Temporal Logic and reasoning about actions

Gisela Mendez
Depto. de Matemáticas
Universidad Central de Venezuela
Caracas, VENEZUELA
gmendez@dino.conicit.ve

Jorge Lobo*
Department of EECS
Univ. of Illinois at Chicago
Chicago, IL 60607, USA
jorge@eecs.uic.edu

Jimena Llopis
Depto de Matemáticas
Universidad Simon Bolivar
Caracas, VENEZUELA
jllopis@usb.ve

Chitta Baral†
Department of Computer Sc.
Univ. of Texas at El Paso
El Paso, TX, 79968, USA
chitta@cs.utep.edu

Abstract

This paper shows how past temporal logic can be incorporated into the action description language $\mathcal{A}$ [4] and its extensions. Incorporating past temporal logic will allow us to elegantly express effects of actions that depend not only on the current state of the world but on previous actions and states. It will also allow us to elegantly express dynamic constraints beyond the one-step dynamic constraints discussed in [2].

We also discuss why using temporal logic increases the elaboration tolerance in an evolving action theory, and consider other language extensions that are necessary from the elaboration tolerance point of view.

1 Introduction

In the past years there has been a significant progress in the development of logical formalisms to reason about actions [5, 1]. The formalisms are intended to precisely capture the effects that an action produces given the current description of the world and the preconditions that specify when the action can be executed. They allow us to do temporal projections to predict the state of the world after the execution of a sequence of actions. These formalisms can reason (sometimes partially) about

*Support was provided by the National Science Foundation, under grant Nr. IRI-92-10220.
†Support was provided by the National Science Foundation, under grant Nr. IRI-9211662 and IRI-9501577.

the concurrent execution of actions, the non-deterministic effects of some actions and avoid many instances of the qualification, ramification and frame problems.

Most of these formalism encode constraints and preconditions that can be verified given a state (either an initial state or state obtained through temporal projection) and determine on one hand, if an action can be executed, and on the other hand, the state that results from the execution of the action (i.e the projected state).

However, in real situations, consequences and preconditions of actions not only depend on the current (physical) state[1] of the world but also on past states and previously executed actions. Consider for example, *the boy who cries wolf.* The fact that the boy cried wolf several times in the past when the wolf was not around has a bearing on his later action in that when the wolf really appears nobody believes the boy. Another example, is the *three strikes and you are out rule* in California for felonies.

The goal of this paper is to demonstrate how past temporal logic can be incorporated into formalisms for reasoning about actions to express constraints and conditions that depend on the evolution of situations rather than simply on the current physical state of the world. The use of temporal logic also positively impacts the elaboration tolerance of action description languages; particularly when new fluents are introduced.

We start with the language $\mathcal{A}$ [4]. This language was designed as a minimal core of a high level language to represent and reason about actions and their effects. Domain descriptions written in this language have direct translations into extended logic programs.

We introduce a new dialect of $\mathcal{A}$ that we call Past Temporal $\mathcal{A}$. Domain descriptions in Past Temporal $\mathcal{A}$ can express the effects of an action that depends on past actions or states. For example, we can express situations such as *a citizen can be nominated to the presidency (become a candidate) only if (s)he has not been* **previously** *a felon.*

We first introduce the syntax and the semantics of Past Temporal $\mathcal{A}$. Then, we present a sound and complete translation of domain descriptions written in Past Temporal $\mathcal{A}$ into extended logic programs. In the second part of the paper we take the state specification language [2] and expand it with past temporal formulas to define temporal state specifications which can express dynamic constraints.

2 Temporal $\mathcal{A}$

The language *Temporal* $\mathcal{A}$ incorporates 'Past Temporal Logic' (PTL) [3] into $\mathcal{A}$. It has two disjoint non-empty sets of symbols called *fluents* and *actions*, and two kinds of *propositions*: *initial* propositions and *effect* propositions. An initial proposition is an expression of the form

$$\textbf{initially } f \tag{1}$$

[1] By physical state we mean the state that can be inferred by the current reading of the sensors without referring to past readings of the sensors.

where f is a fluent literal. A *fluent literal* is either a fluent or a fluent preceded by $\neg$. A fluent literal is *negative* if it is preceded by $\neg$, otherwise it is called positive. An effect proposition is an expression of the form

$$a \textbf{ causes } f \textbf{ if } p_1, \ldots, p_n \tag{2}$$

where a is an action, and f is a fluent literal and each of $p_1, \ldots, p_n$ $(0 \leq n)$ are formulas in *Past Temporal Logic for Actions* (PTLA). A PTLA formula is defined as follows

1. A fluent literal is a PTLA formula.
2. If p is an action or a PTLA formula then $(\textbf{lasttime}\ p)$ and $\neg(\textbf{lasttime}\ p)$ are PTLA formulas.
3. If f is fluent literal and p is either an action or a PTLA formula then $(f \textbf{ since } p)$ and $\neg(f \textbf{ since } p)$ are PTLA formulas.
4. If p_1 and p_2 are PTLA formulas so is $p_1 \wedge p_2$.
5. Nothing else are PTLA formulas.

The symbols **lasttime** and **since** are called temporal connectors. If a PTLA formula does not have occurrences of temporal connectors is called a non-temporal formula. It is shown in [3] that the temporal constructs " **since** " and "**lasttime** " are sufficient to represent a large class of temporal information. For example, **previously** f can be expressed by *true* **since** f, and **throughout_the_past** f by $\neg$**previously** $\neg f$.

We will now consider a domain description that include PTLA formulas.

Example 2.1 *Suppose that we have a water gun and a blow drier. We have the following rules: (1) If we squirt Fred with the water gun he gets wet. (2) If it rains he also gets wet. (3) If we blow dry him he gets dry. (4) If we squirt Fred and we don't dry him and it rains he will get pneumonia. We can specify this four proposition in Temporal $\mathcal{A}$ as follows*

$$D_1 \begin{cases} r_1: \ squirt \textbf{ causes } \neg dry \\ r_2: \ rain \textbf{ causes } \neg dry \\ r_3: \ blow\_dry \textbf{ causes } dry \\ r_4: \ rain \textbf{ causes } pneumonia \textbf{ if } (\neg dry \textbf{ since } squirt) \end{cases}$$

□

In the above example, it is possible to eliminate the PTLA formula in the last proposition by modifying the domain description and introducing a new fluent. However We argue later in the paper that eliminating PTLA formulas indiscriminately may not be elaboration tolerant.

Example 2.2 *Bulbs in slide projectors are very delicate. If in sequence we turn on, turn off and turn on the projector without waiting, the bulb will burn (here we could assume there is an action wait). In temporal $\mathcal{A}$ this can be expressed as follows:*

$$D_2 \begin{cases} on\ \textbf{causes}\ being\_on \\ \quad \textbf{if}\ \neg(\textbf{lasttime}\ \text{off}), (\textbf{lasttime}\ (\textbf{lasttime}\ on)) \\ on\ \textbf{causes}\ burn \\ \quad \textbf{if}\ (\textbf{lasttime}\ \text{off}), (\textbf{lasttime}\ (\textbf{lasttime}\ on)) \\ \text{off}\ \textbf{causes}\ \neg being\_on \end{cases}$$

□

We will now define the semantics of Temporal $\mathcal{A}$.

A *state* is a set of fluents. A *causal interpretation* is a partial function Φ from sequence of actions to states such that:

1. The empty sequence [] belongs to the domain of Φ, and
2. Φ is prefix-closed, i.e. for any sequence of actions α and any action a, if the concatenation of α and a, denoted by $\alpha \circ a$, is in the domain of Φ then α must also be in the domain of Φ.

A sequence of actions α is *possible* in a causal interpretation Φ if $\Phi(\alpha)$ is defined. Before we introduce models of domain descriptions we need to define when a PTLA formula is true for a given sequence of actions and a causal interpretation Φ.

Definition 2.3 *Let p be a PTLA formula, Φ a causal interpretation, and $\alpha = [a_0, \ldots, a_n]$, a possible sequence of actions in Φ. p is true in $\Phi(\alpha)$ iff one of the following cases holds:*

1. *p is a fluent and $p \in \Phi(\alpha)$.*
2. *$p = \neg f$ and f is not true in $\Phi(\alpha)$.*
3. *$p = p_1 \wedge p_2$ and both p_1 and p_2 are true in $\Phi(\alpha)$.*
4. *$p =$ **lasttime** a, a is an action and $a = a_n$.*
5. *$p =$ **lasttime** p' and p' is true in $\Phi([a_0, \ldots, a_{n-1}])$, where p' is a PTLA formula.*
6. *$p = f$ **since** a and a is an action such that there exists j, $0 \leq j \leq n$ where $a = a_j$ and f is true in $\Phi([a_0, \ldots, a_i])$ for every i, $j \leq i \leq n$.*
7. *$p = f$ **since** p' and p' is a PTLA formula such that there exists j, $0 \leq j < n$ where p' is true in $\Phi([a_0, \ldots, a_j])$ and f is true in $\Phi([a_0, \ldots, a_i])$, i, $j < i \leq n$.*

Note that there are no positive temporal PTLA formulas true in $\Phi([])$.

A causal interpretation Φ is a *model* of a domain description D iff

1. For a fluent f, if **initially** $f \in D$, then f is true in $\Phi([])$.

2. For every effect proposition a **causes** f **if** $p_1, \ldots, p_m$ and every possible sequence actions α possible in Φ if $p_1, \ldots, p_m$ are true in $\Phi(\alpha)$, then f is true in $\Phi(\alpha \circ a)$.

3. If D does not include such effect proposition for f, then f is true in $\Phi(\alpha \circ a)$ iff f is true in $\Phi(\alpha)$.

Example 2.4 *Following Example 2.1, let* $s_0 = []$, $s_1 = [squirt]$, $s_2 = [squirt, blow\_dry]$, $s_3 = [squirt, blow\_dry, squirt]$, $s_4 = [squirt, blow\_dry, squirt, rain]$, $s_5 = [squirt, blow\_dry, squirt, rain, blow\_dry]$. *We now show an example of a causal interpretation* Φ *that is also a model. Choose for example* $\Phi(s_0) = dry$ *(i.e.* dry *and* $\neg pneumonia$ *are the fluent literals true in* $\Phi(s_0)$*).* $\Phi(s_1) = \{\}$ *(i.e.* $\neg dry$, $\neg pneumonia$ *are true in* $\Phi(s_1)$*). This must be so since by part 2 of the definition of model* $\neg dry$ *must be true in* $\Phi(s_1)$*; and by 3,* $\neg pneumonia$ *must be true. Similar arguments show that in order of* Φ *to be a model we have:* $\Phi(s_2) = \{dry\}$*;* $\Phi(s_3) = \{\}$*;* $\Phi(s_4) = \{pneumonia\}$*;* $\Phi(s_5) = \{dry, pneumonia\}$*;*

A *query* in Temporal $\mathcal{A}$ is of the form

$$f \textbf{ after } a_1, \ldots, a_n$$

where f is a fluent and a_i's are actions. We say the above query is true w.r.t. a model Φ of a domain description iff f is true in $\Phi([a_1, \ldots, a_n])$. We say a domain description D entails a query q (i.e. $D \models q$) iff q is true in all models of D.

3 Elaboration tolerance in Temporal $\mathcal{A}$

Let us revisit Example 2.1. Suppose at one point in the development of the domain description the domain description consists of the following:

$$D_3^1 \begin{cases} r_1 : & squirt \textbf{ causes } \neg dry \\ r_2 : & rain \textbf{ causes } \neg dry \\ r_3 : & blow\_dry \textbf{ causes } dry \end{cases}$$

Suppose now we are told that *rain* will cause pneumonia to Fred if he is wet since he was squirted. One approach to incorporate this additional knowledge is to add a new effect proposition with temporal formulas. This is the approach taken in Example 2.1.

Another approach would be to introduce a new fluent called *susceptible_to_pneumonia* and add the effect proposition

rain **causes** *pneumonia* **if** *susceptible_to_pneumonia*.

The new fluent is meant to record past (or historical) situations of the domain. But that is not sufficient. We need to reconsider all the previous effect propositions in the domain description and add additional propositions relating actions in the current effect propositions and the newly introduced fluent *susceptible_to_pneumonia*. In our example we would need to add the following effect propositions.

squirt **causes** *susceptible_to_pneumonia*.

blow_dry **causes** $\neg$*susceptible_to_pneumonia*.

It is obvious that the above modification is extensive. In general, every time we introduce a new proposition that has a PTLA formula, to avoid the PTLA formula we will be required to make extensive changes to the rest of the domain description. *Since one of the main goals of knowledge representation is elaboration tolerance we need languages such as "Temporal $\mathcal{A}$ " which allow elegant representation of knowledge.*

On the other hand, there are situations in which "historical" fluents may add more elaboration tolerance to the knowledge represented.[2] Assume that Fred does some volunteer work at the local hospital and consider the action *help_patients* which makes Fred susceptible to pneumonia. If we were to incorporate the effect of this new action to our theory it is more elaboration tolerant if we have an explicit fluent *susceptible_to_pneumonia*.

One elegant way to take into account both these concerns is to use the idea of *indirect effects and constraints [7]*. In the above example we introduce a constraint that relates the newly introduced fluent *susceptible_to_pneumonia* and the PTLA formula used before. The resultant theory would be as follows:

$$D_3^2 \left\{ \begin{array}{l} r_1 : \ \mathit{squirt} \ \mathbf{causes} \ \neg \mathit{dry} \\ r_2 : \ \mathit{rain} \ \mathbf{causes} \ \neg \mathit{dry} \\ r_3 : \ \mathit{blow\_dry} \ \mathbf{causes} \ \mathit{dry} \\ r_4' : \ \mathit{rain} \ \mathbf{causes} \ \mathit{pneumonia} \ \mathbf{if} \ \mathit{susceptible\_to\_pneumonia} \\ r_5 : \ \mathit{help\_patients} \ \mathbf{causes} \ \mathit{susceptible\_to\_pneumonia} \\ r_6 : \ (\neg \mathit{dry} \ \mathbf{since} \ \mathit{squirt}) \ \mathbf{influences} \ \mathit{susceptible\_to\_pneumonia} \end{array} \right.$$

The last proposition in the above domain description is a constraint. Such propositions are not allowed in Temporal $\mathcal{A}$. In a later section we discuss a language where such constraints are expressible.

[2] This was pointed to us by Norm McCain.

4 Translation to extended logic programs

In this section we will present a translation from domain descriptions to extended logic programs. We will also show that there is a one-to-one correspondence between the models of a domain description D and the answer sets of the resulting logic program Π_D.

The program Π_D consists of rules derived from the propositions in D plus an extra-set of rules that is independent of D. The extended logic program uses variables of five sorts: *situation* variables denoted by S or S' with or without sub-indices; *fluent* variables denoted by F or F' with or without sub-indices; *action* variables denoted by A or A' with or without sub-indices, *Past Temporal formula* variables denoted by P or P' with or without sub-indices, and variables R or R' for actions or PTLA formulas. Constant and function symbols are represented with words starting with lower case letters. There is a special constant that represent the initial situation and is denoted by s_0.

The translation of an initial proposition of the form **initially** f is

$$holds(f, s_0) \leftarrow,$$

if f is a positive fluent literal. If $f = \neg g$, then the translation is

$$\neg holds(g, s_0) \leftarrow .$$

An effect proposition of the form a **causes** f **if** $p_1, \ldots, p_n$ where f is a positive fluent literal is translated into

$$\begin{array}{l} holds(f, res(a, S)) \leftarrow holds(p_1, S), \ldots, holds(p_n, S). \\ non\_inertia(f, a, S) \leftarrow \\ \qquad\qquad\qquad holds(p_1, S), \ldots, holds(p_n, S). \end{array}$$

If $f = \neg g$, then a **causes** f **if** $p_1, \ldots, p_n$ is translated into

$$\begin{array}{l} \neg holds(g, res(a, S)) \leftarrow \\ \quad holds(p_1, S), \ldots, holds(p_n, S). \\ non\_inertia(\overline{g}, a, S) \leftarrow \\ \quad holds(p_1, S), \ldots, holds(p_n, S). \end{array}$$

The remaining rules are rules that capture the general behavior of the domain. The first rule is the *full awareness* rules. For each fluent f we have the disjunction

$$holds(f, s_0) \vee \neg holds(f, s_0) \leftarrow .$$

which encodes that in the initial state the value of every fluent must be known. The second set of rules are the two rules of *inertia*

$$\begin{array}{l} holds(F, res(A, S)) \leftarrow \\ \qquad holds(F, S), \textbf{not } non\_inertia(\overline{F}, A, S). \\ \neg holds(F, res(A, S)) \leftarrow \\ \qquad \neg holds(F, S), \textbf{not } non\_inertia(F, A, S). \end{array}$$

These rules encode that only minimal changes happen as a consequence of an action (from part 3 of the causal model definition). The next set of rules encode the evaluation of Past Temporal formulas in a given situation S.

Conjunction rule:

$holds(P_1 \wedge P_2, S) \leftarrow holds(P_1, S), holds(P_2, S).$

Negation rule:

$holds(\neg P, S) \leftarrow \neg holds(P, S).$

Since rules:

$holds(F \textbf{ since } A, res(A, S)) \leftarrow$
$\quad holds(F, res(A, S)).$
$holds(F \textbf{ since } P, res(A, S)) \leftarrow$
$\quad \neg action(P), holds(F, res(A, S)), holds(P, S).$
$holds(F \textbf{ since } P, res(A, S)) \leftarrow$
$\quad holds(F, res(A, S)), holds(F \textbf{ since } P, S).$

Last time rules:

$holds(\textbf{lasttime } A, res(A, S)) \leftarrow .$
$holds(\textbf{lasttime } P, res(A, S)) \leftarrow$
$\quad \neg action(P), holds(P, S).$

The final set of rules define the relation *action*. For each action a in the language, we add the rule:

$action(a).$

We complete the definition of actions with the following rule:

$\neg action(A) \leftarrow \textbf{not } action(A).$

Example 4.1 *The translation of the temporal domain description in Example 2.2 is:*

$$holds(being\_on, s_0) \vee \neg holds(being\_on, s_0) \leftarrow .$$
$$holds(burn, s_0) \vee \neg holds(burn, s_0) \leftarrow .$$

$holds(being\_on, res(on, S)) \leftarrow$
$\quad holds(\neg(\textbf{lasttime } off), S),$
$\quad holds((\textbf{lasttime } (\textbf{lasttime } on)), S).$
$non\_inertia(\neg being\_on, on, S) \leftarrow$
$\quad holds(\neg(\textbf{lasttime} off), S),$
$\quad holds((\textbf{lasttime}(\textbf{lasttime} on)), S).$
$holds(burn, res(on, S)) \leftarrow$
$\quad holds((\textbf{lasttime } off), S),$
$\quad holds((\textbf{lasttime } (\textbf{lasttime } on)), S).$
$non\_inertia(\neg burn, on, S) \leftarrow$
$\quad holds((\textbf{lasttime } off), S),$
$\quad holds((\textbf{lasttime } (\textbf{lasttime } on)), S).$
$\neg holds(being\_on, res(off, S)) \leftarrow .$

$non\_inertia(being\_on, off, S) \leftarrow .$
$action(on) \leftarrow .$
$action(off) \leftarrow .$

plus the two rules of inertia, the conjunction rule, the negation rule, the last time rule, the since rules, and $\neg action(A) \leftarrow$ **not** $action(A)$.

Theorem 4.2 *Let D be a domain description. There exists a causal model Φ of D such that a fluent literal F is true in $\Phi([a_1, \ldots, a_n])$ for a possible sequence of actions $[a_1, \ldots, a_n]$ and $n \geq 0$ iff Π_D has an answer set X_Φ such that*

$$holds(F, res(a_n, res(\ldots, res(a_1, s_0)\ldots))) \in X_\Phi$$

if F is a positive fluent literal or

$$\neg holds(F', res(a_n, res(\ldots, res(a_1, s_0)\ldots))) \in X_\Phi$$

if $F = \neg F'$.

5 Temporal State Specification

State specifications were introduced in [2] to reason about actions and their effects. The language of state specifications allows the description of direct, indirect and non-deterministic effects of actions by the direct specification of the effects of an action or by global constraints (static and one-step dynamic constraints only) that apply to the domain.

A specification consists of a series of statements about preconditions that must hold in the state reached after the execution of an action a given the current state of the domain. Thus, from the current state and a state specification we can generate the effects E_a that an action a has over the state, i.e. the fluents that change their truth values from the current state to the resulting state.

There are four operators that can be applied to a fluent in a state specification: **in**, **out**, **was_in** and **was_out**. Informally **in**(f) means that the fluent f is true in the resulting state; **out**(f) that f is false. **was_in**(f) says that f is true in the current state (before the action is executed) and **was_out**(f) that f is false.

A state specification is a finite set of rules of the form:

$$\begin{aligned}\mathbf{in}(b_0) \vee \ldots \vee \mathbf{in}(b_k) \vee \mathbf{out}(c_0) \vee \ldots \vee \mathbf{out}(c_t) \leftarrow \\ \mathbf{in}(d_0), \ldots, \mathbf{in}(d_m), \\ \mathbf{out}(e_0), \ldots, \mathbf{out}(e_n), \\ \mathbf{was\_in}(f_0), \ldots, \mathbf{was\_in}(f_p), \\ \mathbf{was\_out}(g_0), \ldots, \mathbf{was\_out}(g_q)\end{aligned}$$

where k, l, m, n, p and q could be 0. The intuitive meaning of the rule

$$\mathbf{in}(p) \leftarrow \mathbf{in}(q), \mathbf{out}(s), \mathbf{was\_in}(t), \mathbf{was\_out}(u)$$

with respect to an action a is that if t is true and u is false in the initial state and q is true and s is false in the updated state then p must also be true in the updated state.

For every action a, there is a corresponding state specification P_a, called the *update specification*, that specifies the effects of a together with the preconditions and the executability conditions of the action a.

For example, for the action "shoot", the *update specification* P_{shoot} is given by:

$$P_{shoot} \begin{cases} \mathbf{out}(alive) \leftarrow \mathbf{was\_in}(loaded) \\ \leftarrow \mathbf{was\_out}(has\_gun) \end{cases}$$

The first rules says that if in the initial or current state the gun is loaded then when the action is executed then the "turkey" stops being alive (this is represented by removing alive from the current state). The second rule says that this action cannot be executed if there does not exist a gun in the current state.

But, state specifications of [2] can only represent one-step dynamic constraints. For example, the rule r_6 in D_3^2 can not be directly represented (without introducing changes to the rest of the domain description) using the state specification language of [2]. Hence, we introduce *Temporal state specification* which are extension of state specifications. For any action a, the corresponding temporal state specification P_a will specify the effect of a but with preconditions or executability conditions that can refer to previous executed actions or past states of the domain. *Using Temporal state specification we can not only represent effect of actions that depend on the history, but also represent dynamic constraints.*

In Temporal state specifications we will have three operators **was**, **in** and **out** referring to previous, current an resulting states of an action a. As in [2], for any fluent f, $\mathbf{in}(f)$ ($\mathbf{out}(f)$) intuitively means that f is true (is not true) in the updated state. The operator **was** applies to PTLA formulas. For a PTLA formula P, $\mathbf{was}(P)$ intuitively means that the formula P is true in the "history" or evolution of the domain.

Hence, a *temporal state specification* is a set of rules of the form

$$\begin{aligned} &\mathbf{in}(b_0) \vee \ldots \vee \mathbf{in}(b_k) \vee \mathbf{out}(c_0) \vee \ldots \vee \mathbf{out}(c_l) \leftarrow \\ &\quad \mathbf{in}(d_0), \ldots, \mathbf{in}(d_m), \\ &\quad \mathbf{out}(e_0), \ldots, \mathbf{out}(e_n), \\ &\quad \mathbf{was}(P_0), \ldots, \mathbf{was}(P_q) \end{aligned} \tag{3}$$

where k, l, m, n and q could be 0, b_i c_j, d_r and e_s are fluent literals and $P_0, \ldots, P_q$ are PTLA formulas.

Note that the (non-temporal) states specifications of [2] can be represented in *temporal states specification* by changing every $\mathbf{was\_in}(f)$ by $\mathbf{was}(f)$ and any $\mathbf{was\_out}(f)$ by

$\mathbf{was}(\neg f)$. So in our language we will have that the temporal state specification for P_{shoot} is

$$P'_{shoot} \begin{cases} \mathbf{out}(alive) \leftarrow \mathbf{was}(loaded) \\ \leftarrow \mathbf{was}(\neg has\_gun) \end{cases}$$

The following example illustrates how temporal state specifications can be used to specify temporal constraints.

Example 5.1 *Suppose that Deborah is on vacation visiting London and she does not have her car. She needs to use either public transportation or taxis to visit the different tourist sites. As a policy to save money, Deborah never takes a taxi twice in a row. This situation can be represented in our language by*

$$P_{transport} \begin{cases} \mathbf{in}(bus) \vee \mathbf{in}(taxi) \leftarrow \\ \leftarrow \mathbf{in}(taxi), \mathbf{was}(\neg bus \ \mathbf{since} \ taxi) \end{cases}$$

The constraint in the second rule says that a taxi can not be used ($\mathbf{in}(taxi)$*) if no bus has been used since the last time Deborah used a cab.*

5.1 Semantics of Temporal state Specification

The state specifications described in [2] depend only on two states, the initial state and the updated state. However, *Temporal State Specifications* refer to situations on the past, situations that come one after another when some action is executed. Hence, any interpretation that we give to the rules and constraints has to make reference to the history of the states through the execution of different actions. For this reason our semantics is based on sequences of actions and states.

We will define now, the set of states $Res(a_n, (s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n))$ as the set of states that can be reached by performing an action a_n, given that we know that in the past the following sequence of actions and states $(s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n)$, has occurred, where action a_i is executed in a situation corresponding to state s_i.

Our definition of $Res(a_n, (s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n))$ when effects and constraints are represented by *temporal state specification* is based on translating temporal state specification to extended logic programs. Let us denote $res(a_0, s_0)$ by the sequence $\{a_0\}$, and $res(a_{i+1}, res(a_i, \ldots, (res(a_0, s_0)) \ldots))$ as $\{a_0, \ldots, a_{i+1}\}$

Algorithm 5.2 (Translating Temporal State Specification) *INPUT:*
an action a, a sequence $(s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s)$, and a state specification T.
OUTPUT:
$D_{(s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s), a, T}$ *disjunctive logic program.*
Step1. Initial database

For every $i = 0, 1, \ldots, n-1$ and for any fluent f, if f is true in the state s_i, then the program contains

$$holds(f, \{a_0, \ldots, a_i\}) \leftarrow$$

otherwise the program contains

$$\neg holds(f, \{a_0, \ldots, a_i\}) \leftarrow$$

Step2. Inertia rules

The program contains all the following rules:

$$\begin{aligned}
&holds(F, \{a_0, \ldots, a_{n-1}, a\}) \leftarrow \\
&\qquad holds(F, \{a_0, \ldots, a_{n-1}\}) \\
&\qquad \textbf{not } ab(\neg F, a, \{a_0, \ldots, a_{n-1}\}) \\
&\neg holds(F, \{a_0, \ldots, a_{n-1}, a\}) \leftarrow \\
&\qquad \neg holds(F, \{a_0, \ldots, a_{n-1}\}) \\
&\qquad \textbf{not } ab(\neg F, a, \{a_0, \ldots, a_{n-1}\})
\end{aligned}$$

Step 3. Translating the update rules

Each rule of type 3 in the state specification T is translated to the rule

$$\begin{aligned}
&(holds(b_0, \{a_0, \ldots, a_{n-1}, a\})) \wedge \\
&ab(\overline{b_0}, a, \{a_0, \ldots, a_{n-1}\}) \\
&\qquad \vee \ldots \vee \\
&(holds(b_k, \{a_0, \ldots, a_{n-1}, a\})) \wedge \\
&ab(\overline{b_k}, a, \{a_0, \ldots, a_{n-1}\}) \\
&\qquad \vee \ldots \vee \\
&(\neg holds(c_0, \{a_0, \ldots, a_{n-1}, a\})) \wedge \\
&ab(c_0, a, \{a_0 \ldots, a_{n-1}, a\}) \\
&\qquad \vee \ldots \vee \\
&(\neg holds(c_l, \{a_0, \ldots, a_{n-1}, a\})) \wedge \\
&ab(c_l, a, \{a_0, \ldots, a_{n-1}\}), \\
&\qquad\qquad\qquad \leftarrow \quad holds(d_0, \{a_0, \ldots, a_{n-1}, a\}), \ldots \\
&\qquad\qquad\qquad\qquad holds(d_m, \{a_0, \ldots, a_{n-1}, a\}) \\
&\qquad\qquad\qquad\qquad \neg holds(e_0, \{a_0, \ldots, a_{n-1}, a\}), \ldots \\
&\qquad\qquad\qquad\qquad \neg holds(e_n, \{a_0, \ldots, a_{n-1}, a\}), \\
&\qquad\qquad\qquad\qquad holds(P_0, \{a_0, \ldots, a_{n-1}\}), \ldots, \\
&\qquad\qquad\qquad\qquad holds(P_q, \{a_0, \ldots, a_{n-1}\})
\end{aligned}$$

Step 4. PTLA formulas rules

Finally the program contains the formulas

$$\begin{aligned}
&holds(\textbf{lasttime } a, \{a_0, \ldots, a_i, a\}) \leftarrow . \\
&holds(\textbf{lasttime } R, \{a_0 \ldots, a_i, a_{i+1}\}) \leftarrow \\
&\qquad \neg action(R), holds(R, \{a_0, \ldots, a_i\}). \\
&holds(F \textbf{ since } a, \{a_0, \ldots, a_i, a\}) \leftarrow \\
&\qquad holds(F, \{a_0, \ldots, a_i, a\}). \\
&holds(F \textbf{ since } R, \{a_0, \ldots, a_{i+1}\}) \leftarrow \\
&\qquad \neg action(R), holds(R, \{a_0, \ldots, a_i\}),
\end{aligned}$$

$$holds(F, \{a_0, \ldots, a_{i+1}\}).$$
$$holds(F \textbf{ since } R, \{a_0, \ldots, a_i, a\}) \leftarrow holds(F, \{a_0, \ldots, a_i, a\}), holds(F \textbf{ since } R, \{a_0, \ldots, a_i\}).$$

Note that similar to causal interpretations for domain descriptions in $\mathcal{A}$, we have that a PTLA formula **lasttime** R will be true in $Res(a_n, (s_0, a_0, \ldots, s_{n-1}, a_{n-1} s_n))$, if either $R = a_n$ or if R is a PTLA formula true in $(s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n)$. Also for a PTLA formula F **since** R we have that it will be true in $Res(a_n, (s_0) \ldots, s_{n-1}, a_{n-1}, s_n))$ if either there exists $0 \le i < n$ such that $R = a_i$ and $F \in Res(a_j, (s_0, a_0, \ldots, (s_{j-1}, a_{j-1}, s_j))$, for any $i < j \le n$, or if R is a PTLA formula true in some prefix sequence $(s_0, a_0, \ldots,$ $s_{i-1}, a_{i-1}, s_i)$ with $i < n$, and F is true in $(s_0, \ldots, s_{j-1}, a_{j_1}, s_j$, for j, $i < j \le n$.

Definition 5.3 *Given an action a, a sequence $(s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s)$, and a state specification T. Let $D_{(s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s), a, T}$ be the translation of $s_0, \ldots, s_{n-1}$ and T to logic programs, obtained by 5.2. If M is a consistent answer set of the program $D_{(s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s), a, T}$, then $s' = \{f : holds(f, \{a_0, \ldots, a_{n-1}, a\}) \in M\} \in Res(a, (s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s))$.*

Example 5.4 *Let $s_0 = \{bus\}$ and let the state specification for the action transport be given by $P_{transport}$. then $Res(transport, (s_0)) = \{\{bus\}, \{taxi\}\}$.*

Let s_1 be $\{bus\}$, then $Res(transport, (s_0, transport, s_1)) = \{\{bus\}, \{taxi\}\}$. Let $s_1' = \{taxi\}$, then $Res(transport, (s_0, transport, s_1')) = \{\{bus\}\}$.

6 Elaboration tolerance of temporal state specifications

Let us return to the domain description D_3^2. Now, we can specify r_6 in the language of temporal state specification by the following:

s_6 : **in**(*susceptible_to_pneumonia*)←**was**(¬*dry* **since** *squirt*)
s_7 : **out**(*susceptible_to_pneumonia*)←**was**(¬(¬*dry* **since** *squirt*))

Following is the complete specification of the domain description in D_3^2 in the language of temporal state specifications.

$$D_3^3 \left\{ \begin{array}{l} P_{squirt} \left\{ \; s_1 : \ \textbf{out}(dry) \leftarrow \right. \\ P_{rain} \left\{ \begin{array}{l} s_2 : \ \textbf{out}(dry) \leftarrow \\ s_4 : \ \textbf{in}(pneumonia) \leftarrow \textbf{was}(susceptible\_to\_pneumonia) \end{array} \right. \\ P_{blow\_dry} \left\{ \; s_3 : \ \textbf{in}(dry) \leftarrow \right. \\ P_{help\_patients} \left\{ \; s_5 : \ \textbf{in}(susceptible\_to\_pneumonia) \leftarrow \right. \\ Constraints \left\{ \begin{array}{l} s_6 : \ \textbf{in}(susceptible\_to\_pneumonia) \leftarrow \textbf{was}(\neg dry \textbf{ since } squirt) \\ s_7 : \textbf{out}(susceptible\_to\_pneumonia) \leftarrow \textbf{was}(\neg(\neg dry \textbf{ since } squirt)) \end{array} \right. \end{array} \right.$$

So far, the constraints that we have considered in temporal state specifications were associated with a particular action. This class of constraints can be considered to be qualification constraints, i.e. constraints that could prevent the execution of an action (see Example 5.1).

Constraints s_6 and s_7 above are ramification constraints. These constraints are associated with the whole domain description and apply any time an action is performed. The reader can think that before Algorithm 5.2 is applied to translate a temporal specification T, the (ramification) constraints are added to T. For example, the constrained specification of P_{blow_dry} will be:

$$CP_{blow\_dry} \begin{cases} s_3: \ \mathbf{in}(dry) \leftarrow \\ s_6: \ \mathbf{in}(susceptible\_to\_pneumonia) \leftarrow \mathbf{was}(\neg dry \ \mathbf{since} \ squirt) \\ s_7: \mathbf{out}(susceptible\_to\_pneumonia) \leftarrow \mathbf{was}(\neg(\neg dry \ \mathbf{since} \ squirt)) \end{cases}$$

Notice, however, that constraint s_7 cannot be appended directly to $P_{help\_patients}$. This implies that the representation is not elaboration tolerant. This is because the implication encoded in s_7 is a strict one. What we need for elaboration tolerance is a defeasible implication. (To our knowledge the other papers based on the action description language $\mathcal{A}$ that allow constraints and causal relation between fluents [7, 9, 8] do not currently allow defeasible causal implication.)

To be able to do the above reasoning we need an appropriate representation of the defeasible implication in the constraint, where the consequences of an action take precedence over the consequences of the constraints. One approach is to extend temporal specifications to allow non-monotonic operators (similar to the negation as failure operator in logic programming) in the body. For example the constraint:

$$g \ \mathbf{normally\ influences} f$$

will be translated into:

$$\begin{array}{l} \mathbf{in}(f) \leftarrow \mathbf{in}(g), \mathbf{not\ out}(f) \\ \mathbf{out}(f) \leftarrow \mathbf{out}(g), \mathbf{not\ in}(f) \end{array}$$

The semantics of this extended language is described as before through translation to logic programs, with the added stipulation that the **not** be treated as negation-as-failure.

With the new translation s_6 and s_7 become:

$$\begin{array}{rcl} s_6: \ \mathbf{in}(susceptible\_to\_pneumonia) & \leftarrow & \mathbf{was}(\neg dry \ \mathbf{since} \ squirt), \\ & & \mathbf{not\ out}(susceptible\_to\_pneumonia) \\ s_7: \mathbf{out}(susceptible\_to\_pneumonia) & \leftarrow & \mathbf{was}(\neg(\neg dry \ \mathbf{since} \ squirt)), \\ & & \mathbf{not\ in}(susceptible\_to\_pneumonia) \end{array}$$

Let us consider further evolution of the specification D_3^3. We would now like to incorporate the information that the action of taking a vaccine causes Fred to be not susceptible to pneumonia. This information can be represented by itself by the state specification:

$$P_{take\_vaccine} \left\{ \; s_7 : \; \mathbf{out}(susceptible\_to\_pneumonia) \leftarrow \right.$$

With the current translation of defeasible constraints we can also add this new state specification to our general theory without creating inconsistencies in the description.

7 Final remarks

In this paper we have shown how past temporal logic can be incorporated into two formalisms of reasoning about actions: the language $\mathcal{A}$ [4] and the state specification language in [2]. We have shown how we can express ramifications and qualifications of actions that depend not only in the current state of the world but also on previous actions and states.

In [6] a non-Markovian language to formalize the notion of dependent fluents is described. We believe the language of "Temporal $\mathcal{A}$ " may provide an alternative approach. For example, the relationship between an object being 'on_table' and being 'safe' as formalized in [6] can be expressed using the following temporal state specification.

$\mathbf{in}(safe) \leftarrow \mathbf{in}(on\_table).$

$\mathbf{out}(safe) \leftarrow \mathbf{out}(on\_table), \mathbf{was}(\neg(true \; \mathbf{since} \; (\neg on\_table \wedge safe)))$

The intuitive meaning of the second rule is that removing the object from the table will indirectly cause the object to be unsafe if never in the past the object was safe and not on the table at the same time. The above is only a preliminary observation. We plan to further study the relationship between "Temporal $\mathcal{A}$ " and dependent fluents.

We are also planning to extend the class of temporal formulas used in Temporal $\mathcal{A}$ and temporal state specifications with a more expressive past temporal language.[3] To evaluate conditions in the past sometimes it is required that the reasoning agent has access to a clock. Conditions can depend on when an action took place or how long a particular fluent was true or false. For example, it could be the case that Deborah would like to overwrite her policy of not taking a taxi twice in a row if she has been waiting for a bus for more than half an hour. There are also consequences of an action that may depend on time. Consider two blocks that we are holding together in the air, and the face where they are touching has glue. If the glue dries in one minute the effect of releasing one of the blocks will be different if we do it after 30

[3]The language we use here is a subset of the Past Temporal Logic language described in [3].

second or after 2 minutes. In the first case the block will fall. In the second case the block will stick to the other block.

References

[1] AAAI, Spring Symposium. *Extending Theories of Action: Formal Theory and Practical Applications.* AAAI Press, 1995.

[2] C. Baral. Reasoning about Actions : Non-deterministic effects, Constraints and Qualification. In *IJCAI 95*, pages 2017–2023, 95.

[3] J. Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Transactions on Database Systems.* (to appear).

[4] M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.

[5] M. Georgeff, editor. *Journal of Logic and Computation, Special issue on Action and Processes, vol 4, number 5.* Oxford University Press, October 1994.

[6] E. Giunchiglia and V. Lifschitz. Dependent fluents. In *IJCAI 95*, pages 1964–1969, 95.

[7] G. Kartha and V. Lifschitz. Actions with indirect effects (preliminary report). In *KR 94*, pages 341–350, 1994.

[8] V. Lifschitz. Two components of an action language. In *Common Sense 96*, 1996.

[9] H. Turner. Representing actions in default logic: A situation calculus approach. In *Proceedings of the Symposium in honor of Michael Gelfond's 50th birthday*, 1995.