Classical Logic Event Calculus as Answer Set Programming

Joohyung Lee and Ravi Palla

School of Computing and Informatics
Arizona State University, Tempe, AZ, USA
{joolee, Ravi.Palla}@asu.edu

Abstract. Recently, Ferraris, Lee and Lifschitz presented a generalized definition of a stable model that applies to the syntax of arbitrary first-order sentences, under which a logic program is viewed as a special class of first-order sentences. The new definition of a stable model is similar to the definition of circumscription, and can even be characterized in terms of circumscription. In this paper, we show the opposite direction, that is, how to embed circumscription into the new stable model semantics, and based on this, how to turn some versions of the classical logic event calculus into the general language of stable models. By turning the latter to answer set programs under certain conditions, we show that answer set solvers can be used for classical logic event calculus reasoning, allowing more expressive query answering than what can be handled by the current SAT-based implementations of the event calculus. We prove the correctness of our translation method and compare our work with the related work by Mueller.

1 Introduction

Recently, Ferraris, Lee and Lifschitz [1] presented a generalized definition of a stable model that applies to the syntax of arbitrary first-order sentences. Under this framework, a logic program is viewed as a special class of first-order sentences, in which negation as failure (*not*) is identified with classical negation ($\neg$) under the stable model semantics. The new definition of a stable model is given by a translation into second-order logic, and does not refer to grounding to define the meaning of variables. This allowed to lift the notion of stable models to a special class of first-order models, not restricted to Herbrand models. The new definition is similar to the definition of circumscription [2; 3], and was even characterized in terms of circumscription [1], extending the work by Lin [4]. The same characterization was also independently given in [5].

The opposite direction, turning (parallel) circumscription into the stable model semantics, was shown in [6], limited to the propositional case. In this paper, we start with generalizing this result: turning first-order circumscription into the generalized language of stable models. This leads to the following natural question: how are the formalisms for reasoning about actions and change that are based on circumscription, related to the stable model semantics? Recall that as nonmonotonic formalisms, circumscription and the stable model semantics have served to provide (different) solutions to the frame problem. A group of action formalisms, such as the classical logic event calculus [7] and temporal action logic [8], take (monotonic) first-order logic as the basis, augmented with circumscription to handle the frame problem. On the other hand, action language $\mathcal{A}$ and many of its descendants [9] refer to logic programs under the stable model semantics (a.k.a. answer set programs) as the underlying formalism. Although there have been some papers that relate classical logic based action formalisms to each other (e.g., [10; 11]), not much work was done in relating them to action languages and to answer set programs.

As an initial step, we show how to turn the classical logic event calculus into the general language of stable models. Note that the event calculus is a family of languages with some variance. Here we consider versions of the event calculus that are based on classical logic, one defined by Miller and Shanahan [12], and the other by Mueller [13], which is a simplified version of the former. The fact that circumscription can be reduced to completion [14] under certain syntactic conditions ([15, Proposition 2]) allowed efficient satisfiability solvers (SAT) to be used for event calculus reasoning [16; 13], similar to the idea of SAT-based answer set programming. Interestingly, early versions of the event calculus [17] were based on logic programs but this was the time before the invention of the stable model semantics, while more extensive later developments of the event calculus were carried out under the classical logic setting. Our work here can be viewed as turning back to the logic program tradition, in the modern form of answer set programming. This is not only interesting from a theoretical perspective, but also interesting from a computational perspective, as it allows answer set solvers to be used for event calculus reasoning. In contrast to the SAT-based approaches from [16; 13] which rely on completion and hence cannot allow certain recursive axioms in the event calculus, we show that the answer set programming approach handles all the axioms correctly, modulo grounding. Our work shows that the new language of stable models is a suitable nonmonotonic formalism as general as circumscription to be applied in commonsense reasoning, with the unique advantage of having efficient ASP solvers as computational tools.

Our work is motivated by Erik Mueller's work that is available on the webpage `http://decreasoner.sourceforge.net/csr/ecas/`, where a few example answer set programs were used to illustrate that event calculus like reasoning can be done in answer set programming. However, this was a kind of "proof of concept" [1] and no formal justification was provided.

The paper is organized as follows. In the following two sections, we review the syntax of the event calculus and the generalized language of stable models. In Section 4, we present the language RASPL^M ("Many-sorted extension of Reductive Answer Set Programming Language"), for which syntactically similar codes are accepted by LPARSE,[2] the front-end of SMODELS and several other answer set solvers. We show how to turn circumscription into the first-order language of stable models in Section 5, and how to turn a description in the event calculus into a RASPL^M program in Section 6. We compare our method with Mueller's work in Section 7.

[1] Personal communication with Erik Mueller.

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

2 Review of the Event Calculus

Since the notion of equivalence under classical logic is weaker than the notion of equivalence under the stable model semantics, classically equivalent formulas do not necessarily have the same stable models. Thus any translation from classical logic based formalisms into the stable model semantics will need to fix the syntax of the former. Here we follow the syntax of the classical logic event calculus as described in [18, 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* is a term whose sort is an event and a *timepoint term* is a term whose sort is a time point. A *condition* in the event calculus 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.

In all the subsequent sections, 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

$$\begin{array}{l}\mathrm{CIRC}[\Sigma\ ;\ \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{CIRC}[\Delta_1 \wedge \Delta_2\ ;\ \mathit{Happens}]\\ \wedge\mathrm{CIRC}[\Theta\ ;\ \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge E\end{array}$$

where

- Σ is a conjunction of axioms of the form

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

  where each π_1 and π_2 is either *Initiates* or *Terminates*;
- Δ_1 is a conjunction of axioms of the form $\mathit{Happens}(e,t)$ and *temporal ordering formulas* which are comparisons between timepoint terms;
- Δ_2 is a conjunction of axioms of the form

$$\begin{array}{ll}\gamma \rightarrow \mathit{Happens}(e,t) & \\ \sigma(e,t) \wedge \pi_1(e_1,t) \wedge \cdots \wedge \pi_n(e_n,t) \rightarrow \mathit{Happens}(e,t) & \\ \mathit{Happens}(e,t) \rightarrow \mathit{Happens}(e_1,t) \vee \cdots \vee \mathit{Happens}(e_n,t) & \text{(“disjunctive event axiom”)}\end{array}$$

where σ is *Started* or *Stopped* and each π_j $(1 \leq j \leq n)$ is either *Initiated* or *Terminated*. Predicates *Started*, *Stopped*, *Initiated* and *Terminated* are defined as follows:

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

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

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

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

- Θ is a conjunction of axioms of the form $\gamma \rightarrow Ab_i(\dots,t)$;
- Ω is a conjunction of unique name axioms ;
- Ψ is a conjunction of axioms of the form [3]

$$\begin{gathered}\gamma, \quad \gamma_1 \rightarrow \gamma_2, \quad \gamma_1 \leftrightarrow \gamma_2 \\ \mathit{Happens}(e,t) \rightarrow \gamma \\ \mathit{Happens}(e_1,t) \wedge \gamma \wedge [\neg]\mathit{Happens}(e_2,t) \rightarrow \bot \ ;\end{gathered}$$

- Π is a conjunction of trajectory axioms and anti-trajectory axioms of the form
$$\gamma \rightarrow \mathit{(Anti)Trajectory}(f_1,t_1,f_2,t_2) \ ;$$
- Γ is a conjunction of observations of the form $\mathit{HoldsAt}(f,t)$ and $\mathit{ReleasedAt}(f,t)$;
- E is a conjunction of the event calculus axioms *DEC* or *EC*. [4]

As shown, a classical logic event calculus description may contain existential quantifiers; some parts of the description are circumscribed on a partial list of predicates, while some others are not circumscribed. These features look different from logic programs. Nonetheless we show that the classical logic event calculus can be embedded into logic programs.

3 Review of the New Stable Model Semantics and the New Splitting Theorem

Under the new definition of stable models presented in [19] that is applicable to arbitrary first-order sentences, a logic program is identified as a universal formula, called the *FOL-representation*. First, we identify the logical connectives—the comma, the semicolon, and *not* with their counterparts in classical logic $\wedge$,

[3] The last formula is a minor rewriting of the formula from [18] which is $\mathit{Happens}(e_1,t) \wedge \gamma \rightarrow [\neg]\mathit{Happens}(e_2,t)$. This rewriting simplifies the later presentation.

[4] Due to lack of space, we refer the reader to [18, Chapter 2] for these axioms.

$\vee$ and $\neg$. The *FOL-representation of a rule Head $\leftarrow$ Body* is the universal closure of the implication *Body* $\rightarrow$ *Head*. The *FOL-representation of a program* is the conjunction of the FOL-representations of its rules. For example, the FOL-representation of the program

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

is

$$p(a) \wedge q(b) \wedge \forall x((p(x) \wedge \neg q(x)) \rightarrow r(x)) \tag{1}$$

We review the new definition of stable models from [19]. 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 $\mathbf{u} = \mathbf{p}$ we denote the conjunction of the formulas $\forall \mathbf{x}(u_i(\mathbf{x}) \leftrightarrow p_i(\mathbf{x}))$, where $\mathbf{x}$ is a list of distinct object variables of the same arity as the length of p_i, for all $i = 1, \ldots n$. 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 $i = 1, \ldots n$, and $\mathbf{u} < \mathbf{p}$ stands for $(\mathbf{u} \leq \mathbf{p}) \wedge \neg(\mathbf{u} = \mathbf{p})$.

For any first-order sentence $F(\mathbf{p})$, expression SM$[F; \mathbf{p}]$ stands for the second-order sentence

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

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:

–
$$p_i(t_1, \ldots, t_m)^* = \begin{cases} u_i(t_1, \ldots, t_m) & \text{if } p_i \text{ belongs to } \mathbf{p}, \\ p_i(t_1, \ldots, t_m) & \text{otherwise;} \end{cases}$$

– $(t_1 = t_2)^* = (t_1 = t_2)$;
– $\bot^* = \bot$;
– $(F \wedge G)^* = F^* \wedge G^*$;
– $(F \vee G)^* = F^* \vee G^*$;
– $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;
– $(\forall x F)^* = \forall x F^*$;
– $(\exists x F)^* = \exists x F^*$.

As before, 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 $F \rightarrow \bot$.

SM$[F]$ defined in [1] is identical to SM$[F; \mathbf{p}]$ where intensional predicate constants $\mathbf{p}$ range over all predicate constants that occur in F. According to [1], the models of SM$[F]$ whose signature σ consists of the object, function and predicate constants occurring in F are called the *stable models* of F. Among those stable models we call the Herbrand models of signature σ, the *answer sets* of F. The definition of stable models is closely related to the definition of quantified equilibrium model [20; 1]. The answer sets of a logic program Π are defined as the answer sets of the FOL-representation of Π. Proposition 1 from [1] shows that, for normal logic programs, this definition is equivalent to the definition of answer sets from [21].

As shown in [19], the extended notion of SM by a partial list of intensional predicates is not essential in the sense that it can be rewritten so that intensional predicates become exactly those that occur in the formula. 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 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 $\mathbf{p}$; by $pr(F)$ we denote the list of all predicate constants occurring in F.

Proposition 1 ([19])

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

is logically valid.

However, it is convenient to describe our main results and the following splitting theorem using the generalized notion of SM.

Recall that the occurrence of one formula in another is called *positive* if the number of implications containing that occurrence in the antecedent is even, and *negative* otherwise. We say that an occurrence of a subformula or a predicate constant in a formula F is *strictly positive* if the number of implications in F containing that occurrence in the antecedent is 0. For example, in (1), both occurrences of q are positive, but only the first is strictly positive. By the head predicates of F, denoted by $h(F)$, we mean the set of predicate constants that have at least one strictly positive occurrence in F. We call a formula *negative* if it has no strictly positive occurrences of predicate constants. We say that a predicate constant p *depends* on a predicate constant q in an implication $G \rightarrow H$ if

- p has a strictly positive occurrence in H, and
- q has a positive occurrence in G that does not belong to any occurrence of a negative formula in G.

The *predicate dependency graph* of a formula F is the directed graph such that

- its vertices are the predicate constants occurring in F, and
- it has an edge from a vertex p to a vertex q if p depends on q in an implication that has a strictly positive occurrence in F.

A nonempty finite subset $\mathbf{l}$ of V is called a *loop* of F if the subgraph of the predicate dependency graph of F induced by $\mathbf{l}$ is strongly connected.

We say that F and G *interact* on $\mathbf{p}$ if $F \wedge G$ has a loop $\mathbf{l}$ such that

- $\mathbf{l}$ is contained in $\mathbf{p}$,
- $\mathbf{l}$ contains an element of $h(F)$, and
- $\mathbf{l}$ contains an element of $h(G)$.

The following theorem shows how formula $\mathrm{SM}[F \wedge G;\mathbf{p}]$ can be split :

Theorem 1 ([22]) *If F and G don't interact on $\mathbf{p}$, then $\mathrm{SM}[F \wedge G;\mathbf{p}]$ is equivalent to*

(a) $\mathrm{SM}[F; \mathbf{p} \setminus h(G)] \wedge \mathrm{SM}[G; \mathbf{p} \setminus h(F)]$*, and to*
(b) $\mathrm{SM}[F; \mathbf{p} \setminus h(G)] \wedge \mathrm{SM}[G; \mathbf{p} \cap h(G)]$*, and to*
(c) $\mathrm{SM}[F; \mathbf{p} \cap h(F)] \wedge \mathrm{SM}[G; \mathbf{p} \cap h(G)] \wedge \mathit{False}(p \setminus h(F) \setminus h(G))$*.*

The theorem will be used to justify our translation method.

4 RASPL^M Programs

The definition of SM above 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 [15]). We define RASPL^M programs as a special class of sentences under this extension, which are essentially a many-sorted extension of RASPL-1 programs from [23]. We assume that the underlying signature contains an integer sort and contains several built-in symbols, such as integer constants, built-in arithmetic functions $+$, $-$, and comparison operators $<$, $\leq$, $>$ $\geq$. 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

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

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

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

The semantics of a RASPL^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 will be evaluated in the standard way, and we will consider only those "standard" interpretations. The answer sets of a RASPL^M program are the Herbrand interpretations of the signature consisting of object, function and predicate constants occurring in the program, that satisfies $\mathrm{SM}[F]$, where F is the FOL-representation of the program.

Though RASPL^M programs have no implementation, syntactically similar codes are accepted by LPARSE, whose language is essentially many-sorted.

5 Turning Circumscription to SM

Definition 1. *For any list* $\mathbf{p}$ *of predicate constants, and any formulas* G *and* H *in each of which every occurrence of predicate constants from* $\mathbf{p}$ *is strictly positive, we call implication* $G \rightarrow H$ canonical *w.r.t.* $\mathbf{p}$.

Proposition 2 *Let* F *be the universal closure of a conjunction of canonical implications w.r.t.* $\mathbf{p}$*. Then*

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

is logically valid.

Note that in the syntax of the event calculus described in Section 2, all axioms in Σ are already canonical implications w.r.t. *Initiates*,*Terminates*,*Releases*; all axioms in $\Delta_1 \wedge \Delta_2$ are canonical implications w.r.t. *Happens*; all axioms in Θ are canonical implications w.r.t. Ab_i.[5]

The proof of Proposition 2 is immediate from the following lemma, which can be proved by induction.

Lemma 1. *For any formula F in which every occurrence of predicate constants from $\mathbf{p}$ is strictly positive,*

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

is logically valid, where $\mathbf{u}$ is a list of distinct predicate variables of the same length as $\mathbf{p}$.

6 Turning Event Calculus Descriptions to SM

Theorem 2 *Given an event calculus description, let F be the conjunction of $\Omega, \Psi, \Pi, \Gamma$ and E, and let $\mathbf{p}$ be the set of all predicates (other than equality and comparisons) occurring in the event calculus description. The following theories are equivalent:*

(a) $\text{CIRC}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \text{CIRC}[\Delta; \mathit{Happens}]$ $\wedge \text{CIRC}[\Theta; Ab_1, \dots, Ab_n] \wedge F$;

(b) $\text{SM}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \text{SM}[\Delta; \mathit{Happens}]$ $\wedge \text{SM}[\Theta; Ab_1, \dots, Ab_n] \wedge F$;

(c) $\text{SM}[\Sigma \wedge \Delta \wedge \Theta \wedge F; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}, \mathit{Happens}, Ab_1, \dots, Ab_n]$;

(d) $\text{SM}[\Sigma \wedge \Delta \wedge \Theta \wedge F \wedge \mathit{Choice}(\mathbf{p} \setminus \{\mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}, \mathit{Happens}, Ab_1, \dots, Ab_n\})]$.

Proof. *Between (a) and (b):* Follows immediately from Proposition 2.

Between (b) and (c): Note first that F is equivalent to $\text{SM}[F; \emptyset]$. Since Σ, Δ, Θ, F do not interact on $\{\mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}, \mathit{Happens}, Ab_1, \dots, Ab_n\}$, from Theorem 1 (b) (applying it multiple times), it follows that (b) and (c) are equivalent.

Between (c) and (d): Follows immediately from Proposition 1. ∎

6.1 Turning Event Calculus Descriptions to RASPL^M Programs

The formulas in Theorem 2 may still contain existential quantifiers, which are not allowed in RASPL^M programs. The following procedure turns an event calculus description into a RASPL^M program by eliminating existential quantifiers using new atoms.

Definition 2 (Translation EC2ASP).

[5] We understand an axiom such as $\mathit{Happens}(e, t)$ as an abbreviation for implication $\top \rightarrow \mathit{Happens}(e, t)$.

1. *Simplify all the definitional axioms of the form*

$$\forall \mathbf{x}(p(\mathbf{x}) \overset{def}{\leftrightarrow} \exists \mathbf{y} G(\mathbf{x}, \mathbf{y})) \tag{2}$$

   except for $CC_1 - CC_4$, where $\mathbf{y}$ is a list of all free variables in G that are not in $\mathbf{x}$, as $\forall \mathbf{xy}(G(\mathbf{x}, \mathbf{y}) \rightarrow p(\mathbf{x}))$.
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*

$$\forall \mathbf{x} y(G(\mathbf{x}, y) \rightarrow p_G(\mathbf{x})). \tag{3}$$

3. *Add choice formulas $\forall x(p(\mathbf{x}) \vee \neg p(\mathbf{x}))$ for all the predicate constants p except for $\{Initiates, Terminates, Releases, Happens, Ab_1, \ldots, Ab_n, \mathbf{p_1}, \mathbf{p_2}\}$ where*
   – *$\mathbf{p_1}$ is a list of all predicate constants p considered in Step 1.*
   – *$\mathbf{p_2}$ is a list of all new predicate constants p_G introduced in Step 2.*
4. *Apply the conversion from [24] that turns programs with nested expressions into disjunctive logic programs.*

For example, consider *DEC5* axiom:

$$\begin{aligned}\forall ft((&HoldsAt(f,t) \wedge \neg ReleasedAt(f,t{+}1) \wedge \\ &\neg\exists e(Happens(e,t) \wedge Terminates(e,f,t))) \rightarrow HoldsAt(f,t{+}1)).\end{aligned} \tag{4}$$

In order to eliminate the positive occurrence of $\exists e(Happens(e,t) \wedge Terminates(e,f,t))$ in the formula, we apply Step 2(b), introducing the formula

$$\forall eft(Happens(e,t) \wedge Terminates(e,f,t) \rightarrow q(f,t)),$$

and replacing (4) with

$$\forall ft((HoldsAt(f,t) \wedge \neg ReleasedAt(f,t+1) \wedge \neg\neg\neg q(f,t)) \rightarrow HoldsAt(f,t+1)),$$

from which $\neg\neg\neg q(f,t)$ is simplified as $\neg q(f,t)$ by Step 4.[6]

We will present the proof in the next section. Here we attempt to give the idea of the translation. Step 1 can be dropped without affecting the correctness, but it yields a more succinct transformation. The simplification does not apply for $CC_1 - CC_4$ since these axioms, together with other axioms in the description, may yield loops (A more detailed explanation follows in the next section). Step 2 (a) is one of the steps in prenex normal form conversion. Instead of Skolemization, which will introduce an infinite Herbrand universe, Step 2 (b) eliminates existential quantifiers using new atoms. The transformation yields a set of implications where each antecedent and consequent are formed

[6] In general we put $\neg\neg$ in front of $p_G(\mathbf{y})$ in order to prevent from introducing unnecessary loops. A more precise explanation is given in the proof of Theorem 3.

from atoms by allowing $\neg$, $\wedge$, and $\vee$ nested arbitrarily, similar to the syntax of a program with nested expressions from [24]. The transformation that turns a program with nested expressions into a disjunctive logic program from [24] can be straightforwardly extended to turn these set of implications into a RASPL^M program.

Turning the resulting RASPL^M program further into the input language of LPARSE requires minor rewriting, such as moving equality or negated atoms to the body (e.g., $\neg p(\mathbf{t}) \leftarrow \dots$ into $\leftarrow \dots, p(\mathbf{t})$), and adding domain predicates in the body for all variables occurring in the rule. [7]

6.2 Proof of the Correctness of the Translation

The following theorem states the correctness of the translation.

Theorem 3 *Let T be an event calculus domain description, and let Π be a RASPL^M program obtained by applying the translation* EC2ASP *to T. The stable models of Π restricted to the signature of T are precisely the models of T.*

We will use the following fact for the proof.

Lemma 2. *Let F be a first-order formula, let p be a predicate constant not occurring in F, let $G(\mathbf{x})$ be a subformula of F where $\mathbf{x}$ is the list of all free variables of $G(\mathbf{x})$, and let F' be a formula obtained from F by replacing an occurrence of $G(\mathbf{x})$ with $\neg\neg p(\mathbf{x})$. The models of $F' \wedge \forall \mathbf{x}(G(\mathbf{x}) \leftrightarrow p(\mathbf{x}))$ restricted to the signature of F are exactly the models of F.*

We will also use the proposition below that relates SM to completion, extending the results of Propositions 6,8 from [1].

Let Π be a finite set of rules that have the form

$$A \leftarrow F \tag{5}$$

where A is an atom and F is a first-order formula (that may contain quantifiers). We say that Π is in normal form w.r.t. a set $\mathbf{p}$ of predicate constants if, for each predicate constant p in $\mathbf{p}$, there is exactly one rule

$$p(\mathbf{x}) \leftarrow F \tag{6}$$

where $\mathbf{x}$ is the list of object variables whose length is the same as the arity of p and F is a formula. It is clear that every program whose rules have the form (5) can be turned into a normal form w.r.t $\mathbf{p}$. Given a program Π in normal form w.r.t. $\mathbf{p}$, the *completion* of Π w.r.t $\mathbf{p}$ is the conjunction of the universal closure of formulas obtained from Π by replacing (6) with

$$p(\mathbf{x}) \leftrightarrow \exists \mathbf{y} F$$

where $\mathbf{y}$ is the list of free variables occurring in F that are not in $\mathbf{x}$.

We say that a first-order formula F is tight on $\mathbf{p}$ if the subgraph of the dependency graph of F induced by $\mathbf{p}$ is acyclic.

[7] If we are only interested in answer sets, rather than stable models (note the distinction made in Section 3), UNA axioms can be disregarded.

Proposition 3 *Let Π be a program in normal form w.r.t.* $\mathbf{p}$ *and let F be the FOL-representation of Π. If F is tight on* $\mathbf{p}$*, then* $\mathrm{SM}[F;\mathbf{p}]$ *is equivalent to the completion of Π w.r.t.* $\mathbf{p}$.

Proof of Theorem 3 Assume that T is

$$\begin{array}{l}\mathrm{CIRC}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{CIRC}[\Delta; \mathit{Happens}] \\ \wedge\ \mathrm{CIRC}[\Theta; Ab_1, \dots, Ab_n] \wedge F\ ,\end{array}$$

which is equivalent to

$$\begin{array}{l}\mathrm{SM}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{SM}[\Delta; \mathit{Happens}] \\ \wedge\ \mathrm{SM}[\Theta; Ab_1, \dots, Ab_n] \wedge F\end{array}$$

by Theorem 2.

Let D_1 be the set of all definitions of the form (2) except for $CC_1 - CC_4$. Let Step 1′ be the transformation that turns each formula (2) in D_1 into

$$\mathrm{SM}[\forall \mathbf{xy}(G(\mathbf{x},\mathbf{y}) \rightarrow p(\mathbf{x}));\ p], \tag{7}$$

and let Step 2′ be a modification of Step 2 in EC2ASP by introducing

$$\forall \mathbf{x}y(G(\mathbf{x},y) \leftrightarrow p_G(\mathbf{x})) \tag{8}$$

instead of (3) in Step 2 (b).

Let Σ', Δ', Θ', F' be the formulas obtained from Σ, Δ, Θ, F by applying Steps 1′ and 2′. By Proposition 3, Step 1′ is an equivalent transformation. By Lemma 2, the models of

$$\begin{array}{l}\mathrm{SM}[\Sigma'; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{SM}[\Delta'; \mathit{Happens}] \\ \wedge\ \mathrm{SM}[\Theta'; Ab_1, \dots, Ab_n] \wedge F'\end{array} \tag{9}$$

restricted to the signature of T are precisely the models of T. Let D_2 be the set of all definitions (8) that are introduced in Step 2′. Note that these formulas are introduced only for some formulas in F (other than D_1), and not for Σ, Δ and Θ; formulas Σ', Δ' and Θ' are obtained by applying Step 2 (a) only since, according to the syntax of the event calculus (Section 2), every occurrence of existential quantification is negative in each of Σ, Δ and Θ. Let F'' be the axioms in F' excluding D_2 and all formulas (7) for D_1.

By Proposition 3 again, each formula (8) in D_2 is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}y(G(\mathbf{x},y) \rightarrow p_G(\mathbf{x}));\ p_G].$$

Consequently (9) is equivalent to

$$\begin{array}{l}\mathrm{SM}[\Sigma'; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{SM}[\Delta'; \mathit{Happens}] \\ \wedge\ \mathrm{SM}[\Theta'; Ab_1, \dots, Ab_n] \wedge \mathrm{SM}[F''; \emptyset] \\ \wedge \bigwedge_{(2)\in D_1} \mathrm{SM}[\forall \mathbf{xy}(G(\mathbf{x},\mathbf{y}) \rightarrow p(\mathbf{x}));\ p] \\ \wedge \bigwedge_{(8)\in D_2} \mathrm{SM}[\forall \mathbf{x}y(G(\mathbf{x},y) \rightarrow p_G(\mathbf{x}));\ p_G].\end{array} \tag{10}$$

Since Σ' and Δ' do not interact on $\{Initiates, Terminates, Releases, Happens\}$, by Theorem 1 (b), formula (10) is equivalent to

$$\begin{aligned}&\mathrm{SM}[\Sigma' \wedge \Delta'; Initiates, Terminates, Releases, Happens]\\ &\wedge \mathrm{SM}[\Theta'; Ab_1, \dots, Ab_n] \wedge \mathrm{SM}[F''; \emptyset]\\ &\wedge \textstyle\bigwedge_{(2)\in D_1} \mathrm{SM}[\forall \mathbf{xy}(G(\mathbf{x},\mathbf{y}) \rightarrow p(\mathbf{x}));\ p]\\ &\wedge \textstyle\bigwedge_{(8)\in D_2} \mathrm{SM}[\forall \mathbf{x}y(G(\mathbf{x},y) \rightarrow p_G(\mathbf{x}));\ p_G].\end{aligned} \tag{11}$$

Similarly, by applying Theorem 1 multiple times, it is clear that formula (11) is equivalent to

$$\begin{aligned}&\mathrm{SM}[\Sigma' \wedge \Delta' \wedge \Theta' \wedge F''; Initiates, Terminates, Releases, Happens, Ab_1, \dots, Ab_n]\\ &\wedge \mathrm{SM}[\textstyle\bigwedge_{(2)\in D_1} \forall \mathbf{xy}(G(\mathbf{x},\mathbf{y}) \rightarrow p(\mathbf{x}));\ \mathbf{p_1}]\\ &\wedge \mathrm{SM}[\textstyle\bigwedge_{(8)\in D_2} \forall \mathbf{x}y(G(\mathbf{x},y) \rightarrow p_G(\mathbf{x}));\ \mathbf{p_2}]\end{aligned} \tag{12}$$

where $\mathbf{p_1}$ is a list of all predicate constants p defined in D_1, and $\mathbf{p_2}$ is a list of all new predicate constants p_G defined in D_2.

According to the syntax of the event calculus (Section 2), $\Sigma' \wedge \Delta' \wedge \Theta' \wedge F''$ and

$$\bigwedge_{(2)\in D_1} \forall \mathbf{xy}(G(\mathbf{x},\mathbf{y}) \rightarrow p(\mathbf{x}))$$

do not interact on

$$\{Initiates, Terminates, Releases, Happens, Ab_1, \dots, Ab_n, \mathbf{p_1}\},$$

so that, by Theorem 1 (b), (12) is equivalent to

$$\begin{aligned}&\mathrm{SM}[\Sigma' \wedge \Delta' \wedge \Theta' \wedge F'' \wedge \textstyle\bigwedge_{(2)\in D_1} \forall \mathbf{xy}(G(\mathbf{x},\mathbf{y}) \rightarrow p(\mathbf{x}));\\ &\qquad Initiates, Terminates, Releases, Happens, Ab_1, \dots, Ab_n, \mathbf{p_1}]\\ &\wedge \mathrm{SM}[\textstyle\bigwedge_{(8)\in D_2} \forall \mathbf{x}y(G(\mathbf{x},y) \rightarrow p_G(\mathbf{x})); \mathbf{p_2}].\end{aligned} \tag{13}$$

From the fact that every occurrence of a predicate constant from $\mathbf{p_2}$ in F'' is preceded with $\neg\neg$, we conclude that the formula in the first SM and the formula in the second SM in (13) do not interact on

$$\{Initiates, Terminates, Releases, Happens, Ab_1, \dots, Ab_n, \mathbf{p_1}, \mathbf{p_2}\}$$

so that, by Theorem 1 (b), (13) is equivalent to

$$\begin{aligned}&\mathrm{SM}[\Sigma' \wedge \Delta' \wedge \Theta' \wedge F'' \wedge \textstyle\bigwedge_{(2)\in D_1} \forall \mathbf{xy}(G(\mathbf{x},\mathbf{y}) \rightarrow p(\mathbf{x}))\\ &\wedge \textstyle\bigwedge_{(8)\in D_2} \forall \mathbf{x}y(G(\mathbf{x},y) \rightarrow p_G(\mathbf{x}));\\ &Initiates, Terminates, Releases, Happens, Ab_1, \dots, Ab_n, \mathbf{p_1}, \mathbf{p_2}]\end{aligned} \tag{14}$$

Formula (14) is exactly the formula obtained from Steps 1 and 2. The rest of the proof follows immediately from Proposition 1, and a straightforward extension of Proposition 7 from [24]. ∎

If we were to treat $CC_1 - CC_4$ same as the other definitional axioms, then Theorem 1 (b) won't justify that (12) is equivalent to (13), since there may be a loop, such as $\{Started, Happens, Initiated\}$, on which Δ' and $CC_1 - CC_4$ interact. Indeed, (12) and (13) are not equivalent in general if $CC_1 - CC_4$ were regarded to belong to D_1.

7 Comparison with Mueller's Work

7.1 Comparison with Mueller's ASP Approach

The answer set programming approach on the webpage

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

illustrates the idea using some examples only, and misses formal justification. Still we observe a few differences.[8]

First, these examples use classical negation, while our method does not. We do not need both negations—negation as failure (*not*) and classical negation ($\neg$)—to embed the two-valued event calculus into answer set programming. Second, no choice rules were used, which resulted 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 logic programs. The following example is a logic program counterpart of the planning problem example on page 244 of [18].

```
agent(james).
fluent(awake(A)) :- agent(A).
event(wakeUp(A)) :- agent(A).
initiates(wakeUp(A), awake(A), T) :- agent(A).
:- holdsAt(awake(james), 0).
holdsAt(awake(james), 1).
:- releasedAt(F, 0).
0 {happens(E, T)} 1 :- T<1.
```

When the above program along with the *DEC* axioms is provided as input, SMODELS returns an answer set that contains `happens(wakeup(james), 0)`.

More examples can be found from `http://reasoning.eas.asu.edu/ecasp`.

7.2 Comparison with the *DEC* Reasoner

The *DEC* reasoner[9] is an implementation of the event calculus written by Erik Mueller. The system reduces event calculus reasoning into satisfiability and calls SAT solvers [25]. Since circumscription is not always reducible to completion, some event calculus axioms like effect constraints and disjunctive event axioms

[8] Our version of EC/DEC axioms is available at `http://reasoning.eas.asu.edu/ecasp`.

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

(Section 2) cannot be handled by the *DEC* reasoner. For example, consider the following event calculus axioms that describe the indirect effects of the agent walking from one room to another on the objects that he is holding:

$$\begin{array}{l} \mathit{HoldsAt}(\mathit{Holding}(a,o),t) \wedge \mathit{Initiates}(e,\mathit{InRoom}(a,r),t) \\ \qquad \rightarrow \mathit{Initiates}(e,\mathit{InRoom}(o,r),t) \\ \mathit{HoldsAt}(\mathit{Holding}(a,o),t) \wedge \mathit{Terminates}(e,\mathit{InRoom}(a,r),t) \\ \qquad \rightarrow \mathit{Terminates}(e,\mathit{InRoom}(o,r),t) \end{array} \tag{15}$$

Since these axioms involve non-trivial loops, they cannot be reduced to completion. On the other hand, the logic program corresponding to (15) can be directly handled by answer set solvers.

8 Conclusion

Our contributions are as follows.

– We showed how to embed circumscription into the new language of stable models.
– Based on it we showed how to turn the classical logic event calculus into answer set programs, and proved the correctness of the translation. This approach can handle the *full* version of the event calculus, modulo grounding.

We plan to implement this transformation method, and compare it with the *DEC* reasoner. Another future work is to extend the embedding method to other action formalisms, such as temporal action logics and situation calculus.

Acknowledgements

We are grateful to Vladimir Lifschitz, Erik Mueller and anonymous referees for useful comments on the ideas of this paper and several pointers to earlier work. We are also grateful to Tae-Won Kim for useful discussions. The authors were partially supported by the National Science Foundation under Grant IIS-0839821.

References

1. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI). (2007) 372–379
2. McCarthy, J.: Circumscription—a form of non-monotonic reasoning. Artificial Intelligence **13** (1980) 27–39,171–172
3. McCarthy, J.: Applications of circumscription to formalizing common sense knowledge. Artificial Intelligence **26** (1986) 89–116
4. Lin, F.: A Study of Nonmonotonic Reasoning. PhD thesis, Stanford University (1991)

5. Lin, F., Zhou, Y.: From answer set logic programming to circumscription via logic of GK. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI). (2007)
6. Lee, J., Lin, F.: Loop formulas for circumscription. Artificial Intelligence **170** (2006) 160–185
7. Shanahan, M.: A circumscriptive calculus of events. Artif. Intell. **77** (1995) 249–284
8. Doherty, P., Gustafsson, J., Karlsson, L., Kvarnström, J.: TAL: Temporal action logics language specification and tutorial.[10] Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841 **3** (1998)
9. Gelfond, M., Lifschitz, V.: Action languages.[11] Electronic Transactions on Artificial Intelligence **3** (1998) 195–210
10. Mueller, E.T.: Event calculus and temporal action logics compared. Artif. Intell. **170** (2006) 1017–1029
11. Belleghem, K.V., Denecker, M., Schreye, D.D.: On the relation between situation calculus and event calculus. J. Log. Program. **31** (1997) 3–37
12. Miller, R., Shanahan, M.: The event calculus in classical logic - alternative axiomatisations. Electron. Trans. Artif. Intell. **3** (1999) 77–105
13. Mueller, E.T.: Event calculus reasoning through satisfiability. J. Log. Comput. **14** (2004) 703–730
14. Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Data Bases. Plenum Press, New York (1978) 293–322
15. Lifschitz, V.: Circumscription. In Gabbay, D., Hogger, C., Robinson, J., eds.: Handbook of Logic in AI and Logic Programming. Volume 3. Oxford University Press (1994) 298–352
16. Shanahan, M., Witkowski, M.: Event calculus planning through satisfiability. J. Log. Comput. **14** (2004) 731–745
17. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing **4** (1986) 67–95
18. Mueller, E.: Commonsense reasoning. Elsevier (2006)
19. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artificial Intelligence (2008) To appear.
20. Pearce, D., Valverde, A.: A first order nonmonotonic extension of constructive logic. Studia Logica **80** (2005) 323–348
21. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K., eds.: Proceedings of International Logic Programming Conference and Symposium, MIT Press (1988) 1070–1080
22. Ferraris, P., Lee, J., Lifschitz, V., Palla, R.: Two types of splitting in the theory of stable models. Unpublished Draft (2008)
23. Lee, J., Lifschitz, V., Palla, R.: A reductive semantics for counting and choice in answer set programming. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). (2008) 472–479
24. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Annals of Mathematics and Artificial Intelligence **25** (1999) 369–389
25. Mueller, E.T.: A tool for satisfiability-based commonsense reasoning in the event calculus. In Barr, V., Markov, Z., eds.: FLAIRS Conference, AAAI Press (2004)

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

[11] `http://www.ep.liu.se/ea/cis/1998/016/` .