Two-Valued Logic Programs

Vladimir Lifschitz

University of Texas at Austin, USA

Abstract

We define a nonmonotonic formalism that shares some features with three other systems of nonmonotonic reasoning—default logic, logic programming with strong negation, and nonmonotonic causal logic—and study its possibilities as a knowledge representation tool.

1 Introduction

This note is motivated by the desire to understand the existing methodologies of answer set programming (ASP) [11, 14, 2, 4, 9]—the approach to knowledge representation based on the answer set semantics of logic programs [5]. An answer set is a set of ground literals that is consistent but possibly incomplete. Thus an answer set can be thought of as a function that assigns to each ground atom A one of three values: *true* (A belongs to the set), *false* ($\neg A$ belongs to the set), or *unknown* (the set contains neither A nor $\neg A$). Most applications of ASP do not exploit the possibility of distinguishing between three truth values of an atom in an answer set, but there are important exceptions. The nonmonotonic formalism introduced in this note is designed to facilitate the discussion of differences between "three-valued" and "two-valued" uses of ASP.

Two-valued logic programs are essentially a special case of nondisjunctive logic programs with strong (classical) negation under the answer set semantics. They also share some features with default logic [16] and with nonmonotonic causal logic in the sense of [13]. As in the case of default logic, the nonmonotonicity of two-valued logic programs is determined by the use of "justifications." On the other hand, literals play a special role in their syntax, as they do in the definition of an answer set in [5], and this fact allows us to make their semantics relatively simple: it does not refer to deductive closure in the sense of classical logic. Finally, as in nonmonotonic causal logic, their semantics is defined in terms of two-valued truth assignments—or, in other words, consistent and complete sets of literals—rather than (possibly incomplete) extensions or (possibly incomplete) answer sets.

This note is simultaneously submitted to NMR 2012, a workshop that does not retain copyright.

2 Definitions

2.1 Syntax

In this note, *formulas* are propositional formulas formed from a fixed set σ of atoms. A *(two-valued) rule* is an expression of the form

$$L_0 \leftarrow L_1, \dots, L_n \ : \ F, \tag{1}$$

where the *head* L_0 and the *premises* $L_1, \dots, L_n$ ($n \geq 0$) are literals, and the *justification* F is a formula. Rule (1) reads: derive L_0 from $L_1, \dots, L_n$ if F is a consistent assumption.

A pair of rules of the form

$$\begin{aligned} A &\leftarrow L_1, \dots, L_n \ : \ F \wedge A, \\ \neg A &\leftarrow L_1, \dots, L_n \ : \ F \wedge \neg A, \end{aligned}$$

where A is an atom, can be abbreviated as

$$\{A\} \leftarrow L_1, \ldots, L_n \ : \ F \tag{2}$$

("derive any of the literals A, $\neg A$ from $L_1, \ldots, L_n$ if that literal is consistent with assumption F"). This abbreviation is similar to choice rules in the sense of [15]. Both in (1) and in (2), if F is $\top$ (truth) then we will drop the colon and F at the end of the rule. If, in addition, $n = 0$ then $\leftarrow$ can be dropped too.

A *(two-valued) program* is a set of rules.

2.2 Semantics

As in classical propositional logic, an *interpretation* is a function from σ to $\{false, true\}$. We will identify an interpretation I with the set of literals that are satisfied by I.

The *reduct* of a program Π relative to an interpretation I is the set of rules

$$L_0 \leftarrow L_1, \ldots, L_n \tag{3}$$

corresponding to the rules (1) of Π for which $I \models F$. We say that I is a *model* of Π if the smallest set of literals closed under the rules (3) equals I. In other words, models of Π are fixpoints of the operator α_Π from interpretations to sets of literals defined as follows: $\alpha_\Pi(I)$ is the smallest set of literals closed under the reduct of Π relative to I.

It is clear that the set of models of a program is not affected by replacing the justification of a rule with an equivalent formula. It is clear also that every literal that belongs to a model of Π is the head of a rule of Π. It follows that if some atom from σ does not occur in the heads of rules then the program is inconsistent (that is, has no models). This is a property that two-valued programs share with causal theories in the sense of [13].

2.3 Example

Let Π be the program

$$\begin{array}{l} \{a\}, \\ b \leftarrow a, \end{array} \tag{4}$$

or, written in full,

$$\begin{array}{rl} a & \leftarrow : a, \\ \neg a & \leftarrow : \neg a, \\ b & \leftarrow a \ : \ \top, \end{array}$$

with $\sigma = \{a, b\}$. Since Π has no rules with the head $\neg b$, the only possible models are $I_1 = \{a, b\}$ and $I_2 = \{\neg a, b\}$. The reduct of Π relative to I_1 consists of the rules a and $b \leftarrow a$, so that $\alpha_\Pi(I_1) = \{a, b\} = I_1$; I_1 is a model. The reduct relative to I_2 consists of the rules $\neg a$ and $b \leftarrow a$, so that $\alpha_\Pi(I_2) = \{\neg a\} \neq I_2$; I_2 is not a model.

2.4 Constraints

Adding a pair of rules of the form

$$\begin{array}{rl} A & \leftarrow : F, \\ \neg A & \leftarrow : F \end{array} \tag{5}$$

to a program Π eliminates the models of Π that satisfy F. (Proof: adding these rules makes the reduct of the program relative to I inconsistent if I satisfies F, and does not affect the reduct otherwise.) We will call (5) a *constraint* and write it as $\leftarrow F$.

2.5 Clausal Form

We say that a program Π is in *clausal form* if each of its justifications is a conjunction of literals (possibly the empty conjunction ⊤). For instance, program (4) is in clausal form.

Replacing a rule of the form

$$L_0 \leftarrow L_1, \dots, L_n \ : \ F \vee G$$

in any program with the pair of rules

$$\begin{array}{l} L_0 \leftarrow L_1, \dots, L_n \ : \ F, \\ L_0 \leftarrow L_1, \dots, L_n \ : \ G \end{array}$$

does not affect the set of models. (Proof: for any interpretation I, the reduct relative to I remains the same.) It follows that any program can be converted to clausal form by rewriting the justifications in disjunctive normal form and then breaking every rule into several rules corresponding to the disjunctive terms of its justification.

3 Relation to Traditional ASP Programs

3.1 Reduction to Programs with Strong Negation

As mentioned in the introduction, two-valued programs are essentially a special case of nondisjunctive programs with strong negation. To make that claim precise, we will define a simple translation that turns any two-valued program Π in clausal form into a program with strong negation. That program, *tv2sn*(Π), is the set of rules

$$L_0 \leftarrow L_1, \dots, L_n, \mathit{not}\ \overline{L_{n+1}}, \dots, \mathit{not}\ \overline{L_p}$$

for all rules

$$L_0 \leftarrow L_1, \dots, L_n \ : \ L_{n+1} \wedge \dots \wedge L_p$$

of Π. (By $\overline{L}$ we denote the literal complementary to L.) For instance, *tv2sn* turns program (4) into

$$\begin{array}{rl} a & \leftarrow \mathit{not}\ \neg a, \\ \neg a & \leftarrow \mathit{not}\ a, \\ b & \leftarrow a. \end{array} \tag{6}$$

An interpretation I is a model of Π iff I is an answer set of *tv2sn*(Π). (Proof: the reduct of *tv2sn*(Π) relative to I in the sense of [5] is identical to the reduct of Π relative to I.) In other words, models of Π are identical to complete answer sets of *tv2sn*(Π). For instance, program (6) has two answer sets, $\{a, b\}$ and $\{\neg a\}$. The first of them is the only model of (4); the second is incomplete.

Incomplete answer sets of a program with strong negation can be eliminated by adding the rules

$$\leftarrow \mathit{not}\ A, \mathit{not}\ \neg A \tag{7}$$

for all atoms A. Consequently models of a program Π in clausal form are identical to the answer sets of the program obtained from *tv2sn*(Π) by adding rules (7) for all A from σ.

3.2 Complete Answer Sets in Disguise

In many ASP programs, strong negation is not used at all. Answer sets of such a program are sets of positive literals; the intuition is that the falsity of an atom is indicated by its absence in the answer set, rather than the presence of its negation. In this situation, we can think of an answer set consisting of positive literals as a "complete answer set in disguise"—as a complete answer set X with all negative literals removed (symbolically, $X \cap \sigma$).

Similarly, a program without strong negation can be viewed as a "two-valued program in disguise." Let Π be a set of rules of the form

$$A_0 \leftarrow A_1, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_p, \tag{8}$$

where each A_i is an atom. By $\mathit{lp2tv}(\Pi)$ we denote the two-valued program consisting of the rules

$$A_0 \leftarrow A_1, \ldots, A_n\ :\ \neg A_{n+1} \wedge \cdots \wedge \neg A_p$$

for all rules (8) of Π, and the rules

$$\neg A \leftarrow :\ \neg A \tag{9}$$

for all atoms A. Rule (9) makes the closed world assumption for A explicit.

Answer sets of Π can be characterized as sets of the form $X \cap \sigma$, where X is a model of $\mathit{lp2tv}(\Pi)$. (Proof: $\mathit{tv2sn}(\mathit{lp2tv}(\Pi))$ is the closed world interpretation of Π in the sense of [5, Section 6].) Thus the map $X \mapsto X \cap \sigma$ is a 1–1 correspondence between the models of $\mathit{lp2tv}(\Pi)$ and the models of Π.

Consider, for instance, the program Π consisting of one rule $a \leftarrow \mathit{not}\ b$. The corresponding two-valued program is

$$\begin{aligned} a &\leftarrow :\ \neg b, \\ \neg a &\leftarrow :\ \neg a, \\ \neg b &\leftarrow :\ \neg b. \end{aligned}$$

Its only model is $\{a, \neg b\}$. By removing the negative literal $\neg b$ from it, we get $\{a\}$, the only answer set of Π.

4 Relation to Causal Logic

Recall that a causal theory in the sense of [13] is a set of rules of the form $F \leftarrow G$, where F and G are propositional formulas. The *reduct* of a causal theory T relative to an interpretation I is the set of the heads F of all rules $F \leftarrow G$ of T for which I satisfies G. An interpretation I is a *model* of a causal theory T if the reduct of T relative to I is satisfied by I and is not satisfied by any other interpretation. This semantics formalizes the philosophical principle that McCain and Turner call the law of universal causation.

A causal theory is *definite* if the head of each of its rules is a literal. For any definite causal theory T, we define the corresponding two-valued program $\mathit{ct2tv}(T)$ as the set of rules $F \leftarrow :\ G$ for all rules $F \leftarrow G$ of T. Models of any definite causal theory T are identical to models of program $\mathit{ct2tv}(T)$. (Proof: consider the reduct X of a definite causal theory T relative to an interpetation I; I is the only interpretation satisfying X iff $X = I$.) In other words, definite causal theories are essentially two-valued programs whose rules have no premises. We can say also that two-valued programs generalize definite causal theories by allowing "logic programming style premises" in the bodies of rules.

If the bodies of rules of a definite causal theory T are conjunctions of literals then $\mathit{ct2tv}(T)$ is a program in clausal form, and the transformation *tv2sn* defined above can be used to

turn that program into a program with strong negation. By composing *ct2tv* with *tv2sn* we get the translation from the language of causal theories into logic programming with strong negation familiar from [12, Section 6.3.3].

5 Representing Action Descriptions by Two-Valued Programs

Consider a finite set σ of propositional atoms divided into two groups, *fluents* and *elementary actions*. An *action* is a function from elementary actions to truth values. A *transition system* T is determined by a set of functions from fluents to truth values, called the *states* of T, and a set of triples $\langle s_0, a, s_1\rangle$, where s_0 and s_1 are states of T, and a is an action. These triples are called the *transitions* of T. A transition system can be visualized as a directed graph that has states as its vertices, with an edge from s_0 to s_1 labeled a for every transition $\langle s_0, a, s_1\rangle$. Informally speaking, a transition $\langle s_0, a, s_1\rangle$ expresses the possibility of the system changing its state from s_0 to s_1 when the elementary actions to which a assigns the value *true* are concurrently executed.

Action description languages $\mathcal{B}$ and $\mathcal{C}$, defined in [6, Section 5, 6] and [8], and reviewed in [7, Section 2], serve for describing action domains by specifying transition systems. They are closely related to logic programs under the answer set semantics [1, 10]. In this section we show how the semantics of $\mathcal{B}$ and of a large ("definite") fragment of $\mathcal{C}$ can be characterized in terms of two-valued programs.

5.1 Translating $\mathcal{B}$-Descriptions

This review of the syntax of $\mathcal{B}$ follows [7, Section 2.1.1]. A *fluent literal* is a literal containing a fluent. A *condition* is a set of fluent literals. An *action description in the language* $\mathcal{B}$, or a $\mathcal{B}$*-description*, is a set of expressions of two forms: *static laws*

$$L \textbf{ if } C,$$

where L is a fluent literal, and C is a condition, and *dynamic laws*

$$e \textbf{ causes } L \textbf{ if } C,$$

where E is an elementary action, L is a fluent literal, and C is a condition. The semantics of the language (see, for instance, [7, Section 2.1.2]) defines, for every $\mathcal{B}$-description D, which transition system it represents.

The set of transitions of that system can be described by the program $b2tv(D)$, defined as follows. Its signature σ_1 consists of the symbols of the forms

$$f(0),\ e(0),\ f(1), \tag{10}$$

where f is a fluent and e is an elementary action. Its rules are

(i) $L(t) \leftarrow L_1(t), \dots, L_n(t)$, where $t = 0, 1$, for each static law

$$L \textbf{ if } L_1, \dots, L_n$$

from D;

(ii) $L(1) \leftarrow e(0), L_1(0), \dots, L_n(0)$ for each dynamic law

$$e \textbf{ causes } L \textbf{ if } L_1, \dots, L_n$$

from D;

(iii) $L(1) \leftarrow L(0) : L(1)$ for every fluent literal L,
(iv) $\{A(0)\}$ for every atom A of σ.

Rules (iii) solve the frame problem by formalizing the commonsense law of inertia [17]; they are similar to the "frame default" from [16]. Rules (iv) express that both the initial values of fluents and the elementary actions to be executed can be chosen arbitrarily.

Recall that we agreed to identify truth-valued functions with sets of literals (Section 2.2). Using this convention, we can characterize the set of transitions of an arbitrary $\mathcal{B}$-description D in terms of models of *b2tv*(D) as follows:

Proposition. *For any sets s_0, s_1 of fluent literals, and any action a, $\langle s_0, a, s_1\rangle$ is a transition of $T(D)$ iff the set*

$$\{L(0) : L \in s_0 \cup a\} \cup \{L(1) : L \in s_1\}$$

is a model of b2tv(D).

This fact is a reformulation of Lemma 2 from [7], in view of the property of the transformation *tv2sn* noted in Section 3.1. It establishes a 1–1 correspondence between the transitions of D and the models of *b2tv*(D).

5.2 Translating Definite $\mathcal{C}$-Descriptions

This review of the syntax of $\mathcal{C}$ follows [7, Section 2.2.1]. An *action description in the language* $\mathcal{C}$, or *$\mathcal{C}$-description*, is a set of expressions of the two forms: *static laws*

$$\textbf{caused } F \textbf{ if } G, \tag{11}$$

where F and G are formulas that do not contain elementary actions, and *dynamic laws*

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H, \tag{12}$$

where F and G are formulas that do not contain elementary actions, and H is a formula. The semantics of the language (see, for instance, [7, Section 2.2.2]) defines, for every $\mathcal{C}$-description D, which transition system it represents.

A $\mathcal{C}$-description is *definite* if, in each of its laws (11), (12), the head F is a literal. For any definite $\mathcal{C}$-description D, the set of transitions of the corresponding system can be described by the program *c2tv*(D), defined as follows. Its signature σ_1 consists of the same symbols (10) as in the case of $\mathcal{B}$-descriptions. For any formula F of the signature σ, by $F(0)$ we will denote the formula of the signature σ_1 obtained from F by appending the string $'(0)'$ to each atom. For any formula F of the signature σ that does not contain elementary actions, by $F(1)$ we will denote the formula of the signature σ_1 obtained from F by appending the string $'(1)'$ to each atom. The rules of *c2tv*(D) are

(i) $F(t) \leftarrow : G(t)$, where $t = 0, 1$, for each static law (11) from D;
(ii) $F(1) \leftarrow : G(1) \wedge H(0)$ for each dynamic law (12) from D;
(iii) $\{A(0)\}$ for every atom A of σ.

The characterization of transitions given by the proposition from Section 5.1, with *b2tv* replaced by *c2tv*, holds for any definite $\mathcal{C}$-description D. This fact is a corollary to Proposition 2 from [8], in view of the property of the transformation *ct2tv* noted in Section 4 above. It establishes a 1–1 correspondence between the transitions of D and the models of *c2tv*(D).

If H in a dynamic law (12) is a conjunction of literals $L_1 \wedge \cdots \wedge L_n$ then the rule in clause (ii) of the definition of *c2tv* can be rewritten as

$$F(1) \leftarrow L_1(0), \ldots, L_n(0) : G(1),$$

and the models of the theory will remain the same.

6 Conclusion

We have seen that the language of two-valued programs is sufficiently rich for expressing the ASP solution to the frame problem that exploits the distinction between strong negation and negation as failure (Section 5.1), and that it can model the uses of ASP that avoid strong negation altogether (Section 3.2). There are also "mixed" representations, which express the falsity of some atoms explicitly, in terms of strong negation, and treat the falsity of other atoms in the spirit of an implicit closed world assumption. Such representations can be often expressed by two-valued programs as well.

Uses of ASP for which the language of two-valued programs is inadequate are relatively rare, but they do exist. Incomplete answer sets are essential for representing "weak exceptions" to defaults, as in [3, Example 3.2]: birds normally fly; wounded birds *may or may not* fly. Another example is given by the approach to conformant planning presented in [18]. The planner described in that paper operates with "partial states"—incomplete sets of literals that approximate states in the sense of Section 5. The difference between the applications of ASP that can be naturally represented by two-valued programs and the applications for which it is not the case is an important distinction between two kinds of answer set programs.

Acknowledgements

Thanks to Marc Denecker, Michael Gelfond, Joohyung Lee, Yuliya Lierler, and Fangkai Yang for useful discussions related to the topic of this note.

References

1 Marcello Balduccini and Michael Gelfond. Diagnostic reasoning with A-Prolog. *Theory and Practice of Logic Programming*, 3(4-5):425–461, 2003.

2 Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

3 Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994.

4 Michael Gelfond. Answer sets. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2008.

5 Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

6 Michael Gelfond and Vladimir Lifschitz. Action languages[1]. *Electronic Transactions on Artificial Intelligence*, 3:195–210, 1998.

7 Michael Gelfond and Vladimir Lifschitz. The common core of action languages $\mathcal{B}$ and $\mathcal{C}^2$. Unpublished draft, 2012.

8 Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 623–630. AAAI Press, 1998.

9 Vladimir Lifschitz. What is answer set programming? In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1594–1597. MIT Press, 2008.

[1] `http://www.ep.liu.se/ea/cis/1998/016/`
[2] `http://www.cs.utexas.edu/users/vl/papers/bc.pdf`

10 Vladimir Lifschitz and Hudson Turner. Representing transition systems by logic programs. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 92–106, 1999.

11 Victor Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

12 Norman McCain. *Causality in Commonsense Reasoning about Actions*[3]. PhD thesis, University of Texas at Austin, 1997.

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

14 Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

15 Ilkka Niemelä and Patrik Simons. Extending the Smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer, 2000.

16 Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

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

18 Phan Huy Tu, Tran Cao Son, Michael Gelfond, and Ricardo Morales. Approximation of action theories and its application to conformant planning. *Artificial Intelligence*, 175:79–119, 2011.

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