Representing Transition Systems by Logic Programs

Vladimir Lifschitz[1] and Hudson Turner[2]

[1] Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
`vl@cs.utexas.edu`
[2] Department of Computer Science
University of Minnesota at Duluth
Duluth, MN 55812, USA
`hudson@d.umn.edu`

Abstract. This paper continues the line of research on representing actions, on the automation of commonsense reasoning and on planning that deals with causal theories and with action language $\mathcal{C}$. We show here that many of the ideas developed in that work can be formulated in terms of logic programs under the answer set semantics, without mentioning causal theories. The translations from $\mathcal{C}$ into logic programming that we investigate serve as a basis for the use of systems for computing answer sets to reason about action domains described in $\mathcal{C}$ and to generate plans in such domains.

1 Introduction

This paper continues the line of research on representing actions, on the automation of commonsense reasoning and on planning described in [11], [7] and [12]. A large part of that work deals with a new nonmonotonic formalism—"causal theories." We show here that many of the ideas developed in those papers can be formulated also in terms of logic programs under the answer set semantics [5], without even mentioning causal theories.

Specifically, we investigate here translations from action language $\mathcal{C}$ into logic programming. These translations serve as a basis for the use of systems for computing answer sets, such as SMODELS [13][1] and DLV [3][2], for planning in action domains described in $\mathcal{C}$, as proposed in [9].[3] In [2] SMODELS is used to generate plans for action domains described in the language of STRIPS, which is not as expressive as $\mathcal{C}$.

One such translation can be obtained by composing the translation from $\mathcal{C}$ into the language of causal theories defined in [7] with the translation from

[1] `http://saturn.hut.fi/pub/smodels` .
[2] `http://www.dbai.tuwien.ac.at/proj/dlv` .
[3] See also `http://www.cs.utexas.edu/users/esra/experiments/experiments.html` .

causal theories into logic programs given by Proposition 6.1 from [10]. We call this translation *lpn*. Our basic translation *lp* is similar (and equivalent) to *lpn* but a little simpler, and our proof of the soundness of both translations is direct: it does not refer to causal theories.

A modification of the "literal completion" procedure from [11] allows us to describe answer sets for these translations by propositional formulas. This fact provides an alternative explanation for some of the computational procedures that CCALC[4] uses for temporal reasoning and planning. The new explanation is entirely in terms of logic programming; it does not appeal to any specialized logic of causal reasoning.

After a review of the syntax and semantics of $\mathcal{C}$ (Section 2) and of the concept of an answer set (Section 3), we define the basic translation from $\mathcal{C}$ into logic programming and state a theorem expressing its adequacy (Section 4). In Section 5 we show that the result of the basic translation can be simplified if the given domain description has a nontrivial "split mapping." Literal completion is discussed in Section 6. Proofs are postponed to Section 7.

2 Review of $\mathcal{C}$

This review of action language $\mathcal{C}$ follows [7] and [6].

Consider a set σ of propositional symbols partitioned into the *fluent names* σ^{fl} and the *elementary action names* σ^{act}. An *action* is an interpretation of σ^{act}. There are two kinds of *propositions* in $\mathcal{C}$: *static laws* of the form

$$\textbf{caused } F \textbf{ if } G \tag{1}$$

and *dynamic laws* of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } U\ , \tag{2}$$

where F, G are formulas of signature σ^{fl} and U is a formula of signature σ. In a proposition of either kind, the formula F is called the *head*. An *action description* is a set of propositions.

Consider an action description D. A *state* is an interpretation of σ^{fl} that satisfies $G \supset F$ for every static law (1) in D. A *transition* is any triple $\langle s, a, s' \rangle$ where s, s' are states and a is an action; s is the *initial* state of the transition, and s' is its *resulting* state. A formula F is *caused* in a transition $\langle s, a, s' \rangle$ if it is

(i) the head of a static law (1) from D such that s' satisfies G, or
(ii) the head of a dynamic law (2) from D such that s' satisfies G and $s \cup a$ satisfies U.

A transition $\langle s, a, s' \rangle$ is *causally explained* by D if its resulting state s' is the only interpretation of σ^{fl} that satisfies all formulas caused in this transition.

[4] `http://www.cs.utexas.edu/users/mccain/cc` .

The *transition system* described by an action description D is the directed graph which has the states of D as nodes, and which includes an edge from s to s' labeled a for every transition $\langle s, a, s' \rangle$ that is causally explained by D.

Consider two examples. The first describes the action of opening a spring-loaded door using the fluent name *Closed* and the elementary action name *OpenDoor*. In the notation introduced in [6, Section 6], this action description can be written as

$$\begin{array}{l} \textbf{default } \mathit{Closed}\ , \\ \mathit{OpenDoor} \textbf{ causes } \neg \mathit{Closed} \end{array} \tag{3}$$

which is an abbreviation for[5]

$$\begin{array}{l} \textbf{caused } \mathit{Closed} \textbf{ if } \mathit{Closed}\ , \\ \textbf{caused } \neg\mathit{Closed} \textbf{ if } \top \textbf{ after } \mathit{OpenDoor}\ . \end{array}$$

The transition system described by (3) has 2 states ($\overline{\mathit{Closed}}$ and *Closed*) and 4 causally explained transitions:

$$\begin{array}{l} \langle\ \overline{\mathit{Closed}},\ \overline{\mathit{OpenDoor}},\ \mathit{Closed}\ \rangle\ , \\ \langle\ \mathit{Closed},\ \overline{\mathit{OpenDoor}},\ \mathit{Closed}\ \rangle\ , \\ \langle\ \overline{\mathit{Closed}},\ \mathit{OpenDoor},\ \overline{\mathit{Closed}}\ \rangle\ , \\ \langle\ \mathit{Closed},\ \mathit{OpenDoor},\ \overline{\mathit{Closed}}\ \rangle\ . \end{array}$$

The first of these transitions shows that the door is spring-loaded: it closes by itself when we do nothing ($\overline{\mathit{OpenDoor}}$).

The other example describes the effect of putting an object in water. It involves the fluent names *InWater* and *Wet* and the elementary action name *PutInWater*. In abbreviated notation, its propositions are:

$$\begin{array}{l} \mathit{PutInWater} \textbf{ causes } \mathit{InWater}\ , \\ \textbf{caused } \mathit{Wet} \textbf{ if } \mathit{InWater}\ , \\ \textbf{inertial } \mathit{InWater}, \neg\mathit{InWater}, \mathit{Wet}, \neg\mathit{Wet}\ . \end{array} \tag{4}$$

(*InWater* is treated here as a direct effect of the action, and *Wet* is an indirect effect.) Written out in full, (4) becomes:

$$\begin{array}{l} \textbf{caused } \mathit{InWater} \textbf{ if } \top \textbf{ after } \mathit{PutInWater}\ , \\ \textbf{caused } \mathit{Wet} \textbf{ if } \mathit{InWater}\ , \\ \textbf{caused } \mathit{InWater} \textbf{ if } \mathit{InWater} \textbf{ after } \mathit{InWater}\ , \\ \textbf{caused } \neg\mathit{InWater} \textbf{ if } \neg\mathit{InWater} \textbf{ after } \neg\mathit{InWater}\ , \\ \textbf{caused } \mathit{Wet} \textbf{ if } \mathit{Wet} \textbf{ after } \mathit{Wet}\ , \\ \textbf{caused } \neg\mathit{Wet} \textbf{ if } \neg\mathit{Wet} \textbf{ after } \neg\mathit{Wet}\ . \end{array}$$

The corresponding transition system has 3 states

$$\overline{\mathit{InWater}}\ \overline{\mathit{Wet}},\ \ \overline{\mathit{InWater}}\ \mathit{Wet},\ \ \mathit{InWater}\ \mathit{Wet}$$

[5] We assume that the language contains the 0-place connectives $\top$ (true) and $\bot$ (false).

and 6 causally explained transitions:

$$\begin{array}{l}
\langle\ \overline{\mathit{InWater}}\ \overline{\mathit{Wet}},\ \overline{\mathit{PutInWater}},\ \overline{\mathit{InWater}}\ \overline{\mathit{Wet}}\ \rangle\ \ , \\
\langle\ \overline{\mathit{InWater}}\ \overline{\mathit{Wet}},\ \mathit{PutInWater},\ \mathit{InWater}\ \mathit{Wet}\ \rangle\ \ , \\
\langle\ \overline{\mathit{InWater}}\ \mathit{Wet},\ \overline{\mathit{PutInWater}},\ \overline{\mathit{InWater}}\ \mathit{Wet}\ \rangle\ \ , \\
\langle\ \overline{\mathit{InWater}}\ \mathit{Wet},\ \mathit{PutInWater},\ \mathit{InWater}\ \mathit{Wet}\ \rangle\ \ , \\
\langle\ \mathit{InWater}\ \mathit{Wet},\ \overline{\mathit{PutInWater}},\ \mathit{InWater}\ \mathit{Wet}\ \rangle\ \ , \\
\langle\ \mathit{InWater}\ \mathit{Wet},\ \mathit{PutInWater},\ \mathit{InWater}\ \mathit{Wet}\ \rangle\ \ .
\end{array}$$

The translations defined in this note are applicable to an action description D only if it satisfies the following condition: for all static laws (1) and dynamic laws (2) in D,

(i) F is a literal or the symbol $\bot$,
(ii) G and U are conjunctions of literals (possibly empty, understood as $\top$).

If this condition is satisfied, we will say that D is *definite*.[6] For instance, descriptions (3) and (4) are definite.

We will identify an intepretation I of a signature with the set of literals of that signature that are satisfied by I. If D is definite then the condition

$$\begin{array}{c} s' \text{ is the only interpretation of } \sigma^{fl} \text{ that satisfies} \\ \text{all formulas caused in } \langle s, a, s' \rangle \end{array}$$

in the definition of a causally explained transition can be equivalently replaced by

$$\text{the set of formulas caused in } \langle s, a, s' \rangle \text{ is } s'.$$

3 Review of the Answer Set Semantics

The definitions below diverge from [5] in how they treat constraints and inconsistent sets of literals.

Consider a set of propositional symbols, called *atoms*. A *literal* is an expression of the form B or $\neg B$, where B is an atom. A *rule element* is an expression of the form L or *not* L where L is a literal. The symbol $\neg$ is called *classical negation*, and the symbol *not* is *negation as failure*. A *rule* is a pair $\mathit{Head} \leftarrow \mathit{Body}$ where Head is a literal or the symbol $\bot$, and Body is a finite set of rule elements. Thus a rule has the form

$$\mathit{Head} \leftarrow L_1, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n \tag{5}$$

where $n \geq m \geq 0$; we drop $\{\ \}$ around the elements of the body. A rule (5) is a *constraint* if $\mathit{Head} = \bot$. A *program* is a set of rules.

The notion of an answer set is defined first for programs whose rules do not contain negation as failure. Let Π be such a program, and let X be a consistent

[6] Part (ii) of this definition is not essential: it can be dropped at the cost of making the translations slightly more complicated.

set of literals. We say that X is *closed* under Π if, for every rule $\mathit{Head} \leftarrow \mathit{Body}$ in Π, $\mathit{Head} \in X$ whenever $\mathit{Body} \subseteq X$. (For a constraint, this condition means that the body is not contained in X.) We say that X is an *answer set* for Π if X is minimal among the sets closed under Π. It is clear that a program without negation as failure can have at most one answer set.

To extend this definition to arbitrary programs, take any program Π, and let X be a consistent set of literals. The *reduct* Π^X of Π relative to X is the set of rules

$$\mathit{Head} \leftarrow L_1, \dots, L_m$$

for all rules (5) in Π such that $L_{m+1}, \dots, L_n \notin X$. Thus Π^X is a program without negation as failure. We say that X is an *answer set* for Π if X is an answer set for Π^X. (Note that, according to these definitions, all answer sets are consistent.)

A set X of literals is *complete* if for every atom B, $B \in X$ or $\neg B \in X$. The use of the answer set semantics in this paper is similar to its use in [14] in that we will be interested in complete answer sets only. It is clear that the incomplete answer sets for any program can be eliminated by adding the constraints

$$\leftarrow \mathit{not}\ B, \mathit{not}\ \neg B \tag{6}$$

for all atoms B.

4 Basic Translation

Let D be a definite action description. We will define, for every positive integer T, a logic program $\mathit{lp}_T(D)$ whose answer sets correspond to "histories"—paths of length T in the transition system described by D.

The language of $\mathit{lp}_T(D)$ has atoms of two kinds:

(i) *fluent atoms*—the fluent names of D followed by (t) where $t = 0, \dots, T$, and
(ii) *action atoms*—the action names of D followed by (t) where $t = 0, \dots, T-1$.

Thus every literal in this language ends with (t) for some natural number t. This number will be called the *time stamp* of the literal.

Program $\mathit{lp}_T(D)$ consists of the following rules:

(i) for every static law

$$\textbf{caused } F \textbf{ if } L_1 \wedge \cdots \wedge L_m$$

in D, the rules

$$F(t) \leftarrow \mathit{not}\ \overline{L_1(t)}, \dots, \mathit{not}\ \overline{L_m(t)} \tag{7}$$

for all $t = 0, \dots, T$ (we understand $F(t)$ as $\bot$ if F is $\bot$; $\overline{L}$ stands for the literal complementary to L),

(ii) for every dynamic law

$$\textbf{caused } F \textbf{ if } L_1 \wedge \cdots \wedge L_m \textbf{ after } L_{m+1} \wedge \cdots \wedge L_n$$

in D, the rules

$$F(t+1) \leftarrow \ \mathit{not}\ \overline{L_1(t+1)}, \ldots, \mathit{not}\ \overline{L_m(t+1)}, L_{m+1}(t), \ldots, L_n(t) \tag{8}$$

for all $t = 0, \ldots, T-1$,

(iii) the rules

$$\begin{array}{l} \neg B \leftarrow \mathit{not}\ B \ \ , \\ B \leftarrow \mathit{not}\ \neg B \end{array}$$

where B is a fluent atom with the time stamp 0 or an action atom.

For instance, the translation of (3) consists of all rules of the forms

$$\begin{array}{l} \mathit{Closed}(t) \leftarrow \mathit{not}\ \neg \mathit{Closed}(t) \ \ , \\ \neg \mathit{Closed}(t+1) \leftarrow \mathit{OpenDoor}(t) \ \ , \\ \mathit{Closed}(0) \leftarrow \mathit{not}\ \neg \mathit{Closed}(0) \ \ , \\ \neg \mathit{Closed}(0) \leftarrow \mathit{not}\ \mathit{Closed}(0) \ \ , \\ \mathit{OpenDoor}(t) \leftarrow \mathit{not}\ \neg \mathit{OpenDoor}(t) \ \ , \\ \neg \mathit{OpenDoor}(t) \leftarrow \mathit{not}\ \mathit{OpenDoor}(t) \ \ . \end{array} \tag{9}$$

The translation of (4) is

$$\begin{array}{l} \mathit{InWater}(t+1) \leftarrow \mathit{PutInWater}(t) \ \ , \\ \mathit{Wet}(t) \leftarrow \mathit{not}\ \neg \mathit{InWater}(t) \ \ , \\ \mathit{InWater}(t+1) \leftarrow \mathit{not}\ \neg \mathit{InWater}(t+1), \mathit{InWater}(t) \ \ , \\ \neg \mathit{InWater}(t+1) \leftarrow \mathit{not}\ \mathit{InWater}(t+1), \neg \mathit{InWater}(t) \ \ , \\ \mathit{Wet}(t+1) \leftarrow \mathit{not}\ \neg \mathit{Wet}(t+1), \mathit{Wet}(t) \ \ , \\ \neg \mathit{Wet}(t+1) \leftarrow \mathit{not}\ \mathit{Wet}(t+1), \neg \mathit{Wet}(t) \ \ , \\ \mathit{InWater}(0) \leftarrow \mathit{not}\ \neg \mathit{InWater}(0) \ \ , \\ \neg \mathit{InWater}(0) \leftarrow \mathit{not}\ \mathit{InWater}(0) \ \ , \\ \mathit{Wet}(0) \leftarrow \mathit{not}\ \neg \mathit{Wet}(0) \ \ , \\ \neg \mathit{Wet}(0) \leftarrow \mathit{not}\ \mathit{Wet}(0) \ \ , \\ \mathit{PutInWater}(t) \leftarrow \mathit{not}\ \neg \mathit{PutInWater}(t) \ \ , \\ \neg \mathit{PutInWater}(t) \leftarrow \mathit{not}\ \mathit{PutInWater}(t) \ \ . \end{array} \tag{10}$$

Proposition 1. *A complete set X of literals is an answer set for $lp_T(D)$ iff it has the form*

$$\left[\bigcup_{t=0}^{T-1} \{L(t) \ : \ L \in s_t \cup a_t\}\right] \cup \{L(T) \ : \ L \in s_T\}$$

for some path $\langle s_0, a_0, s_1, \ldots, s_{T-1}, a_{T-1}, s_T \rangle$ in the transition system described by D.

As discussed at the end of Section 3, the restriction to complete sets can be dropped if we extend the program by constraints (6). In the case of $lp_T(D)$, it is sufficient to add these constraints for the fluent atoms with nonzero time stamps.

The case of $T = 1$ deserves a special mention:

Corollary 1. *A complete set X of literals is an answer set for $lp_1(D)$ iff it has the form*

$$\{L(0) \,:\, L \in s \cup a\} \cup \{L(1) \,:\, L \in s'\}$$

for some transition $\langle s, a, s' \rangle$ causally explained by D.

5 Simplifying the Basic Translation

A *split mapping* for a definite action description D is a function λ from fluent literals[7] to ordinals such that, for every static law

$$\textbf{caused } F \textbf{ if } L_1 \wedge \cdots \wedge L_m$$

and every dynamic law

$$\textbf{caused } F \textbf{ if } L_1 \wedge \cdots \wedge L_m \textbf{ after } L_{m+1} \wedge \cdots \wedge L_n$$

in D such that F is not $\bot$,

$$\lambda(L_1), \ldots, \lambda(L_m) \leq \lambda(F) \ .$$

If λ is a split mapping for D, then we can sometimes eliminate some occurrences of negation as failure in the translation of static and dynamic laws: by replacing *not* $\overline{L_i(t)}$ $(1 \leq i \leq m)$ with $L_i(t)$ whenever $\lambda(F) > \lambda(L_i)$.

Proposition 2. *If λ is a split mapping for a definite action description D, then in rules (7) and (8) of $lp_T(D)$ with fluent literal heads we can replace any expressions of the form not $\overline{L_i(t)}$ $(1 \leq i \leq m)$ such that $\lambda(F) > \lambda(L_i)$ with $L_i(t)$ without affecting the complete answer sets. Similarly, in rules (7) and (8) of $lp_T(D)$ with head $\bot$ we can replace any expressions of the form not $\overline{L_i(t)}$ $(1 \leq i \leq m)$ with $L_i(t)$.*

For instance, a split mapping for (4) can be defined by

$$\lambda(\mathit{InWater}) = \lambda(\neg \mathit{InWater}) = 0,\ \lambda(\mathit{Wet}) = \lambda(\neg \mathit{Wet}) = 1.$$

It follows that the second rule

$$\mathit{Wet}(t) \leftarrow \mathit{not}\ \neg \mathit{InWater}(t)$$

[7] A *fluent literal* is a literal containing a fluent name; an *action literal* is a literal containing an action name. We apply this terminology both to the language of action descriptions and to the language of their translations into logic programming.

in the translation (10) of (4) can be equivalently replaced by

$$Wet(t) \leftarrow InWater(t) \ .$$

A result analagous to Proposition 2, applied to an extension of causal theories, appears as Theorem 5.15 in [15]. There the issue is when a formula $\mathsf{C}F \supset \mathsf{C}G$, read "$G$ is caused whenever F is caused" can be replaced by $F \supset \mathsf{C}G$ without affecting the "causally explained interpretations." The first of these formulas corresponds to the treatment of static causal laws in the language $\mathcal{B}$ from [6] and in the translation of static causal laws into logic programming from [14].

6 Literal Completion

Programs $lp_T(D)$ are "positive-order-consistent," or "tight." This concept is defined in [4] for the special case of programs without classical negation; for tight programs without classical negation, the answer set semantics is shown in that paper to be equivalent to the completion semantics defined in [1]. According to Proposition 3 below, *complete* answer sets for a finite tight program can be characterized by the propositional formulas generated from the program by "literal completion." This process, similar to Clark's completion, is defined in [11] for causal theories, and here we show how this idea applies to tight programs.

A *level mapping* is a function from literals to ordinals. (For finite programs, we can assume, without the loss of generality, that the values of a level mapping are nonnegative integers.) A program Π is *tight* if there exists a level mapping λ such that, for every rule (5) in Π that is not a constraint,

$$\lambda(L_1), \dots, \lambda(L_m) < \lambda(Head) \ .$$

Note that this condition does not impose any restriction on the rule elements that include negation as failure.

Consider a finite program Π. If H is a literal or the symbol $\bot$, by $Bodies(H)$ we denote the set of the bodies of all rules in Π whose head is H. For any finite set $Body$ of rule elements, the propositional formula $pf(Body)$ is defined by the equation

$$pf(L_1, \dots, L_m, not\ L_{m+1}, \dots, not\ L_n) = L_1 \wedge \dots \wedge L_m \wedge \overline{L_{m+1}} \wedge \dots \wedge \overline{L_n} \ .$$

The *literal completion* of Π consists of the formulas

$$H \equiv \bigvee_{Body \in Bodies(H)} pf(Body)$$

for all H. (The range of values of H includes all literals of the underlying language, even those that do not occur in Π, and the symbol $\bot$.)

Proposition 3. *For any finite tight program Π and any complete set X of literals, X is an answer set for Π iff X is an interpretation satisfying the literal completion of Π.*

Consider, for instance, the program

$$\begin{array}{l} p \leftarrow \mathit{not}\ \neg p\ \ , \\ \neg p \leftarrow \mathit{not}\ p\ \ , \\ q \leftarrow \mathit{not}\ \neg q\ \ , \\ \neg q \leftarrow \mathit{not}\ q\ \ , \\ r \leftarrow q\ \ . \end{array} \tag{11}$$

This program has 4 answer sets:

$$\{p, q, r\},\ \{p, \neg q\},\ \{\neg p, q, r\},\ \{\neg p, \neg q\};$$

two of them are complete. The literal completion for (11) consists of the formulas

$$\begin{array}{l} p \equiv p,\ \neg p \equiv \neg p, \\ q \equiv q,\ \neg q \equiv \neg q, \\ r \equiv q,\ \neg r \equiv \bot. \end{array}$$

This set of formulas is equivalent to $q \wedge r$. Consequently, it is satisfied by two interpretations, $\{p, q, r\}$ and $\{\neg p, q, r\}$. In accordance with Proposition 3, these are the same as the complete answer sets for (11).

It is clear that $lp_T(D)$ is tight: take $\lambda(L)$ to be the time stamp of L. Consequently, the complete answer sets for this program can be characterized as the models of its literal completion. This fact is important for two reasons.

First, it shows that one does not need a system for computing answer sets to plan in a domain described in $\mathcal{C}$; a propositional solver, such as SATO [16], will suffice. This is, in fact, how CCALC operates when the available actions are described by a definite action description in $\mathcal{C}$.

Second, this fact can be used to prove that some modifications of $lp_T(D)$, for finite D, are essentially equivalent to $lp_T(D)$, in the sense that they have the same complete answer sets as $lp_T(D)$. Consider, for instance, a program obtained from $lp_T(D)$ by replacing some of the rule elements $L_i(t)$ $(m < i \leq n)$ in the translations (8) of dynamic laws by $\mathit{not}\ \overline{L_i(t)}$. The result is a finite tight program with the same literal completion; Proposition 3 implies that it has the same complete answer sets as $lp_T(D)$.

In particular, this replacement can be applied to *every* rule element in $lp_T(D)$ that does not contain negation as failure. We will denote the result by $lpn_T(D)$. For instance, if D is (3) then $lpn_T(D)$ is

$$\begin{array}{l} \mathit{Closed}(t) \leftarrow \mathit{not}\ \neg \mathit{Closed}(t)\ \ , \\ \neg \mathit{Closed}(t+1) \leftarrow \mathit{not}\ \neg \mathit{OpenDoor}(t)\ \ , \\ \mathit{Closed}(0) \leftarrow \mathit{not}\ \neg \mathit{Closed}(0)\ \ , \\ \neg \mathit{Closed}(0) \leftarrow \mathit{not}\ \mathit{Closed}(0)\ \ , \\ \mathit{OpenDoor}(t) \leftarrow \mathit{not}\ \neg \mathit{OpenDoor}(t)\ \ , \\ \neg \mathit{OpenDoor}(t) \leftarrow \mathit{not}\ \mathit{OpenDoor}(t)\ \ . \end{array}$$

(Program $lpn_T(D)$ is, in fact, the alternative translation from $\mathcal{C}$ into logic programming mentioned in the introduction.) Proposition 3 shows that $lp_T(D)$ and $lpn_T(D)$ have the same complete answer sets, for finite D.

In the next section, we prove Proposition 3 on the basis of a similar result applicable even when D is not finite. The same result plays a role in the proofs of Propositions 1 and 2.

7 Proofs

We begin with definitions and a theorem from [8] concerning tight programs. Take any program Π, and consistent set X of literals. We say that X is *closed under* Π if, for every rule

$$Head \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \tag{12}$$

$Head \in X$ whenever $L_1, \ldots, L_m \in X$ and $L_{m+1}, \ldots, L_n \notin X$. We say that X is *supported by* Π if, for every $L \in X$, there is a rule (12) in Π s.t. $Head = L$, $L_1, \ldots, L_m \in X$ and $L_{m+1}, \ldots, L_n \notin X$.

Proposition 4 ([4, 8]). *For any tight program Π, a consistent set X of literals is an answer set for Π iff X is closed under and supported by Π.*

Since Proposition 4 appears in [4] in a less general form, and in [8] without proof, we include a proof at the end of this section.

7.1 Proof of Proposition 1

Proposition 1. *A complete set X of literals is an answer set for $lp_T(D)$ iff it has the form*

$$\left[\bigcup_{t=0}^{T-1} \{L(t) \,:\, L \in s_t \cup a_t\}\right] \cup \{L(T) \,:\, L \in s_T\} \tag{13}$$

for some path $\langle s_0, a_0, s_1, \ldots, s_{T-1}, a_{T-1}, s_T \rangle$ in the transition system described by D.

To prove this fact, we will establish three lemmas about the modification lpn_T of the translation lp_T defined in Section 6. Recall that, in the modified translation, a dynamic law

$$\textbf{caused}\ F\ \textbf{if}\ L_1 \wedge \cdots \wedge L_m\ \textbf{after}\ L_{m+1} \wedge \cdots \wedge L_n$$

is represented by the rules

$$F(t+1) \leftarrow \ not\ \overline{L_1(t+1)}, \ldots, not\ \overline{L_m(t+1)}, not\ \overline{L_{m+1}(t)}, \ldots, not\ \overline{L_n(t)}$$

instead of

$$F(t+1) \leftarrow \ not\ \overline{L_1(t+1)}, \ldots, not\ \overline{L_m(t+1)}, L_{m+1}(t), \ldots, L_n(t)$$

$(t = 0, \ldots, T-1)$. The first lemma shows that programs $lp_T(D)$ and $lpn_T(D)$ have the same complete answer sets. The third lemma differs from Proposition 1 only in that $lp_T(D)$ in its statement is replaced by $lpn_T(D)$.

Lemma 1. *Programs $lp_T(D)$ and $lpn_T(D)$ have the same complete answer sets.*

Proof. Both programs are tight. It is easy to verify that a complete, consistent set of literals is closed under one iff it is closed under the other. It is also easy to verify that they have the same complete supported sets. So, by Proposition 4, they have the same complete answer sets.

Lemma 2. *If X has the form* (13)*, for some states $s_0, s_1, \ldots, s_{T-1}, s_T$ and some interpretations $a_0, \ldots, a_{T-1}$ of σ^{act}, then for every fluent literal L and $t \in \{0, \ldots, T-1\}$, $L(t+1) \leftarrow \in lpn_T(D)^X$ iff L is caused in $\langle s_t, a, s_{t+1} \rangle$.*

Proof. Assume that $L(t+1) \leftarrow \in lpn_T(D)^X$. At least one of the following two cases holds.

Case 1: D contains a static law

$$\textbf{caused } L \textbf{ if } L_1 \wedge \cdots \wedge L_m$$

such that $\overline{L_1(t+1)}, \ldots, \overline{L_m(t+1)} \notin X$. Then $L_1(t+1), \ldots, L_m(t+1) \in X$, and consequently $L_1, \ldots, L_m \in s_{t+1}$. It follows that L is caused in $\langle s_t, a, s_{t+1} \rangle$.

Case 2: D contains a dynamic law

$$\textbf{caused } L \textbf{ if } L_1 \wedge \ldots \wedge L_m \textbf{ after } L_{m+1} \wedge \ldots \wedge L_n$$

such that

$$\overline{L_1(t+1)}, \ldots, \overline{L_m(t+1)}, \overline{L_{m+1}(t)}, \ldots, \overline{L_n(t)} \notin X.$$

Then

$$L_1(t+1), \ldots, L_m(t+1), L_{m+1}(t), \ldots, L_n(t) \in X,$$

and consequently $L_1, \ldots, L_m \in s_{t+1}$ and $L_{m+1}, \ldots, L_n \in s_t$. It follows that L is caused in $\langle s_t, a, s_{t+1} \rangle$.

A similar argument shows that if $L(t+1) \leftarrow \notin lpn_T(D)^X$, then L is not caused in $\langle s_t, a, s_{t+1} \rangle$.

Lemma 3. *A complete set X of literals is an answer set for $lpn_T(D)$ iff it has form* (13) *for some path $\langle s_0, a_0, s_1, \ldots, s_{T-1}, a_{T-1}, s_T \rangle$ in the transition system described by D.*

Proof. Left-to-right: Assume that X is a complete answer set for $lpn_T(D)$. Clearly X has the form (13), for some interpretations $s_0, s_1, \ldots, s_{T-1}, s_T$ of σ^{act} and $a_0, \ldots, a_{T-1}$ of σ^{act}. Since X is closed under $lpn_T(D)^X$, we know that for every rule (7) obtained from a static causal law **caused** F **if** $L_1 \wedge \cdots \wedge L_m$, if $\overline{L_1(t)}, \ldots, \overline{L_m(t)} \notin X$, then $F(t) \in X$. Hence, for all $t \in \{0, \ldots, T\}$, s_t satisfies $L_1 \wedge \cdots \wedge L_m \supset F$, for every static causal law **caused** F **if** $L_1 \wedge \cdots \wedge L_m$. That is, each s_t is a state. It remains to show that for each $t \in \{0, \ldots, T-1\}$, the set of formulas caused in $\langle s_t, a, s_{t+1} \rangle$ is s_{t+1}. This follows easily from Lemma 2.

Right-to-left: Assume that $\langle s_0, a_0, s_1, \ldots, s_{T-1}, a_{T-1}, s_T \rangle$ is a path in the transition system described by D, and let X be the associated complete set of literals of form (13). We complete the proof by showing that $L(t) \in X$ iff $L(t) \leftarrow\ \in \mathit{lpn}_T(D)^X$. Consider three cases.

Case 1: $L(t)$ is an action literal. The claim is trivial in this case, since in $\mathit{lpn}_T(D)$ such literals appear in the heads only of rules obtained in clause (iii) of the translation.

Case 2: $L(t)$ is a fluent literal, and $t = 0$. It is clear that if $L(0) \in X$, then $L(0) \leftarrow\ \in \mathit{lpn}_T(D)^X$, because of the rules obtained in clause (iii) of the translation. Assume $L(0) \notin X$. Then, with regard to the rule $L(0) \leftarrow \mathit{not}\ \overline{L(0)}$ obtained by clause (iii) of the translation, notice that $\overline{L(0)} \in X$. All other rules in $\mathit{lpn}_T(D)$ with $L(0)$ in the head have the form $L(0) \leftarrow \mathit{not}\ \overline{L_1}(0), \ldots, \mathit{not}\ \overline{L_m}(0)$ for some static causal law **caused** L **if** $L_1 \wedge \cdots \wedge L_m$, and since s_0 is a state to which L does not belong, we can conclude that at least one of $\overline{L_1}, \ldots, \overline{L_m}$ belongs to s_0. Hence, at least one of $\overline{L_1}(0), \ldots, \overline{L_m}(0)$ belongs to X. Consequently, $L(0) \leftarrow\ \notin \mathit{lpn}_T(D)^X$.

Case 3: $L(t)$ is a fluent literal, and $t \neq 0$. The claim in this case follows from Lemma 2.

7.2 Proof of Proposition 2

Proposition 2. *If λ is a split mapping for a definite action description D, then in rules (7) and (8) of $\mathit{lp}_T(D)$ with fluent literal heads we can replace any expressions of the form not $\overline{L_i}(t)$ $(1 \leq i \leq m)$ such that $\lambda(F) > \lambda(L_i)$ with $L_i(t)$ without affecting the complete answer sets. Similarly, in rules (7) and (8) of $\mathit{lp}_T(D)$ with head $\bot$ we can replace any expressions of the form not $\overline{L_i}(t)$ $(1 \leq i \leq m)$ with $L_i(t)$.*

Proof. Let λ be the split mapping for D, and let Π be the program obtained from $\mathit{lp}_T(D)$ by replacing some rule elements *not* $\overline{L_i}(t)$ with $L_i(t)$. It is easy to verify that the same complete, consistent sets of literals are closed under and supported by the two programs. As previously observed, $\mathit{lp}_T(D)$ is tight, so we can conclude by Proposition 4 that the two programs have the same complete answer sets, if we can show that Π is tight also. We do this by constructing a suitable level mapping λ^*.

Take

$$\alpha = \mathit{sup}\ \{\lambda(L) : L \text{ is a fluent literal}\}\ .$$

For all fluent literals L and $t \in \{0, \ldots, T\}$, let

$$\lambda^*(L(t)) = (\alpha + 1) \cdot t + \lambda(L)\ .$$

For all action literals L and $t \in \{0, \ldots, T-1\}$, we define $\lambda^*(L(t)) = 0$.

First observe that for any fluent literals L, L', and any $t \in \{0, \ldots, T-1\}$,

$$\lambda^*(L(t)) < \lambda^*(L'(t+1))\ ,$$

since

$$\begin{aligned}(\alpha+1)\cdot t+\lambda(L) &< (\alpha+1)\cdot t+(\alpha+1)\\ &= (\alpha+1)\cdot t+(\alpha+1)\cdot 1\\ &= (\alpha+1)\cdot(t+1)\\ &\leq (\alpha+1)\cdot(t+1)+\lambda(L')\ .\end{aligned}$$

(The first step uses the fact that $\lambda(L) < \alpha+1$, along with the right monotonicity of ordinal addition. The third step uses the fact that ordinal multiplication distributes from the left over addition.) Hence, level mapping λ^* establishes that $lp_T(D)$ itself is tight. It remains to show that the allowed replacements of rule elements preserve tightness.

Consider any rule in $lp_T(D)$ with a fluent literal head $L(t)$ in which a rule element $not\ \overline{L_i(t)}$ has been replaced with $L_i(t)$ in Π. In this case, we know that $\lambda(L_i) < \lambda(L)$, and we complete the proof by observing that, consequently,

$$\lambda^*(L_i(t)) = (\alpha+1)\cdot t+\lambda(L_i) < (\alpha+1)\cdot t+\lambda(L) = \lambda^*(L(t))$$

(again by the right monotonicity of ordinal addition).

7.3 Proof of Proposition 3

Proposition 3. *For any finite tight program Π and any complete set X of literals, X is an answer set for Π iff X is an interpretation satisfying the literal completion of Π.*

Proof. Assume X is a complete answer set for Π. Since Π is tight, we know by Proposition 4 that X is closed under and supported by Π. Let H be any literal or $\bot$. We must show that X satisfies

$$H \equiv \bigvee_{Body \in Bodies(H)} pf(Body)\ .$$

Case 1: $H \in X$. Since X is supported by Π, there is at least one rule

$$H \leftarrow L_1,\ldots,L_m, not\ L_{m+1},\ldots, not\ L_n$$

in Π such that $L_1,\ldots,L_m \in X$ and $L_{m+1},\ldots,L_n \notin X$. It follows that X satisfies the corresponding element of $Bodies(H)$.

Case 2: $H \notin X$. Since X is closed under Π, we know that for every rule

$$H \leftarrow L_1,\ldots,L_m, not\ L_{m+1},\ldots, not\ L_n$$

in Π either $\{L_1,\ldots,L_m\} \not\subseteq X$ or $\{L_{m+1},\ldots,L_n\} \cap X \neq \emptyset$. It follows that X satisfies no element of $Bodies(H)$.

Proof in the other direction is similar.

It may be worth noting that Proposition 3 holds even when Π is not finite, as long as there are finitely many rules in Π with any given head (so that the literal completion can be defined). Moreover, even this restriction can be dropped in the case of constraints (rules with head $\bot$), if we modify the definition of literal completion slightly, so that H ranges only over literals, and we add to the resulting propositional theory the formula $\neg pf(Body)$ for each constraint $\bot \leftarrow Body$ in Π.

7.4 Proof of Proposition 4

Proposition 4. *For any tight program Π, a consistent set X of literals is an answer set for Π iff X is closed under and supported by Π.*

Lemma 4. *For any tight program Π without negation as failure, and consistent set X of literals, if X is closed under and supported by Π, then X is an answer set for Π.*

Proof. We need to show that X is minimal among sets closed under Π. Suppose otherwise; let Y be a proper subset of X that is also closed under Π. Let λ be a level mapping establishing that Π is tight. Choose a literal $L \in X \setminus Y$ such that $\lambda(L)$ is minimal. Since X is supported by Π, there is a rule

$$L \leftarrow L_1, \ldots, L_m$$

in Pi such that $L_1, \ldots, L_m \in X$. Since Π is tight, $\lambda(L_1), \ldots, \lambda(L_m) < \lambda(L)$. Hence, by choice of L, we can conclude that $L_1, \ldots, L_m \in Y$, which shows that Y is not closed under Π, contrary to the choice of Y.

Proof of Proposition 4: The left-to-right direction is straightforward, and does not rely on tightness. For the other direction, assume X is closed under and supported by Π. It follows that X is closed under and supported by Π^X. Since Π is tight, so is Π^X. Hence, by Lemma 4, X is an answer set for Π^X, and, consequently, an answer set for Π.

Acknowledgements

Thanks to Esra Erdem for comments on a draft of this note. Some of these results were presented by the first author in his seminar on planning at the University of Texas in the Spring semester of 1999, and he is grateful to the participants of the seminar for interesting discussion. His work was partially supported by National Science Foundation under grant IRI-9732744. The second author is partially supported by University of Minnesota Grant-in-Aid of Research, Artistry & Scholarship #17831.

References

1. Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.
2. Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in non-monotonic logic programs. In *Proc. European Conf. on Planning 1997*, pages 169–181, 1997.
3. Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR system DLV: Progress report, comparisons and benchmarks. In Anthony Cohn, Lenhart Schubert, and Stuart Shapiro, editors, *Proc. Sixth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 406–417, 1998.
4. François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.
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. *Electronic Transactions on AI*, 3, 1998. Available at `http://www.ep.liu.se/ea/cis/1998/016/`.
7. Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630, 1998.
8. Vladimir Lifschitz. Foundations of logic programming. In *Principles of Knowledge Representation*, pages 69–127. CSLI Publications, 1996.
9. Vladimir Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.
10. Norman McCain. *Causality in Commonsense Reasoning about Actions.* PhD thesis, University of Texas at Austin, 1997.
11. Norman McCain and Hudson Turner. Causal theories of action and change. In *Proc. AAAI-97*, pages 460–465, 1997.
12. Norman McCain and Hudson Turner. Satisfiability planning with causal theories. In Anthony Cohn, Lenhart Schubert, and Stuart Shapiro, editors, *Proc. Sixth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 212–223, 1998.
13. Ilkka Niemelä and Patrik Simons. Efficient implementation of the well-founded and stable model semantics. In *Proc. Joint Int'l Conf. and Symp. on Logic Programming*, pages 289–303, 1996.
14. Hudson Turner. Representing actions in logic programs and default theories: a situation calculus approach. *Journal of Logic Programming*, 31:245–298, 1997.
15. Hudson Turner. *Causal Action Theories and Satisfiability Planning.* PhD thesis, University of Texas at Austin, 1998.
16. Hantao Zhang. An efficient propositional prover. In *Proc. CADE-97*, 1997.