Action Language $\mathcal{BC}$: Preliminary Report

Joohyung Lee[1], **Vladimir Lifschitz**[2] and **Fangkai Yang**[2]
[1]School of Computing, Informatics and Decision Systems Engineering, Arizona State University
joolee@asu.edu
[2] Department of Computer Science, Univeristy of Texas at Austin
{vl,fkyang}@cs.utexas.edu

Abstract

The action description languages $\mathcal{B}$ and $\mathcal{C}$ have significant common core. Nevertheless, some expressive possibilities of $\mathcal{B}$ are difficult or impossible to simulate in $\mathcal{C}$, and the other way around. The main advantage of $\mathcal{B}$ is that it allows the user to give Prolog-style recursive definitions, which is important in applications. On the other hand, $\mathcal{B}$ solves the frame problem by incorporating the commonsense law of inertia in its semantics, which makes it difficult to talk about fluents whose behavior is described by defaults other than inertia. In $\mathcal{C}$ and in its extension $\mathcal{C}+$, the inertia assumption is expressed by axioms that the user is free to include or not to include, and other defaults can be postulated as well. This paper defines a new action description language, called $\mathcal{BC}$, that combines the attractive features of $\mathcal{B}$ and $\mathcal{C}+$. Examples of formalizing commonsense domains discussed in the paper illustrate the expressive capabilities of $\mathcal{BC}$ and the use of answer set solvers for the automation of reasoning about actions described in this language.

1 Introduction

Action description languages are formal languages for describing the effects and executability of actions. "Second generation" action description languages, such as $\mathcal{B}$ [Gelfond and Lifschitz, 1998, Section 5], $\mathcal{C}$ [Giunchiglia and Lifschitz, 1998], and $\mathcal{C}+$ [Giunchiglia *et al.*, 2004, Section 4], differ from the older languages STRIPS [Fikes and Nilsson, 1971] and ADL [Pednault, 1989] in that they allow us to describe indirect effects of an action—effects explained by interaction between fluents.

The languages $\mathcal{B}$ and $\mathcal{C}$ have significant common core [Gelfond and Lifschitz, 2012]. Nevertheless, some expressive possibilities of $\mathcal{B}$ are difficult or impossible to simulate in $\mathcal{C}$, and the other way around. The main advantage of $\mathcal{B}$ is that it allows the user to give Prolog-style recursive definitions. Recursively defined concepts, such as the reachability of a node in a graph, play important role in applications of automated reasoning about actions, including the design of the decision support system for the Space Shuttle [Nogueira *et al.*, 2001]. On the other hand, the language $\mathcal{B}$, like STRIPS and ADL, solves the frame problem by incorporating the commonsense law of inertia in its semantics, which makes it difficult to talk about fluents whose behavior is described by defaults other than inertia. The position of a moving pendulum, for instance, is a non-inertial fluent: it changes by itself, and an action is required to prevent the pendulum from moving. The amount of liquid in a leaking container changes by itself, and an action is required to prevent it from decreasing. A spring-loaded door closes by itself, and an action is required to keep it open. Work on the action language $\mathcal{C}$ and its extension $\mathcal{C}+$ was partly motivated by examples of this kind. In these languages, the inertia assumption is expressed by axioms that the user is free to include or not to include. Other default assumptions about the relationship between the values of a fluent at different time instants can be postulated as well. On the other hand, some recursive definitions cannot be easily expressed in $\mathcal{C}$ and $\mathcal{C}+$.

In this paper we define a new action description language, called $\mathcal{BC}$, that combines the attractive features of $\mathcal{B}$ and $\mathcal{C}+$. This language, like $\mathcal{B}$, can be implemented using computational methods of answer set programming [Marek and Truszczynski, 1999; Niemelä, 1999; Lifschitz, 2008].

The main difference between $\mathcal{B}$ and $\mathcal{BC}$ is similar to the difference between inference rules and default rules. Informally speaking, a default rule allows us to derive its conclusion from its premise if its justification can be consistently assumed; default logic [Reiter, 1980] makes this idea precise. In the language $\mathcal{B}$, a static law has the form

$$<\text{conclusion}> \textbf{ if } <\text{premise}> .$$

In $\mathcal{BC}$, a static law may include a justification:

$$<\text{conclusion}> \textbf{ if } <\text{premise}> \textbf{ if cons } <\text{justification}>$$

(**if cons** is an acronym for "if consistent"). Dynamic laws may include justifications also.

The semantics of $\mathcal{BC}$ is defined by transforming action descriptions into logic programs under the stable model semantics. When static and dynamic laws of the language $\mathcal{B}$ are translated into the language of logic programming, as in [Balduccini and Gelfond, 2003], the rules that we get do not contain negation as failure. Logic programs corresponding to $\mathcal{B}$-descriptions do contain negation as failure, but this is because inertia rules are automatically included in them. In the case of $\mathcal{BC}$, on the other hand, negation as failure is used for translating justifications in both static and dynamic laws.

We define here three translations from $\mathcal{BC}$ into logic programming. Their target languages use slightly different versions of the stable model semantics, but we show that all three translations give the same meaning to $\mathcal{BC}$-descriptions. The first version uses nested occurrences of negation as failure [Lifschitz *et al.*, 1999]; the second involves strong (classical) negation [Gelfond and Lifschitz, 1991] but does not require nesting; the third produces multi-valued formulas under the stable model semantics [Bartholomew and Lee, 2012]. The third translation is particularly simple, because $\mathcal{BC}$ and multi-valued formulas have much in common: both languages are designed for talking about non-Boolean fluents. But we start with defining the other two translations, because their target languages are more widely known.

Examples of formalizing commonsense domains discussed in this paper illustrate the expressive capabilities of $\mathcal{BC}$ and the use of answer set solvers for the automation of reasoning about actions described in this language. We state also two theorems relating $\mathcal{BC}$ to $\mathcal{B}$ and to $\mathcal{C}+$.

2 Syntax

An action description in the language $\mathcal{BC}$ includes a finite set of symbols of two kinds, *fluent constants* and *action constants*. Fluent constants are further divided into *regular* and *statically determined*. A finite set of cardinality ≥ 2, called the *domain*, is assigned to every fluent constant.

An *atom* is an expression of the form $f = v$, where f is a fluent constant, and v is an element of its domain. If the domain of f is $\{\mathbf{f}, \mathbf{t}\}$ then we say that f is *Boolean*.

A *static law* is an expression of the form

$$A_0 \textbf{ if } A_1, \ldots, A_m \textbf{ if cons } A_{m+1}, \ldots, A_n \tag{1}$$

($n \geq m \geq 0$), where each A_i is an atom. It expresses, informally speaking, that every state satisfies A_0 if it satisfies $A_1, \ldots, A_m$, and $A_{m+1}, \ldots, A_n$ can be consistently assumed. If $m = 0$ then we will drop **if**; if $m = n$ then we will drop **if cons**.

A *dynamic law* is an expression of the form

$$A_0 \textbf{ after } A_1, \ldots, A_m \textbf{ if cons } A_{m+1}, \ldots, A_n \tag{2}$$

($n \geq m \geq 0$), where

- A_0 is an atom containing a regular fluent constant,
- each of $A_1, \ldots, A_m$ is an atom or an action constant, and
- $A_{m+1}, \ldots, A_n$ are atoms.

It expresses, informally speaking, that the end state of any transition satisfies A_0 if its beginning state and its action satisfy $A_1, \ldots, A_m$, and $A_{m+1}, \ldots, A_n$ can be consistently assumed about the end state. If $m = n$ then we will drop **if cons**.

For any action constant a and atom A,

$$a \textbf{ causes } A$$

stands for

$$A \textbf{ after } a.$$

For any action constant a and atoms $A_0, \ldots, A_m$ ($m > 0$),

$$a \textbf{ causes } A_0 \textbf{ if } A_1, \ldots, A_m$$

stands for

$$A_0 \textbf{ after } a, A_1, \ldots, A_m.$$

An *action description* in the language $\mathcal{BC}$ is a finite set consisting of static and dynamic laws.

3 Defaults and Inertia

Static laws of the form

$$A_0 \textbf{ if } A_1, \ldots, A_m \textbf{ if cons } A_0 \tag{3}$$

and dynamic laws of the form

$$A_0 \textbf{ after } A_1, \ldots, A_m \textbf{ if cons } A_0 \tag{4}$$

will be particularly useful. They are similar to normal defaults in the sense of [Reiter, 1980]. We will write (3) as

$$\textbf{default } A_0 \textbf{ if } A_1, \ldots, A_m,$$

and we will drop **if** when $m = 0$. We will write (4) as

$$\textbf{default } A_0 \textbf{ after } A_1, \ldots, A_m.$$

For any regular fluent constant f, the set of the dynamic laws

$$\textbf{default } f = v \textbf{ after } f = v$$

for all v in the domain of f expresses the commonsense law of inertia for f. We will denote this set by

$$\textbf{inertial } f. \tag{5}$$

4 Semantics

For every action description D, we will define a sequence of logic programs with nested expressions $PN_0(D), PN_1(D), \ldots$ so that the stable models of $PN_l(D)$ represent paths of length l in the transition system corresponding to D. The signature $\sigma_{D,l}$ of $PN_l(D)$ consists of

- expressions $i : A$ for nonnegative integers $i \leq l$ and all atoms A, and
- expressions $i : a$ for nonnegative integers $i < l$ and all action constants a.

Thus every element of the signature $\sigma_{D,l}$ is a "time stamp" i followed by an atom in the sense of Section 2 or by an action constant. The program consists of the following rules:

- the translations

  $$i{:}A_0 \leftarrow i{:}A_1, \ldots, i{:}A_m, \\ \quad \textit{not not } i{:}A_{m+1}, \ldots, \textit{not not } i{:}A_n$$

  ($i \leq l$) of all static laws (1) from D,
- the translations

  $$(i+1){:}A_0 \leftarrow i{:}A_1, \ldots, i{:}A_m, \\ \quad \textit{not not } (i+1){:}A_{m+1}, \ldots, \textit{not not } (i+1){:}A_n$$

  ($i < l$) of all dynamic laws (2) from D,
- the choice rule[1] $\{0 : A\}$ for every atom A containing a regular fluent constant,

[1] A choice rule $\{E\}$ can be viewed as shorthand for the disjunctive nested expression $E; \textit{not } E$ [Ferraris and Lifschitz, 2005], or, equivalently, for the rule

$$E \leftarrow \textit{not not } E.$$

- the choice rule $\{i : a\}$ for every action constant a and every $i < l$,
- the existence of value constraint

$$\leftarrow \mathit{not}\ i{:}(f{=}v_1), \ldots, \mathit{not}\ i{:}(f{=}v_k)$$

for every fluent constant f and every $i \leq l$, where $v_1, \ldots, v_k$ are all elements of the domain of f,

- the uniqueness of value constraint

$$\leftarrow i{:}(f{=}v), i{:}(f{=}w)$$

for every fluent constant f, every pair of distinct elements v, w of its domain, and every $i \leq l$.

The transition system $T(D)$ represented by an action description D is defined as follows. For every stable model X of $PN_0(D)$, the set of atoms A such that $0 : A$ belongs to X is a state of $T(D)$. In view of the existence of value and uniqueness of value constraints, for every state s and every fluent constant f there exists exactly one v such that $f = v$ belongs to s; this v is considered the value of f in state s. For every stable model X of $PN_1(D)$, $T(D)$ includes the transition $\langle s_0, \alpha, s_1\rangle$, where s_i $(i = 0, 1)$ is the set of atoms A such that $i{:}A$ belongs to X, and α is the set of action constants a such that $0{:}a$ belongs to X.

The soundness of this definition is guaranteed by the following fact:

Theorem 1 *For every transition $\langle s_0, \alpha, s_1\rangle$, s_0 and s_1 are states.*

We promised that stable models of $PN_l(D)$ would represent paths of length l in the transition system corresponding to D. For $l = 0$ and $l = 1$, this is clear from the definition of $T(D)$; for $l > 1$ this needs to be verified. For every set X of elements of the signature $\sigma_{D,l}$, let X^i $(i < l)$ be the triple consisting of

- the set of atoms A such that $i{:}A$ belongs to X,
- the set of action constants a such that $i{:}a$ belongs to X, and
- the set of atoms A such that $(i+1){:}A$ belongs to X.

Theorem 2 *For every $l \geq 1$, X is a stable model of $PN_l(D)$ iff $X^0, \ldots, X^{l-1}$ are transitions.*

The rules contributed to $PN_l(D)$ by static law (3) have the form

$$i{:}A_0 \leftarrow i{:}A_1, \ldots, i{:}A_m, \mathit{not\ not}\ i{:}A_0.$$

They can be equivalently rewritten as

$$\{i{:}A_0\} \leftarrow i{:}A_1, \ldots, i{:}A_m$$

(see [Lifschitz *et al.*, 2001]). Similarly, the rules contributed to $PN_l(D)$ by dynamic law (4) have the form

$$(i+1){:}A_0 \leftarrow i{:}A_1, \ldots, i{:}A_m, \mathit{not\ not}\ (i+1){:}A_0.$$

They can be equivalently rewritten as

$$\{(i+1){:}A_0\} \leftarrow i{:}A_1, \ldots, i{:}A_m.$$

In particular, the rules contributed by the commonsense law of inertia (5) can be rewritten as

$$\{(i+1){:}f = v\} \leftarrow i{:}f = v.$$

5 Other Abbreviations

In $\mathcal{BC}$-descriptions that involve Boolean fluent constants we will use abbreviations similar to those established for multi-valued formulas in [Giunchiglia *et al.*, 2004, Section 2.1]: if f is Boolean then we will write the atom $f = \mathbf{t}$ as f, and the atom $f = \mathbf{f}$ as $\sim f$.

A *static constraint* is a pair of static laws of the form

$$\begin{array}{l} f{=}v \ \textbf{if}\ A_1, \ldots, A_m \\ f{=}w \ \textbf{if}\ A_1, \ldots, A_m \end{array} \tag{6}$$

where $v \neq w$ and $m > 0$. We will write (6) as

$$\textbf{impossible}\ A_1, \ldots, A_m.$$

The use of this abbreviation depends on the fact that the choice of f, v, and w in (6) is inessential, in the sense of Theorem 3 below. About action descriptions D_1 and D_2 we say that they are *strongly equivalent* to each other if, for any action description D (possibly of a larger signature), $T(D \cup D_1) = T(D \cup D_2)$. This is similar to the definition of strong equivalence for logic programs [Lifschitz *et al.*, 2001].

Theorem 3 *Any two static constraints (6) with the same atoms $A_1, \ldots, A_m$ are strongly equivalent to each other.*

The rules contributed to $PN_l(D)$ by (6) can be equivalently written as

$$\bot \leftarrow i{:}A_1, \ldots, i{:}A_m.$$

A *dynamic constraint* is a pair of dynamic laws of the form

$$\begin{array}{l} f{=}v \ \textbf{after}\ a_1, \ldots, a_k, A_1, \ldots, A_m \\ f{=}w \ \textbf{after}\ a_1, \ldots, a_k, A_1, \ldots, A_m \end{array} \tag{7}$$

where $v \neq w$, $a_1, \ldots, a_k$ $(k > 0)$ are action constants, and $A_1, \ldots, A_m$ are atoms. We will write (7) as

$$\textbf{nonexecutable}\ a_1, \ldots, a_k\ \textbf{if}\ A_1, \ldots, A_m,$$

and we will drop **if** in this abbreviation when $m = 0$. The use of this abbreviation depends on the following fact:

Theorem 4 *Any two dynamic constraints (7) with the same action constants $a_1, \ldots, a_k$ and the same atoms $A_1, \ldots, A_m$ are strongly equivalent to each other.*

The rules contributed to $PN_l(D)$ by (7) can be equivalently written as

$$\bot \leftarrow i{:}a_1, \ldots, i{:}a_k, i{:}A_1, \ldots, i{:}A_m.$$

6 Example: The Blocks World

The description of the blocks world below ensures that every block belongs to a tower that rests on the table; there are no blocks or groups of blocks "floating in the air."

Let *Blocks* be a finite non-empty set of symbols (block names) that does not include the symbol *Table*. The action description below uses the following fluent and action constants:

- for each $B \in$ *Blocks*, regular fluent constant *Loc*(B) with domain *Blocks* $\cup \{$*Table*$\}$, and statically determined Boolean fluent constant *InTower*(B);
- for each $B \in$ *Blocks* and each $L \in$ *Blocks* $\cup \{$*Table*$\}$, action constant *Move*(B, L).

In the list of static and dynamic laws, B, B_1 and B_2 are arbitrary elements of *Blocks*, and L is an arbitrary element of *Blocks* $\cup$ {*Table*}. Two different blocks cannot rest on the same block:

impossible $Loc(B_1) = B, Loc(B_2) = B \qquad (B_1 \neq B_2).$

The definition of $InTower(B)$:

$InTower(B)$ **if** $Loc(B) = Table$,
$InTower(B)$ **if** $Loc(B) = B_1, InTower(B_1)$,
default $\sim InTower(B)$.

Blocks don't float in the air:

impossible $\sim InTower(B)$.

The commonsense law of inertia:

inertial $Loc(B)$.

The effect of moving a block:

$Move(B, L)$ **causes** $Loc(B) = L$.

A block cannot be moved unless it is clear:

nonexecutable $Move(B, L)$ **if** $Loc(B_1) = B$.

Here is a representation of logic programs $PN_l(D)$ (Section 4), for this action description D, in the input language of the grounder GRINGO:[2]

```
% declarations of variables for steps,
% blocks, and locations
step(0..l).
#domain step(I).
block(b(1..n)).
#domain block(B).
#domain block(B1).
#domain block(B2).
location(X) :- block(X).
location(table).
#domain location(L).

% translations of static laws
:- loc(B1,B,I), loc(B2,B,I), B1!=B2.
intower(B,true,I) :- loc(B,table,I).
intower(B,true,I) :- loc(B,B1,I),
            intower(B1,true,I).
{intower(B,false,I)}.
:- intower(B,false,I).

% translations of dynamic laws
{loc(B,L,I+1)} :- loc(B,L,I), I<l.
loc(B,L,I+1) :- move(B,L,I), I<l.
:- move(B,L,I), loc(B1,B,I), I<l.

% standard choice rules
{loc(B,L,0)}.
{move(B,L,I)} :- I<l.

% uniqueness and existence of value
:- not 1{loc(B,LL,I) : location(LL)}1.
:- not 1{intower(B,false,I),
       intower(B,true,I)}1.
```

[2] http://potassco.sourceforge.net/

The values of the symbolic constants `l` (the number of steps) and `n` (the number of blocks) are supposed to be specified in command line. The stable models generated by an answer set solver for this input file will represent all trajectories of length `l` in the transition system corresponding to the blocks world with `n` blocks. For instance, if we ground this program with the GRINGO options `-c l=0 -c n=3` then the resulting program will have 13 stable models, corresponding to to all possible configurations of 3 blocks.

The rules involving `intower` can be written more economically if we use strong (classical) negation and replace

```
intower(B,true,I), intower(B,false,I)
```

with

```
intower(B,I), -intower(B,I).
```

That would make the uniqueness of value constraint for `intower` redundant.

7 Example: A Leaking Container

The example above includes the inertiality assumption for all regular fluents. In some cases, the commonsense law of inertia for a regular fluent is not acceptable and needs to be replaced by a different default.

Consider, for instance, a container of capacity n that has a leak, so that it loses k units of liquid per unit of time, unless more liquid is added. This domain can be described using the regular fluent constants *Amt* with domain $\{0, \ldots, n\}$, for the amount of liquid in the container, and the action constant *FillUp*. There are two dynamic laws:

default $Amt = \max(a - k, 0)$ **after** $Amt = a \ \ (a = 0, \ldots, n)$,
FillUp **causes** $Amt = n$.

(When $k = 0$, the first of them turns into **inertial** *Amt*.)

Consider the following temporal projection problem involving this domain, with $n = 10$ and $k = 3$: initially the container is full, and it is filled up at time 3; we would like to know how the amount of liquid in the container will change with time. The program below consists of the rules of $PN_l(D)$ and rules encoding the temporal projection problem.

```
% declarations of variables for steps
% and amounts
step(0..l).
#domain step(I).
amount(0..n).
#domain amount(A).

% translations of dynamic laws
{amt(AA,I+1)} :- amt(A,I),
         AA=(|A-k|+(A-k))/2, I<l.
amt(n,I+1) :- fillup(I), I<l.

% standard choice rules
{amt(A,0)}.
{fillup(I)} :- I<l.

% uniqueness and existence of value
:- not 1 {amt(AA,I) : amount(AA)} 1.
```

```
% temporal projection
amt(n,0).
fillup(3).  -fillup(0..2;4..l).

#hide.  #show amt/2.
```

The solver CLINGO produces the following output:

```
amt(10,0) amt(10,4) amt(7,5) amt(7,1)
amt(4,2) amt(4,6) amt(1,7) amt(1,3)
amt(0,9) amt(0,8)
```

8 Translation into the Language of Programs with Strong Negation

In the definition of the semantics of $\mathcal{BC}$ in Section 4 the programs $PN_l(D)$ can be replaced by the programs with strong negation $PS_l(D)$ that consist of the following rules:

- the translations

  $i{:}A_0 \leftarrow i{:}A_1, \dots, i{:}A_m, \mathit{not}\ \neg\, i{:}A_{m+1}, \dots, \mathit{not}\ \neg\, i{:}A_n$

  ($i \leq l$) of all static laws (1) from D,
- the translations

  $(i+1){:}A_0 \leftarrow i{:}A_1, \dots, i{:}A_m,$
  $\mathit{not}\ \neg\,(i+1){:}A_{m+1}, \dots, \mathit{not}\ \neg\,(i+1){:}A_n$

  ($i < l$) of all dynamic laws (2) from D,
- the disjunctive rules $0{:}A\ \vee \neg\, 0{:}A$ for every atom A containing a regular fluent constant,
- the disjunctive rules $i{:}a\ \vee \neg\, i{:}a$ for every action constant a and every $i < l$,
- the existence of value constraint

  $$\leftarrow \mathit{not}\ i{:}(f{=}v_1), \dots, \mathit{not}\ i{:}(f{=}v_k)$$

  for every fluent constant f and every $i \leq l$, where $v_1, \dots, v_k$ are all elements of the domain of f,
- the uniqueness of value rule

  $$\neg\, i{:}(f{=}v) \leftarrow i{:}(f{=}w)$$

  for every fluent constant f, every pair of distinct elements v, w of its domain, and every $i \leq l$.

The stable models of the program $PN_l(D)$ from Section 4 can be obtained from the (complete) answer sets of $PS_l(D)$ by removing all negative literals:

Theorem 5 *A set X of atoms of the signature $\sigma_{D,l}$ is a stable model of $PN_l(D)$ iff $X \cup \{\neg A \mid A \in \sigma_{D,l} \setminus X\}$ is an answer set of $PS_l(D)$.*

It follows that the translation *PN* in the definition of $T(D)$ can be replaced with the translation *PS*.

9 Translation into the Language of Multi-Valued Formulas

Multi-valued formulas are defined in [Giunchiglia *et al.*, 2004, Section 2.1], and the stable model semantics is extended to such formulas in [Bartholomew and Lee, 2012].

A *multi-valued signature* is a set σ of symbols, called *constants*, along with a nonempty finite set $\mathit{Dom}(c)$ of symbols, disjoint from σ, assigned to each constant c, called the *domain* of c. An *atom* of the signature σ is an expression of the form $c = v$ ("the value of c is v"), where $c \in \sigma$ and $v \in \mathit{Dom}(c)$. If $\mathit{Dom}(c)$ is $\{\mathbf{f}, \mathbf{t}\}$ then we say that the constant c is *Boolean*. A *multi-valued formula* is a propositional combination of atoms. (Note that the symbol $\neg$ in multi-valued formulas corresponds to negation as failure in logic programs.)

A *multi-valued interpretation* of σ is a function that maps every element of σ to an element of its domain. An interpretation I *satisfies* an atom $c = v$ if $I(c) = v$. The satisfaction relation is extended from atoms to arbitrary formulas according to the usual truth tables for the propositional connectives.

The *reduct* F^I of a multi-valued formula F relative to a multi-valued interpretation I is the formula obtained from F by replacing each maximal subformula that is not satisfied by I with $\bot$. We say that I is a *stable model* of F if I is the only interpretation satisfying F^I.[3]

Consider the multi-valued signature consisting of

- the constants $i{:}f$ for nonnegative integers $i \leq l$ and all fluent constants f, with the same domain as f, and
- the Boolean constants $i{:}a$ for nonnegative integers $i < l$ and all action constants a.

If F is a propositional combination of atoms $f{=}v$ and action constants then $i{:}F$ stands for the formula of this signature obtained from F by prepending $i{:}$ to every fluent constant and to every action constant.

For any action description D, by $MV_l(D)$ we denote the conjunction of the following multi-valued formulas:

- the translations

  $$i{:}(A_1 \wedge \dots \wedge A_m \wedge \neg\neg A_{m+1} \wedge \dots \wedge \neg\neg A_n \rightarrow A_0)$$

  ($i \leq l$) of all static laws (1) from D,
- the translations

  $$i{:}(A_1 \wedge \dots \wedge A_m) \wedge (i{+}1){:}(\neg\neg A_{m+1} \wedge \dots \wedge \neg\neg A_n) \rightarrow (i+1){:}A_0$$

  ($i < l$) of all dynamic laws (2) from D,
- the formula $0{:}(f = v \vee f \neq v)$ for every regular fluent constant f and every element v of its domain,
- the formula $i{:}(a{=}\mathbf{t} \vee a{=}\mathbf{f})$ for every action constant a and every $i < l$.

By σ^A we denote the set of all action constants.

Theorem 6 *A set X of atoms of the signature $\sigma_{D,l}$ is a stable model of $PN_l(D)$ iff $X \cup \{i{:}a{=}\mathbf{f} \mid a \in \sigma^A,\ i < l,\ i{:}a \notin X\}$ is a stable model of $MV_l(D)$.*

It follows that the translation *PN* in the definition of $T(D)$ can be replaced with the translation *MV*.

[3] This formulation is based on the characterization of the stable model semantics of multi-valued formulas given by [Bartholomew and Lee, 2012, Theorem 5].

10 Relation to $\mathcal{B}$

The version of the action language $\mathcal{B}$ referred to in this section is defined in [Gelfond and Lifschitz, 2012]. For any action description D in the language $\mathcal{B}$, by $D^{\neg}_{\sim}$ we denote the result of replacing each negative literal $\neg f$ in D with the atom $\sim f$ (that is, $f = \mathbf{f}$). The abbreviations introduced in Sections 2 and 5 above allow us to view $D^{\neg}_{\sim}$ as an action description in the sense of $\mathcal{BC}$, provided that all fluent constants are treated as regular Boolean. We define the translation of D into $\mathcal{BC}$ as the result of extending $D^{\neg}_{\sim}$ by adding the inertiality assumptions (5) for all fluent constants f.

We will loosely refer to states and transitions of the transition system represented by D as states and transitions of D.

To state the claim that this translation preserves the meaning of D, we need to relate states and transitions in the sense of the semantics of $\mathcal{B}$ to states and transitions in the sense of Section 4. In $\mathcal{B}$, a state is a consistent and complete set of literals f, $\neg f$ for fluent constants f. For any set s of atoms f, $\sim f$, by $s^{\sim}_{\neg}$ we denote the set of literals obtained from s by replacing each atom $\sim f$ with the negative literal $\neg f$. Furthermore, an action in $\mathcal{B}$ is a consistent and complete set of literals a, $\neg a$ for action constants a.

Theorem 7 *For any action description D in the language $\mathcal{B}$,*

(a) a set s of atoms is a state of the translation of D into the language $\mathcal{BC}$ iff $s^{\sim}_{\neg}$ is a state of D;

(b) for any sets s_0, s_1 of atoms and any set α of action constants, $\langle s_0, \alpha, s_1\rangle$ is a transition of the translation of D into the language $\mathcal{BC}$ iff

$$\langle (s_0)^{\sim}_{\neg}, \alpha \cup \{\neg 0 : a \mid a \notin \alpha\}, (s_1)^{\sim}_{\neg}\rangle$$

is a transition of D.

The description of the blocks world from Section 6 does not correspond to any $\mathcal{B}$-description, in the sense of this translation, for two reasons. First, some fluent constants in it are not regular: it uses statically determined fluents *InTower*(B), defined recursively in terms of *Loc*(B). They are similar to "defined fluents" allowed in the extension of $\mathcal{B}$ introduced in [Gelfond and Inclezan, 2009]. Second, some fluent constants in it are not Boolean: the values of *Loc*(B) are locations.

The leaking container example (Section 7) does not correspond to any $\mathcal{B}$-description either: the regular fluent *Amt* is not Boolean, and the default describing how the value of this fluent changes is different from the commonsense law of inertia. An alternative approach to describing the leaking container is based on an extension of $\mathcal{B}$ by "process fluents," called $\mathcal{H}$ [Chintabathina *et al.*, 2005].

11 Relation to $\mathcal{C}+$

The semantics of $\mathcal{C}+$ is based on the idea of universal causation [McCain and Turner, 1997]. Formal relationships between universal causation and stable models are investigated in [McCain, 1997; Ferraris *et al.*, 2012], and it is not surprising that a large fragment of $\mathcal{BC}$ is equivalent to a large fragment of $\mathcal{C}+$.

In $\mathcal{C}+$, just as in $\mathcal{BC}$, some fluent symbols can be designated as "statically determined." (Other fluents are called "simple" in $\mathcal{C}+$; they correspond to regular fluents in our terminology.) Fluent symbols in $\mathcal{C}+$ may be non-exogenous; in our first version of $\mathcal{BC}$ such fluents are not allowed. Action symbols in $\mathcal{C}+$ may be non-Boolean; in this respect, that language is more general than the version of $\mathcal{BC}$ defined above.

Consider a $\mathcal{BC}$-description such that, in each of its static laws (1), $m = 0$. In other words, we assume that every static law has the form

$$A_0 \ \mathbf{ifcons}\ A_1, \dots, A_n. \tag{8}$$

Such a description can be translated into $\mathcal{C}+$ as follows:

- all action constants are treated as Boolean;
- every static law (8) is replaced with

$$\mathbf{caused}\ A_0\ \mathbf{if}\ A_1 \wedge \dots \wedge A_n;$$

- every dynamic law (2) is replaced with

$$\mathbf{caused}\ A_0\ \mathbf{if}\ A_{m+1} \wedge \dots \wedge A_n\ \mathbf{after}\ A_1 \wedge \dots \wedge A_m;$$

- for every action constant a,

$$\mathbf{exogenous}\ a$$

is added.

Theorem 8 *For any action description D in the language $\mathcal{BC}$ such that in each of its static laws (1) $m = 0$,*

(a) the states of the translation of D into the language $\mathcal{C}+$ are identical to the states of D;

(b) the transitions of the translation of D into the language $\mathcal{C}+$ can be characterized as the triples

$$\langle\, s_0,\ \{a{=}\mathbf{t} \mid a \in \alpha\} \cup \{a{=}\mathbf{f} \mid a \in \sigma^A \setminus \alpha\},\ s_1\,\rangle$$

for all transitions $\langle s_0, \alpha, s_1\rangle$ of D.

This translation is applicable, for instance, to the leaking container example. The description of the blocks world from Section 6 cannot be translated into $\mathcal{C}+$ in this way, because the static laws in the recursive definition of *InTower*(B) violate the condition $m = 0$.

12 Future Work

The version of $\mathcal{BC}$ described in this preliminary report is propositional; expressions with variables, as in the examples from Sections 6 and 7, need to be grounded before they become syntactically correct in the sense of $\mathcal{BC}$. We plan to define the syntax and semantics of $\mathcal{BC}$ with variables, in the sprit of [Lifschitz and Ren, 2007], using the generalization of stable models proposed in [Ferraris *et al.*, 2011].

The version of the Causal Calculator described in [Casolary and Lee, 2011] will be extended to cover the expressive capabilities of $\mathcal{BC}$.

Acknowledgements

Joohyung Lee 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. Many thanks to Michael Gelfond and to the anonymous referees for valuable advice.

References

[Balduccini and Gelfond, 2003] Marcello Balduccini and Michael Gelfond. Diagnostic reasoning with A-Prolog. *Theory and Practice of Logic Programming*, 3(4-5):425–461, 2003.

[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)*, 2012.

[Casolary and Lee, 2011] Michael Casolary and Joohyung Lee. Representing the language of the Causal Calculator in Answer Set Programming. In *Technical Communications of the 27th International Conference on Logic Programming (ICLP)*, pages 51–61, 2011.

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

[Ferraris and Lifschitz, 2005] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5:45–74, 2005.

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

[Ferraris *et al.*, 2012] Paolo Ferraris, Joohyung Lee, Yuliya Lierler, Vladimir Lifschitz, and Fangkai Yang. Representing first-order causal theories by logic programs. *Theory and Practice of Logic Programming*, 12(3):383–412, 2012.

[Fikes and Nilsson, 1971] Richard Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.

[Gelfond and Inclezan, 2009] Michael Gelfond and Daniela Inclezan. Yet another modular action language. In *Proceedings of the Second International Workshop on Software Engineering for Answer Set Programming*, pages 64–78, 2009.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

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

[Gelfond and Lifschitz, 2012] Michael Gelfond and Vladimir Lifschitz. The common core of action languages $\mathcal{B}$ and $\mathcal{C}$. In *Working Notes of the International Workshop on Nonmonotonic Reasoning (NMR)*. 2012.

[Giunchiglia and Lifschitz, 1998] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 623–630. AAAI Press, 1998.

[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.

[Lifschitz and Ren, 2007] Vladimir Lifschitz and Wanwan Ren. The semantics of variables in action descriptions. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 1025–1030, 2007.

[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 *et al.*, 2001] Vladimir Lifschitz, David Pearce, and Agustin Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.

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

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

[McCain and Turner, 1997] Norman McCain and Hudson Turner. Causal theories of action and change. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 460–465, 1997.

[McCain, 1997] Norman McCain. *Causality in Commonsense Reasoning about Actions*. PhD thesis, University of Texas at Austin, 1997.

[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.

[Nogueira *et al.*, 2001] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An A-Prolog decision support system for the Space Shuttle. In *Proceedings of International Symposium on Practical Aspects of Declarative Languages (PADL)*, pages 169–183, 2001.

[Pednault, 1989] Edwin Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In Ronald Brachman, Hector Levesque, and Raymond Reiter, editors, *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 324–332, 1989.

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