Reformulating Action Language $\mathcal{C}$+ in Answer Set Programming

Joohyung Lee

School of Computing, Informatics and Decision Systems Engineering
Arizona State University
Tempe, AZ, 85287, USA
joolee@asu.edu

Abstract. Action language $\mathcal{C}$+ is a high level notation of nonmonotonic causal logic for describing properties of actions. The definite fragment of $\mathcal{C}$+ is implemented in Version 2 of the Causal Calculator (CCALC) based on the reduction of nonmonotonic causal logic to propositional logic. On the other hand, here we present two reformulations of the definite fragment of $\mathcal{C}$+ in terms of different versions of the stable model semantics. The first reformulation is in terms of the recently proposed stable model semantics of formulas with intensional functions, and can be encoded in the input language of CSP solvers. The second reformulation is in terms of the stable model semantics of propositional logic programs, which can be encoded in the input language of ASP systems. The second one is obtained from the first one by eliminating intensional functions in favor of intensional predicates.

1 Introduction

Action languages are formal models of parts of natural language that are used for describing properties of actions. Among them, language $\mathcal{C}$+ [1] and its predecessor $\mathcal{C}$ [2] are based on nonmonotonic causal logic. The definite fragment of nonmonotonic causal logic can be turned into propositional logic by the literal completion method, which resulted in an efficient way to compute $\mathcal{C}$+ using propositional satisfiability (SAT) solvers. The Causal Calculator (CCALC) is an implementation of this idea. Language $\mathcal{C}$+ has many features that are not available in $\mathcal{C}$, such as being able to represent multi-valued formulas, defined fluents, additive fluents, rigid constants and defeasible causal laws.

Nonmonotonic causal logic is closely related to logic programs under the stable model semantics [5, 6]. Proposition 6.7 from [3] states how a fragment of Boolean-valued causal logic can be turned into logic programs under the stable model semantics [6]. This result was extended to non-definite theories and to first-order causal theories in [7]. Based on these embeddings, Casolary and Lee [8] show how to represent the language of CCALC in the input language of ASP systems following these steps: (i) turn the given $\mathcal{C}$+ action description D into the corresponding multi-valued causal theory D_m; (ii) turn D_m into a Boolean-valued causal theory D'_m; (iii) turn D'_m into formulas with intensional predicates under the stable model semantics; (iv) turn the result further into an answer set program. The prototype implementation CPLUS2ASP

reported there takes the advantage of answer set solvers to yield efficient computation that is orders of magnitude faster than CCALC on several benchmark examples.

In this note, we provide an alternative reformulation of $\mathcal{C}$+ in answer set programming. Instead of step (ii) above, we turn D_m into multi-valued propositional formulas under the stable model semantics, which is a special case of first-order formulas with intensional functions [9]. The resulting theory can be encoded in the input language of CSP solvers, or it can be further turned into the input language of ASP systems by eliminating intensional functions in favor of intensional predicates.

2 Preliminaries

2.1 Multi-Valued Propositional Formulas

We first review the definition of a multi-valued propositional formula from [1], where atomic parts of a formula can be equalities of the kind found in constraint satisfaction problems.

A *(multi-valued propositional) signature* is a set σ of symbols called *constants*, along with a nonempty finite set $Dom(c)$ of symbols, disjoint from σ, assigned to each constant c. We call $Dom(c)$ the *domain* of c. A *Boolean* constant is one whose domain is the set $\{\text{TRUE}, \text{FALSE}\}$. An *atom* of a signature σ is an expression of the form $c{=}v$ ("the value of c is v") where $c \in \sigma$ and $v \in Dom(c)$. A *(multi-valued propositional) formula* of σ is a propositional combination of atoms.

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

2.2 Nonmonotonic Causal Theories and $\mathcal{C}$+

Let σ be a multi-valued propositional signature. A *(multi-valued propositional) causal rule* is an expression of the form

$$F \Leftarrow G\,, \tag{1}$$

where F and G are multi-valued propositional formulas. A *(multi-valued propositional) causal theory* is a finite set of causal rules.

Let T be a causal theory, and let I be a multi-valued propositional interpretation of its signature. The *reduct* of T relative to I, denoted by T^I, is the set of the heads of all rules in T whose bodies are satisfied by I. We say that I is a *(causal) model* of T if I is the unique model of T^I.

A causal theory is called *definite* if the heads of the rules are either an atom or $\bot$.

Language $\mathcal{C}$+ is a high level notation for causal theories that was designed for describing transition systems—directed graphs whose vertices represent states and edges are labeled by actions that affect the states. In $\mathcal{C}$+, constants in σ are partitioned into the set σ^{fl} of *fluent* constants and the set σ^{act} of *action* constants. Fluent constants are further partitioned into *simple* and *statically determined* fluents. A *fluent formula* is a formula where 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

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

where F and G are fluent formulas. An *action dynamic law* is an expression of the form (2) in which F is an action formula and G is a formula. A *fluent dynamic law* is an expression of the form

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

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

The semantics of $\mathcal{C}$+ in [1] is described via a translation into causal logic. For any action description D and any nonnegative integer m, the causal theory D_m is defined as follows. The signature of D_m consists of the pairs $i:c$ such that

- $i \in \{0,\dots,m\}$ and c is a fluent constant of D, or
- $i \in \{0,\dots,m-1\}$ 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 a formula F, and similarly for a set of formulas. The rules of D_m are

$$i{:}F \Leftarrow i{:}G \tag{4}$$

for every static law (2) in D and every $i \in \{0,\dots,m\}$, and for every action dynamic law (2) in D and every $i \in \{0,\dots,m-1\}$;

$$i{+}1{:}F \Leftarrow (i{+}1{:}G) \wedge (i{:}H) \tag{5}$$

for every fluent dynamic law (3) in D and every $i \in \{0,\dots,m-1\}$;

$$0{:}c{=}v \Leftarrow 0{:}c{=}v \tag{6}$$

for every simple fluent constant c and every $v \in \mathit{Dom}(c)$.

The causal models of D_m correspond to the paths of length m in a transition system — a directed graph whose vertices represent the states and edges are labeled by actions that affect the states. A *state* is an interpretation s of σ^{fl} such that $0:s$ is a model of D_0. States are the vertices of the transition system represented by D. 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 D_1. 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. These labels e are called *events*.

Notation: b, b_1, b_2 range over the blocks in the domain
l ranges over the locations (the blocks and the table)

Simple fluent constant:	Domain:
$Loc(b)$	the set of locations

Action constant:	Domain:
$Move(b, l)$	Boolean

Causal laws:

constraint $\neg(Loc(b_1)=b \wedge Loc(b_2)=b)$ for $b_1 \neq b_2$

$Move(b, l)$ **causes** $Loc(b)=l$
nonexecutable $Move(b, l)$ **if** $Loc(b_1)=b$
nonexecutable $Move(b, b_1) \wedge Move(b_1, l)$

exogenous $Move(b, l)$

inertial $Loc(b)$

Fig. 1. Blocks World in $\mathcal{C}$+

Example 1. Figure 1 shows a description of the Blocks World in $\mathcal{C}$+. The semantics of $\mathcal{C}$+ turns the causal laws in Figure 1 into a causal theory D_m:

$$\begin{array}{rcll}
\bot & \Leftarrow & j:(Loc(b_1)=b \wedge Loc(b_2)=b) & (b_1 \neq b_2) \\
i+1:Loc(b)=l & \Leftarrow & i:Move(b,l)=\text{TRUE} & \\
\bot & \Leftarrow & i:(Move(b,l)=\text{TRUE} \wedge Loc(b_1)=b) & \\
\bot & \Leftarrow & i:(Move(b,b_1)=\text{TRUE} \wedge Move(b_1,l)=\text{TRUE}) & \\
i:Move(b,l)=\text{TRUE} & \Leftarrow & i:Move(b,l)=\text{TRUE} & \\
i:Move(b,l)=\text{FALSE} & \Leftarrow & i:Move(b,l)=\text{FALSE} & \\
i+1:Loc(b)=l & \Leftarrow & i+1:Loc(b)=l \wedge i:Loc(b)=l & \\
0:Loc(b)=l & \Leftarrow & 0:Loc(b)=l &
\end{array} \tag{7}$$

$(0 \leq j \leq m, 0 \leq i \leq m-1)$.

3 Stable Model Semantics

We review two versions of the stable model semantics. One is the stable model semantics for propositional formulas defined by Ferraris [10]. The other is the stable model semantics for multi-valued propositional formulas defined by Bartholomew and Lee [9]. We understand propositional logic programs (multi-valued logic programs, respectively) as an alternative notation of some special syntactic class of propositional formulas (multi-valued propositional formulas, respectively).

3.1 Stable Models of a Propositional Formulas

The following definition is from [10]. For any propositional formula F, the *reduct* F^X of F relative to a set X of atoms is the formula obtained from F by replacing each maximal subformula that is not satisfied by X with $\bot$. We say that X is a *(propositional) stable model* of F if X is a minimal set of atoms satisfying F^X.

By a *propositional* logic program, we denote a set of rules that have the form

$$F \leftarrow G \tag{8}$$

where F and G are propositional formulas that do not contain implications. We identify a logic program with the conjunction of propositional formulas $G \rightarrow F$ for each rule (8) in it.

3.2 Stable Models of a Multi-Valued Propositional Logic Programs

Bartholomew and Lee [9] define stable models of first-order formulas containing intensional functions. There, stable models of a multi-valued propositional formula are understood as a special case of stable models of a first-order formula with intensional functions. We review the stable model semantics of multi-valued propositional formulas by using the notion of a reduct that is similar to the reduct in the previous section.

Let F be a multi-valued propositional formula of signature σ, and let I be a multi-valued propositional interpretation of σ. The reduct F^I of a multi-valued propositional formula F relative to a multi-valued propositional interpretation I is the formula obtained from F by replacing each maximal subformula that is not satisfied by I with $\bot$. I is a *(multi-valued) stable model* of F if I is the unique multi-valued interpretation of σ that satisfies F^I.

By a *multi-valued logic program*, we denote the set of rules that have the form

$$F \leftarrow G \tag{9}$$

where F and G are multi-valued propositional formulas as defined in Section 2.1. We identify a multi-valued logic program with the conjunction of multi-valued propositional formulas $G \rightarrow F$ for each rule (9) in it.

3.3 Turning Multi-Valued Propositional Formulas into Propositional Formulas under the Stable Model Semantics

Note that even when we restrict attention to Boolean constants only, the stable model semantics for multi-valued propositional formulas does not coincide with the stable model semantics for propositional formulas. Syntactically, they are different (one uses an expression of the form $c = \text{TRUE}$, $c = \text{FALSE}$ and the other uses the usual notion of an atom). Semantically, the former relies on the uniqueness of (Boolean)-functions, while the latter relies on the minimization of atoms. Nonetheless there is a simple reduction from the former to the latter.

Begin with a multi-valued propositional signature σ. By σ^p we denote the signature consisting of Boolean constants $c(v)$ for all constants c in $\mathbf{c}$ and all $v \in \mathit{Dom}(c)$.

For any multi-valued propositional formula F of σ, by F_σ we denote the propositional formula that is obtained from F by replacing each occurrence of a multi-valued atom $c{=}v$ with $c(v)$, and adding the formulas

$$\neg(c(v) \wedge c(v')) \tag{10}$$

for all $v, v' \in \mathit{Dom}(c)$ such that $v \neq v'$, and also adding

$$\neg\neg \bigvee_{v \in Dom(c)} c(v). \tag{11}$$

For any interpretation I of σ, by I_σ we denote the interpretation of σ^P that is obtained from I by defining $c(I(c))^I = \text{TRUE}$ iff $c^I = I(c)$.

The following proposition is a special case of Corollary 2 of [9].

Theorem 1 *Let F be a multi-valued propositional formula of a signature σ such that, for every constant c in σ, $\mathit{Dom}(c)$ has at least two elements. (i) An interpretation I of σ is a multi-valued stable model of F iff I_σ is a propositional stable model of F_σ. (ii) An interpretation J of σ^P is a propositional stable model of F_σ iff $J = I_\sigma$ for some multi-valued stable model I of F.*

4 Representing Definite C+ in Multi-Valued Propositional Formulas Under SM

4.1 Turning Definite Causal Theories into Multi-Valued Logic Programs

For any definite causal theory T, by *cl2mvlp*(T) we denote the multi-valued logic program consisting of rules

$$F \leftarrow \neg\neg G$$

for each rule (1) in T. The causal models of such T coincide with the multi-valued stable models of *cl2mvlp*(T).

The following theorem is a special case of Theorem 13 from [9].

Theorem 2 *For any definite causal theory T of a signature σ, a multi-valued interpretation I of σ is a causal model of T iff it is a multi-valued stable model of cl2mvlp(T).*

4.2 Reformulating Definite C+ in Multi-Valued Logic Programs

We consider a finite definite C+ description D of signature σ, where the heads of the rules are either an atom or $\bot$. Without loss of generality, we assume that, for any constant c in σ, $\mathit{Dom}(c)$ has at least two elements. Description D can be turned into a logic program following these steps: (i) turn D into the corresponding multi-valued causal theory D_m (as explained in Section 2.2); (ii) turn D_m into a logic program with multi-valued constants *cl2mvlp*(D_m); (iii) Eliminate multi-valued atoms in favor of propositional atoms. The resulting program can be executed by ASP solvers.

For any definite action description D and any nonnegative integer m, the logic program Π_m is defined as follows. The signature of Π_m consists of the pairs $i : c$ such that

– $i \in \{0,\dots,m\}$ and c is a fluent constant of D, or
– $i \in \{0,\dots,m-1\}$ 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 a formula F, and similarly for a set of formulas. The rules of Π_m are:

$$i{:}F \leftarrow \ \neg\neg\,(i{:}G) \tag{12}$$

for every static law (2) in D and every $i \in \{0,\dots,m\}$, and for every action dynamic law (2) in D and every $i \in \{0,\dots,m-1\}$;

$$i{+}1{:}F \leftarrow \ \neg\neg(i{+}1{:}G) \wedge (i{:}H) \tag{13}$$

for every fluent dynamic law (3) in D and every $i \in \{0,\dots,m-1\}$;

$$0{:}c{=}v \leftarrow \neg\neg\,(0{:}c{=}v) \tag{14}$$

for every simple fluent constant c and every $v \in \mathit{Dom}(c)$.

Example 2. In view of Theorem 2, the causal theory D_m in Example 1 can be represented in multi-valued logic programs as follows.

$$\begin{array}{rcll}
\bot & \leftarrow & \neg\neg(j{:}(\mathit{Loc}(b_1){=}b \wedge \mathit{Loc}(b_2){=}b)) & (b_1 \neq b_2)\\
i{+}1{:}\mathit{Loc}(b){=}l & \leftarrow & i{:}\mathit{Move}(b,l){=}\textsc{true} & \\
\bot & \leftarrow & \neg\neg(i{:}(\mathit{Move}(b,l){=}\textsc{true} \wedge \mathit{Loc}(b_1){=}b)) & \\
\bot & \leftarrow & \neg\neg(i{:}(\mathit{Move}(b,b_1){=}\textsc{true} \wedge \mathit{Move}(b_1,l){=}\textsc{true}) & \\
i{:}\mathit{Move}(b,l){=}\textsc{true} & \leftarrow & \neg\neg(i{:}\mathit{Move}(b,l){=}\textsc{true}) & \\
i{:}\mathit{Move}(b,l){=}\textsc{false} & \leftarrow & \neg\neg(i{:}\mathit{Move}(b,l){=}\textsc{false}) & \\
i{+}1{:}\mathit{Loc}(b){=}l & \leftarrow & \neg\neg(i{+}1{:}\mathit{Loc}(b){=}l) \wedge i{:}\mathit{Loc}(b){=}l & \\
0{:}\mathit{Loc}(b){=}l & \leftarrow & \neg\neg(0{:}\mathit{Loc}(b){=}l) &
\end{array} \tag{15}$$

$(0 \le j \le m; 0 \le i \le m{-}1)$.

According to the theorem on strong equivalence in [9], replacing a rule $\bot \leftarrow \neg\neg F$ with $\bot \leftarrow F$ does not affect the stable models.

Let Π be a multi-valued logic program of signature σ such that the heads of the rules are either an atom or $\bot$. The *dependency graph* of Π, denoted by DG$[\Pi]$, is the directed graph that

– has all multi-valued constants of σ as its vertices, and
– has an edge from c to d if, for some rule $F \leftarrow G$ of Π, c occurs in F and d has a positive occurrence in G that is not in the scope of any negation.

We say that Π is *tight* if the graph DG$[\Pi]$ is acyclic. For example, program (15) is tight. Indeed, it is not difficult to check that *cl2mvlp*(T) for any definite causal theory T is tight.

Any tight multi-valued logic programs can be turned into "completion," similar to Clark's completion [11]. We say that a multi-valued logic program Π is in *Clark normal form* if it is a conjunction of sentences of the form

$$c{=}v \ \leftarrow \ F \tag{16}$$

one for each pair of c and v, and sentences of the form

$$\bot \ \leftarrow \ F \tag{17}$$

The *(functional) completion* of a multi-valued logic program Π is obtained from Π by replacing each conjunctive term (16) in Π with $c{=}v \leftrightarrow F$ and (17) with $\neg F$.

Theorem 3 *Let Π be a multi-valued logic program such that for each multi-valued constant c, $Dom(c)$ has at least two elements. For any multi-valued interpretation I, I is a multi-valued stable model of Π iff I is a model of the completion of Π.*

Example 3. The following theory is the completion of this program. Its stable models are the same as the models of the completion according to Theorem 3.

$$\begin{array}{ll} i{+}1 : \mathit{Loc}(b){=}l \leftrightarrow i{:}\mathit{Move}(b,l){=}\mathrm{TRUE} \vee & (i{+}1 : \mathit{Loc}(b){=}l) \wedge i{:}\mathit{Loc}(b){=}l \\ j{:}(\mathit{Loc}(b_1){=}b \wedge \mathit{Loc}(b_2){=}b) & (b_1 \neq b_2) \\ i{:}(\mathit{Move}(b,l){=}\mathrm{TRUE} \wedge \mathit{Loc}(b_1){=}b) & \\ i{:}(\mathit{Move}(b,b_1){=}\mathrm{TRUE} \wedge \mathit{Move}(b_1,l){=}\mathrm{TRUE}) & \end{array}$$

$(0 \leq j \leq m;\ 0 \leq i \leq m{-}1)$.

The completion can be computed by CSP solvers, as shown in [9].

4.3 Reformulating Definite $\mathcal{C}$+ in Propositional Logic Programs

Multi-valued logic program Π_m in the previous section can be further turned into propositional logic program $(\Pi_m)_\sigma$, as described in Section 3.3. We abbreviate a rule $F \leftarrow \neg\neg F \wedge G$ as $\{F\} \leftarrow G$.

The rules of $(\Pi_m)_\sigma$ are:

$$i{:}F_\sigma \leftarrow \neg\neg\,(i{:}G_\sigma) \tag{18}$$

for every static law (2) in D and every $i \in \{0,\dots,m\}$, and for every action dynamic law (2) in D and every $i \in \{0,\dots,m-1\}$;

$$i{+}1{:}F_\sigma \leftarrow \neg\neg(i{+}1{:}G_\sigma) \wedge (i{:}H_\sigma) \tag{19}$$

for every fluent dynamic law (3) in D and every $i \in \{0,\dots,m-1\}$;

$$\{0{:}c(v)\} \tag{20}$$

for every simple fluent constant c and every $v \in \mathit{Dom}(c)$. Also, we add rules

$$\bot \leftarrow i{:}(c(v) \wedge c(v')) \tag{21}$$

$$\bot \leftarrow i : \Big(\bigwedge_{v \in Dom(c)} \neg c(v)\Big). \tag{22}$$

for all $c \in \mathbf{c}$ and all $v, v' \in \mathit{Dom}(c)$ such that $v \neq v'$.

Example 4. Action description D in Figure 1 is represented by the following propositional logic program:

$$\begin{array}{rl}
\bot \leftarrow j\!:\!(\mathit{Loc}(b_1,b) \wedge \mathit{Loc}(b_2,b)) & (b_1 \neq b_2)\\
i{+}1\!:\!\mathit{Loc}(b,l) \leftarrow i\!:\!\mathit{Move}(b,l,\text{TRUE}) & \\
\bot \leftarrow i\!:\!(\mathit{Move}(b,l,\text{TRUE}) \wedge \mathit{Loc}(b_1,b)) & \\
\bot \leftarrow i\!:\!(\mathit{Move}(b,b_1,\text{TRUE}) \wedge \mathit{Move}(b_1,l,\text{TRUE})) & \\
\{i\!:\!\mathit{Move}(b,l,\text{TRUE})\} & \\
\{i\!:\!\mathit{Move}(b,l,\text{FALSE})\} & \\
\{i{+}1\!:\!\mathit{Loc}(b,l)\} \leftarrow i\!:\!\mathit{Loc}(b,l) & \\
\{0\!:\!\mathit{Loc}(b,l)\} & \\
 & \\
\bot \leftarrow i:(\mathit{Loc}(b,l) \wedge \mathit{Loc}(b,l')) & (l \neq l')\\
\bot \leftarrow i:(\mathit{Move}(b,l,\text{TRUE}) \wedge \mathit{Move}(b,l,\text{FALSE})) & \\
 & \\
\bot \leftarrow i:(\bigwedge_{l \in Locations} \neg \mathit{Loc}(b,l)) & \\
\bot \leftarrow i:(\neg\mathit{Move}(b,l,\text{TRUE}) \wedge \neg\mathit{Move}(b,l,\text{FALSE})) &
\end{array} \tag{23}$$

We can simplify some rules containing Boolean constants. Replace $\mathit{Move}(b,l,\text{TRUE})$ with $\mathit{Move}(b,l)$ and $\mathit{Move}(b,l,\text{FALSE})$ with $\neg\mathit{Move}(b,l)$. We also drop rules that contain $\mathit{Move}(b,l,\text{FALSE})$ from program (23).

$$\begin{array}{rl}
\bot \leftarrow j\!:\!(\mathit{Loc}(b_1,b) \wedge \mathit{Loc}(b_2,b)) & (b_1 \neq b_2)\\
i{+}1\!:\!\mathit{Loc}(b,l) \leftarrow i\!:\!\mathit{Move}(b,l) & \\
\bot \leftarrow i\!:\!(\mathit{Move}(b,l) \wedge \mathit{Loc}(b_1,b)) & \\
\bot \leftarrow i\!:\!(\mathit{Move}(b,b_1) \wedge \mathit{Move}(b_1,l)) & \\
\{i\!:\!\mathit{Move}(b,l)\} & \\
\{i{+}1\!:\!\mathit{Loc}(b,l)\} \leftarrow i\!:\!\mathit{Loc}(b,l) & \\
\{0\!:\!\mathit{Loc}(b,l)\} & \\
 & \\
\bot \leftarrow i:(\mathit{Loc}(b,l) \wedge \mathit{Loc}(b,l')) & (l \neq l')\\
\bot \leftarrow i:(\bigwedge_{l \in Locations} \neg \mathit{Loc}(b,l)) &
\end{array} \tag{24}$$

4.4 Representing Definite $\mathcal{C}$+ in the Language of ASP

The logic program representation of $\mathcal{C}$+ introduced in the previous section can be encoded in the input language of ASP grounders.

We rewrite $i : G$ as $h(G,i)$, where $h(G,i)$ is obtained from $i : G$ by replacing every atomic formula $i : c(v)$ in it by

- $h(c(v),i)$ if c is non-Boolean,
- $h(c,i)$ if c is Boolean and v is TRUE, and
- $\sim h(c,i)$ if c is Boolean and v is FALSE. (' $\sim$' is the symbol for strong negation.)

Each rule (4) is represented by

$$h(F_\sigma,i) \leftarrow \neg\neg h(G_\sigma,i)\ ;$$

Each rule (5) is represented by

$$h(F_\sigma, i{+}1) \leftarrow \neg\neg h(G_\sigma, i) \wedge h(H_\sigma, i)\ ;$$

Each rule (6) is represented by

$$\{h(c(v), 0)\}.$$

Rules (21) and (22) can be succinctly represented by cardinality constraints [12]. If c is nonBoolean, rule (21) can be encoded as

$$\leftarrow 2\{h(c(v), i) : \mathit{Domain}(v)\}$$

(*Domain* is a domain predicate that defines the range of variable v) and rule (22) can be encoded as

$$\bot \leftarrow \{c(v) : \mathit{Domain}(v)\}0.$$

If c is Boolean, rule (22) can be encoded as

$$\bot \leftarrow \{h(c, i), {\sim}h(c, i)\}0.$$

and we do not need to represent rule (10).

Example 5. Program (23) can be encoded in the input language of GRINGO as follows:

```
step(0..maxstep).                    astep(0..maxstep-1) :- maxstep > 0.

#domain step(ST).                    #domain astep(T).
#domain block(B).                    #domain block(B1).
#domain location(L).

% every block is a location
location(B) <- block(B).

% the table is a location
location(table).

% two blocks can't be on the same block at the same time
<- 2{h(loc(BB,B),ST): block(BB)}.

% direct effect
h(loc(B,L),T+1) <- h(move(B,L),T).

% preconditions
<- h(move(B,L),T) & h(loc(B1,B),T).
<- h(move(B,B1),T) & h(move(B1,L),T).

{h(loc(B,L),0)}.
{h(move(B,L),T)}.
{h(loc(B,L),T+1)} <- h(loc(B,L),T).

% existence constraint
```

```
<- {h(loc(B,LL),ST): location(LL)}0.

% uniqueness constraint
<- 2{h(loc(B,LL),ST): location(LL)}.
```

5 Monkey and Bananas in the Language of F2LP

The monkey and bananas domain is the main example used in [1] to illustrate the expressivity of definite $\mathcal{C}$+. The $\mathcal{C}$+ action description *MB* is reproduced in Figure 2. The propositional logic program representation of *MB* may not be directly accepted by an ASP solver as it may contain syntactically complex formulas. For example, the causal rule

$$\textbf{nonexecutable}\ \mathit{PushBox}(l)\ \textbf{if}\ \neg\Big(\bigvee_{l'\in\{L_1,L_2,L_3\}}\Big(\mathit{Loc}(\mathit{Monkey})=l'\wedge\mathit{Loc}(\mathit{Box})=l'\Big)\Big)$$

is turned into [1]

$$\bot\leftarrow i\colon\left(\mathit{PushBox}(l)\wedge\neg\left(\bigwedge_{l'\in\{L_1,L_2,L_3\}}\mathit{Loc}(\mathit{Monkey})=l'\wedge\mathit{Loc}(\mathit{Box})=l'\right)\right).$$

In order to handle this, we use system F2LP [13] ("*f*ormulas *to l*ogic *p*rograms")[2], a front-end that allows ASP solvers to compute stable models of the general programs defined in [14, 15]. Figure 3 is the propositional logic program representation of *MB* in the input language of F2LP. We show how planning, prediction, and postdiction problems can be answered by using the combination of F2LP and CLINGO[3].

Planning

Find the shortest sequence of actions that would allow the monkey to have the bananas.

The problem can be formalized as follows: Find an answer set of $(\mathit{MB}_m)_\sigma$, where σ is the underlying signature, that satisfies the initial conditions

$$0\colon\mathit{Loc}(\mathit{Monkey})=L_1,\ 0\colon\mathit{Loc}(\mathit{Bananas})=L_2,\ 0\colon\mathit{Loc}(\mathit{Box})=L_3 \tag{25}$$

and the goal

$$m\colon\mathit{HasBananas} \tag{26}$$

where m is the smallest number for which such a model exists. To solve this problem, we take consecutively $m=0,1,\ldots$ and look for an answer set of $(\mathit{MB}_m)^{\mathbf{c}}$ that satisfies the constraint in File `planning`. Such an interpretation will be first found for $m=4$.

---

[1] In multi-valued propositional logic, $\mathit{Loc}(\mathit{Monkey})=\mathit{Loc}(\mathit{Box})$ is shorthand for $\bigvee_{l'\in\{L_1,L_2,L_3\}}(\mathit{Loc}(\mathit{Monkey})=l'\wedge\mathit{Loc}(\mathit{Box})=l')$.

[2] `http://reasoning.eas.asu.edu/f2lp`

[3] `http://potassco.sourceforge.net`

Notation: x ranges over $\{Monkey, Bananas, Box\}$; l ranges over $\{L_1, L_2, L_3\}$.

Simple fluent constants:	Domains:
$Loc(x)$	$\{L_1, L_2, L_3\}$
$HasBananas$, $OnBox$	Boolean

Action constants:	Domains:
$Walk(l)$, $PushBox(l)$, $ClimbOn$, $ClimbOff$, $GraspBananas$	Boolean

Causal laws:

caused $Loc(Bananas) = l$ **if** $HasBananas \wedge Loc(Monkey) = l$
caused $Loc(Monkey) = l$ **if** $OnBox \wedge Loc(Box) = l$

$Walk(l)$ **causes** $Loc(Monkey) = l$
nonexecutable $Walk(l)$ **if** $Loc(Monkey) = l$
nonexecutable $Walk(l)$ **if** $OnBox$

$PushBox(l)$ **causes** $Loc(Box) = l$
$PushBox(l)$ **causes** $Loc(Monkey) = l$
nonexecutable $PushBox(l)$ **if** $Loc(Monkey) = l$
nonexecutable $PushBox(l)$ **if** $OnBox$
nonexecutable $PushBox(l)$ **if** $Loc(Monkey) \neq Loc(Box)$

$ClimbOn$ **causes** $OnBox$
nonexecutable $ClimbOn$ **if** $OnBox$
nonexecutable $ClimbOn$ **if** $Loc(Monkey) \neq Loc(Box)$

$ClimbOff$ **causes** $\neg OnBox$
nonexecutable $ClimbOff$ **if** $\neg OnBox$

$GraspBananas$ **causes** $HasBananas$
nonexecutable $GraspBananas$ **if** $HasBananas$
nonexecutable $GraspBananas$ **if** $\neg OnBox$
nonexecutable $GraspBananas$ **if** $Loc(Monkey) \neq Loc(Bananas)$

nonexecutable $Walk(l) \wedge PushBox(l)$
nonexecutable $Walk(l) \wedge ClimbOn$
nonexecutable $PushBox(l) \wedge ClimbOn$
nonexecutable $ClimbOff \wedge GraspBananas$

exogenous c	for every action constant c
inertial c	for every simple fluent constant c

Fig. 2. Action description *MB*

```
% File: mb

step(0..maxstep).
astep(0..maxstep-1) :- maxstep > 0.

#domain step(ST).                          #domain astep(T).
#domain thing(TH).

thing(monkey;bananas;box).

#domain location(L).

location(l1;l2;l3).

% state description
h(loc(bananas,L),ST) <- h(hasBananas,ST) & h(loc(monkey,L),ST).
h(loc(monkey,L),ST) <- h(onBox,ST) & h(loc(box,L),ST).


%% effect and preconditions of actions
h(loc(monkey,L),T+1) <- h(walk(L),T).
<- h(walk(L),T) & h(loc(monkey,L),T).
<- h(walk(L),T) & h(onBox,T).

h(loc(box,L),T+1) <- h(pushBox(L),T).
h(loc(monkey,L),T+1) <- h(pushBox(L),T).
<- h(pushBox(L),T) & h(loc(monkey,L),T).
<- h(pushBox(L),T) & h(onBox,T).
<- h(pushBox(L),T) & - (?[L]: (h(loc(monkey,L),T) & h(loc(box,L),T))).

h(onBox,T+1) <- h(climbOn,T).
<- h(climbOn,T) & h(onBox,T).
<- h(climbOn,T) & - (?[L]: (h(loc(monkey,L),T) & h(loc(box,L),T))).

-h(onBox,T+1) <- h(climbOff,T).
<- h(climbOff,T) & -h(onBox,T).

h(hasBananas,T+1) <- h(graspBananas,T).
<- h(graspBananas,T) & h(hasBananas,T).
<- h(graspBananas,T) & -h(onBox,T).
<- h(graspBananas,T) & - (?[L]: (h(loc(monkey,L),T) & h(loc(bananas,L),T))).

% no concurrency
<- h(walk(L),T) & h(pushBox(L),T).
<- h(walk(L),T) & h(climbOn,T).
<- h(pushBox(L),T) & h(climbOn,T).
<- h(climbOff,T) & h(graspBananas,T).


% fluents are initially exogenous
{h(hasBananas,0), -h(hasBananas,0)}.       {h(onBox,0), -h(onBox,0)}.
{h(loc(TH,L),0)}.

% actions are exogenous
{h(walk(LL),T): location(LL)}.             {h(pushBox(LL),T): location(LL)}.
{h(climbOn,T)}.      {h(climbOff,T)}.      {h(graspBananas,T)}.

% commonsense law of inertia
{h(hasBananas,T+1)} <- h(hasBananas,T).    {-h(hasBananas,T+1)} <- -h(hasBananas,T).
{h(onBox,T+1)} <- h(onBox,T).              {-h(onBox,T+1)} <- -h(onBox,T).
{h(loc(TH,L),T+1)} <- h(loc(TH,L),T).

% Eliminating multi-valued constants
<- {h(onBox,ST), -h(onBox,ST)}0.           <- {h(hasBananas,ST), -h(hasBananas,ST)}0.
<- {h(loc(TH,LL),ST): location(LL)}0.

<- 2{h(loc(TH,LL),ST): location(LL)}.
```

Fig. 3. Action description *MB* in ASP

```
% File: planning

% initial condition
<- not (-h(hasBananas,0) & -h(onBox,0) & h(loc(monkey,l1),0) &
         h(loc(box,l3),0) & h(loc(bananas,l2),0)).

% goal
<- not h(hasBananas,maxstep).
```

The following is the trace of the program. AS2TRANSITION [4] is a utility program that displays answer sets in the format of a transition system.

```
$ f2lp mb planning | clingo -c maxstep=4 | as2transition
Solution 1:

0:  h(loc(bananas,l2),0)  h(loc(box,l3),0)  h(loc(monkey,l1),0)

  ACTIONS:  h(walk(l3),0)

1:  h(loc(bananas,l2),1)  h(loc(box,l3),1)  h(loc(monkey,l3),1)

  ACTIONS:  h(pushBox(l2),1)

2:  h(loc(bananas,l2),2)  h(loc(box,l2),2)  h(loc(monkey,l2),2)

  ACTIONS:  h(climbOn,2)

3:  h(loc(bananas,l2),3)  h(loc(box,l2),3)  h(loc(monkey,l2),3)
h(onBox,3)

  ACTIONS:  h(graspBananas,3)

4:  h(hasBananas,4)  h(loc(bananas,l2),4)  h(loc(box,l2),4)
h(loc(monkey,l2),4)  h(onBox,4)


Models      : 1
Time        : 0.000  (Parsing: 0.000)
```

Prediction

Initially, the monkey is at L_1, the bananas are at L_2, and the box is at L_3. The monkey walks to L_3 and then pushes the box to L_2. Does it follow that in the resulting state the monkey, the bananas and the box are at the same location?

[4] http://reasoning.eas.asu.edu/cplus2asp/downloads.html

This question can be formalized as follows: Determine whether every answer set of MB_2 satisfies the following formula:

$$\begin{aligned}&[(0:Loc(Monkey)=L_1)\wedge(0:Loc(Bananas)=L_2)\wedge(0:Loc(Box)=L_3)\\&\quad\wedge(0:Walk(L_3))\wedge(1:PushBox(L_2))]\\&\qquad\rightarrow 2:(Loc(Monkey)=Loc(Bananas)\wedge Loc(Bananas)=Loc(Box)).\end{aligned}\tag{27}$$

This is equivalent to checking if MB_2 conjoined with the negation of the formula above has no answer sets. The negation of the formula above can be represented in the input language of F2LP as follows: [5]

```
% File: prediction

not
(h(loc(monkey,l1),0) & h(loc(bananas,l2),0) & h(loc(box,l3),0) &
 h(walk(l3),0) &  h(pushBox(l2),1)
 -> ?[L]:(h(loc(monkey,L),2) & h(loc(bananas,L),2) & h(loc(box,L),2))).
```

The following command is used to answer the prediction query.

```
$ f2lp mb prediction | clingo -c maxstep=2 | as2transition
```

CLINGO returns no answer set as expected.

Postdiction

The monkey walked to location L_3 and then pushed the box. Does it follow that the box was initially at L_3?

This question can be formalized as follows: Determine whether MB_2 entails the formula

$$\left[(0:Walk(L_3))\wedge\left(1:\bigvee_l PushBox(l)\right)\right]\rightarrow 0:Loc(Box)=L_3.\tag{28}$$

It can be reduced to the satisfiability problem in the same way as the prediction problem above. The answer to this question is yes. Similarly, the negation of the query can be represented as follows.

```
% File: postdiction

not (h(walk(l3),0) & ?[L]: h(pushBox(L),1) -> h(loc(box,l3),0)).
```

[5] F2LP allows us to represent a formula of the form $\neg F$ where F is an arbitrary formula, including implication (`->`), and quantifiers (`?` for $\exists$, `!` for $\forall$).

6 Discussion

Based on the theoretical result that turns nonmonotonic causal logic into the stable model semantics, we presented a method that represents the definite fragment of $\mathcal{C}$+ in the language of answer set programming.

Our reformulation always yields a tight logic program due to the use of double negations, and in this sense the use of SAT solvers and ASP solvers are not distinguishable. However, it is worthwhile to note that the reformulation in terms of the stable model semantics may provide a way to extend language $\mathcal{C}$+ by allowing recursive definitions. For instance, one may consider extending static causal laws to

$$\mathbf{caused}\ F\ \mathbf{if}\ G\ \mathbf{assuming}\ H\ ,$$

which can be translated into propositional logic program rules

$$i{:}F_\sigma \leftarrow i{:}(G_\sigma \wedge \neg\neg H_\sigma)\ .$$

In the absence of $\mathbf{if}\ G$, this is essentially the translation (18). In the absence of $\mathbf{assuming}\ H$, this is close to the treatment in language $\mathcal{B}$ [16].

References

1. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artificial Intelligence **153(1–2)** (2004) 49–104
2. 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
3. McCain, N.: Causality in Commonsense Reasoning about Actions[6]. PhD thesis, University of Texas at Austin (1997)
4. Lee, J.: Automated Reasoning about Actions[7]. PhD thesis, University of Texas at Austin (2005)
5. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K., eds.: Proceedings of International Logic Programming Conference and Symposium, MIT Press (1988) 1070–1080
6. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385
7. Ferraris, P., Lee, J., Lierler, Y., Lifschitz, V., Yang, F.: Representing first-order causal theories by logic programs. Theory and Practice of Logic Programming (2011) Available on CJO 2011 doi:10.1017/S1471068411000081.
8. Casolary, M., Lee, J.: Representing the language of the causal calculator in answer set programming. In: ICLP (Technical Communications). (2011) 51–61
9. 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) To appear.
10. Ferraris, P.: Answer sets for propositional theories. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). (2005) 119–131

[6] `ftp://ftp.cs.utexas.edu/pub/techreports/tr97-25.ps.gz`

[7] `http://peace.eas.asu.edu/joolee/papers/dissertation.pdf`

11. Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Data Bases. Plenum Press, New York (1978) 293–322
12. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138** (2002) 181–234
13. Lee, J., Palla, R.: System F2LP – computing answer sets of first-order formulas. In: Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). (2009) 515–521
14. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI). (2007) 372–379
15. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artificial Intelligence **175** (2011) 236–263
16. Gelfond, M., Lifschitz, V.: Action languages[8]. Electronic Transactions on Artificial Intelligence **3** (1998) 195–210

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