n] \wedge \Xi\end{aligned} \tag{41}$$

by Theorem 6.

Let Ξ_{def} be the set of all definitions (25) in Ξ, and let Ξ' be the formula obtained from Ξ by applying Step 1. By Theorem 11, it follows that each formula (25) in Ξ_{def} is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}(G' \rightarrow p(\mathbf{x}));\ p]$$

where G' is as described in Step 1. Consequently, (41) is equivalent to

$$\begin{aligned}&\mathrm{SM}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{SM}[\Delta; \mathit{Happens}]\\ &\wedge\ \mathrm{SM}[\Theta; \mathit{Ab}_1, \dots, \mathit{Ab}_n] \wedge \textstyle\bigwedge_{(25)\in\Xi_{def}} \mathrm{SM}[\forall \mathbf{x}(G' \rightarrow p(\mathbf{x}));\ p] \wedge \Xi''\end{aligned} \tag{42}$$

where Ξ'' is the conjunction of all the axioms in Ξ' other than the ones obtained from definitional axioms (25).

Applying Theorem 9 repeatedly, it follows that (42) is equivalent to

$$\begin{aligned}&\mathrm{SM}[\Sigma \wedge \Delta \wedge \Theta \wedge \Xi'' \wedge \textstyle\bigwedge_{(25)\in\Xi_{def}} \forall \mathbf{x}(G' \rightarrow p(\mathbf{x}));\\ &\mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}, \mathit{Happens}, \mathit{Ab}_1, \dots, \mathit{Ab}_n, \mathbf{p}]\ .\end{aligned} \tag{43}$$

According to the syntax of the event calculus reviewed in Section 5.1,

- every positive occurrence of a formula of the form $\exists y G(y)$ in (43) is contained in a subformula that is negative on $\{\mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}, \mathit{Happens}, \mathit{Ab}_1, \dots, \mathit{Ab}_n, \mathbf{p}\}$, and
- there are no negative occurrences of any formula of the form $\forall y G(y)$ in (43).

Consequently, the statement of the theorem follows from Theorem 5. ∎

References

[Cabalar and Ferraris, 2007] Pedro Cabalar and Paolo Ferraris. Propositional theories are strongly equivalent to logic programs. *TPLP*, 7(6):745–759, 2007.

[Cabalar *et al.*, 2005] Pedro Cabalar, David Pearce, and Agustin Valverde. Reducing propositional theories in equilibrium logic to logic programs. In *Proceedings of Portuguese Conference on Artificial Intelligence (EPIA)*, pages 4–17, 2005.

[Clark, 1978] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[Doherty *et al.*, 1998] Patrick Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnström. TAL: Temporal action logics language specification and tutorial[20]. *Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841*, 3(015), 1998.

[Doğandağ *et al.*, 2004] Semra Doğandağ, Paolo Ferraris, and Vladimir Lifschitz. Almost definite causal theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 74–86, 2004.

[Erdem and Lifschitz, 2003] Esra Erdem and Vladimir Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3:499–518, 2003.

[Ferraris *et al.*, 2007] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A new perspective on stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 372–379, 2007.

[Ferraris *et al.*, 2009] Paolo Ferraris, Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. Symmetric splitting in the general theory of stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 797–803, 2009.

[Ferraris *et al.*, 2010] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. Stable models and circumscription. *Artificial Intelligence*, 2010. To appear.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

[Janhunen and Oikarinen, 2004] Tomi Janhunen and Emilia Oikarinen. Capturing parallel circumscription with disjunctive logic programs. In *Proc. of 9th European Conference in Logics in Artificial Intelligence (JELIA-04)*, pages 134–146, 2004.

[20]`http://www.ep.liu.se/ea/cis/1998/015/`

[Kim *et al.*, 2009] Tae-Won Kim, Joohyung Lee, and Ravi Palla. Circumscriptive event calculus as answer set programming. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 823–829, 2009.

[Kowalski and Sergot, 1986] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[Lee and Lin, 2004] Joohyung Lee and Fangzhen Lin. Loop formulas for circumscription. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 281–286, 2004.

[Lee and Palla, 2007] Joohyung Lee and Ravi Palla. Yet another proof of the strong equivalence between propositional theories and logic programs. In *Working Notes of the Workshop on Correspondence and Equivalence for Nonmonotonic Theories*, 2007.

[Lee and Palla, 2009] Joohyung Lee and Ravi Palla. System F2LP – computing answer sets of first-order formulas. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 515–521, 2009.

[Lee and Palla, 2010] Joohyung Lee and Ravi Palla. Situation calculus as answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 309–314, 2010.

[Lee *et al.*, 2008] Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. A reductive semantics for counting and choice in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 472–479, 2008.

[Lifschitz *et al.*, 1999] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.

[Lifschitz, 1994] Vladimir Lifschitz. Circumscription. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in AI and Logic Programming*, volume 3, pages 298–352. Oxford University Press, 1994.

[Lifschitz, 2008a] Vladimir Lifschitz. Twelve definitions of a stable model. In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 37–51, 2008.

[Lifschitz, 2008b] Vladimir Lifschitz. What is answer set programming? In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1594–1597. MIT Press, 2008.

[Lin and Shoham, 1992] Fangzhen Lin and Yoav Shoham. A logic of knowledge and justified assumptions. *Artificial Intelligence*, 57:271–289, 1992.

[Lin and Zhou, 2007] Fangzhen Lin and Yi Zhou. From answer set logic programming to circumscription via logic of GK. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[Lin, 1991] Fangzhen Lin. *A Study of Nonmonotonic Reasoning.* PhD thesis, Stanford University, 1991.

[Lin, 1995] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1985–1991, 1995.

[Marek and Truszczyński, 1999] Victor Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

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

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

[Miller and Shanahan, 1999] Rob Miller and Murray Shanahan. The event calculus in classical logic - alternative axiomatisations. *Electron. Trans. Artif. Intell.*, 3(A):77–105, 1999.

[Mueller, 2004a] Erik T. Mueller. Event calculus reasoning through satisfiability. *Journal of Logic and Computation*, 14(5):703–730, 2004.

[Mueller, 2004b] Erik T. Mueller. A tool for satisfiability-based commonsense reasoning in the event calculus. In Valerie Barr and Zdravko Markov, editors, *FLAIRS Conference*. AAAI Press, 2004.

[Mueller, 2006] Erik Mueller. *Commonsense reasoning.* Elsevier, 2006.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[Pearce and Valverde, 2005] David Pearce and Agustin Valverde. A first order nonmonotonic extension of constructive logic. *Studia Logica*, 80:323–348, 2005.

[Shanahan and Witkowski, 2004] Murray Shanahan and Mark Witkowski. Event calculus planning through satisfiability. *J. Log. Comput.*, 14(5):731–745, 2004.

[Shanahan, 1995] Murray Shanahan. A circumscriptive calculus of events. *Artif. Intell.*, 77(2):249–284, 1995.

[Shanahan, 1997] Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

[Shanahan, 1999] Murray Shanahan. The event calculus explained. In *Artificial Intelligence Today*, LNCS 1600, pages 409–430. Springer, 1999.