Circumscriptive Event Calculus as Answer Set Programming

Tae-Won Kim, Joohyung Lee, and Ravi Palla
Computer Science and Engineering
Arizona State University
Tempe, AZ 85281, USA
{Tae-Won.Kim, joolee, Ravi.Palla}@asu.edu

Abstract

Recently, Ferraris, Lee and Lifschitz presented a general definition of a stable model that is similar to the definition of circumscription, and can even be characterized in terms of circumscription. In this paper, we show the opposite direction, which is, how to turn circumscription into the general stable model semantics, and based on this, how to turn circumscriptive event calculus into answer set programs. The reformulation of the event calculus in answer set programming allows answer set solvers to be applied to event calculus reasoning, handling more expressive reasoning tasks than the current SAT-based approach. Our experiments also show clear computational advantages of the answer set programming approach.

1 Introduction

The stable model semantics is the mathematical basis of answer set programming [Marek and Truszczyński, 1999; Niemelä, 1999; Lifschitz, 2008]. Recently, Ferraris, Lee and Lifschitz [2007] presented a general definition of a stable model that applies to the syntax of arbitrary first-order sentences, under which logic programs are viewed as a special class of first-order sentences. The new semantics is similar to the definition of circumscription [McCarthy, 1980; 1986], and can even be characterized in terms of circumscription [Ferraris *et al.*, 2007, Corollary 1]. Essentially the same characterization was also independently given in [Lin and Zhou, 2007].

The opposite direction, turning (parallel) circumscription into the stable model semantics, was shown in [Lee and Lin, 2004; Janhunen and Oikarinen, 2004] limited to the propositional case. The first contribution of this paper is a generalization of the result from [Lee and Lin, 2004]: turning first-order circumscription into the general stable model semantics. Such close relationship between the two formalisms is curious since they have formed rather disparate traditions in knowledge representation. In particular, they have provided different solutions to the frame problem. A group of action formalisms, such as circumscriptive event calculus [Shanahan, 1995] and temporal action logics [Doherty *et al.*, 1998], is based on classical logic, using circumscription to handle the frame problem. On the other hand, the solution provided by answer set programming (ASP), that is carried over to high level action language $\mathcal{A}$ [Gelfond and Lifschitz, 1998] and many of its descendants that are based on ASP, uses both default negation (*not*) and strong negation ($\neg$)—the idea of which is closely related to Reiter's default logic solution [Reiter, 1980]. 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 under the classical logic setting via circumscription [Shanahan, 1995; Miller and Shanahan, 1999].

In this paper we show how to embed circumscriptive event calculus into the general language of stable models, and furthermore into answer set programs under certain conditions. The reformulation of the event calculus in the stable model semantics here can be viewed as turning back the event calculus to the logic program tradition, in the modern form of answer set programming. In contrast to the SAT-based approaches from [Shanahan and Witkowski, 2004; Mueller, 2004a] which turn circumscription into completion and hence cannot handle certain recursive axioms of the event calculus, we show that the ASP approach can compute the *full* version of the event calculus assuming that the domain is given and finite. Our work shows that the new stable model semantics is a nonmonotonic formalism as general as circumscription, with the unique advantage of having efficient ASP solvers as computational tools.

Our work is motivated by Mueller's work that is available on the webpage

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

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" and no formal justification is provided there.

2 Review of the Event Calculus

Here we follow the extensive syntax of circumscriptive event calculus described in [Mueller, 2006, Chapter 2].

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:

- A comparison ($\tau_1 < \tau_2$, $\tau_1 \leq \tau_2$, $\tau_1 \geq \tau_2$, $\tau_1 > \tau_2$, $\tau_1 = \tau_2$, $\tau_1 \neq \tau_2$) for terms τ_1, τ_2 is a condition;
- If f is a fluent term and t is a timepoint term, then $\mathit{HoldsAt}(f,t)$ and $\neg\mathit{HoldsAt}(f,t)$ are conditions;
- If γ_1 and γ_2 are conditions, then $\gamma_1 \wedge \gamma_2$ and $\gamma_1 \vee \gamma_2$ are conditions;
- 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 understand formula $F \leftrightarrow G$ as shorthand for $(F \rightarrow G) \wedge (G \rightarrow F)$; formula $\top$ as shorthand for $\bot \rightarrow \bot$; formula $\neg F$ as shorthand for $F \rightarrow \bot$.

An event calculus domain description is defined as [1]

$$\text{CIRC}[\Sigma\ ;\ \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \\ \wedge\ \text{CIRC}[\Delta\ ;\ \mathit{Happens}] \wedge \Xi. \tag{1}$$

where

- Σ is the conjunction of universal closures of axioms of the form
  $\gamma \rightarrow \mathit{Initiates}(e,f,t)$
  $\gamma \rightarrow \mathit{Terminates}(e,f,t)$
  $\gamma \rightarrow \mathit{Releases}(e,f,t)$
  $\gamma \wedge \pi_1(e,f_1,t) \rightarrow \pi_2(e,f_2,t)$ ("effect constraint")
  $\gamma \wedge [\neg]Happens(e_1,t) \wedge \cdots \wedge [\neg]Happens(e_n,t) \rightarrow \mathit{Initiates}(e,f,t)$
  $\gamma \wedge [\neg]Happens(e_1,t) \wedge \cdots \wedge [\neg]Happens(e_n,t) \rightarrow \mathit{Terminates}(e,f,t)$
  where each of π_1 and π_2 is either *Initiates* or *Terminates*;
- Δ is the conjunction of universal closures of *temporal ordering formulas* (comparisons between timepoint terms) and axioms of the form
  $\gamma \rightarrow \mathit{Happens}(e,t)$
  $\sigma(e,t) \wedge \pi_1(e_1,t) \wedge \cdots \wedge \pi_n(e_n,t) \rightarrow \mathit{Happens}(e,t)$ ("causal constraints")
  $\mathit{Happens}(e,t) \rightarrow \mathit{Happens}(e_1,t) \vee \cdots \vee \mathit{Happens}(e_n,t)$ ("disjunctive event axiom")
  where σ is *Started* or *Stopped* and each π_j $(1 \leq j \leq n)$ is either *Initiated* or *Terminated*, which are defined as follows:

$$\mathit{Started}(f,t) \overset{def}{\leftrightarrow} (\mathit{HoldsAt}(f,t) \vee \exists e(\mathit{Happens}(e,t) \wedge \mathit{Initiates}(e,f,t))) \tag{CC_1}$$

$$\mathit{Stopped}(f,t) \overset{def}{\leftrightarrow} (\neg\mathit{HoldsAt}(f,t) \vee \exists e(\mathit{Happens}(e,t) \wedge \mathit{Terminates}(e,f,t))) \tag{CC_2}$$

$$\mathit{Initiated}(f,t) \overset{def}{\leftrightarrow} (\mathit{Started}(f,t) \wedge \neg\exists e(\mathit{Happens}(e,t) \wedge \mathit{Terminates}(e,f,t))) \tag{CC_3}$$

$$\mathit{Terminated}(f,t) \overset{def}{\leftrightarrow} (\mathit{Stopped}(f,t) \wedge \neg\exists e(\mathit{Happens}(e,t) \wedge \mathit{Initiates}(e,f,t))) \tag{CC_4}$$

- Ξ is a conjunction of first-order sentences (outside the scope of circumscription) including unique name axioms, state constraints, and one of the two axiomatizations of the event calculus, *EC* and *DEC* axioms [Mueller, 2006].

As listed, circumscriptive event calculus descriptions look different from logic programs. The former may contain existential quantifiers; only some parts of the description are circumscribed on selected lists of predicates. Nonetheless, we will see that the differences are not essential.

3 Review of the General Stable Model Semantics

Under the new definition of stable models presented in [Ferraris *et al.*, 2007] that is applicable to arbitrary first-order sentences, a logic program is identified with a universal formula, called the *FOL-representation*. For example, the FOL-representation of the program

$$p(a) \qquad q(b) \qquad r(x) \leftarrow p(x), \mathit{not}\ q(x)$$

is

$$p(a) \wedge q(b) \wedge \forall x((p(x) \wedge \neg q(x)) \rightarrow r(x)). \tag{2}$$

The review here follows [Ferraris *et al.*, 2010], a journal version of [Ferraris *et al.*, 2007], in which the stable model operator was extended to distinguish between intensional and extensional predicates. Let $\mathbf{p}$ be a list of distinct predicate constants $p_1, \ldots, p_n$, and let $\mathbf{u}$ be a list of distinct predicate variables $u_1, \ldots, u_n$ 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 $i = 1, \ldots n$ 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 any first-order sentence F, expression $\text{SM}[F;\mathbf{p}]$ stands for the second-order sentence[2]

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

where $\mathbf{p}$ is the list $p_1, \ldots, p_n$ of predicate constants that are called *intensional*, $\mathbf{u}$ is a list $u_1, \ldots, u_n$ of distinct predicate variables corresponding to $\mathbf{p}$, and $F^*(\mathbf{u})$ is defined recursively:

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

$\text{SM}[F]$ defined in [Ferraris *et al.*, 2007] is identical to $\text{SM}[F;\mathbf{p}]$ where intensional predicate constants $\mathbf{p}$ range over all predicate constants occurring in F. Let $\sigma(F)$ be the signature consisting of the object, function and predicate constants occurring in F. According to [Ferraris *et al.*, 2007], an interpretation of $\sigma(F)$ that satisfies $\text{SM}[F;\mathbf{p}]$ is called a *stable model* of F, where $\mathbf{p}$ is the list of all predicate constants

[1] The syntax from [Mueller, 2006, Chapter 2] allows $\text{CIRC}[\Theta\ ;\ \mathit{Ab}_1, \ldots, \mathit{Ab}_n]$ where Θ is a conjunction of cancellation axioms, which we omit here for simplicity of presentation.

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

in $\sigma(F)$. If F contains at least one object constant, an Herbrand stable model of 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. It turns out that this definition, applied to the syntax of logic programs, is equivalent to the traditional definition of answer sets based on grounding and fixpoint construction [Ferraris *et al.*, 2007].

The fact that intensional predicates $\mathbf{p}$ in SM$[F;\mathbf{p}]$ is allowed not to include some predicates occurring in the formula is not essential in view of the following proposition. By $pr(F)$ we denote the list of all predicate constants occurring in F; by *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.

Proposition 1 *[Ferraris* et al.*, 2010, Theorem 2] If* $\mathbf{p}$ *is a subset of* $pr(F)$*, then*

$$\text{SM}[F;\mathbf{p}] \leftrightarrow \text{SM}[F \wedge \mathit{Choice}(pr(F)\backslash\mathbf{p});\ pr(F)]$$

is logically valid.

However, it will be convenient to describe our main results in this paper by allowing $\mathbf{p}$ to be partial.

4 Turning Circumscription and Event Calculus Descriptions into SM

We say that an occurrence of a predicate constant in a formula F is *strictly positive* if that occurrence is not in the antecedent of any implication. For instance, in $(p \to q) \to r$, only r has a strictly positive occurrence.

For any set $\mathbf{p}$ of predicate constants and any formulas G and H, we call implication $G \to H$ *canonical (relative to* $\mathbf{p}$*)* if

- every occurrence of every predicate constant from $\mathbf{p}$ in G is strictly positive in G, and
- every occurrence of every predicate constant from $\mathbf{p}$ in H is strictly positive in H.

For instance, $p(x) \to q(x)$ is a canonical implication relative to $\{p, q\}$, while its contraposition $\neg q(x) \to \neg p(x)$ is not. Also $\exists x\ p(x) \to \exists x\ q(x)$ is a canonical implication relative to $\{p, q\}$.

By a *canonical theory (relative to* $\mathbf{p}$*)* we mean the conjunction of sentences of the form

$$\forall\mathbf{x}(G \to H)$$

where $\mathbf{x}$ is a list of variables (possibly empty), and $G \to H$ is a canonical implication (relative to $\mathbf{p}$). For instance,

$$(\top \to p(a)) \wedge (\exists x\, p(x) \to \exists x\, q(x)) \tag{3}$$

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

$$(\top \to p(a, a)) \wedge \exists x(p(x, a) \to p(b, x)) \tag{4}$$

is not a canonical theory relative to $\{p\}$ (it becomes canonical if $\exists$ is replaced with $\forall$). Note that any quantifier-free formula (and universal formula) can be equivalently rewritten as a conjunction of canonical implications. One way is by converting to a conjunctive normal form and then rewriting each clause as a canonical implication. An arbitrary sentence that contains existential quantifiers can still be turned into a canonical theory using Skolemization, which preserves satisfiability.

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

Proposition 2 *For any canonical theory* F *relative to* $\mathbf{p}$*,*

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

is logically valid.

For instance, for (3) that is canonical relative to $\{p, q\}$, formulas CIRC$[(3); p, q]$ and SM$[(3); p, q]$ are equivalent to each other. Also any sentence F is clearly equivalent to a canonical theory 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, for (4) that is not canonical relative to p, CIRC$[(4); p]$ is not equivalent to SM$[(4); p]$: the only Herbrand model of the signature $\{p/2, a, b\}$ that satisfies CIRC$[(4); p]$ is $\{p(a, a)\}$ while the Herbrand models of the same signature that satisfy SM$[(4); p]$ are $\{p(a, a)\}$ and $\{p(a, a), p(b, a)\}$.

It turns out that canonical theories cover a wide range of formalisms based on circumscription. Indeed, in the syntax of the event calculus described in Section 2, all axioms in Σ are already canonical relative to $\{$*Initiates*,*Terminates*,*Releases*$\}$; all axioms in Δ are already canonical relative to $\{$*Happens*$\}$. Though not described here in detail, one can check that descriptions in temporal action logics are canonical as well.

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 it contains no strictly positive occurrences of *Initiates*, *Terminates*, *Releases*, *Happens* (by prepending $\neg\neg$ to such occurrences).

Theorem 1 *Given an event calculus description (1), let* $\mathbf{p}$ *be the set of all predicate constants (other than equality and comparisons) occurring in it. The following theories are equivalent to each other:*[3]

(a) CIRC$[\Sigma; I, T, R] \wedge$ CIRC$[\Delta; H] \wedge \Xi$

(b) SM$[\Sigma; I, T, R] \wedge$ SM$[\Delta; H] \wedge \Xi$;

(c) SM$[\Sigma \wedge \Delta \wedge \Xi; I, T, R, H]$;

(d) SM$[\Sigma \wedge \Delta \wedge \Xi \wedge \mathit{Choice}(\mathbf{p} \setminus \{I, T, R, H\}); \mathbf{p}]$.

The equivalence between (a) and (b) is immediate from Proposition 2; the equivalence between (b) and (c) follows from the general splitting theorem for the stable model semantics [Ferraris *et al.*, 2009]; the equivalence between (c) and (d) is immediate from Proposition 1.

5 Turning Event Calculus Descriptions into Answer Set Programs

Like answer set programs, the intensional predicates of formula (d) in Theorem 1 are all the predicates that occur in

[3] Due to lack of space, we abbreviate the names of circumscribed predicates.

the event calculus description. On the other hand, the description contains formulas that may not be in the rule form, and may contain existential quantifiers, which are not allowed in answer set programs. In this section we provide a translation of the event calculus into answer set programs so that ASP solvers can be applied for event calculus reasoning. To facilitate the discussion, we introduce the notion of RASPL-1^M programs. Syntactically similar codes are accepted by LPARSE[4] —the front-end of several ASP solvers, such as SMODELS[5], CMODELS[6], CLASP, and CLINGO[7].

5.1 RASPL-1^M Programs

The definition of stable models reviewed in Section 3 can be easily extended to many-sorted first-order languages, similar to the extension of circumscription to many-sorted first-order languages (Section 2.4 of [Lifschitz, 1994]). We define RASPL-1^M programs as a special class of sentences under this extension, which are essentially a many-sorted extension of RASPL-1 programs from [Lee *et al.*, 2008]. We assume that the underlying signature contains an integer sort and contains several built-in symbols, such as integer constants, built-in arithmetic functions, such as $+$, $-$, and comparison operators, such as $<$, $\leq$, $>$, $\geq$. Since we do not need counting aggregates in this paper, for simplicity, we will assume that every "aggregate expression" is an atom or a negated atom. That is, a *rule* is an expression of the form

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

($0 \leq k \leq m \leq n \leq p$), where each A_i is an atom, possibly equality or comparison. A *program* is a finite list of rules.

The "choice rule" of the form $\{A\} \leftarrow$ *Body* where A is an atom, stands for $A \leftarrow$ *Body*, *not not* A.

The semantics of a RASPL-1^M program is understood by turning it into its corresponding many-sorted FOL-representation, as in RASPL-1. The integer constants and built-in symbols are evaluated in the standard way, and we consider only those "standard" interpretations. The answer sets of a RASPL-1^M program are the Herbrand interpretations of the signature consisting of the object, function and predicate constants occurring in the program, that satisfy SM$[F;\mathbf{p}]$, where F is the FOL-representation of the program and $\mathbf{p}$ is the list of predicate constants occurring in F.

5.2 Turning Event Calculus Descriptions to RASPL-1^M Programs

The following procedure turns an event calculus description into a RASPL-1^M program. As before, we assume that Ξ was already equivalently rewritten so that it contains no strictly positive occurrences of *Initiates*, *Terminates*, *Releases*, *Happens* (by prepending $\neg\neg$ to such occurrences).

Definition 1 (Translation EC2ASP**)** *1. Rewrite all the definitional axioms of the form*

$$\forall\mathbf{x}(p(\mathbf{x}) \overset{def}{\leftrightarrow} G) \tag{5}$$

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

2. For each axiom that contains existential quantifiers, repeat the following until there are no existential quantifiers:

(a) Replace maximal negative occurrences of $\exists y G(y)$ *in the axiom by* $G(z)$ *where* z *is a new variable.*

(b) Replace maximal positive occurrences of $\exists y G(\mathbf{x}, y)$ *in the axiom where* $\mathbf{x}$ *is the list of all free variables of* $\exists y G(\mathbf{x}, y)$*, by the formula* $\neg\neg p_G(\mathbf{x})$ *where* p_G *is a new predicate constant, and add the axiom*

$$\forall\mathbf{x}y(G(\mathbf{x},y) \rightarrow p_G(\mathbf{x})). \tag{6}$$

3. Add choice formulas $\forall\mathbf{x}(p(\mathbf{x}) \vee \neg p(\mathbf{x}))$ *for all the predicate constants* p *except for those in* $\{\mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}, \mathit{Happens}\} \cup \mathbf{p_1} \cup \mathbf{p_2}$ *where*

- $\mathbf{p_1}$ *is the set of all predicate constants* p *considered in Step 1.*
- $\mathbf{p_2}$ *is the set of all new predicate constants* p_G *introduced in Step 2.*

4. Identifying the formulas with programs with nested expressions *[Lifschitz* et al.*, 1999], apply the conversion from [Lifschitz* et al.*, 1999] that turns programs with nested expressions into disjunctive logic programs.*

For example, consider *DEC5* axiom:

$$(\mathit{HoldsAt}(f,t) \wedge \neg \mathit{ReleasedAt}(f,t{+}1) \wedge \neg\exists e(\mathit{Happens}(e,t) \wedge \mathit{Terminates}(e,f,t))) \rightarrow \mathit{HoldsAt}(f,t{+}1). \tag{7}$$

In order to eliminate the positive occurrence of the existential quantifier in the formula, we apply Step 2(b), introducing the formula

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

and replacing (7) with [8]

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

Step 4 turns these formulas into rules

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

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

Due to lack of space, instead of presenting a formal proof, we attempt to give the idea of the translation. Step 1 can be dropped without affecting the correctness, but keeping it yields a simpler program. Step 2 eliminates existential quantifiers. Step 2(a) is one of the steps in prenex normal form conversion. Eliminating positive occurrences of existential quantifiers using new predicates as in Step 2(b) works only if G contains no strictly positive occurrences of predicates to minimize, which is the case with the syntax of the event calculus (Section 2). The following proposition justifies this step of eliminating existential quantifiers.

[4] *http://www.tcs.hut.fi/Software/smodels*

[5] *http://www.tcs.hut.fi/Software/smodels* .

[6] *http://www.cs.utexas.edu/users/tag/cmodels.html* .

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

[8] The proof of Theorem 2 uses the fact that $\neg\neg$ is put in front of $p_G(\mathbf{x})$ in Step 2(b). Indeed, the translation would be incorrect if $\neg\neg$ was simply dropped.

Proposition 3 *Let F be a sentence, let $\mathbf{p}$ be a list of distinct predicate constants and let q be a predicate constant that does not belong to the signature of F. For any occurrence of a subformula $\exists x G(x, \mathbf{y})$ of F where $\mathbf{y}$ is the list of all free variables in $\exists x G(x, \mathbf{y})$, let F' be the formula obtained from F by replacing that occurrence with $\neg\neg q(\mathbf{y})$. If $G(x, \mathbf{y})$ contains no strictly positive occurrences of predicate constants from $\mathbf{p}$, then the models of*

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

restricted to the signature of F are precisely the models of $\text{SM}[F; \mathbf{p}]$.

Alternatively, if the domain is known and finite, existential quantifiers can be replaced with multiple disjunctions. But this translation is not modular, and such multiple disjunctions need to be updated when the domain changes. Also one would imagine Skolemization, but there is a problem since answer sets are Herbrand models. Consider

$$p(a) \wedge \exists x\, p(x) \wedge \forall xy(p(x) \wedge p(y) \wedge x \neq y \rightarrow \bot),$$

which asserts that p is a singleton. If Step 2 was replaced with Skolemization, the translation would yield RASPL-1^M program

$$p(a) \qquad p(b) \qquad \leftarrow p(x), p(y), x \neq y \qquad p(x)\ ;\mathit{not}\ p(x)$$

(b is a Skolem constant), which has no answer sets. Step 3 is to expand the list of intensional predicates to cover all the predicates occurring in the description, as in Proposition 1. The transformation until Step 3 yields a set of implications where each antecedent and consequent is formed from atoms by allowing $\neg$, $\wedge$, and $\vee$ nested arbitrarily, similar to the syntax of a program with nested expressions from [Lifschitz *et al.*, 1999]. The well-known transformation that turns a program with nested expressions into a disjunctive logic program from [Lifschitz *et al.*, 1999] can be applied to turn these set of implications into a RASPL-1^M program.

Theorem 2 *Let T be an event calculus domain description, let σ be the signature consisting of the object, function and predicate constants occurring in T, and let Π be a RASPL-1^M program obtained by applying the translation* EC2ASP *to T. The stable models of Π restricted to σ (disregarding all new predicate constants p_G introduced in Step 2 (b)) are precisely the models of T of signature σ.*

Turning the resulting RASPL-1^M program further into the input language of LPARSE requires minor rewriting, such as moving equality or negated atoms to the body (e.g., *not* $p(\mathbf{t}) \leftarrow \ldots$ into $\leftarrow \ldots, p(\mathbf{t})$), and adding domain predicates in the body for all variables occurring in the rule. [9]

6 Experiments

We have implemented a prototype of an ASP-based event calculus reasoner called ECASP, based on the translation in Section 5.2. The implementation, along with the results of our experiments are available at the ECASP homepage:

http://reasoning.eas.asu.edu/ecasp .

The system turns an event calculus description into the input language of LPARSE.

6.1 Comparison with SAT-based Approach

The Discrete Event Calculus (DEC) reasoner[10] is an implementation of the event calculus written by Erik Mueller [2004b]. The system reduces event calculus reasoning into satisfiability checking by turning circumscription into predicate completion ([Lifschitz, 1994, Proposition 2]) and then finds the models using SAT solvers. Thanks to the availability of efficient SAT solvers, the system outperforms traditional abductive event calculus planners. Compared to another similar approach in [Shanahan and Witkowski, 2004], the system handles a wider range of reasoning tasks. Out of the 14 benchmark problems from [Shanahan, 1997; 1999], the DEC reasoner was shown to be able to handle 11 of them, while the other approach can handle only one [Mueller, 2004a].

However, since circumscription is not always reducible to completion, the DEC reasoner is unable to handle effect constraints, disjunctive event axioms and compound events. For example, it is well known that the following effect constraints that describe the indirect effects of the agent's walking on the objects that he is holding cannot be handled by predicate completion.

$$\mathit{HoldsAt}(\mathit{Holding}(a, o), t) \wedge \mathit{Initiates}(e, \mathit{InRoom}(a, r), t) \rightarrow \mathit{Initiates}(e, \mathit{InRoom}(o, r), t)$$

On the other hand, the rule corresponding to this formula can be directly handled by answer set solvers. Indeed, our implementation can handle all event calculus axioms reviewed in Section 2, and solved all 14 benchmark problems.

We compared the performance of (i) the DEC reasoner (v 1.0) running RELSAT (v 2.0) with (ii) ECASP with LPARSE (v 1.1.1)+CMODELS (v 3.75) running RELSAT (v 2.0), (iii) ECASP with GRINGO (v 2.0.2) +CLASP (v 1.1.3) (CLASPD (v 1.1) used for disjunctive programs), and (iv) ECASP with CLINGO (v 2.0.2). ECASP turns the input in the language of the DEC reasoner into the language of LPARSE. Both 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 generating clauses. CLINGO is a system that combines GRINGO and CLASP in a monolithic way. We have tested many examples and some notable differences are reported in the table. The first five examples are part of the benchmark problems from [Shanahan, 1997; 1999]. The next four are from [Mueller, 2006]. (We increased timepoints to see more notable differences.) All experiments were done on a Pentium machine with 3.00 GHz CPU and 2GB RAM running 64 bit Linux. The reported run times 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

[9] If we are only interested in answer sets, rather than (non-Herbrand) stable models (recall the distinction described in Section 3), UNA axioms can be dropped.

[10] *http://decreasoner.sourceforge.net* .

Problem (max. step)	DEC reasoner	ECASP w/ LPA + CMO	ECASP w/ GRI + CLA	ECASP w/ CLI
BusRide (15)	—	0.48 (0.42+0.06) A:156/R:7899 C:188	0.04 (0.03+0.01) A:733/R:3428	—
Commuter (15)	—	498.11 (447.50+50.61) A:4913/R:7383943 C:4952	44.42 (37.86 + 6.56) A:24698/R:5381620	28.79
Kitchen Sink (25)	71.10 (70.70+0.40) A:1014/C:12109	43.17 (37.17+6.00) A:123452/R:482018 C:0	2.47 (1.72+0.75) A:114968/R:179195	2.03
Thielscher Circuit (20)	13.9 (13.6+0.3) A:5138/C:16122	0.53 (0.48+0.05) A:4336/R:11399 C:0	0.07 (0.05+0.02) A:1743/R:5669	0.05
Walking Turkey (15)	—	0.05 (0.04+0.01) A:556/R:701 C:0	0.04 (0.01+0.03) A:364/R:503	0.01
Falling w/ AntiTraj (15)	270.2 (269.3+0.9) A:416/C:3056	0.74 (0.66+0.08) A:5757/R:10480 C:0	0.10 (0.08+0.02) A:4121/R:7820	0.08
Falling w/ Events (25)	107.70 (107.50+0.20) A:1092/C:12351	34.77 (30.99+3.78) A:1197/R:390319 C:1393	2.90 (2.01+0.89) A:139995/R:208282	2.32
HotAir Baloon (15)	61.10 (61.10+0.00) A:288/C:1163	0.19 (0.16+0.03) A:489/R:2958 C:678	0.04 (0.03+0.01) A:1137/R:1909	0.03
Telephone1 (40)	18.20 (17.70+0.50) A:5419/C:41590	1.70 (1.51+0.19) A:23978/R:29841 C:0	0.31 (0.26+0.05) A:21333/R:27037	0.25

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

spent by the DEC reasoner in producing output in a neat format. 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 into a set of clauses, and calling the SAT solver. The time spent by ECASP 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.

Overall, ECASP with CLINGO was the clear winner, followed by ECASP with GRINGO+CLASP. Though the DEC reasoner and CMODELS call the same SAT solver RELSAT, the number of atoms produced by the DEC reasoner is in general much smaller. This is because the DEC reasoner adopts an encoding method (that is based on predicate completion) which avoids a large number of ground instances of atoms such as *Initiates*(e, f, t), *Terminates*(e, f, t), and *Releases*(e, f, t). 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; 1999], eight of them belong to this case. Another reason for the good performance of the ASP approach is the efficient grounding methods implemented in LPARSE and GRINGO, which are drastically faster than that of the DEC reasoner.

6.2 Comparison with Mueller's ASP Approach

Though no general method was given in Mueller's ASP approach on the webpage cited in the introduction, we observe a few notable differences. One difference is that no choice rules were used in Mueller's examples so that all predicates are to be minimized, which resulted in considering prediction problems only. On the other hand, our approach can handle planning and postdiction problems as well. To solve a planning problem, *Happens* should be exempt from minimization in logic programs, which can be achieved by adding choice rules for it. More examples can be found from the ECASP homepage.

7 Conclusion

The possibility of embedding circumscriptive theories into the new language of stable models indicates that the latter is as general as the former, and that ASP is a viable approach to computing circumscriptive theories. We observe that the ASP approach can compute the *full* version of the event calculus, assuming that the domain is known and finite. Thanks to remarkable progress of ASP solvers in computing stable models, together with efficient grounding methods, our experiments showed promising results.

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 merging the descriptions written in each of them.

Acknowledgements

We are grateful to 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. The authors were partially supported by the National Science Foundation under Grant IIS-0839821.

References

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

[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. 2009. This volume.

[Ferraris *et al.*, 2010] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. Stable models and circumscription.[11] *Artificial Intelligence*, 2010. To appear.

[Gelfond and Lifschitz, 1998] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 3:195–210, 1998.

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

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

[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.

[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. *J. Log. Comput.*, 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.

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[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.

[11] *http://peace.eas.asu.edu/joolee/papers/smcirc.pdf* .