Action Language $\mathcal{BC}$+

Joseph Babb and Joohyung Lee*

School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, USA

Abstract. Action languages are formal models of parts of natural language that are designed to describe effects of actions. Many of these languages can be viewed as high level notations of answer set programs structured to represent transition systems. However, the form of answer set programs considered in the earlier work is quite limited in comparison with the modern Answer Set Programming (ASP) language, which allows several useful constructs for knowledge representation, such as choice rules, aggregates, and abstract constraint atoms. We propose a new action language called $\mathcal{BC}$+, which closes the gap between action languages and the modern ASP language. The main idea is to define the semantics of $\mathcal{BC}$+ in terms of general stable model semantics for propositional formulas, under which many modern ASP language constructs can be identified with shorthands for propositional formulas. Language $\mathcal{BC}$+ turns out to be sufficiently expressive to encompass the best features of other action languages, such as languages $\mathcal{B}$, $\mathcal{C}$, $\mathcal{C}$+, and $\mathcal{BC}$. Computational methods available in ASP solvers are readily applicable to compute $\mathcal{BC}$+, which led to an implementation of the language by extending system CPLUS2ASP.

1 Introduction

Action languages are formal models of parts of natural language that are used for describing properties of actions. The semantics of action languages describe transition systems—directed graphs whose vertices represent states and whose edges represent actions that affect the states. Many action languages, such as languages $\mathcal{A}$ [Gelfond and Lifschitz, 1993] and $\mathcal{B}$ [Gelfond and Lifschitz, 1998, Section 5], can be viewed as high level notations of answer set programs structured to represent transition systems. Languages $\mathcal{C}$ [Giunchiglia and Lifschitz, 1998] and $\mathcal{C}$+ [Giunchiglia *et al.*, 2004] are originally defined in terms of nonmonotonic causal theories, but their "definite" fragments can be equivalently turned into answer set programs as well [Ferraris *et al.*, 2012], which led to the implementation CPLUS2ASP, which uses ASP solvers for computation [Babb and Lee, 2013].

The main advantage of using action languages over answer set programs is their structured abstract representations for describing transition systems, which allows their users to focus on high level descriptions and avoids the "cryptic" syntax and the recurring pattern of ASP rules for representing transition systems. However, existing work on action languages has two limitations. First, they do not allow several useful ASP language constructs, such as choice rules, aggregates, abstract constraint atoms, and external atoms, that have recently been introduced into ASP, and contributed to widespread

* Corresponding author: joolee@asu.edu

use of ASP in many practical applications. The inability to express these modern constructs in action languages is what often forces the users to write directly in the language of ASP rather than in action languages.

Another issue arises even such constructs are not used: there are certain limitations that each action language has in comparison with one another. For instance, in language $\mathcal{B}$, the frame problem is solved by enforcing in the semantics that every fluent be governed by the commonsense law of inertia, which makes it difficult to represent fluents whose behavior is described by defaults other than inertia, such as the amount of water in a leaking container. Languages $\mathcal{C}$ and $\mathcal{C}$+ do not have this limitation, but instead they do not handle Prolog-style recursive definitions available in $\mathcal{B}$. The recently proposed language $\mathcal{BC}$ [Lee *et al.*, 2013] combines the attractive features of $\mathcal{B}$ and $\mathcal{C}$+, but it is not a proper generalization. In comparison with $\mathcal{C}$+, it does not allow us to describe complex dependencies among actions, thus it is unable to describe several concepts that $\mathcal{C}$+ is able to express, such as "defeasible" causal laws [Giunchiglia *et al.*, 2004, Section 4.3] (causal laws that are retracted by adding additional causal laws) and "attributes" of an action [Giunchiglia *et al.*, 2004, Section 5.6], which are useful for elaboration tolerant representation.

We present a simple solution to these problems. The main idea is to define an action language in terms of a general stable model semantics, which has not been considered in the work on action languages. We present a new action language called $\mathcal{BC}$+, which is defined as a high level notation of propositional formulas under the stable model semantics [Ferraris, 2005]. It has been well studied in ASP that several useful constructs, such as aggregates, abstract constraint atoms, and conditional literals, can be identified with abbreviations of propositional formulas (e.g.,[Ferraris, 2005; Pelov *et al.*, 2003; Son and Pontelli, 2007; Harrison *et al.*, 2014]). Thus, $\mathcal{BC}$+ employs such constructs as well. Further, it is more expressive than the other action languages mentioned above, allowing them to be easily embedded. Computational problems involving $\mathcal{BC}$+ descriptions can be reduced to computing answer sets. This fact led to an implementation of $\mathcal{BC}$+ by modifying system CPLUS2ASP [Babb and Lee, 2013], which was originally designed to compute $\mathcal{C}$+ using ASP solvers.

The paper is organized as follows. Section 2 reviews propositional formulas under the stable model semantics for describing multi-valued constants. Sections 3 and 4 present the syntax and the semantics of $\mathcal{BC}$+, and Section 5 presents useful abbreviations of causal laws in that language, followed by Section 6, which formalizes an example using such abbreviations. Section 7 shows how to embed propositional formulas under the stable model semantics into $\mathcal{BC}$+. Sections 8 and 9 relate $\mathcal{BC}$+ to each of $\mathcal{BC}$ and $\mathcal{C}$+. Section 10 describes an implementation of $\mathcal{BC}$+ as an extension of system CPLUS2ASP. The proofs are given in Appendix A.

This is an extended version of the conference paper [Babb and Lee, 2015].

2 Review: Propositional Formulas under the Stable Model Semantics

A *propositional signature* is a set of symbols called *atoms*. A propositional formula is defined recursively using atoms and the following set of primitive propositional connectives: $\bot$ (falsity), $\wedge$, $\vee$, $\rightarrow$. We understand $\neg F$ as an abbreviation of $F \rightarrow \bot$; symbol $\top$ stands for $\bot \rightarrow \bot$, expression $G \leftarrow F$ stands for $F \rightarrow G$, and expression $F \leftrightarrow G$ stands for $(F \rightarrow G) \wedge (G \rightarrow F)$.

An *interpretation* of a propositional signature is a function from the signature into $\{\mathbf{f}, \mathbf{t}\}$. We identify an interpretation with the set of atoms that are true in it.

A *model* of a formula is an interpretation that *satisfies* the formula. According to [Ferraris, 2005], the models are divided into stable models and non-stable models as follows. The *reduct* F^X of a propositional formula F relative to a set X of atoms is the formula obtained from F by replacing every maximal subformula that is not satisfied by X with $\bot$. Set X is called a *stable model* of F if X is a minimal set of atoms satisfying F^X.

Throughout this paper, we assume that the signature σ is constructed from "constants" and their "values." A *constant* c is a symbol that is associated with a finite set $Dom(c)$ of cardinality ≥ 2, called the *domain*. The signature σ is constructed from a finite set of constants, consisting of atoms $c\!=\!v$ [1] for every constant c and every element v in $Dom(c)$. If the domain of c is $\{\mathbf{f}, \mathbf{t}\}$ then we say that c is *Boolean*, and abbreviate $c\!=\!\mathbf{t}$ as c and $c\!=\!\mathbf{f}$ as $\sim\!c$.

Bartholomew and Lee [2014] show that this form of propositional formulas is useful for expressing the concept of default values on multi-valued fluents. By UEC_σ ("Uniqueness and Existence Constraint") we denote the conjunction of

$$\bigwedge_{v \neq w \,:\, v,w \in Dom(c)} \neg(c = v \wedge c = w), \tag{1}$$

and

$$\neg\neg \bigvee_{v \in Dom(c)} c = v\ . \tag{2}$$

for all constants c of σ. It is clear that any propositional interpretation of σ that satisfies UEC_σ can be identified with a function that maps each constant c into an element in its domain.

Example 1. Consider a signature σ to be $\{c\!=\!1,\ c\!=\!2,\ c\!=\!3\}$, where c is a constant and $Dom(c) = \{1, 2, 3\}$. Formula UEC_σ is

$$\neg(c\!=\!1 \wedge c\!=\!2) \wedge \neg(c\!=\!2 \wedge c\!=\!3) \wedge \neg(c\!=\!1 \wedge c\!=\!3) \wedge \neg\neg(c\!=\!1 \vee c\!=\!2 \vee c\!=\!3).$$

Let F_1 be $(c = 1 \vee \neg(c = 1)) \wedge UEC_\sigma$. Due to UEC_σ, each of $\{c = 1\}$, $\{c = 2\}$, and $\{c\!=\!3\}$ is a model of F_1, but $\{c\!=\!1\}$ is the only stable model of F_1. The reduct $F_1^{\{c=1\}}$ is equivalent to $c = 1$, for which $\{c = 1\}$ is the minimal model. On the other hand, for instance, the reduct $F_1^{\{c=2\}}$ is equivalent to $\top$, for which the minimal model is $\emptyset$, not $\{c\!=\!2\}$.

Let F_2 be F_1 conjoined with $c = 2$. Interpretation $\{c = 1\}$ is not a stable model of F_2. Indeed, the reduct $F_2^{\{c=1\}}$ is $\bot$, for which there is no model. However, $\{c = 2\}$ is a stable model of F_2. The reduct $F_2^{\{c=2\}}$ is equivalent to $c = 2$, for which $\{c = 2\}$ is the minimal model. This case illustrates the nonmonotonicity of the semantics.

[1] Note that here "=" is just a part of the symbol for propositional atoms, and is not equality in first-order logic.

Note that the presence of double negations is essential in (2). Without them, F_1 would have three stable models: $\{c=1\}$, $\{c=2\}$, and $\{c=3\}$.

In ASP, formulas of the form $F \vee \neg F$ are called *choice formulas*, which we denote by $\{F\}^{\rm ch}$. For example, F_1 in Example 1 can be written as $\{c=1\}^{\rm ch} \wedge \mathit{UEC}_\sigma$. As shown in Example 1, in the presence of UEC_σ, a formula of the form $\{c=v\}^{\rm ch}$ expresses that c has the value v by default, which can be overridden in the presence of other evidences [Bartholomew and Lee, 2014].

Given that the domain is finite, aggregates in ASP can be understood as shorthand for propositional formulas as shown in [Ferraris, 2005; Pelov *et al.*, 2003; Son and Pontelli, 2007; Lee and Meng, 2009]. For instance, cardinality constraint (i.e., count aggregate) $l \le Z$, where l is a nonnegative integer, and Z is a finite set of atoms, is the disjunction of the formulas $\bigwedge_{L\in Y} L$ over all l-element subset Y of Z. For instance, the cardinality constraint $2 \le \{p, q, r\}$ is shorthand for the propositional formula

$$(p \wedge q) \vee (q \wedge r) \vee (p \wedge r).$$

Expression $Z \le u$, where u is a nonnegative integer, denotes $\neg((u{+}1) \le Z)$. Expression $l \le Z \le u$ stands for $(l \le Z) \wedge (Z \le u)$.

More generally, abstract constraint atoms [Marek and Truszczynski, 2004] can be understood as shorthand for propositional formulas [Lee and Meng, 2012].

3 Syntax of $\mathcal{BC}$+

The syntax of language $\mathcal{BC}$+ is similar to the syntax of $\mathcal{C}$+.[2] In language $\mathcal{BC}$+, a signature σ is a finite set of propositional atoms of the form $c = v$, where constants c are divided into two groups: *fluent constants* and *action constants*. Fluent constants are further divided into *regular* and *statically determined*.

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. [3]

A *static law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \tag{3}$$

where F and G are fluent formulas.

An *action dynamic law* is an expression of the form (3) in which F is an action formula and G is a formula.

A *fluent dynamic law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{4}$$

[2] Strictly speaking, $\mathcal{C}$+ considers "multi-valued" formulas, an extension of propositional formulas, but Theorem 1 from [Bartholomew and Lee, 2014] shows that multi-valued formulas under the stable model semantics can be identified with propositional formulas under the stable model semantics in the presence of the uniqueness and existence of value constraints.

[3] The definition implies that formulas that contain no constants (but may contain $\bot$ and $\top$) are fluent formulas.

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 to describe direct effects of actions.

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 causal laws (3) and (4) is called the *head*.

4 Semantics of $\mathcal{BC}$+

For any action description D of a signature σ, we define a sequence of propositional formulas $PF_0(D), PF_1(D), \dots$ so that the stable models of $PF_m(D)$ can be visualized as paths in a "transition system"—a directed graph whose vertices are states of the world and edges represent transitions between states. The signature σ_m of $PF_m(D)$ consists of atoms of the form $i\!:\!c\!=\!v$ such that

- for each fluent constant c of D, $i \in \{0, \dots, m\}$ and $v \in Dom(c)$, and
- for each action constant c of D, $i \in \{0, \dots, m-1\}$ and $v \in Dom(c)$.

By $i : F$ we denote the result of inserting $i :$ in front of every occurrence of every constant in formula F. This notation is similarly extended when F is a set of formulas. The translation $PF_m(D)$ is the conjunction of

- $$i\!:\!F \;\leftarrow\; i\!:\!G \tag{5}$$
  for every static law (3) in D and every $i \in \{0, \dots, m\}$, and (5) for every action dynamic law (3) in D and every $i \in \{0, \dots, m-1\}$;
- $$i\!+\!1\!:\!F \;\leftarrow\; (i\!+\!1\!:\!G) \wedge (i\!:\!H) \tag{6}$$
  for every fluent dynamic law (4) in D and every $i \in \{0, \dots, m-1\}$;
- $$\{0\!:\!c\!=\!v\}^{\rm ch} \tag{7}$$
  for every regular fluent constant c and every $v \in Dom(c)$;
- UEC_{σ_m}, which can also be abbreviated using the count aggregate as
  $$\bot \leftarrow \neg\,(1 \le \{i\!:\!c\!=\!v_1, \dots, i\!:\!c\!=\!v_m\} \le 1) \tag{8}$$
  where $\{v_1, \dots, v_m\}$ is $Dom(c)$.

Note how the translation $PF_m(D)$ treats regular and statically determined fluent constants differently: formulas (7) are included only when c is regular. Statically determined fluents are useful for describing defined fluents, whose values are determined by the fluents in the same state only. For instance, $\mathit{NotClear}(B)$ is a statically determined Boolean fluent constant defined by the static causal law

$$\begin{array}{l}\textbf{caused } \mathit{NotClear}(B)=\mathbf{t} \textbf{ if } \mathit{Loc}(B_1)= B,\\ \textbf{caused } \{\mathit{NotClear}(B)=\mathbf{f}\}^{\rm ch}.\end{array}$$

If we added (7) for $\mathit{NotClear}(B)$, that is,

$$\{0\!:\!\mathit{NotClear}(B) = \mathbf{t}\}^{\rm ch},\ \ \{0\!:\!\mathit{NotClear}(B) = \mathbf{f}\}^{\rm ch},$$

to the translation $\mathit{PF}_m(D)$, the value of $\mathit{NotClear}(B)$ at time 0 would have been arbitrary, which does not conform to the intended definition of $\mathit{NotClear}(B)$. We refer the reader to [Giunchiglia *et al.*, 2004, Section 5] for more details about the difference between regular and statically determined fluent constants.

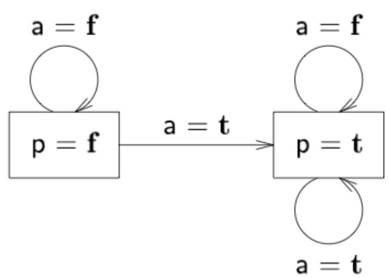

Fig. 1. The transition system described by *SD*.

Example 2. 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.

$$\begin{array}{l}\textbf{caused } p \textbf{ if } \top \textbf{ after } a,\\ \textbf{caused } \{a\}^{\rm ch} \textbf{ if } \top,\\ \textbf{caused } \{\sim\! a\}^{\rm ch} \textbf{ if } \top,\\ \textbf{caused } \{p\}^{\rm ch} \textbf{ if } \top \textbf{ after } p,\\ \textbf{caused } \{\sim\! p\}^{\rm ch} \textbf{ if } \top \textbf{ after } \sim\! p.\end{array} \tag{9}$$

The translation $\mathit{PF}_m(\mathit{SD})$ turns this description into the following propositional formulas. The first line of (9) is turned into the formulas (disregarding $\top$)

$$i\!+\!1\!:\!p \ \leftarrow \ i\!:\!a$$

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

$$\begin{array}{l}\{i\!:\!a\}^{\rm ch},\\ \{i\!:\!\sim\! a\}^{\rm ch}\end{array} \tag{10}$$

$(0 \le i < m)$, and the fourth and the fifth lines into

$$\begin{array}{l} \{i{+}1{:}p\}^{\rm ch} \leftarrow i{:}p, \\ \{i{+}1{:}{\sim}p\}^{\rm ch} \leftarrow i{:}{\sim}p \end{array} \tag{11}$$

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

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

come from (7), and

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

come from (8).

Let σ^{fl} be the subset of the signature σ consisting of atoms containing fluent constants, and let σ^{act} be the subset of σ consisting of atoms containing action constants. Since we identify an interpretation I with the set of atoms that are true in it, an interpretation of the signature σ_m can be represented in the form

$$(0 : s_0) \cup (0 : e_0) \cup (1 : s_1) \cup (1 : e_1) \cup \cdots \cup (m : s_m)$$

where $s_0, \ldots, s_m$ are interpretations of σ^{fl}, and $e_0, \ldots, e_{m-1}$ are interpretations of σ^{act}.

We define states and transitions in terms of stable models of $PF_0(D)$ and $PF_1(D)$ as follows.

Definition 1 (States and Transitions). *For any action description D of signature σ, a* state *of D is an interpretation s of σ^{fl} such that $0 : s$ is a stable model of $PF_0(D)$. A* transition *of D 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 stable model of $PF_1(D)$.*

In view of the uniqueness and existence 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.

Given these definitions, we define the transition system $T(D)$ represented by an action description D as follows.

Definition 2 (Transition System). *A transition system $T(D)$ represented by an action description D is a labeled directed graph such that the vertices are the states of D, and the edges are obtained from the transitions of D: for every transition $\langle s, e, s'\rangle$ of D, an edge labeled e goes from s to s'.*

Since the vertices and the edges of a transition system $T(D)$ are identified with the states and the transitions of D, we simply extend the definitions of a state and a transition to transition systems: A *state* of $T(D)$ is a state of D. A *transition* of $T(D)$ is a transition of D.

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

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

The stable models of $PF_m(D)$ represent the paths of length m in the transition system represented by D. For $m=0$ and $m=1$, this is clear from the definition of a transition system (Definition 2); for $m>1$ this needs to be verified as the following theorem shows.

For every set X_m of elements of the signature σ_m, let X^i ($i<m$) be the triple consisting of

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

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

For example, $\{0:\sim p, 0:\sim a, 1:\sim p, 1:a, 2:p\}$ is a stable model of $PF_2(SD)$, and each of $\langle\{\sim p\},\{\sim a\},\{\sim p\}\rangle$ and $\langle\{\sim p\},\{a\},\{p\}\rangle$ is a transition of *SD*.

5 Useful Abbreviations

Like $\mathcal{C}$+, several intuitive abbreviations of causal laws can be defined for $\mathcal{BC}$+.

Expression

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

stands for

$$\textbf{caused}\ \{c{=}v\}^{\rm ch}\ \textbf{if}\ F.$$[4]

This abbreviation is intuitive in view of the reading of choice formulas in the presence of the uniqueness and existence of value constraints (recall Example 1). Similarly,

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

stands for

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

Other abbreviations of $\mathcal{BC}$+ causal laws are defined similarly to abbreviations in $\mathcal{C}$+.

- If c is a Boolean action constant, we express that F is an effect of executing c by

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

  which stands for the fluent dynamic law

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

[4] Here and after, we often omit **if** F if F is $\top$.

- If c is an action constant, the expression

$$\textbf{exogenous } c$$

stands for the action dynamic laws

$$\textbf{default } c\!=\!v$$

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

$$\textbf{inertial } c$$

stands for the fluent dynamic laws

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

for all $v \in \mathit{Dom}(c)$.
-
$$\textbf{constraint } F$$

where F is a fluent formula stands for the static law

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

-
$$\textbf{always } F$$

stands for the fluent dynamic law

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

-
$$\textbf{nonexecutable } F \textbf{ if } G$$

stands for the fluent dynamic law

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

6 Example: Blocks World

An attractive feature of $\mathcal{BC}$+ is that aggregates are directly usable in causal laws because they can be understood as abbreviations of propositional formulas [Ferraris, 2005; Pelov *et al.*, 2003; Son and Pontelli, 2007; Lee and Meng, 2009]. We illustrate this advantage by formalizing an elaboration of the Blocks World from [Lee *et al.*, 2013].

Let *Blocks* be a nonempty finite set $\{\mathit{Block}_1, \dots, \mathit{Block}_n\}$. The action description below uses the following fluent and action constants:

- for each $B \in \mathit{Blocks}$, regular fluent constant $\mathit{Loc}(B)$ with the domain $\mathit{Blocks} \cup \{\mathit{Table}\}$, and statically determined Boolean fluent constant $\mathit{InTower}(B)$;

- for each $B \in \mathit{Blocks}$, Boolean action constant $\mathit{Move}(B)$;
- for each $B \in \mathit{Blocks}$, action constant $\mathit{Destination}(B)$ with the domain $\mathit{Blocks} \cup \{\mathit{Table}\} \cup \{\mathit{None}\}$, where *None* is a symbol for denoting an "undefined" value.

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 $\mathit{Blocks} \cup \{\mathit{Table}\}$. Below we list causal laws describing this domain.

Blocks are not on itself:

$$\textbf{constraint } \mathit{Loc}(B) \neq B.$$

The definition of $\mathit{InTower}(B)$:

$$\begin{array}{l}\textbf{caused } \mathit{InTower}(B) \textbf{ if } \mathit{Loc}(B) = \mathit{Table},\\ \textbf{caused } \mathit{InTower}(B) \textbf{ if } \mathit{Loc}(B) = B_1 \wedge \mathit{InTower}(B_1),\\ \textbf{default } \sim\mathit{InTower}(B).\end{array} \tag{12}$$

Blocks do not float in the air:

$$\textbf{constraint } \mathit{InTower}(B).$$

No two blocks are on the same block:

$$\textbf{constraint } \{b : \mathit{Loc}(b) = B\} \leq 1,$$

which is shorthand for

$$\textbf{constraint } \{\mathit{Loc}(\mathit{Block}_1) = B, \ldots, \mathit{Loc}(\mathit{Block}_n) = B\} \leq 1.$$

Only k towers are allowed to be on the table (k is a positive integer):

$$\textbf{constraint } \{b : \mathit{Loc}(b) = \mathit{Table}\} \leq \mathrm{k}.$$

The effect of moving a block:

$$\mathit{Move}(B) \textbf{ causes } \mathit{Loc}(B) = L \textbf{ if } \mathit{Destination}(B) = L.$$

A block cannot be moved unless it is clear:

$$\textbf{nonexecutable } \mathit{Move}(B) \textbf{ if } \mathit{Loc}(B_1) = B.$$

Concurrent actions are limited by the number g of grippers:

$$\textbf{always } \{b : \mathit{Move}(b)\} \leq \mathrm{g}.$$

The commonsense law of inertia:

$$\textbf{inertial } \mathit{Loc}(B).$$

Actions are exogenous:

$$\begin{array}{l}\textbf{exogenous } \mathit{Move}(B),\\ \textbf{exogenous } \mathit{Destination}(B).\end{array}$$

Destination is an attribute of *Move*:

$$\textbf{always}\ \mathit{Destination}(B) = \mathit{None} \leftrightarrow \neg \mathit{Move}(B).$$

Besides the inability to represent aggregates, other action languages have other difficulties in representing this example. Under the semantics of $\mathcal{C}$ and $\mathcal{C}$+, the recursive definition of *InTower* in (12) does not work correctly. Languages $\mathcal{B}$ and $\mathcal{BC}$ do not allow us to represent action attributes like *Destination* because they lack nonBoolean actions and action dynamic laws (The usefulness of attributes in expressing elaboration tolerance was discussed in [Lifschitz, 2000].)

7 Embedding Formulas under SM 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.

For any propositional formula F, we define the translation $\mathsf{pf2bcp}(F)$, which turns F into an "equivalent" action description in $\mathcal{BC}$+ as follows: reclassify every atom in the signature of F as a statically determined fluent constant with Boolean values, and rewrite F as the static law

$$\textbf{caused}\ F$$

and add

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

for every constant c.

We identify an interpretation I of the signature of F with an interpretation I' of the signature of $\mathsf{pf2bcp}(D)$ as follows: for all atoms c in the signature of F, $I(c) = v$ iff $I' \models c = v$ ($v \in \{\mathbf{t}, \mathbf{f}\}$). Due to the presence of $\mathit{UEC}_{\{c=\mathbf{t}, c=\mathbf{f}\}}$ in $\mathsf{pf2bcp}(D)$, the mapping also tells us that any interpretation satisfying $\mathsf{pf2bcp}(D)$ has a corresponding interpretation of the signature of F.

Theorem 3 *For any propositional formula F of a finite signature and any interpretation I of that signature, I is a stable model of F iff I' is a state of the transition system represented by the $\mathcal{BC}$+ description* $\mathsf{pf2bcp}(F)$.

It is known that the problem of determining the existence of stable models of propositional formulas is Σ_2^P-complete [Ferraris, 2005]. The same complexity applies to $\mathcal{BC}$+ in view of Proposition 3. On the other hand, from the translation $\mathit{PF}_m(D)$, a useful fragment in NP can be defined based on the known results in ASP. The following is an instance, which we call "simple" action descriptions.

We say that action description D is *definite* if the head of every causal law is either $\bot$, an atom $c = v$, or a choice formula $\{c = v\}^{\mathrm{ch}}$. We say that a formula is a *simple* conjunction if it is a conjunction of atoms and count aggregate expressions, each of which possibly preceded by negation. A *simple* action description is a definite action description such that G in every causal law (3) is a simple conjunction, and G and H in every causal law (4) are simple conjunctions. The Blocks World formalization in the previous section is an example of a simple action description.

8 Relation to Language $\mathcal{BC}$

8.1 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-valued. The main syntactic differences between $\mathcal{BC}$ causal laws and $\mathcal{BC}$+ causal laws are that the former allows only the conjunction of atoms in the body, and distinguishes between **if** and **ifcons** clauses.

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

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

$(n \geq m \geq 0)$ where each A_i is an atom containing a fluent constant. It expresses, informally speaking, that every state satisfies A_0 if it satisfies $A_1, \dots, A_m$, and $A_{m+1}, \dots, A_n$ can be consistently assumed.

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

$$A_0 \textbf{ after } A_1, \dots, A_m \textbf{ ifcons } A_{m+1}, \dots, A_n \tag{14}$$

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

- A_0 is an atom containing a regular fluent constant,
- each of $A_1, \dots, A_m$ is an atom containing a fluent constant, or $a = \mathbf{t}$ where a is an action constant, and
- $A_{m+1}, \dots, A_n$ 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 $A_1, \dots, A_m$, and $A_{m+1}, \dots, A_n$ can be consistently assumed about the end state.

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

Like $\mathcal{BC}$+, the semantics of $\mathcal{BC}$ is defined by reduction $PF_m^{\mathcal{BC}}$ to a sequence of logic programs under the stable model semantics. The signature σ_m of $PF_m^{\mathcal{BC}}$ is defined the same as that of PF_m defined in Section 4.

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 $i \in \{0, \dots, m\}$;
- $$(i+1) : A_0 \;\leftarrow\; i : (A_1 \wedge \cdots \wedge A_m) \wedge (i+1) : (\neg\neg A_{m+1} \wedge \cdots \wedge \neg\neg A_n) \tag{16}$$
  for every $\mathcal{BC}$ dynamic law (14) in D and every $i \in \{0, \dots, m-1\}$;
- the formula $i : (a = \mathbf{t} \vee a = \mathbf{f})$ for every action constant a and every $i \in \{0, \dots, m-1\}$;
- the formula (7) for every regular fluent constant c and every element $v \in \mathit{Dom}(c)$;
- UEC_{σ_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 $\mathcal{BC}$+, only one **if** clause is enough since the formulas are understood under the stable model semantics. We explore this difference in more detail in Section 8.3.

8.2 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 $\mathsf{bc2bcp}(D)$, which turns a $\mathcal{BC}$ description into an equivalent $\mathcal{BC}$+ description as follows:

- replace every causal law (13) with

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

- replace every causal law (14) with

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

- add the causal laws

$$\textbf{exogenous } a$$

  for every action constant a.

Theorem 4 *For any action description D in language $\mathcal{BC}$, the transition system described by D is identical to the transition system described by the description* $\mathsf{bc2bcp}(D)$ *in language $\mathcal{BC}$+.*

8.3 Comparing $\mathcal{BC}$+ with $\mathcal{BC}$

In $\mathcal{BC}$, every action is assumed to be Boolean, and action dynamic laws are not available, which prevents us from describing defeasible causal laws [Giunchiglia *et al.*, 2004, Section 4.3] and action attributes [Giunchiglia *et al.*, 2004, Section 5.6], that $\mathcal{BC}$+ and $\mathcal{C}$+ are able to express conveniently. Also, syntactically, $\mathcal{BC}$ is not expressive enough to describe dependencies among actions. For a simple example, in $\mathcal{BC}$+ and $\mathcal{C}$+, we can express that action a_1 is not executable when a_2 is not executed at the same time by the fluent dynamic law

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

but this is not even 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 the subtlety can be avoided by using the **default** proposition (Section 5) as the following example illustrates.

Consider the leaking container example from [Lee *et al.*, 2013] 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

$$\textbf{default } \mathit{Amount}\!=\!x \textbf{ after } \mathit{Amount}\!=\!x\!+\!k, \tag{17}$$

which stands for fluent dynamic law

$$\textbf{caused } \{\mathit{Amount}\!=\!x\}^{\rm ch} \textbf{ after } \mathit{Amount}\!=\!x\!+\!k,$$

which is shorthand for propositional formulas

$$\{i+1:\mathit{Amount}=x\}^{\rm ch} \leftarrow i:\mathit{Amount}=x+k \qquad (18)$$

$(i < m)$.

The **default** abbreviation is also defined in $\mathcal{BC}$ in a syntactically different, but semantically equivalent way. In $\mathcal{BC}$, the assertion (17) stands for the causal law

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

which is further turned into

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

$(i < m)$, which is strongly equivalent to (18).[5]

9 Relation to $\mathcal{C}$+

9.1 Review: $\mathcal{C}$+

As mentioned earlier, the syntax of $\mathcal{C}$+ is similar to the syntax of $\mathcal{BC}$+. The signature is defined the same as in $\mathcal{BC}$+. A $\mathcal{C}$+ *static law* is an expression of the form (3) where F and G are fluent formulas. A $\mathcal{C}$+ *action dynamic law* is an expression of the form (3) in which F is an action formula and G is a formula. A $\mathcal{C}$+ *fluent dynamic law* is an expression of the form (4) where F and G are fluent formulas and H is a formula, provided that F does not contain statically determined constants. A $\mathcal{C}$+ *causal law* is a static law, an action dynamic law, or a fluent dynamic law. A $\mathcal{C}$+ *action description* is a set of $\mathcal{C}$+ causal laws. We say that $\mathcal{C}$+ action description D is *definite* if the head of every causal law is either $\bot$ or an atom $c=v$.

The original semantics of $\mathcal{C}$+ is defined in terms of reduction to nonmonotonic causal theories in [Giunchiglia *et al.*, 2004]. In [Lee, 2012], the semantics of the definite $\mathcal{C}$+ description is equivalently reformulated in terms of reduction to propositional formulas under the the stable model semantics as follows. [6]

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

- $$i:F \leftarrow\ \neg\neg\,(i:G) \qquad (19)$$
  for every static law (3) in D and every $i \in \{0,\dots,m\}$, and for every action dynamic law (3) in D and every $i \in \{0,\dots,m-1\}$;
- $$i+1:F \leftarrow\ \neg\neg(i+1:G) \wedge (i:H) \qquad (20)$$
  for every fluent dynamic law (4) in D and every $i \in \{0,\dots,m-1\}$;
- the formula (7) for every regular fluent constant c and every $v \in \mathit{Dom}(c)$;
- UEC_{σ_m}.

[5] About propositional formulas F and G we say that F is *strongly equivalent* to G if, for every propositional formula H, $F \wedge H$ has the same stable models as $G \wedge H$ [Lifschitz *et al.*, 2001].

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

Notation: s, s' range over $\{Switch_1, Switch_2\}$; x, y range 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** $Status(s)\!=\!y$ $\quad (x \neq y)$
caused $Status(s)\!=\!x$ **if** $Status(s')\!=\!y$ $\quad (s \neq s', x \neq y)$
inertial $Status(s)$
exogenous $Flip(s)$

Fig. 2. Two Switches in $\mathcal{BC}$+

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

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

- replace every $\mathcal{C}$+ causal law (3) with

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

- replace every $\mathcal{C}$+ causal law (4) with

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

The following theorem asserts the correctness of this translation.

Theorem 5 *For any definite action description D in language $\mathcal{C}$+, the transition system described by D is identical to the transition system described by the description* $\mathsf{cp2bcp}(D)$ *in language $\mathcal{BC}$+.*

9.3 Comparing $\mathcal{BC}$+ with $\mathcal{C}$+

The embedding of $\mathcal{C}$+ in $\mathcal{BC}$+ tells us that **if** clauses always introduce double negations, whose presence leads to the fact that stable models are not necessarily minimal models. This accounts for the fact that the definite fragment of $\mathcal{C}$+ does not handle the concept of transitive closure correctly. For example, the recursive definition of $InTower(B)$ in Section 6 does not work correctly if it is understood as $\mathcal{C}$+ causal laws. The inability to consider minimal models in such cases introduces some unintuitive behavior of $\mathcal{C}$+ in representing causal dependencies among fluents, as the following example shows.

Consider two switches which can be flipped but cannot be both up or down at the same time.[7] If one of them is down and the other is up, the direct effect of flipping only

[7] This example is similar to Lin's suitcase example [Lin, 1995], but a main difference is that the same fluent is affected by both direct and indirect effects of an action.

one switch is changing the status of that 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 because, according to the semantics of $\mathcal{C}$+, this is causally explained by the cyclic causality. This is obviously unintuitive.

10 Implementation

System CPLUS2ASP [Babb and Lee, 2013] was originally designed to compute the definite fragment of $\mathcal{C}$+ using ASP solvers as described in [Lee *et al.*, 2013]. Its version 2 supports extensible multi-modal translations for other action languages as well. As the translation $PF_m^{\mathcal{C}+}(D)$ for $\mathcal{C}$+ is similar to the translation $PF_m(D)$ for $\mathcal{BC}$+, the extension is straightforward. We modified system CPLUS2ASP to be able to accept $\mathcal{BC}$+ as another input language.

The $\mathcal{BC}$+ formalization of the Blocks World domain can be represented in the input language of CPLUS2ASP under the $\mathcal{BC}$+ mode as shown in Figure 3.

The input language of CPLUS2ASP allows the users to conveniently express declarations and causal laws. Its syntax follows the syntax of Version 2 of the CCALC input language.[8] The extent of each sort (i.e., domain) is defined in the object declaration section. The sort declaration denotes that `block` is a subsort of `location`, meaning that every object of sort `block` is an object of `location` as well. The constant declaration

```
loc(block)              :: inertialFluent(location)
```

has the same meaning as the declaration

```
loc(block)              :: simpleFluent(location)
```

(`simpleFluent` is a keyword for regular fluent) accompanied by the dynamic law

```
inertial loc(B).
```

The constant declaration

```
destination(block)    :: attribute(location*) of move(block).
```

has the same meaning as the declaration

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

```
% File 'blocks'

:- sorts
    location >> block.

:- objects
    b(1..10)                :: block;
    table                   :: location.

:- constants
    loc(block)              :: inertialFluent(location);
    in_tower(block)         :: sdFluent;
    move(block)             :: exogenousAction;
    dest(block)             :: attribute(location*) of move(block).

:- variables
    B,B1,B2                 :: block;
    L                       :: location.

constraint -(loc(B)=B).

caused in_tower(B) if loc(B)=table.
caused in_tower(B) if loc(B)=B1 & in_tower(B1).
default ~in_tower(B).

constraint in_tower(B).
constraint {B1| loc(B1)=B}1.
constraint {B1| loc(B1)=table}k.
move(B) causes loc(B)=L if dest(B)=L.
nonexecutable move(B) if loc(B1)=B.
always {B1| move(B1)}g.

:- query
label :: test;
0: loc(b(1))=table & loc(b(2))=b(1) & loc(b(3))=b(2)
   & loc(b(4))=b(3) & loc(b(5))=b(4);
0: loc(b(6))=table & loc(b(7))=b(6) & loc(b(8))=b(7)
   & loc(b(9))=b(8) & loc(b(10))=b(9);
maxstep: loc(b(1))=b(10).
```

Fig. 3. Blocks World in the input language of CPLUS2ASP under the $\mathcal{BC}$+ mode

```
destination(block)    :: action(location*)
```

accompanied by the dynamic law

```
exogenous destination(B).
always destination(B)=none <-> -move(B).
```

where `location*` is a new sort implicitly declared by CPLUS2ASP and consists of `locations` and the auxiliary symbol `none`.

We refer the reader to the system homepage

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

for the details of the input language syntax. In order to run this program we invoke CPLUS2ASP as follows.

```
cplus2asp -l bc+ blocks k=3 g=2 query=test
```

The option `-l bc+` instructs CPLUS2ASP to operate under the $\mathcal{BC}$+ semantics. "`k=3 g=2`" are constant definitions for the number `k` of towers, and the number `g` of grippers for the domain.

```
% 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 :: test;
   maxstep :: 1;
   0: sw_status(s1)=off & sw_status(s2)=on.
```

Fig. 4. Two Switches in the input language of CPLUS2ASP under the $\mathcal{BC}$+ mode

For another example, Figure 4 represents the $\mathcal{BC}$+ description in Figure 2 in the input language of CPLUS2ASP. The following is the command line to find all four transitions described in Section 9.3.

```
$ cplus2asp -l bc+ switch query=test 0
```

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
```

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

```
cplus2asp -l c+ switch query=test 0
```

one more (unintuitive) transition is returned:

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

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

11 Conclusion

Unlike many other action languages which can be understood as high level notations of limited forms of logic programs, $\mathcal{BC}$+ is defined as a high level notation of the general stable model semantics for propositional formulas. This approach allows for employing modern ASP language constructs directly in $\mathcal{BC}$+, as they can be understood as shorthand for propositional formulas, and thus allows for closing the gap between action languages and the modern ASP language. It also accounts for the expressivity of $\mathcal{BC}$+ for embedding other action languages, and allows reasoning about transition systems described in $\mathcal{BC}$+ to be computed by ASP solvers.

A further extension of $\mathcal{BC}$+ is possible by replacing the role of propositional formulas with a more expressive generalized stable model semantics. It is straightforward to extend $\mathcal{BC}$+ to the first-order level by using the first-order stable model semantics from [Ferraris *et al.*, 2011] or its extension with generalized quantifiers [Lee and Meng, 2012] in place of propositional formulas. This will allow $\mathcal{BC}$+ to include other constructs, such as external atoms and nonmonotonic dl-atoms, as they are instances of generalized quantifiers as shown in [Lee and Meng, 2012].

Also, some recent advances in ASP solving can be applied to action languages. Our future work includes extending $\mathcal{BC}$+ to handle external events arriving online based on the concept of online answer set solving [Gebser *et al.*, 2011], and compute it using online answer set solver OCLINGO.[9]

Acknowledgements

We are grateful to Michael Bartholomew, Vladimir Lifschitz and the anonymous referees for their useful comments. This work was partially supported by the National Science Foundation under Grant IIS-1319794, South Korea IT R&D program MKE/KIAT 2010-TD-300404-001, and ICT R&D program of MSIP/IITP 10044494 (WiseKB).

References

[Babb and Lee, 2013] Joseph Babb and Joohyung Lee. Cplus2ASP: Computing action language $\mathcal{C}$+ in answer set programming. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 122–134, 2013.

[Babb and Lee, 2015] Joseph Babb and Joohyung Lee. Action language $\mathcal{BC}$+: Preliminary report. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

[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, 2014] Michael Bartholomew and Joohyung Lee. Stable models of multi-valued formulas: partial vs. total functions. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 583–586, 2014.

[Ferraris *et al.*, 2009] Paolo Ferraris, Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. Symmetric splitting in the general theory of stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 797–803. AAAI Press, 2009.

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

[9] http://www.cs.uni-potsdam.de/wv/oclingo/

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

[Ferraris, 2005] Paolo Ferraris. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 119–131, 2005.

[Gebser *et al.*, 2011] Martin Gebser, Torsten Grote, Roland Kaminski, and Torsten Schaub. Reactive answer set programming. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 54–66. Springer, 2011.

[Gelfond and Lifschitz, 1993] Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

[Gelfond and Lifschitz, 1998] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 3:195–210, 1998. `http://www.ep.liu.se/ea/cis/1998/016/`.

[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.

[Harrison *et al.*, 2014] Amelia J. Harrison, Vladimir Lifschitz, and Fangkai Yang. The semantics of Gringo and infinitary propositional formulas. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014*, 2014.

[Lee and Meng, 2009] Joohyung Lee and Yunsong Meng. On reductive semantics of aggregates in answer set programming. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 182–195, 2009.

[Lee and Meng, 2012] Joohyung Lee and Yunsong Meng. Stable models of formulas with generalized quantifiers. In *Proceedings of International Workshop on Nonmonotonic Reasoning (NMR)*, 2012. `http://peace.eas.asu.edu/joolee/papers/smgq-nmr.pdf`.

[Lee *et al.*, 2013] Joohyung Lee, Vladimir Lifschitz, and Fangkai Yang. Action language $\mathcal{BC}$: Preliminary report. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[Lee, 2012] Joohyung Lee. Reformulating action language $\mathcal{C}$+ in answer set programming. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning*, volume 7265 of *Lecture Notes in Computer Science*, pages 405–421. Springer, 2012.

[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, 2000] Vladimir Lifschitz. Missionaries and cannibals in the Causal Calculator. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 85–96, 2000.

[Lin, 1995] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1985–1991, 1995.

[Marek and Truszczynski, 2004] Victor W. Marek and Miroslaw Truszczynski. Logic programs with abstract constraint atoms. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 86–91, 2004.

[Pelov *et al.*, 2003] Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe. Translation of aggregate programs to normal logic programs. In *Proceedings Answer Set Programming*, 2003.

[Son and Pontelli, 2007] Tran Cao Son and Enrico Pontelli. A constructive semantic characterization of aggregates in answer set programming. *TPLP*, 7(3):355–375, 2007.

A Proofs of Theorems 1 and 2

For the proofs below, it is convenient to use the following generalization over the stable model semantics from [Ferraris, 2005], which is the propositional case of the first-order stable model semantics from [Ferraris *et al.*, 2011].

For any two interpretations I, J of the same propositional signature and any list $\mathbf{p}$ of distinct atoms, we write $J <^{\mathbf{p}} I$ if

- J and I agree on all atoms not in $\mathbf{p}$, and
- J is a proper subset of I.

I is a stable model of F relative to $\mathbf{p}$, denoted by $I \models \text{SM}[F; \mathbf{p}]$, if I is a model of F and there is no interpretation J such that $J <^{\mathbf{p}} I$ and J satisfies F^I.

When $\mathbf{p}$ is empty, this notion of a stable model coincides with the notion of a model in propositional logic. When $\mathbf{p}$ is the same as the underlying signature, the notion reduces to the notion of a stable model from [Ferraris, 2005].

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

Proof. We will use the following notations: $\mathit{SD}(i)$ is the set of formulas (5) in $\mathit{PF}_m(D)$ obtained from the static laws (3) in D; $\mathit{AD}(i)$ is the set of formulas (5) in $\mathit{PF}_m(D)$ obtained from the action dynamic laws (3) in D; $\mathit{FD}(i)$ is the set of formulas (6) in $\mathit{PF}_m(D)$ obtained from the fluent dynamic laws (4) in D. For the signature σ of D, signature σ^r is the subset of σ consisting of atoms containing regular fluent constants; signature σ^{sd} is the subset of σ consisting of atoms containing statically determined fluent constants; signature σ^{fl} is the union of σ^r and σ^{sd}; signature σ^{act} is the subset of σ consisting of atoms containing action constants.

Since $\langle s, e, s' \rangle$ is a transition,

$$\begin{aligned} & 0\!:\!s \cup 0\!:\!e \cup 1\!:\!s' \models \\ & \quad \text{SM}[\mathit{SD}(0) \cup \mathit{AD}(0) \cup \mathit{FD}(0) \cup \mathit{SD}(1);\ 0\!:\!\sigma^{sd} \cup 0\!:\!\sigma^{act} \cup 1\!:\!\sigma^{r} \cup 1\!:\!\sigma^{sd}] \\ & \quad \wedge \mathit{UEC}_{0:\sigma^{fl} \cup 0:\sigma^{act} \cup 1:\sigma^{fl}}. \end{aligned}$$

By the splitting theorem [Ferraris *et al.*, 2009], it follows that

$$0\!:\!s \cup 0\!:\!e \cup 1\!:\!s' \models \text{SM}[\mathit{SD}(0);\ 0\!:\!\sigma^{sd}]; \tag{21}$$

$$0\!:\!s \cup 0\!:\!e \cup 1\!:\!s' \models \text{SM}[\mathit{FD}(0) \wedge \mathit{SD}(1);\ 1\!:\!\sigma^{r} \cup 1\!:\!\sigma^{sd}]; \tag{22}$$

$$0\!:\!s \cup 0\!:\!e \cup 1\!:\!s' \models \mathit{UEC}_{0:\sigma^{fl} \cup 0:\sigma^{act} \cup 1:\sigma^{fl}}.$$

From (21), we have $0\!:\!s \models \text{SM}[\mathit{SD}(0);\ 0\!:\!\sigma^{sd}]$, and consequently, $0\!:\!s \models \text{SM}[\mathit{SD}(0) \wedge \mathit{UEC}_{0:\sigma^{fl}};\ 0\!:\!\sigma^{sd}]$, so s is a state.

From (22), by Theorem 2 from [Ferraris *et al.*, 2011], it follows that

$$0\!:\!s \cup 0\!:\!e \cup 1\!:\!s' \models \text{SM}[\mathit{FD}(0) \wedge \mathit{SD}(1);\ 1\!:\!\sigma^{sd}]. \tag{23}$$

Since $\mathit{FD}(0)$ is negative on $1\!:\!\sigma^{sd}$ (cf. [Ferraris *et al.*, 2011]), (23) is equivalent to

$$0\!:\!s \cup 0\!:\!e \cup 1\!:\!s' \models \text{SM}[\mathit{SD}(1);\ 1\!:\!\sigma^{sd}] \wedge \mathit{FD}(0),$$

Consequently, we get

$$1\!:\!s' \models \mathrm{SM}[\mathit{SD}(1) \wedge \mathit{UEC}_{1:\sigma^{fl}};\ 1\!:\!\sigma^{sd}],$$

which can be rewritten as

$$0\!:\!s' \models \mathrm{SM}[\mathit{SD}(0) \wedge \mathit{UEC}_{0:\sigma^{fl}};\ 0\!:\!\sigma^{sd}],$$

so s' is a state.

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

Proof. When $m = 1$, the claim is immediate from the definition of a transition.

I.H. Assume that X_m is a stable model of $\mathit{PF}_m(D)$ iff $X^0, \ldots, X^{m-1}$ are transitions of D ($m \geq 1$).

We first prove that if X_{m+1} is a stable model of $\mathit{PF}_{m+1}(D)$, then $X^0, \ldots, X^{m+1}$ are transitions of D. Assume that X_{m+1} is a stable model of $\mathit{PF}_{m+1}(D)$.

Note that X_{m+1} is a stable model of $\mathit{PF}_{m+1}(D)$ iff X_{m+1} satisfies $\mathit{UEC}_{\sigma_{m+1}}$ and

$$\begin{array}{l} \mathrm{SM}[\mathit{SD}(0) \wedge \mathit{AD}(0) \wedge \mathit{FD}(0) \wedge \mathit{SD}(1) \\ \qquad\qquad \wedge\, \mathit{AD}(1) \wedge \mathit{FD}(1) \wedge \mathit{SD}(2) \\ \qquad\qquad \wedge\ \ldots \\ \qquad\qquad \wedge\, \mathit{AD}(m{-}1) \wedge \mathit{FD}(m{-}1) \wedge \mathit{SD}(m) \\ \qquad\qquad \wedge\, \mathit{AD}(m) \wedge \mathit{FD}(m) \wedge \mathit{SD}(m{+}1)\ ; \\ \qquad\qquad\qquad\qquad 0\!:\!\sigma^{sd} \cup 0\!:\!\sigma^{act} \cup 1\!:\!\sigma^{r} \cup 1\!:\!\sigma^{sd} \\ \qquad\qquad\qquad\qquad\quad \cup\, 1\!:\!\sigma^{act} \cup 2\!:\!\sigma^{r} \cup 2\!:\!\sigma^{sd} \\ \qquad\qquad\qquad\qquad\quad \cup \ldots \\ \qquad\qquad\qquad\qquad\quad \cup\, (m{-}1)\!:\!\sigma^{act} \cup m\!:\!\sigma^{r} \cup m\!:\!\sigma^{sd} \\ \qquad\qquad\qquad\qquad\quad \cup\, m\!:\!\sigma^{act} \cup (m{+}1)\!:\!\sigma^{r} \wedge (m{+}1)\!:\!\sigma^{sd}]. \end{array} \tag{24}$$

By the splitting theorem [Ferraris *et al.*, 2009], the fact that X_{m+1} satisfies (24) is equivalent to saying that X_{m+1} satisfies

$$\begin{array}{l} \mathrm{SM}[\mathit{SD}(0) \wedge \mathit{AD}(0) \wedge \mathit{FD}(0) \wedge \mathit{SD}(1) \\ \qquad\qquad \wedge\, \mathit{AD}(1) \wedge \mathit{FD}(1) \wedge \mathit{SD}(2) \\ \qquad\qquad \wedge\ \ldots \\ \qquad\qquad \wedge\, \mathit{AD}(m{-}1) \wedge \mathit{FD}(m{-}1) \wedge \mathit{SD}(m)\ ; \\ \qquad\qquad\qquad\qquad 0\!:\!\sigma^{sd} \cup 0\!:\!\sigma^{act} \cup 1\!:\!\sigma^{r} \cup 1\!:\!\sigma^{sd} \\ \qquad\qquad\qquad\qquad\quad \cup\, 1\!:\!\sigma^{act} \cup 2\!:\!\sigma^{r} \cup 2\!:\!\sigma^{sd} \\ \qquad\qquad\qquad\qquad\quad \cup \ldots \\ \qquad\qquad\qquad\qquad\quad \cup\, (m{-}1)\!:\!\sigma^{act} \cup m\!:\!\sigma^{r} \cup m\!:\!\sigma^{sd}] \end{array} \tag{25}$$

and

$$\mathrm{SM}[\mathit{AD}(m) \wedge \mathit{FD}(m) \wedge \mathit{SD}(m{+}1);\ m\!:\!\sigma^{act} \cup (m{+}1)\!:\!\sigma^{r} \cup (m{+}1)\!:\!\sigma^{sd}]. \tag{26}$$

The fact that $X_{m+1} \models$ (25) is equivalent to saying that

$$X_{m+1} \cap \{0\!:\!\sigma \cup \cdots \cup m\!:\!\sigma\} \models \text{(25)}.$$

By I.H., the latter is equivalent to saying that $X^0, \ldots, X^{m-1}$ are transitions of D.

Observe that, using the splitting theorem, (25) entails

$$\text{SM}[\mathit{FD}(m-1) \wedge \mathit{SD}(m);\ m\!:\!\sigma^r \cup m\!:\!\sigma^{sd}]. \tag{27}$$

By Theorem 2 from [Ferraris *et al.*, 2011], (27) entails

$$\text{SM}[\mathit{FD}(m-1) \wedge \mathit{SD}(m);\ m\!:\!\sigma^{sd}]. \tag{28}$$

Since $\mathit{FD}(m-1)$ is negative on $m\!:\!\sigma^{sd}$, (28) entails

$$\text{SM}[\mathit{SD}(m);\ m\!:\!\sigma^{sd}]. \tag{29}$$

By the splitting theorem on (26) and (29), X_{m+1} satisfies

$$\begin{array}{r}\text{SM}[\mathit{SD}(m) \wedge \mathit{AD}(m) \wedge \mathit{FD}(m) \wedge \mathit{SD}(m+1); \qquad\qquad\qquad\qquad \\ m\!:\!\sigma^{sd} \cup m\!:\!\sigma^{act} \cup (m+1)\!:\!\sigma^r \cup (m+1)\!:\!\sigma^{sd}],\end{array} \tag{30}$$

or equivalently,

$$0\!:\!X^m \models \text{SM}[\mathit{SD}(0) \wedge \mathit{AD}(0) \wedge \mathit{FD}(0) \wedge \mathit{SD}(1);\ 0\!:\!\sigma^{sd} \cup 0\!:\!\sigma^{act} \cup 1\!:\!\sigma^r \cup 1\!:\!\sigma^{sd}].$$

The latter, together with the fact that $0 : X^m$ satisfies UEC_{σ_1}, means that X^m is a transition of D.

We next prove that if $X^0, \ldots, X^{m+1}$ are transitions of D, then X_{m+1} is a stable model of $\mathit{PF}_{m+1}(D)$. Assume that $X^0, \ldots, X^{m+1}$ are transitions of D. By I.H., it follows that X_{m+1} satisfies (25) and (30). By the splitting theorem on (30), X_{m+1} satisfies

$$\text{SM}[\mathit{AD}(m) \wedge \mathit{FD}(m) \wedge \mathit{SD}(m+1);\ m\!:\!\sigma^{act} \cup (m+1)\!:\!\sigma^r \cup (m+1)\!:\!\sigma^{sd}]. \tag{31}$$

From the fact that X_{m+1} satisfies (25) and (31), by the splitting theorem, we get X_{m+1} satisfies (24). It is clear that X_{m+1} satisfies $\mathit{UEC}_{\sigma_{m+1}}$. Consequently, X_{m+1} is a stable model of $\mathit{PF}_m(D)$.

Theorem 3 *For any propositional formula F of a finite signature and any interpretation I of that signature, I is a stable model of F iff I' is a state of the transition system represented by the $\mathcal{BC}$+ description* $\mathsf{pf2bcp}(F)$.

Proof. We refer the reader to [Bartholomew and Lee, 2014] for the definition of a multi-valued formula.

Let $\mathbf{c}$ be the propositional signature of F. We identify $\mathbf{c}$ with the multi-valued signature where each atom is identified with a Boolean constant, and identify F with the multi-valued formula in the multi-valued signature by identifying every occurrence of an atom c with $c = \mathbf{t}$. Further, we identify an interpretation of the propositional signature $\mathbf{c}$ with an interpretation of the multi-valued signature as follows: a propositional interpretation I satisfies a propositional atom c iff I, understood as a multi-valued interpretation, satisfies the multi-valued atom $c = \mathbf{t}$.

By Corollary 1 (a) from [Bartholomew and Lee, 2012],

$$I \text{ is a propositional stable model of } F \text{ relative to } \mathbf{c}$$

is equivalent to saying that

$$I \text{ is a multi-valued stable model of } F \wedge DF \text{ relative to } \mathbf{c}, \tag{32}$$

where DF is the conjunction of $\{c = \mathbf{f}\}^{\text{ch}}$ for all $c \in \mathbf{c}$.

Let $\mathbf{c}'$ be the propositional signature consisting of $c = \mathbf{t}$, $c = \mathbf{f}$ for all c in the multi-valued signature $\mathbf{c}$, and let I' be the interpretation of $\mathbf{c}'$ such that $I(c) = v$ iff $I' \models c = v$ ($v \in \{\mathbf{t}, \mathbf{f}\}$).

By Theorem 1 from [Bartholomew and Lee, 2014], (32) is equivalent to saying that

$$I' \text{ is a propositional stable model of } F \wedge DF \wedge UEC_{\mathbf{c}'} \text{ relative to } \mathbf{c}'.$$

The conclusion follows because $0 : (F \wedge DF \wedge UEC_{\mathbf{c}'})$ is the formula resulting from $\mathsf{pf2bcp}(F)$.

Theorem 4 *For any action description D in language $\mathcal{BC}$, the transition system described by D is identical to the transition system described by the description* $\mathsf{bc2bcp}(D)$ *in language $\mathcal{BC}$+.*

Proof. The proof can be established by showing strong equivalence between the propositional formula $PF_m^{\mathcal{BC}}(D)$ and the propositional formula $PF_m(\mathsf{bc2bcp(D)})$. The only non-trivial thing to check is that, for each action a,

$$(a = \mathbf{t} \vee a = \mathbf{f}) \wedge UEC_{\{a=\mathbf{t}, a=\mathbf{f}\}}$$

is strongly equivalent to $\{a = \mathbf{t}\}^{\text{ch}} \wedge \{a = \mathbf{f}\}^{\text{ch}} \wedge UEC_{\{a=\mathbf{t}, a=\mathbf{f}\}}$. In view of Theorem 9 from [Ferraris *et al.*, 2011], it is sufficient to check that under the assumption that $(a^* \rightarrow a) \wedge UEC_{\{a=\mathbf{t}, a=\mathbf{f}\}}$,

$$(a^* = \mathbf{t} \vee a^* = \mathbf{f})$$

is classically equivalent to

$$(a^* = \mathbf{t} \vee \neg(a = \mathbf{t})) \wedge (a^* = \mathbf{f} \vee \neg(a = \mathbf{f})),$$

which is clear.

Theorem 5 *For any definite action description D in language $\mathcal{C}$+, the transition system described by D is identical to the transition system described by the description* $\mathsf{cp2bcp}(D)$ *in language $\mathcal{BC}$+.*

Proof. The proof can be established by showing the strong equivalence between the propositional formula $PF_m^{\mathcal{C}+}(D)$ and the propositional formula $PF_m(\mathsf{cp2bcp}(D))$. This is easy to check.