Stable Models of Formulas with Intensional Functions

Michael Bartholomew and Joohyung Lee
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, AZ, USA
{mjbartho,joolee}@asu.edu

Abstract

In classical logic, nonBoolean fluents, such as the location of an object and the color of a ball, can be naturally described by functions, but this is not the case with the traditional stable model semantics, where the values of functions are pre-defined, and nonmonotonicity of the semantics is related to minimizing the extents of predicates but has nothing to do with functions. We extend the first-order stable model semantics by Ferraris, Lee and Lifschitz to allow intensional functions. The new formalism is closely related to multi-valued nonmonotonic causal logic, logic programs with intensional functions, and other extensions of logic programs with functions, while keeping similar properties as those of the first-order stable model semantics. We show how to eliminate intensional functions in favor of intensional predicates and vice versa, and use these results to encode fragments of the language in the input language of ASP solvers and CSP solvers.

Introduction

NonBoolean fluents, such as the location of an object and the color of a ball, are important in describing the states of the world. In classical logic, nonBoolean fluents can be naturally represented by functions, but this is not the case with the traditional stable model semantics (Gelfond & Lifschitz 1988) and various extensions thereafter, where the values of functions are pre-defined, and the principle of "minimal belief with negation as failure" is related to the minimality condition for predicates in the definition of a stable model, but has nothing to do with functions. Also, most extensions of the stable model semantics are limited to Herbrand models,[1] in which the behavior of functions is governed by the built-in unique name assumption. This is often too strong an assumption for expressing nonBoolean fluents by functions. For instance, $loc(b) = loc(b_1)$ is always false for two different blocks b and b_1, which prevents us from expressing that the two blocks can be on the table.

On the other hand, in nonmonotonic causal logic and its high level notation action language $\mathcal{C}$+ (Giunchiglia *et al.* 2004), nonBoolean fluents are represented by "multi-valued atoms" that have the form $c = v$. In view of (Lifschitz 1997), which provides a first-order semantics of nonmonotonic causal logic, c is essentially a function that is mapped to a value v in the domain. Under the principle of the universal causation, every "causally explained" symbol should be uniquely characterized by the reduct of a program. In other words, nonmonotonicity of this semantics is related to the uniqueness of function values.

Recently, Lifschitz (2011; 2012) introduced yet another nonmonotonic formalism called "logic programs with intensional functions (a.k.a. IF-programs)", which is related to both the stable model semantics and nonmonotonic causal logic. This formalism takes into account the negation as failure as in logic programs, but also can express nonBoolean fluents by functions as in nonmonotonic causal logic. However, the semantics diverges from the stable model semantics in some essential aspects (as we discuss in this paper), and as mentioned in (Lifschitz 2012), it is not obvious how various mathematical results established for the first-order stable model semantics, such as strong equivalence, the splitting theorem and the theorem on completion, can be extended to this formalism.

In this paper, we present an alternative approach to incorporate intensional functions in the stable model semantics by a simple modification to the first-order stable model semantics from (Ferraris, Lee, & Lifschitz 2011). It turns out that in comparison with IF-programs, this formalism is closer to the first-order stable model semantics than to nonmonotonic causal logic. On the other hand, IF-programs are closer to nonmonotonic causal logic.

The paper is organized as follows. In the next section, we introduce the stable model semantics for formulas with intensional functions, in terms of translation into second-order logic formulas, and also in terms of a reduct. Next, we show how to eliminate intensional predicates in favor of intensional functions, and vice versa. We extend several theorems established in the absence of intensional functions, such as the theorem on constraints, the theorem on strong equivalence, the splitting theorem and the completion theorem, to allow intensional functions. We also extend the language RASPL-1 (Lee, Lifschitz, & Palla 2008) to allow intensional functions and how its fragments can be computed by ASP solvers and CSP solvers. We also compare this formalism with several other related nonmonotonic logics. Proofs are

[1] There are exceptions, such as those defined in (Ferraris, Lee, & Lifschitz 2011; Lin & Zhou 2011).

omitted due to the space limit.

Definition and Examples

Stable Model Semantics of Formulas with Intensional Functions

Formulas are built the same as in first-order logic. A signature consists of function constants and predicate constants. Function constants of arity 0 are called object constants. We assume the following set of primitive propositional connectives and quantifiers:

$$\bot \text{ (falsity)},\ \wedge,\ \vee,\ \rightarrow,\ \forall,\ \exists\,.$$

We understand $\neg F$ as an abbreviation of $F \rightarrow \bot$; symbol $\top$ stands for $\bot \rightarrow \bot$, and $F \leftrightarrow G$ stands for $(F \rightarrow G) \wedge (G \rightarrow F)$.

For predicate symbols (constants or variables) u and c, we define $u \leq c$ as $\forall \mathbf{x}(u(\mathbf{x}) \rightarrow c(\mathbf{x}))$. We define $u = c$ as $\forall \mathbf{x}(u(\mathbf{x}) \leftrightarrow c(\mathbf{x}))$ if u and c are predicate symbols, and $\forall \mathbf{x}(u(\mathbf{x}) = c(\mathbf{x}))$ if they are function symbols.

Let $\mathbf{c}$ be a list of distinct predicate and function constants and let $\widehat{\mathbf{c}}$ be a list of distinct predicate and function variables corresponding to $\mathbf{c}$. We call members of $\mathbf{c}$ *intensional* constants. By $\mathbf{c}^{pred}$ we mean the list of the predicate constants in $\mathbf{c}$, and by $\widehat{\mathbf{c}}^{pred}$ the list of the corresponding predicate variables in $\widehat{\mathbf{c}}$. We define $\widehat{\mathbf{c}} < \mathbf{c}$ as

$$(\widehat{\mathbf{c}}^{pred} \leq \mathbf{c}^{pred}) \wedge \neg(\widehat{\mathbf{c}} = \mathbf{c})$$

and $\mathrm{SM}[F;\ \mathbf{c}]$ as

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

where $F^*(\widehat{\mathbf{c}})$ is defined as follows.

- When F is an atomic formula, F^* is $F' \wedge F$, where F' is obtained from F by replacing all intensional (function and predicate) constants in it with the corresponding (function and predicate) variables;[2]
- $(F \wedge G)^* = F^* \wedge G^*$; $\quad (F \vee G)^* = F^* \vee G^*$;
- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;
- $(\forall x F)^* = \forall x F^*$; $\quad (\exists x F)^* = \exists x F^*$.

When F is a sentence, the models of $\mathrm{SM}[F;\ \mathbf{c}]$ are called the $\mathbf{c}$-*stable* models of F. They are the models of F that are "stable" on $\mathbf{c}$.

If $\mathbf{c}$ contains predicate constants only, this definition of a stable model reduces to the one in (Ferraris, Lee, & Lifschitz 2011). The definition of F^* above is the same as in (Ferraris, Lee, & Lifschitz 2011) except for the case when F is an atomic formula.

Example 1 *Let F_1 be $f = 1 \vee f = 2$. $\mathrm{SM}[F_1; f]$ is*

$$F_1 \wedge \neg\exists\widehat{f}(\widehat{f} \neq f \wedge ((\widehat{f} = 1 \wedge f = 1) \vee (\widehat{f} = 2 \wedge f = 2))),$$

which is equivalent to F_1.

[2] If an atomic formula F contains no intensional function constants, then F^* can be defined as F', as in (Ferraris, Lee, & Lifschitz 2011).

Example 2 *Let F_2 be $(f = 1 \vee g = 1) \wedge (f = 2 \vee g = 2)$. $\mathrm{SM}[F_2; fg]$ is*

$$\begin{aligned} F_2 \wedge \neg\exists\widehat{fg}((\widehat{fg} \neq fg) \wedge \\ ((\widehat{f} = 1 \wedge f = 1) \vee (\widehat{g} = 1 \wedge g = 1)) \wedge \\ ((\widehat{f} = 2 \wedge f = 2) \vee (\widehat{g} = 2 \wedge g = 2))), \end{aligned}$$

which is equivalent to $(f = 1 \wedge g = 2) \vee (f = 2 \wedge g = 1)$.

The following lemma is often useful.

Lemma 1 *Formula $\widehat{\mathbf{c}} < \mathbf{c} \rightarrow ((\neg F)^*(\widehat{\mathbf{c}}) \leftrightarrow \neg F)$ is logically valid.*

Example 3 (From (Lifschitz 2012)) *Let F_3 be the formula*

$$\begin{aligned} &\forall x(\neg\neg(f(x) = a) \rightarrow f(x) = a) \\ &\wedge \forall x(p(x) \rightarrow f(x) = b), \end{aligned}$$

where f, a, b are function constants, and p is a predicate constant. Using Lemma 1, $\mathrm{SM}[F_3; f]$ is equivalent to

$$\begin{aligned} F_3 \wedge \neg\exists\widehat{f}\big((\widehat{f} \neq f) \wedge \forall x(f(x) = a \rightarrow \widehat{f}(x) = a) \\ \wedge \forall x(p(x) \rightarrow \widehat{f}(x) = b)\big), \end{aligned}$$

which in turn is equivalent to the first-order formula

$$\forall x(p(x) \rightarrow f(x) = b) \wedge \forall x(\neg p(x) \rightarrow f(x) = a). \qquad (1)$$

Constraints, Choice, Defaults and Strong Equivalence

Constraints

Following Ferraris *et al.* (2009), we say that an occurrence of a constant, or any other subexpression, in a formula F is *positive* if the number of implications containing that occurrence in the antecedent is even, and *negative* otherwise. We say that the occurrence is *strictly positive* if the number of implications in F containing that occurrence in the antecedent is 0. For example, in $\neg(f = 1) \rightarrow g = 1$, the occurrences of f and g are both positive, but only the occurrence of g is strictly positive.

We say that a formula F is *negative* on a list $\mathbf{c}$ of predicate and function constants if members of $\mathbf{c}$ have no strictly positive occurrences in F. We say that F is a *constraint* if it has no strictly positive occurrences of any constant. Clearly, a constraint is negative on any list of constants. For instance, a formula of the form $\neg H$ (shorthand for $H \rightarrow \bot$) is a constraint.

Theorem 1 *For any first-order formulas F and G, if G is negative on $\mathbf{c}$, $\mathrm{SM}[F \wedge G; \mathbf{c}]$ is equivalent to $\mathrm{SM}[F; \mathbf{c}] \wedge G$.*

Example 4 *Consider $\mathrm{SM}[F_2 \wedge \neg(f = 1); fg]$ where F_2 is the formula in Example 2. Since $\neg(f = 1)$ is negative on $\{f, g\}$, according to Theorem 1, $\mathrm{SM}[F_2 \wedge \neg(f = 1); fg]$ is equivalent to $\mathrm{SM}[F_2; fg] \wedge \neg(f = 1)$, which is equivalent to $f = 2 \wedge g = 1$.*

Choice and Defaults

Similar to Theorem 2 from (Ferraris, Lee, & Lifschitz 2011), the theorem below shows that making the set of intensional constants smaller can only make the result of applying SM weaker, and that this can be compensated by

adding "choice formulas." For any predicate constant p, by *Choice*(p) we denote the formula $\forall \mathbf{x}(p(\mathbf{x}) \vee \neg p(\mathbf{x}))$, where $\mathbf{x}$ is a list of distinct object variables. For any function constant f, by *Choice*(f) we denote the formula $\forall \mathbf{x} y((f(\mathbf{x}) = y) \vee \neg(f(\mathbf{x}) = y))$, where y is an object variable that is distinct from $\mathbf{x}$. For any finite list of predicate and function constants $\mathbf{c}$, *Choice*$(\mathbf{c})$ stands for the conjunction of the formulas *Choice*(c) for all members c of $\mathbf{c}$. We sometimes identify a list with the corresponding set when there is no confusion.

Theorem 2 *For any first-order formula F and any disjoint lists $\mathbf{c}$, $\mathbf{d}$ of distinct constants, the following formulas are logically valid:*

$$\text{SM}[F; \mathbf{cd}] \rightarrow \text{SM}[F; \mathbf{c}],$$
$$\text{SM}[F \wedge \mathit{Choice}(\mathbf{d}); \mathbf{cd}] \leftrightarrow \text{SM}[F; \mathbf{c}].$$

For example, $\text{SM}[g = 1 \rightarrow f = 1;\ f]$ is equivalent to $\text{SM}[(g=1 \rightarrow f=1) \wedge \forall y(g=y \vee \neg(g=y));\ fg]$, which in turn is equivalent to $g = 1 \wedge f = 1$.

We abbreviate the formula $F \vee \neg F$ ("the law of excluded middle") as $\{F\}$. Then *Choice*(f) can be written as $\forall \mathbf{x} y\{f(\mathbf{x}) = y\}$; *Choice*$(p)$ can be written as $\forall \mathbf{x}\{p(\mathbf{x})\}$. A formula $\{\mathbf{t} = \mathbf{t}'\}$, where $\mathbf{t}$ contains an intensional function constant and $\mathbf{t}'$ does not, represents that $\mathbf{t}$ takes the value $\mathbf{t}'$ by default. For example, the f-stable models of

$$\{f = 1\},$$

maps f to 1. On the other hand, the default behavior is overridden when we conjoin the formula with $(f = 2)$: the f-stable models of

$$\{f = 1\} \wedge (f = 2)$$

maps f to 2, not to 1.

A default formula is useful to describe the commonsense law of inertia:

$$\mathit{Loc}(b, t)=l \ \rightarrow\ \{\mathit{Loc}(b, t+1)=l\}, \tag{2}$$

where *Loc* is an intensional function constant, represents that the location of a block b at next step keeps its previous value by default. The default behavior can be overridden if some action moves the block.

Strong Equivalence

Strong equivalence (Lifschitz, Pearce, & Valverde 2001) is an important notion that allows us to substitute one subformula for another subformula without affecting the stable models. The theorem on strong equivalence can be extended to formulas with intensional functions as follows.

About first-order formulas F and G we say that F is *strongly equivalent* to G if, for any formula H, any occurrence of F in H, and any list $\mathbf{c}$ of distinct predicate and function constants, $\text{SM}[H; \mathbf{c}]$ is equivalent to $\text{SM}[H'; \mathbf{c}]$, where H' is obtained from H by replacing the occurrence of F by G. In this definition, H is allowed to contain function and predicate constants that do not occur in F, G; Theorem 3 below shows, however, that this is not essential.

Theorem 3 *Let F and G be first-order formulas, let $\mathbf{c}$ be the list of all constants occurring in F or G and let $\widehat{\mathbf{c}}$ be a list of distinct predicate and function variables corresponding to $\mathbf{c}$. The following conditions are equivalent to each other.*

- *F and G are strongly equivalent to each other;*
- *Formula*

$$(F \leftrightarrow G) \wedge (\widehat{\mathbf{c}} < \mathbf{c} \rightarrow (F^*(\widehat{\mathbf{c}}) \leftrightarrow G^*(\widehat{\mathbf{c}})))$$

is logically valid.

According to the theorem, formula $\{F\}$ (shorthand for $F \vee \neg F$) is strongly equivalent to $\neg\neg F \rightarrow F$. This allows us to rewrite formula F_3 in Example 3 as

$$\forall x\{f(x) = a\} \wedge \forall x(p(x) \rightarrow f(x) = b),$$

which represents that by default $f(x)$ is mapped to a, but when x belongs to p, $f(x)$ is mapped to b. This is in agreement with (1). Also, the theorem allows us to rewrite formula (2) as an implication in which the consequent is an atomic formula:

$$\mathit{Loc}(b, t)=l \wedge \neg\neg(\mathit{Loc}(b, t+1)=l)\ \rightarrow\ \mathit{Loc}(b, t+1)=l.$$

Reduct-Based Semantics for Ground Formulas

For a ground formula (i.e., a first-order formula with no variables), the stable models above can be alternatively defined in terms of a reduct, which is similar to the one given in (Ferraris 2005) for propositional formulas.

For two interpretations I, J of the same signature and a list $\mathbf{c}$ of distinct predicate and function constants, we write $J <^{\mathbf{c}} I$ if

- J and I have the same universe and agree on all constants not in $\mathbf{c}$,
- $p^J \subseteq p^I$ for all predicates p in $\mathbf{c}$, and
- J and I do not agree on $\mathbf{c}$.

The reduct F^I of a formula F relative to an interpretation I is the formula obtained from F by replacing every maximal subformula that is not satisfied by I with $\bot$.

Theorem 4 *Let F be a ground formula of signature σ and $\mathbf{c}$ a list of intensional constants. For any interpretation I of σ, $I \models \text{SM}[F; \mathbf{c}]$ iff*

- *I satisfies F, and*
- *every interpretation J of σ such that $J <^{\mathbf{c}} I$ does not satisfy F^I.*

Example 5 *For F_2 in Example 2, recall that an interpretation I that maps f to 1, g to 2, and maps numbers to themselves satisfies $\text{SM}[F_2; fg]$. In accordance with Theorem 4, $(F_2)^I = (f = 1 \wedge g = 2)$, and there is no interpretation J of the same signature such that $J <^{fg} I$ and satisfies $(F_2)^I$.*

Stable Models of Multi-Valued Propositional Formulas

Review of Multi-Valued Propositional Formulas

We first review the definition of a multi-valued propositional formula from (Giunchiglia *et al.* 2004), 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 of 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.

Stable Models of a Multi-Valued Propositional Formula

Multi-valued propositional formulas can be viewed as a special class of ground first-order formulas of many-sorted signatures. We identify a multi-valued propositional signature with a signature in first-order logic by understanding each multi-valued constant c as an intensional object constant, and $Dom(c)$ as a set of non-intensional object constants. A multi-valued propositional atom $c = v$ can be understood as an equality between an intensional object constant c and a non-intensional object constant v.

We identify a multi-valued propositional interpretation with the standard first-order logic interpretation of many-sorted signature in which

- the sort of an intensional object constant c is represented by $Dom(c)$, and
- each non-intensional object constant is mapped to itself, and is identified with an element in $Dom(c)$ for some intensional object constant c.

Example 6 *Consider a multi-valued propositional signature* $\sigma = \{ColorBlue, ColorRed, TapeColor\}$*, where* $Dom(ColorBlue) = Dom(ColorRed) = \{\text{TRUE}, \text{FALSE}\}$ *and* $Dom(TapeColor) = \{Red, Blue, Green\}$*. The following is a multi-valued propositional formula* F*:*

$$\begin{array}{l}(ColorBlue = \text{TRUE} \vee ColorBlue = \text{FALSE}) \\ \wedge\ (ColorRed = \text{TRUE} \vee ColorRed = \text{FALSE}) \\ \wedge\ (ColorBlue = \text{TRUE} \to TapeColor = Blue) \\ \wedge\ (ColorRed = \text{TRUE} \to TapeColor = Red)\end{array}$$

An interpretation I *such that* $I(ColorBlue) = \text{FALSE}$*,* $I(ColorRed) = \text{TRUE}$ *and* $I(TapeColor) = Red$ *satisfies* F*.*

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 $\bot$. The following theorem provides an alternative definition of a stable model in terms of a reduct.

Theorem 5 *Let* F *be a multi-valued propositional formula of signature* σ*, let* I *be a multi-valued propositional interpretation of* σ*.* $I \models \text{SM}[F; \sigma]$ *iff*

- I *satisfies* F*, and*
- *no multi-valued propositional interpretation* J *of* σ *that disagrees with* I *(on* σ*) satisfies* F^I*.*

Example 6 continued *The reduct* F^I *is*

$$\begin{array}{l}(\bot \vee ColorBlue = \text{FALSE}) \wedge (ColorRed = \text{TRUE} \vee \bot) \\ \wedge\ (\bot \to \bot) \wedge (ColorRed = \text{TRUE} \to TapeColor = Red),\end{array}$$

or equivalently

$$\begin{array}{l}(ColorBlue = \text{FALSE}) \wedge\ (ColorRed = \text{TRUE}) \\ \wedge\ (ColorRed = \text{TRUE} \to TapeColor = Red).\end{array}$$

No interpretation J *of the same signature as* I *that differs from* I *on* $TapeColor, ColorBlue, ColorRed$ *satisfies* F^I*. In accordance with Theorem 5,* I *satisfies* $\text{SM}[F;\ TapeColor, ColorBlue, ColorRed]$*.*

The following proposition tells us that any stable model of a multi-valued propositional formula maps a constant only to values that occur in the formula.

Proposition 1 *Let* σ *be a multi-valued propositional signature such that, for each* $c \in \sigma$*,* $Dom(c)$ *has at least two elements. For any multi-valued propositional formula* F *of signature* σ*,*

$$\text{SM}[F; \sigma] \ \to \bigwedge_{c \in \sigma} \ \bigvee_{\substack{v \text{ is a value in } Dom(c) \\ \text{that occurs in } F}} c = v$$

is logically valid.

For example, for the formula F in Example 6,

$$\begin{array}{l}\text{SM}[F;\ TapeColor, ColorBlue, ColorRed] \to \\ \quad (ColorBlue = \text{TRUE} \vee ColorBlue = \text{FALSE}) \\ \quad \wedge\ (ColorRed = \text{TRUE} \vee ColorRed = \text{FALSE}) \\ \quad \wedge\ (TapeColor = Red \vee TapeColor = Blue)\end{array}$$

is logically valid. In view of Proposition 1, an interpretation that maps *TapeColor* to *Green* cannot be a stable model.

Relation to the 1988 Definition of a Stable Model

Let Π be a finite set of rules of the form

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \qquad (3)$$

$(n \geq m \geq 0)$, where each A_i is a propositional atom. The stable models of Π in the sense of (Gelfond & Lifschitz 1988) can be characterized in terms of SM, in the same way as is handled in IF programs (Lifschitz 2012). Lifschitz (2012) defines the *functional image* of Π as follows. First, reclassify all propositional atoms as intensional object constants, and add to the signature two non-intensional object constants 0 and 1. Each rule (3) is rewritten as

$$A_0 = 1 \leftarrow A_1 = 1 \wedge \cdots \wedge A_m = 1 \wedge A_{m+1} \neq 1 \wedge \cdots \wedge A_n \neq 1$$

($A \neq 1$ is shorthand for $\neg(A = 1)$). For each atom A in the signature of Π we add the default rule

$$A = 0 \leftarrow \neg\neg(A = 0)$$

(by default, atoms get the value false). Finally, we add constraints

$$\begin{array}{c} 0 \neq 1, \\ x = 0 \vee x = 1. \end{array} \tag{4}$$

The resulting program is called the *functional image* of Π. Clearly, the models of (4) can be viewed as sets of propositional atoms. The following theorem is similar to Proposition 5 from (Lifschitz 2012), but applies to the stable model semantics presented in this paper.

Theorem 6 *The functional image of* Π *has the same stable models as* Π.

Eliminating Intensional Predicates

The process in the previous section can be extended to eliminate intensional predicates in favor of intensional functions. Given a formula F and an intensional predicate constant p, formula F_f^p is obtained from F as follows:

- in the signature of F, replace p with a new intensional function constant f of arity n, where n is the arity of p, and add two non-intensional object constants 0 and 1;
- replace each subformula $p(\mathbf{t})$ in F with $f(\mathbf{t}) = 1$.

By FC_f ("Functional Constraint on f") we denote the conjunction of the following formulas, which enforces f to behave like predicates:

$$0 \neq 1, \tag{5}$$

$$\neg\neg\forall\mathbf{x}(f(\mathbf{x}) = 0 \vee f(\mathbf{x}) = 1). \tag{6}$$

where $\mathbf{x}$ is a list of distinct object variables. By DF_f ("Default False on f") we denote the following formula:

$$\forall\mathbf{x}(\neg\neg(f(\mathbf{x}) = 0) \rightarrow f(\mathbf{x}) = 0). \tag{7}$$

Example 7 *Let F be the conjunction of the universal closures of the following formulas, which describes the effect of a monkey moving:*

$$\begin{array}{c} \mathit{Loc}(\mathit{Monkey}, 0) = L1, \\ \mathit{Loc}(\mathit{Monkey}, 1) = L2, \\ \mathit{Move}(\mathit{Monkey}, l, t) \rightarrow \mathit{Loc}(\mathit{Monkey}, t+1) = l \end{array}$$

(lower case symbols are variables). We eliminate the intensional predicate Move in favor of an intensional function Move_f to obtain $F^{\mathit{Move}}_{\mathit{Move}_f} \wedge FC_{\mathit{Move}_f} \wedge DF_{\mathit{Move}_f}$, which is the conjunction of the universal closures of the following formulas:

$$\begin{array}{c} \mathit{Loc}(\mathit{Monkey}, 0) = L1, \\ \mathit{Loc}(\mathit{Monkey}, 1) = L2, \\ \mathit{Move}_f(\mathit{Monkey}, l, t) = 1 \rightarrow \mathit{Loc}(\mathit{Monkey}, t+1) = l, \\ 0 \neq 1, \\ \neg\neg\forall xyz(\mathit{Move}_f(x, y, z) = 0 \vee \mathit{Move}_f(x, y, z) = 1) \\ \forall xyz(\neg\neg(\mathit{Move}_f(x, y, z) = 0) \rightarrow \mathit{Move}_f(x, y, z) = 0). \end{array}$$

Theorem 7 *Formulas $\forall\mathbf{x}(f(\mathbf{x}) = 1 \leftrightarrow p(\mathbf{x}))$, FC_f entail* $\mathrm{SM}[F; p\mathbf{c}] \leftrightarrow \mathrm{SM}[F_f^p \wedge DF_f; f\mathbf{c}]$.

The following corollary shows that there is a 1–1 correspondence between the stable models of F and the stable models of its "functional image" $F_f^p \wedge DF_f \wedge FC_f$. For any interpretation I of the signature of F, by I_f^p we denote the interpretation of the signature of F_f^p obtained from I by replacing the set p^I with the function f^I such that

$$\begin{array}{l} f^I(\xi_1, \ldots, \xi_n) = 1 \text{ if } p^I(\xi_1, \ldots, \xi_n) = \text{TRUE} \\ f^I(\xi_1, \ldots, \xi_n) = 0 \text{ otherwise .} \end{array}$$

We also assume that I_f^p satisfies (5). Consequently, I_f^p satisfies FC_f.

Corollary 1 *(a) An interpretation I of the signature of F is a model of* $\mathrm{SM}[F; p\mathbf{c}]$ *iff I_f^p is a model of* $\mathrm{SM}[F_f^p \wedge DF_f; f\mathbf{c}]$. *(b) An interpretation J of the signature of F_f^p is a model of* $\mathrm{SM}[F_f^p \wedge DF_f \wedge FC_f; f\mathbf{c}]$ *iff $J = I_f^p$ for some model I of* $\mathrm{SM}[F; p\mathbf{c}]$.

Eliminating Intensional Functions

We discuss how to eliminate intensional functions in favor of intensional predicates. Unlike the previous section, the result is first established for "f-plain" formulas,[3] and then extended to allow "synonymity" rules.

Let f be a function constant. A first-order formula is called *f-plain* if each atomic formula

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

For a list $\mathbf{f}$ of function constants, we say that F is $\mathbf{f}$-plain if F is f-plain for each member f of $\mathbf{f}$.

Let F be an f-plain formula, where f is an intensional function constant. Formula F_p^f is obtained from F as follows:

- in the signature of F, replace f with a new intensional predicate constant p of arity $n+1$, where n is the arity of f;
- replace each subformula $f(\mathbf{t}) = c$ in F with $p(\mathbf{t}, c)$.

By UEC_p we denote the following formulas that enforce the functional image on the predicates:

$$\begin{array}{c} \forall\mathbf{x}yz(y \neq z \wedge p(\mathbf{x}, y) \wedge p(\mathbf{x}, z) \rightarrow \bot), \\ \neg\neg\forall\mathbf{x}\exists y\, p(\mathbf{x}, y), \end{array} \tag{8}$$

where $\mathbf{x}$ is a n-tuple of variables, and all variables in $\mathbf{x}$, y, and z are pairwise distinct. Note that each formula is a constraint. Clearly, UEC_p is strongly equivalent to

$$\neg\neg\forall\mathbf{x}\exists!y\, p(\mathbf{x}, y) \tag{9}$$

and also classically equivalent to

$$\forall\mathbf{x}\exists!y\, p(\mathbf{x}, y)\ . \tag{10}$$

[3] The notion of f-plain formulas are adapted from f-plain causal theories from (Lifschitz & Yang 2011a).

Example 8 *Consider the same formula F in Example 7. We eliminate the function Loc in favor of an intensional predicate Loc_p to obtain $F^{Loc}_{Loc_p} \wedge UEC_{Loc_p}$, which is the conjunction of the universal closures of the following formulas:*

$$\begin{gathered} Loc_p(Monkey, 0, L1), \\ Loc_p(Monkey, 1, L2), \\ Move(Monkey, l, t) \to Loc_p(Monkey, t+1, l), \\ \forall wxyz(y \neq z \wedge Loc_p(w, x, y) \wedge Loc_p(w, x, z) \to \bot), \\ \neg\neg\forall wx\exists y(Loc_p(w, x, y)). \end{gathered}$$

Theorem 8 *For any f-plain formula F, formulas $\forall \mathbf{x}y(p(\mathbf{x}, y) \leftrightarrow f(\mathbf{x}) = y)$, $\exists xy(x \neq y)$ entail*

$$\text{SM}[F; f\mathbf{c}] \leftrightarrow \text{SM}[F^f_p; p\mathbf{c}].$$

The following corollary shows that there is a simple 1–1 correspondence between the stable models of F and the stable models of $F^f_p \wedge UEC_p$. Recall that the signature of F^f_p is obtained from the signature of F by replacing f with p. For any interpretation I of the signature of F, by I^f_p we denote the interpretation of the signature of F^f_p obtained from I by replacing the function f^I with the set p^I that consists of the tuples

$$\langle \xi_1, \dots, \xi_n, f^I(\xi_1, \dots, \xi_n) \rangle$$

for all $\xi_1, \dots, \xi_n$ from the universe of I.

Corollary 2 *Let F be an f-plain sentence. (a) An interpretation I of the signature of F that satisfies $\exists xy(x \neq y)$ is a model of $\text{SM}[F; f\mathbf{c}]$ iff I^f_p is a model of $\text{SM}[F^f_p;\ p\mathbf{c}]$. (b) An interpretation J of the signature of F^f_p that satisfies $\exists xy(x \neq y)$ is a model of $\text{SM}[F^f_p \wedge UEC_p;\ p\mathbf{c}]$ iff $J = I^f_p$ for some model I of $\text{SM}[F; f\mathbf{c}]$.*

Theorem 8 and Corollary 2 are similar to Theorem 3 and Corollary 5 from (Lifschitz & Yang 2011a), which are about eliminating explainable functions in nonmonotonic causal logic in favor of explainable predicates.

The method above eliminates only one intensional function constant at a time, but repeated applications can eliminate all intensional functions $\mathbf{f}$ from a given $\mathbf{f}$-plain formula. This allows us to represent the $\mathbf{f}$-plain formula by a logic program, which we illustrate in the next section.

We expect that many domains can be described by $\mathbf{f}$-plain formulas, but we know one place where $\mathbf{f}$-plain formulas are limited. This is when we want to express "synonymity" rules (Lee *et al.* 2010; Lifschitz & Yang 2011a) that have the form

$$B \ \to \ f_1(\mathbf{t}_1) = f_2(\mathbf{t}_2), \tag{11}$$

where f_1, f_2 are intensional function constants in $\mathbf{f}$, and $\mathbf{t}_1$, $\mathbf{t}_2$ are tuples of terms not containing members of $\mathbf{f}$. This rule expresses that we believe $f_1(\mathbf{t}_1)$ to be "synonymous" to $f_2(\mathbf{t}_2)$ under condition B. We can eliminate f_1 and f_2 in favor of predicate constants p_1 and p_2 as follows.

We consider a more general case than an $\mathbf{f}$-plain formula. We define a new class of *$\mathbf{f}$-plain-syn formulas* in which every atomic formula

- does not contain any member of $\mathbf{f}$, or
- is of the form $f(\mathbf{t}) = u$ where f is in $\mathbf{f}$, symbol $\mathbf{t}$ is a tuple of terms not containing any member of $\mathbf{f}$, and u is a term not containing any member of $\mathbf{f}$, or
- is of the form $f_1(\mathbf{t}_1) = f_2(\mathbf{t}_2)$ where f_1, f_2 are in $\mathbf{f}$, symbols $\mathbf{t}_1$ and $\mathbf{t}_2$ are tuples of terms not containing any member of $\mathbf{f}$.

Let F be an $\mathbf{f}$-plain-syn formula. The elimination is done by extending the previous method by turning atomic formulas of the form $f_1(\mathbf{t}_1) = f_2(\mathbf{t}_2)$ into

$$\forall y(p_1(\mathbf{t}_1, y) \leftrightarrow p_2(\mathbf{t}_2, y)),$$

where p_1, p_2 are new intensional predicate constants corresponding to f_1, f_2.

$F^{\mathbf{f}}_{\mathbf{p}}$ is defined similar to F^f_p except that it applies to the list of symbols.

Theorem 9 *For any $\mathbf{f}$-plain-syn formula F, the set of formulas $\forall \mathbf{x}y(p(\mathbf{x}, y) \leftrightarrow f(\mathbf{x}) = y)$ for each $f \in \mathbf{f}$ and the corresponding p, and $\exists xy(x \neq y)$ entail*

$$\text{SM}[F; \mathbf{fq}] \leftrightarrow \text{SM}[F^{\mathbf{f}}_{\mathbf{p}}; \mathbf{pq}].$$

Unlike in Theorem 8, the elimination in Theorem 9 applies to the list of intensional functions simultaneously.

RASPL-1 with Intensional Functions

In (Lee, Lifschitz, & Palla 2008), the language RASPL-1 is introduced, which defines the meaning of counting and choice in answer set programming by reducing them to first-order formulas. The restriction in RASPL-1 that does not allow function constants of positive arity is removed in (Bartholomew & Lee 2010), but such functions are restricted to be non-intensional.

Here we extend RASPL-1 to allow intensional functions, and note that this extension provides a convenient way to describe nonBoolean fluents.

As in (Lee, Lifschitz, & Palla 2008), an *aggregate expression* has the form

$$b\,\{\mathbf{x} : F(\mathbf{x})\} \tag{12}$$

($k \geq 1$), where b is a positive integer ("the bound"), $\mathbf{x}$ is a list of variables (possibly empty), and $F(\mathbf{x})$ is a first-order formula. This expression reads: there are at least b values of $\mathbf{x}$ such that $F(\mathbf{x})$.

A *rule* is an expression of the form

$$A_1\ ;\dots;\ A_l \leftarrow\ E_1, \dots, E_m, \mathit{not}\ E_{m+1}, \dots, \mathit{not}\ E_n \tag{13}$$

($l \geq 0$; $n \geq m \geq 0$), where each A_i is an atom, and each E_i is an aggregate expression. A *program* is a list of rules.

The semantics of RASPL-1 is defined by a procedure that turns every aggregate expression, every rule, and every program into a formula of first-order logic, called its *FOL-representation*. The FOL-representation of an aggregate expression $b\,\{\mathbf{x} : F(\mathbf{x})\}$ is the formula

$$\exists \mathbf{x}^1 \cdots \mathbf{x}^b \left[\bigwedge_{1 \leq i \leq b} F(\mathbf{x}^i) \wedge \bigwedge_{1 \leq i < j \leq b} \neg(\mathbf{x}^i = \mathbf{x}^j) \right] \tag{14}$$

where $\mathbf{x}^1, \ldots \mathbf{x}^b$ are lists of new variables of the same length as $\mathbf{x}$. An expression $1\{: A\}$, where A is an atomic formula, can be identified with A. A choice rule of the form

$$\{A\} \leftarrow E_1, \ldots, E_m, \mathit{not}\ E_{m+1}, \ldots, \mathit{not}\ E_n$$

where A is an atomic formula, stands for

$$A;\ \mathit{not}\ A \leftarrow E_1, \ldots, E_m, \mathit{not}\ E_{m+1}, \ldots, \mathit{not}\ E_n,$$

which is strongly equivalent to

$$A \leftarrow E_1, \ldots, E_m, \mathit{not}\ E_{m+1}, \ldots, \mathit{not}\ E_n, \mathit{not\ not}\ A.$$

For instance, the Blocks World domain can be succinctly described in RASPL-1 as follows. First, we say that no two blocks can be on the same block at the same time.

$$\bot \leftarrow 2\{b_1 : \mathit{Loc}(b_1, t) = b\} \tag{15}$$

(b, b_1 are variables for blocks; t is a variable for timepoints). Next we describe the effect of moving a block b to a location l:

$$\mathit{Loc}(b, t+1) = l \leftarrow \mathit{Move}(b, l, t). \tag{16}$$

The preconditions of the action are described by the following constraints:

$$\begin{array}{l} \bot \leftarrow (\mathrm{g}+1)\{b\, l : \mathit{Move}(b, l, t)\}, \\ \bot \leftarrow \mathit{Move}(b, l, t) \wedge \mathit{Loc}(b_1, t) = b, \\ \bot \leftarrow \mathit{Move}(b, b_1, t) \wedge \mathit{Move}(b_1, l, t). \end{array} \tag{17}$$

The first rule of (17) describes that the concurrent move actions are limited by the number of grippers (denoted by the symbol g). The next rule describes that a block can be moved only when it is clear, and the last rule describes that a block cannot be moved onto a block that is being moved also. We describe that the fluents are initially exogenous by

$$\{\mathit{Loc}(b, 0) = l\}, \tag{18}$$

and that actions are exogenous by

$$\{\mathit{Move}(b, l, t)\}. \tag{19}$$

The commonsense law of inertia is expressed by

$$\{\mathit{Loc}(b, t+1) = l\} \leftarrow \mathit{Loc}(b, t) = l. \tag{20}$$

Corollary 2 allows us to represent the theory in the input language of ASP grounder GRINGO as follows, by eliminating function constant *Loc* in favor of predicate constant *On*. Lines 1–13 are sort and variable declarations. Lines 16, 19 are UEC_{On}. The rest is the representation of (15)–(20).

Splitting Theorem and Completion

A *rule* of a first-order formula F is a strictly positive occurrence of an implication in F.

Let F be a $\mathbf{c}$-plain formula. The *dependency graph* of F (relative to $\mathbf{c}$), denoted by $\mathrm{DG}_{\mathbf{c}}[F]$, is the directed graph that

- has all members of $\mathbf{c}$ as its vertices, and
- has an edge from c to d if, for some rule $G \to H$ of F,
  - c has a strictly positive occurrence in H, and
  - d has a positive occurrence in G that does not belong to any subformula of G that is negative on $\mathbf{c}$.

```
1  step(0..maxstep).
2  astep(0..maxstep-1) :- maxstep > 0.
3
4  #domain step(ST).
5  #domain astep(T).
6  #domain block(B;B1).
7  #domain location(L;L1).
8
9  % every block is a location
10 location(B) :- block(B).
11
12 % the table is a location
13 location(table).
14
15 % uniqueness constraint
16 :- 2{on(B,LL,ST): location(LL)}.
17
18 % existence constraint
19 :- {on(B,LL,ST): location(LL)}0.
20
21 :- 2{on(BB,B,ST): block(BB)}.
22
23 on(B,L,T+1) :- move(B,L,T).
24
25 % preconditions
26 :- (g+1) {move(BB,LL,T): block(BB): location(LL)}.
27 :- move(B,L,T), on(B1,B,T).
28 :- move(B,B1,T), move(B1,L,T).
29
30 {on(B,L,0)}.
31
32 {move(BB,LL,T): block(BB): location(LL)}.
33
34 {on(B,L,T+1)} :- on(B,L,T).
```

Figure 1: Blocks World Encoding in ASP

A *loop* of F (relative to a list $\mathbf{c}$ of intensional constants) is a nonempty subset $\mathbf{l}$ of $\mathbf{c}$ such that the subgraph of $\mathrm{DG}_{\mathbf{c}}[F]$ induced by $\mathbf{l}$ is strongly connected.

The following theorem extends the splitting lemma from (Ferraris *et al.* 2009) to allow intensional functions.

Theorem 10 *Let $\mathbf{c}$ be a list of constants, and let F be a $\mathbf{c}$-plain formula. If $\mathbf{l}^1, \ldots, \mathbf{l}^n$ are all the loops of F relative to $\mathbf{c}$ then*

$$\mathrm{SM}[F; \mathbf{c}] \quad \textit{is equivalent to} \quad \mathrm{SM}[F; \mathbf{l}^1] \wedge \cdots \wedge \mathrm{SM}[F; \mathbf{l}^n].$$

The following theorem extends the splitting theorem from (Ferraris *et al.* 2009) to allow intensional functions.

Theorem 11 *Let $\mathbf{c}$, $\mathbf{d}$ be finite disjoint lists of distinct constants, and let F, G be $(\mathbf{c} \cup \mathbf{d})$-plain sentences. If*

(a) each strongly connected component of the dependency graph of $F \wedge G$ relative to $\mathbf{c}$, $\mathbf{d}$ is either a subset of $\mathbf{c}$ or a subset of $\mathbf{d}$,

(b) F is negative on $\mathbf{d}$, and

(c) G is negative on $\mathbf{c}$

then

$$\mathrm{SM}[F \wedge G;\ \mathbf{c} \cup \mathbf{d}] \leftrightarrow \mathrm{SM}[F;\ \mathbf{c}] \wedge \mathrm{SM}[G;\ \mathbf{d}]$$

is logically valid.

It is clear that Theorem 1 is a special case of Theorem 11, when $\mathbf{d}$ is empty.

As shown in (Lee, Lifschitz, & Palla 2008), the splitting theorem is useful in reasoning about ASP programs. It is also useful in proving other theorems, such as the theorem on completion below. We say that a formula F is in *Clark normal form* (relative to the list $\mathbf{c}$ of intensional constants) if it is a conjunction of sentences of the form

$$\forall \mathbf{x}(G \rightarrow p(\mathbf{x})) \tag{21}$$

and

$$\forall \mathbf{x} y(G \rightarrow f(\mathbf{x}) = y) \tag{22}$$

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

The *completion* of a formula F in Clark normal form (relative to $\mathbf{c}$) is obtained from F by replacing each conjunctive term (21) with

$$\forall \mathbf{x}(p(\mathbf{x}) \leftrightarrow G) \tag{23}$$

and each conjunctive term (22) with

$$\forall \mathbf{x} y(f(\mathbf{x}) = y \leftrightarrow G). \tag{24}$$

We say that F is *tight* (on $\mathbf{c}$) if the dependency graph of F (relative to $\mathbf{c}$) is acyclic. The following theorem tells us that, for a tight theory, completion is a process that allows us to reclassify intensional constants as non-intensional ones. It is similar to the main theorem of (Lifschitz & Yang 2011b), which describes functional completion in nonmonotonic causal logic.

Theorem 12 *For any formula F in Clark normal form that is tight on $\mathbf{c}$, an interpretation I that satisfies $\exists xy(x \neq y)$ is a model of* SM$[F; \mathbf{c}]$ *iff I is a model of the completion of F relative to $\mathbf{c}$.*

For example, the conjunction of the universal closures of (16), (18), (19), (20) can be turned into Clark normal form, and is equivalent to the completion relative to $\{Loc, Move\}$:

$$\begin{aligned} &Loc(b, t_1) = l \\ &\quad \leftrightarrow (Move(b, l, t) \wedge (t_1 = t + 1)) \\ &\quad \vee (\neg\neg(Loc(b, 0) = l) \wedge t_1 = 0) \\ &\quad \vee (\neg\neg(Loc(b, t + 1) = l) \wedge Loc(b, t) = l \wedge (t_1 = t + 1)), \end{aligned}$$

$$Move(b, l, t) \leftrightarrow \neg\neg Move(b, l, t).$$

The assumption $\exists xy(x \neq y)$ in the statement of Theorem 12 is essential. For instance, take F to be $\top$ and $\mathbf{c}$ to be an intensional function constant f. If the universe $|I|$ of an interpretation I is a singleton, then I satisfies SM$[F]$, but does not satisfy the completion formula $\forall \mathbf{x} y(f(\mathbf{x}) = y \leftrightarrow \bot)$.

In view of Theorem 1, the RASPL-1 program consisting of (15)–(20) is equivalent to the conjunction of the completion above and the FOL-representation of the constraints (15) and (17). The resulting theory can be represented in the language of CSP solvers. Below we show an encoding in the language of EZCSP (Balduccini 2009), a system that allows us to use an ASP grounder to generate CSP instances. We take the advantage of EZCSP with grounding, but do not rely on any nonmonotonic feature of the input language. In fact, the completion can be expressed in the input language of other CSP solvers.

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

#domain step(ST).
#domain astep(T).
#domain block(B;B1;B2).
#domain location(L;L2;L2).

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

% the table (denoted by 0) is a location
location(0).

% constraint variable declarations
cspvar(loc(B,ST),0,blocks).
cspvar(move(B,L,T),0,1).

% no two blocks can be on a single block
required((loc(B,T)=B2 /\ loc(B1,T)=B2 /\ B!=B1)
             -> 1=2).

% preconditions
required((move(B,L,T)=1 /\ move(B1,L1,T)=1 /\
  move(B2,L2,T)=1 /\ (B!=B1 \/ L!=L1) /\
   (B!=B2 \/ L!=L2) /\ (B1!=B2 \/ L1!=L2)) -> 1=2).
required((move(B,L,T)=1 /\ loc(B1,T)=B) -> 1=2).
required((move(B,B1,T)=1 /\ move(B1,L,T)=1) -> 1=2).

% completion
required(((move(B,L,T)=1 /\ ST=T+1)
        \/ (loc(B,ST)=L /\ ST=0)
        \/ (loc(B,T)=L /\ loc(B,ST)=L /\ ST=T+1))
            -> loc(B,ST)=L).
required((loc(B,ST)=L /\ ST=T+1)
        -> ((move(B,L,T)=1)
        \/ (loc(B,T)=L /\ loc(B,ST)=L /\ ST=T+1))).
```

Figure 2: Blocks World Encoding in EZCSP

Our preliminary experiments indicates that for the blocks world encoding in Figures 1 and 2, CLINGO runs significantly faster than EZCSP using B-PROLOG, but a more systematic comparison has to be conducted.

Relation to Nonmonotonic Causal Logic

Review: Nonmonotonic Causal Logic

A *(nonmonotonic) causal theory* is a finite list of rules of the form

$$F \Leftarrow G$$

where F and G are formulas. We identify a rule with the universal closure of the implication $G \rightarrow F$. A *causal model* of a causal theory T is defined as the models of the second-order sentence

$$\text{CM}[T; \mathbf{f}] = T \wedge \neg\exists \widehat{\mathbf{f}}(\widehat{\mathbf{f}} \neq \mathbf{f} \wedge T^{\dagger}(\widehat{\mathbf{f}}))$$

where $\mathbf{f}$ is a list of *explainable* function constants, and $T^\dagger(\widehat{\mathbf{f}})$ denotes the conjunction of the formulas

$$\widetilde{\forall}(G \to F(\widehat{\mathbf{f}})) \tag{25}$$

for all rules $F \Leftarrow G$ of T.

By a *definite* casual theory, we mean the causal theory whose rules have the form either

$$f(\mathbf{t}) = t_1 \Leftarrow B \tag{26}$$

or

$$\bot \Leftarrow B, \tag{27}$$

where f is an explainable function constant, $\mathbf{t}$ is a list of terms that does not contain explainable function constants, and t_1 is a term that does not contain explainable function constants. By $Tr(T)$ we denote the theory consisting of conjunction of the following formulas: $\widetilde{\forall}(\neg\neg B \to f(\mathbf{t}) = t_1)$ for each rule (26) in T, and $\widetilde{\forall}\neg B$ for each rule (27) in T. The causal models of such T coincide with the stable models of $Tr(T)$.

Theorem 13 *For any definite causal theory T, $I \models \mathrm{CM}[T; \mathbf{f}]$ iff $I \models \mathrm{SM}[Tr(T); \mathbf{f}]$.*

Relation to IF-Programs

Review of IF-Programs

We consider rules of the form

$$H \leftarrow B, \tag{28}$$

where H and B are formulas that do not contain $\to$. As before, we identify a rule with the universal closure of the implication $B \to H$. An IF-program is a finite list of those rules.

An occurrence of a symbol in a formula is *negated* if it belongs to a subformula of that begins with negation, and is *non-negated* otherwise. Let F be a formula, let $\mathbf{f}$ be a list of distinct function constants, and let $\widehat{\mathbf{f}}$ be a list of distinct function variables corresponding to $\mathbf{f}$. By $F^\diamond(\widehat{\mathbf{f}})$ we denote the formula obtained from F by replacing each non-negated occurrence of a member of $\mathbf{f}$ with the corresponding function variable in $\widehat{\mathbf{f}}$. By $\mathrm{IF}[F; \mathbf{f}]$ we denote the second-order sentence

$$F \wedge \neg\exists\widehat{\mathbf{f}}(\widehat{\mathbf{f}} \neq \mathbf{f} \wedge F^\diamond(\widehat{\mathbf{f}})).$$

According to (Lifschitz 2012), the $\mathbf{f}$-*stable models* of an IF-program Π are defined as the models of $\mathrm{IF}[F; \mathbf{f}]$, where F is the FOL-representation of Π.

Reduct-based Semantics of IF-Programs

For any ground formula F, $F^{\underline{I}}$ is a formula obtained from F by replacing every negated formula that is not satisfied by I with $\bot$.

Let Π be a ground IF-program. The IF-reduct $\Pi^{\underline{I}}$ of an IF-program Π relative to an interpretation I consists of rules

$$H^{\underline{I}} \leftarrow B^{\underline{I}}$$

for every rule $H \leftarrow B$ in Π.

Theorem 14 *Let F be the FOL-representation of a ground IF-program of signature σ and let $\mathbf{f}$ be a list of intensional function constants. For any interpretation I of σ, $I \models \mathrm{IF}[F; \mathbf{f}]$ iff*

- *I satisfies Π, and*
- *no interpretation J of σ that disagrees with I only on $\mathbf{f}$ satisfies $\Pi^{\underline{I}}$.*

Comparison

The definition of the IF operator above looks close to our definition of the SM operator. However, they often behave quite differently. Neither semantics is stronger than the other.

Example 9 *Let F be the following program*

$$d = 2 \leftarrow c = 1,$$
$$d = 1$$

and let I be an interpretation such that $|I| = \{1, 2\}$, $I(c) = 2$ and $I(d) = 1$. I is a model of $\mathrm{IF}[F; cd]$, but not a model of $\mathrm{SM}[F; cd]$.

Example 10 *Let F be the following program*

$$(c = 1 \vee d = 1) \wedge (c = 2 \vee d = 2)$$

and let I_1 and I_2 be interpretations such that $|I_1| = |I_2| = \{1, 2, 3\}$ and $I_1(c) = 1$, $I_1(d) = 2$, $I_2(c) = 2$, $I_2(d) = 1$. I_1 and I_2 are models of $\mathrm{SM}[F; cd]$. On the other hand, $\mathrm{IF}[F; cd]$ has no models.

Example 11 *Let F be $c \neq 1 \leftarrow \top$ and let F_1 be $\bot \leftarrow c = 1$. Under our semantics, they are strongly equivalent to each other, and neither of them has a stable model. However, this is not the case with IF-programs. For instance, let I be an interpretation such that $|I| = \{1, 2\}$ and $I(c) = 2$. I satisfies $\mathrm{IF}[F_1; c]$ but not $\mathrm{IF}[F; c]$.*

While $\bot \leftarrow F$ is a constraint in our formalism, in view of Theorem 1, the last example illustrates that $\bot \leftarrow F$ is not considered as a constraint in the semantics of IF-programs. Indeed, the definition of a constraint given in (Lifschitz 2012) is stronger than ours.

Also, the statement of Proposition 1 does not apply to IF-programs. Example 11 illustrates that a model of an IF-program may map a function to a value that does not even occur in the program.

Let T be an IF-program whose rules have the form

$$f(\mathbf{t}) = t_1 \leftarrow \neg\neg B \tag{29}$$

where f is an intensional function constant, $\mathbf{t}$ and t_1 do not contain intensional function constants, and B is an arbitrary formula. We identify T with the corresponding first-order formula.

Theorem 15 *$I \models \mathrm{SM}[T; \mathbf{f}]$ iff $I \models \mathrm{IF}[T; \mathbf{f}]$.*

Relation to Cabalar's Functional Logic Programs

Cabalar defines functional answer set programs based on an extension of HT-models that allows "partial" functions. The presence of partial functions is essential in Cabalar's semantics to enforce a strict order between the two worlds, H and T. When all constants other than f are evaluated the same for both worlds, H is "strictly smaller" than T when function f is undefined in H and defined in T. In the absence of partial functions, H and T evaluate the same for all functions.

If we require every function to be total, it is easy to check that the semantics from (Cabalar 2011) is equivalent to the first-order stable model semantics from (Ferraris, Lee, & Lifschitz 2007), or, put in another way, results in the special case of our semantics in which every function constant is non-intensional.

As described by Lifschitz (2012), the essential difference between IF-programs and Cabalar's functional answer set programs is that the latter relies on partial functions and minimization, instead of total functions and uniqueness as in the former. This remark also applies to the relationship between our formalism and Cabalar's.

Relation to Lin and Wang's Logic Programs with Functions

Lin & Wang (2008) extended answer set semantics with functions by extending the definition of a reduct, and also provided loop formulas for such programs. We can provide an alternative account of their results by considering the notions there as special cases of the definitions presented in this paper. For simplicity, we assume non-sorted languages.[4] Essentially, they restricted attention to a special case of non-Herbrand interpretations such that object constants form the universe, and ground terms other than object constants are mapped to object constants. According to (Lin & Wang 2008), an *LW-program* P consists of *type definitions* and a set of rules. Type definitions introduce the domains for a many-sorted signature consisting of some object constants, and includes the evaluation of each function symbol of positive arity that maps a list of object constants to an object constant. Since we assume here non-sorted languages, we consider only a single domain (universe). We say that an interpretation I is a *P-interpretation* if the universe is the set of object constants specified by P, object constants are evaluated to itself, and ground terms other than object constants are evaluated conforming to the type definitions of P.

Theorem 16 *Let P be an LW-program and let F be the FOL-representation of the set of rules in P. The following conditions are equivalent to each other:*

(a) I is an answer set of P in the sense of (Lin & Wang 2008);

(b) I is a P-interpretation that satisfies $\mathrm{SM}[F;\mathbf{p}]$ *where* $\mathbf{p}$ *is the list of all predicate constants occurring in F.*

Thus the definition does not allow functions to be intensional.

[4] (Lin & Wang 2008) considers essentially many-sorted languages. The result of this section can be extended to that case by considering many-sorted SM (Kim, Lee, & Palla 2009).

Conclusion

The absence of intensional functions in several extensions of the stable model semantics prevents us from expressing the commonsense law of inertia involving nonBoolean fluents, such as formula (2). IF-programs introduced by Lifschitz is distinct in the sense that functions are allowed to be intensional. Here we presented an alternative stable model semantics that allows intensional functions. Unlike IF-programs, where the syntax is limited and only function constants are intensional, our semantics is applied to arbitrary first-order formulas and allows both function and predicate constants to be intensional. A major difference between IF-programs and our formalism is that the former is closer to nonmonotonic causal logic, and the latter is closer to the first-order stable model semantics. Indeed, we observed that several properties established for first-order stable model semantics can be easily extended to the formalism presented here.

RASPL-1 with intensional functions is shorthand for formulas with intensional functions, and allows succinct representation of domains that involve nonBoolean constants. The computation can be done by either turning it into answer set programs or into CSP.

Acknowledgements

We are grateful to Vladimir Lifschitz and Yunsong Meng for many useful discussions and comments. We are also grateful to the anonymous referees for their comments and suggestions. This work was partially supported by the National Science Foundation under Grant IIS-0916116.

References

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

Bartholomew, M., and Lee, J. 2010. A decidable class of groundable formulas in the general theory of stable models. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 477–485.

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

Ferraris, P.; Lee, J.; Lifschitz, V.; and Palla, R. 2009. Symmetric splitting in the general theory of stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 797–803.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2007. A new perspective on stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 372–379.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2011. Stable models and circumscription. *Artificial Intelligence* 175:236–263.

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

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R., and

Bowen, K., eds., *Proceedings of International Logic Programming Conference and Symposium*, 1070–1080. MIT Press.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153(1–2):49–104.

Kim, T.-W.; Lee, J.; and Palla, R. 2009. Circumscriptive event calculus as answer set programming. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 823–829.

Lee, J.; Lierler, Y.; Lifschitz, V.; and Yang, F. 2010. Representing synonymity in causal logic and in logic programming[5]. In *Proceedings of International Workshop on Nonmonotonic Reasoning (NMR)*.

Lee, J.; Lifschitz, V.; and Palla, R. 2008. A reductive semantics for counting and choice in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 472–479.

Lifschitz, V., and Yang, F. 2011a. Eliminating function symbols from a nonmonotonic causal theory. In Lakemeyer, G., and McIlraith, S. A., eds., *Knowing, Reasoning, and Acting: Essays in Honour of Hector J. Levesque*. College Publications.

Lifschitz, V., and Yang, F. 2011b. Functional completion[6]. Unpublished Draft.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic* 2:526–541.

Lifschitz, V. 1997. On the logic of causal explanation. *Artificial Intelligence* 96:451–465.

Lifschitz, V. 2011. Logic programs with intensional functions (preliminary report). In *Working Notes of the Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*.

Lifschitz, V. 2012. Logic programs with intensional functions. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*. This volume.

Lin, F., and Wang, Y. 2008. Answer set programming with functions. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 454–465.

Lin, F., and Zhou, Y. 2011. From answer set logic programming to circumscription via logic of GK. *Artificial Intelligence* 175:264–277.

[5]http://peace.eas.asu.edu/joolee/papers/syn.pdf

[6]http://www.cs.utexas.edu/users/vl/papers/if.pdf