Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Answer Set Programming Modulo Theories and Reasoning about Continuous Changes

Joohyung Lee and Yunsong Meng
School of Computing, Informatics and Decision Systems Engineering
Arizona State University, Tempe, USA
{*joolee, Yunsong.Meng*}*@asu.edu*

Abstract

Answer Set Programming Modulo Theories is a new framework of tight integration of answer set programming (ASP) and satisfiability modulo theories (SMT). Similar to the relationship between first-order logic and SMT, it is based on a recent proposal of the functional stable model semantics by fixing interpretations of background theories. Analogously to a known relationship between ASP and SAT, "tight" ASPMT programs can be translated into SMT instances. We demonstrate the usefulness of ASPMT by enhancing action language $\mathcal{C}$+ to handle continuous changes as well as discrete changes. We reformulate the semantics of $\mathcal{C}$+ in terms of ASPMT, and show that SMT solvers can be used to compute the language. We also show how the language can represent cumulative effects on continuous resources.

1 Introduction

The success of answer set programming (ASP) [Lifschitz, 2008; Brewka *et al.*, 2011] is in part thanks to efficiency of ASP solvers that utilize (i) intelligent grounding—the process that replaces variables with ground terms—and (ii) efficient search methods that originated from propositional satisfiability solvers (SAT solvers). However, this method is not scalable when we need to represent functional fluents that range over large numeric domains. Since answer sets (a.k.a. stable models) are limited to Herbrand models, *functional* fluents are represented by *predicates*, but not by *functions*, as in the following example that represents the inertia assumption on *Speed*:

$$Speed_1(x) \leftarrow Speed_0(x),\ not\ \neg Speed_1(x)\ . \qquad (1)$$

Here the subscripts 0 and 1 are time stamps, and x is a variable ranging over the value domain of fluent *Speed*. A more natural representation using functions which replaces $Speed_i(x)$ with $Speed_i = x$ does not work under the ASP semantics because (i) answer sets are Herbrand models (e.g., $Speed_1 = Speed_0$ is always false under any Herbrand interpretation), and (ii) nonmonotonicity of the stable model semantics has to do with minimizing the extents of predicates but has nothing to do with functions. On the other hand, grounding rule (1) yields a large set of ground instances when *Speed* ranges over a large integer domain. Moreover, a real number domain is not supported because grounding cannot be applied.

In order to alleviate this "grounding problem," there have been several recent efforts to integrate ASP with constraint solving and satisfiability modulo theories (SMT), where functional fluents can be expressed by variables in Constraint Satisfaction Problems or uninterpreted constants in SMT. Balduccini [2009] and Gebser *et al.* [2009] combine ASP and constraint solving in a way that is similar to the "lazy approach" of SMT solvers. Niemelä [2008] shows that ASP programs can be translated into a particular SMT fragment, namely difference logic. Janhunen *et al.* [2011] presents a tight integration of ASP and SMT. All these approaches aim at addressing the issue (i) above by avoiding grounding and exploiting efficient constraint solving methods applied to functions. However, they do not address the issue (ii) because, like the standard ASP, nonmonotonicity of those extensions has to do with predicates only. For instance, a natural counterpart of (1) in the language of CLINGCON [Gebser *et al.*, 2009],

$$Speed_1 =^{\$} x \leftarrow Speed_0 =^{\$} x,\ not\ Speed_1 \neq^{\$} x\ ,$$

does not correctly represent inertia.

In this paper, we present a framework of combining answer set programming with satisfiability modulo theories, which we call *Answer Set Programming Modulo Theories (ASPMT)*. Just as SMT is a generalization of SAT and, at the same time, a special case of first-order logic in which certain predicate and function constants in background theories (such as difference logic and the theory of reals) have fixed interpretations, ASPMT is a generalization of the standard ASP and, at the same time, a special case of the functional stable model semantics [Bartholomew and Lee, 2012] that assumes background theories. Like the known relationship between SAT and ASP that *tight* ASP programs can be translated into SAT instances, *tight* ASPMT programs can be translated into SMT instances, which allows SMT solvers for computing ASPMT programs.

These results allow us to enhance action language $\mathcal{C}$+ [Giunchiglia *et al.*, 2004] to handle reasoning about continuous changes. Language $\mathcal{C}$+ is an expressive action description language but its semantics was defined in terms of

propositional causal theories, which limits the language to express discrete changes only. By reformulating $\mathcal{C}$+ in terms of ASPMT, we naturally extend the language to overcome the limitation, and use SMT solvers to compute the language. Our experiments show that this approach outperforms the existing implementations of $\mathcal{C}$+ by several orders of magnitude for some benchmark problems.

Section 2 reviews the functional stable model semantics by Bartholomew and Lee [2012], and defines ASPMT as its special case. Section 3 reformulates language $\mathcal{C}$+ in terms of ASPMT, and Section 4 shows how the reformulation can be used to represent continuous changes. The language is further extended in Section 5 to represent cumulative effects on continuous resources. Related work and the conclusion are presented in Section 6.

2 Functional SM and ASPMT

2.1 Review: Functional Stable Model Semantics

We review the semantics from [Bartholomew and Lee, 2012]. Formulas are built the same as in first-order logic. A signature consists of *function constants* and *predicate constants*. Function constants of arity 0 are called *object constants*, and predicate constants of arity 0 are called *propositional constants*.

Similar to circumscription, for predicate symbols (constants or variables) u and c, expression $u \leq c$ is defined as shorthand for $\forall \mathbf{x}(u(\mathbf{x}) \rightarrow c(\mathbf{x}))$. Expression $u = c$ is defined as $\forall \mathbf{x}(u(\mathbf{x}) \leftrightarrow c(\mathbf{x}))$ if u and c are predicate symbols, and $\forall \mathbf{x}(u(\mathbf{x}) = c(\mathbf{x}))$ if they are function symbols. For lists of symbols $\mathbf{u} = (u_1, \ldots, u_n)$ and $\mathbf{c} = (c_1, \ldots, c_n)$, expression $\mathbf{u} \leq \mathbf{c}$ is defined as $(u_1 \leq c_1) \wedge \cdots \wedge (u_n \leq c_n)$, and similarly, expression $\mathbf{u} = \mathbf{c}$ is defined as $(u_1 = c_1) \wedge \cdots \wedge (u_n = c_n)$.

Let $\mathbf{c}$ be a list of distinct predicate and function constants, and let $\widehat{\mathbf{c}}$ be a list of distinct predicate and function variables corresponding to $\mathbf{c}$. By $\mathbf{c}^{pred}$ ($\mathbf{c}^{func}$, respectively) we mean the list of all predicate constants (function constants, respectively) in $\mathbf{c}$, and by $\widehat{\mathbf{c}}^{pred}$ the list of the corresponding predicate variables in $\widehat{\mathbf{c}}$.

For any formula F, expression $\mathrm{SM}[F; \mathbf{c}]$ is defined as

$$F \wedge \neg \exists \widehat{\mathbf{c}} (\widehat{\mathbf{c}} < \mathbf{c} \wedge F^*(\widehat{\mathbf{c}})),$$

where $\widehat{\mathbf{c}} < \mathbf{c}$ is shorthand for $(\widehat{\mathbf{c}}^{pred} \leq \mathbf{c}^{pred}) \wedge \neg(\widehat{\mathbf{c}} = \mathbf{c})$, and $F^*(\widehat{\mathbf{c}})$ is defined recursively as follows.

- When F is an atomic formula, F^* is $F' \wedge F$ where F' is obtained from F by replacing all intensional (function and predicate) constants $\mathbf{c}$ in it with the corresponding (function and predicate) variables from $\widehat{\mathbf{c}}$;
- $(G \wedge H)^* = G^* \wedge H^*$; $\quad (G \vee H)^* = G^* \vee H^*$;
- $(G \rightarrow H)^* = (G^* \rightarrow H^*) \wedge (G \rightarrow H)$;
- $(\forall x G)^* = \forall x G^*$; $\quad (\exists x F)^* = \exists x F^*$.

(We understand $\neg F$ as shorthand for $F \rightarrow \bot$; $\top$ as $\neg\bot$; and $F \leftrightarrow G$ as $(F \rightarrow G) \wedge (G \rightarrow F)$.) Members of $\mathbf{c}$ are called *intensional* constants.

When F is a sentence, the models of $\mathrm{SM}[F; \mathbf{c}]$ are called the *stable* models of F *relative to* $\mathbf{c}$. They are the models of F that are "stable" on $\mathbf{c}$. The definition can be easily extended to formulas of many-sorted signatures.

This definition of a stable model is a proper generalization of the one from [Ferraris *et al.*, 2011]: in the absence of intensional function constants, it reduces to the one in [Ferraris *et al.*, 2011].

2.2 Answer Set Programming Modulo Theories

Formally, an SMT instance is a formula in many-sorted first-order logic, where some designated function and predicate constants are constrained by some fixed background interpretation. SMT is the problem of determining whether such a formula has a model that expands the background interpretation [Barrett *et al.*, 2009].

The syntax of ASPMT is the same as that of SMT. Let σ^{bg} be the (many-sorted) signature of the background theory bg. An interpretation of σ^{bg} is called a *background interpretation* if it satisfies the background theory. For instance, in the theory of reals, we assume that σ^{bg} contains the set $\mathcal{R}$ of symbols for all real numbers, the set of arithmetic functions over real numbers, and the set $\{<, >, \leq, \geq\}$ of binary predicates over real numbers. Background interpretations interpret these symbols in the standard way.

Let σ be a signature that is disjoint from σ^{bg}. We say that an interpretation I of σ satisfies F w.r.t. the background theory bg, denoted by $I \models_{bg} F$, if there is a background interpretation J of σ^{bg} that has the same universe as I, and $I \cup J$ satisfies F. For any ASPMT sentence F with background theory σ^{bg}, interpretation I is a stable model of F relative to $\mathbf{c}$ (w.r.t. background theory σ^{bg}) if $I \models_{bg} \mathrm{SM}[F; \mathbf{c}]$.

We will often write $G \leftarrow F$, in a rule form as in logic programs, to denote the universal closure of $F \rightarrow G$. A finite set of formulas is identified with the conjunction of the formulas in the set.

Example 1 *The following set F of formulas describes the inertia assumption on the speed of a car and the effect of acceleration assuming the theory of reals as the background theory.*

$$\begin{array}{lcl} \mathit{Speed}_1 = x & \leftarrow & \neg\neg(\mathit{Speed}_1 = x) \wedge \mathit{Speed}_0 = x \\ \mathit{Speed}_1 = x & \leftarrow & (x = \mathit{Speed}_0 + 3 \times \mathit{Duration}) \\ & & \wedge\ \mathit{Accelerate} = \mathrm{TRUE}\ . \end{array} \qquad (2)$$

Here x is a variable of sort $\mathcal{R}_{\geq 0}$; Speed_0, Speed_1 and $\mathit{Duration}$ are object constants with value sort $\mathcal{R}_{\geq 0}$ and $\mathit{Accelerate}$ is an object constant with value sort Boolean. For interpretation I of signature $\{\mathit{Speed}_0, \mathit{Speed}_1, \mathit{Duration}, \mathit{Accelerate}\}$ such that $\mathit{Accelerate}^I = \mathrm{FALSE}$, $\mathit{Speed}_0^I = 1$, $\mathit{Speed}_1^I = 1$, $\mathit{Duration}^I = 1.5$, we check that $I \models_{bg} \mathrm{SM}[F; \mathit{Speed}_1]$.

For another interpretation I_1 of the same signature that agrees with I except that $\mathit{Accelerate}^{I_1} = \mathrm{TRUE}$, $\mathit{Speed}_1^{I_1} = 5.5$, we check that $I_1 \models_{bg} \mathrm{SM}[F; \mathit{Speed}_1]$.

2.3 Review: Completion

We review the *theorem on completion* from [Bartholomew and Lee, 2013], which turns the functional stable model semantics into first-order logic.

A formula F is said to be in *Clark normal form* (relative to the list $\mathbf{c}$ of intensional constants) if it is a conjunction of

sentences of the form

$$\forall \mathbf{x}(G \to p(\mathbf{x})) \tag{3}$$

and

$$\forall \mathbf{x}y(G \to f(\mathbf{x}) = y) \tag{4}$$

one for each intensional predicate p and each intensional function f, where $\mathbf{x}$ is a list of distinct object variables, y is an object variable, and G is an arbitrary formula that has no free variables other than those in $\mathbf{x}$ and y.

The *completion* of a formula F in Clark normal form (relative to $\mathbf{c}$) is obtained from F by replacing each conjunctive term (3) with $\forall \mathbf{x}(p(\mathbf{x}) \leftrightarrow G)$ and each conjunctive term (4) with $\forall \mathbf{x}y(f(\mathbf{x}) = y \leftrightarrow G)$.

An occurrence of a symbol or a subformula in a formula F is called *strictly positive* in F if that occurrence is not in the antecedent of any implication in F. The *t-dependency graph* of F (relative to $\mathbf{c}$) is the directed graph that

- has all members of $\mathbf{c}$ as its vertices, and
- has an edge from c to d if, for some strictly positive occurrence of $G \to H$ in F,
  - c has a strictly positive occurrence in H, and
  - d has a strictly positive occurrence in G.

We say that F is *tight* (on $\mathbf{c}$) if the t-dependency graph of F (relative to $\mathbf{c}$) is acyclic. For example, formula (1) is tight on its signature.

Theorem 1 *[Bartholomew and Lee, 2013, Theorem 2] For any sentence F in Clark normal form that is tight on $\mathbf{c}$, an interpretation I that satisfies $\exists xy(x \neq y)$ is a model of* $\mathrm{SM}[F; \mathbf{c}]$ *iff I is a model of the completion of F relative to $\mathbf{c}$.*

Theorem 1 can be applied to formulas in non-Clark normal form if they can be rewritten in Clark normal form. The following theorem is often useful in splitting conjunctive formulas, and applying completion to subformulas.

Theorem 2 *[Bartholomew and Lee, 2012, Theorem 1] For any first-order formulas F and G, if G has no strictly positive occurrence of a constant from $\mathbf{c}$,* $\mathrm{SM}[F \wedge G; \mathbf{c}]$ *is equivalent to* $\mathrm{SM}[F; \mathbf{c}] \wedge G$.

Theorem 1 is applicable to ASPMT formulas as well. Since F in Example 1 is tight on Speed_1, according to Theorem 1, $\mathrm{SM}[F; \mathit{Speed}_1]$ is equivalent to the following SMT instance with the same background theory:

$$\begin{aligned}\mathit{Speed}_1 = x \;\leftrightarrow\; &\neg\neg(\mathit{Speed}_1 = x) \wedge (\mathit{Speed}_0 = x) \\ &\vee ((x = \mathit{Speed}_0 + 3 \times \mathit{Duration}) \wedge \mathit{Accelerate} = \mathrm{TRUE})\,.\end{aligned}$$

3 Reformulating C+ in ASPMT

3.1 Syntax

We consider a many-sorted first-order signature σ that is partitioned into three signatures: the set σ^{fl} of object constants called *fluent constants*, the set σ^{act} of object constants called *action constants*, and the background signature σ^{bg}. The signature σ^{fl} is further partitioned into the set σ^{sim} of *simple* fluent constants and the set σ^{sd} of *statically determined* fluent constants.

We assume the same syntax of formulas as in Section 2. A *fluent formula* is a formula of signature $\sigma^{fl} \cup \sigma^{bg}$. An *action formula* is a formula of $\sigma^{act} \cup \sigma^{bg}$ that contains at least one action constant and no fluent constants.

A *static law* is an expression of the form

$$\mathbf{caused}\ F\ \mathbf{if}\ G \tag{5}$$

where F and G are fluent formulas. An *action dynamic law* is an expression of the form (5) in which F is an action formula and G is a formula. A *fluent dynamic law* is an expression of the form

$$\mathbf{caused}\ F\ \mathbf{if}\ G\ \mathbf{after}\ H \tag{6}$$

where F and G are fluent formulas and H is a formula, provided that F does not contain statically determined constants. A *causal law* is a static law, or an action dynamic law, or a fluent dynamic law. A *C+ action description* is a finite set of causal laws.

For any function constant f, we say that a first-order formula is *f-plain* if each atomic formula in it

- does not contain f, or
- is of the form $f(\mathbf{t}) = t_1$ where $\mathbf{t}$ is a list of terms not containing f, and t_1 is a term not containing f.

For any list $\mathbf{c}$ of predicate and function constants, we say that F is $\mathbf{c}$-plain if F is f-plain for each function constant f in $\mathbf{c}$.

We call an action description *definite* if the head F of every causal law (5) and (6) is an atomic formula that is $(\sigma^{fl} \cup \sigma^{act})$-plain. Throughout this paper we consider definite action descriptions only, which covers the fragment of C+ that is implemented in the Causal Calculator (CCALC)[1].

3.2 Semantics

In [Giunchiglia *et al.*, 2004] the semantics of C+ is defined in terms of nonmonotonic propositional causal theories, in which every constant has a finite domain. The semantics of the enhanced C+ below is similar to the one in [Giunchiglia *et al.*, 2004] except that it is defined in terms of ASPMT in place of causal theories. This reformulation is essential for the language to represent continuous changes as it is not limited to finite domains only.

For a signature σ and a nonnegative integer i, expression $i : \sigma$ is the signature consisting of the pairs $i : c$ such that $c \in \sigma$, and the value sort of $i : c$ is the same as the value sort of c. Similarly, if s is an interpretation of σ, expression $i : s$ is an interpretation of $i : \sigma$ such that $c^s = (i : c)^{i:s}$.

For any definite action description D of signature $\sigma^{fl} \cup \sigma^{act} \cup \sigma^{bg}$ and any nonnegative integer m, the ASPMT program D_m is defined as follows. The signature of D_m is $0 : \sigma^{fl} \cup \cdots \cup m : \sigma^{fl} \cup 0 : \sigma^{act} \cup \cdots \cup (m-1) : \sigma^{act} \cup \sigma^{bg}$. By $i : F$ we denote the result of inserting $i :$ in front of every occurrence of every fluent and action constant in a formula F.

ASPMT program D_m is the conjunction of

$$\neg\neg\, i : G \to i : F$$

for every static law (5) in D and every $i \in \{0, \ldots, m\}$, and for every action dynamic law (5) in D and every $i \in \{0, \ldots, m-1\}$;

$$\neg\neg\, (i+1) : G \wedge i : H \to (i+1) : F$$

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

for every fluent dynamic law (6) in D and every $i \in \{0,\dots,m-1\}$.

The transition system represented by an action description D consists of states (vertices) and transitions (edges). A *state* is an interpretation s of σ^{fl} such that $0{:}s \models_{bg} \mathrm{SM}[D_0;\ 0{:}\sigma^{sd}]$. A *transition* is a triple $\langle s, e, s'\rangle$, where s and s' are interpretations of σ^{fl} and e is an interpretation of σ^{act}, such that

$$(0{:}s) \cup (0{:}e) \cup (1{:}s') \models_{bg} \mathrm{SM}[D_1;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl})]\ .$$

Theorems 3 and 4 below extend Propositions 7 and 8 from [Giunchiglia *et al.*, 2004] to our reformulation of $\mathcal{C}$+. These theorems justify the soundness of the language.

The definition of the transition system above implicitly relies on the following property of transitions:

Theorem 3 *For every transition $\langle s, e, s'\rangle$, s and s' are states.*

The following theorem states the correspondence between the stable models of D_m and the paths in the transition system represented by D:

Theorem 4

$$(0{:}s_0) \cup (0{:}e_0) \cup (1{:}s_1) \cup (1{:}e_1) \cup \cdots \cup (m{:}s_m) \models_{bg} \mathrm{SM}[D_m;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl}) \cup (1{:}\sigma^{act}) \cup \cdots \cup (m{-}1{:}\sigma^{act}) \cup (m{:}\sigma^{fl})]$$

iff each triple $\langle s_i, e_i, s_{i+1}\rangle$ $(0 \le i < m)$ is a transition.

It is not difficult to check that ASPMT program D_m that is obtained from action description D is always tight. In view of Theorem 1, D_m can be represented in the language of SMT as the next section demonstrates.

4 Reasoning about Continuous Changes in $\mathcal{C}$+

In order to represent continuous changes in the enhanced $\mathcal{C}$+, we distinguish between steps and real clock times. We assume the theory of reals as the background theory, and introduce a simple fluent constant *Time* with value sort $\mathcal{R}_{\ge 0}$, which denotes the clock time, and an action constant *Dur* with value sort $\mathcal{R}_{\ge 0}$, which denotes the time elapsed between the two consecutive states. We postulate the following causal laws:

$$\begin{array}{l}\textbf{exogenous } \mathit{Time}, \mathit{Dur},\\ \textbf{constraint } \mathit{Time}{=}t+t' \textbf{ after } \mathit{Time}{=}t \wedge \mathit{Dur}{=}t'\ .\end{array} \tag{7}$$

These causal laws are shorthand for

$$\begin{array}{l}\textbf{caused } \mathit{Time}{=}t \textbf{ if } \mathit{Time}{=}t\ ,\\ \textbf{caused } \mathit{Dur}{=}t \textbf{ if } \mathit{Dur}{=}t\ ,\\ \textbf{caused } \bot \textbf{ if } \neg(\mathit{Time}{=}t+t') \textbf{ after } \mathit{Time}{=}t \wedge \mathit{Dur}{=}t'\end{array}$$

where t, t' are variables of sort $\mathcal{R}_{\ge 0}$. (See Appendix B in [Giunchiglia *et al.*, 2004] for the abbreviations of causal laws.)

Continuous changes can be described as a function of duration using fluent dynamic laws of the form

$$\textbf{caused } c{=}f(\mathbf{x},\mathbf{x}',t) \textbf{ if } \mathbf{c}'{=}\mathbf{x}' \textbf{ after } (\mathbf{c}{=}\mathbf{x}) \wedge (\mathit{Dur}{=}t) \wedge G$$

Notation: d, v, v', t, t' are variables of sort $\mathcal{R}_{\ge 0}$
A, MS are real numbers.

Simple fluent constants:	Domains:
Speed, *Distance*, *Time*	$\mathcal{R}_{\ge 0}$
Action constants:	Domains:
Accelerate, *Decelerate*	Boolean
Dur	$\mathcal{R}_{\ge 0}$

$\textbf{caused } \mathit{Speed}{=}v{+}\mathrm{A}{\times}t \textbf{ after } \mathit{Accelerate} \wedge \mathit{Speed}{=}v \wedge \mathit{Dur}{=}t$
$\textbf{caused } \mathit{Speed}{=}v{-}\mathrm{A}{\times}t \textbf{ after } \mathit{Decelerate} \wedge \mathit{Speed}{=}v \wedge \mathit{Dur}{=}t$
$\textbf{caused } \mathit{Distance}{=}d{+}0.5{\times}(v{+}v'){\times}t \textbf{ if } \mathit{Speed}{=}v'$
$\qquad\qquad \textbf{after } \mathit{Distance}{=}d \wedge \mathit{Speed}{=}v \wedge \mathit{Dur}{=}t$
$\textbf{constraint } \mathit{Time}{=}t{+}t' \textbf{ after } \mathit{Time}{=}t \wedge \mathit{Dur}{=}t'$
$\textbf{constraint } \mathit{Speed} \le \mathrm{MS}$

$\textbf{inertial } \mathit{Speed}$
$\textbf{exogenous } \mathit{Time}, c$ for every action constant c

Figure 1: Car Example in $\mathcal{C}$+

where (i) c is a simple fluent constant, (ii) $\mathbf{c}$, $\mathbf{c}'$ are lists of fluent constants, (iii) $\mathbf{x}$, $\mathbf{x}'$ are lists of object variables, (iv) G is a formula, and (v) $f(\mathbf{x},\mathbf{x}',t)$ is a term constructed from σ^{bg}, and variables in $\mathbf{x}$, $\mathbf{x}'$, and t.

For instance, the fluent dynamic law

$$\begin{array}{l}\textbf{caused } \mathit{Distance}{=}d{+}0.5{\times}(v{+}v'){\times}t \textbf{ if } \mathit{Speed}{=}v'\\ \quad\textbf{after } \mathit{Distance}{=}d \wedge \mathit{Speed}{=}v \wedge \mathit{Dur}{=}t\end{array}$$

describes how fluent *Distance* changes according to the function of real time.

Consider the following problem by Lifschitz (http://www.cs.utexas.edu/vl/tag/continuous_problem).
If the accelerator of a car is activated, the car will speed up with constant acceleration A until the accelerator is released or the car reaches its maximum speed MS, whichever comes first. If the brake is activated, the car will slow down with acceleration $-$A until the brake is released or the car stops, whichever comes first. Otherwise, the speed of the car remains constant. The problem asks to find a plan satisfying the following condition: at time 0, the car is at rest at one end of the road; at time K, it should be at rest at the other end.

A $\mathcal{C}$+ description of this example is shown in Figure 1. The

Notation: x, d, v, v', t, t' are variables of sort $\mathcal{R}_{\ge 0}$;
y is a variable of sort Boolean.
Intensional object constants: $i{:}\mathit{Speed}$ for $i > 0$

$i{+}1{:}\mathit{Speed}{=}x \leftarrow (x = v{+}\mathrm{A}{\times}t) \wedge i{:}(\mathit{Accelerate} \wedge \mathit{Speed}{=}v \wedge \mathit{Dur}{=}t)$
$i{+}1{:}\mathit{Speed}{=}x \leftarrow (x = v{-}\mathrm{A}{\times}t) \wedge i{:}(\mathit{Decelerate} \wedge \mathit{Speed}{=}v \wedge \mathit{Dur}{=}t)$
$i{+}1{:}\mathit{Distance}{=}x \leftarrow (x = d{+}0.5{\times}(v'{+}v){\times}t) \wedge i{+}1{:}\mathit{Speed}{=}v' \wedge i : (\mathit{Speed}{=}v \wedge \mathit{Distance}{=}d \wedge \mathit{Dur}{=}t)$
$\bot \leftarrow \neg(i{+}1{:}\mathit{Time}{=}t{+}t') \wedge i : (\mathit{Time}{=}t \wedge \mathit{Dur}{=}t')$
$\bot \leftarrow \neg(i{:}\mathit{Speed} \le \mathrm{MS})$
$i{+}1{:}\mathit{Speed}{=}x \leftarrow \neg\neg(i{+}1{:}\mathit{Speed}{=}x) \wedge i{:}\mathit{Speed}{=}x$
$i{:}\mathit{Time}{=}t \leftarrow \neg\neg(i{:}\mathit{Time}{=}t)$
$i{:}c{=}y \leftarrow \neg\neg(i{:}c{=}y)$ for every action constant c

Figure 2: Car Example in ASPMT

Figure 3: Solution Found by iSAT

actions *Accelerate* and *Decelerate* has direct effects on *Speed* and indirect effects on *Distance*. For an object constant c that has the Boolean domain, we abbreviate $c = \text{TRUE}$ as c and $c = \text{FALSE}$ as $\sim c$. According to the semantics in Section 3.2, the description is turned into an ASPMT program with the theory of reals as the background theory, which can be further rewritten in Clark normal form. Some occurrences of $\neg\neg$ can be dropped without affecting stable models, which results in the program in Figure 2.

The program can be viewed as $F \wedge G$ where F is the conjunction of the rules that has intensional constants in the heads, and G is the conjunction of the rest rules. In view of Theorem 2, the stable models and $F \wedge G$ are the same as the stable models of F that satisfies G. By Theorem 1, F can be turned into completion. For example, the completion on $i{+}1{:}\mathit{Speed}$ yields a formula that is equivalent to

$$\begin{aligned} i{+}1{:}\mathit{Speed} = x \leftrightarrow \big(&x = (i{:}\mathit{Speed} + A \times i{:}\mathit{Dur}) \ \wedge \ i{:}\mathit{Accelerate}\big) \\ \vee \big(&x = (i{:}\mathit{Speed} - A \times i{:}\mathit{Dur}) \ \wedge \ i{:}\mathit{Decelerate}\big) \\ \vee \big(&i{+}1{:}\mathit{Speed} = x \ \wedge \ i{:}\mathit{Speed} = x\big). \end{aligned}$$

Variable x in the formula can be eliminated by equivalent transformations using equality:

$$\begin{aligned} &i{:}\mathit{Accelerate} \rightarrow i{+}1{:}\mathit{Speed} = (i{:}\mathit{Speed} + A \times i{:}\mathit{Dur}) \\ &i{:}\mathit{Decelerate} \rightarrow i{+}1{:}\mathit{Speed} = (i{:}\mathit{Speed} - A \times i{:}\mathit{Dur}) \\ &(i{+}1 : \mathit{Speed} = (i{:}\mathit{Speed} + A \times i{:}\mathit{Dur}) \wedge i{:}\mathit{Accelerate}) \\ &\quad \vee (i{+}1 : \mathit{Speed} = (i{:}\mathit{Speed} - A \times i{:}\mathit{Dur}) \wedge i{:}\mathit{Decelerate}) \\ &\quad \vee (i{:}\mathit{Speed} = i{+}1{:}\mathit{Speed}) \ . \end{aligned}$$

The whole translation can be encoded in the input language of SMT solvers. The shortest plan found by iSAT (http://isat.gforge.avacs.org) on this input formula when the road length is 10, $\mathrm{A} = 3, \mathrm{MS} = 4, \mathrm{K} = 4$ is shown in Figure 3.

5 Reasoning about Additive Continuous Resources in $\mathcal{C}+$

Additive fluents are fluents with numerical values such that the effect of several concurrently executed actions on it can be computed by adding the effects of the individual actions. Lee and Lifschitz [2003] show how to describe additive fluents in $\mathcal{C}+$ by understanding "increment laws" as shorthand for some causal laws. However, some additive fluents are real-valued, and cannot be represented in the language described in [Lee and Lifschitz, 2003] as the language is limited to finite domains only. This made the discussion of additive fluents in [Lee and Lifschitz, 2003] limited to integer domains only. For example, the effect of firing multiple jets on the velocity of a spacecraft is described by "increment laws"

$$\mathit{Fire}(j) \ \textbf{increments} \ \mathit{Vel}(ax) \ \textbf{by} \ n//\mathit{Mass} \ \textbf{if} \ \mathit{Force}(j, ax) = n \ .$$

Mass stands for an integer constant, and the symbol $//$ stands for integer division; the duration of firing action is assumed to be 1, and all components of the position and the velocity vectors at any time are assumed to be integers, and even the forces applied have to be integers. Obviously these are too strong assumptions.

These limitations are not present in our enhanced $\mathcal{C}+$ and its SMT-based computation. The representation in [Lee and Lifschitz, 2003] can be straightforwardly extended to handle continuous motions by distinguishing between steps and real time as in the previous section. For example, we can describe the effect that firing multiple jets has on the velocity of a spacecraft by

$$\begin{aligned} \mathit{Fire}(j) \ \textbf{increments} \ \mathit{Vel}(ax) \ \textbf{by} \ (n/\mathit{Mass}) \times t \\ \textbf{if} \ \mathit{Force}(j, ax) = n \wedge \mathit{Dur} = t \end{aligned}$$

In general, additive fluent constants are simple fluent constants with numerical values with certain restrictions. First, the heads of static and fluent dynamic laws are not allowed to contain additive fluent constants. Second, the effects of concurrent execution of actions on additive fluents are expressed by *increment laws*—expressions of the form

$$a \ \textbf{increments} \ c \ \textbf{by} \ f(\mathbf{x}, t) \ \textbf{if} \ (\mathbf{d}, \mathit{Dur}) = (\mathbf{x}, t) \wedge G \quad (8)$$

where (i) a is a Boolean action constant; (ii) c is an additive fluent constant; (iii) $\mathbf{d}$ is a list of fluent constants, and $\mathbf{x}$ is a list of corresponding variables; (iv) $f(\mathbf{x}, t)$ is an arithmetic expression over $\mathbf{x}$ and the duration t; (v) G is a formula that contains no Boolean action constants.

Similar to [Lee and Lifschitz, 2003], the semantics of increment laws is described by a translation that replaces increment laws with action dynamic laws (5) and fluent dynamic laws (6) using additional action constants. This translation largely repeats the one in [Lee and Lifschitz, 2003] and we omit the details due to lack of space.

The $\mathcal{C}+$ encoding of the spacecraft example is shown in Figure 4.

Notation: $j \in \{J_1, J_2\}$, $ax \in \{X, Y, Z\}$
x, y, y' are variables of sort $\mathcal{R}$; t, t' are variables of sort $\mathcal{R}_{\geq 0}$

Additive fluent constants:	Domains:
$\mathit{Vel}(ax)$	$\mathcal{R}$
Other Simple fluent constants:	Domains:
$\mathit{Pos}(ax)$	$\mathcal{R}$
Time	$\mathcal{R}_{\geq 0}$
Action constants:	Domains:
$\mathit{Fire}(j)$	Boolean
$\mathit{Force}(j, ax)$	$\mathcal{R}$
Dur	$\mathcal{R}_{\geq 0}$

$\mathit{Fire}(j)$ **increments** $\mathit{Vel}(ax)$ **by** $(x/\mathrm{Mass}) \times t$
if $\mathit{Force}(j, ax) = x \wedge \mathit{Dur} = t$
caused $\mathit{Pos}(ax) = x + (0.5 \times (y + y') \times t)$ **if** $\mathit{Vel}(ax) = y'$
after $\mathit{Pos}(ax) = x \wedge \mathit{Vel}(ax) = y \wedge \mathit{Dur} = t$
always $\mathit{Force}(j, ax) = 0 \ \leftrightarrow \sim \mathit{Fire}(j)$
constraint $\mathit{Time} = t + t'$ **after** $\mathit{Time} = t \wedge \mathit{Dur} = t'$
exogenous Time
exogenous c for every action constant c

Figure 4: Spacecraft Example in $\mathcal{C}+$ with Additive Fluents

Max Step	CCALC v2.0 using RELSAT v2.2		CPLUS2ASP v1.0 using GRINGO v3.0.3+CLASP v2.0.2		$\mathcal{C}$+ in iSAT v1.0	
	Run Time (grounding+completion+solving)	# of atoms / clauses	Run Time (grounding+solving)	# of atoms / rules	Run Time last/total	# of variables / clauses (bool + real)
1	0.16 (0.12+0.04+0.00)	488 / 1872	0.005 (0.005+0)	1864 / 2626	0/0	(42+53) / 182
2	0.57 (0.40+0.17+0.00)	3262 / 14238	0.033 (0.033+0)	6673 / 12035	0/0	(82+98) / 352
3	10.2 (2.62+1.58+6)	32772 / 155058	0.434 (0.234+0.2)	42778 / 92124	0/0	(122+143) / 520
4	505.86 (12.94+13.92+479)	204230 / 992838	12.546 (3.176+9.37)	228575/ 503141	0/0	(162+188) / 688
5	failed (51.10+115.58+ failed)	897016 / 4410186	73.066 (15.846+57.22)	949240/ 2060834	0/0.03	(202+233) / 856
6	time out	–	3020.851 (62.381+2958.47)	3179869/ 6790167	0/0.03	(242+278) / 1024
10	time out	–	time out	–	0.03/0.09	(402+458) / 1696
50	time out	–	time out	–	0.09/1.39	(2002+2258) / 8416
100	time out	–	time out	–	0.17/5.21	(4002+4508) / 16816
200	time out	–	time out	–	0.33/21.96	(8002+9008) / 33616

Table 1: Experiment results on Spacecraft Example (time out after 2 hours)

Table 1 compares the performance of SMT-based computation of $\mathcal{C}$+ and existing implementations of $\mathcal{C}$+: CCALC 2 and CPLUS2ASP. System CPLUS2ASP translates $\mathcal{C}$+ into ASP programs and use GRINGO and CLASP for computation. For the sake of comparison, we assume that the duration of each action is exactly 1 unit of time so that the plans found by the systems are of the same kind. We assume that initially the spacecraft is rest at coordinate $(0,0,0)$. The task is to find a plan such that at each integer time t, the spacecraft is at (t^2, t^2, t^2). The experiment was performed on an Intel Core 2 Duo CPU 3.00 GHz with 4 GB RAM running on Ubuntu version 11.10. It shows a clear advantage of the SMT-based computation of $\mathcal{C}$+ for this example.

Notation: $x \in \{\mathit{Tap}_1, \mathit{Tap}_2\}$; $\mathrm{W}(x)$, V, Low, High are real numbers; t, t' are variables of sort $\mathcal{R}_{\geq 0}$.

$\mathit{TurnOn}(x)$ **causes** $\mathit{On}(x) \wedge \mathit{Dur}=0$
$\mathit{TurnOff}(x)$ **causes** $\sim\mathit{On}(x) \wedge \mathit{Dur}=0$

$\mathit{On}(x)$ **increments** *Level* **by** $\mathrm{W}(x)\times t$ **if** $\mathit{Dur}=t$
Leaking **increments** *Level* **by** $-(\mathrm{V}\times t)$ **if** $\mathit{Dur}=t$

constraint $(\mathrm{Low} \leq \mathit{Level}) \wedge (\mathit{Level} \leq \mathrm{High})$
inertial $\mathit{On}(x)$, *Leaking*
exogenous c for every action constant c

exogenous *Time*
constraint $\mathit{Time}=t+t'$ **after** $\mathit{Time}=t \wedge \mathit{Dur}=t'$

Figure 5: Two Taps Water Tank Example $\mathcal{C}$+

The language $\mathcal{C}$+ is flexible enough to represent the *start-process-end* model [Reiter, 1996; Fox and Long, 2006], where instantaneous actions may initiate or terminate processes. Processes run over time and have a continuous effect on numeric fluents. They are initiated and terminated either by the direct action or by events that are triggered. This model can be encoded in $\mathcal{C}+$ by introducing *process fluents*, which are Boolean-valued. Such a process fluent is assumed to be inertial, and is caused to be true or false by instantaneous events. Once true, the process fluent p determines the changes of additive fluents c by increment laws

$$p \textbf{ increments } c \textbf{ by } f(\mathbf{x},t) \textbf{ if } (\mathbf{d}, \mathit{Dur}) = (\mathbf{x}, t) \wedge G\ .$$

Here, **increments** laws are defined similar to (8) except that p is a process fluent, instead of a Boolean action constant; we require that G contain no process fluents and no Boolean action constants. For example,

$$\mathit{On}(\mathit{Tap}_1) \textbf{ increments } \mathit{Level} \textbf{ by } \mathrm{W}(\mathit{Tap}_1)\times t \textbf{ if } \mathit{Dur}=t$$

specifies that, when the process fluent $\mathit{On}(\mathit{Tap}_1)$ is true, the process contributes to increasing the water level by $\mathrm{W}(\mathit{Tap}_1)$ multiplied by duration.

Figure 5 describes the level of a water tank that has two taps with different flow rates and possible leaking.

6 Related Work and Conclusion

Besides the functional stable model semantics we considered in this paper, there are other approaches to incorporate "intensional" functions in ASP [Cabalar, 2011; Lifschitz, 2012; Balduccini, 2012].

Action language $\mathcal{H}$ [Chintabathina *et al.*, 2005; Chintabathina and Watson, 2012] is similar to our enhanced $\mathcal{C}$+ in that it can handle reasoning about continuous changes. One notable difference is there, each state represents an interval of time, rather than a particular timepoint. Language $\mathcal{H}$ does not have action dynamic laws, and consequently does not allow additive fluents.

The durative action model of PDDL 2.1 [Fox and Long, 2003] is similar to our $\mathcal{C}$+ representation in Section 4. The start-process-stop model of representing continuous changes in PDDL+ [Fox and Long, 2006] is similar to the representation in Section 5. Our work is also related to [Shin and Davis, 2005], which used SMT solvers for computing PDDL+, and a recent work on planning modulo theories [Gregory *et al.*, 2012].

Continuous changes have also been studied in the situation calculus, the event calculus and temporal action logics. In [Lee and Palla, 2012b; 2012a], these action formalisms were reformulated in ASP. We expect that the techniques we developed in this paper can be applied for more effective computation of these formalisms using SMT solvers.

Acknowledgements: We are grateful to Michael Bartholomew for useful discussions related to this paper. We are also grateful to the anonymous referees for their useful comments. This work was partially supported by the National Science Foundation under Grant IIS-0916116 and by the South Korea IT R&D program MKE/KIAT 2010-TD-300404-001.

References

[Balduccini, 2009] Marcello Balduccini. Representing constraint satisfaction problems in answer set programming. In *Working Notes of the Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*, 2009.

[Balduccini, 2012] Marcello Balduccini. A "conservative" approach to extending answer set programming with nonherbrand functions. In *Correct Reasoning - Essays on Logic-Based AI in Hon our of Vladimir Lifschitz*, pages 24–39, 2012.

[Barrett *et al.*, 2009] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[Bartholomew and Lee, 2012] Michael Bartholomew and Joohyung Lee. Stable models of formulas with intensional functions. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 2–12, 2012.

[Bartholomew and Lee, 2013] Michael Bartholomew and Joohyung Lee. Functional stable model semantics and answer set programming modulo theories. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2013. This volume.

[Brewka *et al.*, 2011] Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[Cabalar, 2011] Pedro Cabalar. Functional answer set programming. *TPLP*, 11(2-3):203–233, 2011.

[Chintabathina and Watson, 2012] Sandeep Chintabathina and Richard Watson. A new incarnation of action language h. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning*, volume 7265 of *Lecture Notes in Computer Science*, pages 560–575. Springer, 2012.

[Chintabathina *et al.*, 2005] Sandeep Chintabathina, Michael Gelfond, and Richard Watson. Modeling hybrid domains using process description language[2]. In *Proceedings of Workshop on Answer Set Programming: Advances in Theory and Implementation (ASP'05)*, 2005.

[Ferraris *et al.*, 2011] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. Stable models and circumscription. *Artificial Intelligence*, 175:236–263, 2011.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

[Fox and Long, 2006] Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res. (JAIR)*, 27:235–297, 2006.

[Gebser *et al.*, 2009] M. Gebser, M. Ostrowski, and T. Schaub. Constraint answer set solving. In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 235–249, 2009.

[Giunchiglia *et al.*, 2004] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1–2):49–104, 2004.

[Gregory *et al.*, 2012] Peter Gregory, Derek Long, Maria Fox, and J. Christopher Beck. Planning modulo theories: Extending the planning paradigm. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

[Janhunen *et al.*, 2011] Tomi Janhunen, Guohua Liu, and Ilkka Niemel. Tight integration of non-ground answer set programming and satisfiability modulo theories. In *Working notes of the 1st Workshop on Grounding and Transformations for Theories with Variables*, 2011.

[Lee and Lifschitz, 2003] Joohyung Lee and Vladimir Lifschitz. Describing additive fluents in action language $\mathcal{C}$+. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1079–1084, 2003.

[Lee and Palla, 2012a] Joohyung Lee and Ravi Palla. Reformulating temporal action logics in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 786–792, 2012.

[Lee and Palla, 2012b] Joohyung Lee and Ravi Palla. Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. *Journal of Artificial Inteligence Research (JAIR)*, 43:571–620, 2012.

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

[Lifschitz, 2012] Vladimir Lifschitz. Logic programs with intensional functions. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 24–31, 2012.

[Niemelä, 2008] Ilkka Niemelä. Stable models and difference logic. *Ann. Math. Artif. Intell.*, 53(1-4):313–329, 2008.

[Reiter, 1996] Raymond Reiter. Natural actions, concurrency and continuous time in the situation calculus. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 2–13, 1996.

[Shin and Davis, 2005] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artificial Intelligence*, 166(1-2):194–253, 2005.

[2] `http://ceur-ws.org/vol-142/page303.pdf`