Artificial Intelligence 175 (2011) 25–48

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Non-Markovian control in the Situation Calculus ☆

Alfredo Gabaldon

Center for Artificial Intelligence, New University of Lisbon, Lisbon, Portugal

ARTICLE INFO

Article history:
Available online 3 April 2010

Keywords:
Reasoning about actions
Situation Calculus

ABSTRACT

In reasoning about actions, it is commonly assumed that the dynamics of domains satisfies the *Markov Property*: the executability conditions and the effects of all actions are fully determined by the present state of the system. This is true in particular in Reiter's Basic Action Theories in the Situation Calculus. In this paper, we generalize Basic Action Theories by removing the Markov property restriction, making it possible to directly axiomatize actions whose effects and executability conditions may depend on past and even alternative, hypothetical situations. We then generalize Reiter's regression operator, which is the main computational mechanism used for reasoning with Basic Action Theories, so that it can be used with non-Markovian theories.

Since the 1960's when John McCarthy's papers (in particular the 1969 paper with Pat Hayes) appeared introducing the Situation Calculus, researchers have been studying and working on this language for reasoning about dynamic domains. The Situation Calculus, one of John's many great inventions, is the topic of this paper and I am delighted to have this opportunity to make a contribution to a special issue in John's honor.

1. Introduction

An assumption commonly made in formalisms for reasoning about the effects of actions is the so called *Markov property*: the executability of an action and its effects are entirely determined by the current state or situation. In particular, Reiter's Basic Action Theories [2], a Situation Calculus [3,4] based axiomatization, define the value of a fluent after the execution of an action in terms of a formula that can only talk about the situation in which the action would be executed. The preconditions of an action are specified by formulas with the same restriction. In this paper we generalize Basic Action Theories by removing this restriction. The generalized theories will allow the executability conditions and the effects of an action to depend not only on what holds when the action is to occur, but also on whether certain conditions were satisfied at different points in the past and even alternative hypothetical evolutions of the system.

As an example, imagine a robot that works in a biological research facility with different safety-level areas. The dynamics is such that a material will be considered contaminated after the robot touches it if the robot has been to a low safety area or has directly been in contact with a hazardous material, and has not been to the disinfection station since then. So the effect of touching the material depends on the history of robot activities. We could also imagine that the robot cannot execute the action $open(Entrance, Lab1)$ if $temp(Lab1) > 30$ was ever true since the last time $closed(Entrance, Lab1)$ occurred. The latter is an example of an action with non-Markovian preconditions.

In simple scenarios, it is not difficult to extend a theory to preserve the necessary history by means of new state variables, especially when the domain is finite. But in complex domains it may not be obvious how to do it, and the

☆ A preliminary abstract of this paper appeared in Proc. of AAAI'02 (A. Gabaldon (2002) [1]).

E-mail address: ag@di.fct.unl.pt.

0004-3702/$ – see front matter
doi:10.1016/j.artint.2010.04.012

resulting theory may be substantially more complex, with a larger number of state variables and corresponding axioms describing their dynamics.

Our goal in this paper is to generalize Reiter's Basic Action Theories [5,2] by removing the Markov property requirement and generalize the main reasoning mechanism used with these theories, namely the regression operator $\mathcal{R}$, so that it can be used with non-Markovian theories, and applied to Situation Calculus formulas that refer to the past, to alternative evolutions, and to definite future situations.

This generalized regression operator is not only useful in cases where the action theory is non-Markovian. Even if the background theory is Markovian, the operator is useful for answering queries that refer to past situations through quantification and the subsequence relation $\sqsubset$. This is not possible with Reiter's original regression operator. In [2, Section 4.8], Reiter presents a few specialized procedures for evaluating certain *historical queries* with respect to a database log (a sequence of ground action terms) and a Markovian action theory. Those queries are a very small subset of the class of queries that the generalized regression operator we present here can handle.

Our work is relevant to a variety of research problems that involve the formalization of dynamic properties:

(1) Some work in database theory has been concerned with the semantics of dynamic integrity constraints [6,7]. These constraints are typically expressed in Past Linear Temporal Logic, a logic with temporal connectives *Previous*, *Sometime* in the past, *Always* in the past, and *Since*. In a formalization of a database system in the Situation Calculus, such temporal connectives amount to references to past situations, and the constraints to restrictions on when a sequence of actions can be considered a "legal" database system evolution. These past temporal logic connectives have an encoding as formulas in the non-Markovian Situation Calculus and hence the latter can be used as a logical framework for the study, specification and modeling of databases with dynamic integrity constraints. The advantage of carrying out such work in this framework is that all the different aspects of the problem, i.e. database dynamics, transactions and constraints, can be captured within the same Situation Calculus framework.

(2) Also in the area of databases, more specifically in work on database transaction systems, the *rollback* operation, which reverts a database back to its original state after a long transaction fails or is canceled, clearly has a non-Markovian flavor: its effects depend not on what is true in the state it is executed, but on the state right before the transaction being reversed started. Indeed, Kiringa [8] and Kiringa and Gabaldon [9,10] present logical specifications of database transactions in the non-Markovian Situation Calculus.

(3) In planning, domain dependent knowledge for search control has been used with great success [11,12]. Bacchus and Kabanza's forward-chaining planning system, TLPlan, uses search control knowledge in the form of temporal logic formulas. The same approach has been applied in the Situation Calculus with some simple planners written in Golog [2]. The latter planners perform a forward search, eliminating partial plans if they lead to "bad situations." Search control knowledge is encoded through a predicate $badSituation(s)$ whose definition is restricted to properties of the current situation s. The generalization of the action theories and the regression operator we shall develop here allows the definition of this predicate to refer to any situation that precedes s and bounded future situations. As we mention above, past temporal logic expressions can be encoded as Situation Calculus formulas suitable for regression with our generalized operator and be used in the definition of $badSituation(s)$. In other words, the generalized regression operator allows one to use temporal search control knowledge of a similar form and expressive power as used in TLPlan directly in Golog planners with the $badSituation(s)$ predicate. Search control knowledge in this context is further explored in our recent work [13,14].

(4) Another area where non-Markovian features arise naturally is in specifying reward functions in decision theoretic planning. There, agents are often rewarded based on their long-term behavior rather than just on the current state of affairs. Bacchus, Boutilier and Grove [15,16] have developed techniques for solving such non-Markovian Decision Processes. More recent work on non-Markovian rewards appears in [17,18].

(5) Finally, some time ago John McCarthy [19] described a programming language called "Elephant 2000" which, among other features, "does not forget." This is a language that would allow one to write programs that explicitly and directly refer to past states of the programming environment. The generalized regression operator we present here could form the foundation for a non-forgetting Golog [20]. Such a dialect of Golog would allow test conditions that refer to the past, for instance, as in the statement **if** $(P$ **since** $Q)$ **then** δ.

This paper is organized as follows: we start in Section 2 with an overview of the Situation Calculus and Reiter's Basic Action Theories. In Section 3 we introduce a class of Situation Calculus formulas that can refer to past and finite future situations and can be regressed, and based on this, our generalization of action theories for non-Markovian control. In Section 4 we present a regression operator that works for those formulas and theories and prove its correctness, followed by a Prolog implementation in Section 5 and our concluding remarks in Section 6.

2. Overview of the Situation Calculus and Basic Action Theories

The Situation Calculus [3,4] is a dialect of classical logic for representing and reasoning about dynamically changing worlds. A theory in this language consists of a collection of axioms describing how the world changes when actions occur. Accordingly, the ontology of the Situation Calculus includes three main ingredients: actions, situations, and fluents. Situ-

ations refer to possible evolutions or "histories" of the world and are identified with the sequences of actions that have occurred in such a history. Fluents are the properties of the world that change over time.

Since McCarthy introduced it, the Situation Calculus has been adopted in various languages and differing axiomatizations. In this paper we use the version described in [5,21,2], whose formal definition we review next.

2.1. Situation Calculus

The Situation Calculus (as we will refer to this version from now on), denoted by $\mathcal{L}_{sitcalc}$, is a second-order language with equality and with three disjoint sorts called *action*, *situation* and *object*. In addition to standard logical symbols, its vocabulary includes:

- A countably infinite number of variable symbols of each sort and predicate variables of all arities.
- The constant symbol S_0, of sort *situation*, which is used to represent the *initial situation* of the world, before any action occurs.
- The function symbol *do* of sort $action \times situation \mapsto situation$, which is used to form sequences of actions. A term $do(a, s)$ represents the situation after action a is executed in situation s.
- The binary predicate symbol $\sqsubset$, used to represent an ordering relation on situations. Intuitively, $s \sqsubset s'$ means that situation s precedes s', or, in terms of sequences of actions, that s is a proper prefix of the sequence of actions s'.
- The binary predicate symbol $Poss : action \times situation$. The intuitive meaning of $Poss(a, s)$ is that in situation s, it is possible to execute action a.
- For each $n \geqslant 0$, a finite or countably infinite number of function symbols of sort $(action \cup object)^n \mapsto action$ called *action functions*. For example: $move(A, B)$ and $write(agent3, nextChapter(Thesis, s))$. We will sometimes refer to an action function as an *action type*, since each action function can have many instances.
- For each $n \geqslant 0$, a finite or countably infinite number of predicate symbols of sort $(action \cup object)^n \times situation$ called *relational fluents*, and a countably infinite number of function symbols of sort $(action \cup object)^n \times situation \mapsto action \cup object$ called *functional fluents*. For example,

$$on\big(A, B, do\big(move(A, B), S_0\big)\big).$$

- For each $n \geqslant 0$, a finite or countably infinite number of n-ary predicates and functions without a situation argument which are used for expressing situation independent relations, i.e. properties of the world that do not change as a result of actions. For example: $distance(csDept, mathDept)$.

The following four *Foundational Axioms*, denoted by Σ, characterize situations and the precedence relation $\sqsubset$:

$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2, \tag{1}$$

$$(\forall P).P(S_0) \wedge (\forall a, s)\big[P(s) \supset P\big(do(a, s)\big)\big] \supset (\forall s)P(s), \tag{2}$$

$$\neg s \sqsubset S_0, \tag{3}$$

$$s \sqsubset do\big(a, s'\big) \equiv s \sqsubset s' \vee s = s'. \tag{4}$$

Axiom (1) is a unique names axiom for the function *do*, and it captures the intuition that different histories (sequences of actions) are different situations. Axiom (2) defines how situations are built using the function *do*. This definition is inductive and so this axiom is second-order. Axiom (3) stipulates that S_0 has no preceding situations, thus capturing the intuition that it is the initial situation, and together with axiom (4) and the induction axiom it defines the precedence relation $\sqsubset$.

2.2. Basic Action Theories

The foundational axioms Σ are the domain independent part of a Situation Calculus axiomatization of a dynamic world. The domain dependent part consists of: a set of axioms describing the initial state of the world, a set of axioms describing the conditions under which actions are executable and another set of axioms describing how each of the fluents changes when actions occur. The notion of a uniform formula is useful in describing these axioms.

Definition 1 *(Uniform formulas).* Let σ be a term of sort *situation*. The $\mathcal{L}_{sitcalc}$ terms *uniform in* σ are inductively defined as the smallest set of terms such that:

(1) Any term that does not mention a term of sort *situation* is uniform in σ.
(2) If g is an n-ary non-fluent function symbol, and $t_1, \ldots, t_n$ are terms that are uniform in σ and whose sorts are appropriate for g, then $g(t_1, \ldots, t_n)$ is uniform in σ.
(3) If f is an $(n+1)$-ary functional fluent symbol, and $t_1, \ldots, t_n$ are terms that are uniform in σ and whose sorts are appropriate for f, then $f(t_1, \ldots, t_n, \sigma)$ is uniform in σ.

The formulas of $\mathcal{L}_{sitcalc}$ that are *uniform in* σ are inductively defined as the smallest set of formulas such that:

(1) Any formula that does not mention a term of sort *situation* is uniform in σ.
(2) When F is an $(n+1)$-ary relational fluent and $t_1, \ldots, t_n$ are terms uniform in σ whose sorts are appropriate for F, then $F(t_1, \ldots, t_n, \sigma)$ is a formula uniform in σ.
(3) If U_1 and U_2 are formulas uniform in σ, then so are $\neg U_1$, $U_1 \wedge U_2$ and $(\exists v)U_1$ provided v is a variable not of sort *situation*.

For each action type $A(\vec{x})$,[1] a dynamic world axiomatization includes an axiom defining when such an action is executable. These axioms define the relation $Poss(a, s)$:

Definition 2 *(Action precondition axiom).* An *action precondition axiom* is a sentence of the form:

$$Poss\big(A(x_1, \ldots, x_n), s\big) \equiv \Pi_A(x_1, \ldots, x_n, s),$$

where A is an n-ary action function symbol and $\Pi_A(x_1, \ldots, x_n, s)$ is a formula with free variables among $x_1, \ldots, x_n, s$ that is uniform in s.

For each fluent, relational or functional, an axiom defining the fluent's value after the execution of an action is included in the dynamic world axiomatization:

Definition 3 *(Successor state axioms).* A *successor state axiom* for an $(n+1)$-ary relational fluent F is a sentence of the form:

$$F\big(x_1, \ldots, x_n, do(a, s)\big) \equiv \Phi_F(x_1, \ldots, x_n, a, s),$$

where $\Phi_F(x_1, \ldots, x_n, a, s)$ is a formula with free variables among $x_1, \ldots, x_n, a, s$ that is uniform in s.

A *successor state axiom* for an $(n+1)$-ary functional fluent f is a sentence of the form:

$$f\big(x_1, \ldots, x_n, do(a, s)\big) = y \equiv \phi_f(x_1, \ldots, x_n, y, a, s),$$

where $\phi_f(x_1, \ldots, x_n, y, a, s)$ is a formula with free variables among $x_1, \ldots, x_n, y, a, s$ that is uniform in s.

The right-hand sides of action precondition and successor state axioms are required to be uniform in situation variable s. This is exactly how the Markov property is enforced in these theories. Intuitively, the executability conditions of an action and the value of fluents in a successor situation can only be defined in terms of the current situation.

A collection of axioms of the above forms is known as a Basic Action Theory:

Definition 4 *(Basic Action Theories).* An $\mathcal{L}_{sitcalc}$ *Basic Action Theory*, $\mathcal{D}$, is a collection of axioms $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ where:

- Σ are the foundational axioms (1)–(4).
- $\mathcal{D}_{ss}$ is a set of successor state axioms, one for each fluent.
- $\mathcal{D}_{ap}$ is a set of action precondition axioms, one for each action type.
- $\mathcal{D}_{una}$ is a set of unique name axioms for actions, such as: $pickup(x) \neq putdown(y)$ and $stack(x, y) = stack(x', y') \supset x = x' \wedge y = y'$.
- $\mathcal{D}_{S_0}$ is a set of first order sentences that are uniform in S_0. These axioms describe the initial state of the world and are sometimes referred to as the *initial database*.

3. Non-Markovian theories

In this section we introduce Non-Markovian Basic Action Theories. In Markovian action theories, the Markov assumption is realized by requiring that the formulas in the action precondition axioms and successor state axioms refer only to one situation, a variable s, which is prenex universally quantified in the axioms. In non-Markovian action theories, situation terms other than s will be allowed under the restriction that they refer to the past or to an alternative, explicitly bounded evolution. To make this formal, we need to introduce the notion of situation-bounded formulas. Intuitively, an $\mathcal{L}_{sitcalc}$ formula is bounded by a situation term σ if all the situation *variables* it mentions are restricted, through equality or the $\sqsubset$ predicate, to range over subsequences of σ. This notion is useful because in order to apply regression to a formula, one needs to know how many actions there are in each situation, i.e., how many regression steps to apply. A formula that mentions a situation variable can be regressed provided that the variable is restricted to be a subsequence of some situation term with a known number of actions in it. This intuitions are made precise below.

[1] $\vec{t}$ denotes a tuple of terms $t_1, \ldots, t_n$.

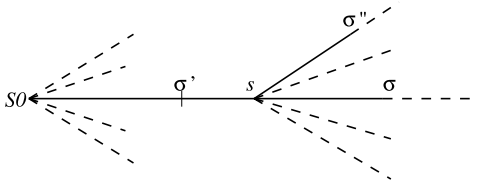

Fig. 1. Intuitively, that a formula is bounded by a situation term rooted at ρ, say by σ rooted at S_0 (shown as a solid line in the tree of situations), means that (a) any quantified situation variable s in the formula is restricted to range over situations between S_0 and σ (it must fall on the solid line, as depicted) and (b) any successor situation of s mentioned in the formula, say σ'', must explicitly enumerate the actions that occur after s, i.e., σ'' must be rooted at s. The formula may in turn have a subformula bounded by a situation term rooted at s, say σ'', which may quantify over variables as long as they fall on the solid line from S_0 to σ'', and so on. A formula bounded by σ can be regressed because it is possible to determine the length of all its situation terms during the regression. For example, if we choose a value for s, we will know its length because we know the length of σ, and we will then know the length of σ'' as well. On the other hand, the formula $(\exists s^*)\sigma' \sqsubset s^* \wedge F(s^*)$, for example, with a situation variable ranging over the situations on the dotted branches, cannot be regressed since it is not possible to put a bound on the length of s^*.

The following notation is used throughout: for $n \geqslant 0$, we write $do([\alpha_1, \ldots, \alpha_n], \rho)$ to denote the term $do(\alpha_n, do(\alpha_{n-1}, \ldots, do(\alpha_1, \rho)\ldots))$ of sort *situation*, where $\alpha_1, \ldots, \alpha_n$ are terms of sort *action* and ρ stands for a variable s of sort *situation* or the constant S_0. For such a situation term, we will refer to ρ as the *root* of the term. If $n = 0$, $do([\alpha_1, \ldots, \alpha_n], \rho)$ stands for the root ρ. Also, for each $i = 1, \ldots, n$, we say that the term $do([\alpha_1, \ldots, \alpha_i], \rho)$ is a *prefix* of $do([\alpha_1, \ldots, \alpha_n], \rho)$. If $i < n$ then we say it is a *proper prefix*. For any term $do([\alpha_1, \ldots, \alpha_n], \rho)$, $n \geqslant 0$, we say n is the *length* of the term.

Definition 5 *(Rooted situation terms).* For $n \geqslant 0$, let $\alpha_1, \ldots, \alpha_n$ be terms of sort *action*. A term $do([\alpha_1, \ldots, \alpha_n], s)$ is *rooted at* s iff s is the only (if any) variable of sort *situation* mentioned by $\alpha_1, \ldots, \alpha_n$. A term $do([\alpha_1, \ldots, \alpha_n], S_0)$ is *rooted at* S_0 iff $\alpha_1, \ldots, \alpha_n$ mention no variables of sort *situation*.

Recall that action functions may take terms of sorts *object* and *action* as arguments and that the language includes functional fluents, which may appear in these terms. For instance, $do(A(f(do(B, s'))), s)$ is a perfectly legal situation term. Notice that according to the above definition, this term is not rooted. The reason is that s is not the only variable the term mentions, there is also s' in the functional fluent term $f(B, s')$.

Next, we introduce the related notion of uniformly rooted formulas. These formulas have situation terms with a common root and thus, intuitively, their truth value is relativized to such a root.

Definition 6 *(Uniformly rooted formulas).* Let ρ be the constant S_0 or a variable of sort *situation*. The $\mathcal{L}_{sitcalc}$ formulas *uniformly rooted* at ρ are inductively defined as follows:

(1) If σ_1, σ_2 are terms of sort *situation* rooted at ρ, then $\sigma_1 \sqsubset \sigma_2$ is uniformly rooted at ρ;
(2) If t_1, t_2 are terms of any sort all of whose subterms of sort *situation* are rooted at ρ, then $t_1 = t_2$ is uniformly rooted at ρ;
(3) If W is a formula that does not mention terms of sort *situation*, then W is uniformly rooted at ρ;
(4) If $F(t_1, \ldots, t_n, \sigma)$ is a relational fluent atom, all terms of sort *situation* mentioned by $t_1, \ldots, t_n$ are rooted at ρ and σ is rooted at ρ, then $F(t_1, \ldots, t_n, \sigma)$ is a formula uniformly rooted at ρ;
(5) If W_1, W_2 are formulas uniformly rooted at ρ, then so are $\neg W_1$, $W_1 \wedge W_2$ and $(\exists v)W_1$ provided v is different than ρ.

3.1. Bounded and strictly bounded formulas

We define next the class of bounded and strictly bounded formulas of $\mathcal{L}_{sitcalc}$ upon which Non-Markovian Basic Action Theories will be based. The following abbreviations will be very useful for writing this kind of formulas:

$$\begin{aligned}
&\big(\exists s\colon \sigma' \sqsubset \sigma'' \sqsubset \sigma\big)W \stackrel{\text{def}}{=} (\exists s)\big[\sigma' \sqsubset \sigma'' \wedge \sigma'' \sqsubset \sigma \wedge W\big],\\
&\big(\forall s\colon \sigma' \sqsubset \sigma'' \sqsubset \sigma\big)W \stackrel{\text{def}}{=} (\forall s)\big[\big(\sigma' \sqsubset \sigma'' \wedge \sigma'' \sqsubset \sigma\big) \supset W\big].
\end{aligned} \tag{5}$$

Here, σ'' is rooted at s. Any $\sqsubset$ may be replaced by $\sqsubseteq$ or $=$. We will sometimes write $(\exists s\colon \sigma'' \sqsubset \sigma)W$ as a shorthand for $(\exists s\colon S_0 \sqsubseteq \sigma'' \sqsubset \sigma)W$ ($(\forall s)S_0 \sqsubseteq s$ is a logical consequence of the foundational axioms Σ).

The regression operator as defined in [5,21] requires all terms of sort situation to be of the form $do(\vec{\alpha}, S_0)$. Intuitively, this allows determining how many iterations are required in order to regress a formula into one relative to S_0 only. Our goal is to generalize regression for formulas that refer to situations through quantified variables. Since it is not possible to regress formulas with variables that refer to arbitrary situations, e.g. $(\exists s)F(s)$, we define a class of regressable formulas based on the notion of bounded formulas, which we introduce below. Quantified situation variables are restricted in these formulas in such a way that the number of regression steps remains bounded. Fig. 1 explains the intuition behind them.

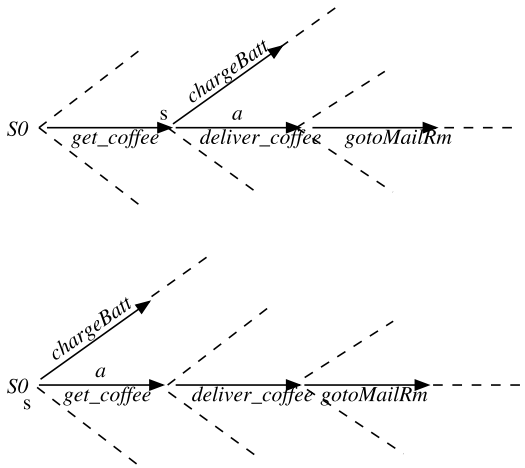

Fig. 2. Situation tree from Example 1. The existentially quantified variable s' must be equal either to $do(get\_coffee, S_0)$ (top tree) or to S_0 (bottom tree).

Definition 7 *(Bounded formulas).* Let ρ be a situation variable or the constant S_0. The formulas of $\mathcal{L}_{sitcalc}$ *bounded* by a situation term rooted at ρ are the smallest set of formulas such that:

(1) If W is an atom uniformly rooted at ρ, then W is bounded by a situation term rooted at ρ.
(2) If
  (a) W is a propositional combination of formulas each bounded by a situation term rooted at s or at ρ,
  (b) σ' does not mention variable s,
  (c) σ'' is rooted at s,
  (d) σ is rooted at ρ,
  then[2] $(\exists s{:}\ \sigma' \sqsubset \sigma'' \sqsubset \sigma)W$ is a formula bounded by a situation term rooted at ρ.
(3) If W_1, W_2 are formulas bounded by situation terms rooted at ρ, then $\neg W_1$, $W_1 \wedge W_2$ and $(\exists v)W_1$, where v is of sort *action* or *object*, are formulas bounded by a situation term rooted at ρ.

In the definition of bounded formulas above, no particular situation term σ is defined as the bound of a formula. We only say that a formula is bounded by "a situation term rooted at ρ." The reason for this is that the situation terms are not what is important, but the root ρ is. Indeed, in the last item of the definition, W_1, W_2 may have no situation term in common other than ρ. But if each is bounded by a term rooted at ρ, that is enough to guarantee that the combination be bounded by a term rooted at ρ. Throughout the paper, we will often talk about a formula being bounded by a specific term σ, usually one that occurs in the formula.

The following example attempts to illustrate these definitions.

Example 1. Suppose that σ denotes the following situation term:

$$do\big([get\_coffee, deliver\_coffee, gotoMailRm], S_0\big)$$

and consider the sentence:

$$(\exists a)\big(\exists s{:}\ S_0 \sqsubseteq do(a, s) \sqsubset \sigma\big)batteryCharged\big(do(chargeBatt, s)\big).$$

Intuitively, this sentence says that there is a situation s strictly preceding σ, after which some action a occurs, and in which action *chargeBatt* would have successfully charged the battery. This sentence is bounded by a term rooted at S_0, in particular by σ. Indeed, all situation variables mentioned by it are restricted to range over subsequences of σ. The subformula

$$batteryCharged\big(do(chargeBatt, s)\big)$$

is in turn bounded by a term rooted at s, $do(chargeBatt, s)$. The existentially quantified variable s ranges over the subsequences of $do(get\_coffee, S_0)$. Notice that the subformula refers to the situation $do(chargeBatt, s)$ which is not a prefix of σ. See Fig. 2 for a depiction of the situation tree of this example.

In the above example, we showed that bounded formulas may refer to situations that do not directly lie on the bounding sequence of actions. This allows reasoning about what would have held if the evolution of the world had diverted in a

[2] As before, any $\sqsubset$ may be replaced by $\sqsubseteq$ or $=$.

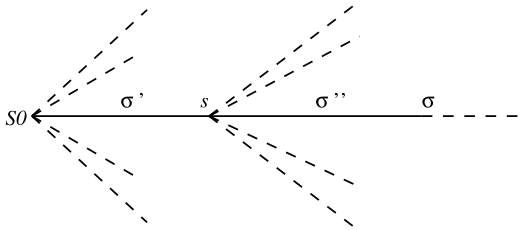

Fig. 3. In contrast to bounded formulas, in strictly bounded formulas the term σ'' rooted at the quantified variable s must itself be one of σ's preceding situations.

different direction starting from some past point. Another class of formulas with a further restriction will be useful later for defining Non-Markovian Basic Action Theories. Formulas in this class are only allowed to refer to situations that are predecessors of the bounding situation. In terms of Fig. 1, situation variables would be restricted to the situations between s and σ. They would not be allowed to deviate from this line as does σ'' in that figure. Formally, we introduce a strict version of boundedness below.

Definition 8 *(Simple situation terms).* A situation term $do([\alpha_1, \ldots, \alpha_n], s)$ is *simple* iff all the terms of sort situation mentioned by $\alpha_1, \ldots, \alpha_n$ are prefixes of $do([\alpha_1, \ldots, \alpha_n], s)$.

Clearly, simple situation terms are rooted. The situation term

$$do\big(A\big(f\big(do(B, s)\big)\big), s\big)$$

is rooted but not simple since $do(B, s)$ is not a prefix of it. The term

$$do\big(A\big(f(s)\big), s\big)$$

is simple.

Definition 9 *(Strictly bounded formulas).* Let σ be a simple situation term. The formulas of $\mathcal{L}_{sitcalc}$ *strictly bounded* by σ are the smallest set of formulas such that:

(1) If W is an atom whose terms of sort *situation* are all prefixes of σ, then W is strictly bounded by σ.
(2) If
  (a) σ' is a simple situation term that does not mention variable s,
  (b) σ'' is a simple situation term rooted at s, and
  (c) W is a propositional combination of formulas each strictly bounded by σ'' or by σ,
  then[3] $(\exists s\colon\ \sigma' \sqsubset \sigma'' \sqsubset \sigma)W$ is a formula strictly bounded by σ.
(3) If W_1, W_2 are formulas strictly bounded by σ, then $\neg W_1$, $W_1 \wedge W_2$ and $(\exists v)W_1$, where v is of sort *action* or *object*, are formulas strictly bounded by σ.

Clearly, the formula from Example 1, whose possible situation instances are shown in Fig. 2, is not a strictly bounded formula. If we replace the situation term $do(chargeBatt, s)$ in that formula with $do(a, s)$ we obtain the following formula which is strictly bounded:

$$(\exists a)\big(\exists s\colon\ S_0 \sqsubseteq do(a, s) \sqsubset \sigma\big)batteryCharged\big(do(a, s)\big).$$

The following example shows that strictly bounded formulas are expressive enough to encode Past Linear Temporal Logic modalities.

Example 2. In the Situation Calculus, referring to the past means referring to past situations. In this sense, one can write formulas that capture the intuitive meaning of the past linear temporal logic connectives **S** (since), $\overleftarrow{\Diamond}$ (sometime in the past), $\overleftarrow{\Box}$ (always in the past), and $\overleftarrow{\bigcirc}$ (previous) as follows (the φ's below denote Situation Calculus formulas with a single free variable of sort *situation* that has been suppressed. $\varphi[s]$ denotes a situation suppressed formula with a variable s reinstated):

$$(\varphi_1\ \mathbf{S}\ \varphi_2)[s] \stackrel{\text{def}}{=} \big(\exists s'\colon\ s' \sqsubset s\big)\big\{\varphi_2[s'] \wedge \big(\forall s''\colon\ s' \sqsubset s'' \sqsubseteq s\big)\varphi_1\big[s''\big]\big\},$$

[3] Any $\sqsubset$ may be replaced by $\sqsubseteq$ or $=$.

$$\big(\overleftarrow{\Diamond}\,\varphi\big)[s] \stackrel{\text{def}}{=} \big(\exists s'\colon\ s' \sqsubset s\big)\varphi\big[s'\big],$$
$$\big(\overleftarrow{\Box}\,\varphi\big)[s] \stackrel{\text{def}}{=} \big(\forall s'\colon\ s' \sqsubset s\big)\varphi\big[s'\big],$$
$$\big(\overleftarrow{\bigcirc}\varphi\big)[s] \stackrel{\text{def}}{=} (\exists a)\big(\exists s'\colon\ do\big(a, s'\big) = s\big)\varphi\big[s'\big].$$

It is easy to see that the above formulas are strictly bounded.

3.2. Non-Markovian Basic Action Theories

We are now ready to define action precondition axioms and successor state axioms for Non-Markovian Basic Action Theories.

Definition 10 *(Action precondition axioms).* An *action precondition axiom* is a sentence of the form:

$$Poss\big(A(x_1, \ldots, x_n), s\big) \equiv \Pi_A(x_1, \ldots, x_n, s),$$

where A is an n-ary action function symbol and $\Pi_A(x_1, \ldots, x_n, s)$ is a first order formula with free variables among $x_1, \ldots, x_n, s$ that is bounded by a situation term rooted at s and does not mention the predicate symbol *Poss*.

Example 3. Suppose that a robot works in a lab where there is a door that must not be opened if the temperature inside reached some dangerous level d after it was closed. The robot's theory would include a precondition axiom:

$$\begin{aligned} &Poss\big(open(Door), s\big) \equiv \\ &\quad \big(\exists s_1\colon\ S_0 \sqsubseteq do\big(close(Door), s_1\big) \sqsubseteq s\big). \\ &\qquad \big(\forall a, s_2 : s_1 \sqsubset do(a, s_2) \sqsubseteq s\big)a \neq open(Door) \wedge \\ &\qquad (\forall s_2\colon\ s_1 \sqsubset s_2 \sqsubseteq s)temp(Door, s_2) < d. \end{aligned}$$

Using the above temporal logic abbreviations plus the abbreviation $occ(a)[s] \stackrel{\text{def}}{=} (\exists s')do(a, s') = s$ we can put this formula in the following more readable form:

$$Poss\big(open(Door), s\big) \equiv \big(\neg occ\big(open(Door)\big) \wedge temp(Door) < d\big)\ \mathbf{S}\ occ\big(close(Door)\big)[s].$$

The rhs of action precondition axioms is not required to be strictly bounded but only bounded, which allows referring to situations branching away into "hypothetical futures." For instance, the above axiom could include a condition saying that after the robot does *open*(*Door*) it should not be possible for a human or robot to do something to get the human hurt: $Poss(open(Door), s) \equiv \cdots \wedge \neg(\exists a)hurt(person, do(a, do(open(Door), s)))$.

Definition 11 *(Successor state axioms).* A *successor state axiom* for an $(n+1)$-ary relational fluent F is a sentence of the form:

$$F\big(x_1, \ldots, x_n, do(a, s)\big) \equiv \Phi_F(x_1, \ldots, x_n, a, s),$$

where $\Phi_F(x_1, \ldots, x_n, a, s)$ is a first order formula with free variables among $x_1, \ldots, x_n, a, s$ that is strictly bounded by s and does not mention the constant S_0 nor the predicate symbol *Poss*.

A *successor state axiom* for an $(n+1)$-ary functional fluent f is a sentence of the form:

$$f\big(x_1, \ldots, x_n, do(a, s)\big) = y \equiv \phi_f(x_1, \ldots, x_n, y, a, s),$$

where $\phi_f(x_1, \ldots, x_n, y, a, s)$ is a first order formula with free variables among $x_1, \ldots, x_n,\ y, a, s$ that is strictly bounded by s and does not mention the constant S_0 nor the predicate symbol *Poss*.

The requirement that S_0 not appear in the rhs of successor state axioms is a technical detail introduced to simplify the proofs and is not essential (after all, it is variables, not S_0, which can cause trouble and require restrictions through notions like bounded formulas). Moreover, it is still possible to express conditions on the initial situation by referring to it indirectly. For instance, a condition that P hold initially can be expressed as $(\exists s_1\colon\ s_1 \sqsubset s)\{P(s_1) \wedge \neg(\exists s_2\colon\ s_2 \sqsubset s_1)True\}$.

On the other hand, the condition that the rhs of successor state axioms be strictly bounded cannot be relaxed. Bounded but not strictly bounded formulas can cause regression to fail. Consider the following successor state axioms:

$$P\big(do(a, s)\big) \equiv \big(\exists s'\colon\ s' \sqsubset s\big)Q\big(do\big([B_1, B_2, B_3], s'\big)\big),$$
$$Q\big(do(a, s)\big) \equiv \big(\exists s'\colon\ s' \sqsubset s\big)P\big(do\big([C_1, C_2, C_3], s'\big)\big).$$

Applying a few steps of regression to the atom $P(do([A_1, A_2], S_0))$, for instance, results in the atom $Q(do([B_1, B_2, B_3], S_0))$ and this in turn is regressed into

$$P\big(do\big([B_1, C_1, C_2, C_3], S_0\big)\big) \vee P\big(do\big([C_1, C_2, C_3], S_0\big)\big).$$

Clearly, regression is not working here since the situation terms will continue to lengthen. For action precondition axioms this problem does not occur because the predicate *Poss* is not allowed to appear on the rhs of these axioms.

Example 4. Consider again the robot that works at a bio-research lab. One of the successor state axioms in its theory could be:

$$\begin{aligned} &polluted\big(mat, do(a,s)\big) \equiv polluted(mat,s) \vee \\ &\qquad a = touch(mat) \wedge (\exists loc).safetyLevel(loc, Low) \wedge \\ &\qquad \big(\neg atLoc(DisinfSt)\ \mathbf{S}\ atLoc(loc)\big)[s]. \end{aligned}$$

Definition 12 *(Non-Markovian Basic Action Theories).* A *Non-Markovian Basic Action Theory* $\mathcal{D}$ is a theory of $\mathcal{L}_{sitcalc}$ consisting of the following set of axioms:

- The foundational axioms Σ.
- A set of successor state axioms $\mathcal{D}_{ss}$ as in Definition 10.
- A set of action precondition axioms $\mathcal{D}_{ap}$ as in Definition 11.
- A set of unique names axioms for actions $\mathcal{D}_{una}$.
- A set of first order sentences $\mathcal{D}_{S_0}$ that mention no situation terms other than S_0 and represent the initial state of the world.

3.3. Two relativity properties

Intuitively, a strictly bounded formula has the property that its truth value depends only on the past relative to the situation that is the bound, i.e., its truth value depends only on the truth value of fluents throughout such a situation (history) and on situation independent predicates and functions. The following theorem confirms this intuition.

Theorem 1. *Let S, S' be structures of $\mathcal{L}_{sitcalc}$ with the same domain Act for sort action, Obj for sort object and Sit for sort situation, and let v be a variable assignment ranging over these domains. Furthermore, let ϕ be an $\mathcal{L}_{sitcalc}$ formula that is strictly bounded by σ, does not mention Poss, and whose only free variable of sort situation, if any, is the root of σ. If*

(1) *S and S' satisfy the foundational axioms Σ;*
(2) *S' is the same;*
(3) *the interpretation of all action functions and situation independent functions and predicates in S and S' is the same;*
(4) *for each relational fluent $F(\vec{x}, s')$, if $S, v \models (s' \sqsubseteq \sigma)$ then*

$$S, v \models F\big(\vec{x}, s'\big) \quad \textit{iff} \quad S', v \models F\big(\vec{x}, s'\big);$$

(5) *for each functional fluent $f(\vec{x}, s')$, if $S, v \models (s' \sqsubseteq \sigma)$ then*[4]

$$f^S\big(v(\vec{x}), v\big(s'\big)\big) = f^{S'}\big(v(\vec{x}), v\big(s'\big)\big)$$

then

$$S, v \models \phi \quad \textit{iff} \quad S', v \models \phi.$$

Intuitively, conditions (1)–(3) make sure the structures S, S' coincide on the interpretation of the tree of situations and the situation independent part of the language. Then conditions (4) and (5) require S, S' to coincide on the value of all fluents at situations preceding σ. Then, according to the theorem, under these conditions structures S, S' coincide on the truth value of every formula ϕ that is strictly bounded by σ.

Lemma 1. *Let σ be a simple situation term, S, S', and v be as in the statement of Theorem* 1 *and suppose the conditions* (1)–(5) *are satisfied. Let t be a term of any sort such that all its subterms of sort situation are prefixes of σ. Then, $v^S(t) = v^{S'}(t)$.*

Proof. If t is a variable, the lemma is obvious. Suppose that t is not a variable. Suppose that $v^S(t) \neq v^{S'}(t)$. Then there must be a subterm $g(r_1, \ldots, r_n)$ of t such that $v^S(r_i) = v^{S'}(r_i)$, $i = 1, \ldots, n$, and $g^S(v^S(r_1), \ldots, v^S(r_n)) \neq g^{S'}(v^{S'}(r_1), \ldots, v^{S'}(r_n))$. If g is the function do, we get a contradiction by condition (2). If g is an action or a situation independent function, by condition (3) we get a contradiction. Finally, if g is a functional fluent symbol, then its last argument must be a prefix of σ. Then by condition (5) we get a contradiction. □

Corollary 1. *Let S, S' and v be as in Lemma* 1 *above, and let σ', σ be simple situation terms. Then*

$$S, v \models \sigma' = \sigma \quad \textit{iff} \quad S', v \models \sigma' = \sigma; \quad \textit{and}$$
$$S, v \models \sigma' \sqsubset \sigma \quad \textit{iff} \quad S', v \models \sigma' \sqsubset \sigma.$$

[4] $v(\vec{x})$ denotes $v(x_1), \ldots, v(x_n)$ where the x_i's are the variables in $\vec{x}$. $v(x_i)$ is the value assigned to x_i by the variable assignment v.

Proof. Since both σ' and σ are simple situation terms, by Lemma 1 $v^S(\sigma') = v^{S'}(\sigma')$ and $v^S(\sigma) = v^{S'}(\sigma)$. The corollary then follows from this and condition (1). □

Proof of Theorem 1. Let the structures S, S', variable assignment v, and formula ϕ be as described in the statement of the theorem and suppose that the conditions (1)–(5) are satisfied.

The proof is by induction on the syntactic structure of ϕ. We have the following cases:

(1) Suppose ϕ is an atom of the form $\sigma' \sqsubset \sigma$ or $\sigma' = \sigma$. Then by definition of strictly bounded formulas, all terms of sort situation in σ' must be prefixes of σ and hence σ' is simple. The theorem then follows from Corollary 1.
(2) Suppose ϕ is an atom of the form $t_1 = t_2$ where t_1, t_2 are terms of sort *action* or *object*. As before, all terms of sort situation in t_1, t_2 must be prefixes of σ. The theorem then follows from Lemma 1.
(3) Suppose ϕ is a situation independent relational atom $P(\vec{t})$, where $\vec{t}$ are terms of sort *action* or *object*. Then all terms of sort situation in $\vec{t}$ are prefixes of σ. The theorem then follows from Lemma 1 and condition (3).
(4) Suppose ϕ is a relational fluent atom $F(\vec{t}, \sigma')$. Then all terms of sort situation in $\vec{t}$, σ' are prefixes of σ. The theorem then follows from Lemma 1 and condition (4).
(5) Suppose ϕ is a non-atomic formula of the form $(\exists s'\colon\ \sigma'' \sqsubset \sigma' = \sigma)W$ or $(\exists s'\colon\ \sigma'' \sqsubset \sigma' \sqsubset \sigma)W$,[5] where W is a propositional combination of formulas each strictly bounded by σ' or by σ. Let $v^{\mathbf{s}}$ be a variable assignment identical to v except for mapping the variable s' to element $\mathbf{s}$ of Sit. By definition of strictly bounded formulas, σ'' and σ' are simple situation terms. Thus by Corollary 1, $S, v^{\mathbf{s}} \models \sigma'' \sqsubset \sigma' = \sigma$ iff $S', v^{\mathbf{s}} \models \sigma'' \sqsubset \sigma' = \sigma$, for any $\mathbf{s}$.
Suppose that $S, v \not\models (\exists s'\colon\ \sigma'' \sqsubset \sigma' = \sigma)$. Then by the corollary, $S', v \not\models (\exists s'\colon\ \sigma'' \sqsubset \sigma' = \sigma)$. It follows that $S, v \not\models \phi$ and $S', v \not\models \phi$ and thus the theorem holds.
Suppose that $S, v \models (\exists s'\colon\ \sigma'' \sqsubset \sigma' = \sigma)$. Then by the corollary $S', v \models (\exists s'\colon\ \sigma'' \sqsubset \sigma' = \sigma)$. By induction, for each subformula W' of W that is strictly bounded by σ, we have $S, v \models W'$ iff $S, v \models W'$. Similarly, by induction, for each subformula W' of W that is strictly bounded by σ', we have $S, v \models W'$ iff $S, v \models W'$. It follows that the theorem holds.
(6) For non-atomic formulas of the form $\neg\phi$, $\phi_1 \wedge \phi_2$, and $(\exists u)\phi$ (see item (3) of Definition 9) the theorem follows by induction on the structure of ϕ. □

An important property of Markovian basic action theories, called *Relative Satisfiability*, says that if the initial database $\mathcal{D}_{S_0}$ together with a set $\mathcal{D}_{una}$ of unique names axioms for actions is satisfiable, then adding the foundational axioms, Σ, and any set of action precondition and successor state axioms, results in a satisfiable theory, provided the successor state axioms for functional fluents satisfy a consistency property (see Eq. (6) below).

We prove next that satisfiability is also preserved after adding non-Markovian action precondition and successor state axioms to a satisfiable initial database with unique names axioms.

Theorem 2 *(Relative satisfiability). A Non-Markovian Basic Action Theory $\mathcal{D}$ is satisfiable iff $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ is satisfiable.*

Proof. Starting from a model M_0 of $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, we construct a model M of $\mathcal{D}$.

(1) Satisfying Σ:
Let *Act* and *Obj* be the domains for sorts *action* and *object*, respectively, in M_0. We define a structure M by complete induction.
(a) Let the same *Act* and *Obj* be the domains for sorts *action* and *object* in M. We define the domain *Sit* for sort *situation* as the set of all finite sequences of elements of *Act*. That is, if $\{\alpha_1, \ldots, \alpha_n\} \subseteq Act$ then $[\alpha_1, \ldots, \alpha_n] \in Sit$.
(b) Next define the interpretation of S_0 and *do* in M as follows:

$$S_0^M = [\,],$$
$$do^M\big(\alpha, [\alpha_1, \ldots, \alpha_n]\big) = [\alpha_1, \ldots, \alpha_n, \alpha],$$

where $\alpha \in Act$ and $[\alpha_1, \ldots, \alpha_n] \in Sit$.
(c) Define the interpretation of $\sqsubset$ to be $\sigma \sqsubset^M \sigma'$ whenever σ is a proper prefix of σ'.
It is easy to verify that M as defined so far satisfies the foundational axioms Σ.
We continue by defining the interpretation of all the predicates other than *Poss*, whose interpretation is defined afterward, and $\sqsubset$ whose interpretation is already defined. We will also define now the interpretation of all functions, except for *do* whose interpretation is given above. The predicates and functions include fluents, and so we need to provide an interpretation with respect to all situations. This is where the use of a definition by complete induction is needed. We use induction on the length of the sequences and complete induction is required as the successor state axioms may refer to past situations. After giving such an interpretation, the only remaining step will be to define the interpretation

[5] The proof is the same if the first $\sqsubset$ is replaced with $\sqsubseteq$ or $=$.

of *Poss*, which will be straight forward since action precondition axioms do not themselves mention the *Poss* predicate in their rhs (see Definition 10).

Let us proceed to define the interpretation under M of predicate and function symbols other than *do*, *Poss*, and $\sqsubset$.

(2) Interpretation of situation independent predicates and functions:
For situation independent predicates and functions, the sort *situation* and its domain *Sit* are irrelevant and the domains *Act* and *Obj* are the same in M as in M_0. Thus, let M interpret situation independent predicates and functions just as they are in M_0. It follows that since M_0 satisfies $\mathcal{D}_{una}$ so does M.

(3) Interpretation of fluents:
Let σ be a sequence in *Sit* and assume that for every variable assignment v that assigns σ to s, the following holds: if $M, v \models s' \sqsubseteq s$ then M, v interprets every relational fluent $F(\vec{x}, s')$ and assigns a value in $Obj \cup Act$ to every functional fluent $f(\vec{x}, s')$.

(a) Interpretation of fluents in the initial situation:
Suppose that σ is the empty sequence [ ]. Then $S_0^M = \sigma$. Let F be a relational fluent, f a functional fluent and v_0 a variable assignment for variables of sorts *object* and *action* (for either structure since they share domains for these sorts). Define M such that:

$$M, v_0 \models F(\vec{x}, S_0) \quad \text{iff} \quad M_0, v_0 \models F(\vec{x}, S_0),$$
$$f^M\big(\vec{x}[v_0], S_0^M\big) = f^{M_0}\big(\vec{x}[v_0], S_0^{M_0}\big).$$

Recall that functional fluents range over sorts *action* or *object*. They cannot range over sort *situation*. Also, all the sentences in $\mathcal{D}_{S_0}$ are uniform in S_0. This and the fact that M_0 satisfies $\mathcal{D}_{S_0}$ implies that M is also a model of $\mathcal{D}_{S_0}$.

(b) Interpretation of fluents in situations other than S_0:
Suppose that σ is the sequence $[\alpha_1, \ldots, \alpha_n]$, $n \geqslant 1$.
Let F be a relational fluent and let its successor state axiom be

$$F\big(\vec{x}, do(a, s)\big) \equiv \Phi_F(\vec{x}, a, s).$$

For every variable assignment v assigning σ to s we have by induction that if $M, v \models s' \sqsubseteq s$ then M, v interprets all relational fluents $F(\vec{x}, s')$.
From Theorem 1, it follows that M, v must assign a value to $\Phi_F(\vec{x}, a, s)$, which is strictly bounded by s and does not mention *Poss*. If M, v did not assign a value to this formula, then we could extend M so that it does assign a value to it, and construct another extended model M' that assigns a different value to $\Phi_F(\vec{x}, a, s)$, which would contradict the theorem.
Furthermore, since M satisfies the foundational axioms Σ, we have that $M, v \models s \sqsubset do(a, s)$, which implies that $M, v \not\models do(a, s) \sqsubseteq s$ by the same axioms Σ. Thus M, v does not yet assign a value to $F(\vec{x}, do(a, s))$.
We can then define M to interpret $F(\vec{x}, do(a, s))$ as follows:

$$M, v \models F\big(\vec{x}, do(a, s)\big) \quad \text{iff} \quad M, v \models \Phi_F(\vec{x}, a, s).$$

The interpretation of functional fluents in situation $do(a, s)$ is defined similarly. For each functional fluent f with successor state axiom

$$f\big(\vec{x}, do(a, s)\big) = y \equiv \phi_f(\vec{x}, y, a, s)$$

and each variable assignment v assigning σ to s, define

$$M, v \models f\big(\vec{x}, do(a, s)\big) = y \quad \text{iff} \quad M, v \models \phi_F(\vec{x}, y, a, s).$$

A similar argument to the one above shows that this is well defined. Furthermore, the functional fluent consistency property which states that:

$$\mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models (\forall a, s, \vec{x}).(\exists y)\phi_f(\vec{x}, y, a, s) \wedge \big[\big(\forall y, y'\big).\phi_f(\vec{x}, y, a, s) \wedge \phi_f\big(\vec{x}, y', a, s\big) \supset y = y'\big], \tag{6}$$

together with the fact that M satisfies $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, guarantee that one and only one value y is assigned to $f(\vec{x}, do(a, s))$ above.
This completes the inductive definition of how M interprets fluents. By construction, M satisfies all successor state axioms.

It remains to specify how M interprets $Poss(a, s)$.

(4) Interpretation of $Poss(a, s)$:
Consider an element $\alpha \in Act$. There are two possible cases:

(a) There is a variable assignment v such that v assigns α to a and there is an action function $A(\vec{x})$ such that $M, v \models a = A(\vec{x})$.
By definition, $\mathcal{D}_{ap}$ includes an axiom $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$ and $\Pi_A(\vec{x}, s)$ does not mention predicate *Poss*. Hence, M already assigns a value to $\Pi_A(\vec{x}, s)$. We specify that

$$M, v \models Poss\big(A(\vec{x}), s\big) \quad \text{iff} \quad M, v \models \Pi_A(\vec{x}, s).$$

Since M satisfies the unique names axioms for actions, the value assigned to *Poss* is unique.

(b) For every variable assignment v that assigns α to a and every function $A(\vec{x})$, $M, v \not\models a = A(\vec{x})$. In this case, any value assigned to $Poss(a, s)$ will satisfy the action precondition axioms.

By construction, M satisfies all the action precondition and successor state axioms of $\mathcal{D}$. This completes the proof of relative satisfiability. □

4. Regression of bounded formulas

The particular syntactic form of action precondition and successor state axioms is computationally advantageous. These axioms are definitions: the right-hand side serves as the definition of the predicate on the left-hand side, and this makes them exchangeable in a logically equivalent way. Reiter's regression operator [5,21], $\mathcal{R}$, exploits this property to provide a specialized theorem proving mechanism for the $\mathcal{L}_{sitcalc}$ class of regressable formulas.

In this section, we make two generalizations:

- we define a strictly larger class of formulas that includes Reiter's original class of regressable formulas;
- we define a generalized operator for regressing these formulas and prove that it is sound and complete.

The generalized class of regressable formulas is based on the notion of bounded formula defined earlier. This will allow us to apply regression with respect to Non-Markovian Action Theories. We remark, however, that even when the background theory consists of standard Markovian axioms, the generalized operator is useful since it allows regressing many formulas which are not regressable by the original operator. In particular, it will allow us to prove entailment of formulas that mention the $\sqsubset$ relation, which is not possible with the original regression operator. We make further comments on this below.

4.1. Regressable formulas

We start by generalizing the class of formulas on which the regression operator will be applicable.

Definition 13 *(Regressable formulas).* A formula W of $\mathcal{L}_{sitcalc}$ is *regressable* iff

(1) W is first order; and
(2) W is bounded by a situation term rooted at S_0 and has no free variables of sort *situation*; and
(3) for every atom of the form $Poss(\alpha, \sigma)$ mentioned by W, α has the form $A(t_1, \ldots, t_n)$ for some n-ary action function symbol A of $\mathcal{L}_{sitcalc}$.

Example 5. The original definition of regressable formulas requires all terms of sort *situation* to be rooted at S_0, does not allow quantifying over situations, nor the mention of predicate $\sqsubset$, nor equality between situation terms.

The above definition of regressable formulas also includes, for example, the following: Suppose that σ is a ground situation term, e.g., a database transaction log or a ground plan. We can use regression to answer queries such as:

- has block x always been on top of block y during (the execution of) σ?

  $(\forall s\colon s \sqsubseteq \sigma) on(x, y, s)$;

- is there an action that would have resulted in block x being on top of block y ever during σ?

  $(\exists a)(\exists s\colon s \sqsubseteq \sigma) on\big(x, y, do(a, s)\big)$;

- same as the previous query, but also requiring that the action actually be executable:

  $(\exists a)(\exists s\colon s \sqsubseteq \sigma) Poss(a, s) \wedge on\big(x, y, do(a, s)\big)$;

- was action *shoot* ever possible during σ and would it have killed the turkey?

  $(\exists s\colon s \sqsubseteq \sigma) Poss(shoot, s) \wedge dead\big(turkey, do(shoot, s)\big)$.

4.2. Generalized regression operator

Let us proceed to generalize Reiter's regression operator $\mathcal{R}$ for the above class of formulas. We assume henceforth, without loss of generality, that formulas have had their quantified variables renamed to be distinct from all free variables. We use $W|^{t_1}_{t_2}$ to denote the formula obtained from W by replacing the term t_1 with t_2.

Definition 14 *(Prime functional fluent term).* (See [21].) A functional fluent term is *prime* iff it has the form $f(\vec{t}, do([\alpha_1, \ldots, \alpha_n], S_0))$ for $n \geqslant 1$ and each of the terms $\vec{t}, \alpha_1, \ldots, \alpha_n$ is uniform in S_0.

Situation Calculus terms without free variables of sort *situation* have the property of containing a prime functional fluent:

Remark 1. (See [21].) Let $g(\vec{t}, do(\alpha, \sigma))$ be an $\mathcal{L}_{sitcalc}$ term without free variables of sort situation, i.e., all its subterms of sort situation are of the form $do([\alpha_1, \ldots, \alpha_n], S_0)$, $n \geqslant 0$. Then $g(\vec{t}, do(\alpha, \sigma))$ mentions a prime functional fluent term.

Definition 15 *(Generalized regression operator).* Let W be a regressable formula of $\mathcal{L}_{sitcalc}$.

(1) If W is an atom, then by the definition of regressable it is bounded by a situation term rooted at S_0 and from this and the definition of bounded formulas, it follows that W must be uniformly rooted at S_0. Reiter's regression operator as originally defined[6] works for these formulas. For completeness we define the operator here anyway. By virtue of being regressable, the atom W must be of one of the following forms:
   (a) An equality atom of the form
   $$do\big([\alpha'_1, \ldots, \alpha'_m], S_0\big) = do\big([\alpha_1, \ldots, \alpha_n], S_0\big).$$
   If $m = n = 0$, $\mathcal{R}[W] = true$.
   If $m \neq n$, $\mathcal{R}[W] = false$.
   If $m = n \geqslant 1$ then $\mathcal{R}[W] = \mathcal{R}[\alpha'_1 = \alpha_1 \wedge \cdots \wedge \alpha'_m = \alpha_m]$.
   (b) A $\sqsubset$-atom of the form
   $$do\big([\alpha'_1, \ldots, \alpha'_m], S_0\big) \sqsubset do\big([\alpha_1, \ldots, \alpha_n], S_0\big).$$
   If $m = 0$ and $n \geqslant 1$, then $\mathcal{R}[W] = true$.
   If $m \geqslant n$, then $\mathcal{R}[W] = false$.
   If $1 \leqslant m < n$, then $\mathcal{R}[W] = \mathcal{R}[\alpha'_1 = \alpha_1 \wedge \cdots \wedge \alpha'_m = \alpha_m]$.
   (c) An atom $Poss(\alpha, \sigma)$ where α and σ are terms of sort *action* and *situation* respectively. By definition of regressable formulas, the term α is of the form $A(\vec{t})$, where A is an action function symbol with a corresponding action precondition axiom $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$ in $\mathcal{D}_{ap}$. Then,
   $$\mathcal{R}[W] = \mathcal{R}\big[\Pi_A(\vec{t}, \sigma)\big].$$
   The remaining possible cases are the following:
   (d) An atom whose only situation term is S_0 or a situation independent atom.
   Then $\mathcal{R}[W] = W$.
   (e) An atom that mentions a functional fluent term of the form
   $$g\big(\vec{t}', do(\alpha', \sigma')\big).$$
   Since it is an atom bounded by a situation term rooted at S_0, by definition of bounded it must be uniformly rooted at S_0. Hence, the atom does not mention any variables of sort situation and thus, by Remark 1, it must mention a prime functional fluent term $f(\vec{t}, do(\alpha, \sigma))$. This fluent term is such that α is uniform in S_0 and σ is rooted at S_0. Let $f(\vec{x}, do(a, s)) = y \equiv \phi_f(\vec{x}, y, a, s)$ be this functional fluent term's successor state axiom in $\mathcal{D}_{ss}$. Then,
   $$\mathcal{R}[W] = \mathcal{R}\big[(\exists v).\phi_f(\vec{t}, v, \alpha, \sigma) \wedge W\big|_v^{f(\vec{t}, do(\alpha, \sigma))}\big],$$
   where v is a fresh new variable.
   (f) A relational fluent atom $F(\vec{t}, do(\alpha, \sigma))$ that contains no functional fluent terms.
   Let $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$ be this fluent's successor state axiom in $\mathcal{D}_{ss}$.
   Then $\mathcal{R}[W] = \mathcal{R}[\Phi_F(\vec{t}, \alpha, \sigma)]$.

(2) When W is a regressable formula of the form[7] $(\exists s\colon\ \sigma' \sqsubset \sigma'' \sqsubset \sigma)W'$ or $(\exists s\colon\ \sigma' \sqsubset \sigma'' = \sigma)W'$ we have the following subcases:
   (a) Suppose W is a regressable formula of the form
   $$\big(\exists s\colon\ \sigma' \sqsubset do\big([\alpha'_1, \ldots, \alpha'_m], s\big) \sqsubset do\big([\alpha_1, \ldots, \alpha_n], S_0\big)\big)W',$$
   where $m \geqslant 0$.
   If $m \geqslant n$, then $\mathcal{R}[W] = false$.
   If $m < n$, then
   $$\mathcal{R}[W] = \mathcal{R}\big[\big(\exists s\colon\ \sigma' \sqsubset do\big([\alpha'_1, \ldots, \alpha'_m], s\big) = do\big([\alpha_1, \ldots, \alpha_{n-1}], S_0\big)\big)W'\big] \vee \mathcal{R}\big[\big(\exists s\colon\ \sigma' \sqsubset do\big([\alpha'_1, \ldots, \alpha'_m], s\big) \sqsubset do\big([\alpha_1, \ldots, \alpha_{n-1}], S_0\big)\big)W'\big].$$

[6] We are referring to the definition of the regression operator as it appears in [21], which is slightly more general than the definition given in [2].
[7] The definition of regression of the corresponding formulas with $\sigma' = \sigma''$ instead of $\sigma' \sqsubset \sigma''$ is defined in the same way.

(b) Suppose W is a regressable formula of the form

$$\big(\exists s{:}\ \sigma' \sqsubset do\big(\big[\alpha'_1, \ldots, \alpha'_m\big], s\big) = do\big([\alpha_1, \ldots, \alpha_n], S_0\big)\big)W',$$

where $m \geqslant 1$. By the same argument as in the previous case, σ' is rooted at S_0.
If $m > n$, then $\mathcal{R}[W] = \mathit{false}$.
If $m \leqslant n$, then

$$\mathcal{R}[W] = \mathcal{R}\big[\big(\exists s{:}\ S_0 \sqsubseteq do\big(\big[\alpha'_1, \ldots, \alpha'_{m-1}\big], s\big) = do\big([\alpha_1, \ldots, \alpha_{n-1}], S_0\big)\big)$$
$$\big(\alpha''_m = \alpha_n\big) \wedge \sigma' \sqsubset do\big([\alpha_1, \ldots, \alpha_n], S_0\big) \wedge W'\big],$$

where α''_m stands for $\alpha'_m|^s_{do([\alpha_1,\ldots,\alpha_{n-m}], S_0)}$.[8]

(c) Suppose W is a regressable formula of the form

$$\big(\exists s{:}\ \sigma' \sqsubset s = do\big([\alpha_1, \ldots, \alpha_n], S_0\big)\big)W'.$$

By definition of bounded formulas, σ' does not mention the variable s. Moreover, quantifiers on situation variables can only be introduced through the abbreviations (5). Hence, if σ' does mention a variable, then this variable must be a free variable. This is not possible by definition of regressable, thus σ' must be rooted at S_0.
The regression is then defined as follows:

$$\mathcal{R}[W] = \mathcal{R}\big[\sigma' \sqsubset do\big([\alpha_1, \ldots, \alpha_n], S_0\big) \wedge W'|^s_{do([\alpha_1,\ldots,\alpha_n], S_0)}\big].$$

(3) For the remaining possibilities, regression is defined as follows:

$$\mathcal{R}[\neg W] = \neg\mathcal{R}[W],$$
$$\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2].$$

For a variable v of any sort other than *situation*:

$$\mathcal{R}\big[(\exists v)W\big] = (\exists v)\mathcal{R}[W].$$

Example 6. Consider a sentence $G(s)$ saying that at some point in the past relative to s, doing action $putdown(x)$, for some block x, would have made action $stack(A, B)$ possible:

$$G(s) \stackrel{\text{def}}{=} \big(\exists s'{:}\ S_0 \sqsubseteq s' \sqsubseteq s\big)(\exists x)Poss\big(stack(A, B), do\big(putdown(x), s'\big)\big).$$

Suppose that we want to determine whether G holds after executing the sequence $[unstack(D, B), stack(D, A)]$, i.e., whether

$$\mathcal{D} \models G\big(do\big(stack(D, A), do\big(unstack(D, B), S_0\big)\big)\big).$$

It is easy to check that $G(do(stack(D, A), do(unstack(D, B), S_0)))$ is regressable according to Definition 13. Applying regression to this formula yields the following:

$$\begin{aligned}
&\mathcal{R}\big[G\big(do\big(stack(D, A), do\big(unstack(D, B), S_0\big)\big)\big)\big]\\
&=\\
&\mathcal{R}\big[\big(\exists s'{:}\ S_0 \sqsubseteq s' \sqsubseteq do\big(stack(D, A), do\big(unstack(D, B), S_0\big)\big)\big)\\
&\qquad\qquad (\exists x)Poss\big(stack(A, B), do\big(putdown(x), s'\big)\big)\big]\\
&= \qquad\qquad \text{(by Case 3)}\\
&\mathcal{R}\big[\big(\exists s'{:}\ S_0 \sqsubseteq s' = do\big(stack(D, A), do\big(unstack(D, B), S_0\big)\big)\big)\\
&\qquad\qquad (\exists x)Poss\big(stack(A, B), do\big(putdown(x), s'\big)\big)\big] \vee\\
&\mathcal{R}\big[\big(\exists s'{:}\ S_0 \sqsubseteq s' \sqsubset do\big(stack(D, A), do\big(unstack(D, B), S_0\big)\big)\big)\\
&\qquad\qquad (\exists x)Poss\big(stack(A, B), do\big(putdown(x), s'\big)\big)\big]\\
&= \qquad\qquad \text{(by Cases 2c, 2a)}\\
&\mathcal{R}\big[(\exists x)Poss\big(stack(A, B), do\big(\big[unstack(D, B), stack(D, A), putdown(x)\big], S_0\big)\big)\big]\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee\\
&\mathcal{R}\big[\big(\exists s'{:}\ s' = do\big(unstack(D, B), S_0\big)\big)(\exists x)Poss\big(stack(A, B), do\big(putdown(x), s'\big)\big)\big]\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee\\
&\mathcal{R}\big[\big(\exists s'{:}\ s' \sqsubset do\big(unstack(D, B), S_0\big)\big)(\exists x)Poss\big(stack(A, B), do\big(putdown(x), s'\big)\big)\big]
\end{aligned}$$

[8] Replacing s with $do([\alpha_1, \ldots, \alpha_{n-m}], S_0)$ is only necessary to comply with the definition of bounded formulas. In fact, the replacement would take place in subsequent regression steps if it were not done here.

$$
\begin{aligned}
&= && \text{(by Cases 2c, 2a)}\\
&\mathcal{R}\big[(\exists x)Poss\big(stack(A,B), do\big(\big[unstack(D,B), stack(D,A), putdown(x)\big], S_0\big)\big)\big] \vee\\
&\mathcal{R}\big[(\exists x)Poss\big(stack(A,B), do\big(\big[unstack(D,B), putdown(x)\big], S_0\big)\big)\big] \vee\\
&\mathcal{R}\big[(\exists s'\colon s' = S_0)(\exists x)Poss\big(stack(A,B), do\big(putdown(x), s'\big)\big)\big] \vee\\
&\mathcal{R}\big[\big(\exists s'\colon s' \sqsubset S_0\big)(\exists x)Poss\big(stack(A,B), do\big(putdown(x), s'\big)\big)\big]\\
&= && \text{(by Cases 2c, 2a)}\\
&\mathcal{R}\big[(\exists x)Poss\big(stack(A,B), do\big(\big[unstack(D,B), stack(D,A), putdown(x)\big], S_0\big)\big)\big] \vee\\
&\mathcal{R}\big[(\exists x)Poss\big(stack(A,B), do\big(\big[unstack(D,B), putdown(x)\big], S_0\big)\big)\big] \vee\\
&\mathcal{R}\big[(\exists x)Poss\big(stack(A,B), do\big(putdown(x), S_0\big)\big)\big].
\end{aligned}
$$

The remaining steps are the same as with Reiter's original regression operator.

4.3. Correctness of regression

Our proof of correctness of operator $\mathcal{R}$ is based on the proof of correctness of the original regression operator [21]. The main technical difficulty here is that with Non-Markovian Action Theories and the generalized regressable formulas, during regression situation terms can grow "longer" before they start "shrinking" down to S_0, which makes the induction a bit more complicated.

Theorem 3. *Suppose W is a regressable formula of $\mathcal{L}_{sitcalc}$ and $\mathcal{D}$ is a basic Non-Markovian Action Theory. Then,*

(1) $\mathcal{R}[W]$ *is a formula uniform in* S_0.
(2) $\mathcal{D} \models (\forall).W \equiv \mathcal{R}[W]$.

Proof. The proof is by induction based on an ordering relation $\prec$ on tuples of integers. We start by defining this ordering.

Let Λ be the set of all countably infinite sequences of natural numbers with a finite number of nonzero elements. Define the binary relation $\prec$ on this set as the reverse lexicographic ordering:

$$(\lambda_1, \lambda_2, \ldots) \prec \big(\lambda'_1, \lambda'_2, \ldots\big) \quad \text{iff} \quad \text{for some } m,\ \lambda_m < \lambda'_m, \text{ and for all } n > m,\ \lambda_n = \lambda'_n.$$

The set Λ satisfies well ordering. Next, overload $\prec$ by defining the following lexicographic ordering on $\Lambda \times \mathbb{N}$:

$$(\lambda, n) \prec \big(\lambda', n'\big) \quad \text{iff} \quad n < n' \text{ or } n = n' \text{ and } \lambda \prec \lambda'.$$

Clearly, any subset of $\Lambda \times \mathbb{N}$ has a minimal element and so $\prec$ provides a well-ordering on this set. It is worth remarking that the ordering priority in tuple elements increases from left to right.

Before we define our mapping from $\mathcal{L}_{sitcalc}$ regressable formulas to tuples in $\Lambda \times \mathbb{N}$ we need the following notions.

If $g(t_1, \ldots, t_n)$ is an $\mathcal{L}_{sitcalc}$ term, then $t_1, \ldots, t_n$ are said to be *proper subterms* of $g(t_1, \ldots, t_n)$. An occurrence of a situation term in an $\mathcal{L}_{sitcalc}$ formula W is said to be *maximal* iff its occurrence is not as a proper subterm of some situation term. Intuitively, the occurrence of all terms $do([\alpha_1, \ldots, \alpha_i], \sigma)$, $i = 0, \ldots, n-1$, in a term $do([\alpha_1, \ldots, \alpha_n], \sigma)$ are not maximal since each is a proper subterm of $do([\alpha_1, \ldots, \alpha_{i+1}], \sigma)$.

For a regressable formula W, let $L(W)$ be the sum of the length of σ for each occurrence of σ in W such that:

- σ is a term of sort *situation* rooted at a situation variable;
- the occurrence of σ in W is maximal;
- if the occurrence is in a quantifier expression $(Qs\colon \sigma' \sqsubset \sigma'' \sqsubset \sigma)$, then its length is larger than the length of σ''.

We are now ready to define the mapping. Define $index(W)$ to be:

$$index(W) \stackrel{\text{def}}{=} \big((C, E, I, \lambda_1, \lambda_2, \ldots), P\big),$$

where

(1) C is the number of connectives and quantifiers,
(2) E is the number of equality atoms between situation terms,
(3) I is the number of $\sqsubset$-atoms,
(4) for $m \geqslant 1$, λ_m is the number of maximal occurrences in W of situation terms of length $m - L(W)$ that are rooted at S_0,
(5) P is the number of *Poss* atoms mentioned by W.

Notice that the λ's here are "shifted" to the right a number $L(W)$ of places. In other words, if one maximal term of length k occurs in W, then $\lambda_{k+L(W)} = 1$. The reason behind this is that after a regression step on a formula with a situation variable, it is possible that a situation term that mentions such a variable be replaced by a longer one. For instance, the formula $(\exists s\colon s = do(A, S_0))P(do(B, s))$ which has $\lambda_1 = 0$, $\lambda_{1+1} = 1$, would be regressed into $P(do([A, B], S_0))$ which also has

$\lambda_1 = 0$, $\lambda_{0+2} = 1$. Without the shift of the λ's, the former formula would have $\lambda_1 = 1$, $\lambda_2 = 0$ while the latter would have the same $\lambda_1 = 0$, $\lambda_{0+2} = 1$. Thus the index of the first formula would precede the index of the second one, which has been regressed one step. With the shifted λ's, the precedence in this case is decided based only on the number of connectives and quantifiers.

Let us proceed with the proof. Consider a regressable formula W and assume the theorem for all regressable formulas with index $\prec \mathit{index}(W)$.

(1) Suppose that W is a regressable atom. Then W is an atom bounded by a situation term rooted at S_0 and has no free variables of sort *situation*. By definition of bounded formulas, W is uniformly rooted at S_0. This implies that $L(W) = 0$. We have the following cases:

(a) Suppose that W is an equality atom of the form

$$do\big([\alpha'_1, \ldots, \alpha'_m], S_0\big) = do\big([\alpha_1, \ldots, \alpha_n], S_0\big).$$

This atom is regressed into *true* if $m = n = 0$, *false* if $m \neq n$. Otherwise, we have that

$$\mathcal{R}[W] = \mathcal{R}\big[\alpha'_1 = \alpha_1 \wedge \cdots \wedge \alpha'_m = \alpha_m\big].$$

In the former two cases, the theorem follows immediately. Consider the case where $m = n \geqslant 1$ and let W^* stand for $\alpha'_1 = \alpha_1 \wedge \cdots \wedge \alpha'_m = \alpha_m$.

Since W is a regressable atom, it is bounded by a situation term rooted at S_0, and therefore uniformly rooted at S_0. This implies that each $\alpha'_i = \alpha_i$ is also uniformly rooted at S_0 and hence that the conjunction W^* is uniformly rooted at S_0. Therefore W^* is bounded by a situation term rooted at S_0 and thus regressable.

The index of W^* differs from the index of W in that λ_m is smaller by at least two in $\mathit{index}(W^*)$ while the number P and all other λ_i remain the same. For instance, if α_i's and α'_j's do not mention any situation terms, then $\mathit{index}(W^*) = ((m, 0, 0, 0, \ldots), 0)$ and $\mathit{index}(W) = ((0, 1, 0, 0, \ldots, 2, 0, \ldots), 0)$.

Thus $\mathit{index}(W^*) \prec \mathit{index}(W)$. Hence, by the induction hypothesis, we have that $\mathcal{R}[W] = \mathcal{R}[W^*]$ is uniform in S_0.

Also from the induction hypothesis, we have that $\mathcal{D} \models (\forall) W^* \equiv \mathcal{R}[W^*]$. Furthermore, it is easy to prove the following entailment from the foundational axioms Σ:

$$\Sigma \models (\forall) W \equiv \alpha'_1 = \alpha_1 \wedge \cdots \wedge \alpha'_m = \alpha_m.$$

Therefore we have that $\mathcal{D} \models (\forall).W \equiv W^* \equiv \mathcal{R}[W^*] = \mathcal{R}[W]$ as wanted.

(b) Suppose that W is a $\sqsubset$-atom of the form

$$do\big([\alpha'_1, \ldots, \alpha'_m], S_0\big) \sqsubset do\big([\alpha_1, \ldots, \alpha_n], S_0\big).$$

When $m = 0$ and $n \geqslant 1$, $\mathcal{R}[W] = \mathit{true}$; and when $m \geqslant n$ $\mathcal{R}[W] = \mathit{false}$. In both of these cases the theorem follows immediately from $\Sigma \models (\forall) W \equiv \mathit{true}$ (*false*, respectively).

In the remaining case, when $1 \leqslant m < n$, we have that

$$\mathcal{R}[W] = \mathcal{R}\big[\alpha'_1 = \alpha_1 \wedge \cdots \wedge \alpha'_m = \alpha_m\big].$$

Let W^* stand for $\alpha'_1 = \alpha_1 \wedge \cdots \wedge \alpha'_m = \alpha_m$. The proof proceeds in a similar way as in the case of equality atoms above. By the same argument as in the equality case, W^* is regressable. The $\mathit{index}(W^*)$ differs from $\mathit{index}(W)$ in that λ_m and λ_n decrease by at least one in $\mathit{index}(W^*)$. Other λ values remain the same and the number of *Poss*-atoms, P, is zero in both cases. The number of $\sqsubset$-atoms, I, is also smaller by one in $\mathit{index}(W^*)$. The number of connectives, C, increases but, being the leftmost element, it has the lowest priority. Clearly, then, $\mathit{index}(W^*) \prec \mathit{index}(W)$. Furthermore, it is easy to prove the entailment:

$$\Sigma \models (\forall).do\big([\alpha'_1, \ldots, \alpha'_m], S_0\big) \sqsubset do\big([\alpha_1, \ldots, \alpha_n], S_0\big) \equiv W^*.$$

As in the previous case, we have by induction that $\mathcal{D} \models (\forall).W^* \equiv \mathcal{R}[W^*] = \mathcal{R}[W]$, and finally the theorem follows from this and the above entailment.

(c) Suppose W is a regressable atom of the form $\mathit{Poss}(\alpha, \sigma)$ for terms α and σ of sorts *action* and *situation* respectively. Then α must be of the form $A(\vec{t})$ and there must be a corresponding action precondition axiom $\mathit{Poss}(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$ in $\mathcal{D}$. After suitably renaming variables if necessary, we have that

$$\mathcal{D} \models (\forall).W \equiv \Pi_A(\vec{t}, \sigma). \tag{7}$$

By definition, $\Pi_A(\vec{x}, s)$ does not mention predicate *Poss*. Therefore, $\mathit{index}(\Pi_A(\vec{t}, \sigma)) \prec \mathit{index}(W)$. (Recall that in the definition of $\mathit{index}(W)$, the number P of *Poss*-atoms is the rightmost element, and the ordering priority increases from left to right.)

Moreover, since $\mathit{Poss}(A(\vec{t}), \sigma)$ is a regressable atom, σ and all terms of sort *situation* mentioned by $\vec{t}$ must be rooted at S_0. From this and the fact that $\Pi_A(\vec{x}, s)$ is by definition bounded by a situation term rooted at s, it follows that $\Pi_A(\vec{t}, \sigma)$ is regressable. Therefore, by induction, $\mathcal{R}[\Pi_A(\vec{t}, \sigma)]$ is uniform in S_0 and

$$\mathcal{D} \models (\forall).\Pi_A(\vec{t}, \sigma) \equiv \mathcal{R}\big[\Pi_A(\vec{t}, \sigma)\big]. \tag{8}$$

By definition, $\mathcal{R}[W] = \mathcal{R}[\Pi_A(\vec{t}, \sigma)]$. The theorem follows from this, (7) and (8).

(d) The remaining cases when W is an atom are the following:

(i) Suppose W is an atom that does not mention any terms of sort *situation* or whose only term of this sort is S_0. Then W is uniform in S_0. Hence $\mathcal{R}[W] = W$ by definition of regression and the theorem is immediate. Note that for such a W, $index(W) = ((0, \ldots), 0)$.

(ii) Suppose W is an atom that mentions a functional fluent term. Since W is regressable, it has no free variables of sort *situation* and hence, by Remark 1, it mentions a prime functional fluent term $f(\vec{t}, do(\alpha, \sigma))$. Let the successor state axiom corresponding to this fluent be:

$$f\big(\vec{x}, do(a, s)\big) = y \equiv \phi_f(\vec{x}, y, a, s).$$

Assuming suitable variable renaming, it is easy to see that

$$\mathcal{D} \models (\forall).W \equiv (\exists y).\phi_f(\vec{t}, y, \alpha, \sigma) \wedge W\big|_y^{f(\vec{t}, do(\alpha, \sigma))}. \tag{9}$$

Let W^* denote the rhs of the above equivalence. Let us show that W^* is regressable and that $index(W^*) \prec index(W)$. First, since W is an atom and it is bounded by a situation term rooted at S_0, then σ must be rooted at S_0. Since, by definition of successor state axioms, $\phi_f(\vec{x}, y, a, s)$ is bounded by s, W^* is bounded by a situation term rooted at S_0 and hence is regressable.

Second, $\phi_f(\vec{x}, y, a, s)$ does not mention the predicate *Poss*, so the number of *Poss*-atoms P is zero for both W and W^*. Further, since $f(\vec{t}, do(\alpha, \sigma))$ is a prime functional fluent term (see Definition 14), the only term of sort *situation* mentioned by $\vec{t}, \alpha$ is S_0. Thus replacing $f(\vec{t}, do(\alpha, \sigma))$ in W does not change any of the λ values except for $\lambda_{k+1+L(W)}$, where $k+1$ is the length of $do(\alpha, \sigma)$, which decreases. By definition of successor state axioms, $\phi_f(\vec{x}, y, a, s)$ is strictly bounded by s and does not mention the constant S_0. Then all terms of sort *situation* mentioned by $\phi_f(\vec{t}, y, \alpha, \sigma)$ which are different from σ are rooted at a situation variable. This means that the conjunct $(\exists y).\phi_f(\vec{t}, y, \alpha, \sigma)$ contributes only to one λ value, namely to $\lambda_{k+L(W)}$, which has lower ordering priority than $\lambda_{k+1+L(W)}$. Therefore, $index(W^*) \prec index(W)$.

As an example, consider the atom $f(do(B, do(A, S_0))) = c$ and a successor state axiom

$$f\big(do(a, s)\big) = y \equiv \big(\exists s'\colon s' \sqsubset s\big) P\big(y, s'\big).$$

In the case of this W,

$$index(W) = \big((0, 0, 0, 0, \ldots, \lambda_1 = 0, \lambda_2 = 1, 0, \ldots), 0\big)$$

and for $W^* = (\exists s'\colon s' \sqsubset do(A, S_0))P(y, s')$ we have

$$index\big(W^*\big) = \big((1, 0, 1, 0, \ldots, \lambda_1 = 1, \lambda_2 = 0, 0, \ldots), 0\big).$$

Thus $index(W^*) \prec index(W)$.

By induction, then, $\mathcal{R}[W^*]$ is uniform in S_0 and $\mathcal{D} \models (\forall).W^* \equiv \mathcal{R}[W^*]$. The theorem follows from this, $\mathcal{R}[W] = \mathcal{R}[W^*]$, and (9).

(iii) Suppose otherwise that W is a relational fluent atom of the form $F(\vec{t}, do(\alpha, \sigma))$. Let the successor state axiom corresponding to this fluent be:

$$F\big(\vec{x}, do(a, s)\big) \equiv \Phi_F(\vec{x}, a, s).$$

Assuming suitable variable renaming as before, we have that

$$\mathcal{D} \models (\forall).W \equiv \Phi_F(\vec{t}, \alpha, \sigma). \tag{10}$$

Let W^* denote the rhs of the above equivalence. We need to show that W^* is regressable and that $index(W^*) \prec index(W)$.

Since $F(\vec{t}, do(\alpha, \sigma))$ is regressable, σ and all the terms of sort *situation* mentioned by $\vec{t}$ and α must be rooted at S_0. Since, by definition, $\Phi_F(\vec{x}, a, s)$ is bounded by s, $\Phi_F(\vec{t}, \alpha, \sigma)$ is bounded by σ, which is rooted at S_0, and therefore regressable.

Now, by definition, $\Phi_F(\vec{x}, a, s)$ does not mention the predicate *Poss* nor constant S_0. Since $F(\vec{t}, do(\alpha, \sigma))$ is a regressable atom, all subterms of sort *situation* of $\vec{t}, \alpha$ and σ are uniformly rooted at S_0. Moreover, since $\Phi_F(\vec{x}, a, s)$ is strictly bounded by s, it follows that all terms of sort *situation* mentioned by $\Phi_F(\vec{t}, \alpha, \sigma)$ either appear in $\vec{t}, \alpha$ and σ or are rooted at a situation variable. Therefore, $index(W^*) \prec index(W)$.

As an example, consider the atom

$$F\big(g\big(do(C, S_0)\big), do\big(B, do(A, S_0)\big)\big)$$

and the successor state axiom $F(x, do(a, s)) \equiv P(x, s)$.

In the case of this W,

$$index(W) = \big((0, 0, 0, \lambda_1 = 1, \lambda_2 = 1, 0, \ldots), 0\big)$$

and for $W^* = P(g(do(C, S_0)), do(A, S_0))$ we have

$$index(W^*) = \big((0, 0, 0, \lambda_1 = 2, \lambda_2 = 0, 0, \ldots), 0\big).$$

By induction, $\mathcal{R}[W^*]$ is uniform in S_0 and $\mathcal{D} \models (\forall).W^* \equiv \mathcal{R}[W^*]$. The theorem follows from this, $\mathcal{R}[W] = \mathcal{R}[W^*]$, and (10).

(2) When W is a regressable non-atomic formula, we have the following possible cases:

(a) Suppose W is a regressable formula of the form

$$\big(\exists s\colon \sigma' \sqsubset do\big(\big[\alpha'_1, \ldots, \alpha'_m\big], s\big) \sqsubset do\big([\alpha_1, \ldots, \alpha_n], S_0\big)\big)W'.$$

If $m \geqslant n$, then $\mathcal{R}[W] = false$. Then the theorem follows immediately from $\mathcal{D} \models W \equiv false$.
If $m < n$, the it is easy to prove that

$$\mathcal{D} \models (\forall).W \equiv W_1 \vee W_2, \tag{11}$$

where

$$W_1 \stackrel{\text{def}}{=} \big(\exists s\colon \sigma' \sqsubset do\big(\big[\alpha'_1, \ldots, \alpha'_m\big], s\big) = do\big([\alpha_1, \ldots, \alpha_{n-1}], S_0\big)\big)W', \tag{12}$$

$$W_2 \stackrel{\text{def}}{=} \big(\exists s\colon \sigma' \sqsubset do\big(\big[\alpha'_1, \ldots, \alpha'_m\big], s\big) \sqsubset do\big([\alpha_1, \ldots, \alpha_{n-1}], S_0\big)\big)W'. \tag{13}$$

Clearly, both W_1 and W_2 are regressable. It is also clear that $L(W) = L(W_1) = L(W_2)$. Since $\lambda_{n+L(W)}$ is smaller in $index(W_1)$ and $index(W_2)$ than in $index(W)$ and the values λ_i, $i > n + L(W)$, are identical as is the number of *Poss* atoms, we have that $index(W_1) \prec index(W)$ and similarly $index(W_2) \prec index(W)$.
By the induction hypothesis $\mathcal{R}[W_1]$ and $\mathcal{R}[W_2]$ are both uniform in S_0, $\mathcal{D} \models (\forall).\mathcal{R}[W_1] \equiv W_1$, and $\mathcal{D} \models (\forall).\mathcal{R}[W_2] \equiv W_2$. The theorem follows from this, $\mathcal{R}[W] = \mathcal{R}[W_1] \vee \mathcal{R}[W_2]$ and (11).

(b) Suppose W is a regressable formula of the form

$$\big(\exists s\colon \sigma' \sqsubset s = do\big([\alpha_1, \ldots, \alpha_n], S_0\big)\big)W'.$$

It is easy to see that, after suitable variable renaming,

$$\mathcal{D} \models (\forall).W \equiv \sigma' \sqsubset do\big([\alpha_1, \ldots, \alpha_n], S_0\big) \wedge W'\big|^{s}_{do([\alpha_1, \ldots, \alpha_n], S_0)}. \tag{14}$$

Let W^* stand for the rhs of the above equivalence. By definition of regression, $\mathcal{R}[W] = \mathcal{R}[W^*]$. As we show in item (2) of Definition 15, σ' must be rooted at S_0. Moreover, since W' is a propositional combination of formulas each bounded by a situation term rooted at S_0 or at s, then $W'|^{s}_{do([\alpha_1, \ldots, \alpha_n], S_0)}$ is bounded by a situation term rooted at S_0. Therefore, W^* is regressable.
Let us show that $index(W^*) \prec index(W)$. Clearly W^* contains the same number P of *Poss* atoms as W. Let us show that the λ values are the same for both formulas. Consider a maximal term $do(\vec{\alpha}_1, s)$ from W' and let m be its length. Let $do(\vec{\alpha}_2, S_0)$ stand for $do([\alpha_1, \ldots, \alpha_n], S_0)$ and W'' for $W'|^{do(\alpha_1, s)}_{do(\alpha_1, do(\alpha_2, S_0))}$. Note that $n + L(W)$ is the index of the value λ that accounts for term $do(\vec{\alpha}_2, S_0)$ in $index(W)$ and $n + m + L(W'')$ the index of the value λ that accounts for $do(\vec{\alpha}_1, do(\vec{\alpha}_2, S_0))$. Also, since $L(W'') = L(W) - m$, $n + L(W) = n + m + L(W'')$. This implies that after the substitution that results in W'' the λ's are the same. Since this is true after the substitution of any term rooted at s, $index(W)$ and $index(W^*)$ have the same λ's.
From the above argument it follows that $index(W^*)$ differs from $index(W)$ only in the number of equality atoms, E, which is smaller in $index(W^*)$. Therefore, $index(W^*) \prec index(W)$.
Then, by induction, $\mathcal{R}[W^*]$ is uniform in S_0 and $\mathcal{D} \models (\forall).W^* \equiv \mathcal{R}[W^*]$. The theorem follows from this, $\mathcal{R}[W] = \mathcal{R}[W^*]$, and (14).

(c) Suppose W is a regressable formula of the form

$$\big(\exists s\colon \sigma' \sqsubset do\big(\big[\alpha'_1, \ldots, \alpha'_m\big], s\big) = do\big([\alpha_1, \ldots, \alpha_n], S_0\big)\big)W',$$

where $m > 1$ (otherwise we are in the previous case).
If $m > n$, $\mathcal{R}[W] = false$ by definition and, clearly, $\mathcal{D} \models W \equiv false$. The theorem then is immediate.
For $m \leqslant n$, it is easy to prove that

$$\begin{aligned}\mathcal{D} \models (\forall).W \equiv \big(\exists s\colon\ & do\big(\big[\alpha'_1, \ldots, \alpha'_{m-1}\big], s\big) = do\big([\alpha_1, \ldots, \alpha_{n-1}], S_0\big)\big)\\ & \big(\alpha''_m = \alpha_n\big) \wedge \sigma' \sqsubset do\big([\alpha_1, \ldots, \alpha_n], S_0\big) \wedge W',\end{aligned} \tag{15}$$

where α''_m stands for $\alpha'_m|^{s}_{do([\alpha_1, \ldots, \alpha_{n-m}], S_0)}$.
Let W^* denote this formula. By definition of regression, $\mathcal{R}[W] = \mathcal{R}[W^*]$.
Let us show that W^* is regressable. For the same reason as in the previous case, σ' is rooted at S_0, and thus the conjunct $\sigma' \sqsubset do([\alpha_1, \ldots, \alpha_n], S_0)$ is uniformly rooted at S_0. Since W is bounded by a situation term rooted at S_0, $do([\alpha'_1, \ldots, \alpha'_m], s)$ must be rooted at s. This means that all the terms of sort situation mentioned by the α''_m are

rooted at S_0 and therefore that the conjunct $\alpha''_m = \alpha_n$ is also uniformly rooted at S_0. Finally, W' is a propositional combination of formulas each bounded by a situation term rooted at S_0 or at s. Hence formula (15) is bounded by a situation term rooted at S_0 and therefore regressable.

Consider next the index of W^*. Clearly, the number of *Poss* atoms is the same in W and W^*. Based on $L(W^*)$, there are two possible cases:

- Suppose that all the terms of sort *situation* mentioned by α'_m are of length smaller than m. Then, by definition of $L(W)$, the lengths of the terms in α'_m are not included in $L(W)$. This implies that $L(W^*) = L(W)$. Let σ be any such term in α'_m and let $m-k$, $m \geqslant k > 0$, be its length. Then the corresponding term $\sigma|^{S}_{do([\alpha_1,\ldots,\alpha_{n-m}],S_0)}$ in α''_m has length $n-m+m-k = n-k$.
  We thus have that $n+L(W) = n+L(W^*) > n-k+L(W) = n-k+L(W^*)$ and that $\lambda_{n+L(W)}$ is smaller in $index(W^*)$ (by one) than in $index(W)$. All λ_i, $i > n+L(W)$, are equal for either formula. Therefore, $index(W^*) \prec index(W)$.
- Suppose that α'_m mentions terms of sort *situation* of length at least m. For simplicity, assume there is only one such term, σ, and let its length be $m+k$, $k \geqslant 0$. Then, by definition of $L(W)$, the length of σ was included in $L(W)$. Hence $L(W^*) = L(W) - m - k$. The corresponding term $\sigma|^{S}_{do([\alpha_1,\ldots,\alpha_{n-m}],S_0)}$ in α''_m has length $n-m+m+k = n+k$. We thus have that $n+k+L(W^*) = n+k+L(W)-m-k = n-m+L(W)$. Then, $\lambda_{n-m+L(W)}$ is larger in $index(W^*)$ but $\lambda_{n+L(W)}$ is smaller. Since $n > n-m$ and $L(W) > L(W^*)$ all λ_i, $i > n+L(W)$ are equal for either formula. It follows that $index(W^*) \prec index(W)$.

Therefore, by induction, $\mathcal{R}[W^*]$ is uniform in S_0 and $\mathcal{D} \models (\forall).W^* \equiv \mathcal{R}[W^*]$. The theorem follows from this, $\mathcal{R}[W] = \mathcal{R}[W^*]$, and (15).

(3) The cases when W is a regressable formula of the forms $\neg W_1$, $W_1 \wedge W_2$ and $(\exists v)W_1$ are straightforward. □

Soundness and completeness of the regression operator $\mathcal{R}$ follows from Theorems 2 and 3:

Theorem 4. *Suppose W is a regressable sentence of $\mathcal{L}_{sitcalc}$ and $\mathcal{D}$ is a basic non-Markovian theory of actions*:

$$\mathcal{D} \models W \quad \textit{iff} \quad \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \mathcal{R}[W].$$

As is the case with Markovian theories, this theorem is computationally very important. It tells us that, for the class of regressable formulas, we can prove that a Basic Action Theory entails a formula by first applying regression, a purely syntactic procedure, and then proving that the initial database plus unique names axioms entail the regressed formula. In particular, it tells us that for the second part, proving entailment of the regressed formula, the only second order axiom in the Basic Action Theory, i.e. the induction axiom, is no longer relevant.

5. Implementation

In this section we present an implementation in Prolog of a bounded formula evaluator. The evaluator is of course based on regression, but instead of computing a regressed formula uniform in S_0, it actually evaluates it against a background action theory and initial (Prolog) database and returns the answer.

Bounded formulas are encoded as Prolog terms of the following forms:

```
somes(sit-var, lower-bnd, upper-bnd, W)
alls(sit-var, lower-bnd, upper-bnd, W)
```

corresponding to formulas of the form $(\exists s_1{:}\ s' \sim s_1 \sim s)W$ and $(\forall s_1{:}\ s' \sim s_1 \sim s)W$, respectively. The first argument `sit-var` is the quantified variable of sort *situation*, s_1. The arguments `lower-bnd` and `upper-bnd` correspond to $s' \sim s_1$ and $s_1 \sim s$, respectively, where $\sim$ stands for either of $=$, $\sqsubset$, or $\sqsubseteq$. The third argument is the subformula W.

Bounded Formula Regression Evaluator

```
%%%%% (Case 1) %%%%%

%% Equality between situations and <-atoms

regr(s0 =$= s0).
regr(do(A,S) =$= do(A,S)).
regr(S1 << do(_,S)):- regr(S1 =$= S) ; regr(S1 << S).
regr(S1 <<= S):- regr(S1 =$= S) ; regr(S1 << S).

%% Poss, fluent, and situation independent atoms

regr(W):- isAtomEx(W), W.

%% case Sv is already a Prolog variable
```

```
regr(somes(Sv,S1 =$= Sv, Sv =$= S2,W)):-
        var(Sv), !, Sv=S2,
        regr(S1 =$= S2 & W).

regr(somes(Sv,S1 << Sv, Sv =$= S2,W)):-
        var(Sv), !, Sv=S2,
        regr(S1 << S2 & W).

regr(somes(Sv,S1 <<= Sv, Sv =$= S2,W)):-
        var(Sv), !, Sv=S2,
        regr(S1 <<= S2 & W).

%%%%% (Case 2) %%%%%
% (2a)
regr(somes(Sv,LBound,Sm <<= Sn,W)):- !,
        (regr(somes(Sv,LBound, Sm =$= Sn,W)) -> true ;
                   regr(somes(Sv,LBound, Sm << Sn,W))).

regr(somes(Sv,LBound,Sm << do(_,Sn),W)):- !,
        (regr(somes(Sv,LBound, Sm =$= Sn,W)) -> true ;
                  regr(somes(Sv,LBound, Sm << Sn,W)) ).

%% (2b)
regr(somes(Sv, S1 =$= do(A,S), do(A,S) =$= do(A,S2),W)):- !,
        regr(somes(Sv,s0 <<= S, S =$= S2, (S1 =$= do(A,S2)) & W)).

regr(somes(Sv,S1 << do(A,S), do(A,S) =$= do(A,S2),W)):- !,
        regr(somes(Sv,s0 <<= S, S =$= S2, (S1 << do(A,S2)) & W)).

regr(somes(Sv,S1 <<= do(A,S), do(A,S) =$= do(A,S2),W)):- !,
        regr(somes(Sv,s0 <<= S, S =$= S2, (S1 <<= do(A,S2)) & W)).

%% (2c)
% Sv is a constant representing a sitcalc variable
regr(somes(Sv,S1 =$= Sv, Sv =$= S2,W)):-
        sub(Sv,SNew,W,W1) ,
        SNew = S2,
        regr(S1 =$= S2 & W1).

regr(somes(Sv,S1 << Sv, Sv =$= S2,W)):-
        sub(Sv,SNew,W,W1),
        SNew = S2,
        regr(S1 << S2 & W1).

regr(somes(Sv,S1 <<= Sv, Sv =$= S2,W)):-
        sub(Sv,SNew,W,W1),
        SNew = S2,
        regr(S1 <<= S2 & W1).

%% (Case 3)
regr(-(S1 =$= S)):- not regr(S1 =$= S).
regr(-(S1 << S)):- not regr(S1 << S).

regr(- somes(Sv,LBound, UBound,W)):- not regr(somes(Sv,LBound,UBound,W)).
regr(alls(Sv,LBound, UBound, W)):- regr(-somes(Sv,LBound,UBound,-W)).
regr(-alls(Sv,LBound, UBound, W)):- regr(somes(Sv,LBound,UBound,-W)).

regr(P & Q):- regr(P), regr(Q).
regr(P v Q):- regr(P) -> true ; regr(Q).
regr(P => Q):- regr(P) -> regr(Q) ; true.
regr(P <=> Q):- regr((P => Q) & (Q => P)).
regr(-(-P)):- regr(P).
regr(-(P & Q)):- regr(-P v -Q).
regr(-(P v Q)):- regr(-P & -Q).
regr(-(P => Q)):- regr(-(-P v Q)).
regr(-(P <=> Q)):- regr(-((P => Q) & (Q => P))).
regr(-all(V,W)):- regr(some(V,-W)).
regr(-some(V,W)):- not regr(some(V,W)).
regr(-W):- isAtomEx(W), not regr(W).
regr(all(V,W)):- regr(-some(V,-W)).
regr(some(V,W)):- sub(V,_,W,W1), regr(W1).

%% extended atom definition
isAtomEx(A) :- not (A = -W ; A = (W1 & W2) ; A = (W1 => W2) ;
        A = (W1 <=> W2) ; A = (W1 v W2) ; A = some(X,W) ; A = all(X,W) ;
        A = (S1 <<= S2) ; A = (S1 << S2) ; A = (S1 =$= S2) ;
        A = somes(_,_,_,_) ; A = alls(_,_,_,_) ).
```

We use the following simple blocks world implementation.

Blocks World Basic Action Theory

```
poss(pickup(X),S):- clear(X,S), ontable(X,S), handempty(S).
poss(putdown(X),S):- holding(X,S).
poss(stack(X,Y),S):- holding(X,S), clear(Y,S).
poss(unstack(X,Y),S):- handempty(S), clear(X,S), on(X,Y,S).

clear(X,do(A,S)):- A=putdown(X) ; A=stack(X,Y) ; A=unstack(Y,X) ;
                   clear(X,S), not A=pickup(X),
                   not ( A=stack(Y,X) ; A=unstack(X,Y) ).

on(X,Y,do(A,S)):-  A=stack(X,Y) ;
                   on(X,Y,S), not A=unstack(X,Y).

ontable(X,do(A,S)):- A=putdown(X) ;
                     ontable(X,S) , not A=pickup(X).

handempty(do(A,S)):- A=putdown(X) ;
                     A=stack(X,Y) ;
                     handempty(S), not A=pickup(X), not A=unstack(X,Y).

holding(X,do(A,S)):- A=pickup(X) ; A=unstack(X,Y) ;
                     holding(X,S), not A=putdown(X), not A=stack(X,Y).

        % Primitive actions
primitive_action(pickup(X)).  primitive_action(putdown(X)).
primitive_action(stack(X,Y)). primitive_action(unstack(X,Y)).

        % Restore atoms.
restoreSitArg(clear(X),S,clear(X,S)). restoreSitArg(on(X,Y),S,on(X,Y,S)).
restoreSitArg(ontable(X),S,ontable(X,S)). restoreSitArg(handempty,S,handempty(S)).
restoreSitArg(holding(X),S,holding(X,S)).

        % Initial Situation
on(d,b,s0). on(a,c,s0).
clear(d,s0). clear(a,s0).
ontable(b,s0). ontable(c,s0).
handempty(s0).
block(a). block(b). block(c). block(d).
```

Consider the following query:

$$\big(\exists s_1\colon\ S_0 \sqsubset s_1 \sqsubseteq do\big(\big[unstack(D,B), stack(D,A)\big], S_0\big)\big)$$
$$(\exists x, y)(\forall a_2)\big(\forall s_2\colon\ S_0 \sqsubset do(a_2, s_2) \sqsubseteq$$
$$do\big(\big[putdown(y), unstack(A,x), stack(A,B)\big], s_1\big)\big)$$
$$Poss(a_2, s_2)$$

This query says: is there any situation s_1 in history

$$do\big(\big[unstack(D,B), stack(D,A)\big], S_0\big)$$

after S_0, such that the sequence

$$do\big(\big[putdown(y), unstack(A,x), stack(A,B)\big], s_1\big)$$

is "legal" (i.e. all actions in it are executable) for some blocks x, y?

Intuitively, the only chance for executing this sequence is after $unstack(D,B)$ but before $stack(D,A)$. That means block y must be D and x must be C. Any other sequence is impossible. We obtain the following results with the implementation.

A sample query

```
[eclipse 2]: regr(somes(s1,s0<<s1,s1<<=do(stack(d,a),do(unstack(d,b),s0)),
          some(x, some(y,
            all(act2,
            alls(s2,s0<<do(act2,s2),
                    do(act2,s2)<<=do(stack(a,b),do(unstack(a,x),do(putdown(y),s1))),
                                  poss(act2,s2) ))))).

Yes (0.00s cpu)
```

```
%%% Same query but requiring 's1' to be after 'do(unstack(d,b),s0)'
[eclipse 3]: regr(somes(s1,do(unstack(d,b),s0)<<s1,s1<<=do(stack(d,a),do(unstack(d,b),s0)),
          some(x, some(y,
            all(act2,
            alls(s2,s0<<do(act2,s2),
                    do(act2,s2)<<=do(stack(a,b),do(unstack(a,x),do(putdown(y),s1))),
                                poss(act2,s2) ))))).

No (0.00s cpu)

%%% Using variables and predicate block(_) to get actual blocks
[eclipse 5]: regr(somes(S1,s0<<S1,S1<<=do(stack(d,a),do(unstack(d,b),s0)),
          some(X, some(Y, block(X) & block(Y) &
            all(act2,
            alls(s2,s0<<do(act2,s2),
                    do(act2,s2)<<=do(stack(a,b),do(unstack(a,X),do(putdown(Y),S1))),
                                poss(act2,s2) ))))).

X = c
Y = d
S1 = do(unstack(d,b),s0)
Yes (0.01s cpu)
```

As an illustration of an action theory with non-Markovian features, consider modifying the action precondition axiom of $stack(x, y)$ to include the condition that $on(x, y)$ must not have held in the past unless it holds since the initial situation. This condition is a form of planning search control that prevents repeating stack actions:

$$Poss\big(stack(x, y), s\big) \equiv holding(x, s) \wedge clear(y, s) \wedge \\ \neg(\exists s_1\colon s_1 \sqsubset s)\big(on(x, y, s_1) \wedge (\exists s_2\colon s_2 \sqsubset s_1)\neg on(x, y, s_2)\big).$$

The corresponding, modified Prolog rule would be

```
poss(stack(X,Y),S):- holding(X,S), clear(Y,S),
    regr( -somes(s1, s0<<=s1, s1<<=S, on(X,Y,s1) &
                  somes(s2, s0<<=s2, s2<<s1, -on(X,Y,s2)))).
```

Here we use evaluator `regr` to handle the bounded formula. In general, a better option would perhaps be to reify all the fluent formulas and refer to them through the *holds* predicate. We refrain from doing that here though.

The following are the results obtained with two simple queries and the blocks world theory with the modified precondition axiom:

Simple query with Non-Markovian APA

```
%% This is possible because on(a,c) holds only in s0.
[eclipse 2]: poss(stack(a,c),do(unstack(a,c),s0)).

Yes (0.00s cpu)

[eclipse 3]: poss(stack(a,d),do(unstack(a,d),do(stack(a,d),do(unstack(a,c),s0)))).

No (0.00s cpu)
```

6. Conclusion

We have presented a generalization of Reiter's Basic Action Theories [5,21] where the Markov property is not assumed, hence allowing representing and reasoning with non-Markovian systems. The main challenge in doing this is to develop a framework that still supports regression as a computational device. We have identified a syntactically restricted class of formulas, which we have called *bounded*, to which regression can be applied and that form the basis of our generalization of Basic Action Theories for non-Markovian control. We have then generalized Reiter's regression operator, $\mathcal{R}$, to handle the extended class of regressable formulas. The modified operator, as the original, can be used to compute entailment of regressable formulas with respect to a background action theory. This problem intuitively consists in reducing the task of proving the entailment of the original formula into the task of proving entailment of the regressed formula with respect to the initial first-order database plus the unique names axioms for actions. This is especially important because it makes the only second-order axiom in these theories irrelevant with respect to solving this problem. Finally, we have proved that the generalized regression operator is correct and shown a simple Prolog implementation of a formula evaluator based on this operator.

Removing the Markov assumption from Basic Action Theories without any changes to the Situation Calculus ontology and language is possible thanks to the fact that histories are first class objects in these theories. Considerably more effort would have been required to remove this assumption from other formalizations where this is not the case.

Some recent work has considered non-Markovian features of domains described in $\mathcal{A}$ type languages [22]. In [23], Giunchiglia and Lifschitz are concerned with problems where actions have ramifications, i.e., indirect effects. In particular, they are concerned with ramifications due to fluents whose truth value depends on that of other fluents, but where this dependency is unknown or implicit. In this case, knowledge about past states may be useful to determine the value of fluents with an implicit dependency on others, so their framework is non-Markovian to some degree. Although their formalism does not allow one to explicitly write non-Markovian definitions for fluents, the problem they are concerned with also serves as a motivation for a formalism that can handle non-Markovian systems.

Mendez et al. [24] essentially consider the same problem we do here: reasoning about actions whose dynamics depends on past states. They extend the language $\mathcal{A}$ with the Past Linear Temporal logic connectives *previous* and *since*. This language is propositional and so there is no quantifying into temporal modalities. More recently, Gonzalez et al. [25] introduced an $\mathcal{A}$-like language for modeling non-Markovian domains. One of their motivations is the practical problem of modeling multimedia presentations that involve temporal conditions constraining how a presentation evolves. Similar to the language in [24], this language does not allow quantifying into temporal modalities since it is also propositional. Computation in this language is done through a translation into logic programming with answer set semantics [26] and systems such as Smodels [27] and DLV [28]. Answer sets of a domain description contain every state (set of literals) the system goes through for a given set of action occurrences, which may be disadvantageous in large domains. In Non-Markovian Action Theories, all the reasoning is done in terms of a sequence of actions and the initial database, hence the size of the domain does not have as big an impact.

An important question that we have not addressed in this paper is whether Non-Markovian Action Theories are more expressive than standard action theories. In [13] we introduced a procedure for compiling the class of non-Markovian theories that refer to past situations only through Past LTL-like formulas as in Example 2. In [29] we give a procedure for a much larger class of action theories, and conjecture that the transformation can be extended to the general case. In developing this transformation for a larger class of formulas, we gain the insight that a transformation for the general case would likely have to essentially produce formulas and axioms that "simulate" the steps of the generalized regression procedure. In this case, it seems more reasonable and practical simply to use the generalized regression operator on the Non-Markovian Action Theories instead of applying a transformation.

Acknowledgements

I am very grateful to Fahiem Bacchus, Gerhard Lakemayer, Yves Lespérance and Hector Levesque for lots of comments and suggestions that helped improve this paper. We also thank the anonymous reviewers for their extremely useful and detailed feedback.

References

[1] A. Gabaldon, Non-Markovian control in the Situation Calculus, in: Proc. of the 18th National Conference on Artificial Intelligence (AAAI'02), Edmonton, Canada, 2002, pp. 519–524.
[2] R. Reiter, Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems, MIT Press, Cambridge, MA, 2001.
[3] J. McCarthy, Situations, actions and causal laws, Tech. Rep., Stanford University, 1963; reprinted in: M. Minsky (Ed.), Semantic Information Processing, MIT Press, Cambridge, MA, 1968, pp. 410–417.
[4] J. McCarthy, P. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in: B. Meltzer, D. Michie (Eds.), Machine Intelligence, vol. 4, Edinburgh University Press, 1969, pp. 463–502, also appears in: N. Nilsson, B. Webber (Eds.), Readings in Artificial Intelligence, Morgan–Kaufmann, Edmonton, 1987, pp. 431–450.
[5] R. Reiter, The frame problem in the Situation Calculus: A simple solution (sometimes) and a completeness result for goal regression, in: V. Lifschitz (Ed.), Artificial Intelligence and Mathematical Theory of Computation, Academic Press, 1991, pp. 359–380.
[6] G. Saake, U.W. Lipeck, Foundations of temporal integrity monitoring, in: C. Rolland, F. Bodart, M. Leonard (Eds.), Proc. of the IFIP Working Conference on Temporal Aspects in Information Systems, North-Holland, Amsterdam, 1988, pp. 235–249.
[7] J. Chomicki, Efficient checking of temporal integrity constraints using bounded history encoding, ACM Transactions on Database Systems 20 (2) (1995) 148–186.
[8] I. Kiringa, Simulation of advanced transaction models using Golog, in: Proc. 8th Biennial Workshop on Data Bases and Programming Languages (DBPL'01), Rome, 2001.
[9] I. Kiringa, A. Gabaldon, Expressing transactions with savepoints as non-Markovian theories of actions, in: F. Bry, C. Lutz, U. Sattler, M. Schoop (Eds.), 10th International Workshop on Knowledge Representation Meets Databases, CEUR, Hamburg, Germany, 2003, http://ceur-ws.org/Vol-79.
[10] I. Kiringa, A. Gabaldon, Synthesizing advanced transaction models using the Situation Calculus, Jounal of Intelligent Information Systems (2009).
[11] F. Bacchus, F. Kabanza, Using temporal logics to express search control knowledge for planning, Artificial Intelligence 116 (2000) 123–191.
[12] J. Kvarnström, P. Doherty, TALplanner: A temporal logic based forward chaining planner, Annals of Mathematics and Artificial Intelligence 30 (2000) 119–169.
[13] A. Gabaldon, Compiling control knowledge into preconditions for planning in the Situation Calculus, in: Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI'03), Acapulco, Mexico, 2003.
[14] A. Gabaldon, Precondition control and the progression algorithm, in: Proc. of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR'04), Whistler, B.C., Canada, 2004.
[15] F. Bacchus, C. Boutilier, A. Grove, Rewarding behaviors, in: Proc. of 13th National Conference on Artificial Intelligence (AAAI-96), 1996, pp. 1160–1167.

[16] F. Bacchus, C. Boutilier, A. Grove, Structured solution methods for non-Markovian decision processes, in: Proc. 14th National Conference on Artificial Intelligence (AAAI-97), 1997, pp. 112–117.
[17] S. Thiébaux, F. Kabanza, J. Slaney, Anytime state-based solution methods for decision processes with non-Markovian rewards, in: Proc. of the 18th Conference on Uncertainty in Artificial Intelligence (UAI'02), Morgan Kaufmann, Edmonton, Canada, 2002, pp. 501–510.
[18] C. Gretton, D. Price, S. Thiébaux, Implementation and comparison of solution methods for decision processes with non-Markovian rewards, in: Proc. of the 19th Conference on Uncertainty in Artificial Intelligence (UAI'03), Morgan Kaufmann, Acapulco, Mexico, 2003.
[19] J. McCarthy, Elephant 2000: A programming language based on speech acts, available at http://www-formal.stanford.edu/jmc/, 1992.
[20] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, R.B. Scherl, Golog: A logic programming language for dynamic domains, Journal of Logic Programming 31 (1–3) (1997) 59–83.
[21] F. Pirri, R. Reiter, Some contributions to the metatheory of the Situation Calculus, Journal ACM 46 (3) (1999) 325–364.
[22] M. Gelfond, V. Lifschitz, Action languages, Electronic Transactions on Artificial Intelligence 3 (1998) 195–210, http://www.ep.liu.se/ea/cis/1998/016.
[23] E. Giunchiglia, V. Lifschitz, Dependent fluents, in: Proc. of IJCAI-95, 1995, pp. 1964–1969.
[24] G. Mendez, J. Lobo, J. Llopis, C. Baral, Temporal logic and reasoning about actions, in: Third Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense'96), 1996.
[25] G. Gonzalez, C. Baral, M. Gelfond, Alan: An action language for non-Markovian domains, in: Proc. of the NonMonotonic Reasoning, Action and Change Workshop (NRAC'03), Acapulco, Mexico, 2003.
[26] M. Gelfond, V. Lifschitz, Representing actions and change by logic programs, Journal of Logic Programming 17 (1993) 301–322.
[27] I. Niemelä, P. Simons, Smodels—An implementation of the stable models and well-founded semantics for normal logic programs, in: Proc. 4th International Conference on Logic Programming and Non-Monotonic Reasoning, 1997.
[28] S. Citrigno, T. Eiter, W. Faber, G. Gottlob, C. Koch, N. Leone, C. Mateis, G. Pfeifer, F. Scarcello, The dlv system: Model generator and application frontends, in: F. Bry, B. Freitag, D. Seipel (Eds.), Proc. of the 12th Workshop on Logic Programming (WLP'97), LMU München, 1997, pp. 128–137.
[29] A. Gabaldon, Non-Markovian control in dynamical systems and planning, PhD thesis, University of Toronto, Toronto, Canada, 2004.