ID-logic and the Ramification Problem for the Situation Calculus

Eugenia Ternovskaia [1]

Abstract. The goal of this paper is to extend the general solution to the ramification problem for the situation calculus to the case where causal rules specifying indirect effects of actions may form cycles or cycles through negation. We formulate causal dependencies as rules of inductive definitions. Then we give a semantics to a logic for representing inductive definitions (ID-logic) by translating definitions to sentences of fixed-point logic. We describe a regression-based procedure for generating successor state axioms from inductive rules, and consider several benchmark examples. Finally, we show that boundedness is a sufficient condition for translating causal theories to successor state axioms.

1 INTRODUCTION

The ramification problem arises in the context of knowledge representation, when one wants to capture indirect effects of actions in a logic-based formalism. It has been shown (e.g., [11]) that state constraints are generally inadequate for deriving indirect effects of actions, and that some notion of causation is needed. Unlike material implication, causal implications are not contrapositive which makes them similar to the rules of logic programs. This similarity shows most strongly in two directions of research. The work of McCain and Turner on causal logic [12], developed further by Lifschitz [9], bears close connections with Reiter's default logic; its implementation in logic programming combines classical negation with negation as failure under answer set semantics.(cf. [10]). The other line of research relates causal rules with stratified [14] and well-founded [6] semantics for logic programming. McIlraith [14] interprets the implication connective as causal or definitional, and describes a procedure for compiling successor state axioms from a set of effect axioms and a set of ramification axioms. The solution applies to acyclic stratified theories. The authors of [6] concentrate on the semantical aspects of the ramification problem, and formulate direct and indirect effects of actions as rules of inductive definitions. For collections of such definitions, they introduce the term "causal theories". The meaning of causal theories is given by the well-founded generalization of inductive definitions [4]. The authors conjectured that inductive rules can be translated to successor state axioms of the situation calculus. The following questions needed to be answered. (1) Since the well-founded semantics is three-valued, there are causal theories in which some literals have the third truth value "undefined". Is there a counterpart of such causal theories in classical two-valued logic? (2) It is well-known that, in general, recursion cannot be represented in first-order logic. On the other hand, successor state axioms are first order. Does this imply that causal theories cannot be translated to the situation calculus?

The answer to the first question is positive. Fortunately, there is a way to capture well-founded semantics in classical logic. We give the corresponding definition in the following section. The truth set of a defined predicate is expressed by a formula with nested fixed-point operators which are expressible by second-order circumscription [13] formulas. Regarding the second question, we demonstrate that it is possible to provide a translation of causal theories to successor state axioms for the class of bounded causal theories, i.e., those which are equivalent to recursion-free sets of rules.

For several years, Denecker has been arguing for the use of inductive definitions for knowledge representation. Recently, he proposed a special logic for these purposes [5]. He terms it the logic for inductive definitions, or ID-logic. The syntax of ID-logic is the syntax of first- or second-order logic, combined with formulas for representing inductive definitions. The semantics for the general case of non-propositional definitions is presented as a generalization of the propositional case, using so-called grounding technique. In this paper, we develop the syntax of ID-logic further and define the semantics of the logic by translating its formulas to the formulas of fixed-point logic. The translation demonstrates that definitions add at most the power of fixed-point logic to the original language, before definitions.

This paper is organized as follows. First, in Section 2, we describe a logic for representing inductive definitions, and define its semantics. Section 3 is devoted to the situation calculus with definitions. In Section 4, we provide a procedure for translating causal rules to successor state axioms. To this end, we redefine regression as applying causal rules "backwards". This process is similar to top-down evaluation of logic programs. In the case where static causal rules (ramifications) are not present, our regression is essentially the same as Reiter's original definition. We consider several benchmark examples, and formulate a general theorem about translating causal theories to successor state axioms of the situation calculus. This theorem extends our previous results [18]. Finally, in Section 5, we discuss the possibility of a computationally useful characterization of which causal theories are representable in the situation calculus, and in first-order logic in general. This general characterization is impossible. This conclusion follows from the fact that boundedness is undecidable for database query languages [8]. There are, however, classes of causal theories where boundedness is decidable.

2 A LOGIC FOR INDUCTIVE DEFINITIONS

Fix a vocabulary τ. A *definition* Δ *in vocabulary* τ is a set of rules of the form $l \leftarrow \Psi$. The formula l must be of the form $R(\bar{t})$ where R is a predicate symbol from τ, $\bar{t}$ is a tuple of first-order terms. It is called the *head* of the rule. The formula Ψ constitutes the *body* of the

[1] Computer Science Department, University of Toronto, Canada, email: eugenia@cs.toronto.edu

rule. Every relation symbol occurring in the head of some rule of Δ is *defined*; all other symbols in τ are *open*. In database theory, defined and open relation symbols are called intentional and extensional, respectively. Denote by $(\tau,\Delta)_{\mathrm{d}}$ and $(\tau,\Delta)_{\mathrm{o}}$ the set of defined and open symbols of Δ. Hence, $(\tau,\Delta)_{\mathrm{o}} = \tau \setminus (\tau,\Delta)_{\mathrm{d}}$. We often write τ_{d} and τ_{o} if no confusion is possible. The values of the open symbols are given by a τ_{o}-structure. All the rules of Δ will be applied simultaneously to generate consecutive stages of the defined symbols. For instance, on the structure $\mathbb{N}$ of natural numbers, Δ_1 is a definition of the form $\{Z(0) \leftarrow, P(x) \leftarrow \neg Z(x)\}$ which defines zero and the positive natural numbers.

Sometimes the applications of the rules of a definition, say Δ_2, must be delayed until the fixed-point of another definition, say Δ_1, is reached. In this case, we write $\Delta_1 \cdot \Delta_2$. In general, the result of evaluation changes with the order of definitions. If Δ_1 and Δ_2 are to be evaluated in parallel, we write $\Delta_1 + \Delta_2$. If Δ_1 and Δ_2 are definitions then so are $\Delta_1 \cdot \Delta_2$ and $\Delta_1 + \Delta_2$. We agree to have $(\tau, \Delta_1 \cdot \Delta_2)_{\mathrm{d}} = (\tau,\Delta_1)_{\mathrm{d}} \cup (\tau,\Delta_2)_{\mathrm{d}}$ and $(\tau, \Delta_1 + \Delta_2)_{\mathrm{d}} = (\tau,\Delta_1)_{\mathrm{d}} \cup (\tau,\Delta_2)_{\mathrm{d}}$, and we require $(\tau,\Delta_1)_{\mathrm{d}} \cap (\tau,\Delta_2)_{\mathrm{d}} = \emptyset$ whenever Δ_1 and Δ_2 are to be evaluated in sequence or in parallel.

We obtain ID-logic by closing first-order (or second-order) logic under inductive definitions. For a vocabulary τ, the class $\mathrm{ID}[\tau]$ of formulas of ID-logic is defined recursively as follows. (The notation $\frac{\phi_1,\ldots,\phi_n}{\phi}$ stands for "If $\phi_1,\ldots,\phi_n$ are formulas then ϕ is a formula".)

- $\frac{}{\phi}$ where ϕ is an atomic first-order (resp., second order) formula over τ
- $\frac{}{[\Delta,P](\bar{x})}$ where Δ is a definition in vocabulary τ, $P \in (\tau,\Delta)_{\mathrm{d}}$, and the length of tuple $\bar{x}$ coincides with the arity of P
- $\frac{\phi}{\neg\phi}$; $\frac{\phi,\psi}{\phi\vee\psi}$; $\frac{\phi}{\exists x\phi}$ where ϕ and ψ are atomic first-order (resp., second order) formulas over τ.

Intuitively, the meaning of formula $[\Delta,P](\bar{t})$ in structure $\mathcal{A}$ is "$\bar{t}^{\mathcal{A}}$ belongs to a relation determined by definition Δ in structure $\mathcal{A}$ and associated with defined symbol P". For example, $[\Delta_1, Z](0)$ where Δ_1 is as above is true on the structure $\mathbb{N}$ of natural numbers. The formal semantics of ID-logic is defined below. Note that if $P \in (\tau,\Delta_1)_{\mathrm{d}}$ and $P \in (\tau,\Delta_2)_{\mathrm{d}}$ and the definition Δ_1 is different from Δ_2 then $[\Delta_1,P]$ and $[\Delta_2,P]$ denote two different relations.

Let $\phi(\bar{x},X)$ be a formula in the vocabulary τ, where $\bar{x} = (x_1,\ldots,x_k)$, X is a relation variable of arity k, and let $\mathcal{A}$ be a τ-structure. Then ϕ and $\mathcal{A}$ give rise to an operation $\Gamma^{\phi} : \mathrm{Pow}(A^k) \to \mathrm{Pow}(A^k)$ defined by $\Gamma^{\phi}(R) := \{\bar{a} \mid \mathcal{A} \models \phi[\bar{a},R]\}$ and $\bar{a} = (a_1,\ldots,a_k)$. Note that $\mathcal{A} \models \phi[\sigma]$ means "assignment σ *satisfies* formula ϕ in $\mathcal{A}$". If $\phi = \phi(\bar{x},X)$, $\bar{a} = \sigma(\bar{x})$ and $R = \sigma[X]$, we write $\mathcal{A} \models \phi[\bar{a},R]$. We say that an occurrence of a predicate symbol in a formula is *negative* (respectively, *positive*) if it is within the scope of an odd (respectively, even) number of negations. If all occurrences of X in $\phi(\bar{x},X)$ are positive, then the operator Γ^{ϕ} is monotone and its least fixed-point is defined by transfinite induction (cf. [15, 2]): $\Gamma^{\phi}_{(\infty)} := \bigcup_{n\geq 0}\Gamma^{\phi}_{(n)}$ where $\Gamma^{\phi}_{(n)} := \{\bar{a} \mid \text{there is } \bar{b} \in A \text{ such that } (\mathcal{A},\Gamma^{\phi}_{<n}) \models \phi[\bar{a},\bar{b}] \text{ and } \Gamma^{\phi}_{<n} := \bigcup_{j<n}\Gamma^{\phi}_{(j)}.\}$

To simplify the presentation of the semantics, we suppose that for any defined symbol P of Δ, say, of arity r, there are distinct variables $\bar{x}_P = x_{P,1}\ldots x_{P,r}$ such that any rule in Δ with head symbol P has the form $P(\bar{x}_P) \leftarrow \Psi$. With every defined P we associate the formula

$$\phi_P(\bar{x}_P) := \bigvee\{\exists\bar{y}\Psi \mid P(\bar{x}_P) \leftarrow \Psi \text{ in } \Delta\}, \tag{1}$$

where $\bar{y}$ contains the variables in Ψ that are distinct from the variables in $\bar{x}_P$. A *positive definition* is a definition without negative occurrences of defined symbols in the body of any rule.

We shall describe the semantics of positive definitions first. Assume that $P^1,\ldots\ P^m$ are the defined symbols of Δ. By definition of a positive definition they do not occur negated in the body of any rule. Let $\mathcal{A}$ be a τ_{o}-structure. Define

$$\begin{aligned} P^i_{(n)} := \{\bar{a} \mid{} & \text{there are } P(\bar{x}_P) \leftarrow \Psi \text{ in } \Delta \text{ and } \bar{b} \in A \text{ such that} \\ & (\mathcal{A}, P^1_{<n},\ldots,P^m_{<n}) \models \Psi[\bar{a},\bar{b}] \text{ and } P^i_{<n} := \textstyle\bigcup_{j<n} P^i_{(j)}.\} \end{aligned}$$

(where $\bar{a}$ interprets the variables in $\bar{x}_{P^i}$, and $\bar{b}$ interprets those remaining). Since $\phi_{P^1},\ldots,\phi_{P^m}$ are positive in $P^1,\ldots,P^m$, their simultaneous fixed-point $(P^1_{(\infty)},\ldots,P^m_{(\infty)})$ given by $P^i_{(\infty)} := \bigcup_{n\geq 0} P^i_{(n)}$ exists. Thus a positive definition Δ and a τ_o structure $\mathcal{A}$ give rise to the τ-structure $(\mathcal{A}, P^1_{(\infty)},\ldots P^k_{(\infty)})$, where $P^1_{(\infty)},\ldots P^k_{(\infty)}$ are the defined relations. This extended structure provides the semantics for Δ.

Now we are ready to define the semantics for a general definition. First we provide some intuition by discussing a procedure for determining the so-called truth set of a defined predicate. This procedure generalizes the procedure originally proposed by Van Gelder [20]. Let Δ be a general definition. For every defined symbol P, replace all positive occurrences of P in the bodies by a new relation symbol P'. Now, the original P does not occur in the head of any rule of the resulting definition Δ'; hence, P is an open symbol of Δ', and thus, Δ' is a positive definition. In order to simplify the presentation, we assume that Δ only contains a single defined relation symbol P. In every τ_{o}-structure $\mathcal{A}$, the definition Δ gives rise to a sequence $(P_n)_{n\geq 0}$ of relations on A defined by

> $P_0 := \emptyset$,
> P_n is the result for P' of the evaluation of the definition Δ' in $(\mathcal{A}, P_{\prec n})$, i.e., taking $P_{\prec n}$ as interpretation of the open symbol P of Δ'.

(where $P_{\prec n} = \bigcup\limits_{2k<n} P_{2k}$ if n is odd, $\bigcap\limits_{2k+1<n} P_{2k+1}$ if n is even).

Example 1 (renaming). In the following definition, renaming selected occurrences of P by P', as described above,

$$\begin{array}{lll} \Delta_0: & & P(x) \leftarrow S(x,y,z) \wedge R(x,u) \wedge \neg P(y) \\ & & P(x) \leftarrow \neg R(y,y) \wedge P(x) \wedge \neg P(u), \end{array}$$

produces

$$\begin{array}{lll} \Delta'_0: & & P'(x) \leftarrow S(x,y,z) \wedge R(x,u) \wedge \neg P(y) \\ & & P'(x) \leftarrow \neg R(y,y) \wedge P'(x) \wedge \neg P(y) \end{array}$$

Let $\psi(x,Q)$ be a first-order formula positive in Q, Γ^{ψ} be the monotone operator associated with it. Then $[\mathrm{LFP}_{x,Q}(\psi(x,Q))]$ denotes the least fixed-point of Γ^{ψ}. In the definition Δ_0, the stages P_n of the evaluation of Δ_0 are the stages Γ^{ϕ}_n of Γ^{ϕ}, where

$$\begin{aligned} \phi(x,P) := {} & [\mathrm{LFP}_{x,P'}(\exists y\exists z\exists u S(x,y,z) \wedge R(x,u) \wedge \neg P(y)) \\ & \vee \exists y \neg R(y,y) \wedge P'(x) \wedge \neg P(y)](x). \end{aligned}$$

Consider again the general τ_{o}-structure $\mathcal{A}$ and the sequence $(P_n)_{n\geq 0}$ of relations on A. Define the *truth set* of this sequence as

$$P^{\Delta}_{\mathrm{true}} := \{a \in A \mid \exists n_0 \forall n \geq n_0 \ : \ a \in P_n\}.$$

Then Δ leads to the τ-structure $(\mathcal{A}, P^{\Delta}_{\mathrm{true}})$. Note that $P^{\Delta}_{\mathrm{true}} = P_{\infty}$ if the fixed-point P_{∞} of the sequence $(P_n)_{n\geq 0}$ exists. The extended structure $(\mathcal{A}, P^{\Delta}_{\mathrm{true}})$ provides the semantics for the general definition Δ. For terms $\bar{t}$, the meaning of formula $[\Delta,P](\bar{t})$ is "$\bar{t}^{\mathcal{A}} \in P^{\Delta}_{\mathrm{true}}$".

We say that Δ is *total* if the fixed-point P_{∞} of the sequence $(P_n)_{n\geq 0}$ exists for all τ_{o}-structures $\mathcal{A}$. Note that we cannot restrict the language to allow total definitions only. By doing so, we would obtain an undecidable syntax. To see this, notice that monotonicity of Γ^ϕ is undecidable. Thus, there is no algorithm to determine whether the fixed-point of Γ^ϕ exists; therefore the question of whether a definition is total is undecidable. A general definition is guaranteed to be total if it is positive or stratified. Let FO+LFP denote the closure of first-order logic under least fixed-point of operations definable by positive formulas. The following proposition makes use of previous work on the well-founded semantics of logic programs and database query languages.

Proposition 1. *For every ID-logic formula $[\Delta, P](\bar{t})$ there is an equivalent* FO+LFP *formula.*

Proof. For simplicity, suppose that P is the only defined symbol of Δ. Then Δ has the form $\{P(\bar{x}) \leftarrow \Psi_1, \ldots, P(\bar{x}) \leftarrow \Psi_s\}$. In formula (2) below, $\exists \bar{y}$ are the variables in Ψ_i distinct from $\bar{x}$ and Ψ_i' is obtained from Ψ_i by replacing positive occurrences of P by P' so that $\phi(\bar{x}, P)$ is negative in P.

$$\phi(\bar{x}, P) := [\mathrm{LFP}_{\bar{x}, P'} \bigvee_{i=1}^{s} \exists \bar{y} \Psi_i'](\bar{x}). \tag{2}$$

This implies that Γ^ϕ is *antitone*, i.e., $X \subseteq Y$ implies $\Gamma^\phi(Y) \subseteq \Gamma^\phi(X)$. Therefore, $\Gamma_0^\phi \subseteq \Gamma_2^\phi \subseteq \Gamma_4^\phi \subseteq \ldots \Gamma_5^\phi \subseteq \Gamma_3^\phi \subseteq \Gamma_1^\phi$. In the literature on well-founded semantics of logic programming (e.g. [1, 20]), the increasing even subsequence $\Gamma_{2 \cdot n}^\phi$ corresponds to the underestimates of the positive facts and the decreasing odd subsequence $\Gamma_{2 \cdot n+1}^\phi$ corresponds to the underestimates of the negative facts.

The operator $\Theta := \Gamma^\phi \cdot \Gamma^\phi$ is monotone. Let $\psi(\bar{x}, P)$ be the formula $\phi(\bar{x}, \phi(\_, P))$ which is positive in P. Then $\Theta = \Gamma^\psi$. Since $\Gamma_n^\psi = \Gamma_{2 \cdot n}^\phi$, then $\Gamma_\infty^\psi = \bigcup_{n \geq 0} \Gamma_{2 \cdot n}^\phi = P_{\mathrm{true}}^\Delta$. Hence, $[\Delta, P](\bar{t})$ is equivalent to the FO+LFP formula $[\mathrm{LFP}_{\bar{x}, P} \psi(\bar{x}, P)](\bar{t})$. □

To extend the parallel with logic programming, one can define the false set $P_{\mathrm{false}}^\Delta$ of Δ as $P_{\mathrm{false}}^\Delta := \{a \in A \mid \exists n_0 \forall n \geq n_0 \,:\, a \notin P_n\}$, and the undefined set as $P_{\mathrm{undef}}^\Delta := A \setminus (P_{\mathrm{true}}^\Delta \cup P_{\mathrm{false}}^\Delta)$. Note that Δ is a total definition if and only if the set $P_{\mathrm{undef}}^\Delta$ is empty for all structures.

Proposition 1 shows that definitions add at most the power of FO+LFP to the expressive power of the original language (before definitions). It can be shown that on finite structures [7], definitions add *exactly* the power of FO+LFP to the original language. Proposition 1, however, is not restricted to finite structures.

3 SITUATION CALCULUS WITH DEFINITIONS

We consider a many-sorted version τ^{sc} of the vocabulary of the situation calculus with equality and with sorts for actions, situations and, possibly with one or more sorts for objects. The primitive non-logical symbols of sort *actions* consist of variables $a, a_1, a_2, \ldots$, and constants $A_0, A_1, A_2, \ldots$ The primitive non-logical symbols of sort *situations* consist of variables $s, s', s'', \tilde{s}, \ldots$, constant S_0, binary function $do(a, s)$, where a is an action, and s is a situation. This function defines a successor situation in terms of a current situation and a performed action. Finitely many predicate symbols $F_1, \ldots, F_n$ called *fluents* represent properties of the world, and have situations and possibly objects as their arguments. These symbols shall be viewed as definable in the rest of this paper. The logical symbols of the language are $\neg, \vee, \exists, =$. Other logical connectives and the universal quantifier $\forall$ are the usual abbreviations. In this paper, we do not consider predicates $Poss$ and $\sqsubseteq$ (cf. [16]).

A basic action theory is a set of axioms $\mathcal{D} = \mathcal{D}_f \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, where $\mathcal{D}_f$ is the set of foundational axioms for situations; $\mathcal{D}_{ss}$ is the set of successor state axioms, one for each fluent; $\mathcal{D}_{\mathrm{una}}$ is the set of unique name axioms for actions; and $\mathcal{D}_{S_0}$ is the description of the initial situation. The foundational axioms for the situation calculus, $\mathcal{D}_f$, are

$$\begin{gathered} S_0 \neq do(a, s), \\ do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2, \\ \forall P\, [P(S_0) \wedge \forall s'\, \forall a\; P(s') \supset P(do(a, s')) \supset \forall s\; P(s)]. \end{gathered} \tag{3}$$

These axioms guarantee that situations compose an infinitely branching tree. Indeed, it can be shown that the class of tree-like structures is completely characterized by the induction principle on situations and unique name assumptions for situations [18]. Successor state axioms, $\mathcal{D}_{ss}$, have the form

$$F_i(\bar{x}, do(a, s)) \equiv [\gamma_{F_i}^+(\bar{x}, a, s) \vee F_i(\bar{x}, s) \wedge \neg \gamma_{F_i}^-(\bar{x}, a, s)]. \tag{4}$$

Formula $\gamma_F^+(\bar{x}, a, s)$ (respectively, $\gamma_F^-(\bar{x}, a, s)$) denotes a first order formula specifying the conditions under which fluent F is true (respectively, false) in the successor situation [17]. The only free variables of these formulas are those among $\bar{x}, a, s$. Function symbol do does not occur in these formulas, neither does S_0. The unique name axioms, $\mathcal{D}_{una}$, specify that any two actions with different names are not equal. The description of the initial situation, $\mathcal{D}_{S_0}$, is a set of first order sentences that are uniform in S_0; i.e., they contain no situation term other than S_0. We shall augment the foundational axioms $\mathcal{D}_f$, unique name assumptions $\mathcal{D}_{una}$, and the initial database $\mathcal{D}_{S_0}$ with two definitions, Δ^c and Δ^h, evaluated consecutively.

Example 2 (suitcase). Several versions of this example (e.g. [19]) have been used to demonstrate that domain constraints are not strong enough and that an explicit notion of causality is necessary. Suppose we have a suitcase which opens if and only if both of its two locks are open. Fluent O represents the fact that the suitcase is open; fluents L_1, L_2 are true whenever the corresponding locks are open. Constant symbols o_1, o_2, c_1, c_2 represent actions of opening and closing the respective lock. Following [11], [6] and other authors, we distinguish between the fact that fluent F (respectively, its negation) holds in a situation and the statement that F (respectively, $\neg F$) is caused to hold. To capture this difference, we introduce predicates $C_F(s)$, $C_{\neg F}(s)$ for each fluent F, where C stands for "caused". For the rest of the paper, we adopt the following notation: Whenever convenient, we omit the situation argument s and denote the situation argument $do(a, s)$ by a prime; for example, L_2 represents $L_2(s)$, and C_{L_1}' denotes $C_{L_1}(do(a, s))$. The rules of Δ_1^c specify direct and indirect effects of actions.

$$\begin{array}{ll} C_{L_1}' \leftarrow a = o_1 & C_{\neg L_1}' \leftarrow a = c_1 \\ C_{L_2}' \leftarrow a = o_2 & C_{\neg L_2}' \leftarrow a = c_2 \\ C_O' \leftarrow C_{L_1 \wedge L_2}' & C_{\neg O}' \leftarrow C_{\neg L_1 \vee \neg L_2}' \end{array}$$

Following [6], we view the last two rules as an abbreviation for

$$\begin{array}{ll} C_O' \leftarrow C_{L_1}' \wedge C_{L_2}' & C_{\neg O}' \leftarrow C_{\neg L_1}' \\ C_O' \leftarrow C_{L_1}' \wedge L_2 \wedge \neg C_{\neg L_2}' & C_{\neg O}' \leftarrow C_{\neg L_2}' \\ C_O' \leftarrow L_1 \wedge \neg C_{\neg L_1}' \wedge C_{L_2}' & \end{array}$$

Let Δ_1^h be given by the following set of rules ($i = 1, \ldots, n$):

$$F_i{}' \leftarrow C_{F_i}' \qquad F_i{}' \leftarrow F_i \wedge \neg C_{\neg F_i}'$$

In our example, $F_i \in \{L_1, L_2, O\}$ and $n = 3$.

4 GENERATING SUCCESSOR STATE AXIOMS

Consider a vocabulary τ. Let $\mathrm{Form}[\tau]$ be the set of well-formed formulas over τ. Let Δ be a propositional definition in vocabulary τ. Define operator $\mathcal{R} : \mathrm{Form}[\tau] \to \mathrm{Form}[\tau]$ inductively as follows.

$$\mathcal{R}[\psi] = \begin{cases} P(\bar{x}_P) & \text{if } \psi = P(\bar{x}_P) \text{ and } P \in (\tau, \Delta)_\circ, \\ \mathcal{R}[\phi_P(\bar{x}_P)] & \text{if } \psi = P(\bar{x}_P),\, P \in (\tau, \Delta)_\mathrm{d} \\ & \text{and } \phi_P(\bar{x}_P) \text{ is obtained as in (1)}, \\ \mathcal{R}[\phi_1] \vee \mathcal{R}[\phi_2] & \text{if } \psi = \phi_1 \vee \phi_2, \\ \neg\mathcal{R}[\phi] & \text{if } \psi = \neg\,\phi. \end{cases}$$

Transformation $\mathcal{R}$ is called the *regression* operator. If the theory does not include static causal rules, the definition of the regression operator above is essentially the same as the original definition of [16], except that in our case it is a purely syntactic transformation. Regression for the non-propositional case is a generalization of the propositional definition using the grounding technique defined in [5]. In what follows, we construct successor state axioms of the form $F' \equiv \mathcal{R}[C'_F] \vee F \wedge \neg\mathcal{R}[C'_{\neg F}]$. If regression is performed successfully, predicate C_F does not appear in this sentence. We discuss the general solution at the end of the section.

Example 3 (suitcase, continued). Consider the regression of C'_O and $C'_{\neg O}$ from Example 2. It is convenient to represent regression in a tree form. Thus, in Figure 1, the regression of each atomic formula P is represented by a tree where (1) each node of the tree is labeled by an atomic or a negated atomic formula; (2) the root is labeled by P; (3) each leaf is labeled by an open atomic or negated atomic formula; and (4) for each internal node, there is a rule in Δ so that the head is the label of that node, and the body is the list of labels of its children. Regression trees are similar to "proof trees", an alternative way of representing inductive definitions [5, 2]. In particular, along a branch of a "proof tree", labels of the nodes do not compose cycles. This property also holds for regression trees.

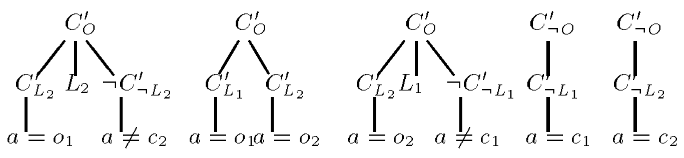


Figure 1. Regression trees for the suitcase example.

The result of regression of C'_O is the disjunction of conjunctions of leaves in each tree with C'_O in the root. Using unique name assumptions, we obtain the following successor state axioms.

$$\begin{gathered} \forall s \forall a\ L_1(do(a,s)) \equiv a = o_1 \vee L_1(s) \wedge a \neq c_1 \\ \forall s \forall a\ L_2(do(a,s)) \equiv a = o_2 \vee L_2(s) \wedge a \neq c_2 \\ \forall s \forall a\ O(do(a,s)) \equiv L_1(s) \wedge a = o_2 \vee L_2(s) \wedge a = o_1 \\ \vee O(s) \wedge a \neq c_1 \wedge a \neq c_2 \end{gathered}$$

Example 4 (gear wheels). This example illustrates the case where causal dependencies form a positive cycle. Suppose we have two connected gear wheels and suppose fluents T_1, T_2 represent the fact that the first (respectively, the second) wheel is turning. If a wheel starts or stops turning, we denote this change by action $start_1$, $start_2$, $stop_1$, or $stop_2$, respectively. The following rules of Δ_2^c specify direct and indirect effects of actions:

$$\begin{array}{ll} C'_{T_1} \leftarrow a = start_1 & C'_{\neg T_1} \leftarrow a = stop_1 \\ C'_{T_2} \leftarrow a = start_2 & C'_{\neg T_2} \leftarrow a = stop_2 \\ C'_{T_1} \leftarrow C'_{T_2} & C'_{T_2} \leftarrow C'_{T_1} \\ C'_{\neg T_1} \leftarrow C'_{\neg T_2} & C'_{\neg T_2} \leftarrow C'_{\neg T_1} \end{array}$$

The definition Δ_2^h is similar to Δ_1^h from the previous example. Figure 2 represents the regression of C'_{T_1} and $C'_{\neg T_1}$. Notice that the third and the sixth tree from the left are discarded as having cyclic branches.

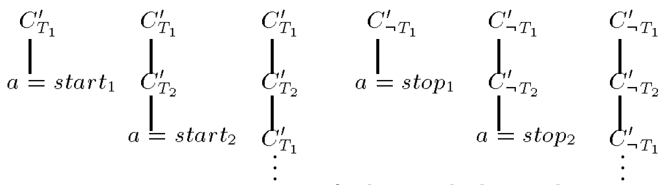
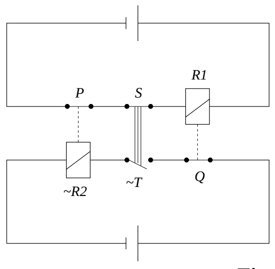


Figure 2. Regression trees for the gear wheels example.

We obtain the following successor state axioms:

$$\begin{gathered} \forall s \forall a\ T_1(do(a,s)) \equiv a = start_1 \vee a = start_2 \\ \vee T_1(s) \wedge a \neq stop_1 \wedge a \neq stop_2 \\ \forall s \forall a\ T_2(do(a,s)) \equiv a = start_1 \vee a = start_2 \\ \vee T_2(s) \wedge a \neq stop_1 \wedge a \neq stop_2 \end{gathered}$$

Example 5 (circuit). The following example from [6] illustrates the case where causal dependencies contain a cycle through negation. Consider the circuit from Figure 3.



$$\begin{array}{ll} C'_T \leftarrow a = c_t & C'_{\neg T} \leftarrow a = o_t \\ C'_{R_1} \leftarrow C'_{P \wedge S} & C'_{R_2} \leftarrow C'_{T \wedge Q} \\ C'_Q \leftarrow C'_{R_1} & C'_{\neg P} \leftarrow C'_{R_2} \end{array}$$

$$\begin{array}{ll} C'_{\neg R_1} \leftarrow C'_{\neg P \vee \neg S} & C'_{\neg R_2} \leftarrow C'_{\neg T \vee \neg Q} \\ C'_{\neg Q} \leftarrow C'_{\neg R_1} & C'_P \leftarrow C'_{\neg R_2} \end{array}$$

$$\begin{array}{ll} C'_{\neg S} \leftarrow C'_T & C'_T \leftarrow C'_{\neg S} \\ C'_S \leftarrow C'_{\neg T} & C'_{\neg T} \leftarrow C'_S \end{array}$$

Figure 3. Electrical circuit.

The circuit consists of two interconnected sub-circuits. On one circuit, there are two serially connected switches, P and S, and a relay, R_1. The other circuit contains two switches, T and Q, and a relay, R_2. The only actions used in the theory are c_t and o_t, which represent closing and opening of switch T. The first relay is on (i.e., true) if and only if switches P and S are on, $R_1(s) \equiv P(s) \wedge S(s)$. Similarly, for the second relay, we have $R_2(s) \equiv T(s) \wedge Q(s)$. Relay R_1 and switch Q, as well as relay R_2 and switch P are connected in the following way: $R_1(s) \equiv Q(s)$, $R_2(s) \equiv \neg P(s)$. The corresponding causal rules, with the same kind of abbreviations as in example 2, are represented in Figure 3.

As usual, we include Δ_3^h, where the defined symbols are all the fluents of the theory. After expanding the abbreviations in Δ_3^c as in Example 2, we notice that the definition contains a negative cycle, $C'_{\neg P} \leftarrow C'_{R_2} \leftarrow C'_Q \leftarrow C'_{R_1} \leftarrow \neg C'_{\neg P}$, so Δ_3^c is not stratified. Reproducing regression trees for this example requires much space, so we can only give the resulting successor state axioms.

$$\begin{gathered} \forall s \forall a\ T(do(a,s)) \equiv a = c_t \vee T(s) \wedge a \neq o_t \\ \forall s \forall a\ Q(do(a,s)) \equiv a = o_t \vee Q(s) \wedge a \neq c_t \\ \forall s \forall a\ S(do(a,s)) \equiv a = o_t \vee S(s) \wedge a \neq c_t \\ \forall s \forall a\ P(do(a,s)) \qquad \forall s \forall a\ \neg R_2(do(a,s)) \\ \forall s \forall a\ R_1(do(a,s)) \equiv a = o_t \vee R_1(s) \wedge a \neq c_t \end{gathered}$$

Example 6 (TC of a graph). This example motivates the need for the boundedness condition for generating successor state axioms. Suppose we have a directed (potentially infinite) graph in which each arc is represented by fluent $A(x,y)$. We want to compute the transitive closure $TC(x,y)$ of the relation A. The action of adding arc (x,y) to the graph is represented by action $add(x,y)$. For simplicity, we assume that arcs cannot be deleted. The rules of definition Δ^{TC} are:

$$\begin{array}{l} C'_{A(x,y)} \leftarrow a = add(x,y) \\ C'_{TC(x,y)} \leftarrow C'_{A(x,y)} \\ C'_{TC(x,y)} \leftarrow C'_{(TC(x,z) \wedge A(z,y)))} \end{array}$$

The last rule is the usual abbreviation. Consider the following regression. $\mathcal{R}[C'_{TC(x,y)}] \equiv a = add(x,y) \vee \exists z(\mathcal{R}[C'_{TC(x,z)}] \wedge a = add(z,y)) \vee \exists z(\mathcal{R}[C'_{TC(x,z)}] \wedge A(z,y)) \equiv \ldots$ The length of the resulting formula grows with every iteration of regression, and the process never terminates. Thus, in general, regression may produce a formula of infinitary predicate logic. This is not acceptable for our purposes and motivates the following study.

We say that a definition Δ is *bounded* if there is a constant c such that, for all τ_o-structures, the fixed-point P_∞ of the sequence $(P_n)_{n\geq 0}$ defined in Section 2 is reached within c steps. Observe that definition Δ^{TC} from Example 6 is unbounded — for a structure τ_o with an infinite domain, the closure ordinal of the corresponding operator is ω. If definition Δ is bounded, then there is a recursion-free definition Δ' such that $\forall \bar{x}[\Delta, P](\bar{x}) \equiv [\Delta', P](\bar{x})$.

Lemma 1. *If Δ^c is bounded then, for each fluent F_i, the regression of C'_{F_i} (respectively, $C'_{\neg F_i}$) in Δ^c is equivalent to a first order formula, call it $\gamma^+_{F_i}$ (respectively, $\gamma^-_{F_i}$).*

In the theorem below, we need the following consistency condition $\bigwedge_{i=1,\ldots,n} \neg\exists\bar{x}\exists a\exists s\ \gamma^+_{F_i}(\bar{x},a,s) \wedge \gamma^-_{F_i}(\bar{x},a,s)$. This sentence requires that the conditions making fluent F_i true ($\gamma^+_{F_i}(\bar{x},a,s)$) and the conditions making it false ($\gamma^-_{F_i}(\bar{x},a,s)$ are never true simultaneously.

Theorem 1. *Suppose the consistency condition holds and Δ^c is bounded. Let σ be a term of sort situation in the vocabulary τ^{sc}, and let F be a fluent. Then $\mathcal{D}_f \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models F(\sigma)$ if and only if, for every structure $\mathcal{A}$ satisfying $\mathcal{D}_f \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, $\sigma^{\mathcal{A}} \in [\Delta^c \cdot \Delta^h]^{\mathcal{A}}$ where $[\Delta^c \cdot \Delta^h]^{\mathcal{A}}$ is the relation defined by $\Delta^c \cdot \Delta^h$ with respect to structure $\mathcal{A}$, and $\sigma^{\mathcal{A}}$ is the interpretation of the situation term σ in this structure.*

The theorem follows from the procedure for generating successor state axioms, from the definition of regression and from the preceding lemma.

5 CONCLUSIONS

In this paper, we have addressed the logic for representing inductive definitions (ID-logic) as proposed by Denecker. We have provided a semantics for this logic, by translating its sentences to the sentences of fixed-point logic. ID-logic may be viewed as a fragment of second-order logic; moreover, we have demonstrated that definitions add at most the power of FO+LFP to the original language (before definitions). This conclusion is very important from a computational point of view, because of the well-known complexity/expressiveness trade-off. Despite its modest power, ID-logic is expressive enough to encode a rather general solution to the ramification problem.

Representing causal theories in the situation calculus, and in first-order logic in general, requires some care. We have demonstrated that boundedness of causal theories is a sufficient condition for translating to successor state axioms of the situation calculus. This condition is not necessary, however: as shown by Gurevich and Ajtai [3], there are first-order representable but unbounded logic programs. In the simple case of negation-free causal theories, boundedness is a necessary and sufficient condition for a first-order translation.

Unfortunately, boundedness of Δ^c is not decidable, either for positive or for general definitions. This conclusion follows from the observation that boundedness of logic programs is an undecidable problem [8]. Thus, in general, it is not possible to give a computationally useful characterization of those causal theories that are representable in the situation calculus. In spite of these negative conclusions, it is possible to identify wide subclasses of definitions for which there are algorithms to detect when a definition is first-order. For instance, using the results about general logic programs, one can demonstrate that boundedness is decidable for stratified causal theories, where the defined predicates contain at most one argument, in addition to a situation term.

ACKNOWLEDGEMENTS

Thanks to Marc Denecker and to the anonymous referees for helpful comments.

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[2] P. Aczel, 'An introduction to inductive definitions', in *Handbook of Mathematical Logic*, ed., J. Barwise, pp. 739–782. Elsevier, (1977).

[3] M. Ajtai and Y. Gurevich, 'Datalog versus first order logic', in *Proc. 30th IEEE FOCS*, pp. 142–148, (1989).

[4] M. Denecker, 'The well-founded semantics is the principle of inductive definitions', in *Logics in Artificial Intelligence*, eds., J. Dix, L. Fari nas del Cerro, and U. Furbach, volume 1489 of *Lecture Notes in Artificial Intelligence*, pp. 1–16. Springer-Verlag, (1998).

[5] M. Denecker, 'Extending classical logic with inductive definitions', in *Proc. CL'2000*, (2000).

[6] M. Denecker, D. Theseider Duprè, and K. Van Belleghem, 'An inductive definition approach to ramifications', *Linköping Electronic Articles in Computer and Information Science*, **3(1998): nr 7**, (1998). URL: http://www.ep.liu.se/ea/cis/1998/007/.

[7] H.-D. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer-Verlag, Berlin, Heidelberg, New York, second edn., 1999.

[8] H. Gaifman, H. Mairson, Y. Sagiv, and M.Y. Vardi, 'Undecidable optimization problems for database logic programs', in *Proc. 2nd IEEE LICS*, pp. 106–115, (1987).

[9] V. Lifschitz, 'On the logic of causal explanation', *J. of Artificial Intelligence*, **96**, 451–465, (1997).

[10] V. Lifschitz, 'Action languages, answer sets and planning', in *The Logic Programming Paradigm: a 25-Year Perspective*, 357–373, Springer Verlag, (1999).

[11] F. Lin, 'Embracing causality in specifying the indirect effects of actions', in *Proc. of IJCAI 95*, pp. 1985–1991, (1995).

[12] N. McCain and H. Turner, 'A causal theory of ramifications and qualifications', in *Proc. of IJCAI 95*, pp. 1978–1984, (1995).

[13] J. McCarthy, 'Circumscription — a form of non-monotonic reasoning', *J. of Artificial Intelligence*, **13**, 27–39, (1980).

[14] S. McIlraith, 'A closed-form solution to the ramification problem (sometimes)', in *Proceedings of the Workshop on Nonmonotonic Reasoning, Action and Change, Fifteenth International Joint Conference on Artificial Intelligence*, pp. 103–126, (997).

[15] Y.N. Moschovakis, *Elementary Induction on Abstract Structures*, Amsterdam, North Holland, 1974.

[16] F. Pirri and R. Reiter, 'Some contributions to the metatheory of the situation calculus', *J. of ACM*, **46(3)**, 261–325, (1999).

[17] R. Reiter, 'The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression', in *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, ed., Vladimir Lifschitz, 359–380, Academic Press, San Diego, CA, (1991).

[18] E. Ternovskaia, 'Inductive definability and the situation calculus', in *Transaction and Change in Logic Databases*, volume 1472 of *Lecture Notes in Computer Science*. Springer-Verlag, (1998).

[19] M. Thielscher, 'Ramification and causality', *J. of Artificial Intelligence*, **89**, 317–364, (1997).

[20] A. Van Gelder, 'An alternating fixpoint of logic programs with negation', *Journal of computer and system sciences*, **47**, 185–221, (1993).