Workshop de Agentes y Sistemas Inteligentes

The Monkey and Bananas Problem Revisited: A Situation Calculus Approach

Gerardo I. Simari Diego R. García Gabriel R. Filocamo

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA)
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur. Av. Alem 1253, (8000) Bahía Blanca, Argentina
Tel: ++54 291 4595135 - Fax: ++54 291 4595136
{gis,drg,grf}@cs.uns.edu.ar

Abstract

This work develops a complete example of how to obtain a Situation Calculus Action Theory. The problem selected is a simple one, yet it is enough to cover the important details of the process without unnecessary complications. After obtaining the action theory, a running Prolog program is derived from the axioms; at this point, some problems, and a proposed solution, are presented.

Key Words: Artificial Intelligence, Cognitive Robotics, Action Theories, Situation Calculus.

1 Background and Motivation

The main objective of this work is to develop a complete example of an action theory in Reiter's Situation Calculus [Rei01], including details of every step of the process [SGF03]. The problem selected covers the process well, being simple enough to not introduce unnecessary complications.

After obtaining an action theory for the problem, we implement it as a Prolog program using the formalisms also described in [Rei01]. At the implementation stage, we encountered some problems while transforming the axioms into Prolog clauses; these problems, and how we solved them, will be described in the corresponding section.

The Situation Calculus is a second order language, quite suitable for representing dynamically changing worlds. Every one of the changes in the world is the result of a specific *action*, which is defined in the language. A *situation* stands for a history of the actions taken in the world (simply a sequence of actions), and is represented as a first order formula. A special situation, the constant s_0, is used to represent the initial state of the world. Every other situation is denoted by the special functional symbol $do(\alpha, s)$; this means that action α executed in situation s leads to a new situation, called $do(\alpha, s)$, where s is a situation built in this same way, or s_0. For example, for action $grab(x)$ (which denotes grabbing object x), $do(grab(x), s)$ is the situation which results from grabbing object x in situation s.

There exist a certain type of relations which have truth values that may vary from situation to situation; such relations are called *Relational Fluents*. They are represented by predicates that take a situation term as their last argument. Functions whose values depend on the situation are called *Functional Fluents*, and also take a situation term as their last argument. For example, the relational fluent $nextTo(x)$, which holds in s_0, no longer holds in $do(walk(y), s_0)$, where object x is different from object y.

Another type of relations, which do not depend on the situation, are represented by predicates without a situation parameter, and are called *Non-Fluent predicates*. Similarly, those functions that are situation independent are called *Non-Fluent functions*.

The theory is completed by a variety of *axioms*, which capture the different aspects of the world that is being modelled. These axioms are: *action precondition* axioms, *effect* axioms, and *successor state* axioms; these will be introduced in the corresponding section, following the example presented.

The rest of this work is organized as follows: in Section 2, we describe the problem to be solved, and the basic components of the problem are identified as elements of the Situation Calculus (actions, fluents, etc). Section 3 uses this description to obtain the full axiomatization for the action theory. Once this theory is obtained, Section 4 transforms the axioms into Lloyd-Topor Normal Form (LTNF) [Llo87], suitable for obtaining corresponding Prolog clauses, which are included in Section 5.

2 The Monkey and Bananas Problem

The Monkey and Bananas Problem is one of the first planning problems. It was proposed by John McCarthy in 1963, and later reprinted in [McC68]. A monkey is in a room that contains a bunch of bananas hanging from the ceiling, and a chair. The monkey can't reach the bananas from the floor. Nevertheless, if the chair were below the bananas, and the monkey were standing on top of it, then the bananas would become reachable. Initially, the monkey is not next to the chair, and the chair is not below the banana bunch. This description can be formalized by the following:

- **Actions:**
  - $walk(x)$: The monkey walks to object x.
  - $pushUnder(x, y)$: The monkey pushes object x to the location under object y.
  - $climb(x)$: The monkey climbs onto object x.
  - $climbDown(x)$: The monkey climbs down from object x.
  - $grab(x)$: The monkey grabs object x.
- **Relational Fluents:**
  - $onCeiling(x, s)$: Object x is on the ceiling in situation s.
  - $holding(x, s)$: The monkey is holding object x in situation s.
  - $nextTo(x, s)$: The monkey is next to object x in situation s.
  - $on(x, s)$: The monkey is on top of object x in situation s.
  - $below(x, y, s)$: Object x is below object y in situation s.
- **Functional Fluents:**
  - $stamina : situation \rightarrow I\!N$: The monkey's stamina in situation s.
- **Non-Fluent Predicates:**
  - $onFloor(x)$: Object x is on the floor.

- **Objects:**
  - Chair
  - Bananas

3 Problem Axiomatization

In this section, we will introduce the axioms necessary to model the problem in the Situation Calculus. In order to do this, the following steps will be followed:

- Specify the Action Precondition Axioms.
- Specify the Effect Axioms.
- Obtain the Normal Forms for the Effect Axioms.
- Derive the Successor State Axioms.
- Specify the initial database.
- Specify Non-fluent Predicates.

In order to expose the solution in a clear, non-redundant manner, some axioms will not be specified. Nevertheless, these axioms may appear as part of those axioms that emerge as the solution evolves.

3.1 Action Precondition Axioms

The Action Precondition Axioms specify the requirements that must be satisfied for an action to be executed. For this reason, there will be *one* axiom for each action. In the definition of these axioms the predicate $poss(a, s)$ is used, meaning that action a is possible (can be executed) in situation s.

An Action Precondition Axiom will have the following general form: $poss(a, s) \equiv \Upsilon$, where Υ is a first order formula. For the running example, the following are the Action Precondition Axioms for *walk* and *pushUnder*:

$$poss(walk(x), s) \equiv onFloor(x) \wedge \neg on(z, s) \wedge stamina(s) = y \wedge y \geq 1$$

$$\begin{aligned} poss(pushUnder(x, y), s) \equiv \; & nextTo(x, s) \wedge onCeiling(y, s) \wedge onFloor(x) \wedge \\ & \neg on(x, s) \wedge stamina(s) = z \wedge z \geq 2 \end{aligned}$$

The intuitive meaning of the first axiom, for example, is that in order to be able to walk to object x, object x must be on the floor, the monkey must not be on any object (denoted by variable z in the formula), and finally, the monkey's stamina must be at least 1. The symbol '$\equiv$' in the axioms means that the reverse implication is also true. The other formula can be read in a similar fashion.

3.2 Effect Axioms

The Effect Axioms specify how the relational and functional fluents' values change after the execution of an action. In this case, there will be *at least one* axiom for each fluent. An Effect Axiom for a relational fluent f will have the following general form:

$$\phi_f \rightarrow f(t^n, do(\alpha, s))$$

and for a functional fluent g:

$$\gamma_g \rightarrow g(t^n, do(\alpha, s)) = r$$

where $t^n = t_1, t_2, \cdots, t_n$ are terms, ϕ_f,γ_g are first order formulas, and '$\rightarrow$' is the logical connective for material implication. The Effect Axioms for relational fluents can be subdivided into positive and negative axioms. An axiom is *positive* if the relational fluent f is not negated in the formula, and *negative* otherwise. The following are the Effect Axioms for *nextTo*, *below*, and *stamina*:

$$nextTo(x, do(walk(x), s))$$
$$nextTo(x, do(climbDown(x), s))$$
$$x \neq y \rightarrow \neg nextTo(x, do(walk(y), s))$$
$$\neg nextTo(x, do(climb(x), s))$$
$$below(x, y, do(pushUnder(x, y), s))$$
$$y \neq z \rightarrow \neg below(x, y, do(pushUnder(x, z), s))$$
$$stamina(do(walk(x), s) = stamina(s) - 1$$
$$stamina(do(pushUnder(x, y), s) = stamina(s) - 2$$
$$stamina(do(climb(x), s) = stamina(s) - 1$$
$$stamina(do(climbDown(x), s) = stamina(s) - 1$$
$$stamina(do(grab(x), s) = stamina(s) - 1$$

The intuitive meaning of the first axiom, for example, is that the monkey will be *next to* object x in the situation resulting from executing action $walk(x)$ in a given situation s.

3.3 Normal Forms for Effect Axioms

In the previous section we saw that there can be more than one Effect Axiom for any given fluent. The result of transforming the Effect Axioms to the Normal Form are *two* axioms for each fluent: one that groups the positive effect axioms, called *positive normal form Effect Axiom*, and one that groups the negative ones, called *negative normal form Effect Axiom*. The need for this separation in positive and negative Normal Form Effect Axioms will become apparent in the following section.

3.3.1 Relational Fluents

For each positive Effect Axiom, for a relational fluent f of the form:

$$\phi_F^+ \rightarrow f(t^n, do(\alpha, s))$$

where $t^n = t_1, t_2, \cdots, t_n$ are terms, and ϕ_F^+ is a first order formula, we rewrite it in the equivalent form:

$$a = \alpha \wedge x^n = t^n \wedge \phi_F^+ \rightarrow f(x^n, do(a, s))$$

where $x^n = t^n$ abbreviates $x_1 = t_1 \wedge x_2 = t_2 \wedge \cdots \wedge x_n = t_n$, and x^n are new and distinct variables. This last formula can be rewritten:

$$(\exists y_1, \cdots, y_m)[a = \alpha \wedge x^n = t^n \wedge \phi_F^+] \rightarrow f(x^n, do(a, s))$$

where $y_1, \cdots, y_m$ are all the free variables (except for s) that appear in the original effect axiom.

This last process is done for every one of the k positive Effect Axioms for a fluent f. The result of this is a set of formulas:

$$\Psi^1 \rightarrow f(x^n, do(a,s)), \cdots, \Psi^k \rightarrow f(x^n, do(a,s))$$

These k formulas can be rewritten:

$$[\Psi^1 \vee \cdots \vee \Psi^k] \rightarrow f(x^n, do(a,s))$$

This is the Positive Normal Form for Effect Axioms for a fluent f. The Negative Normal Form is obtained in the same way, using the negative Effect Axioms instead.

The following are the Normal Forms for last section's Effect Axioms, where the first formulas are the original Effect Axioms, and the formulas following the "$\Longrightarrow$" are the corresponding Normal Forms:

- $nextTo(t, do(walk(t), s)), nextTo(t, do(climbDown(t), s)) \Longrightarrow$

  $\{[(\exists t) a = walk(t) \wedge x_1 = t] \vee [(\exists t) a = climbDown(t) \wedge x_1 = t]\} \rightarrow nextTo(x_1, do(a,s))$

- $below(t_1, t_2, do(pushUnder(t_1, t_2), s)) \Longrightarrow$

  $[(\exists t_1, t_2) a = pushUnder(t_1, t_2) \wedge x_1 = t_1 \wedge x_2 = t_2] \rightarrow below(x_1, x_2, do(a,s))$

- $t_1 \neq t_2 \rightarrow \neg nextTo(t_1, do(walk(t_2), s)), \neg nextTo(t, do(climb(t), s)) \Longrightarrow$

  $\{[(\exists t_1, t_2) a = walk(t_2) \wedge x_1 = t_1 \wedge x_2 = t_2 \wedge t_1 \neq t_2] \vee [(\exists t) a = climb(t) \wedge x_1 = t]\}$
  $\rightarrow \neg nextTo(x_1, do(a,s))$

- $t_2 \neq t_3 \rightarrow \neg below(t_1, t_2, do(pushUnder(t_1, t_3), s)) \Longrightarrow$

  $[(\exists t_1, t_2, t_3) a = pushUnder(t_1, t_3) \wedge x_1 = t_1 \wedge x_2 = t_2 \wedge x_3 = t_3 \wedge t_2 \neq t_3]$
  $\rightarrow \neg below(x_1, x_2, do(a,s))$

3.3.2 Functional Fluents

In much the same way as in the case of the Relational Fluents, for each Effect Axiom, for a functional fluent f of the form:

$$\phi_F \rightarrow f(t^n, do(\alpha, s)) = r$$

where $t^n = t_1, t_2, \cdots, t_n$ are terms, and ϕ_F is a first order formula, we rewrite it in the equivalent form:

$$a = \alpha \wedge x^n = t^n \wedge y = r \wedge \phi_F \rightarrow f(x^n, do(a,s)) = y$$

where $x^n = t^n$ abbreviates $x_1 = t_1 \wedge x_2 = t_2 \wedge \cdots \wedge x_n = t_n$, and x^n are new and distinct variables. This last formula can be rewritten:

$$(\exists y_1, \cdots, y_m)[a = \alpha \wedge x^n = t^n \wedge y = r \wedge \phi_F] \rightarrow f(x^n, do(a,s)) = y$$

where $y_1, \cdots, y_m$ are all the free variables (except for s) that appear in the original effect axiom.

This process is done for every one of the k Effect Axioms for a functional fluent f. The result of this is a set of formulas:

$$\Psi^1 \to f(x^n, do(a,s)), \cdots, \Psi^k \to f(x^n, do(a,s))$$

As before, these k formulas can be rewritten:

$$[\Psi^1 \vee \cdots \vee \Psi^k] \to f(x^n, do(a,s))$$

This is the Normal Form for Effect Axioms for a functional fluent f. The following are the transformations for the functional fluent in our example:

$$stamina(do(walk(t),s)) = stamina(s) - 1,$$

$$stamina(do(pushUnder(t_1,t_2),s)) = stamina(s) - 2,$$

$$stamina(do(climb(t),s)) = stamina(s) - 1,$$

$$stamina(do(climbDown(t),s)) = stamina(s) - 1,$$

$$stamina(do(grab(t),s)) = stamina(s) - 1 \Longrightarrow$$

$$\{[(\exists t_1)a = walk(t_1) \wedge x_1 = t_1 \wedge y = stamina(s) - 1] \vee$$

$$[(\exists t_1, t_2)a = pushUnder(t_1,t_2) \wedge x_1 = t_1 \wedge x_2 = t_2 \wedge y = stamina(s) - 2] \vee$$

$$[(\exists t_1)a = climb(t_1) \wedge x_1 = t_1 \wedge y = stamina(s) - 1] \vee$$

$$[(\exists t_1)a = climbDown(t_1) \wedge x_1 = t_1 \wedge y = stamina(s) - 1] \vee$$

$$[(\exists t_1)a = grab(t_1) \wedge x_1 = t_1 \wedge y = stamina(s) - 1]\}$$

$$\to stamina(do(a,s)) = y$$

3.4 Successor State Axioms

The Successor State Axioms provide a complete way of describing how the world evolves as a response to the execution of actions. Each axiom describes how to compute the value of a fluent for the next situation, given its value for the current situation. Intuitively the successor state axiom for a fluent f has the following structure [RN95]:

True afterwards ⇔ *[An action made it true*
∨ *True already and no action made it false]*

The use of '⇔' means that the axiom will be true after the execution of the action if it is made true or it stays true; and that it will be false otherwise. For each *relational fluent* f, the successor state axiom has the following form:

$$f(x^n, do(a,s)) \equiv \gamma_f^+(x^n,a,s) \vee f(x^n,s) \wedge \neg\gamma_f^-(x^n,a,s)$$

where $\gamma_f^+(x^n,a,s) \to f(x^n, do(a,s))$ and $\gamma_f^-(x^n,a,s) \to f(x^n, do(a,s))$ are the positive and negative normal form effect axioms for the fluent f, respectively.

The successor state axioms for our example are:

- $nextTo(x_1, do(a, s)) \equiv$
  $\{[(\exists t_1)a = walk(t_1) \land x1 = t_1] \lor [(\exists t_1)a = climbDown(t_1) \land x_1 = t_1]\} \lor$
  $nextTo(x_1, s) \land \neg\{[(\exists t_1, t_2)a = walk(t_2) \land x_1 = t_1 \land x_2 = t_2 \land t_1 \neq t_2]\lor$
  $[(\exists t_1)a = climb(t_1) \land x_1 = t_1]\}$

- $below(x_1, x_2, do(a, s)) \equiv$
  $[(\exists t_1, t_2)a = pushUnder(t_1, t_2) \land x_1 = t_1 \land x_2 = t_2]\lor$
  $below(x_1, x_2, s) \land \neg[(\exists t_1, t_2, t_3)a = pushUnder(t_1, t_3) \land x_1 = t_1 \land x_2 = t_2 \land\ x_3 = t_3 \land t_2 \neq t_3]$

For each functional fluent f, the successor state axiom has the following form:

$$F(x^n, do(a, s)) = y \equiv \gamma_F(x^n, y, a, s) \lor y = F(x^n, s) \land \neg(\exists y')\gamma_F(x^n, y', a, s)$$

where $\gamma_F(x^n, y, a, s) \rightarrow F(x^n, do(a, s)) = y$ is the normal form effect axiom for the functional fluent f. Intuitively, the successor state axiom for a functional fluent f has the following structure:

The value of f is v $\Leftrightarrow$ *an action changed its value to v, or its value was already v and no action modified it.*

The following is the Successor State Axiom for the sole functional fluent *stamina*:

$stamina(do(a, s)) = y \equiv$

$\{[(\exists t_1)a = walk(t_1) \land x_1 = t_1 \land y = stamina(s) - 1]\lor$

$[(\exists t_1, t_2)a = pushUnder(t_1, t_2) \land x_1 = t_1 \land x_2 = t_2 \land y = stamina(s) - 2]\lor$

$[(\exists t_1)a = climb(t_1) \land x_1 = t_1 \land y = stamina(s) - 1]\lor$

$[(\exists t_1)a = climbDown(t_1) \land x_1 = t_1 \land y = stamina(s) - 1]\lor$

$[(\exists t_1)a = grab(t_1) \land x_1 = t_1 \land y = stamina(s) - 1]\}$

$\lor\ y = stamina(s)\ \land\ \neg(\exists y')\{[(\exists t_1)a = walk(t_1) \land x_1 = t_1 \land y' = stamina(s) - 1]\lor$

$[(\exists t_1, t_2)a = pushUnder(t_1, t_2) \land x_1 = t_1 \land x_2 = t_2 \land y' = stamina(s) - 2]\lor$

$[(\exists t_1)a = climb(t_1) \land x_1 = t_1 \land y' = stamina(s) - 1]\lor$

$[(\exists t_1)a = climbDown(t_1) \land x_1 = t_1 \land y' = stamina(s) - 1]\lor$

$[(\exists t_1)a = grab(t_1) \land x_1 = t_1 \land y' = stamina(s) - 1]\}$

3.5 The Initial Database

Up to now, we have only defined how the world evolves as a result of the actions that are performed, but have said nothing about the initial situation s_0. In this section, we specify what holds in this situation in order to have a picture of the how the world "begins". The general form of the definitions is:

$$F(x_n, s_0) \equiv \Psi_F(x_n, s_0)$$

where Ψ_F is a first order formula. The following formulas define the initial database for the running example:

$$\begin{array}{ll} onCeiling(x, s_0) \equiv x = bananas & \neg on(x, s) \equiv x = chair \\ \neg holding(x, s_0) \equiv x = bananas & \neg below(x_1, x_2, s) \equiv x_1 = chair \wedge x_2 = bananas \\ \neg nextTo(x, s_0) \equiv x = chair & stamina(s_0) = y \equiv y = 8 \end{array}$$

3.6 Non-Fluent Predicates

All of the predicates used in the example so far have been fluent; this means that they can change their value depending on the situation. There exist, however, predicates whose values don't change, called *Non-Fluent Predicates.* These predicates obey the following general form:

$$P(x^n) \equiv \Theta_n(x^n)$$

where $\Theta_n(x^n)$ is a situation independent first order formula. In our example, the only predicate of this sort is:

$$onFloor(x) \equiv x = chair$$

4 Implementation of the Action Theory

In order to obtain a Prolog program that implements a situation calculus action theory, Lloyd and Topor [Llo87] propose a set of transformations based on Clark's theorem [Cla78] that reformulate the Action Precondition and Successor State axioms in a way that results in a direct translation to Prolog clauses.

Once an action theory is obtained (following the process described above), the following steps are necessary to build a Prolog program:

- **Transform the Action Precondition and Succesor State Axioms:**
  - Obtain the if-halves from the Action Precondition Axioms.
  - Apply the Lloyd-Topor transformation rules to the if-halves in order to obtain formulas that are easy to implement (Lloyd-Topor normal form).
- **Initial Database definition:** Obtain the if-halves from the initial database definition. Recall that these definitions obey the form:

$$F(x_n, s_0) \equiv \Psi_F(x_n, s_0)$$

where $F(x_n, s_0)$ will not be negated in this case. This is due to the fact that negative information is represented in Prolog by the *Closed World Assumption* (CWA). It should also be noted that F cannot be a functional fluent in this case because this type of fluent cannot be directly implemented in Prolog. Notwithstanding, an n-ary functional fluent can be implemented as a $(n+1)$-ary Prolog predicate, where the last argument represents the value of the fluent.

- **Non-Fluent Predicate definition:** Obtain the if-halves from the definitions, and then apply the Lloyd-Topor transformation rules to these if-halves.

4.1 Transformation of Action Precondition Axioms

The following are the Action Precondition Axioms, followed by their corresponding if-halves. It is worth noting that these if-halves are already in Lloyd-Topor Normal Form.

- $poss(walk(x), s) \equiv onFloor(x) \wedge \neg on(y, s)$

  If-half: $onFloor(x) \wedge \neg on(y, s) \rightarrow poss(walk(x), s)$

- $poss(pushUnder(x, y), s) \equiv nextTo(x, s) \wedge onCeiling(y, s) \wedge onFloor(x) \wedge \neg on(x, s)$

  If-half: $nextTo(x, s) \wedge onCeiling(y, s) \wedge onFloor(x) \wedge \neg on(x, s)$
  $\rightarrow poss(pushUnder(x, y), s)$

4.2 Transformation of Successor State Axioms

We now describe the transformed Successor State Axioms. The list includes the axiom, its corresponding if-half, and the resulting formula in Lloyd-Topor Normal Form (LTNF).

- $nextTo(x_1, do(a, s)) \equiv$

  $$\{[(\exists t)a = walk(t) \wedge x_1 = t] \vee [(\exists t)a = climbDown(t) \wedge x_1 = t]\} \vee$$
  $$nextTo(x_1, s) \wedge \neg\{[(\exists t_1, t_2)a = walk(t_2) \wedge x_1 = t_1 \wedge x2 = t_2 \wedge$$
  $$t_1 \neq t_2] \vee [(\exists t)a = climb(t) \wedge x_1 = t]\}$$

  If-half: $[(\exists t)a = walk(t) \wedge x_1 = t] \vee [(\exists t)a = climbDown(t) \wedge x_1 = t] \vee$

  $$nextTo(x_1, s) \wedge \neg\{[(\exists t_1, t_2)a = walk(t_2) \wedge x_1 = t_1 \wedge x2 = t_2 \wedge t_1 \neq t_2] \vee$$
  $$[(\exists t)a = climb(t) \wedge x_1 = t]\} \rightarrow nextTo(x_1, do(a, s))$$

  LTNF: $\{[a = walk(t) \wedge x_1 = t] \vee [a = climbDown(t) \wedge x_1 = t]\} \vee$

  $$nextTo(x_1, s) \wedge \{[\neg(a = walk(t_2)) \vee \neg(x_1 = t_1) \vee \neg(x_2 = t_2) \vee$$
  $$\neg(t_1 \neq t_2)] \vee [\neg(a = climb(t)) \vee \neg(x_1 = t)]\} \rightarrow nextTo(x_1, do(a, s))$$

- $below(x_1, x_2, do(a, s)) \equiv$

  $$[(\exists t_1, t_2)a = pushUnder(t_1, t_2) \wedge x_1 = t_1 \wedge x_2 = t_2] \vee$$
  $$below(x_1, x_2, s) \wedge \neg[(\exists t_1, t_2, t_3)a = pushUnder(t_1, t_3) \wedge x_1 = t_1 \wedge$$
  $$x_2 = t_2 \wedge x_3 = t_3 \wedge t_2 \neq t_3]$$

If-half: $[(\exists t_1, t_2) a = pushUnder(t_1, t_2) \wedge x_1 = t_1 \wedge x_2 = t_2] \vee$

$below(x_1, x_2, s) \wedge \neg[(\exists t_1, t_2, t_3) a = pushUnder(t_1, t_3) \wedge x_1 = t_1 \wedge$

$x_2 = t_2 \wedge x_3 = t_3 \wedge t_2 \neq t_3] \rightarrow below(x_1, x_2, do(a, s))$

LTNF: $[a = pushUnder(t_1, t_2) \wedge x_1 = t_1 \wedge x_2 = t_2] \vee below(x_1, x_2, s) \wedge$

$[\neg(a = pushUnder(t_1, t_3)) \vee \neg(x_1 = t_1) \vee \neg(x_2 = t_2) \vee \neg(x_3 = t_3) \vee$

$\neg(t_2 \neq t_3)] \rightarrow below(x_1, x_2, do(a, s))$

4.3 Non-Fluent Predicate and Initial Database Definitions

Next, we specify the LTNF for the Initial Database. Some of the formulas that appeared in the definition above will no longer apply, due to the fact that negated formulas are captured by the CWA. Note that the database is closed.

- $onCeiling(x, s_0) \equiv x = bananas$

  If-half, LTNF: $x = bananas \rightarrow onCeiling(x, s_0)$

- $\neg holding(x, s_0) \equiv x = bananas$,
  $\neg nextTo(x, s_0) \equiv x = chair, \neg on(x, s) \equiv x = chair$,
  $\neg below(x_1, x_2, s) \equiv x_1 = chair \wedge x_2 = bananas\cdot$

  These formulas don't have corresponding Prolog clauses due to the CWA.

In last place, we have the LTNF for our sole Non-Fluent Predicate:

$onFloor(x) \equiv x = chair$

If-half, LTNF: $x = chair \rightarrow onFloor(x)$

5 A Prolog Program

The prolog clauses of the program that implements the definitional theory are obtained from the LTNF of the If-halves described in the last section. Simply replace the occurrences of "¬" by Prolog's `not`, and replace conjunction and disjunction by Prolog's "`,`" and "`;`" respectively.

To guarantee the soundness of a Prolog implementation of a definitional theory, a *proper* Prolog interpreter must be used. A proper Prolog interpreter is one that evaluates a negative literal `not A`, using negation as failure only when (at the time of evaluation) the atom `A` is ground. When `A` is not ground, the interpreter may suspend its evaluation, working on other literals until `A` becomes ground, or it may abort its evaluation. Either way, it *never* tries to fail on non-ground atoms [Rei01]. The implementation was developed in ECLiPSe Prolog, a proper Prolog interpreter which uses "∼" for negation as failure with the semantics described above.

During the implementation, we encountered certain problems with the clauses obtained from applying these transformations. For example, the following shows the transformation of the LTNF successor state axiom for the fluent *nextTo* into the corresponding Prolog clause:

LTNF: $\{[a = walk(t) \wedge x_1 = t] \vee [a = climbDown(t) \wedge x_1 = t]\}\vee$

$$nextTo(x_1, s) \wedge \{[\neg(a = walk(t_2)) \vee \neg(x_1 = t_1) \vee \neg(x_2 = t_2)\vee$$

$$\neg(t_1 \neq t_2)] \vee [\neg(a = climb(t)) \vee \neg(x_1 = t)]\} \rightarrow nextTo(x_1, do(a, s))$$

```
nextTo(X,do(A,S)):-
 1.                    ( A=walk(T), X=T
                        ;
 2.                      A=climbDown(T), X=T
                        ;
 3.                      nextTo(X,S),
 4.                      ( ~(A=climb(T1)) ; ~(X=T1) ),
 5.                      ( ~(A=walk(T2)) ; ~(X=T1) ; ~(Y=T2); ~(X=Y) )
                       ).
```

Note that in line 5 neither `T2` nor `Y` will be instantiated when ∼`(Y=T2)` and ∼`(X=Y)` are evaluated. This is due to the fact that Prolog does not maintain the bindings of the variables from one clause to another in a disjunction of clauses when using ";".

The solution we found to this problem is to never apply the following transformation:

$$lt(\neg[W_1 \wedge W_2]) = lt(\neg W_1) \vee lt(\neg W_2)$$

given that the application of this transformation generates a disjunction of negated clauses, which gives rise to the problem mentioned above. Instead of applying this transformation, apply:

$$lt(\neg[W_1 \wedge \cdots \wedge W_n]) = \neg p(X_1, \cdots, X_m)$$

where p is a new predicate symbol not appearing in the formulas, and $X_1, \cdots, X_m$ are all the free variables in $W_1, \cdots, W_n$. The new predicate is defined as follows:

$$W_1 \wedge \cdots \wedge W_n \rightarrow p(X_1, \cdots, X_m)$$

The new derivation will be as follows:

If-half: $[(\exists t)a = walk(t) \wedge x_1 = t] \vee [(\exists t)a = climbDown(t) \wedge x_1 = t]\vee$

$$nextTo(x_1, s) \wedge \neg\{[(\exists t_1, t_2)a = walk(t_2) \wedge x_1 = t_1 \wedge x2 = t_2 \wedge t_1 \neq t_2]\vee$$

$$[(\exists t)a = climb(t) \wedge x_1 = t]\} \rightarrow nextTo(x_1, do(a, s))$$

After applying the new transformation:

$$[(\exists t)a = walk(t) \wedge x_1 = t] \vee [(\exists t)a = climbDown(t) \wedge x_1 = t]\vee$$

$$nextTo(x_1, s) \wedge \neg p(x_1, a) \wedge \neg q(x_1, a) \rightarrow nextTo(x_1, do(a, s))$$

where $[(\exists t_1, t_2)a = walk(t_2) \wedge x_1 = t_1 \wedge x2 = t_2 \wedge t_1 \neq t_2] \rightarrow p(x_1, a)$

and $[(\exists t)a = climb(t) \wedge x_1 = t] \rightarrow q(x_1, a)$.

The following is the Prolog code for this axiom:

```
p(X,A):- A = climb(T1), X = T1.
q(X,A):- A = walk(T2), X = T1, Y = T2, ~ (T1 = T2).
nextTo(X,do(A,S)):-( A = climbDown(T), X = T
                     ;
                     A = walk(T), X = T
                     ;
                     nextTo(X,S),
                     ~ p(X,A), ~ q(X,A)
                   ).
```

6 Final Remarks and Future Work

In this paper, we have implemented an action theory for a simple problem. In the process, we have found (and suggested a solution to) a problem in the transformation rules that are the basis for obtaining Prolog implementations from a set of axioms. Even though the Monkey and Bananas problem is a simple one, the process of deriving an implementation can be quite tedious. Furthermore, this process is quite sensitive to modifications to the original axioms. For example, a simple modification such as adding a new object to the world would imply deriving the complete action theory from scratch.

Future work, motivated by the previous observations, involves obtaining an automatic process that derives an executable implementation of a given action theory. In other words, what we want is to be able to compile the language of the Situation Calculus into an object language such as Prolog. There are many advantages to having such a compiler, such as easing the design of action theories. For example, if the initial axioms suffer modifications, the final result can be quickly visualized.

References

[Cla78] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322, New York, 1978. Plenum Press.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming, Second Edition.* Springer-Verlag, 1987.

[McC68] J. McCarthy. Situations, actions, and causal laws. In *Semantic Proceedings of the tri-annual IFIP Conf, Minsky (ed), Machine Intelligence, eds: Meltzer, and Michie, vars. PublishersT Press.* 1968.

[Rei01] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and implementing Dynamical Systems.* MIT Press, Cambridge, Massachusetts - London, England, 2001.

[RN95] Stuart J. Russell and Peter Norvig. *Artificial Intelligence. A Modern Approach.* Prentice-Hall, Englewood Cliffs, 1995.

[SGF03] Gerardo I. Simari, Diego R. García, and Gabriel R. Filocamo. A general approach to the implementation of action theories. In Nelson Acosta, editor, *Proceedings of the V Workshop de Investigadores en Ciencias de la Computación*, pages 541–545. Universidad del Centro de la Provincia de Buenos Aires, Tandil, Buenos Aires, Argentina, 2003.