5-2012

Representing First-Order Causal Theories by Logic Programs

Paolo Ferrarris
Google, Inc.

Joohyung Lee
Arizona State University

Yuliya Lierler
University of Nebraska at Omaha, ylierler@unomaha.edu

Vladimir Lifschitz
University of Kentucky

Fangkai Yang
University of Texas at Austin

Representing First-Order Causal Theories by Logic Programs

Paolo Ferraris
Google, Inc.
(*e-mail:* `otto@cs.utexas.edu`)

Joohyung Lee
School of Computing, Informatics and Decision Systems Engineering,
Arizona State University
(*e-mail:* `joolee@asu.edu`)

Yuliya Lierler
Computer Science Department, University of Kentucky
(*e-mail:* `yuliya@cs.utexas.edu`)

Vladimir Lifschitz and Fangkai Yang
Department of Computer Science, University of Texas at Austin
(*e-mail:* `{vl,fkyang}@cs.utexas.edu`)

submitted Aug, 16, 2010; revised Feb, 21, 2011; accepted Mar, 16, 2011

Abstract

Nonmonotonic causal logic, introduced by Norman McCain and Hudson Turner, became a basis for the semantics of several expressive action languages. McCain's embedding of definite propositional causal theories into logic programming paved the way to the use of answer set solvers for answering queries about actions described in such languages. In this paper we extend this embedding to nondefinite theories and to first-order causal logic.

KEYWORDS: reasoning about actions, nonmonotonic causal logic, answer set programming

1 Introduction

Propositional nonmonotonic causal logic (McCain and Turner 1997) and its generalizations became a basis for the semantics of several expressive action languages (Giunchiglia and Lifschitz 1998; Giunchiglia et al. 2004; Lifschitz and Ren 2006; Lifschitz and Ren 2007; Ren 2009). The Causal Calculator (CCALC)[1] is a partial

[1] `http://www.cs.utexas.edu/users/tag/ccalc/`

implementation of this logic that allows us to automate some kinds of reasoning and planning in action domains described in such languages. It has been used to solve several challenging commonsense reasoning problems, including problems of nontrivial size (Akman et al. 2004), to provide a group of robots with high-level reasoning (Caldiran et al. 2009), to give executable specifications of norm-governed computational societies (Artikis et al. 2009), and to automate the analysis of business processes under authorization constraints (Armando et al. 2009).

An important theorem due to Norman McCain (McCain 1997, Proposition 6.7) shows how to embed a fragment of propositional causal logic into the language of logic programming under the answer set semantics (Gelfond and Lifschitz 1991). This result, reviewed below, paved the way to the development of an attractive alternative to CCALC—the software system COALA (Gebser et al. 2010) that uses answer set programming (Marek and Truszczyński 1999; Niemelä 1999; Lifschitz 2008) for answering queries about actions described in causal logic.

A causal theory in the sense of (McCain and Turner 1997) is a set of "causal rules" of the form $F \Leftarrow G$, where F and G are propositional formulas (the *head* and the *body* of the rule). The rule reads "F is caused if G is true." Distinguishing between being true and having a cause turned out to be essential for the study of commonsense reasoning. The assertion "if the light is on at time 0 and you toggle the switch then the light will be off at time 1" can be written as an implication:

$$on_0 \wedge toggle \rightarrow \neg on_1.$$

In causal logic, on the other hand, we can express that under the same assumption *there is a cause* for the light to be off at time 1:

$$\neg on_1 \Leftarrow on_0 \wedge toggle.$$

(Performing the toggle action is the cause.) McCain and Turner showed that distinctions like this help us solve the frame problem (see Example 5 in Section 5.2) and overcome other difficulties arising in the theory of reasoning about actions.

The semantics of theories of this kind defines when a propositional interpretation (truth assignment) is a model of the given theory (is "causally explained" by the theory, in the terminology of McCain and Turner). We do not reproduce the definition here, because a more general semantics is described below in Section 3. But here is an example: the causal theory

$$\begin{aligned} p &\Leftarrow \neg q \\ \neg q &\Leftarrow p \end{aligned} \tag{1}$$

has one model, according to the semantics from (McCain and Turner 1997). In this model, p is true and q is false. (Since the bodies of both rules are true in this model, both rules "fire"; consequently the heads of the rules are "caused"; consequently the truth values of both atoms are "causally explained." This will be discussed formally in Section 3.)

McCain's translation is applicable to a propositional causal theory T if the head of each rule of T is a literal, and the body is a conjunction of literals:

$$L \Leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \neg A_n. \tag{2}$$

The corresponding logic program consists of the logic programming rules

$$L \leftarrow \mathit{not}\ \neg A_1, \ldots, \mathit{not}\ \neg A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \tag{3}$$

for all rules (2) of T. This program involves two kinds of negation: negation as failure (*not*) and strong, or classical, negation ($\neg$). According to Proposition 6.7 from (McCain 1997), complete answer sets of this logic program are identical to the models of T. (A set of literals is *complete* if it contains exactly one member of each complementary pair of literals $A, \neg A$. We identify a complete set of literals with the corresponding truth assignment.)

For instance, McCain's translation turns causal theory (1) into

$$\begin{array}{rl} p & \leftarrow \mathit{not}\ q \\ \neg q & \leftarrow \mathit{not}\ \neg p. \end{array} \tag{4}$$

The only answer set of this program is $\{p, \neg q\}$. It is complete, and it corresponds to the model of causal theory (1).

In this paper we generalize McCain's translation in several ways. First, we discard the requirement that the bodies of the given causal rules be conjunctions of literals. Second, instead of requiring that the head of each causal rule be a literal, we allow the heads to be disjunctions of literals. In this more general setting, the logic program corresponding to the given causal theory becomes disjunctive as well.

Third, we study causal rules with heads of the form $L_1 \leftrightarrow L_2$, where L_1 and L_2 are literals. Such a rule says that there is a cause for L_1 and L_2 to be equivalent ("synonymous") under some condition, expressed by the body of the rule. Synonymity rules play an important role in the theory of commonsense reasoning in view of the fact that humans often explain the meaning of words by referring to their synonyms. A synonymity rule

$$L_1 \leftrightarrow L_2 \Leftarrow G \tag{5}$$

can be translated into logic programming by rewriting it as the pair of rules

$$\begin{array}{c} L_1 \vee \overline{L_2} \Leftarrow G \\ \overline{L_1} \vee L_2 \Leftarrow G \end{array}$$

($\overline{L}$ stands for the literal complementary to L) and then using our extension of McCain's translation to rules with disjunctive heads. It turns out, however, that there is no need to use disjunctive logic programs in the case of synonymity rules. If, for instance, G in (5) is a literal then the following group of nondisjunctive rules will do:

$$\begin{array}{c} L_1 \leftarrow L_2, \mathit{not}\ \overline{G} \\ L_2 \leftarrow L_1, \mathit{not}\ \overline{G} \\ \overline{L_1} \leftarrow \overline{L_2}, \mathit{not}\ \overline{G} \\ \overline{L_2} \leftarrow \overline{L_1}, \mathit{not}\ \overline{G}. \end{array}$$

Finally, we extend the translation from propositional causal rules to first-order causal rules in the sense of (Lifschitz 1997). This version of causal logic is useful for defining the semantics of variables in action descriptions (Lifschitz and Ren 2007).

As part of motivation for our approach to transforming causal theories into logic programs, we start with a few additional comments on McCain's translation (Section 2). After reviewing the semantics of causal theories and logic programs in Sections 3 and 4, we describe four kinds of causal rules that we are interested in and show how to turn a theory consisting of such rules into a logic program (Section 5). This translation is related to answer set programming in Section 6, and its soundness is proved in Section 7.

Preliminary reports on this work are published in (Ferraris 2006; Ferraris 2007; Lee et al. 2010; Lifschitz and Yang 2010). Some results appear here for the first time, including the soundness of a representation of a synonymity rule with variables by a nondisjunctive logic program.

2 McCain's Translation Revisited

2.1 Incorporating Constraints

In causal logic, a *constraint* is a rule with the head $\bot$ (falsity). McCain's translation can be easily extended to constraints with a conjunction of literals in the body—causal rules of the form

$$\bot \Leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n. \tag{6}$$

In the language of logic programming, (6) can be represented by a rule similar to (3):

$$\bot \leftarrow \mathit{not}\ \neg A_1, \ldots, \mathit{not}\ \neg A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n. \tag{7}$$

Furthermore, each of the combinations *not* $\neg$ in (7) can be dropped without destroying the validity of the translation; that is to say, the rule

$$\bot \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \tag{8}$$

can be used instead of (7).

2.2 Eliminating Strong Negation

As observed in (Gelfond and Lifschitz 1991), strong negation can be eliminated from a logic program in favor of additional atoms. Denote the new atom representing a negative literal $\neg A$ by $\widehat{A}$. Then (3) will become

$$A_0 \leftarrow \mathit{not}\ \widehat{A_1}, \ldots, \mathit{not}\ \widehat{A_m}, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \tag{9}$$

if L is a positive literal A_0, and

$$\widehat{A_0} \leftarrow \mathit{not}\ \widehat{A_1}, \ldots, \mathit{not}\ \widehat{A_m}, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \tag{10}$$

if L is a negative literal $\neg A_0$. The *modified McCain translation* of a causal theory T consisting of rules of the forms (2) and (6) includes

- rules (8) corresponding to the constraints (6) of T,
- rules (9), (10) corresponding to the other rules of T, and

- the completeness constraints

$$\begin{array}{l} \leftarrow A, \widehat{A} \\ \leftarrow not\ A, not\ \widehat{A} \end{array} \qquad (11)$$

  for all atoms A.

For instance, the modified McCain translation of (1) is

$$\begin{array}{ll} p & \leftarrow not\ q \\ \widehat{q} & \leftarrow not\ \widehat{p} \\ & \leftarrow p, \widehat{p} \\ & \leftarrow not\ p, not\ \widehat{p} \\ & \leftarrow q, \widehat{q} \\ & \leftarrow not\ q, not\ \widehat{q}\cdot \end{array} \qquad (12)$$

The only answer set (stable model[2]) of this program is $\{p, \widehat{q}\}$.

This modification is useful to us in view of the fact that eliminating strong negation in favor of aditional atoms is part of the definition of a stable model proposed in (Ferraris et al. 2011, Section 8).

2.3 Rules as Formulas

The definition of a stable model for propositional formulas given in (Ferraris 2005) and the definition of a stable model for first-order sentences proposed in (Ferraris et al. 2011) become generalizations of the original definition (Gelfond and Lifschitz 1988) when we rewrite rules as logical formulas. For instance, rules (9) and (10), rewritten as propositional formulas, become

$$\neg\widehat{A_1} \wedge \cdots \wedge \neg\widehat{A_m} \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \to A_0 \qquad (13)$$

and

$$\neg\widehat{A_1} \wedge \cdots \wedge \neg\widehat{A_m} \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \to \widehat{A_0}\cdot \qquad (14)$$

Rule (8) can be identified with the formula

$$A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \ldots \wedge \neg A_n \to \bot \qquad (15)$$

or, alternatively, with

$$\neg(A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \ldots \wedge \neg A_n)\cdot \qquad (16)$$

The completeness constraints for an atom A turn into the formulas

$$\begin{array}{l} \neg(A \wedge \widehat{A}) \\ \neg(\neg A \wedge \neg\widehat{A})\cdot \end{array} \qquad (17)$$

[2] The term “stable model” was introduced in (Gelfond and Lifschitz 1988) to describe the meaning of logic programs with negation as failure but without strong negation. When the stable model semantics was extended to programs with strong negation in (Gelfond and Lifschitz 1991), the term “answer set” was proposed as a replacement.

Here is program (12) rewritten in the syntax of propositional logic:

$$\begin{array}{l} \neg q \rightarrow p \\ \neg \widehat{p} \rightarrow \widehat{q} \\ \neg(p \wedge \widehat{p}) \\ \neg(\neg p \wedge \neg \widehat{p}) \\ \neg(q \wedge \widehat{q}) \\ \neg(\neg q \wedge \neg \widehat{q}). \end{array} \tag{18}$$

Note that the process of rewriting a rule as a formula is applicable only when the rule does not contain strong negation; the symbol $\neg$ in the resulting formula corresponds to the negation as failure symbol (*not*) in the rule.

One of the advantages of writing rules as formulas is that it allows us to relate properties of stable models to subsystems of classical logic. We know, for instance, that if the equivalence of two sentences can be proved in intuitionistic logic (or even in the stronger logic of here-and-there) then these sentences have the same stable models (Ferraris et al. 2011, Theorem 5). This fact will be used here many times.

2.4 Translating Arbitrary Definite Theories

The requirement, in the definition of McCain's translation, that the bodies of all causal rules should be conjunctions of literals can be lifted by slightly modifying the translation process. Take any set T of causal rules of the forms

$$A \Leftarrow G, \tag{19}$$

$$\neg A \Leftarrow G, \tag{20}$$

$$\bot \Leftarrow G, \tag{21}$$

where A is an atom and G is an arbitrary propositional formula (rules of these forms are called *definite*). For each rule (19), take the formula $\neg\neg G \rightarrow A$; for each rule (20), the formula $\neg\neg G \rightarrow \widehat{A}$; for each rule (21), the formula $\neg G$. Then add completeness constraints (17) for all atoms A. Answer sets of this collection of propositional formulas correspond to the models of T.

In application to example (1), this modification of McCain's translation gives

$$\begin{array}{l} \neg\neg\neg q \rightarrow p \\ \neg\neg p \rightarrow \widehat{q} \\ \neg(p \wedge \widehat{p}) \\ \neg(\neg p \wedge \neg \widehat{p}) \\ \neg(q \wedge \widehat{q}) \\ \neg(\neg q \wedge \neg \widehat{q}). \end{array} \tag{22}$$

It is not surprising that (22) has the same answer set as (18): the two collections of formulas are intuitionistically equivalent to each other.[3]

[3] Indeed, $\neg\neg\neg q$ is intuitionistically equivalent to $\neg q$; the equivalence between $\neg\neg p$ and $\neg\widehat{p}$ is intuitionistically entailed by the formulas $\neg(p \wedge \widehat{p})$ and $\neg(\neg p \wedge \neg\widehat{p})$, which belong both to (18) and to (22).

3 Review: First-Order Causal Theories

According to (Lifschitz 1997), a first-order causal theory T is defined by

- a list $\mathbf{p}$ of distinct predicate constants,[4] called the *explainable symbols* of T,[5] and
- a finite set of *causal rules* of the form $F \Leftarrow G$, where F and G are first-order formulas.

The semantics of first-order causal theories can be described as follows. For each $p \in \mathbf{p}$, choose a new predicate variable υp of the same arity, and let $\upsilon\mathbf{p}$ stand for the list of all these variables. By $T^\dagger(\upsilon\mathbf{p})$ we denote the conjunction of the formulas

$$\forall \mathbf{x}(G \rightarrow F^{\mathbf{p}}_{\upsilon\mathbf{p}}) \tag{23}$$

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

We view T as shorthand for the sentence

$$\forall \upsilon\mathbf{p}(T^\dagger(\upsilon\mathbf{p}) \leftrightarrow (\upsilon\mathbf{p} = \mathbf{p}))\cdot \tag{24}$$

(By $\upsilon\mathbf{p} = \mathbf{p}$ we denote the conjunction of the formulas $\forall\mathbf{x}(\upsilon p(\mathbf{x}) \leftrightarrow p(\mathbf{x}))$ for all $p \in \mathbf{p}$, where $\mathbf{x}$ is a tuple of distinct object variables.) Accordingly, by a model of T we understand a model of (24) in the sense of classical logic. The models of T are characterized, informally speaking, by the fact that the interpretation of the explainable symbols $\mathbf{p}$ in the model is the only interpretation of these symbols that is "causally explained" by the rules of T.

In the definite case (see Section 2.4) second-order formula (24) can be replaced by an equivalent first-order formula using a process similar to Clark's completion (Clark 1978), called literal completion (McCain and Turner 1997), (Lifschitz 1997, Section 5). This process is used in the operation of CCalc.

Example 1. Let T be causal theory (1) with both p and q explainable. Then $T^\dagger(\upsilon p, \upsilon q)$ is

$$(\neg q \rightarrow \upsilon p) \wedge (p \rightarrow \neg \upsilon q)$$

(υp, υq are propositional variables), so that T is understood as shorthand for the second-order propositional formula ("QBF")

$$\forall(\upsilon p)(\upsilon q)((\neg q \rightarrow \upsilon p) \wedge (p \rightarrow \neg \upsilon q) \leftrightarrow (\upsilon p \leftrightarrow p) \wedge (\upsilon q \leftrightarrow q))\cdot \tag{25}$$

This formula is equivalent to $p \wedge \neg q$.[6]

[4] We view propositional symbols as predicate constants of arity 0, so that they are allowed in $\mathbf{p}$. Equality, on the other hand, may not be declared explainable.

[5] To be precise, the definition in (Lifschitz 1997) is more general: object and function constants can be treated as explainable as well.

[6] This fact can be verified by replacing the universal quantifier in (25) with the conjunction of the four propositional formulas obtained by substituting all possible combinations of values for the variables υp, υq, and simplifying the result. Alternatively, one can apply literal completion to rules (1) and simplify the result.

Example 2. Let T be the causal theory consisting of two rules:

$$p(a) \Leftarrow \top$$

(here $\top$ is the logical constant *true*) and

$$\neg p(x) \Leftarrow \neg p(x),$$

with the explainable symbol p. The first rule says that there is a cause for a to have property p. The second rule says that if an object does not have property p then there is a cause for that; including this rule in a causal theory has, informally speaking, the same effect as saying that p is false by default (Lifschitz 1997, Section 3). In this case, $T^{\dagger}(\upsilon p)$ is

$$\upsilon p(a) \wedge \forall x(\neg p(x) \rightarrow \neg \upsilon p(x)),$$

so that T is understood as shorthand for the sentence

$$\forall \upsilon p(\upsilon p(a) \wedge \forall x(\neg p(x) \rightarrow \neg \upsilon p(x)) \leftrightarrow \forall x(\upsilon p(x) \leftrightarrow p(x)))\cdot$$

This sentence is equivalent to the first-order formula

$$\forall x(p(x) \leftrightarrow x = a), \tag{26}$$

as can be verified by applying literal completion to the rules of T.

4 Review: Stable Models

Some details of the definition of a stable model proposed in (Ferraris et al. 2011) depend on which propositional connectives are treated as primitives, and which are viewed as abbreviations. The convention there is to take the 0-place connective $\bot$ and the binary connectives $\wedge$, $\vee$, $\rightarrow$ as primitives; $\neg F$ is shorthand for $F \rightarrow \bot$.

In this paper we adopt the view that first-order formulas are formed using a slightly larger set of propositional connectives:

$$\top,\ \bot,\ \neg,\ \wedge,\ \vee,\ \rightarrow$$

(as well as the quantifiers $\forall$, $\exists$). On the other hand, stable models are only defined here for sentences of a special syntactic form. A first-order sentence is a *rule*[7] if it has the form $\widetilde{\forall}(F \rightarrow G)$ and has no occurrences of $\rightarrow$ other than the one explicitly shown.[8] If a sentence F does not contain implication then we will identify it with the rule $\top \rightarrow F$. For instance, propositional formulas (13)–(18) are rules. A *logic program* is a conjunction of rules. The definition of a stable model below is more limited than the definition from (Ferraris et al. 2011) because it is only applicable to programs, not to arbitrary sentences. For instance, it does not cover the formulas $(p \rightarrow q) \rightarrow r$ and $(p \rightarrow q) \vee r$. On the other hand, it is simpler than the general definition, and it is sufficient for our present purposes.

[7] Or *program rule*, to distinguish it from causal rules in the sense of Section 3.
[8] $\widetilde{\forall}F$ stands for the universal closure of F.

We need the following notation from (Lifschitz 1994). If p and q are predicate constants of the same arity then $p \leq q$ stands for the formula

$$\forall \mathbf{x}(p(\mathbf{x}) \rightarrow q(\mathbf{x})),$$

where $\mathbf{x}$ is a tuple of distinct object variables. If $\mathbf{p}$ and $\mathbf{q}$ are tuples $p_1, \dots, p_n$ and $q_1, \dots, q_n$ of predicate constants then $\mathbf{p} \leq \mathbf{q}$ stands for the conjunction

$$(p_1 \leq q_1) \wedge \cdots \wedge (p_n \leq q_n),$$

and $\mathbf{p} < \mathbf{q}$ stands for $(\mathbf{p} \leq \mathbf{q}) \wedge \neg(\mathbf{q} \leq \mathbf{p})$. In second-order logic, we apply the same notation to tuples of predicate variables.

Let $\mathbf{p}$ be a list of distinct predicate constants; members of $\mathbf{p}$ will be called *intensional predicates*.[9] For each $p \in \mathbf{p}$, choose a predicate variable υp of the same arity, and let $\upsilon\mathbf{p}$ stand for the list of all these variables. For any logic program F, by $\mathrm{SM}_{\mathbf{p}}[F]$ we denote the second-order sentence

$$F \wedge \neg\exists \upsilon\mathbf{p}((\upsilon\mathbf{p} < \mathbf{p}) \wedge F^{\diamond}(\upsilon\mathbf{p})), \tag{27}$$

where $F^{\diamond}(\upsilon\mathbf{p})$ is the formula obtained from F by replacing, for every $p \in \mathbf{p}$, each occurrence of p that is not in the scope of negation with υp. A model of F is *stable* (relative to the set $\mathbf{p}$ of intensional predicates) if it satisfies $\mathrm{SM}_{\mathbf{p}}[F]$.[10]

Example 3. Let F be the propositional formula $\neg p \rightarrow q$ (the one-rule program $q \leftarrow \mathit{not}\ p$, in traditional notation). If both p and q are intensional then $F^{\diamond}(\upsilon p, \upsilon q)$ is

$$\neg p \rightarrow \upsilon q,$$

so that $\mathrm{SM}_{pq}[F]$ is

$$(\neg p \rightarrow q) \wedge \neg\exists(\upsilon p)(\upsilon q)(((\upsilon p, \upsilon q) < (p, q)) \wedge (\neg p \rightarrow \upsilon q))\cdot$$

This formula is equivalent to $\neg p \wedge q$.[11] Consequently F has one stable model: p is false and q is true.

Example 4. Let F be the formula

$$\forall x(\neg p(x) \rightarrow (q(x) \vee \neg q(x))) \tag{28}$$

(it can be thought of as a formula representation of the LPARSE choice rule

[9] This list usually consists of all predicate symbols occurring in the heads of rules; those are the predicates that we "intend to characterize" by the rules of the program. The original definition of a stable model (Gelfond and Lifschitz 1988) treats all predicates as intensional.

[10] We can make two comments about the relation of this treatment of stable models to earlier work. First, if we drop from the definition of SM the words "that is not in the scope of negation" then it will turn into the definition of parallel circumscription (McCarthy 1986; Lifschitz 1985). It follows that if a logic program does not contain negation then the class of its stable models is identical to the class of its minimal models. The stipulation in the description of $F^{\diamond}(\upsilon\mathbf{p})$ that intensional predicates in the scope of negation are not replaced by variables is a reflection of the idea of negation as failure. Second, the operator of SM as defined in (Ferraris et al. 2011) produces, in application to a logic program, a second-order formula that is usually more complex than (27) but is equivalent to it.

[11] Methods for simplifying the result of applying the operator SM are discussed in (Ferraris et al. 2011).

`{q(X)} :- not p(X)`).[12] If we take q to be the only intensional predicate then $F^\diamond(\upsilon q)$ is

$$\forall x(\neg p(x) \rightarrow (\upsilon q(x) \vee \neg q(x))).$$

Consequently $\mathrm{SM}_q[F]$ is

$$\forall x(\neg p(x) \rightarrow (q(x) \vee \neg q(x))) \wedge \neg\exists \upsilon q((\upsilon q < q) \wedge \forall x(\neg p(x) \rightarrow (\upsilon q(x) \vee \neg q(x)))).$$

The first conjunctive term here is logically valid and can be dropped. The second is equivalent to the first-order formula $\neg\exists x(p(x) \wedge q(x))$, which reflects the intuitive meaning of the choice rule above: q is an arbitrary set disjoint from p.

The relationship between the definition of a stable model given above and the operation of answer set solvers is discussed in Section 6.

If programs F and G are intuitionistically equivalent then $\mathrm{SM}_{\mathbf{p}}[F]$ is equivalent to $\mathrm{SM}_{\mathbf{p}}[G]$, that is to say, F and G have the same stable models. Moreover, for establishing that F and G have the same stable models we only need to derive $F \leftrightarrow G$ intuitionistically from the excluded middle formulas $\widetilde{\forall}(H \vee \neg H)$ for some formulas H that do not contain intensional predicates. This fact follows from (Ferraris et al. 2011, Theorem 5).

5 Turning a Causal Theory into a Logic Program

5.1 Four Types of Causal Rules

In the rest of the paper, we assume that the bodies of causal rules do not contain implication. This is not an essential limitation, because in classical logic $\rightarrow$ can be expressed in terms of other connectives, and the meaning of a causal rule does not change if we replace its body (or head) by a classically equivalent formula.

Here are four types of rules that we are going to consider, in the order of increasing complexity of their heads:

- The head is $\bot$, that is, the rule is a constraint. Such causal rules will be also called *C-rules.*
- The head is a literal containing an explainable predicate symbol. These are *L-rules.*
- The head has the form $L_1 \leftrightarrow L_2$, where each L_i is a literal containing an explainable predicate symbol. These are *synonymity rules*, or *S-rules.*
- The head has the form $L_1 \vee \cdots \vee L_n$ $(n \geq 0)$, where each L_i is a literal containing an explainable predicate symbol. These are *D-rules.*

All C-rules and L-rules can be viewed also as D-rules, and any S-rule can be replaced with an equivalent pair of D-rules (see Lemma 11 in Section 7.2). Nevertheless, we give special attention here to rules of the first three types, and the reason is that our translation handles such rules in special ways. It appears that

[12] This rule would not be accepted by LPARSE, however, because it is "nonrestricted." For a description of the language of LPARSE see `http://www.tcs.hut.fi/Software/ smodels/lparse.ps`.

causal rules of types C, L, and S will be more important than general D-rules in applications of this work to the automation of reasoning about actions.

On the other hand, the possibility of reducing types C, L, and S to type D plays an important role in the proof of the soundness of our translation (Section 7). This is one of the reasons why we are interested in general D-rules.

The requirement, in the definitions of types L, S and D, that the literals in the head of the rule contain explainable predicate symbols is not an essential limitation. If, for instance, the predicate symbol in the head of $L \Leftarrow G$ is not explainable then this rule can be equivalently replaced by the C-rule $\bot \Leftarrow G \wedge \overline{L}$. If a rule has the form

$$L_1 \leftrightarrow L_2 \Leftarrow G$$

and the predicate symbol in L_1 is not explainable then the rule can be replaced by

$$\begin{array}{l} L_2 \Leftarrow G \wedge L_1, \\ \overline{L_2} \Leftarrow G \wedge \overline{L_1}. \end{array}$$

If a rule has the form

$$L_1 \vee \cdots \vee L_n \Leftarrow G$$

and the predicate symbol in L_1 is not explainable then the rule can be replaced by

$$L_2 \vee \cdots \vee L_n \Leftarrow G \wedge \overline{L_1}.$$

5.2 Translating C-Rules and L-Rules

The transformation described in this section generalizes McCain's translation, in the form described in Section 2.4, to first-order causal theories.

The operator Tr_c, which transforms any C-rule into a program rule, is defined by the formula

$$\mathrm{Tr}_c[\bot \Leftarrow G] = \widetilde{\forall}\neg G.$$

The operator Tr_l, which transforms any L-rule into a program rule, is defined by the formulas

$$\begin{array}{r} \mathrm{Tr}_l[p(\mathbf{t}) \Leftarrow G] = \widetilde{\forall}(\neg\neg G \rightarrow p(\mathbf{t})), \\ \mathrm{Tr}_l[\neg p(\mathbf{t}) \Leftarrow G] = \widetilde{\forall}(\neg\neg G \rightarrow \widehat{p}(\mathbf{t})) \end{array}$$

($\mathbf{t}$ is a tuple of terms).

If T is a causal theory consisting of C-rules and L-rules then its translation $\mathrm{Tr}[T]$ is the logic program obtained by conjoining

- the rules obtained by applying Tr_c to the C-rules of T,
- the rules obtained by applying Tr_l to the L-rules of T, and
- the completeness constraints

$$\begin{array}{l} \forall\mathbf{x}\neg(p(\mathbf{x}) \wedge \widehat{p}(\mathbf{x})), \\ \forall\mathbf{x}\neg(\neg p(\mathbf{x}) \wedge \neg\widehat{p}(\mathbf{x})) \end{array} \tag{29}$$

  ($\mathbf{x}$ is a tuple of distinct object variables) for all explainable predicate symbols p of T.

Let $\mathbf{p}$ be the list of explainable predicate symbols p of T, and let $\widehat{\mathbf{p}}$ be the list of the corresponding predicate symbols $\widehat{p}$. Take the union of $\mathbf{p}$ and $\widehat{\mathbf{p}}$ to be the set of intensional predicates. Then the stable models of the logic program $\mathrm{Tr}[T]$ are "almost identical" to the models of T; the difference is due to the fact that the language of T does not contain the symbols $\widehat{\mathbf{p}}$. Let CC be the conjunction of all completeness constraints (29). Then the relationship between T and $\mathrm{Tr}[T]$ can be described as follows:

$$\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\mathrm{Tr}[T]] \textit{ is equivalent to } T \wedge CC. \tag{30}$$

This claim, expressing the soundness of our translation, is extended in Sections 5.3 and 5.4 to causal theories containing S-rules and D-rules, and its proof is given in Section 7.

Since the conjunction of formulas (29) is classically equivalent to

$$\forall \mathbf{x}(\widehat{p}(\mathbf{x}) \leftrightarrow \neg p(\mathbf{x})), \tag{31}$$

sentence CC can be viewed as the conjunction of explicit definitions of the predicates $\widehat{\mathbf{p}}$ in terms of the predicates $\mathbf{p}$. Consequently the relationship (30) shows that $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\mathrm{Tr}[T]]$ is a definitional extension of T. The models of $\mathrm{Tr}[T]$ that are stable relative to $\mathbf{p}\widehat{\mathbf{p}}$ can be characterized as the models of T extended by the interpretations of the predicates $\widehat{\mathbf{p}}$ that are provided by definitions (31).

Example 1, continued. If T is causal theory (1) with both p and q explainable then $\mathrm{Tr}[T]$ is the conjunction of formulas (22). The result of applying the operator $\mathrm{SM}_{pq\widehat{p}\widehat{q}}$ to this conjunction is equivalent to

$$p \wedge \neg q \wedge \neg\widehat{p} \wedge \widehat{q}.$$

Recall that T is equivalent to the first half of this conjunction (Section 3). The second half tells us that the truth values of $\widehat{p}$, $\widehat{q}$ are opposite to the truth values of p, q. In the only stable model of (22), p and $\widehat{q}$ are true, and $\widehat{p}$ and q are false; if we "forget" the truth values of $\widehat{p}$ and $\widehat{q}$ then we will arrive at the model of (1).

Example 2, continued. Our translation turns the causal theory from Example 2 into the conjunction of the rules

$$\begin{array}{l}\neg\neg\top \rightarrow p(a),\\ \forall x(\neg\neg\neg p(x) \rightarrow \widehat{p}(x)),\\ \forall x\neg(p(x) \wedge \widehat{p}(x)),\\ \forall x\neg(\neg p(x) \wedge \neg\widehat{p}(x)),\end{array}$$

or, after intuitionistically equivalent transformations,

$$\begin{array}{l}p(a),\\ \forall x(\neg p(x) \rightarrow \widehat{p}(x)),\\ \forall x\neg(p(x) \wedge \widehat{p}(x)),\\ \forall x\neg(\neg p(x) \wedge \neg\widehat{p}(x)).\end{array}$$

The result of applying $\mathrm{SM}_{p\widehat{p}}$ to the conjunction of these formulas is equivalent to the conjunction of (26) with the formula $\forall x(\widehat{p}(x) \leftrightarrow \neg p(x))$, which says that $\widehat{p}$ is the complement of p.

Example 5. Consider the following causal rules:

$$\begin{aligned} on_1(x) &\Leftarrow toggle(x) \wedge \neg on_0(x), \\ \neg on_1(x) &\Leftarrow toggle(x) \wedge on_0(x), \\ on_1(x) &\Leftarrow on_0(x) \wedge on_1(x), \\ \neg on_1(x) &\Leftarrow \neg on_0(x) \wedge \neg on_1(x). \end{aligned} \tag{32}$$

The first pair of rules describes the effect of toggling a switch x: this action causes the fluent $on(x)$ at time 1 to take the value opposite to its value at time 0. The second pair solves the frame problem (Shanahan 1997) for the fluent $on(x)$ by postulating that if the value of that fluent at time 1 is equal to its previous value then there is a cause for this. (Inertia, in the sense of commonsense reasoning, is the cause.) Let T be the causal theory with rules (32) and with on_1 as the only explainable symbol. Using literal completion, we can check that T is equivalent to

$$\forall x(on_1(x) \leftrightarrow ((on_0(x) \wedge \neg toggle(x)) \vee (\neg on_0(x) \wedge toggle(x)))). \tag{33}$$

Our translation turns T into the conjunction of the rules

$$\begin{aligned} &\forall x(\neg\neg(toggle(x) \wedge \neg on_0(x)) \rightarrow on_1(x)), \\ &\forall x(\neg\neg(toggle(x) \wedge on_0(x)) \rightarrow \widehat{on_1}(x)), \\ &\forall x(\neg\neg(on_0(x) \wedge on_1(x)) \rightarrow on_1(x)), \\ &\forall x(\neg\neg(\neg on_0(x) \wedge \neg on_1(x)) \rightarrow \widehat{on_1}(x)), \\ &\forall x \neg(on_1(x) \wedge \widehat{on_1}(x)), \\ &\forall x \neg(\neg on_1(x) \wedge \neg\widehat{on_1}(x)), \end{aligned} \tag{34}$$

or, equivalently,[13]

$$\begin{aligned} &\forall x(toggle(x) \wedge \neg on_0(x) \rightarrow on_1(x)), \\ &\forall x(toggle(x) \wedge on_0(x) \rightarrow \widehat{on_1}(x)), \\ &\forall x(on_0(x) \wedge \neg\widehat{on_1}(x) \rightarrow on_1(x)), \\ &\forall x(\neg on_0(x) \wedge \neg on_1(x) \rightarrow \widehat{on_1}(x)), \\ &\forall x \neg(on_1(x) \wedge \widehat{on_1}(x)), \\ &\forall x \neg(\neg on_1(x) \wedge \neg\widehat{on_1}(x)). \end{aligned} \tag{35}$$

The result of applying $\mathrm{SM}_{on_1 \widehat{on_1}}$ to this program is equivalent to the conjunction of (33) with the formula $\forall x(\widehat{on_1}(x) \leftrightarrow \neg on_1(x))$, which says that $\widehat{on_1}$ is the complement of on_1.

Example 6. The constraint

$$\bot \Leftarrow toggle(badswitch)$$

expresses that *badswitch* is stuck: the action of toggling it is not executable. If we add this constraint to the causal theory from Example 5 then the rule

$$\neg toggle(badswitch)$$

[13] Removing the double negations in the first two lines of (34) is possible because neither *toggle* nor on_0 is intensional (see the comment on equivalent transformations of logic programs at the end of Section 4). In a similar way, the antecedent of the third impication in (34) can be replaced by $on_0(x) \wedge \neg\neg on_1(x)$; the equivalence between $\neg\neg on_1(x)$ and $\neg\widehat{on_1}(x)$ is intuitionistically entailed by the last two lines of (34). The fourth line of (34) is simplified in a similar way.

will be added to its translation (35).

The bodies of causal rules in Examples 5 and 6 are syntactically simple: they are conjunctions of literals. The general definitions of a C-rule and an L-rule do not impose any restrictions on the form of the body, and in applications of causal logic to formalizing commonsense knowledge this generality is often essential. For instance, the statement "each position must have at least one neighbor" in the landscape structure of the Zoo World[14] would be represented in causal logic by a C-rule with a quantifier in the body.

5.3 Translating S-Rules

We will turn now to translating synonymity rules (Section 5.1). The operator Tr_s, transforming any such rule into a logic program, is defined by the formulas

$$\begin{aligned}&\mathrm{Tr}_s[p_1(\mathbf{t}^1) \leftrightarrow p_2(\mathbf{t}^2) \Leftarrow G] = \mathrm{Tr}_s[\neg p_1(\mathbf{t}^1) \leftrightarrow \neg p_2(\mathbf{t}^2) \Leftarrow G]\\ &\quad = \widetilde{\forall}(\neg\neg G \wedge p_1(\mathbf{t}^1) \rightarrow p_2(\mathbf{t}^2)) \wedge \widetilde{\forall}(\neg\neg G \wedge p_2(\mathbf{t}^2) \rightarrow p_1(\mathbf{t}^1)) \wedge\\ &\qquad \widetilde{\forall}(\neg\neg G \wedge \widehat{p_1}(\mathbf{t}^1) \rightarrow \widehat{p_2}(\mathbf{t}^2)) \wedge \widetilde{\forall}(\neg\neg G \wedge \widehat{p_2}(\mathbf{t}^2) \rightarrow \widehat{p_1}(\mathbf{t}^1)),\end{aligned}$$

$$\begin{aligned}&\mathrm{Tr}_s[\neg p_1(\mathbf{t}^1) \leftrightarrow p_2(\mathbf{t}^2) \Leftarrow G] = \mathrm{Tr}_s[p_1(\mathbf{t}^1) \leftrightarrow \neg p_2(\mathbf{t}^2) \Leftarrow G]\\ &\quad = \widetilde{\forall}(\neg\neg G \wedge \widehat{p_1}(\mathbf{t}^1) \rightarrow p_2(\mathbf{t}^2)) \wedge \widetilde{\forall}(\neg\neg G \wedge p_2(\mathbf{t}^2) \rightarrow \widehat{p_1}(\mathbf{t}^1)) \wedge\\ &\qquad \widetilde{\forall}(\neg\neg G \wedge p_1(\mathbf{t}^1) \rightarrow \widehat{p_2}(\mathbf{t}^2)) \wedge \widetilde{\forall}(\neg\neg G \wedge \widehat{p_2}(\mathbf{t}^2) \rightarrow p_1(\mathbf{t}^1))\end{aligned}$$

($\mathbf{t}^1$, $\mathbf{t}^2$ are tuples of terms). The definition of program $\mathrm{Tr}[T]$ from Section 5.2 is extended to causal theories that may contain S-rules, besides C-rules and L-rules, by adding that $\mathrm{Tr}[T]$ includes also

- the rules obtained by applying Tr_s to the S-rules of T.

Example 7. Extend the theory from Example 5 by the rule

$$\mathit{dark} \leftrightarrow \neg \mathit{on}_1(\mathit{myswitch}) \Leftarrow \top, \tag{36}$$

where *dark* is explainable. The corresponding logic program is obtained from (35) by adding the rules

$$\begin{array}{l}\widehat{\mathit{dark}} \rightarrow \mathit{on}_1(\mathit{myswitch}),\\ \mathit{on}_1(\mathit{myswitch}) \rightarrow \widehat{\mathit{dark}},\\ \mathit{dark} \rightarrow \widehat{\mathit{on}_1}(\mathit{myswitch}),\\ \widehat{\mathit{on}_1}(\mathit{myswitch}) \rightarrow \mathit{dark},\\ \neg(\mathit{dark} \wedge \widehat{\mathit{dark}}),\\ \neg(\neg \mathit{dark} \wedge \neg\widehat{\mathit{dark}})\cdot\end{array} \tag{37}$$

We will see that the soundness property (30) holds for arbitary causal theories consisting of rules of types C, L, and S.

[14] The challenge of formalizing the Zoo World was proposed as part of the Logic Modelling Workshop (`http:/www/ida.liu.se/ext/etai/lmw/`). The possibility of addressing this challenge using CCalc is discussed in (Akman et al. 2004, Section 4).

5.4 Translating D-Rules

A D-rule (Section 5.1) has the form

$$\bigvee_{A\in Pos} A \vee \bigvee_{A\in Neg} \neg A \Leftarrow G \tag{38}$$

for some sets *Pos*, *Neg* of atomic formulas.

If A is an atomic formula $p(\mathbf{t})$, where $p \in \mathbf{p}$ and $\mathbf{t}$ is a tuple of terms, then by $\widehat{A}$ we will denote the formula $\widehat{p}(\mathbf{t})$. The operator Tr_d transforms D-rule (38) into the program rule

$$\widetilde{\forall}\left(\neg\neg G \wedge \bigwedge_{A\in Pos}(\widehat{A}\vee\neg\widehat{A}) \wedge \bigwedge_{A\in Neg}(A\vee\neg A) \;\rightarrow\; \bigvee_{A\in Pos} A \vee \bigvee_{A\in Neg}\widehat{A}\right). \tag{39}$$

Example 8. The result of applying Tr_d to the D-rule

$$p \vee \neg q \vee \neg r \Leftarrow s$$

is

$$\neg\neg s \wedge (\widehat{p}\vee\neg\widehat{p}) \wedge (q\vee\neg q)\wedge(r\vee\neg r) \rightarrow p \vee \widehat{q} \vee \widehat{r}.$$

The number of "excluded middle formulas" conjoined with $\neg\neg G$ in (39) equals the number of disjunctive terms in the head of D-rule (38). In particular, if (38) is an L-rule then the antecedent of (39) contains one such formula. For instance, in application to the first rule of (1) Tr_d produces the program rule

$$\neg\neg\neg q \wedge (\widehat{p}\vee\neg\widehat{p}) \rightarrow p,$$

which is more complex than the first rule of (22).

For a fixed collection $\mathbf{p}$ of explainable symbols, let C, L, S, and D be finite sets of causal rules of types C, L, S, and D respectively. By $\mathrm{Tr}[C,L,S,D]$ we denote the logic program obtained by conjoining

- the rules obtained by applying Tr_c to all rules from C,
- the rules obtained by applying Tr_l to all rules from L,
- the programs obtained by applying Tr_s to all rules from S,
- the rules obtained by applying Tr_d to all rules from D,
- the completeness constraints (29) for all explainable symbols p.

Our most general form of the soundness theorem, proved in Section 7, asserts that

$$\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\mathrm{Tr}[C,L,S,D]] \textit{ is equivalent to } T \wedge CC \tag{40}$$

for the causal theory T with the set of rules $C \cup L \cup S \cup D$. In the special case when D is empty this theorem turns into the assertion stated at the end of Section 5.3.

6 Using Answer Set Solvers to Generate Models of a Causal Theory

The discussion of answer set solvers in this section, as almost any discussion of software, is somewhat informal. We assume here that the first-order language under consideration does not contain function constants of nonzero arity.

An answer set solver can be viewed as a system for generating stable models in the sense of Section 4, with three caveats. First, currently available solvers require that the input program have a syntactic form that is much more restrictive than the syntax of first-order logic.[15] Preprocessing based on intuitionistically equivalent transformations often helps us alleviate this difficulty. There exists a tool, called F2LP (Lee and Palla 2009), that converts first-order formulas of a rather general kind into logic programs accepted by LPARSE. The rules produced by the process described in the previous section have no existential quantifiers in their heads, and all quantifiers in their bodies are in the scope of negation. Consequently, these rules satisfy a syntactic condition that guarantees the correctness of the translation implemented in F2LP.

Second, answer set solvers represent stable models by sets of ground atoms. To introduce such a representation, we usually choose a finite set of object constants that includes all object constants occurring in the program, and restrict attention to Herbrand interpretations of the extended language. The `#domain` construct of LPARSE[16] can be used to specify the object constants constituting the domain of the variables in the program.

Third, most existing answer set solvers are unaware of the possibility of non-intensional (or *extensional*) predicates. Treating a predicate constant as extensional can be simulated using a choice rule (Ferraris et al. 2011, Theorem 2). There is also another approach to overcoming this limitation. Take a conjunction E of some ground atoms containing extensional predicates, and assume that we are interested in the Herbrand stable models of a program F that interpret the extensional predicates in accordance with E (every atom from E is true; all other atoms containing extensional predicates are false). Under some syntactic conditions,[17] these stable models are identical to the Herbrand stable models of $F \wedge E$ with all predicate constants treated as intensional. This can be proved using the splitting theorem from (Ferraris et al. 2009).

Example 4, continued. We would like to find the stable models of (28), with q intensional, that have the universe $\{a, b, c, d\}$ and make p true on a, b and false on c, d. This is the same as to look for the Herbrand stable models of the formula

$$\forall x(\neg p(x) \rightarrow (q(x) \vee \neg q(x))) \wedge p(a) \wedge p(b),$$

with c and d viewed as object constants of the language along with a and b, and with both p and q taken to be intensional.

[15] They also require that the input satisfy some safety conditions. See, for instance, Chapter 3 of the DLV manual, `http://www.dbai.tuwien.ac.at/proj/dlv/man/`.

[16] See Footnote ([12]).

[17] Specifically, under the assumption that every occurrence of every extensional predicate in F is in the scope of negation or in the antecedent of an implication.

```
u(a;b;c;d).
#domain u(X).
{q(X)} :- not p(X).
p(a;b).
```

Fig. 1. Example 4 with a 4-element universe in the language of LPARSE

A representation of this example in the language of LPARSE is shown in Figure 1. The auxiliary predicate `u` describes the universe of the interpretations that we are interested in. The first line is shorthand for

```
u(a). u(b). u(c). u(d).
```

and the last line is understood by `lparse` in a similar way.

Given this input, the answer set solver SMODELS generates 4 stable models, representing the subsets of $\{a, b, c, d\}$ that are disjoint from $\{a, b\}$:

```
Answer: 1
Stable Model: p(b) p(a) u(d) u(c) u(b) u(a)
Answer: 2
Stable Model: p(b) p(a) q(d) u(d) u(c) u(b) u(a)
Answer: 3
Stable Model: p(b) p(a) q(c) u(d) u(c) u(b) u(a)
Answer: 4
Stable Model: p(b) p(a) q(d) q(c) u(d) u(c) u(b) u(a)
```

In application to the logic program obtained from a causal theory T as described in Section 5, this process often allows us to find the models of T with a given universe and given extents of extensional predicates.

Example 7, continued. There are two switches, *myswitch* and *hisswitch*. It is dark in my room at time 1 if and only if *myswitch* is not *on* at time 1. At time 0, both switches are *on*; then *hisswitch* is toggled, and *myswitch* is not. Is it dark in my room at time 1? We would like to answer this question using answer set programming.

This example of commonsense reasoning involves inertia (the value of the fluent *on*(*myswitch*) does not change because this fluent is not affected by the action that is executed) and indirect effects of actions: whether or not it is dark in the room at time 1 after performing some actions is determined by the effect of these actions on the fluent *on*(*myswitch*).

Mathematically, we are talking here about the causal theory T with rules (32) and (36), with the object constant *hisswitch* added to the language, and with the explainable symbols on_1 and *dark*. We are interested in the Herbrand models of T in which the extents of the extensional predicates are described by the atoms

$$on_0(myswitch),\ on_0(hisswitch),\ toggle(hisswitch)\cdot$$

As we have seen, the logic program Tr$[T]$ is equivalent to the conjunction of

```
u(myswitch;hisswitch).
#domain u(X).

on1(X) :- toggle(X), not on0(X).
-on1(X) :- toggle(X), on0(X).
on1(X) :- on0(X), not -on1(X).
-on1(X) :- not on0(X), not on1(X).
:- not on1(X), not -on1(X).

on1(myswitch) :- -dark.
-dark :- on1(myswitch).
-on1(myswitch) :- dark.
dark :- -on1(myswitch).
:- not dark, not -dark.

on0(myswitch;hisswitch).
toggle(hisswitch).
```

Fig. 2. Example 7 with two switches in the language of LPARSE

rules (35) and (37). The corresponding LPARSE input file is shown in Figure 2. In this file, the "true negation" symbol `-` is used in the ASCII representations of the symbols $\widehat{on_1}$ and $\widehat{dark}$; the LPARSE counterparts of the rules

$$\forall x \neg(on_1(x) \wedge \widehat{on_1}(x)),$$
$$\neg(dark \wedge \widehat{dark})$$

are dropped, because such "coherence" conditions are verified by the system automatically.

Given this input, SMODELS generates the only model of T satisfying the given conditions:

```
Answer: 1
Stable Model: -on1(hisswitch) on1(myswitch) -dark toggle(hisswitch)
on0(hisswitch) on0(myswitch) u(hisswitch) u(myswitch)
```

The presence of `-dark` in this model tells us that it is not dark in the room at time 1.

The example above is an example of "one-step temporal projection"—predicting the value of a fluent after performing a single action in a given state. Some other kinds of temporal reasoning and planning can be performed by generating models of simple modifications of the given causal theory (Giunchiglia et al. 2004, Section 3.3); this is one of the ideas behind the design of CCALC and COALA. McCain's translation reviewed in the introduction and its generalization presented in Section 5 allow us to solve such problems automatically using an answer set solver.

7 Proof of Soundness

To prove claim (40), which expresses the soundness of our translation, we will first establish it for the case when $C = L = S = \emptyset$ (Section 7.1). In this "leading special case" all rules of the given causal theory are D-rules, and they are converted to program rules using the translation Tr_d. Then we will derive the soundness theorem in full generality (Section 7.2).

7.1 Leading Special Case

Let T be a finite set of causal rules of the form (38). Let Π be the conjunction of the corresponding program rules (39), and let CC, as before, stand for the conjunction of the completeness constraints (29) for all explainable symbols p of T. We want to show that

$$\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi \wedge CC] \textit{ is equivalent to } T \wedge CC\cdot \tag{41}$$

The key steps in the proof below are Lemma 5 (one half of the equivalence) and Lemma 8 (the other half).

In the statement of the following lemma, $\neg\mathbf{p}$ stands for the list of predicate expressions[18] $\lambda\mathbf{x}\neg p(\mathbf{x})$, where $\mathbf{x}$ is a list of distinct object variables, for all p from $\mathbf{p}$. By $\upsilon\mathbf{p}$, $\upsilon\widehat{\mathbf{p}}$ we denote the lists of predicate variables used in the second-order formula $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi \wedge CC]$ (see Section 4).

Lemma 1
Formula $(\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) < (\mathbf{p}, \neg\mathbf{p})$ is equivalent to

$$\bigvee_{p\in\mathbf{p}} (((\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg\mathbf{p})) \wedge \exists\mathbf{x}(\neg\upsilon p(\mathbf{x}) \wedge \neg\upsilon\widehat{p}(\mathbf{x})))\cdot$$

Proof
Note first that

$$\begin{aligned}
&(\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) < (\mathbf{p}, \neg\mathbf{p})\\
&\Leftrightarrow ((\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg\mathbf{p})) \wedge \neg\,((\mathbf{p}, \neg\mathbf{p}) \leq (\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}))\\
&\Leftrightarrow ((\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg\mathbf{p})) \wedge \textstyle\bigvee_{p\in\mathbf{p}} \exists\mathbf{x}((p(\mathbf{x}) \wedge \neg\upsilon p(\mathbf{x})) \vee (\neg p(\mathbf{x}) \wedge \neg\upsilon\widehat{p}(\mathbf{x})))\\
&\Leftrightarrow \textstyle\bigvee_{p\in\mathbf{p}}(((\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg\mathbf{p})) \wedge \exists\mathbf{x}((p(\mathbf{x}) \wedge \neg\upsilon p(\mathbf{x})) \vee (\neg p(\mathbf{x}) \wedge \neg\upsilon\widehat{p}(\mathbf{x}))))\cdot
\end{aligned}$$

The disjunction after $\exists\mathbf{x}$ is equivalent to

$$(p(\mathbf{x}) \vee \neg\upsilon\widehat{p}(\mathbf{x})) \wedge (\neg\upsilon p(\mathbf{x}) \vee \neg p(\mathbf{x})) \wedge (\neg\upsilon p(\mathbf{x}) \vee \neg\upsilon\widehat{p}(\mathbf{x}))\cdot \tag{42}$$

Since $(\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg\mathbf{p})$ entails

$$\upsilon p(\mathbf{x}) \rightarrow p(\mathbf{x}) \text{ and } \upsilon\widehat{p}(\mathbf{x}) \rightarrow \neg p(\mathbf{x}),$$

the first conjunctive term of (42) can be rewritten as $\neg\upsilon\widehat{p}(\mathbf{x})$, and the second term as $\neg\upsilon p(\mathbf{x})$, so that (42) will turn into $\neg\upsilon p(\mathbf{x}) \wedge \neg\upsilon\widehat{p}(\mathbf{x})$. $\square$

[18] See (Lifschitz 1994, Section 3.1).

For any formula F, by $F_{\Sigma 1}$ we denote the formula

$$F^{(\upsilon\mathbf{p})(\upsilon\widehat{\mathbf{p}})}_{(\upsilon\mathbf{p}\wedge\mathbf{p})(\neg\upsilon\mathbf{p}\wedge\neg\mathbf{p})}$$

where $\upsilon\mathbf{p}\wedge\mathbf{p}$ is understood as the list of predicate expressions

$$\lambda\mathbf{x}(\upsilon p(\mathbf{x})\wedge p(\mathbf{x}))$$

for all $p\in\mathbf{p}$, and $\neg\upsilon\mathbf{p}\wedge\neg\mathbf{p}$ is understood in a similar way.[19]

Lemma 2
Formula

$$((\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})<(\mathbf{p},\neg\mathbf{p}))_{\Sigma 1}$$

is equivalent to $\upsilon\mathbf{p}\neq\mathbf{p}$.

Proof
In view of Lemma 1, $((\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})<(\mathbf{p},\neg\mathbf{p}))_{\Sigma 1}$ is equivalent to the disjunction of the formulas

$$\begin{array}{rl}\left(\bigwedge_{p\in\mathbf{p}}\forall\mathbf{x}(\upsilon p(\mathbf{x})\rightarrow p(\mathbf{x}))_{\Sigma 1}\right) & \wedge\quad \left(\bigwedge_{p\in\mathbf{p}}\forall\mathbf{x}(\upsilon\widehat{p}(\mathbf{x})\rightarrow\neg p(\mathbf{x}))_{\Sigma 1}\right)\\ & \wedge\quad \exists\mathbf{x}(\neg\upsilon p(\mathbf{x})\wedge\neg\upsilon\widehat{p}(\mathbf{x}))_{\Sigma 1}\end{array}\qquad(43)$$

for all $p\in\mathbf{p}$. It is easy to verify that

$$\begin{array}{rcl}(\upsilon p(\mathbf{x})\rightarrow p(\mathbf{x}))_{\Sigma 1} & = & (\upsilon p(\mathbf{x})\wedge p(\mathbf{x})\rightarrow p(\mathbf{x}))\ \Leftrightarrow\ \top\,,\\[2ex] (\upsilon\widehat{p}(\mathbf{x})\rightarrow\neg p(\mathbf{x}))_{\Sigma 1} & = & (\neg\upsilon p(\mathbf{x})\wedge\neg p(\mathbf{x})\rightarrow\neg p(\mathbf{x}))\ \Leftrightarrow\ \top,\\[2ex] (\neg\upsilon p(\mathbf{x})\wedge\neg\upsilon\widehat{p}(\mathbf{x}))_{\Sigma 1} & \Leftrightarrow & ((\neg\upsilon p(\mathbf{x})\vee\neg p(\mathbf{x}))\wedge\neg(\neg\upsilon p(\mathbf{x})\wedge\neg p(\mathbf{x})))\\ & \Leftrightarrow & (\upsilon p(\mathbf{x})\leftrightarrow\neg p(\mathbf{x}))\\ & \Leftrightarrow & \neg(\upsilon p(\mathbf{x})\leftrightarrow p(\mathbf{x}))\cdot\end{array}$$

Therefore (43) is equivalent to $\exists\mathbf{x}\neg(\upsilon p(\mathbf{x})\leftrightarrow p(\mathbf{x}))$, so that the disjunction of all formulas (43) is equivalent to $\upsilon\mathbf{p}\neq\mathbf{p}$. $\square$

If A is an atomic formula $p(\mathbf{t})$, where $p\in\mathbf{p}$ and $\mathbf{t}$ is a tuple of terms, then we will write υA for $\upsilon p(\mathbf{t})$, and $\widehat{A}$ for $\upsilon\widehat{p}(\mathbf{t})$. By $\widetilde{\forall}_{obj}F$ we denote the formula $\forall\mathbf{x}F$, where $\mathbf{x}$ is list of all free object variables of F ("object-level universal closure").

Define $H(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})$ to be the conjunction of the implications

$$\widetilde{\forall}_{obj}\left(G\rightarrow\bigvee_{A\in Pos}((\upsilon\widehat{A}\vee A)\rightarrow\upsilon A)\vee\bigvee_{A\in Neg}((\upsilon A\vee\neg A)\rightarrow\upsilon\widehat{A})\right)\qquad(44)$$

for all rules (38) in T.

Lemma 3
Formula $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi\wedge CC]$ is equivalent to

$$\Pi\wedge CC\wedge\forall(\upsilon\mathbf{p})(\upsilon\widehat{\mathbf{p}})(((\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})<(\mathbf{p},\neg\mathbf{p}))\rightarrow\neg H(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}}))\cdot\qquad(45)$$

[19] For the definition of $F^{\mathbf{p}}_{\upsilon\mathbf{p}}$ see Section 3.

Proof
Every occurrence of every intensional predicate in CC is in the scope of a negation. Consequently $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi \wedge CC]$ is

$$\Pi \wedge CC \wedge \neg\exists(\upsilon\mathbf{p})(\upsilon\widehat{\mathbf{p}})(((\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}}) < (\mathbf{p},\widehat{\mathbf{p}})) \wedge \Pi^{\diamond}(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}}) \wedge CC),$$

which is equivalent to

$$\Pi \wedge CC \wedge \forall(\upsilon\mathbf{p})(\upsilon\widehat{\mathbf{p}})(((\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}}) < (\mathbf{p},\neg\mathbf{p})) \to \neg\Pi^{\diamond}(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}}))\cdot$$

We will conclude the proof by showing that CC entails

$$\Pi^{\diamond}(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}}) \leftrightarrow H(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})\cdot$$

The left-hand side of this equivalence is the conjunction of the formulas

$$\widetilde{\forall}_{obj}\left(\neg\neg G \wedge \bigwedge_{A\in Pos}(\upsilon\widehat{A} \vee \neg\widehat{A}) \wedge \bigwedge_{A\in Neg}(\upsilon A \vee \neg A) \to \bigvee_{A\in Pos}\upsilon A \vee \bigvee_{A\in Neg}\upsilon\widehat{A}\right)$$

for all rules (38) in T. Under the assumption CC this formula can be rewritten as

$$\widetilde{\forall}_{obj}\left(G \to \bigvee_{A\in Pos}\neg(\upsilon\widehat{A} \vee A) \vee \bigvee_{A\in Neg}\neg(\upsilon A \vee \neg A) \vee \bigvee_{A\in Pos}\upsilon A \vee \bigvee_{A\in Neg}\upsilon\widehat{A}\right)\cdot$$

The last formula is equivalent to

$$\widetilde{\forall}_{obj}\left(G \to \bigvee_{A\in Pos}(\neg(\upsilon\widehat{A} \vee A) \vee \upsilon A) \vee \bigvee_{A\in Neg}(\neg(\upsilon A \vee \neg A) \vee \upsilon\widehat{A})\right)\cdot$$

and consequently to (44). $\square$

Lemma 4
$T^{\dagger}(\upsilon\mathbf{p})$ is equivalent to $H(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})_{\Sigma 1}$.

Proof
Formula $T^{\dagger}(\upsilon\mathbf{p})$ is the conjunction of the formulas

$$\widetilde{\forall}_{obj}\left(G \to \bigvee_{A\in Pos}\upsilon A \vee \bigvee_{A\in Neg}\neg\upsilon A\right) \tag{46}$$

for all rules (38) in T. On the other hand, $H(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})_{\Sigma 1}$ is the conjunction of the formulas

$$\widetilde{\forall}_{obj}\left(G \to \bigvee_{A\in Pos}((\upsilon\widehat{A} \vee A) \to \upsilon A)_{\Sigma 1} \vee \bigvee_{A\in Neg}((\upsilon A \vee \neg A) \to \upsilon\widehat{A})_{\Sigma 1}\right) \tag{47}$$

for all rules (38) in T. It remains to observe that

$$\begin{array}{rcl} ((\upsilon\widehat{A} \vee A) \to \upsilon A)_{\Sigma 1} & = & (\neg\upsilon A \wedge \neg A) \vee A \to \upsilon A \wedge A \\ & \Leftrightarrow & \neg\upsilon A \vee A \to \upsilon A \wedge A \\ & \Leftrightarrow & (\upsilon A \wedge \neg A) \vee (\upsilon A \wedge A) \\ & \Leftrightarrow & \upsilon A, \end{array}$$

and that, similarly, $((\upsilon A \vee \neg A) \to \upsilon \widehat{A})_{\Sigma 1}$ is equivalent to $\neg \upsilon A$. □

Lemma 5
$\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi \wedge CC] \models T \wedge CC$.

Proof
Recall that, according to Lemma 3, $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi \wedge CC]$ is equivalent to (45). The second conjunctive term of (45) is CC. The first conjunctive term is equivalent to $T^{\dagger}(\mathbf{p})$. From the other two terms we conclude:

$$\forall \upsilon \mathbf{p}(((\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}}) < (\mathbf{p}, \widehat{\mathbf{p}}))_{\Sigma 1} \to \neg H(\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}})_{\Sigma 1})\cdot$$

By Lemma 2 and Lemma 4, this formula is equivalent to

$$\forall \upsilon \mathbf{p}((\upsilon \mathbf{p} \neq \mathbf{p}) \to \neg T^{\dagger}(\upsilon \mathbf{p})),$$

and consequently to

$$\forall \upsilon \mathbf{p}(T^{\dagger}(\upsilon \mathbf{p}) \to (\upsilon \mathbf{p} = \mathbf{p}))\cdot$$

The conjunction of the last formula with $T^{\dagger}(\mathbf{p})$ is equivalent to (24). □

For any formula F, by $F_{\Sigma 2}$ we denote the formula

$$F^{\upsilon \mathbf{p}}_{(((\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg \mathbf{p})) \wedge \neg \upsilon \mathbf{p} \wedge \neg \upsilon \widehat{\mathbf{p}}) \leftrightarrow \neg \mathbf{p}}$$

where the subscript

$$(((\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg \mathbf{p})) \wedge \neg \upsilon \mathbf{p} \wedge \neg \upsilon \widehat{\mathbf{p}}) \leftrightarrow \neg \mathbf{p}$$

is understood as the list of predicate expressions

$$\lambda \mathbf{x}((((\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg \mathbf{p})) \wedge \neg \upsilon p(\mathbf{x}) \wedge \neg \upsilon \widehat{p}(\mathbf{x})) \leftrightarrow \neg p(\mathbf{x}))$$

for all $p \in \mathbf{p}$.

Lemma 6
Formula

$$(\upsilon \mathbf{p} \neq \mathbf{p})_{\Sigma 2}$$

is equivalent to $(\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}}) < (\mathbf{p}, \neg \mathbf{p})$.

Proof
Formula $(\upsilon \mathbf{p} \neq \mathbf{p})_{\Sigma 2}$ is equivalent to

$$\bigvee_{p \in \mathbf{p}} \exists \mathbf{x}(\upsilon p(\mathbf{x}) \leftrightarrow \neg p(\mathbf{x}))_{\Sigma 2}$$

that is,

$$\bigvee_{p \in \mathbf{p}} \exists \mathbf{x}((((\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg \mathbf{p})) \wedge \neg \upsilon p(\mathbf{x}) \wedge \neg \upsilon \widehat{p}(\mathbf{x}) \leftrightarrow \neg p(\mathbf{x})) \leftrightarrow \neg p(\mathbf{x}))\cdot$$

This formula can be equivalently rewritten as

$$\bigvee_{p \in \mathbf{p}} (((\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg \mathbf{p})) \wedge \exists \mathbf{x}(\neg \upsilon p(\mathbf{x}) \wedge \neg \upsilon \widehat{p}(\mathbf{x}))),$$

which is equivalent to $(\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}}) < (\mathbf{p}, \neg \mathbf{p})$ by Lemma 1. □

Lemma 7
The implication

$$(\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg\mathbf{p}) \rightarrow (\,T^{\dagger}(\upsilon\mathbf{p})_{\Sigma 2} \leftrightarrow H(\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}))$$

is logically valid.

Proof
Recall that $T^{\dagger}(\upsilon\mathbf{p})$ is the conjunction of implications (46) for all rules (38) in T. Consequently $T^{\dagger}(\upsilon\mathbf{p})_{\Sigma 2}$ is the conjunction of the formulas

$$\widetilde{\forall}_{obj}\left(G \rightarrow \bigvee_{A\in Pos}(\upsilon A)_{\Sigma 2} \vee \bigvee_{A\in Neg}\neg(\upsilon A)_{\Sigma 2}\right),$$

that is to say,

$$\begin{aligned}\widetilde{\forall}_{obj}(G \rightarrow \quad & \textstyle\bigvee_{A\in Pos}((((\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg\mathbf{p})) \wedge \neg\upsilon A \wedge \neg\upsilon\widehat{A}) \leftrightarrow \neg A) \vee \\ & \textstyle\bigvee_{A\in Neg}\neg((((\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg\mathbf{p})) \wedge \neg\upsilon A \wedge \neg\upsilon\widehat{A}) \leftrightarrow \neg A)\cdot\end{aligned}$$

Under the assumption

$$(\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) \leq (\mathbf{p}, \neg\mathbf{p}) \tag{48}$$

the last formula can be equivalently rewritten as

$$\widetilde{\forall}_{obj}\left(G \rightarrow \bigvee_{A\in Pos}((\upsilon A \vee \upsilon\widehat{A}) \leftrightarrow A) \vee \bigvee_{A\in Neg}((\upsilon A \vee \upsilon\widehat{A}) \leftrightarrow \neg A)\right)\cdot$$

It remains to check that, under assumption (48),

$$(\upsilon A \vee \upsilon\widehat{A}) \leftrightarrow A \tag{49}$$

can be equivalently rewritten as

$$\upsilon\widehat{A} \vee A \rightarrow \upsilon A, \tag{50}$$

and

$$\upsilon A \vee \upsilon\widehat{A} \leftrightarrow \neg A \tag{51}$$

can be rewritten as

$$\upsilon A \vee \neg A \rightarrow \upsilon\widehat{A}\cdot \tag{52}$$

Formula (49) is equivalent to

$$(\upsilon A \rightarrow A) \wedge (\upsilon\widehat{A} \rightarrow A) \wedge (A \rightarrow \upsilon A \vee \upsilon\widehat{A})\cdot \tag{53}$$

Since assumption (48) entails $\upsilon A \rightarrow A$ and $\upsilon\widehat{A} \rightarrow \neg A$, formula (53) can be rewritten as

$$\neg\upsilon\widehat{A} \wedge (A \rightarrow \upsilon A)\cdot \tag{54}$$

On the other hand, formula (50) is equivalent to

$$(\upsilon\widehat{A} \rightarrow \upsilon A) \wedge (A \rightarrow \upsilon A),$$

which, under assumption (48), can be rewritten as (54) as well. In a similar way, each of the formulas (51), (52) can be transformed into

$$\neg \upsilon A \wedge (\neg A \rightarrow \upsilon \widehat{A}) \cdot \quad \Box$$

Lemma 8
$T \wedge CC \models \mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi \wedge CC]$.

Proof
Recall that T is equivalent to

$$T^{\dagger}(\mathbf{p}) \wedge \forall \upsilon \mathbf{p} (T^{\dagger}(\upsilon \mathbf{p}) \rightarrow (\upsilon \mathbf{p} = \mathbf{p})) \cdot \tag{55}$$

Since the first conjunctive term is equivalent to Π, $T \wedge CC$ entails

$$\Pi \wedge CC \cdot \tag{56}$$

From the second conjunctive term of (55) we conclude

$$T^{\dagger}(\upsilon \mathbf{p})_{\Sigma 2} \rightarrow (\upsilon \mathbf{p} = \mathbf{p})_{\Sigma 2}$$

and consequently

$$\forall (\upsilon \mathbf{p})(\upsilon \widehat{\mathbf{p}})((\upsilon \mathbf{p} \neq \mathbf{p})_{\Sigma 2} \rightarrow \neg T^{\dagger}(\upsilon \mathbf{p})_{\Sigma 2}) \cdot$$

By Lemma 6, this is equivalent to

$$\forall (\upsilon \mathbf{p})(\upsilon \widehat{\mathbf{p}})(((\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}}) < (\mathbf{p}, \neg \mathbf{p})) \rightarrow \neg T^{\dagger}(\upsilon \mathbf{p})_{\Sigma 2})$$

and, by Lemma 7, to

$$\forall (\upsilon \mathbf{p})(\upsilon \widehat{\mathbf{p}})(((\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}}) < (\mathbf{p}, \neg \mathbf{p})) \rightarrow \neg H(\upsilon \mathbf{p}, \upsilon \widehat{\mathbf{p}})) \cdot$$

By Lemma 3, the conjunction of this formula with (56) is equivalent to sentence $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi \wedge CC]$. $\Box$

Assertion (41) follows from Lemmas 5 and 8.

7.2 General Case

Lemma 9
For any C-rule R, $\mathrm{Tr}_c[R]$ is intuitionistically equivalent to $\mathrm{Tr}_d[R]$.

Proof
If R is $\bot \Leftarrow G$ then $\mathrm{Tr}_c[R]$ is $\widetilde{\forall} \neg G$, and $\mathrm{Tr}_d[R]$ is $\widetilde{\forall}(\neg\neg G \rightarrow \bot)$. $\Box$

Lemma 10
For any L-rule R, the conjunction CC of completeness constraints intuitionistically entails

$$\mathrm{Tr}_l[R] \leftrightarrow \mathrm{Tr}_d[R] \cdot$$

Proof
If R is $p(\mathbf{t}) \Leftarrow G$ then $\mathrm{Tr}_l[R]$ is

$$\widetilde{\forall}(\neg\neg G \rightarrow p(\mathbf{t})),$$

and $\mathrm{Tr}_d[R]$ is

$$\widetilde{\forall}(\neg\neg G \wedge (\widehat{p}(\mathbf{t}) \vee \neg\widehat{p}(\mathbf{t})) \rightarrow p(\mathbf{t}))\cdot$$

Since CC intuitionistically entails

$$\neg(p(\mathbf{t}) \leftrightarrow \widehat{p}(\mathbf{t})), \tag{57}$$

it is sufficient to check that $p(\mathbf{t})$ can be derived from (57) and

$$\widehat{p}(\mathbf{t}) \vee \neg\widehat{p}(\mathbf{t}) \rightarrow p(\mathbf{t}) \tag{58}$$

by the deductive means of intuitionistic propositional logic. Since (58) is equivalent to $p(\mathbf{t})$ in classical propositional logic, it is easy to see that $\neg\widehat{p}(\mathbf{t})$ can be derived from (57) and (58) in classical propositional logic. By Glivenko's theorem,[20] it follows that it can be derived intuitionistically as well. Since $p(\mathbf{t})$ is intuitionistically derivable from (58) and $\neg\widehat{p}(\mathbf{t})$, we can conclude that $p(\mathbf{t})$ is intuitionistically derivable from (57) and (58).

The case when R is $\neg p(\mathbf{t}) \Leftarrow G$ is similar. $\square$

Lemma 11
If R is an S-rule

$$L_1 \leftrightarrow L_2 \Leftarrow G \tag{59}$$

and R_1, R_2 are the D-rules

$$L_1 \vee \overline{L_2} \Leftarrow G \quad \text{and} \quad \overline{L_1} \vee L_2 \Leftarrow G \tag{60}$$

then the conjunction CC of completeness constraints intuitionistically entails

$$\mathrm{Tr}_s[R] \leftrightarrow \mathrm{Tr}_d[R_1] \wedge \mathrm{Tr}_d[R_2]\cdot$$

Proof
If each of the literals L_i is an atom A_i then $\mathrm{Tr}_s[R]$ is the conjunction of the formulas

$$\begin{array}{l} \widetilde{\forall}(\neg\neg G \wedge A_1 \rightarrow A_2), \\ \widetilde{\forall}(\neg\neg G \wedge A_2 \rightarrow A_1), \\ \widetilde{\forall}(\neg\neg G \wedge \widehat{A_1} \rightarrow \widehat{A_2}), \\ \widetilde{\forall}(\neg\neg G \wedge \widehat{A_2} \rightarrow \widehat{A_1}), \end{array} \tag{61}$$

$\mathrm{Tr}_d[R_1]$ is

$$\widetilde{\forall}(\neg\neg G \wedge (\widehat{A_1} \vee \neg\widehat{A_1}) \wedge (A_2 \vee \neg A_2) \rightarrow A_1 \vee \widehat{A_2}), \tag{62}$$

[20] This theorem (Glivenko 1929), (Mints 2000, Theorem 3.1) asserts that if a formula beginning with negation can be derived from a set Γ of formulas in classical propositional logic then it can be derived from Γ in intuitionistic propositional logic as well.

and $\mathrm{Tr}_d[R_2]$ is

$$\widetilde{\forall}(\neg\neg G \wedge (A_1 \vee \neg A_1) \wedge (\widehat{A_2} \vee \neg\widehat{A_2}) \rightarrow \widehat{A_1} \vee A_2)\cdot \tag{63}$$

We need to show that CC intuitionistically entails the equivalence between the conjunction of formulas (61) and the conjunction of formulas (62), (63). Since CC intuitionistically entails

$$\neg(A_1 \leftrightarrow \widehat{A_1}) \tag{64}$$

and

$$\neg(A_2 \leftrightarrow \widehat{A_2}), \tag{65}$$

it is sufficient to check that the conjunction of formulas (64), (65),

$$A_1 \leftrightarrow A_2 \tag{66}$$

and

$$\widehat{A_1} \leftrightarrow \widehat{A_2} \tag{67}$$

is equivalent in intuitionistic propositional logic to the conjunction of formulas (64), (65),

$$(\widehat{A_1} \vee \neg\widehat{A_1}) \wedge (A_2 \vee \neg A_2) \rightarrow A_1 \vee \widehat{A_2} \tag{68}$$

and

$$(A_1 \vee \neg A_1) \wedge (\widehat{A_2} \vee \neg\widehat{A_2}) \rightarrow \widehat{A_1} \vee A_2\cdot \tag{69}$$

Left-to-right: Assume (64)–(67) and

$$(\widehat{A_1} \vee \neg\widehat{A_1}) \wedge (A_2 \vee \neg A_2); \tag{70}$$

our goal is to derive intuitionistically $A_1 \vee \widehat{A_2}$. Consider two cases, in accordance with the first disjunction in (70). *Case 1:* $\widehat{A_1}$. Then, by (67), $\widehat{A_2}$, and consequently $A_1 \vee \widehat{A_2}$. *Case 2:* $\neg\widehat{A_1}$. Consider two cases, in accordance with the second disjunction in (70). *Case 2.1:* A_2. Then, by (66), A_1, and consequently $A_1 \vee \widehat{A_2}$. *Case 2.2:* $\neg A_2$. Then, by (66), $\neg A_1$, which contradicts (64).

Thus we proved that (68) is intuitionistically derivable from (64)–(67). The proof for (69) is similar.

Right-to-left: Let Γ be the set consisting of formulas (64), (65), (68), (69) and A_1. We claim that A_2 can be derived from Γ in intuitionistic propositional logic. Note that, classically,

- Formula (64) is equivalent to $A_1 \leftrightarrow \neg\widehat{A_1}$,
- Formula (65) is equivalent to $A_2 \leftrightarrow \neg\widehat{A_2}$, and
- Formula (69) is equivalent to $\widehat{A_1} \vee A_2$.

It follows that $\neg\widehat{A_2}$ is derivable from Γ in classical propositional logic. By Glivenko's theorem, it follows that $\neg\widehat{A_2}$ is derivable from Γ intuitionistically as well. Hence the antecedent of (69) is an intuitionistic consequence of Γ, and so is the consequent $\widehat{A_1} \vee A_2$. In combination with A_1 and (64), this gives us A_2.

We conclude that $A_1 \rightarrow A_2$ is intuitionsistically derivable from (64), (65), (68) and (69). The derivability of the implication $A_2 \to A_1$ from these formulas can be proved in a similar way. Thus (66) is an intuitionistic consequence of (64), (65), (68), and (69).

The derivability of (67) from these formulas in propositional intuitionistic logic is proved in a similar way.

The cases when the literals L_i are negative, or when one of them is positive and the other is negative, are similar. $\square$

Proof of the soundness property (40). Let C, L, S, and D be sets of causal rules of types C, L, S, and D respectively, and let T be the causal theory with the set of rules $C \cup L \cup S \cup D$. Consider the causal theory T' obtained from T by replacing each rule (59) from S with the corresponding rules (60). According to the result (41) of Section 7.1,

$$\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi \wedge CC] \text{ is equivalent to } T' \wedge CC,$$

where Π is the conjunction of the program rules $\mathrm{Tr}_d[R]$ for all rules R of T'. It is clear that $\Pi \wedge CC$ is $\mathrm{Tr}[T']$, and that T' is equivalent to T. Consequently

$$\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\mathrm{Tr}[T']] \text{ is equivalent to } T \wedge CC\cdot \tag{71}$$

On the other hand, Lemmas 9, 10 and 11 show that the formulas $\mathrm{Tr}[T']$ and $\mathrm{Tr}[C, L, S, D]$ are intuitionistically equivalent to each other, because each of them contains CC as a conjunctive term. It follows that

$$\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\mathrm{Tr}[T']] \text{ is equivalent to } \mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\mathrm{Tr}[C, L, S, D]]\cdot \tag{72}$$

Assertion (40) follows from (71) and (72).

8 Conclusion

In this paper we generalized McCain's embedding of definite causal theories into logic programming. We expect that this work will provide a theoretical basis for extending the system COALA to more expressive action languages, including the modular action language MAD (Ren 2009). It is essential, from this perspective, that our translation is applicable to synonymity rules, because such rules are closely related to the main new feature of MAD, its **import** construct.

Our translation is not applicable to causal rules with quantifiers in the head. It may be possible to extend it to positive occurrences of existential quantifiers, since an existentially quantified formula can be thought of as an infinite disjunction. But the translation would be a formula with positive occurrences of existential quantifiers as well, and it is not clear how to turn such a formula into executable code.

In the future, we would like to extend the translation described above to causal theories with explainable function symbols, which correspond to non-Boolean fluents in action languages. Since the definition of a stable model does not allow function symbols to be intensional, such a generalization would have to involve extending the language by auxiliary predicate symbols.

Acknowledgements

We are grateful to the anonymous referees for useful comments. Joohyung Lee was partially supported by the National Science Foundation under grant IIS-0916116 and by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), through US Army. Yuliya Lierler was supported by a 2010 Computing Innovation Fellowship. Vladimir Lifschitz and Fangkai Yang were supported by the National Science Foundation under grant IIS-0712113. All statements of fact, opinion or conclusions contained herein are those of the authors and should not be construed as representing the official views or policies of IARPA, the ODNI or the U.S. Government.

References

Akman, V., Erdoğan, S., Lee, J., Lifschitz, V., and Turner, H. 2004. Representing the Zoo World and the Traffic World in the language of the Causal Calculator. *Artificial Intelligence 153(1–2)*, 105–140.

Armando, A., Giunchiglia, E., and Ponta, S. E. 2009. Formal specification and automatic analysis of business processes under authorization constraints: an action-based approach. In *Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business (TrustBus'09)*.

Artikis, A., Sergot, M., and Pitt, J. 2009. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic 9,* 1.

Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., and Patoglu, V. 2009. Bridging the gap between high-level reasoning and low-level control. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. 242–354.

Clark, K. 1978. Negation as failure. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds. Plenum Press, New York, 293–322.

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

Ferraris, P. 2006. Causal theories as logic programs. In *Proceedings of Workshop on Logic Programming (WLP)*. 35–44.

Ferraris, P. 2007. A logic program characterization of causal theories. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*. 366–371.

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

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

Gebser, M., Grote, T., and Schaub, T. 2010. Coala: a compiler from action languages to ASP. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*. 169–181.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of International Logic Programming Conference and Symposium*, R. Kowalski and K. Bowen, Eds. MIT Press, 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

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

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

Glivenko, V. 1929. Sur quelques points de la logique de M. Brouwer. *Académie Royale de Belgique. Bulletins de la Classe des Sciences, se'rie 5 15*, 183–188.

Lee, J., Lierler, Y., Lifschitz, V., and Yang, F. 2010. Representing synonymity in causal logic and in logic programming[21]. In *Proceedings of International Workshop on Nonmonotonic Reasoning (NMR)*.

Lee, J. and Palla, R. 2009. System F2LP — computing answer sets of first-order formulas. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. 515–521.

Lifschitz, V. 1985. Computing circumscription. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*. 121–127.

Lifschitz, V. 1994. Circumscription. In *Handbook of Logic in AI and Logic Programming*, D. Gabbay, C. Hogger, and J. Robinson, Eds. Vol. 3. Oxford University Press, 298–352.

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

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

Lifschitz, V. and Ren, W. 2006. A modular action description language. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*. 853–859.

Lifschitz, V. and Ren, W. 2007. The semantics of variables in action descriptions. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*. 1025–1030.

Lifschitz, V. and Yang, F. 2010. Translating first-order causal theories into answer set programming. In *Proceedings of the European Conference on Logics in Artificial Intelligence (JELIA)*. 247–259.

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

McCain, N. 1997. Causality in commonsense reasoning about actions[22]. Ph.D. thesis, University of Texas at Austin.

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

McCarthy, J. 1986. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence 26,* 3, 89–116.

Mints, G. 2000. *A Short Introduction to Intuitionistic Logic*. Kluwer.

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

Ren, W. 2009. A modular language for describing actions[23]. Ph.D. thesis, University of Texas at Austin.

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

[21] `http://userweb.cs.utexas.edu/users/vl/papers/syn.pdf`
[22] `ftp://ftp.cs.utexas.edu/pub/techreports/tr97-25.ps.gz`
[23] `http://www.cs.utexas.edu/users/rww6/dissertation.pdf`