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

Joseph Babb and Joohyung Lee

School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, USA
{Joseph.Babb, joolee}@asu.edu

Abstract. Recently, action language $\mathcal{BC}$, which combines the attractive features of action languages $\mathcal{B}$ and $\mathcal{C}$+, was proposed. While $\mathcal{BC}$ allows Prolog-style recursive definitions that are not available in $\mathcal{C}$+, it is less expressive than $\mathcal{C}$+ in other ways, such as inability to express non-Boolean and non-exogenous actions. We propose a new action language called $\mathcal{BC}$+, which encompasses all the features of $\mathcal{BC}$ and the definite fragment of $\mathcal{C}$+. The syntax of $\mathcal{BC}$+ is identical to the syntax of $\mathcal{C}$+ allowing arbitrary propositional formulas in the causal laws, but its semantics is defined in terms of propositional formulas under the stable model semantics instead of nonmonotonic causal theories. This approach allows many useful ASP constructs, such as choice rules and aggregates, to be directly used in language $\mathcal{BC}$+, and exploits computational methods available in ASP solvers.

1 Introduction

Action languages are formal models of parts of natural language that are used for describing properties of actions. The semantics of such languages describe transition systems—directed graphs whose vertices represent states and edges represent actions that change the states. These languages are often viewed as high level notations of nonmonotonic logics. For instance, languages $\mathcal{A}$ [1] and $\mathcal{B}$ [2, Section 5] can be translated into logic programs under the stable model semantics; languages $\mathcal{C}$ [3] and $\mathcal{C}$+ [4] are defined in terms of nonmonotonic causal theories.

Recently, action language $\mathcal{BC}$ was proposed, which combines the attractive features of $\mathcal{B}$ and $\mathcal{C}$+. Like $\mathcal{C}$+, the inertia assumption is not built into $\mathcal{BC}$, so fluents whose behavior is described by defaults other than inertia can be conveniently represented in $\mathcal{BC}$.[1] Unlike $\mathcal{C}$+, the language allows the user to represent Prolog-style recursive definitions available in $\mathcal{B}$, which are useful in describing complex relations among fluents.[2]

However, $\mathcal{BC}$ is less expressive than $\mathcal{C}$+ in several other ways. It assumes that every action is Boolean and "exogenous": the action description does not determine whether that action is executed. The assumption is too strong to describe complex relations among actions, defeasible causal laws [4, Section 4.3], action attributes [4, Section 5.6], and additive fluents [6] that $\mathcal{C}$+ is able to express conveniently. Also, $\mathcal{C}$+ allows complex formulas in causal laws, but $\mathcal{BC}$ allows only conjunctions of atoms.

[1] For instance, the Leaking Container example from [5], also reproduced in Section 6.3.

[2] For instance, the definition of *InTower* in the Blocks World formalization from [5], also reproduced in Section 5.

In this paper, we propose a new action description language $\mathcal{BC}$+. The language is sufficiently expressive to encompass all the features of $\mathcal{BC}$ and the definite fragment of $\mathcal{C}$+. The syntax of this language is essentially identical to $\mathcal{C}$+, but its semantics is instead defined in terms of the stable model semantics as with $\mathcal{BC}$. On the other hand, propositions of $\mathcal{BC}$+ are more general than propositions of $\mathcal{BC}$ to allow many useful ASP constructs, such as choice rules and aggregates. The expressiveness of $\mathcal{BC}$+ can be shown by embedding both $\mathcal{BC}$ and the definite fragment of $\mathcal{C}$+ in $\mathcal{BC}$+ in straightforward ways.

The paper is organized as follows. Sections 2 and 3 introduce the syntax and the semantics of $\mathcal{BC}$+. Section 4 introduces useful abbreviations, which follow the ones available in $\mathcal{C}$+. Section 5 presents an example that illustrates some advantages of $\mathcal{BC}$+. Sections 6 and 7 show how $\mathcal{BC}$ and $\mathcal{C}$+ can be embedded in $\mathcal{BC}$+, and explain the advantages of $\mathcal{BC}$+ over the two other languages. Section 8 shows how propositional formulas under the stable model semantics can be viewed as a special case of $\mathcal{BC}$+.

2 Syntax of $\mathcal{BC}$+

The syntax of language $\mathcal{BC}$+ is essentially identical to the syntax of $\mathcal{C}$+.[3] The following description repeats the syntax description in Section 4.2 from [4]. 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 constant c and is denoted by $Dom(c)$.

We consider (propositional) formulas whose atoms have the form $c = v$, where c is a constant, and v is an element of its domain. If the domain of c is $\{\mathbf{f}, \mathbf{t}\}$ then we say that c is *Boolean*; we abbreviate $c = \mathbf{t}$ as c and $c = \mathbf{f}$ as $\sim c$.

A *fluent formula* is a formula such that all constants occurring in it are fluent constants. An *action formula* is a formula that contains at least one action constant and no fluent constants.

A *static law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \tag{1}$$

where F and G are fluent formulas. An *action dynamic law* is an expression of the form (1) in which F is an action formula and G is a formula. A *fluent dynamic law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{2}$$

where F and G are fluent formulas and H is a formula, provided that F does not contain statically determined constants. Static laws can be used to talk about causal dependencies between fluents in the same state; action dynamic laws can be used to express causal dependencies between concurrently executed actions; fluent dynamic laws can be used for describing direct effects of actions.

[3] Strictly speaking, $\mathcal{C}$+ considers "multi-valued" formulas, an extension of propositional formulas, but this difference is not essential in view of Theorem 1 from [9].

A *causal law* is a static law, an action dynamic law, or a fluent dynamic law. An *action description* is a finite set of causal laws.

The formula F in a causal law (1) or (2) is called the *head*. We say that action description D is *definite* if the head of every causal law is either $\bot$, an atom $c{=}v$, or its choice formula $\{c{=}v\}$, that stands for the propositional formula $c{=}v \vee \neg(c{=}v)$.

3 Semantics of $\mathcal{BC}$+

For any action description D with a set σ^{fl} of fluent constants and a set σ^{act} of action constants, we will define a sequence of propositional formulas $PF_0(D), PF_1(D), \ldots$ so that the stable models of $PF_m(D)$ represent paths of length m in the transition system corresponding to D. The signature $\sigma_{D,m}$ of $PF_m(D)$ consists of the pairs $i{:}c$ such that

- $i \in \{0,\ldots,m\}$ and c is a fluent constant of D, and
- $i \in \{0,\ldots,m{-}1\}$ and c is an action constant of D.

The domain of $i:c$ is the same as the domain of c. By $i:F$ we denote the result of inserting $i:$ in front of every occurrence of every constant in formula F. The translation $PF_m(D)$ is the conjunction of

- $$i{:}F \;\leftarrow\; i{:}G \tag{3}$$
  for every static law (1) in D and every $i \in \{0,\ldots,m\}$, and (3) for every action dynamic law (1) in D and every $i \in \{0,\ldots,m{-}1\}$;[4]
- $$i{+}1{:}F \;\leftarrow\; (i{+}1{:}G) \wedge (i{:}H) \tag{4}$$
  for every fluent dynamic law (2) in D and every $i \in \{0,\ldots,m{-}1\}$;
- $$\{0{:}c{=}v\} \tag{5}$$
  for every regular fluent constant c and every $v \in \mathit{Dom}(c)$;
- $$\bigwedge_{v\neq w \;|\; v,w\in Dom(c)} \neg(i{:}c = v \wedge i{:}c = w), \tag{6}$$
  $$\neg\neg \bigvee_{v\in Dom(c)} i{:}c = v\ , \tag{7}$$
  for every constant $i:c$ in $\sigma_{D,m}$, which represent the uniqueness and existence of values for the constant $i:c$. The conjunction of formulas (6) and (7) can also be abbreviated using the count aggregate as
  $$\leftarrow \neg\, 1\{i{:}c{=}v_1,\ldots,i{:}c{=}v_m\}1 \tag{8}$$
  where $\{v_1,\ldots,v_m\}$ is $\mathit{Dom}(c)$.[5]

[4] We identify $F \leftarrow G$ with $G \rightarrow F$.

[5] Aggregates can be viewed as shorthands for propositional formulas [7; 8].

As shown in [9], in the presence of (6) and (7), the choice rule $\{i : c = v\}$ can be understood as assigning default value v to constant $i : c$ (read "by default, $i : c$ maps to v"). In the absence of additional information about $i : c$, choosing $i : c = v$ is the only option due to the existence of value constraint (7). But if there is a conflicting information about $i : c$, then not choosing $i : c = v$ is the only option, in view of the uniqueness of value constraint (6). We will see that this reading of choice rules in the presence of the uniqueness and existence of value constraints is quite convenient in representing dynamic systems.

Note how the translation $PF_m(D)$ treats regular and statically determined fluent constants in different ways: formulas (5) are included only when c is regular.

In the example below, the following abbreviations are used. If c is a Boolean action constant, we express that F is an effect of executing c by the fluent dynamic law

$$\textbf{caused } F \textbf{ if } \top \textbf{ after } c,$$

which can be abbreviated as

$$c \textbf{ causes } F.$$

If c is an action constant, the expression

$$\textbf{exogenous } c$$

stands for the action dynamic laws

$$\textbf{caused } \{c{=}v\}$$

for all $v \in \mathit{Dom}(c)$. If c is a regular fluent constant, the expression

$$\textbf{inertial } c \tag{9}$$

stands for the fluent dynamic laws

$$\textbf{caused } \{c{=}v\} \textbf{ after } c{=}v$$

for all $v \in \mathit{Dom}(c)$.

The transition system shown in Figure 1 can be described by the following action description *SD*, where p is a Boolean regular fluent constant and a is a Boolean action constant:

$$\begin{array}{l} a \textbf{ causes } p, \\ \textbf{exogenous } a, \\ \textbf{inertial } p. \end{array} \tag{10}$$

The translation $PF_m(SD)$ turns this description into the following propositional formulas. The first line of (10) is turned into the formulas

$$i{+}1{:}p \leftarrow i{:}a$$

($0 \leq i < m$), the second line into

$$\begin{array}{l} \{i{:}a\}, \\ \{i{:}{\sim}a\} \end{array} \tag{11}$$

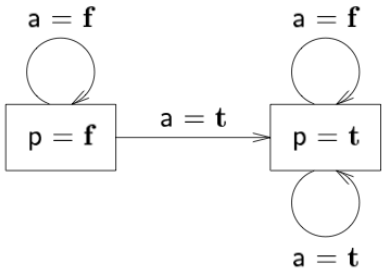

Fig. 1. The transition system described by *SD*.

$(0 \leq i < m)$, and the third into

$$\begin{array}{l} \{i{+}1{:}p\} \leftarrow i{:}p, \\ \{i{+}1{:}{\sim}p\} \leftarrow i{:}{\sim}p \end{array} \tag{12}$$

$(0 \leq i < m)$. In addition,

$$\begin{array}{l} \{0{:}p\}, \\ \{0{:}{\sim}p\} \end{array}$$

come from (5), and

$$\begin{array}{ll} \leftarrow \neg\, 1\{i{:}p,\ i{:}{\sim}p\}1 & (0 \leq i \leq m), \\ \leftarrow \neg\, 1\{i{:}a,\ i{:}{\sim}a\}1 & (0 \leq i < m) \end{array}$$

come from (8).

A *state* is an interpretation s of σ^{fl} such that $0{:}s$ is a stable model of $PF_0(D)$. States are the vertices of the transition system represented by D. In view of the existence and uniqueness of value constraints for every state s and every fluent constant c, there exists exactly one v such that $c{=}v$ belongs to s; this v is considered the value of c in state s.

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 $0{:}s \cup 0{:}e \cup 1{:}s'$ is a model of $PF_1(D)$. Transitions correspond to the edges of the transition system: for every transition $\langle s, e, s' \rangle$, it contains an edge from s to s' labeled e.

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

Theorem 1 *For every transition $\langle s, e, s' \rangle$, s and s' are states.*

The stable models of $PF_m(D)$ represent the paths of length m in the transition system corresponding to D. For $m = 0$ and $m = 1$, this is clear from the definition of a transition system; for $m > 1$ this needs to be verified as the following theorem shows.

For every set X of elements of the signature $\sigma_{D,m}$, let X^i $(i < m)$ be the triple consisting of

– the set consisting of atoms A such that $i : A$ belongs to X, and A contains fluent constants,

– the set consisting of atoms A such that $i : A$ belongs to X, and A contains action constants, and
– the set consisting of atoms A such that $(i + 1) : A$ belongs to X, and A contains fluent constants.

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

We understand choice rules and count aggregates as shorthand for propositional formulas as described in [10; 11], allowing them to be directly used in causal laws. We will show an example in Section 5.

4 Useful Abbreviations

Language $\mathcal{BC}$+ allows the same syntax of the abbreviations in $\mathcal{C}$+ (c.f. [4, Appendix B]), but understands them differently. For instance,

– **default** $c=v$ **if** F stands for **caused** $\{c=v\}$ **if** F. This is consistent with the reading of choice formulas in the presence of uniqueness and existence of value constraints we introduced in Section 3. Similarly,

$$\textbf{default } c=v \textbf{ if } F \textbf{ after } G$$

stands for

$$\textbf{caused } \{c=v\} \textbf{ if } F \textbf{ after } G.$$

This allows us to view the expression (9) as shorthand for

$$\textbf{default } c=v \textbf{ after } c=v.$$

– **constraint** F where F is a fluent formula stands for the static law

$$\textbf{caused } \bot \textbf{ if } \neg F;$$

– **always** F stands for the fluent dynamic law

$$\textbf{caused } \bot \textbf{ if } \top \textbf{ after } \neg F\ ;$$

– **nonexecutable** F **if** G stands for the fluent dynamic law

$$\textbf{caused } \bot \textbf{ if } \top \textbf{ after } F \wedge G.$$

5 Example: Blocks World

An attractive feature of $\mathcal{BC}$+ is that it can use several useful ASP constructs directly in causal laws by resorting to the underlying stable model semantics. For instance, it allows aggregates in causal laws since aggregates can be viewed as shorthand for some propositional formulas [7; 8]. We illustrate this advantage by formalizing in $\mathcal{BC}$+ an elaboration of the Blocks World from [5].

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 the 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*$\}$.

The definition of *InTower*(B):

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

Blocks don't float in the air:

constraint *InTower*(B).

No two blocks are on the same block:

constraint $\{b :$ *Loc*$(b) = B\}1$.

Only k towers are allowed to be on the table ($\mathrm{k} > 0$):

constraint $\{b :$ *Loc*$(b) =$ *Table*$\}\mathrm{k}$.

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*$(B_1) = B$.

Concurrent actions are limited by the number g of grippers:

always $\{bl :$ *Move*$(b, l)\}\mathrm{g}$.

The commonsense law of inertia:

inertial *Loc*(B).

6 Relation to Language $\mathcal{BC}$

6.1 Review: $\mathcal{BC}$

The signature σ for a $\mathcal{BC}$ description D is defined the same as in $\mathcal{BC}$+ except that every action constant is assumed to be Boolean. The main syntactic difference between $\mathcal{BC}$ causal laws and $\mathcal{BC}$+ causal laws is that the former allows only the conjunction of atoms in the body, and distinguishes between **if** and **if cons** clauses. A $\mathcal{BC}$ *static law* is an expression of the form

$$A_0 \textbf{ if } A_1, \ldots, A_m \textbf{ if cons } A_{m+1}, \ldots, A_n \tag{13}$$

$(n \geq m \geq 0)$, where each A_i is an atom containing a fluent constant. 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.

A $\mathcal{BC}$ *dynamic law* is an expression of the form

$$A_0 \textbf{ after } A_1, \ldots, A_m \textbf{ if cons } A_{m+1}, \ldots, A_n \tag{14}$$

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

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

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.

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

The semantics of $\mathcal{BC}$ is defined similar to the semantics of $\mathcal{BC}$+ by using reduction $PF_m^{\mathcal{BC}}(\cdot)$ to a sequence of logic programs under the stable model semantics. The signature $\sigma_{D,m}$ of $PF_m^{\mathcal{BC}}(\cdot)$ is defined the same as that of $PF_m(\cdot)$.

For any $\mathcal{BC}$ action description D, by $PF_m^{\mathcal{BC}}(D)$ we denote the conjunction of

- $$i{:}A_0 \;\leftarrow\; i : (A_1 \wedge \cdots \wedge A_m \wedge \neg\neg A_{m+1} \wedge \cdots \wedge \neg\neg A_n) \tag{15}$$
  for every $\mathcal{BC}$ static law (13) in D and every $i \in \{0, \ldots, m\}$;
- $$(i+1){:}A_0 \;\leftarrow\; i{:}(A_1 \wedge \cdots \wedge A_m) \wedge (i{+}1){:}(\neg\neg A_{m+1} \wedge \cdots \wedge \neg\neg A_n) \tag{16}$$
  for every $\mathcal{BC}$ dynamic law (14) in D and every $i \in \{0, \ldots, m{-}1\}$;
- the formula $i{:}(a = \mathbf{t} \vee a = \mathbf{f})$ for every action constant a and every $i \in \{0, \ldots, m{-}1\}$;
- the formula (5) for every regular fluent constant c and every element $v \in \mathit{Dom}(c)$;
- the formulas (6) and (7) for every constant $i{:}c$ in $\sigma_{D,m}$.

Note how the translations (15) and (16) treat **if** and **ifcons** clauses differently by either prepending double negations in front of atoms or not. In $\mathcal{BC}$+, since the formulas are understood under the stable model semantics, which distinguishes between A and $\neg\neg A$, there is no need to distinguish between these two clauses.

6.2 Embedding $\mathcal{BC}$ in $\mathcal{BC}$+

Despite the syntactic differences, language $\mathcal{BC}$ can be easily embedded in $\mathcal{BC}$+ as follows. For any $\mathcal{BC}$ description D, we define the translation *bc2bcp*(D), which turns a $\mathcal{BC}$ description into $\mathcal{BC}$+, as follows:

– replace every causal law (13) with

$$\textbf{caused } A_0 \textbf{ if } A_1 \wedge \cdots \wedge A_m \wedge \neg\neg A_{m+1} \wedge \cdots \wedge \neg\neg A_n; \tag{17}$$

– replace every causal law (14) with

$$\textbf{caused } A_0 \textbf{ if } A_{m+1} \wedge \ldots A_n \textbf{ after } A_1 \wedge \cdots \wedge A_m;$$

– add the causal laws

$$\textbf{caused } \{a = \mathbf{t}\},$$
$$\textbf{caused } \{a = \mathbf{f}\}$$

for every action constant a.

Theorem 3 *For any action description D in language $\mathcal{BC}$, the transition system described by D is identical to the transition system described by bc2bcp(D) in language $\mathcal{BC}$+.*

The proof can be established by showing strong equivalence between the propositional formula $PF_m^{\mathcal{BC}}(D)$ and the propositional formula PF_m(*bc2bcp(D)*).

6.3 Advantages of $\mathcal{BC}$+ over $\mathcal{BC}$

In $\mathcal{BC}$, every action is assumed to be Boolean and exogenous. This is too strong an assumption that prevents us from describing defeasible causal laws [4, Section 4.3], action attributes [4, Section 5.6], and additive fluents [6] that $\mathcal{BC}$+ and $\mathcal{C}$+ are able to express naturally. Also, syntactically, $\mathcal{BC}$ is not expressive enough to describe dependencies among actions. For a simple example, in $\mathcal{BC}$+ we can express that action a_1 is not executable when a_2 is not executed at the same time as

$$\textbf{caused } \bot \textbf{ after } a_1 \wedge \neg a_2$$

which is not syntactically allowed in $\mathcal{BC}$.

On the other hand, the presence of choice formulas in the head of $\mathcal{BC}$+ causal laws and the different treatment of A and $\neg\neg A$ in the bodies may look subtle to those who are not familiar with the stable model semantics for propositional formulas. Fortunately, in many cases one can hide these constructs by using intuitive **default** abbreviations introduced in Section 4 as the following example illustrates.

Consider the leaking container example from [5] in which a container loses k units of liquid by default. This example was used to illustrate the advantages of $\mathcal{BC}$ over $\mathcal{B}$ that is able to express defaults other than inertia. In this domain, the default decrease of *Amount* over time can be represented in $\mathcal{BC}$+ using the **default** abbreviation

$$\textbf{default } \mathit{Amount} = x \textbf{ after } \mathit{Amount} = x + k, \tag{18}$$

which stands for fluent dynamic law

$$\textbf{caused } \{\mathit{Amount} = x\} \textbf{ after } \mathit{Amount} = x + k,$$

which can be further turned into propositional formulas

$$\{i{+}1 : \mathit{Amount}{=}x\} \leftarrow i : \mathit{Amount}{=}x{+}k \tag{19}$$

($i < m$). On the other hand, the abbreviation (18) in $\mathcal{BC}$ stands for the causal law

$$\textbf{caused } \mathit{Amount}{=}x \textbf{ after } \mathit{Amount}{=}x{+}k \textbf{ if cons } \mathit{Amount}{=}x,$$

which is further turned into

$$i{+}1 : \mathit{Amount}{=}x \ \leftarrow i : \mathit{Amount}{=}x{+}k \wedge \neg\neg(i : \mathit{Amount}{=}x)$$

($i < m$), which is strongly equivalent to (19).

7 Relation to C+

7.1 Review: C+ in ASP

As mentioned earlier, the syntax of C+ is the same as the syntax of BC+. That is, a C+ *static law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \tag{20}$$

where F and G are fluent formulas. A C+ *action dynamic law* is an expression of the form (20) in which F is an action formula and G is a formula. A C+ *fluent dynamic law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{21}$$

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

We say that C+ action description D is *definite* if the head of every causal law is either $\bot$ or an atom $c{=}v$.

The original semantics of C+ is defined in terms of reduction to nonmonotonic causal theories in [4]. In [12], the semantics of the definite C+ description is equivalently reformulated in terms of reduction to propositional formulas under the the stable model semantics as follows. [6]

For any definite C+ action description D and any nonnegative integer m, the propositional formula $PF_m^{\mathcal{C}+}(D)$ is defined as follows. The signature of $PF_m^{\mathcal{C}+}(D)$ is defined the same as $PF_m(D)$. The translation $PF_m^{\mathcal{C}+}(D)$ is the conjunction of

–
$$i : F \leftarrow \ \neg\neg\,(i : G) \tag{22}$$
for every static law (20) in D and every $i \in \{0,\dots,m\}$, and for every action dynamic law (20) in D and every $i \in \{0,\dots,m{-}1\}$;

[6] The translation does not work for nondefinite C+ descriptions, due to the different treatments of the heads under nonmonotonic causal theories and under the stable model semantics.

–

$$i{+}1 : F \leftarrow \ \neg\neg(i{+}1 : G) \wedge (i : H) \tag{23}$$

for every fluent dynamic law (21) in D and every $i \in \{0, \ldots, m{-}1\}$;

– the formula (5) for every regular fluent constant c and every $v \in \mathit{Dom}(c)$.
– the formulas (6) and (7) for every constant $i : c$ in $\sigma_{D,m}$.

Compare (22) and (23) for $\mathcal{C}$+ with (3) and (4) for $\mathcal{BC}$+. They are very similar except that in (22) and (23), double negations are always introduced when **if** G part is translated. This makes definite $\mathcal{C}$+ descriptions not able to represent transitive closures in cyclic graphs correctly.

7.2 Embedding Definite $\mathcal{C}$+ in $\mathcal{BC}$+

For any definite $\mathcal{C}$+ description D, we define the translation *cp2bcp*(D), which turns $\mathcal{C}$+ description into $\mathcal{BC}$+, as follows:

– replace every $\mathcal{C}$+ causal law (20) with

$$\textbf{caused } F \textbf{ if } \neg\neg G;$$

– replace every $\mathcal{C}$+ causal law (21) with

$$\textbf{caused } F \textbf{ if } \neg\neg G \textbf{ after } H.$$

The following theorem asserts the correctness of this translation.

Theorem 4 *For any definite action description D in language $\mathcal{C}$+, the transition system described by D is identical to the transition system described by cp2bcp(D) in language $\mathcal{BC}$+.*

Again, the proof can be established by showing the strong equivalence between the propositional formula $PF_m^{\mathcal{C}+}(D)$ and the propositional formula $PF_m(\mathit{cp2bcp(D)})$.

7.3 Advantages of $\mathcal{BC}$+ over $\mathcal{C}$+

Recall that the syntax of $\mathcal{BC}$+ is identical to the syntax of $\mathcal{C}$+,[7] but its semantics is given via the stable model semantics. An advantage of this approach is that it allows the advances in ASP directly used in the context of action languages. We already obseved that being able to use aggregates in $\mathcal{BC}$+ provides a succinct representation of the Blocks World domain.

Another advantage of $\mathcal{BC}$+ is that it avoids some unintuitive behavior of $\mathcal{C}$+ in representing causal dependencies among fluents. Consider two switches which can be flipped but cannot be both up or down at the same time. If one of them is down and the other is up, the direct effect of flipping only one switch is changing the status of that switch,

[7] Recall that we understand choice formulas and aggregates as abbreviation of propositional formulas.

Notation: s, s' ranges over $\{Switch_1, Switch_2\}$; x, y ranges over $\{Up, Down\}$.

Regular fluent constants:	Domains:
$Status(s)$	$\{Up, Down\}$
Action constants:	Domains:
$Flip(s)$	Boolean

Causal laws:

$Flip(s)$ **causes** $Status(s) = x$ **if** $Status(s) = y$	$(x \neq y)$
caused $Status(s) = x$ **if** $Status(s') = y$	$(s \neq s', x \neq y)$
inertial $Status(s)$	
exogenous $Flip(s)$	

Fig. 2. Two Switch

and the indirect effect is changing the status of the other switch. This domain can be represented in $\mathcal{BC}$+ as shown in Figure 2.

The description in $\mathcal{BC}$+ has the following four transitions possible from the initial state where $Switch_1$ is *Down* and $Switch_2$ is *Up*:
$\langle \{St(Sw_1) = Dn, St(Sw_2) = Up\}, \{\sim Flip(Sw_1), \sim Flip(Sw_2)\}, \{St(Sw_1) = Dn, St(Sw_2) = Up\}\rangle$,
$\langle \{St(Sw_1) = Dn, St(Sw_2) = Up\}, \{Flip(Sw_1), \sim Flip(Sw_2)\}, \{St(Sw_1) = Up, St(Sw_2) = Dn\}\rangle$,
$\langle \{St(Sw_1) = Dn, St(Sw_2) = Up\}, \{\sim Flip(Sw_1), Flip(Sw_2)\}, \{St(Sw_1) = Up, St(Sw_2) = Dn\}\rangle$,
$\langle \{St(Sw_1) = Dn, St(Sw_2) = Up\}, \{Flip(Sw_1), Flip(Sw_2)\}, \{St(Sw_1) = Up, St(Sw_2) = Dn\}\rangle$.

The second and the third transitions exhibit the indirect effect of the action *Flip*. If this description is understood in $\mathcal{C}$+, five transitions are possible from the same initial state: in addition to the four transitions above,
$\langle \{St(Sw_1) = Dn, St(Sw_2) = Up\}, \{\sim Flip(Sw_1), \sim Flip(Sw_2)\}, \{St(Sw_1) = Up, St(Sw_2) = Dn\}\rangle$
is also a transition according to the semantics of $\mathcal{C}$+, which is obviously unintuitive.

8 Embedding ASP Programs in $\mathcal{BC}$+

We defined the semantics of $\mathcal{BC}$+ by reducing the language to propositional formulas under the stable model semantics. The reduction in the opposite direction is also possible.

Any propositional formula F under the stable model semantics can be turned into an action description in $\mathcal{BC}$+ by treating every atom of F as a statically determined fluent constant with Boolean values, and rewriting every formula F as the static law

$$\textbf{caused } F$$

and add

$$\textbf{caused } \{c = \mathbf{f}\}$$

for every constant c.

Proposition 1 *The stable models of a propositional formula F are exactly the states of the transition system described by $\mathcal{BC}$+ description obtained by the above translation.*

The proof uses Theorem 7 from [13].

9 Implementation

Language $\mathcal{BC}$+ is implemented in CPLUS2ASP [15], which was originally designed to compute the definite fragment of $\mathcal{C}$+ using ASP solvers. Version 2 of the system supports extensible multi-modal translations for other action languages. As the translation $PF_m^{\mathcal{C}+}(D)$ for $\mathcal{C}$+ is very similar to the translation $PF_m(D)$ for $\mathcal{BC}$+, this extension is straightforward.

For example, the $\mathcal{BC}$+ description in Figure 2 is represented in the input language of CPLUS2ASP as follows:

```
% File 'switch'

:- sorts
   switch; status.

:- objects
   s1, s2             :: switch;
   on, off            :: status.

:- constants
   sw_status(switch)  :: inertialFluent(status);
   flip(switch)       :: exogenousAction.

:- variables
   S, S1              :: switch;
   X, Y               :: status.

flip(S) causes sw_status(S)=X if sw_status(S)=Y & X\=Y.

caused sw_status(S)=X if sw_status(S1)=Y & S\=S1 & X\=Y.

:- query
   label :: 0;
   maxstep :: 1;
   0: sw_status(s1)=off & sw_status(s2)=on.
```

Due to lack of space we refer the reader to the system homepage

```
http://reasoning.eas.asu.edu/cplus2asp/
```

for the details of the input language.

The following is the command line to find all four transitions described in Section 7.3.

```
$ cplus2asp --language=bc+ switch query=0 0
```

The option `--language=bc+` instructs CPLUS2ASP to operate under the $\mathcal{BC}$+ semantics. The 0 at the end instructs the system to find all stable models.

The following is the output:

```
Solution: 1
        0:  sw_status(s1)=off sw_status(s2)=on

        1:  sw_status(s1)=off sw_status(s2)=on

Solution: 2
        0:  sw_status(s1)=off sw_status(s2)=on

        ACTIONS:  flip(s1) flip(s2)

        1:  sw_status(s1)=on sw_status(s2)=off

Solution: 3
        0:  sw_status(s1)=off sw_status(s2)=on

        ACTIONS:  flip(s2)

        1:  sw_status(s1)=on sw_status(s2)=off

Solution: 4
        0:  sw_status(s1)=off sw_status(s2)=on

        ACTIONS:  flip(s1)

        1:  sw_status(s1)=on sw_status(s2)=off

SATISFIABLE
```

If the same program is run under the $\mathcal{C}$+ mode,

```
cplus2asp --language=c+ switch query=0 0
```

one more solution is returned:

```
        0:  sw_status(s1)=off sw_status(s2)=on

        1:  sw_status(s1)=on sw_status(s2)=off
```

10 Conclusion

While many action languages are shown to be turned into logic programs under the stable model semantics, language $\mathcal{BC}$+ exploits the generality of the propositional formulas under the stable model semantics. This approach can be further generalized by using a more general version of the stable model semantics as the target language, such as the first-order stable model semantics [14], which would yield a proper generalization of $\mathcal{BC}$+ to the first-order level.

Acknowledgements We are grateful to the anonymous referees for their useful comments. This work was partially supported by the National Science Foundation under Grant IIS-1319794 and by the South Korea IT R&D program MKE/KIAT 2010-TD-300404-001.

References

1. Gelfond, M., Lifschitz, V.: Representing action and change by logic programs. Journal of Logic Programming **17** (1993) 301–322
2. Gelfond, M., Lifschitz, V.: Action languages. Electronic Transactions on Artificial Intelligence **3** (1998) 195–210
3. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: Preliminary report. In: Proceedings of National Conference on Artificial Intelligence (AAAI), AAAI Press (1998) 623–630
4. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artificial Intelligence **153(1–2)** (2004) 49–104
5. Lee, J., Lifschitz, V., Yang, F.: Action language $\mathcal{BC}$: Preliminary report. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI). (2013)
6. Lee, J., Lifschitz, V.: Describing additive fluents in action language $\mathcal{C}+$. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI). (2003) 1079–1084
7. Ferraris, P.: Answer sets for propositional theories. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). (2005) 119–131
8. Lee, J., Meng, Y.: On reductive semantics of aggregates in answer set programming. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). (2009) 182–195
9. Bartholomew, M., Lee, J.: Stable models of multi-valued formulas: Partial vs. total functions. In: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR). (2014) To appear.
10. 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
11. Lee, J., Meng, Y.: On reductive semantics of aggregates in answer set programming. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). (2009) 182–195
12. Lee, J.: Reformulating action language $\mathcal{C}+$ in answer set programming. In Erdem, E., Lee, J., Lierler, Y., Pearce, D., eds.: Correct Reasoning. Volume 7265 of Lecture Notes in Computer Science., Springer (2012) 405–421
13. Bartholomew, M., Lee, J.: Stable models of formulas with intensional functions. In: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR). (2012) 2–12
14. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artificial Intelligence **175** (2011) 236–263
15. Babb, J., Lee, J.: Cplus2asp: Computing action language $\mathcal{C}+$ in answer set programming. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). (2013) 122–134
16. Ferraris, P., Lee, J., Lifschitz, V., Palla, R.: Symmetric splitting in the general theory of stable models. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), AAAI Press (2009) 797–803