Non-Markovian Control in the Situation Calculus
An Elaboration

Niklas Hoppe
Seminar Foundations Of Artificial Intelligence
Knowledge-Based Systems Group
RWTH Aachen
May 3, 2009
## Contents

1 Introduction ............................................................... 3

2 The Situation Calculus .................................................. 3
   2.1 The Language $\mathcal{L}$ ........................................... 3
      2.1.1 Example ..................................................... 4
   2.2 The Simple Formulas ............................................... 5
   2.3 The Axioms $W$ .................................................... 5
      2.3.1 Action Precondition Axioms ................................. 5
      2.3.2 Successor State Axioms .................................... 6
      2.3.3 Unique Names Axioms ...................................... 6
      2.3.4 Executability Axiom ....................................... 6
      2.3.5 Example ..................................................... 6
   2.4 Goal Regression .................................................... 7
      2.4.1 The Regression Operator .................................... 7
      2.4.2 Regression Theorem ....................................... 8
      2.4.3 Example ..................................................... 9

3 Non-Markovian Control .................................................. 9
   3.1 Some Definitions .................................................. 9
      3.1.1 Predecessor ............................................... 9
      3.1.2 Maximal terms ............................................ 10
      3.1.3 Length ................................................... 10
      3.1.4 Rootedness ............................................... 10
      3.1.5 Boundedness ............................................... 10
   3.2 Goal Regression .................................................... 11
      3.2.1 Extension .................................................. 12
      3.2.2 Example ..................................................... 13

4 Conclusion ............................................................... 14
1 Introduction

One of the most important formalisms of artificial intelligence is the situation calculus. Despite the expressive power of the situation calculus, it assumes what is known as the Markov property. The Markov property denotes the assumption, that whether an action is executable and what are its effects can be determined entirely by the current state of the world. However, there are situations where the Markov property is not present. For example, whether you can draft a beer from a barrel of nice lager depends on the amount of beer you draught in all previous states since the barrel was full. Assuming that only whole pints\(^1\) are draught and the barrel contained 10 litres of fine lager: To keep track of the barrels content you would have to introduce 20 fluents. One for each pint of lager you might want to drink.

So in this elaboration I will present Gabaldon’s approach [1] in extending the situation calculus for scenarios where the Markov property is not present without reducing the expressive power. It will be based on Reiter’s axiomatization of the situation calculus and his goal regression method [3].

Furthermore, this elaboration aims at conveying the intuition of the concepts. Intuitive explanations will be added where possible and proofs will be left out.

2 The Situation Calculus

This section describes the situation calculus using the definitions by Reiter [3].

2.1 The Language $\mathcal{L}$

In the situation calculus the world is described by a sorted first-order logic language $\mathcal{L}$. This language consists of the following axioms.

1. Function symbols of sort\(^2\) state: The constant $S_0$ and the binary function $do$, which takes arguments of sort action and state. $S_0$ denotes the initial state of the world and $do(a, s)$ denotes the resulting state after doing action $a$ in state $s$.

2. Function symbols of sort action. These are the actions which can be performed in the world.

3. Predicate symbols of sort fluent, which take any amount of arguments but exactly one of sort state. The fluents describe the state of the world.

4. Variables of each sort.

5. A distinguished predefined binary predicate symbol $Poss$ taking arguments of sort action and state. $Poss(a, s)$ states that action $a$ is possible in state $s$.

---

\(^1\)Assume a pint to be half a litre.

\(^2\)Symbols of a certain ’sort’ denote a distinct category or type of symbols.
6. A distinguished predefined unary predicate symbol \textit{ex} taking an argument of sort \textit{state}. \textit{ex}(s) denotes that \(s = do(a_1, do(a_2, \ldots do(a_n, S_0) \ldots ))\) for \(n \geq 0\) is an executable plan.

7. A distinguished predefined equality symbol = with the usual meaning.

8. Arbitrary function symbols with arbitrary arity, not of sort \textit{state} or \textit{action}, none of which take an argument of sort \textit{state}.

9. Arbitrary predicate symbols with arbitrary arity, none of which take an argument of sort \textit{state}.

10. Logical constants and punctuation as usual.

Note the following:

- A state \(s\) is a history of actions
  \[do(a_1, do(a_2, do(\ldots do(a_n, s) \ldots )))\]
  for which we write instead:
  \[do([a_1, a_2, \ldots, a_n], s)\].

- All predicate symbols can be divided into two groups: the domain and the non-domain predicate symbols. The domain predicates describe certain properties of a world state, e.g. the agent is holding an object or two boxes stand next to each other. The domain predicates are considered fluents. However, the non-domain predicates \textit{Poss} and \textit{ex} describe relations between world states and actions. Therefore, they are not considered fluents, although they are predicate symbols which take an argument of sort \textit{state}.

- We write \(\vec{x}\) as an abbreviation for \(x_1, \ldots, x_n\).

2.1.1 Example

Remember the example with the barrel of beer. For simplification assume that there is just a full pint of beer and a thirsty agent. This scenario could look like this:

- State \(S_0\) represents the described initial state: a full pint of beer \(p\) and a thirsty agent \(r\),
- therefore \(\text{full}(p, S_0) = \text{true}\).
- We introduce the fluent \textit{thirsty}(\(R, s\)) which is also true for the agent in the initial state: \textit{thirsty}(\(r, S_0\)) = \text{true}.
• Additionally the agent should have the opportunity to drink the pint: 
  \( drink(R, P) \).

• \( R \) is a variable that denotes an agent, \( P \) is a variable that denotes a pint
  and \( s \) is a variable of sort \textit{state}.

2.2 The Simple Formulas

Let \( s \) be a variable of sort \textit{state}. The set of formulas that are \textit{simple} with respect to \( s \) are:

1. \( F(\vec{t}, s) \) and \( F(\vec{t}, S_0) \) are simple with respect to \( s \) when \( F \) is a fluent and
   \( \vec{t} \) are terms. An equality atom mentioning no state variable at all, or
   mentioning only the state variable \( s \), is simple with respect to \( s \). Any
   other atom with predicate symbol other than \textit{Poss} or \textit{ex} is simple with
   respect to \( s \).

2. If the formulas \( S_1 \) and \( S_2 \) are simple with respect to \( s \), so are \( \neg S_1 \), \( S_1 \land S_2 \),
   \( S_1 \lor S_2 \), \( S_1 \rightarrow S_2 \), \( S_1 \equiv S_2 \).

3. If \( S \) is simple with respect to \( s \), so are \( \exists x : S \) and \( \forall x : S \) whenever \( x \) is a
   variable not of sort \textit{state}.

So the term \textit{simple} formula refers to a formula which only mentions predicate
symbols other than \textit{Poss} and \textit{ex}, which contains fluents that only refer to state
\( S_0 \) or state variable \( s \) and which contain at most one state variable \( s \).

2.3 The Axioms \( \mathcal{W} \)

Based on the first-order language \( \mathcal{L} \) we formulate an axiomatization of the world
\( \mathcal{W} \) consisting of:

2.3.1 Action Precondition Axioms

The action precondition axioms describe the conditions which must be fulfilled
 to perform a certain action. Thus, an action precondition axiom has the form:

\[ c \equiv Poss(a, s). \]

So action \( a \) is possible in state \( s \) if and only if the preconditions \( c \) hold. A
more detailed representation of this axioms is:

\[ \forall \vec{x}, s : \Pi_A \equiv Poss(A(x_1, ..., x_n), s), \]

where \( \Pi_A \) denotes a formula simple with respect to \( s \) containing the preconditions
to perform action \( A \). The free variables in \( \Pi_A \) are among \( x_1, ..., x_n, s \). \( A \)
is a \( n \)-ary function symbol of sort \textit{action}.
2.3.2 Successor State Axioms

The successor state axioms describe what impact a certain action has or has not on a certain fluent. The simple form of the successor state axioms is:

\[ \text{Poss}(a, s) \rightarrow F(\text{do}(a, s)) \equiv \gamma^+_F(a, s) \lor (F(s) \land \neg \gamma^-_F(a, s)), \]

where \( F \) is a fluent, \( a \) is an action, \( s \) is a state variable, \( \gamma^+_F(a, s) \) denotes the conditions such that performing action \( a \) in state \( s \) makes fluent \( F \) true and \( \gamma^-_F(a, s) \) denotes the conditions such that performing action \( a \) in state \( s \) makes fluent \( F \) false.

In plain words: Assuming that action \( a \) is possible in state \( s \), \( F \) will be true after performing action \( a \) when either \( a \) will make \( F \) true (\( \gamma^+_F(a, s) \)), or \( F \) was true before (\( F(s) \)) and \( a \) won’t change that (\( \neg \gamma^-_F(a, s) \)).

The more general but more detailed representation of a successor state axiom is:

\[ \forall a, s, x_1, \ldots, x_n : \text{Poss}(a, s) \rightarrow F(x_1, \ldots, x_n, \text{do}(a, s)) \equiv \gamma^+_F(x_1, \ldots, x_n, a, s) \lor [F(x_1, \ldots, x_n, s) \land \neg \gamma^-_F(x_1, \ldots, x_n, a, s)], \]

which introduces more free all-quantified variables \( x_1, \ldots, x_n \) which are not of sort action or state and without changing the axioms intuition.

2.3.3 Unique Names Axioms

The unique names axioms serve as a distinction between the unique states of the world.

\[ S_0 \neq \text{do}(a, s), \]
\[ \text{do}(a, s) = \text{do}(a', s') \rightarrow a = a' \land s = s'. \]

2.3.4 Executability Axiom

The executability axiom \( ex(s) \) is a predicate that is true if and only if \( s = \text{do}(\text{do}(a_1, \text{do}(a_2, \ldots \text{do}(a_n, S_0), \ldots))) \) is an executable action sequence. The predicate \( ex(s) \) is defined as follows:

\[ ex(s) \equiv s = S_0 \lor (\exists a, s' : s = \text{do}(a, s') \land \text{Poss}(a, s') \land ex(s')). \]

2.3.5 Example

Recall the last example with the thirsty agent and the full pint of beer. In this scenario action precondition axiom for \( \text{drink}(R, P) \) and successor state axioms for \( \text{thirsty}(R, s) \) and \( \text{full}(P, s) \) might look like this:

- The action precondition axiom for \( \text{drink}(R, P) \)
  \[ \forall R, P, s : \text{full}(P, s) \equiv \text{Poss}(\text{drink}(R, P), s) \]
  denotes that an agent \( R \) can only drink a pint \( P \) if this pint is full in state \( s \).
• Assuming the general form of the successor state axiom for a fluent as introduced before, we define the fluent preconditions as follows:

\[
\gamma^{+}_{\text{thirsty}(R,s)}(R,\text{drink}(R,P),s) = \text{false} \\
\gamma^{-}_{\text{thirsty}(R,s)}(R,\text{drink}(R,P),s) = \text{full}(P,s).
\]

So drinking will never make the agent thirsty and to appease the agent’s thirst by drinking the pint, the pint also has to be full.

\[
\gamma^{+}_{\text{full}(P,s)}(R,\text{drink}(R,P),s) = \text{false} \\
\gamma^{-}_{\text{full}(P,s)}(R,\text{drink}(R,P),s) = \text{true}
\]

We assume here that it’s not possible to refill the pint and that drinking a pint will always result in an empty pint.

2.4 Goal Regression

2.4.1 The Regression Operator

With the set of axioms \( \mathcal{W} \) we can now formulate goals, which we want to pursue with the actions at hand. Such a goal is represented by a formula \( G(s) \). The formula \( G(s) \) denotes certain properties of the world in state \( s \), such as some item has been broken or moved to a certain position. As mentioned before \( s \) is a sequence of actions \( \text{do}([a_1,\ldots,a_n],S_0) \) and thus a sequence which leads to the desired goal \( G(s) \) starting from the initial state \( S_0 \).

This leads to the following method for finding such a plan through reasoning:

\[
\mathcal{W} \models \exists s : G(s) \land \text{ex}(s)
\]

In plain words: We can derive from our world axioms (\( \mathcal{W} \)) that there is an action sequence \( (s) \) which leads to our goal \( (G(s)) \) and which can be executed \((\text{ex}(s))\).

To show this entailment, we will now define a regression operator \( \mathcal{R}_\Theta \) and then show that applying this operator is equivalent to the entailment. Let \( \Theta \subseteq \mathcal{L} \) contain one successor state axiom for each fluent and one action precondition axiom for each action of the language \( \mathcal{L} \). The regression operator \( \mathcal{R}_\Theta \) when applied to a formula of \( \mathcal{L} \) is defined recursively as follows:

1. When \( A \) is a non-fluent atom, including equality atoms, and atoms with predicate symbol \( \text{Poss} \) or \( \text{ex} \), or when \( A \) is a fluent atom whose state argument is a state variable, or the state constant \( S_0 \): \( \mathcal{R}_\Theta [A] = A \).

2. When \( F \) is a fluent whose successor state axiom in \( \Theta \) is

\[
\forall a, s, x_1,\ldots,x_n : \text{Poss}(a,s) \rightarrow \\
F(x_1,\ldots,F_n,\text{do}(a,s)) \equiv \gamma^{+}_F(a,s) \lor (F(s) \land \neg \gamma^{-}_F(a,s))
\]
then

\[ R_\theta [F(t_1, \ldots, t_n, do(\alpha, \sigma))] = R_\theta [\gamma^+_\varphi(a, s) \lor (F(s) \land \neg \gamma^-_\varphi(a, s))] \]

3. Whenever \( f \) is a formula:

\[
\begin{align*}
R_\theta [\neg f] & = \neg R_\theta [f] \\
R_\theta [\forall v : f] & = \forall v : R_\theta [f] \\
R_\theta [\exists v : f] & = \exists v : R_\theta [f]
\end{align*}
\]

4. Whenever \( f_1 \) and \( f_2 \) are formulas:

\[
\begin{align*}
R_\theta [f_1 \land f_2] & = R_\theta [f_1] \land R_\theta [f_2] \\
R_\theta [f_1 \lor f_2] & = R_\theta [f_1] \lor R_\theta [f_2] \\
R_\theta [f_1 \rightarrow f_2] & = R_\theta [f_1] \rightarrow R_\theta [f_2] \\
R_\theta [f_1 \equiv f_2] & = R_\theta [f_1] \equiv R_\theta [f_2]
\end{align*}
\]

The idea behind regression is to reason from the goal on state backwards to the initial state. In each state the preconditions for the proposed action must be fulfilled and the proposed actions must legally lead from state to state as mentioned in the action sequence.

### 2.4.2 Regression Theorem

Suppose now that \( W \subseteq L \) contains the executability axiom, unique names axioms for states, a successor state axiom for each fluent of \( L \), and an action precondition axiom for each action function of \( L \). Every state variable of all remaining axioms of \( W \) is explicitly or implicitly universally quantified. Furthermore these remaining axioms mention only predicate symbols other than \( Poss \) and \( ex \). Suppose that \( G(s) \in L \) is simple with respect to \( s \), and that the state variable \( s \) is the only free variable of \( G(s) \). The regression theorem states:

\[ W \models \forall s : G(s) \equiv R(G(s)). \]

In plain words: With our world axiomatization \( W \) the action sequence \( s \) leads to our goal \( G \) if and only if the decomposition of \( G(s) \), \( R(G(s)) \) is true. That \( R(G(s)) \) is true means that the action sequence \( s \) leads to the goal \( G \) and starts from the initial state \( S_0 \) where all preconditions hold for the execution of action sequence \( s \).

Goal regression is sound and complete. For a proof and further details see Reiter [3].
2.4.3 Example

A goal in our beer scenario might be to appease the agent’s thirst:

\[ G_{st}(s) = \neg \text{thirsty}(r,s) \]

where \( A \) is a variable of state action and \( s = do(A,s') \). Since this goal is simple, regressing it is easy:

\[
\neg( \mathcal{R} \{ \neg \text{thirsty}(r,do(A,s')) \} ) = \\
\neg( \mathcal{R}_{\Theta} \{ \neg \text{thirsty}(A,s') \} ) = \\
\neg( \mathcal{R}_{\Theta} \{ \neg \text{thirsty}(A,s') \} ) = \\
\neg( \mathcal{R}_{\Theta} \{ \neg \text{thirsty}(A,s') \} ) = \\
\neg( \mathcal{R}_{\Theta} \{ \neg \text{thirsty}(A,s') \} ) = \\
\neg( \mathcal{R}_{\Theta} \{ \neg \text{thirsty}(A,s') \} ) = \\
\neg( \mathcal{R}_{\Theta} \{ \neg \text{thirsty}(A,s') \} ) = \\
\neg( \mathcal{R}_{\Theta} \{ \neg \text{thirsty}(A,s') \} ) = \\
\neg( \mathcal{R}_{\Theta} \{ \neg \text{thirsty}(A,s') \} ) = \\
\neg( \mathcal{R}_{\Theta} \{ \neg \text{thirsty}(A,s') \} ) = \\
\neg( \text{false} ) = \text{true}.
\]

This method involves simple model checking to instantiate the free variables. Keeping track of the state variable \( s \) provides the plan to achieve the desired goal: \( \text{drink}(r,p) \).

3 Non-Markovian Control

3.1 Some Definitions

We now assume that the language \( L \) of the situation calculus is a second-order language. This enhancement is needed to be able to quantify over predicates as we will see later. The following definitions were introduced by Pirri and Gabaldon. See the corresponding papers [2] and [1] for details.

3.1.1 Predecessor

As mentioned before a state is a sequence of actions \( do([a_1,...,a_n],s) \) with \( n \geq 0 \) and each state is unique. Apparently these unique states are ordered: \( do([a_2,...,a_n],s) \) precedes \( do([a_1,...,a_n],s) \) and so on. We denote this ordering with the \( \prec \) symbol:

\[ do([a_2,...,a_n],s) \prec do([a_1,...,a_n],s) \]
This relation is formalized with the following axioms:

\[
\neg s \prec S_0, \\
s \prec do(a, s') \equiv s \prec s' \lor s = s',
\]

additionally indicating that there is no state before the initial state \(S_0\).

### 3.1.2 Maximal terms

With \(g(t_1, ..., t_n) \in \mathcal{L}\), \(t_1, ..., t_n\) are proper subterms of \(f(t_1, ..., t_n)\). An occurrence of a state term \(s\) in a formula \(f \in \mathcal{L}\) is maximal if and only if its occurrence is not a proper subterm of another state term.

### 3.1.3 Length

The length of a state (respectively an action sequence) \(do([a_1, ..., a_n], s)\) is defined as \(n\).

### 3.1.4 Rootedness

A state \(do([a_1, ..., a_n], s)\) is rooted at state \(s\) if and only if \(a_1, ..., a_n\) do not mention any other state variable than \(s\) or do not mention any state variable at all. A state is rooted at \(S_0\) if and only if \(a_1, ..., a_n\) do not mention any state variable.

A \(s\)-rooted state denotes a state which is independent of any state variable other than \(s\) or which is independent from state variables at all.

### 3.1.5 Boundedness

Let \(\sigma\) be a state \(do([a_1, ..., a_n], s)\) rooted at \(s\). The formulas of \(B \subset \mathcal{L}\) bounded by \(\sigma\) are a set of formulas with:

1. If \(t_1, t_2\) are terms of the same sort whose subterms of sort state (if any) are all rooted at \(s\), then \(t_1 = t_2\) is a formula bounded by \(\sigma\).
2. If \(\sigma'\) is a term of sort state rooted at some state variable or at \(S_0\), then \(\sigma' \prec \sigma\) is a formula bounded by \(\sigma\).
3. If the subterms of the arguments \(t_1, ..., t_n\) of a predicate \(P\), a fluent \(F\), or an action function \(A\) are all rooted at \(s\), then accordingly \(P(t_1, ..., t_n), F(t_1, ..., t_n, \sigma)\) and \(Poss(A(t_1, ..., t_n), \sigma)\) are formulas bounded by \(\sigma\).
4. If \(\sigma'\) is a term of sort state rooted at \(s'\) and \(f\) is a formula bounded by a possibly different term of sort state also rooted at \(s'\), then \(\sigma' \prec \sigma \land f\) and \(\sigma' = \sigma \land f\) are formulas bounded by \(\sigma\).
5. If \(f_1, f_2\) are formulas bounded by state terms rooted at \(s\), then \(\neg f_1, f_1 \land f_2, \text{ and } \exists v : f_1\), provided \(\sigma\) is not rooted at \(v\), are formulas bounded by \(\sigma\).
A formula $f$ bounded by $\sigma$ rooted by $s$ is strictly bounded by $\sigma$ if:

- the only term of sort state rooted at $s$, mentioned by $f$ and different to $\sigma$, is $s$.
- for every maximal term $\sigma'$ of sort state mentioned by $f$ that is different to $\sigma$, $f$ mentions an atom $\sigma' \prec \sigma''$ or $\sigma' = \sigma''$.
- $f$ does not mention the initial state $S_0$.

The intuition behind bounded and strictly bounded formulas is to extend the notion of rootedness from states to formulas. State terms in a formula bounded by a state $\sigma$, depend on this state $\sigma$. State terms in a formula strictly bounded by a state $\sigma$, are restricted to subhistories of $\sigma$.

### 3.2 Goal Regression

The idea of goal regression remains the same, but since the Markov property was removed and some restrictions introduced the regression operator has to be adapted to the new circumstances. Assume $R_{nm}$ to be the new regression operator and $f$ a formula of $\mathcal{L}$.

1. $f$ is first order.
2. $f$ is bounded by a term of sort state rooted at $S_0$.
3. For every atom of the form $\text{Poss}(\alpha, \sigma)$ mentioned by $f$, $\alpha$ has the form $A(t_1, \ldots, t_n)$ for some $n$-ary action function symbol $A$ of $\mathcal{L}$.

For some appearances of $f$ the regression operator $R_{nm}$ behaves just like $R_\Theta$. These are:

- an equality atom of the form $\text{do}([\alpha'_1, \ldots, \alpha'_m], S_0) = \text{do}([\alpha_1, \ldots, \alpha_n], S_0)$,
- an $\prec$-atom $\text{Poss}(A(\bar{t}), \sigma)$ where $A(\bar{t})$ and $\sigma$ are terms of sort action and state respectively,
- an atom whose only state term is $S_0$,
- an atom that mentions a functional fluent term of the form $g(\bar{t}, \text{do}(\alpha, \sigma))$ and is not of the form $\sigma' = \sigma''$ or $\sigma' \prec \sigma''$ where $\sigma'$ is rooted at a variable of sort state,
- a relational fluent atom $F(\bar{t}, \text{do}(\alpha, \sigma))$. 

11
3.2.1 Extension

For other forms of a formula $f$ the regression operator is recursively defined as follows:

1. Suppose $f$ is regressable and of the form:

   $$do([\alpha_1, ..., \alpha_m], s) \prec do([\alpha'_1, ..., \alpha'_n], S_0) \land f'$$

   where $f'$ may be empty.

   If $m \geq n$, then $R_{nm}[f] = false$.

   If $m < n$, then

   $$R_{nm}[f] = \neg \{ \neg R[do([\alpha_1, ..., \alpha_m], s)] = do([\alpha'_1, ..., \alpha'_{n-1}], S_0) \land f' \land

   \neg R[do([\alpha_1, ..., \alpha_m], s) \prec do([\alpha'_1, ..., \alpha'_{n-1}], S_0) \land f'] \}$$

   In plain words: $do([\alpha_1, ..., \alpha_m], s)$ is a proper subhistory of $do([\alpha'_1, ..., \alpha'_n], S_0)$ if it is shorter and if one of the states preceding $do([\alpha'_1, ..., \alpha'_n], S_0)$ is equal to the state $do([\alpha_1, ..., \alpha_m], s)$.

2. Suppose $f$ is regressable and of the form:

   $$do([\alpha_1, ..., \alpha_m], s) = do([\alpha'_1, ..., \alpha'_n], S_0) \land f'$$

   where $m \geq 1$ and $f'$ may be empty.

   If $m > n$, then $R_{nm}[f] = false$.

   If $m \leq n$, then

   $$R_{nm}[f] = R_{nm}[do([\alpha_1, ..., \alpha_n], S_0)] = do([\alpha'_1, ..., \alpha'_m], S_0) \land f']$$

   In plain words: Two states or action histories are equal, if all mentioned actions and states at the same position in the action history are equal. Note that $m > n$ would lead to some state preceding the initial state $S_0$ which is illegal.

3. Suppose $f$ is regressable and of the form:

   $$s = do([\alpha_1, ..., \alpha_n], S_0) \land s = do([\alpha'_1, ..., \alpha'_m], S_0) \land f'$$

   where $f'$ may be empty.

   If $n \neq m$, then $R_{nm}[f] = false$.

   If $n = m$, then

   $$R_{nm}[f] = R_{nm}[do([\alpha_1, ..., \alpha_n], S_0) = do([\alpha'_1, ..., \alpha'_m], S_0) \land f']$$

   In plain words: Two state variables are equal, if the states they are denoting are equal.
4. Suppose $f$ is regressable and of the form:

$$s = \text{do}([\alpha_1, ..., \alpha_n], S_0) \land f'$$

where $f'$ may be empty and does not mention any other equality atoms between $s$ and a state term rooted at $S_0$. Then

$$R_{nm}[f] = R_{nm}\left[f'|_{\text{do}([\alpha_1, ..., \alpha_n], S_0)}\right]$$

5. For the remaining possible appearances of a formula, the regression operator behaves as follows:

$$R_{nm}[\neg f] = \neg R_{nm}[f]$$
$$R_{nm}[f_1 \land f_2] = R_{nm}[f_1] \land R_{nm}[f_2]$$
$$R_{nm}[\exists v : f] = \exists v : R_{nm}[f]$$
$$R_{nm}[\exists s : f] = R_{nm}[f]$$

where $v$ is a variable of sort other than state and $s$ is a state variable.

The theorem to connect the regression operator with the actual goal which we want to achieve, stays the same:

$$\mathcal{W} \models \forall s : G(s) \equiv R_{nm}(G(s)).$$

Regression with the changed operator is still sound and complete. For a proof and further details see Gabaldon [1].

3.2.2 Example

Now consider the example from the introduction: A 10-litre barrel, a pint and a thirsty agent. Our new goal will be emptying the barrel:

$$G_{eb}(s) = \text{empty}(b, s)$$

where $b$ is the barrel and $s = \text{do}([\alpha_1, ..., \alpha_n], S_0)$. Furthermore, we define the situation as follows:
empty(b, S₀) = false
empty(p, S₀) = true
∀B, P, s : ¬empty(B, s) ∧ empty(P, s) ≡ Poss(draft(B, P), s)
∀P, s : ¬empty(P, s) ≡ Poss(drink(R, P), s)

\[ \gamma^+_{\text{empty}(B, s)}(B, \text{draft}(B, P), s) = \exists s_1, ..., s_{19}, p_1, ..., p_{19} : \]
\[ \text{do(draft}(B, p_1), s_1) \prec \text{do(draft}(B, p_2), s_2) \prec ... \]
\[ \prec \text{do(draft}(B, p_{19}), s_{19}) \prec \text{do(draft}(B, P), s) \]

\[ \gamma^-_{\text{empty}(B, s)}(B, \text{draft}(B, P), s) = false \]
\[ \gamma^+_{\text{empty}(P, s)}(P, \text{drink}(R, P), s) = true \]
\[ \gamma^-_{\text{empty}(P, s)}(P, \text{drink}(R, P), s) = false \]

In plain words: In the initial state the barrel is full and the pint is empty. Drafting beer from the barrel into the pint is possible, if the barrel is not empty and the pint is empty. Drinking from the pint is possible, if the pint is not empty. The barrel will become empty by drafting, if there were 19 drafting actions before. Refilling the barrel is not possible. Drinking from the pint makes it definitely empty.

Regression works just like before, but with the additional feature of the regression operator introduced in the preceding section in point 1, we can also decompose the \( \gamma^+_{\text{empty}(B, s)}(B, \text{draft}(B, P), s) \)-statement in our current scenario. This leads us backwards in the action sequence, thus showing that to achieve the goal 20 beers have to be draught.

4 Conclusion

As we have seen, the Markov property can be removed from the situation calculus by Reiter, by introducing other restrictions and adapting the regression operator. Backwards reasoning with the regression operator and without the Markov property appeared to be an efficient method for application situations where the Markov property is not present.

However, with a scenario where the Markov property is not present the following facts have to be considered. A scenario without the Markov property has not necessarily to be formalized with the extended situation calculus. Such a scenario can also be formalized with the original situation calculus and several new fluents like I have mentioned in the introduction. In the extended situation calculus the occurrences of those new fluents can be replaced by formulas that compute the fluents semantic statements on-the-fly. The fluents serve as explicit
knowledge about properties of the world. In the extended situation calculus the knowledge about these properties are derived from the agent’s action history.

Clearly, deriving knowledge from the fluents is quick, but requires more memory to store the explicit knowledge. Computing the knowledge from the action history is slower, but does not require memory for storing explicit knowledge.

One can easily imagine scenarios where one of these two approaches is preferable to the other one. Anyway, the advantage of the extended situation calculus is the possibility to use both approaches and even mix them.

References

