Logic Programs with Intensional Functions

Vladimir Lifschitz
Department of Computer Science, University of Texas
Austin, TX 78712, USA

Abstract

The stable model semantics treats a logic program as a mechanism for specifying its intensional predicates. In this paper we discuss a modification of that semantics in which functions, rather than predicates, are intensional. The idea of the new definition comes from nonmonotonic causal logic.

Introduction

The definition of a stable model proposed in (Ferraris, Lee, & Lifschitz 2011) treats a logic program as a mechanism for specifying its "intensional predicates." The model-theoretic meaning of that definition can be described in terms of a first-order version of equilibrium logic (Pearce 1997). Equilibrium models of a formula are defined as the Kripke models with two worlds that satisfy a certain minimality condition.

In this note we discuss a modification of the definition from (Ferraris, Lee, & Lifschitz 2011) in which functions, rather than predicates, are intensional. The difficulty here is that it is not clear how to apply the idea of minimality to functions. Predicates can be viewed as sets, and the subset relation can be used to compare them. Functions can be viewed as sets also—as sets of ordered pairs. But a function from A to B cannot be a subset of any other function from A to B; minimality becomes trivial. The solution adopted in the semantics of functional logic programs (Cabalar 2011) is based on making functions partial. We discuss here another approach.

The idea of the semantics of logic programs with intensional functions defined below comes from nonmonotonic causal logic (McCain & Turner 1997), extended to the first-order case in (Lifschitz 1997). We discard the minimality condition in the definition of equilibrium logic altogether and replace it by a uniqueness condition. The result is a language that bears strong resemblance to causal logic; in fact, many programs in this language can be easily reformulated as causal theories. At the same time, the new language is closely related to the traditional stable model semantics of logic programs: the latter can be embedded into it by treating predicates as Boolean-valued functions and by adding standard "minimization rules."

Because programs with intensional functions are similar both to nonmonotonic causal theories and to traditional logic programs, they provide a new perspective on the relationship between these two knowledge representation languages. This is one of the reasons why they may be of interest. Another reason is that they allow us to describe effects of actions on non-Boolean fluents directly, in pretty much the same way as causal theories in the sense of (Lifschitz 1997). In traditional logic programming, non-Boolean fluents have to be encoded by Boolean fluents; to express, for instance, that the location of an object x changed between times t and $t+1$ we have to write

$$at(x, y, t) \wedge at(x, z, t+1) \wedge y \neq z$$

instead of simply $loc(x, t+1) \neq loc(x, t)$. Expressing the commonsense law of inertia (Shanahan 1997) for locations requires two rules if *loc* is replaced with *at*: a positive inertia rule and a uniqueness rule (see, for instance, the description of the blocks world in (Lifschitz 2002, Section 5.1)). The new language is more concise.

Syntax

In (Ferraris, Lee, & Lifschitz 2011), stable models are defined for arbitrary first-order sentences. As observed in (Lifschitz 2011), that definition can be simplified in the important special case when the formula is a conjunction of "rules." The definition of a program with intensional functions in this paper is restricted to formulas of this form also.[1]

In first-order formulas, we take the connectives and quantifiers

$$\top \ \bot \ \neg \ \wedge \ \vee \ \rightarrow \ \forall \ \exists$$

as primitives. A first-order sentence is a *rule* if it has the form

$$\widetilde{\forall}(B \rightarrow H) \tag{1}$$

[1] The paper (Bartholomew & Lee 2012) describes an alternative semantics of intensional functions that is applicable to arbitrary first-order sentences.

and has no occurrences of $\rightarrow$ other than the one explicitly shown.[2] Formula B is the *body* of rule (1), and H is its *head*. A *logic program with intensional functions*, or *IF-program* for short, is a pair $(R, \mathbf{f})$, where R is a conjunction of rules, and $\mathbf{f}$ is a tuple of distinct function constants.[3] We will represent R by the list of its conjunctive terms (1) written as $H \leftarrow B$, and we will drop $\leftarrow B$ if B is $\top$. The members of $\mathbf{f}$ will be called the *intensional functions* of the program.

Consider, for instance, the IF-program with the rules

$$\begin{array}{ll} f(x) = a & \leftarrow \neg(f(x) \neq a), \\ f(x) = b & \leftarrow P(x) \end{array} \tag{2}$$

and the intensional function f. (Since the formula $f(x) \neq a$ is shorthand for $\neg(f(x) = a)$, the body of the first rule is the double negation of its head.) As discussed below, the first rule expresses that, by default, the values of f are equal to a. The second rule expresses that on every argument from P the value of f is b.

Semantics

We will define the semantics of IF-programs by specifying which models of R are considered "stable models" of $(R, \mathbf{f})$. Like the definitions of a stable model introduced in (Ferraris, Lee, & Lifschitz 2011) and (Lifschitz 2011), the new definition is based on a syntactic transformation that turns logic programs into second-order sentences.[4] An occurrence of a symbol in a formula F is *negated* if it belongs to a subformula of F that begins with negation, and *nonnegated* otherwise. Let F be a formula, and let $\mathbf{f}$ be a tuple of distinct function constants. For each member f of $\mathbf{f}$, choose a new function variable υf of the same arity as f, and let $\upsilon\mathbf{f}$ be the list of all these function variables. By $F^{\diamond}(\upsilon\mathbf{f})$ we denote the formula obtained from F by replacing each nonnegated occurrence of each member f with the variable υf. By $\mathrm{SM}_{\mathbf{f}}[F]$ we denote the sentence

$$\forall \upsilon\mathbf{f}(F^{\diamond}(\upsilon\mathbf{f}) \leftrightarrow \upsilon\mathbf{f} = \mathbf{f}). \tag{3}$$

(Here $\upsilon\mathbf{f} = \mathbf{f}$ stands for the conjunction of the equalities $\upsilon f = f$ for all members f of the list $\mathbf{f}$ and the corresponding members υf of the list $\upsilon\mathbf{f}$.) Formula (3) expresses that $\mathbf{f}$ is the only tuple of functions satisfying the condition $F^{\diamond}(\upsilon\mathbf{f})$.

A *stable model* of an IF-program $(R, \mathbf{f})$ is a model of $\mathrm{SM}_{\mathbf{f}}[R]$ in the sense of classical second-order logic.

For example, the stable models of (2) are models of the formula

$$\begin{array}{l} \forall \upsilon f((\forall x(\neg(f(x) \neq a) \rightarrow \upsilon f(x) = a) \\ \qquad \wedge \forall x(P(x) \rightarrow \upsilon f(x) = b)) \\ \qquad\qquad\qquad\qquad \leftrightarrow \upsilon f = f). \end{array} \tag{4}$$

The second-order quantifier here can be eliminated:

Proposition 1 *Formula (4) is equivalent to the first-order sentence*

$$\forall x(P(x) \rightarrow f(x) = b) \wedge \forall x(\neg P(x) \rightarrow f(x) = a). \tag{5}$$

Formula (5) expresses that on every argument from P the value of f is b, and on every other argument the value of f is a. This is consistent with understanding the first rule of (2) as a default, and the second rule as an exception.

Proofs of propositions can be found in the appendix.

Expression (3) for $\mathrm{SM}_{\mathbf{f}}[F]$ can be equivalently rewritten as

$$F \wedge \forall \upsilon\mathbf{f}(F^{\diamond}(\upsilon\mathbf{f}) \rightarrow \upsilon\mathbf{f} = \mathbf{f}).$$

Consequently the stable models of $(R, \mathbf{f})$ can be characterized as the models of R that satisfy the "stability condition"

$$\forall \upsilon\mathbf{f}(R^{\diamond}(\upsilon\mathbf{f}) \rightarrow \upsilon\mathbf{f} = \mathbf{f}). \tag{6}$$

Comparison with Similar Definitions

To see the similarity between the definition above and the semantics introduced in (Lifschitz 2011), recall that a *Datalog program* in that paper is a pair $(R, \mathbf{p})$, where R is a conjunction of rules, and $\mathbf{p}$ is a tuple of distinct predicate constants. The result of applying the operator $\mathrm{SM}_{\mathbf{p}}$ to F is defined there as

$$F \wedge \neg\exists \upsilon\mathbf{p}((\upsilon\mathbf{p} < \mathbf{p}) \wedge F^{\diamond}(\upsilon\mathbf{p})), \tag{7}$$

where $\upsilon\mathbf{p}$ is a tuple of predicate variables; the definition of $F^{\diamond}(\upsilon\mathbf{p})$ is completely parallel to the definiton of $F^{\diamond}(\upsilon\mathbf{f})$ given above: this is the result of replacing each nonnegated occurrence of each member of $\mathbf{p}$ by the corresponding member of $\upsilon\mathbf{p}$.[5] Both (3) and (7) use $F^{\diamond}$, although in different ways: the former is a uniqueness condition, and the latter is a minimality condition.

On the other hand, the semantics of causal theories in (Lifschitz 1997) refers to formulas of the form

$$\forall \upsilon\mathbf{c}(\ \cdots \leftrightarrow \upsilon\mathbf{c} = \mathbf{c}),$$

where $\mathbf{c}$ is the list of explainable symbols of the theory (it may include both predicate and function constants), and $\upsilon\mathbf{c}$ is a tuple of variables; see the section on causal logic below for details. The definition (3) of $\mathrm{SM}_{\mathbf{f}}[F]$ has the same syntactic form, but the left-hand side of the equivalence is formed now in a different way.

Additional information about the relationship between IF-programs on the one hand and Datalog programs and causal theories on the other is provided in Propositions 4 and 6 below.

[2] $\widetilde{\forall} F$ stands for the universal closure of F.

[3] Object constants can be viewed as function constants of arity 0 and thus are allowed to be members of $\mathbf{f}$.

[4] For a brief discussion of the syntax and semantics of second-order logic see, for instance, (Lifschitz, Morgenstern, & Plaisted 2008, Section 1.2.3).

[5] If $\mathbf{p}$ is a single unary predicate constant p then $\upsilon\mathbf{p} < \mathbf{p}$ stands for

$$\forall x(\upsilon p(x) \rightarrow p(x)) \wedge \neg\forall x(p(x) \rightarrow \upsilon p(x)).$$

For the general definition of $<$ on tuples of predicate symbols see (Lifschitz 2011, Section 4).

Stability-Preserving Transformations

From (Lifschitz, Pearce, & Valverde 2007) we know that two logic programs have the same stable models if the equivalence of their sets of rules can be justified in intuitionistic predicate logic with some additional postulates, such as the weak law of the excluded middle

$$\neg F \vee \neg\neg F \tag{8}$$

and the law of the excluded middle for equalities

$$t_1 = t_2 \vee t_1 \neq t_2. \tag{9}$$

(For the list of additional axioms see (Lifschitz, Pearce, & Valverde 2007, Section 3).) In the world of IF-programs, the situation is somewhat different. If the equivalence between R_1 and R_2 can be proved in positive logic (that is to say, using axiom schemas and inference rules of intuitionistic propositional logic without postulates for negation) then $(R_1, \mathbf{f})$ and $(R_2, \mathbf{f})$ have the same stable models, because $R_1^\diamond(\upsilon\mathbf{f})$ is equivalent to $R_2^\diamond(\upsilon\mathbf{f})$ in this case. But the class of stable models does change after some intuitionistically acceptable transformations. For instance, the rule

$$\neg(f(x) \neq y) \leftarrow f(x) = y$$

is intuitionistically trivial, but adding it to a logic program $(R, \mathbf{f})$ contributes the nontrivial conjunctive term

$$\forall xy(\upsilon f(x) = y \rightarrow \neg(f(x) \neq y))$$

to the antecedent of (6). This conjunctive term is equivalent to $\upsilon f = f$, and it makes (6) significantly weaker.

We need to distinguish between the rules $\bot \leftarrow F$ and $\neg F \leftarrow \top$, even though they are intuitionistically equivalent: the former contributes $\neg F^\diamond$ to the antecedent of the stability condition (6); the latter contributes $\neg F$. According to the definition of a constraint in the next section, rules of the form $\neg F$ are constraints, and rules of the form $\bot \leftarrow F$ are generally not.

On the other hand, replacing a rule of the form

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

with

$$F \vee \neg F,$$

which is not an intuitionistically equivalent transformation, preserves the class of stable models, because $(F \leftarrow \neg\neg F)^\diamond$ is equivalent to $(F \vee \neg F)^\diamond$. Replacing $\neg(F \wedge G)$ with $\neg F \vee \neg G$ in the head or in the body of a rule does not change the class of stable models either. These two transformations can be justified in intuitionistic logic with the weak law of the excluded middle (8).

The law of the excluded middle for equalities (9) is not acceptable in equivalent transformations of IF-programs when the terms t_1, t_2 contain explainable functions. It would allow us to replace the body of the first rule of (2) with $f(x) = a$, which would make the rule trivial.

It appears that "the logic of IF-programs" is intermediate between positive logic and the first-order logic of here-and-there introduced in (Lifschitz, Pearce, & Valverde 2007). It is neither weaker nor stronger than intuitionistic logic. Stability-preserving transformations for IF-programs require further study.

Constraints

In the context of IF-programs, a *constraint* is a rule without nonnegated occurrences of intensional functions. Adding a constraint to an IF-program affects the class of its stable models monotonically—it eliminates the models that violate the constraint:

Proposition 2 *For any IF-program $(R, \mathbf{f})$ and any constraint C, an interpretation I is a stable model of $(R \wedge C, \mathbf{f})$ iff I is a stable model of $(R, \mathbf{f})$ that satisfies C.*

Defaults and Choice

The first rule of program (2), which expresses that the equality $f(x) = a$ "holds by default," can be equivalently replaced with

$$f(x) = a \vee f(x) \neq a \tag{10}$$

(see the discussion of stability-preserving transformation above). More generally, for any terms t_1, t_2 the idea that the equality $t_1 = t_2$ "holds by default" can be expressed by the formula

$$t_1 = t_2 \vee t_1 \neq t_2.$$

We will denote this formula by $t_1 \approx t_2$. For instance, (10) can be written as $f(x) \approx a$.

Rules of the form

$$t \approx x \leftarrow B, \tag{11}$$

where x is a variable that occurs neither in t nor in B, are similar to choice rules in traditional answer set programming (Niemelä, Simons, & Soininen 1999). Rule (11) allows us to choose the value of t arbitrarily if B holds. We will write it as

$$\{t\} \leftarrow B.$$

For instance, the rules of the IF-program

$$\begin{array}{rl} f(x) = a & \leftarrow P(x), \\ \{f(x)\} & \leftarrow \neg P(x) \end{array} \tag{12}$$

(with intensional f) say that the value of f on any element of P is a, and that otherwise the values of f are arbitrary:

Proposition 3 *The stable models of program (12) are characterized by the formula*

$$\forall x(P(x) \rightarrow f(x) = a). \tag{13}$$

Relation to Causal Logic

IF-programs in which negated occurrences of intensional functions are "separated" from nonnegated occurrences can be translated into causal logic in the sense of (Lifschitz 1997).

Recall that a first-order causal theory T is defined by

- a list $\mathbf{c}$ of distinct function and/or predicate constants, called the *explainable symbols* of T, and
- a finite set of *causal rules* of the form $F \Leftarrow G$, where F and G are first-order formulas.

For each member c of $\mathbf{c}$, choose a new variable υc similar to c (that is to say, if c is a function constant then υc should be a function variable of the same arity; if c is a predicate constant then υc should be a predicate variable of the same arity). Let $\upsilon\mathbf{c}$ stand for the list of all these variables. By $T^\dagger(\upsilon\mathbf{c})$ we denote the conjunction of the formulas

$$\forall\mathbf{x}(G \to F^{\mathbf{c}}_{\upsilon\mathbf{c}}) \tag{14}$$

for all rules $F \Leftarrow G$ of T, where $\mathbf{x}$ is the list of all free variables of F, G. (The expression $F^{\mathbf{c}}_{\upsilon\mathbf{c}}$ denotes the result of substituting the variables $\upsilon\mathbf{c}$ for the corresponding constants $\mathbf{c}$ in F.) Semantically, T is considered shorthand for the sentence

$$\forall\upsilon\mathbf{c}(T^\dagger(\upsilon\mathbf{c}) \leftrightarrow \upsilon\mathbf{c} = \mathbf{c}). \tag{15}$$

Let $(R, \mathbf{f})$ be an IF-program such that all its rules have the form

$$H^+ \vee H^- \leftarrow B^+ \wedge B^-, \tag{16}$$

where H^+, B^+ are formulas without negated occurrences of intensional functions, and H^-, B^- are formulas without nonnegated occurrences of intensional functions. (If a conjunctive term of the body doesn't contain intensional functions, such as $P(x)$ in the second rule of (2), then we are free to choose whether to include it in B^+ or in B^-. Similarly, a disjunctive term of the head that doesn't contain intensional functions can be included either in H^+ or in H^-.) By T we denote the causal theory consisting of the rules

$$H^+ \vee \neg B^+ \Leftarrow B^- \wedge \neg H^- \tag{17}$$

for all rules (16) from R, with the explainable symbols $\mathbf{f}$. The idea of this transformation is that the difference between negated and nonnegated occurrences of symbols in an IF-program corresponds to the difference between occurrences of symbols in the body and in the head of a causal rule.

Proposition 4 *An interpretation I is a stable model of $(R, \mathbf{f})$ iff I is a model of causal theory T.*

For example, program (2) corresponds to the causal theory with the rules

$$\begin{array}{l} f(x) = a \ \Leftarrow \neg(f(x) \neq a), \\ f(x) = b \ \Leftarrow P(x), \end{array}$$

or, equivalently,

$$\begin{array}{l} f(x) = a \ \Leftarrow f(x) = a, \\ f(x) = b \ \Leftarrow P(x). \end{array}$$

(In causal logic, replacing the head or the body of a rule with an equivalent formula does not affect the class of models.) The modification of (2) in which the first rule is replaced with (10) corresponds to the same causal theory.

Using Proposition 4 in combination with stability-preserving transformations, we can turn any IF-program with quantifier-free rules into an equivalent causal theory. Conversely, if all rules of a causal theory are quantifier-free and all its explainable symbols are function symbols then it can be converted into an equivalent IF-program. Take, for instance, causal theory T_1 from (Lifschitz & Yang 2011):

$$\begin{array}{rl} \bot & \Leftarrow a = b, \\ c = a & \Leftarrow c = a, \\ c = b & \Leftarrow q, \end{array} \tag{18}$$

with the explainable object constant c. It can be converted into the IF-program

$$\begin{array}{l} a \neq b, \\ c \approx a, \\ c = b \leftarrow q. \end{array} \tag{19}$$

Describing Actions by IF-Programs

IF-programs, like nonmonotonic causal theories, can be used for describing effects of actions. The following program describes the effect of moving an object. (It is similar to causal theory T_2 from (Lifschitz & Yang 2011).) For simplicity, we only consider the time instants 0, 1 and the execution of the move action at time 0. The auxiliary symbol *none* is used as the value of $loc(x,t)$ when the arguments are "of a wrong kind" (that is, when x is not a physical object or when t is not a time instant). The rules are

$$\begin{array}{l} loc(x,0) \approx y \leftarrow obj(x) \wedge place(y), \\ loc(x,1) = y \leftarrow move(x,y) \wedge obj(x) \wedge place(y), \\ loc(x,1) \approx y \leftarrow loc(x,0) = y \wedge obj(x) \wedge place(y), \\ loc(x,t) = none \leftarrow \neg obj(x) \vee (t \neq 0 \wedge t \neq 1), \\ 0 \neq 1 \wedge 0 \neq none \wedge 1 \neq none, \end{array}$$

and the only intensional function is *loc*. The first rule says that initially an object can be at an arbitrary place. The second rule describes the effect of moving an object, and the third rule expresses the commonsense law of inertia for locations.

To describe the effect of an action on a Boolean-valued fluent by an IF-program, we represent Boolean values by object constants, say 0 and 1. For instance, the effect of the action *toggle* on the Boolean-valued fluent *on* can be described by the IF-program consisting of the constraints

$$\begin{array}{c} 0 \neq 1, \\ x = 0 \vee x = 1 \end{array} \tag{20}$$

and the rules

$$\begin{array}{l} \{on(0)\}, \\ on(1) = x \leftarrow on(0) \neq x \wedge toggle, \\ on(1) \approx x \leftarrow on(0) = x. \end{array}$$

Relation to the 1988 Definition of a Stable Model

Let Π be a finite set of rules of the form

$$A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n \tag{21}$$

($n \geq m \geq 0$), 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 IF-programs as follows. We reclassify all propositional atoms as intensional object constants, and add to the signature two non-intensional object constants 0, 1. Each rule (21) is rewritten as

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

For each atom A in the signature of Π we add the "minimization rule" $A \approx 0$ (by default, atoms get the value *false*). Finally, we add constraints (20). The resulting IF-program will be called the *functional image* of Π. For instance, the functional image of the one-rule program

$$p \leftarrow not\ q$$

consists of the rules

$$p = 1 \leftarrow q \neq 1,$$
$$p \approx 0,$$
$$q \approx 0$$

and constraints (20).

It is clear that models of (20) can be viewed as sets of propositional atoms.

Proposition 5 *The functional image of* Π *has the same stable models as* Π.

The toggle program from the section on describing actions is similar in some ways to functional images as defined above, but there is an essential difference: it includes the inertia rule

$$on(1) \approx x \leftarrow on(0) = x$$

instead of the minimization rule

$$on(1) \approx 0.$$

The definition of functional image and Proposition 5 can be extended to disjunctive programs in a straightforward way. In the next section we show how to extend them to Datalog programs in the sense of (Lifschitz 2011).

Relation to Datalog Programs

Let $(R, \mathbf{p})$ be a Datalog program that has two object constants in its signature, say 0 and 1, and includes the constraint $0 \neq 1$. The *functional image* of $(R, \mathbf{p})$ is formed as follows. We reclassify each predicate constant p from $\mathbf{p}$ as a function constant of the same arity. In the rules of R, we replace every atomic subformula $p(\mathbf{t})$ such that p is a member of $\mathbf{p}$ with $p(\mathbf{t}) = 1$. Finally, for each p from $\mathbf{p}$ we add the minimization rule

$$p(\mathbf{x}) \approx 0 \tag{22}$$

and the constraint

$$\neg\neg(p(\mathbf{x}) = 0 \vee p(\mathbf{x}) = 1), \tag{23}$$

where $\mathbf{x}$ is a tuple of distinct object variables.

We will identify the interpretations of the original signature that satisfy $0 \neq 1$ with the interpretations of the modified signature that satisfy $0 \neq 1$ and

$$\forall \mathbf{x}(p(\mathbf{x}) = 0 \vee p(\mathbf{x}) = 1). \tag{24}$$

Proposition 6 *The functional image of* $(R, \mathbf{p})$ *has the same stable models as* $(R, \mathbf{p})$.

In other words, in the presence of two distinct object constants a Datalog program has the same meaning as its functional image.

An Approach to Implementation

In some cases, answer set solvers can be used to generate the models of a causal theory that have a given finite universe even in the presence of explainable functions (Lifschitz & Yang 2011). The idea is to represent an n-ary function constant f by its graph—a predicate constant of arity $n+1$. In view of the close relationship between IF-programs and causal theories with explainable functions, the stable models of an IF-program that have a given finite universe can be sometimes generated in a similar way.

Consider, for instance, the problem of generating all stable models I of program (19) such that

$$|I| = \{a, b\},\ a^I = a,\ b^I = b. \tag{25}$$

(Here $|I|$ is the universe of the interpretation I; appending the superscript I turns a constant into the object that interprets that constant.) This is equivalent to the problem of generating the models of causal theory (18) that satisfy these conditions. As discussed in (Lifschitz & Yang 2011, Section 7.2), this can be accomplished by running the solver CLINGO[6] on the following input:

```
u(a;b).  #domain u(X).
{q}.
p(a) :- not -p(a).
p(b) :- q.
-p(X) :- not p(X).
:- not 1{p(Z):u(Z)}1.
:- not p(X), not -p(X).
```

In this program, $p(x)$ represents the condition $x = c$. The first line expresses that the universe `u` consists of `a` and `b`, and that `X` is a variable for arbitrary elements of `u`. The choice rule in the second line says that `q` can be assigned an arbitrary value. (It is included because `q`, unlike `p`, does not correspond to an intensional constant of program (19).) The next two lines correspond to the last two rules of (19). (There is no need to represent the constraint $a \neq b$; it is taken by CLINGO for granted.) The rest is standard for the translation process described in (Lifschitz & Yang 2011). In particular, the second line from the end expresses the uniqueness of an object with the property p.

Given this input, CLINGO generates two stable models: one containing `q` and `p(b)`, the other containing `p(a)`. They correspond to the two stable models of IF-program (19) that satisfy conditions (25): in one of them

$$q^I = true,\ c^I = b,$$

in the other

$$q^I = false,\ c^I = a.$$

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

The range of applicability of this approach requires further study.

Related Work

Logic programs with functions have received considerable attention in the literature on answer set programming,[7] and function symbols are allowed in the input languages of most answer set solvers. But many researchers make all functions "predefined": a function either corresponds to a specific arithmetical operation or operates by simply prepending its name to the list of arguments, as in Herbrand interpretations. The values of such a function cannot be characterized using the rules of a program.

Answer set programming with functions in the sense of (Lin & Wang 2008) is different. Under that approach, the values taken by a function are object constants. We can express, for instance, that the color of x is red by writing $clr(x) = red$. The formalization of the graph coloring problem in (Lin & Wang 2008) uses the constraint

$$\leftarrow arc(x, y), clr(x) = clr(y).$$

On the other hand, the language of that paper allows equalities in the bodies of rules only, not in the heads. It appears that assumptions about default values of a function, such as the first rule of IF-program (2), cannot be expressed in that language.

Functional answer set programming in the sense of (Cabalar 2011) is free of this limitation. An implementation of this language, called LPPF, can be downloaded from the Equilibrium Logic Workbench.[8] As pointed out in the introduction, the main difference between the two nonmonotonic logics is that we use total functions and uniqueness, instead of partial functions and minimization. The definition in (Cabalar 2011) is stated in model-theoretic terms, and the universe of a model is assumed to be the set of all ground terms that do not contain evaluable (in our terminology, intensional) functions.

Default values of functions can be specified also in the language of weight constraint programs with evaluable functions (Wang *et al.* 2010). For example, the counterpart of the first rule of (2) in that language is

$$f(x) = a \leftarrow [\, f(x) \neq a : 1 \,]\, 0.$$

The language of the inference engine EZCSP,[9] which integrates answer set programming with constraint programming, allows us to talk about default values of functions as well.

Conclusion

Some features of logic programs with intensional functions make them similar to traditional logic programs under the stable model semantics; in other ways they are reminiscent of nonmonotonic causal theories. Many questions about properties of these programs remain at this point unanswered, and they provide topics for future work. What is the model-theoretic meaning of the semantics of IF-programs? How can one characterize the strong equivalence relation (Lifschitz, Pearce, & Valverde 2001; 2007) for IF-programs? What are advantages and disadvantages of the language of IF-programs as a knowledge representation tool, in comparison with causal theories? What kinds of IF-programs can be translated into the input languages of the existing answer set solvers? Can IF-programs be related to the systems from (Cabalar 2011; Lin & Wang 2008; Wang *et al.* 2010) in a mathematically precise way?

Acknowledgements

Thanks to Joohyung Lee for his comments on a draft of this paper and for detecting an error in an earlier attempt to prove Proposition 6. I am grateful also to Marcello Balduccini, Pedro Cabalar, Luis Fariñas del Cerro, Michael Gelfond, Yuliya Lierler, Fangzhen Lin, Ilkka Niemelä, David Pearce, Yisong Wang, Fangkai Yang, and the anonymous referees for their comments and suggestions. This research was partially supported by the National Science Foundation under Grant IIS-0712113.

Appendix

Proof of Proposition 1

Proposition 1 *Formula (4) is equivalent to (5).*

Proof Formula (4) is equivalent to

$$\forall \upsilon f((\forall x(f(x) = a \rightarrow \upsilon f(x) = a) \wedge \forall x(P(x) \rightarrow \upsilon f(x) = b)) \leftrightarrow \upsilon f = f).$$

The implication right-to-left is equivalent to the first conjunctive term of (5); it expresses that the set of values of x satisfying $f(x) = b$ is a superset of P. The implication left-to-right expresses that, subject to this restriction on f, the set of values of x satisfying $f(x) = a$ is maximal. This is equivalent to the second conjunctive term of (5).

Proof of Proposition 2

Proposition 2 *For any IF-program $(R, \mathbf{f})$ and any constraint C, an interpretation I is a stable model of $(R \wedge C, \mathbf{f})$ iff I is a stable model of $(R, \mathbf{f})$ that satisfies C.*

Proof

$$\begin{aligned}
&\mathrm{SM}_{\mathbf{f}}[R \wedge C] \\
&\quad \leftrightarrow R \wedge C \wedge \forall \upsilon \mathbf{f}(R^{\diamond}(\upsilon \mathbf{f}) \wedge C^{\diamond}(\upsilon \mathbf{f}) \rightarrow \upsilon \mathbf{f} = \mathbf{f}) \\
&\quad = R \wedge C \wedge \forall \upsilon \mathbf{f}(R^{\diamond}(\upsilon \mathbf{f}) \wedge C \rightarrow \upsilon \mathbf{f} = \mathbf{f}) \\
&\quad \leftrightarrow R \wedge C \wedge \forall \upsilon \mathbf{f}(R^{\diamond}(\upsilon \mathbf{f}) \rightarrow \upsilon \mathbf{f} = \mathbf{f}) \\
&\quad \leftrightarrow \mathrm{SM}_{\mathbf{f}}[R] \wedge C.
\end{aligned}$$

[7] See, for instance, (Syrjänen 2001; Calimeri *et al.* 2008).

[8] `http://www.equilibriumlogic.net`

[9] `http://marcy.cjb.net/ezcsp/`

Proof of Proposition 3

Proposition 3 *The stable models of program (12) are characterized by formula (13).*

Proof The stable models of program (12) are characterized by the formula

$$\forall \upsilon f((\forall x(P(x) \to \upsilon f(x) = a) \\ \wedge \forall xy(\neg P(x) \to \upsilon f(x) = y \vee f(x) \neq y)) \\ \leftrightarrow \upsilon f = f)$$

or, equivalently,

$$\forall \upsilon f((\forall x(P(x) \to \upsilon f(x) = a) \\ \wedge \forall x(\neg P(x) \to \upsilon f(x) = f(x))) \\ \leftrightarrow \upsilon f = f).$$

The implication right to left is equivalent to (13). The implication left to right is entailed by (13).

Proof of Proposition 4

In the statement of Proposition 4, the IF-program $(R, \mathbf{f})$ and the causal theory T are as described at the beginning of the section on causal logic.

Proposition 4 *An interpretation I is a stable model of $(R, \mathbf{f})$ iff I is a model of causal theory T.*

Proof Formula $R^{\diamond}(\upsilon\mathbf{f})$ is the conjunction of the formulas

$$\forall \mathbf{x}\left((B^+)^{\mathbf{f}}_{\upsilon\mathbf{f}} \wedge B^- \to (H^+)^{\mathbf{f}}_{\upsilon\mathbf{f}} \vee H^-\right) \tag{26}$$

for all rules (16) of R, where $\mathbf{x}$ is the list of free variables of H^+, H^-, B^+, B^-. Formula $T^{\dagger}(\upsilon\mathbf{f})$ is the conjunction of the formulas

$$\forall \mathbf{x}\left(B^- \wedge \neg H^- \to (H^+)^{\mathbf{f}}_{\upsilon\mathbf{f}} \vee \neg(B^+)^{\mathbf{f}}_{\upsilon\mathbf{f}}\right). \tag{27}$$

It is clear that (27) is equivalent to (26).

Proof of Proposition 5

Recall that Π is a finite set of rules of the form (21).

Proposition 5 *The functional image of Π has the same stable models as Π.*

Proof We will identify Π with the conjunction of its rules written as propositional formulas. The stable models of Π can be characterized as the models of the second-order propositional formula ("QBF")

$$\Pi \wedge \neg\exists \upsilon\mathbf{p}((\upsilon\mathbf{p} < \mathbf{p}) \wedge \Pi^{\diamond}(\upsilon\mathbf{p})) \tag{28}$$

(Lifschitz 2011, Remark 2). For any second-order propositional formula F, by ϕF we will denote the expression obtained by appending the symbols $=1$ to each atomic part of F. This expression can be viewed as a first-order formula if we treat the propositional constants occurring in F as object constants, and the propositional variables occurring in F as object variables. We will identify truth assignments with corresponding models of formulas (20); then ϕF has the same meaning as F. Our goal is to prove that the result of applying ϕ to formula (28) is equivalent to $\mathrm{SM}_{\mathbf{p}}[R]$, where R is the set of rules of the functional image of Π.

By the definition of functional image, R is obtained from $\phi\Pi$ by adding the rules $p \approx 0$ for all members p of $\mathbf{p}$ and constraints (20). Assuming (20),

$$\begin{aligned}
&\mathrm{SM}_{\mathbf{p}}[R] \\
&\quad\leftrightarrow \mathrm{SM}_{\mathbf{p}}\left[\phi\Pi \wedge \textstyle\bigwedge_p (p = 0 \vee p \neq 0)\right] \\
&\quad\leftrightarrow \phi\Pi \wedge \forall \upsilon\mathbf{p}(\phi\Pi^{\diamond}(\upsilon\mathbf{p}) \wedge \textstyle\bigwedge_p(\upsilon p = 0 \vee p \neq 0) \\
&\qquad\qquad\qquad\qquad \to \mathbf{p} = \upsilon\mathbf{p}) \\
&\quad\leftrightarrow \phi\Pi \wedge \forall \upsilon\mathbf{p}(\phi\Pi^{\diamond}(\upsilon\mathbf{p}) \wedge \textstyle\bigwedge_p(\upsilon p = 1 \to p = 1) \\
&\qquad\qquad\qquad\qquad \to \mathbf{p} = \upsilon\mathbf{p}) \\
&\quad\leftrightarrow \phi\Pi \wedge \forall \upsilon\mathbf{p}(\phi(\Pi^{\diamond}(\upsilon\mathbf{p}) \wedge (\upsilon\mathbf{p} \leq \mathbf{p})) \to \mathbf{p} = \upsilon\mathbf{p}) \\
&\quad\leftrightarrow \phi(\Pi \wedge \forall \upsilon\mathbf{p}(\Pi^{\diamond}(\upsilon\mathbf{p}) \wedge (\upsilon\mathbf{p} \leq \mathbf{p}) \to \mathbf{p} = \upsilon\mathbf{p})) \\
&\quad\leftrightarrow \phi(\Pi \wedge \forall \upsilon\mathbf{p}\neg(\Pi^{\diamond}(\upsilon\mathbf{p}) \wedge (\upsilon\mathbf{p} < \mathbf{p}))) \\
&\quad\leftrightarrow \phi(\Pi \wedge \neg\exists \upsilon\mathbf{p}((\upsilon\mathbf{p} < \mathbf{p}) \wedge \Pi^{\diamond}(\upsilon\mathbf{p}))).
\end{aligned}$$

Proof of Proposition 6

Recall that $(R, \mathbf{p})$ is a Datalog program that includes the constraint $0 \neq 1$.

Proposition 6 *The functional image of $(R, \mathbf{p})$ has the same stable models as $(R, \mathbf{p})$.*

Proof The rules R' of the functional image of $(R, \mathbf{p})$ include ϕR (see the proof of Proposition 5 for the definition of ϕ), rules (22) and constraints (23). Assuming $0 \neq 1$ and (24),

$$\begin{aligned}
&\mathrm{SM}_{\mathbf{p}}[R'] \\
&\quad\leftrightarrow \mathrm{SM}_{\mathbf{p}}\left[\phi R \wedge \textstyle\bigwedge_p \forall \mathbf{x}(p(\mathbf{x}) = 0 \vee p(\mathbf{x}) \neq 0)\right] \\
&\quad\leftrightarrow \phi R \wedge \forall \upsilon\mathbf{p}(\phi R^{\diamond}(\upsilon\mathbf{p}) \\
&\qquad\quad \wedge \textstyle\bigwedge_p \forall \mathbf{x}(\upsilon p(\mathbf{x}) = 0 \vee p(\mathbf{x}) \neq 0) \\
&\qquad\qquad\qquad\qquad \to \mathbf{p} = \upsilon\mathbf{p}) \\
&\quad\leftrightarrow \phi R \wedge \forall \upsilon\mathbf{p}(\phi R^{\diamond}(\upsilon\mathbf{p}) \\
&\qquad\quad \wedge \textstyle\bigwedge_p \forall \mathbf{x}(p(\mathbf{x}) = 0 \to \upsilon p(\mathbf{x}) = 0) \\
&\qquad\qquad \to \textstyle\bigwedge_p \forall \mathbf{x}y(p(\mathbf{x}) = y \to \upsilon p(\mathbf{x}) = y) \\
&\quad\leftrightarrow \phi R \wedge \forall \upsilon\mathbf{p}(\phi R^{\diamond}(\upsilon\mathbf{p}) \\
&\qquad\quad \wedge \textstyle\bigwedge_p \forall \mathbf{x}(p(\mathbf{x}) = 0 \to \upsilon p(\mathbf{x}) = 0)) \\
&\qquad\qquad \to \textstyle\bigwedge_p \forall \mathbf{x}(p(\mathbf{x}) = 1 \to \upsilon p(\mathbf{x}) = 1) \\
&\quad\leftrightarrow \phi R \wedge \forall \upsilon\mathbf{p}(\textstyle\bigwedge_p \forall \mathbf{x}(p(\mathbf{x}) = 0 \to \upsilon p(\mathbf{x}) = 0) \\
&\qquad\qquad\qquad\qquad \to \phi(R^{\diamond}(\upsilon\mathbf{p}) \to \mathbf{p} \leq \upsilon\mathbf{p})).
\end{aligned}$$

Consider the second conjunctive term of the last formula:

$$\forall \upsilon\mathbf{p}(\textstyle\bigwedge_p \forall \mathbf{x}(p(\mathbf{x}) = 0 \to \upsilon p(\mathbf{x}) = 0) \\ \to \phi(R^{\diamond}(\upsilon\mathbf{p}) \to \mathbf{p} \leq \upsilon\mathbf{p})). \tag{29}$$

Under the assumption $0 \neq 1$, it is equivalent to

$$\forall \upsilon\mathbf{p}(\textstyle\bigwedge_p \forall \mathbf{x}(p(\mathbf{x}) = 0 \to \upsilon p(\mathbf{x}) \neq 1) \\ \to \phi(R^{\diamond}(\upsilon\mathbf{p}) \to \mathbf{p} \leq \upsilon\mathbf{p})). \tag{30}$$

Indeed, the implication from (30) to (29) is obvious; assume (29) and

$$\textstyle\bigwedge_p \forall \mathbf{x}(p(\mathbf{x}) = 0 \to \upsilon p(\mathbf{x}) \neq 1). \tag{31}$$

Define $\upsilon\mathbf{p}'$ by the conditions

$$\forall \mathbf{x}(\upsilon p(\mathbf{x}) = 1 \to \upsilon p'(\mathbf{x}) = 1), \\ \forall \mathbf{x}(\upsilon p(\mathbf{x}) \neq 1 \to \upsilon p'(\mathbf{x}) = 0). \tag{32}$$

From (31) and the second of conditions (32),

$$\bigwedge_p \forall \mathbf{x}(p(\mathbf{x}) = 0 \to \upsilon p'(\mathbf{x}) = 0).$$

By (29), it follows that

$$\phi(R^\diamond(\upsilon \mathbf{p}') \to \mathbf{p} \leq \upsilon \mathbf{p}'). \tag{33}$$

On the other hand, from (32),

$$\forall \mathbf{x}(\upsilon p(\mathbf{x}) = 1 \leftrightarrow \upsilon p'(\mathbf{x}) = 1). \tag{34}$$

Since every occurrence of $\upsilon p'$ in (33) is within a subformula of the form $\upsilon p'(\cdots) = 1$, from (33) and (34) we can conclude that

$$\phi(R^\diamond(\upsilon \mathbf{p}) \to \mathbf{p} \leq \upsilon \mathbf{p}).$$

This completes the proof of the equivalence between (29) and (30). Consequently

$$\begin{aligned}
&\mathrm{SM}_{\mathbf{p}}[R'] \\
&\quad \leftrightarrow \phi R \wedge \forall \upsilon \mathbf{p}(\bigwedge_p \forall \mathbf{x}(p(\mathbf{x}) = 0 \to \upsilon p(\mathbf{x}) \neq 1) \\
&\qquad\qquad \to \phi(R^\diamond(\upsilon \mathbf{p}) \to \mathbf{p} \leq \upsilon \mathbf{p})) \\
&\quad \leftrightarrow \phi R \wedge \forall \upsilon \mathbf{p}(\bigwedge_p \forall \mathbf{x}(\upsilon p(\mathbf{x}) = 1 \to p(\mathbf{x}) = 1) \\
&\qquad\qquad \to \phi(R^\diamond(\upsilon \mathbf{p}) \to \mathbf{p} \leq \upsilon \mathbf{p})) \\
&\quad \leftrightarrow \phi(R \wedge \forall \upsilon \mathbf{p}((\upsilon \mathbf{p} \leq \mathbf{p}) \to (R^\diamond(\upsilon \mathbf{p}) \to \mathbf{p} \leq \upsilon \mathbf{p}))) \\
&\quad \leftrightarrow \phi(R \wedge \neg \exists \upsilon \mathbf{p}(R^\diamond(\upsilon \mathbf{p}) \wedge (\upsilon \mathbf{p} \leq \mathbf{p}) \wedge \neg(\mathbf{p} \leq \upsilon \mathbf{p}))) \\
&\quad \leftrightarrow \phi(R \wedge \neg \exists \upsilon \mathbf{p}((\upsilon \mathbf{p} < \mathbf{p}) \wedge R^\diamond(\upsilon \mathbf{p}))) \\
&\quad = \phi\left(\mathrm{SM}_{\mathbf{p}}[R]\right).
\end{aligned}$$

References

Bartholomew, M., and Lee, J. 2012. Stable models of formulas with intensional functions. In this volume.

Cabalar, P. 2011. Functional answer set programming. *Theory and Practice of Logic Programming* 11:203–234.

Calimeri, F.; Cozza, S.; Ianni, G.; and Leone, N. 2008. Computable functions in ASP: theory and implementation. In *Proceedings of International Conference on Logic Programming (ICLP)*, 407–424.

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

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.

Lifschitz, V., and Yang, F. 2011. 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.; Morgenstern, L.; and Plaisted, D. 2008. Knowledge representation and classical logic. In van Harmelen, F.; Lifschitz, V.; and Porter, B., eds., *Handbook of Knowledge Representation*. Elsevier. 3–88.

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

Lifschitz, V.; Pearce, D.; and Valverde, A. 2007. A characterization of strong equivalence for logic programs with variables. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 188–200.

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

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence* 138:39–54.

Lifschitz, V. 2011. Datalog programs and their stable models[10]. In *Datalog Reloaded: First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers*. Springer.

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.

McCain, N., and Turner, H. 1997. Causal theories of action and change. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 460–465.

Niemelä, I.; Simons, P.; and Soininen, T. 1999. Stable model semantics for weight constraint rules. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 317–331.

Pearce, D. 1997. A new logical characterization of stable models and answer sets. In Dix, J.; Pereira, L.; and Przymusinski, T., eds., *Non-Monotonic Extensions of Logic Programming (Lecture Notes in Artificial Intelligence 1216)*, 57–70. Springer.

Shanahan, M. 1997. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press.

Syrjänen, T. 2001. Omega-restricted logic programs. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning*, 267–279.

Wang, Y.; You, J.-H.; Lin, F.; Yuan, L.-Y.; and Zhang, M. 2010. Weight constraint programs with evaluable functions. *Annals of Mathematics and Artificial Intelligence* 60:341–380.

[10] `http://www.cs.utexas.edu/users/vl/papers/dpsm.pdf`