Reasoning about Actions in Dynamic Linear Time Temporal Logic

Laura Giordano
Dipartimento di Scienze e Tecnologie Avanzate
Università del Piemonte Orientale "A. Avogadro"
Corso Borsalino 54 - 15100 Alessandria, Italy, E-mail: laura@di.unito.it

Alberto Martelli
Dipartimento di Informatica - Università di Torino
C.so Svizzera 185 - 10149 Torino, Italy, E-mail: mrt@di.unito.it

Camilla Schwind
Faculté des Sciences de Luminy
Laboratoire d'Informatique de Marseille, CNRS
16 Avenue de Luminy, Case 901 13288 Marseille Cedex 9, France,
E-mail: schwind@lim.univ.mrs.fr

Abstract

In this paper we present a theory for reasoning about actions which is based on Dynamic Linear Time Temporal Logic (DLTL). DLTL is a simple extension of propositional temporal logic of linear time in which regular programs of propositional dynamic logic can be used for indexing temporal modalities. The action theory we define allows to reason with incomplete initial states, to do postdiction and to deal with ramifications and with nondeterministic actions, which are captured by possibly alternative extensions (temporal models). The expressiveness of temporal logic is exploited to enhance the action language by allowing the definition of general temporal constraints as well as complex actions in the specification of the domain description. We show that the temporal projection problem and the planning problem can be modelled as satisfiability problems in DLTL.

Keywords: Reasoning about actions, Temporal logic.

1 Introduction

In [6] we have proposed an action logic based on a modal language, where modalities represent actions. In that language the ramification problem is tackled by introducing a modal causality operator to represent causal rules and the frame problem is addressed by using a nonmonotonic formalism which maximizes persistency assumptions. Assumptions on the value of fluents in the initial state allow to reason with incomplete initial states, to do postdiction and to deal with nondeterministic actions.

In this paper we want to relate our modal approach to temporal logics. In particular, we want to investigate, on the one hand, whether the modal approach in [6] can be enhanced by moving to the setting of temporal logics and, on the other hand, whether the standard and well known results and techniques developed for temporal logics can be exploited in dealing with reasoning about actions.

We start our investigation by focusing on linear time temporal logics and, in particular, on Dynamic Linear Time Temporal Logic [11]. The motivation for choosing linear time temporal logic is that it has become a well established tool for specifying the behaviour of distribuited systems, for which a rich theory has been developed, and the verification task can be automated.

Dynamic Linear Time Temporal Logic (DLTL) [11] is a simple extension of propositional temporal logic of linear time obtained by strengthening the until modality by indexing it with the regular programs of propositional dynamic logic. Indeed a sublogic of DLTL is Propositional Dynamic Logic (PDL) [10] equipped with a linear time semantics. DLTL has the full expressive power of the monadic second order theory of ω-sequences, and it has an exponential time decision procedure.

The adoption of DLTL as the underlying logic for our action theories provides an extension of the action language in [6] in different respects: it allows general goals, like achievement and maintenance goals to be specified through temporal modalities and, moreover, it provides a simple way of constraining the (possibly infinite) evolutions of the system by making use of regular programs (as in PDL). The need of temporally extended goals has been motivated by Bacchus and Kabanza [1] and by Kabanza et al. [12], who proposed an approach to planning based on a linear temporal logic. The formalization of properties of planning domains as temporal formulas in CTL has also been proposed in [9], where the idea of planning as model checking in a temporal logic has been explored, by formalizing planning domains as semantic models.

Other authors have made use of the μ-calculus for reasoning about actions [20, 5, 4]. The μ-calculus allows complex goals to be formalized in a planning context, including achievement and maintenance goals [20]. Apart from [4] all the proposals mentioned above do not deal with the problem of providing a logical specification of the dynamic of the system in a temporal logic. Rather, they assume that the system is fully specified, for instance, by a state transition function [12] or as a process in a process algebra [5].

As a difference, as it is usual in the litterature of reasoning about actions, in this paper we provide the specification of the dynamic of the system through a domain description. In our case a domain description is a set of temporal formulas which contains laws describing the effects and executability conditions for atomic actions as well as general constraints on the possible executions of the system.

The approach we follow here to define an action theory in the temporal logic DLTL is similar to the one we have proposed in [6]. However, rather than modelling the extensions of a domain description as abductive solutions as in [6], here we make use of a completion construction to deal with the frame problem (essentially, we introduce successor state axioms in this temporal logic setting, in the style of those introduced by Reiter in the situation calculus [19]). In this way we are able to capture the extensions of the domain description as the (temporal) models of a theory in temporal logic, and to formalize the temporal projection problem and the planning problem as satisfiability problems in the temporal logic.

In [4] De Giacomo and Rossati combine the modal μ-calculus with an autoepistemic approach to deal with the problem of reasoning about actions in the presence of incomplete information and sensing actions. As a difference, we do not deal with epistemic knowledge in this paper. On the other hand, we aim at showing that a fairly simple solution to the problems of persistency and ramification can be provided in this temporal logic setting, in such a way that well established results achieved for temporal logic can be used for reasoning about actions.

As the action theory in [6], the action theory we define in this paper allows to reason with incomplete initial states, to do postdiction and to deal with ramifications and with nondeterministic actions, which are captured by possibly alternative extensions (models). Concerning the ramification problem, the solution adopted here is different from the one proposed in [6]. Here, we do not introduce a causality operator, but provide a representation of causal rules which only makes use of the modalities of temporal logics. We show that the expressiveness of temporal logic can be exploited to enhance the action language in [6] by allowing the specification of general temporal queries as well as the definition of general temporal constraints in the specification of the domain description. In particular, the availability of regular expressions in DLTL can be exploited to define conditions on the possible executions of the system (and, essentially, to define complex actions). The possible behaviours of the system can be characterized as the executions of a non-deterministic program π. In this context we can formalize the problem of finding a terminating execution of a program which makes a (goal) formula true. The temporal projection problem and the planning problem can be seen as instances of this problem. We will show that they can be modelled as satisfiability problems in the temporal logic DLTL.

2 Dynamic Linear Time temporal Logic

In this section we shortly define the syntax and semantics of DLTL [11]. In such a linear time temporal logic the next state modatity is indexed by actions. Moreover, (and this is the extension to LTL) the until operator is indexed by programs in Propositional Dynamic Logic (PDL) [10].

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ, where $\omega = \{0, 1, 2, \ldots\}$. Let Σ^∞ $=\Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω and by τ, τ' the words over Σ^*. Moreover, we denote by $\leq$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, we denote by $prf(u)$ the set of finite prefixes of u.

We define the set of programs (regular expressions) $Prg(\Sigma)$ generated by Σ as follows:

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1 ; \pi_2 \mid \pi^{*}$$ [1]

[1] Notice that the test action of dynamic logic cannot occur in a program, nor the language can be easily extended to include it. This is due to the fact that DLTL is a linear time logic and at each state there is only a single future state. However, as we want to use test actions in our action theories, we will model test actions as ordinary actions by introducing suitable precondition laws and action laws for them. We will come back to this in section 3.

where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$. A set of finite words is associated to each program by the mapping $[[]] : Prg(\Sigma) \rightarrow 2^{\Sigma^*}$, which is defined in the standard way.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions. The set of formulas of DLTL(Σ) si defined as follows:

$$\text{DLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \mathcal{U}^{\pi} \beta$$

where $p \in \mathcal{P}$ and α, β range over DLTL(Σ).

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^{\omega}$ and $V : prf(\sigma) \rightarrow 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula α, the satisfiability of a formula α at τ in M, written $M, \tau \models \alpha$, is defined as follows:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \alpha \mathcal{U}^{\pi} \beta$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \leq \tau'' < \tau'$, $M, \tau\tau'' \models \alpha$;

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in prf(\sigma)$ such that $M, \tau \models \alpha$.

The formula $\alpha \mathcal{U}^{\pi} \beta$ is true at τ if $\alpha \mathcal{U} \beta$ is true on a finite stretch of behaviour which is in the linear time behaviour of the program π.

The derived modalities $\langle\pi\rangle$ and $[\pi]$ can be defined as follows:

$\langle\pi\rangle\alpha \equiv \top \mathcal{U}^{\pi} \alpha$
$[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$

It is easy to see that $M, \tau \models \langle\pi\rangle\alpha$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \alpha$. Also, $M, \tau \models [\pi]\alpha$ iff for all $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \alpha$.

Furthermore, if we let $\Sigma = \{a_1, \ldots, a_n\}$, the $\mathcal{U}$, O (next), $\Diamond$ and $\Box$ of LTL can be defined as follows:

$O\alpha \equiv \bigvee_{a \in \Sigma} \langle a \rangle \alpha$
$\alpha \mathcal{U} \beta \equiv \alpha \mathcal{U}^{\Sigma^*} \beta$
$\Diamond\alpha \equiv \top \mathcal{U} \alpha$
$\Box \equiv \neg\Diamond\neg\alpha$,

where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Hence both LTL(Σ) and PDL are fragments of DLTL(Σ). As shown in [11], DLTL(Σ) is strictly more expressive than LTL(Σ). For instance, the formula $[(\Sigma; \Sigma)^*]p$, which says that "p holds at all the even positions" cannot be represented in LTL(Σ).

3 Action Theories

In this section we define our action theory along the line of the one introduced in [6]. There are some relevant differences, however. First, the logic introduced in [6] is a branching time logic rather than a linear time one. The language in [6] deals with the ramification problem (which is concerned with the additional effects produced by an action besides its immediate effects) by providing a modality © to model causal dependencies among fluents. The same solution to the ramification problem cannot be adopted in a linear time setting. Hence, in the following we will adopt a different solution to the ramification problem. As a major difference with the language in [6], in this paper we show that the expressive power of the logic DLTL can be exploited to define general temporal constraints (including program expressions) as part of domain descriptions.

A further difference with [6] is that in this paper we will not make use of an abductive approach to deal with persistency of fluents. Rather, as we aim at modelling the temporal projection problem and the planning problem as satisfiability problems in DLTL, we will make use of a completion construction to define successor state axioms in the style of those proposed by Reiter [19] in the context of the situation calculus.

In our language we use atomic propositions in $\mathcal{P}$ for *fluent names*. A *fluent literal* l is a fluent name f or its negation $\neg f$. Given a fluent literal l, such that $l = f$ or $l = \neg f$, we define $|l| = f$. Moreover, we will denote by *Lit* the set of all fluent literals.

We define a *domain description* D as a tuple $(\Pi, Frame, \mathcal{C})$, where Π is a set of *action laws* and *causal laws*; $\mathcal{C}$ is a set of *constraints*; *Frame* provides a classification of fluents as frame fluents and nonframe fluents in the sense we will define below.

Action laws have the form:

$$\Box(\alpha \to [a]\beta)$$

for every primitive action a with precondition α and effect β: executing action a in a state where α holds causes β to hold[2].

Causal laws have the form:

$$\Box(\textstyle\bigwedge_{a\in\Sigma}([a]\alpha \to [a]\beta)),$$

with the meaning that "for all actions a, if α holds after the execution of a, then β also holds after its execution". Such laws are intended to expresses "causal" dependencies among fluents, and, intuitively, their directionality makes them similar to inference rules: if we are able to derive α then we can conclude β. The usefulness of causality has been widely recognized in the literature [3, 17, 16, 21]. Indeed, directionality of causal rules provides information on *how* dependencies among fluents can be restored when they are possibly violated. The formalization of causal rules in this paper is different from the one in [6] as it does not allow reasoning by cases[3].

Integrity constraints are arbitrary formulas of DLTL(Σ). The kind of constraints that can be expressed are essentially those that have been studied in the verification

[2] Since, we do not put restrictions on the form of α and β action laws may also have the form $\Box(\alpha_1 \wedge [a]\alpha_2 \to [a]\beta)$, which is similar to the form of action laws in the language proposed by Giunchiglia and Lifschitz [7].

[3] In [6] from the causal laws $\Box(p \to ©r)$ and $\Box(q \to ©r)$ and the formula $[a](p \vee q)$ we can conclude $[a]r$. Conversly, from $\Box([a]p \to [a]r)$, $\Box([a]q \to [a]r)$ and $[a](p \vee q)$ we cannot conclude $[a]r$.

litterature, namely safety and liveness constraints. Safety constraints state that something bad must never happen during the execution of an action sequence. Liveness constraints say that something good must eventually happen. Constraints not only put conditions on the value of fluents at the different states, but they also determine which are the possible behaviours of the system.

Typical safety constraints have the form

$$[\pi]\alpha \tag{1}$$

saying that α has to hold in all states that can be reached by any action sequence which is a possible behaviour of the program π. The *domain constraints* of the form **always** α introduced in [13], which enforces the condition α to hold on all possible states, are a special case of (1) above, as we can write them as $\Box\alpha$ (or equivalently $[\Sigma^*]\alpha$).

Precondition laws, which put conditions on the executability of actions in a state, can be formalized, in this linear time temporal logic, by formulas of the form

$$\Box(\alpha \to [a]\bot),$$

meaning that action a cannot be executed in those states in which α holds[4].

It us usual to include in the domain description a set of *observations* about the value of fluents in different states. Observations can be formalized as state constraints of the form $[a_1; \ldots; a_j]\alpha$, meaning that α holds after executing the action sequence $a_1, \ldots, a_j$. In particular, *Obs* may contain observations about fluents in the initial state (α).

Constraints can be used to define the possible behaviours of the system, that is to describe the sequences of actions which are possible for the system. For instance, the liveness constraint

$$\langle\pi\rangle\top$$

is true in a model (σ, V) when there is a prefix τ of σ which is a possible behaviour of program π. It constrains all the executions of the system to start with a sequence which is a possible behaviour of the program π[5].

In general, a constraint may affect both the value of fluents at the states and the behaviour of the system. For instance, $\langle\pi\rangle\alpha$ says that it is possible to execute π and to terminate in a state in which α holds. $\langle\pi_1\rangle[\pi_2]\alpha$ says that it is possible to execute π_1 and to reach a state from which any execution of π_2 leads to a state where α holds.

Frame is a set of pairs (f, a), where $f \in \mathcal{P}$ is a fluent and $a \in \mathcal{A}$ is an action, meaning that f is a frame fluent for action a, that is, f is a fluent to which persistency applies when action a is executed. On the other hand, those fluents which are not frame with respect to a do non persist and may change in a nondeterministic way, when executing a.

[4]Elswere, for instance in [2, 4] precondition laws have been formalized differently, namely by laws of the form $\Box\ (\alpha \to \langle a\rangle\top)$. However, this is not a correct formalization of precondition laws in a linear time logic, in which only a single action can be executed at each state. Our representation is equivalent to the converse formula $\Box\ (\langle a\rangle\top \to \alpha)$.

[5]Notice that, since the logic is linear time, if there are several constraints of the form $\langle\pi_i\rangle\alpha$, in each model (σ, V) they all must be satisfied on the sequence σ. Hence, they do not define alternative executions, but they constrain the same execution.

As in [15, 13] we call *frame* fluents those fluents to which the law of inertia applies. However, as in [6], we consider frame fluents as being dependent on the actions.

It is clear that action laws and causal laws can themselves be regarded as special kinds of constraints. The reason why we have distinguished them from all other constraints is that they deserve a special treatment when dealing with the frame problem. In fact, we assume that frame fluents only change values according to the immediate and indirect effects of actions described by the action laws and causal laws.

Moreover, we assume that for each causal law $\Box(\bigwedge_{a\in\Sigma}([a]\alpha \rightarrow [a]\beta)$ in Π, there is a corresponding constraint formula $\alpha \rightarrow \beta$ in the set $\mathcal{C}$, which assures that also in the initial state if α holds, β also holds.

As we mentioned above, DLTL does not include test actions which, as we will see in the following examples, would be needed to write interesting programs expressions. Infact, test action allow the choice among different actions to be controlled. In the following, we will introduce test actions as atomic actions with suitably defined preconditions and effects. A test action $\phi?$ (where ϕ is a formula) is executable in a state in which ϕ holds and it has no effect on the state. Therefore, we can introduce the following laws which rule the modality $[\phi?]$ (regarded as an atomic action):

$\Box(\neg\phi \rightarrow [\phi?]\bot)$
$\Box(< \phi? > \top \rightarrow (L \leftrightarrow [\phi?]L))$, for all fluent litterals L.

The first law is a precondition law, saying that action $\phi?$ is only executable in a state in which ϕ holds. The second law describes the effects of the action on the state: the execution of the action $\phi?$ leaves the state unchanged. As we will see in the following, the second law has precisely the form of the frame axioms that we will introduce to deal with frame fluents[6]. In the following we will assume that, for all test actions occurring in the examples, the corresponding action laws are implicitly added.

As an example of domain description consider the following formalization of the *russian turkey problem*.

Example 1 *Russian turkey problem.* There is a gun and two actions (*load* and *spin*) of loading and spinning the gun. After loading, the gun is loaded and after spinning it we do not know if it is loaded or not. The action of shooting the turkey makes the turkey dead if the gun is loaded. The action of waiting has no effect. We assume that initially the turkey is alive and the gun is not loaded.

We define a domain description containing the following set of action laws and observations on the initial state (we will assume $\mathcal{A} = \{load,\ wait,\ shoot,\ spin\}$ and $\mathcal{P} = \{loaded,\ alive\}$):

Π: $\Box[load]loaded$ $\quad\quad$ $\Box(loaded \rightarrow [shoot]\neg alive)$
$\Box[shoot]\neg loaded$
$\mathcal{C}$: $alive \wedge \neg loaded$
Frame$= \{(f,a) : a \in \mathcal{A}, f \in \mathcal{F}\} \setminus \{(loaded, spin)\}$.

In this example the set Π only contains action laws, and all fluents are frame fluents except for *loaded* which is nonframe with respect to *spin*. This means that

[6] We can observe that such axioms are entailed by the axiom for test actions in PDL. As a difference with PDL, test actions the execution of test actions, as introduced here, causes the state to change, though the value of propositions in the state is kept unaltered.

the value of fluent *loaded* can change nondeterministically when the action *spin* is executed. Notice that there is no action law for spin.

In the domain description above the set of constraints only contains observation on the initial state. Let us extend the domain description above as follows.

Example 2 Assume that, the hunter does not load the gun until the turkey is in sight, and that, after loading the gun, eventually he executes the actions of spinning and shooting. This can be formalized with the following constraints in $\mathcal{C}$:

$$\neg loaded \mathcal{U} turkey\_in\_sight$$
$$\Box([load]\Diamond < spin; shoot > True$$

We might also assume that the turkey being in sight causes it to be frightened, if it is alive. And that if the turkey is not alive it is not frightened. This can be expressed by adding in Π the following causal laws:

$$\Box(\bigwedge_{a\in\Sigma}([a](turkey\_in\_sight \wedge alive) \rightarrow [a]frightened)$$
$$\Box(\bigwedge_{a\in\Sigma}([a]\neg alive \rightarrow [a]\neg frightened)$$

Finally, we can think that the turkey can suddenly appear after waiting, that is, the value of fluent $turkey\_in\_sight$ can nondeterministically change after $wait$: $Frame = \{(f,a) : a \in \mathcal{A}, f \in \mathcal{F}\} \setminus \{(loaded, spin), (turkey\_in\_sight, wait)\}$. Assume that in the initial state the gun is unloaded and the turkey is alive, not frightened and not in sight. We expect the query "is there a sequence of actions leading to a state where the turkey is not alive?" to succeed (a possible action sequence being $wait, load, spin, shoot$)[7].

Notice that our domain description might even provide more information about the actions which are possibly executed by the hunter. For instance, we might now that Fred loads the gun, waits for a turkey until it appears and, when there is one in sight, he spins the gun and shoots. This behaviour can be described by the following program:

$$\pi = load; ((\neg turkey\_in\_sight)?; wait)^*; (turkey\_in\_sight)?; spin; shoot.$$

The query $\langle\pi\rangle\neg alive$ succeeds if there is a possible execution of the program π which lead to the state in which the turkey is not alive.

Example 3 Let us consider a very simple example of a robot which moves in an environment consisting of three rooms, 1, 2 and 3, each one conected to a corridor. Inside some of the rooms there is a box and the robot, who knows where boxes are, has to pick them up. The robot can be in a room or at one among three positions in the corridor: at(1), in front of room 1, at(2), in front of room 2, or at(3). The fluents are:

$$in\_room(1), in\_room(2), in\_room(3), at(1), at(2), at(3), box\_in(1), box\_in(2),$$
$$box\_in(3), has\_box(1), has\_box(2), has\_box(3).$$

[7]As actions can be nondeterministic, the execution of an action sequence may lead to different states, it may give rise to different extensions. Here, as in [6], we adopt a credolous rather than a skeptical approach and we look for an extension satisfying the query.

The fluent $box\_in(i)$ means that there is a box in room i; the fluent $has\_box(i)$ means that the robot has the box that was in room i. The fluents $in\_room(i)$ and $in\_room(j)$ are mutually exclusive for $i \neq j$, as well as $in\_room(i)$ and $at(j)$, for all i, j.

This can be expressed by the following constraints in $\mathcal{C}$ (where $i, j \in \{1, 2, 3\}$):

$\Box\neg(in\_room(i) \wedge in\_room(j))$, for $i \neq j$
$\Box\neg(in\_room(i) \wedge at(j))$, for all i, j
$\Box\neg(at(i) \wedge at(j))$, for $i \neq j$

The robot can execute the following actions:

goto(i, j) : go from point i to point j in the corridor
getin(i) : get in room i
getout(i): get out of room i
pickup(i): pick up the box in room i

$\mathcal{C}$ also contains the following precondition laws:

$\Box(\neg at(i) \rightarrow [goto(i, j)]\bot)$
$\Box(\neg at(i) \rightarrow [getin(i)]\bot)$
$\Box(\neg in\_room(i) \rightarrow [getout(i)]\bot)$
$\Box(\neg in\_room(i) \vee \neg box\_in(i) \rightarrow [pickup(i)]\bot)$.

The action laws in Π are the following (where $i, j \in \{1, 2, 3\}$ and $i \neq j$):

$\Box[goto(i, j)]at(j)$
$\Box[getin(i)]in\_room(i)$
$\Box[getout(i)]at(i)$
$\Box[pickup(i)](has\_box(i) \wedge \neg box\_in(i))$

The set $Frame$ contains the following pairs (for all i, j, j'): $(at(i), pickup(j))$, $(in\_room(i), pickup(j))$, $(box\_in(i), goto(j, j'))$ $(box\_in(i), getin(j))$ $(box\_in(i), getout(j))$ $(has\_box(i), pickup(j))$ $(has\_box(i), goto(j, j'))$ $(has\_box(i), getin(j))$ $(has\_box(i), getout(j))$.
In this example we are making the assumptions that the robot has complete information about each fluent and about the effects of actions. No exogenous actions can occur. Only the robot can change the state of the world.
The behaviour of the robot can be described by a program expression as follows:

$$\pi = (box\_in(i)?; ((\neg at(i)?; ((in\_room(j)?; getout(j)) + \varepsilon); at(j)?; goto(j, i)) \\ + at(i)?);\ getin(i); pickup(i)))^*$$

Let $\mathcal{C}$ contain the following formulas describing the initial state:

$at(1), \neg at(2), \neg at(3), \neg in\_room(1), \neg in\_room(2),$
$\neg in\_room(3), \neg box\_in(1), box\_in(2), box\_in(3),$
$has\_box(1), \neg has\_box(2), \neg has\_box(3)$

The query $\langle\pi\rangle(has\_box(1) \wedge has\_box(2) \wedge has\_box(3))$ succeeds if there is an execution of the robot program π leading to a state in which the robot has all the three boxes. We expect it to succeed form the given initial state as there is an execution of the program π (omitting test actions) $goto(1,2)$, $getin(2)$, $pickup(2)$, $get\_out(2)$, $goto(2,3)$, $getin(3)$, $pickup(3)$ leading to a state in which the agent has all boxes.

In this language complex temporal queries can be formalized. For instance, we may want to know if there is a possible action sequence which achieves the above goal in such a way that room 2 is not entered before room 3. We can formalize this query as follows: $\langle\pi\rangle(has\_box(1) \wedge has\_box(2) \wedge has\_box(3)) \wedge \neg in\_room(2) \mathcal{U} in\_room(3)$. Notice that the action sequence above is not a solution to this query, since the constraint $\neg in\_room(2) \mathcal{U} in\_room(3)$ is not satisfied.

In general, given a domain description, we formalize the *temporal projection problem* "is it possible to execute a given action sequence $a_1, \ldots, a_n$ to reach a state in which the condition α holds?" by the query $\langle a_1\rangle \ldots \langle a_n\rangle\alpha$. Moreover, we can formalize the *planning problem*: "given an initial state and a condition α, is there a sequence of actions that leads to a state in which α holds?" by the query $\Diamond\alpha$. Both of the problems can then be represented by a query of the form: $\beta \mathcal{U}^{\pi} \alpha$, where $\beta = \top$ and, for the temporal projection problem above $\pi = a_1; \ldots; a_n$, while for the planning problem above $\pi = \Sigma^*$. Notice that, in particular, a query $\langle\pi\rangle\alpha$ allows a planning problem to be formalized in which the wanted action sequence is constrained to be one of the possible executions of π.

In the next section we will see how a planning problem can be modelled as a satisfiability problem in the logic DLTL.

4 Extensions for a domain description

In [6] we have introduced a notion of extension for domain descriptions by making use of assumptions on the persistency of fluents form a state to the next one. The underlying idea is that persistency of frame fluents has to be assumed if it does not lead to an inconsistent state. Persistency laws are regarded as being *defeasible*: they are assumptions to be maximized.

In this paper, rather than adopting this abductive notion of extension, we aim at defining extensions of domain descriptions as (temporal) models of the action theory, so that a planning problem can be defined as a satisfiability problem. To this purpose we define a completion construction which, given a domain description as defined above, introduces frame axioms for all the frame fluents (in the style of the successor state axioms in [19]). The completion construction is applied only to the action laws and causal laws in Π and not to the constraints in $\mathcal{C}$, as we assume that the rules in Π are those that only may produce changes in fluents values, while the formulas in $\mathcal{C}$ are just constraints on the possible models.

Given a domain description $D = (\Pi, Frame, \mathcal{C})$, assume that the consequents of all action laws and causal rules in Π are fluent literals rather than general formulas. In such a case, Π will contain formulas of the form:

$$\Box(\alpha_i \rightarrow [a]f) \qquad \Box(\beta_j \rightarrow [a]\neg f),$$

where the α_i's and the β_i's are arbitrary formulas (and, in particular, they can be formulas of the form $[a]\delta$)[8].

We define the completion of D as the set of formulas $Comp(D)$ containing:

- (i) all the formulas in Π and in $\mathcal{C}$; and

- (ii) for all actions a and fluents f such that $(f, a) \in Frame$, the following axioms:
$$\Box(\langle a\rangle\top \rightarrow ([a]f \leftrightarrow \bigvee_i \alpha_i \vee (f \wedge \neg[a]\neg f))) \tag{2}$$
$$\Box(\langle a\rangle\top \rightarrow ([a]\neg f \leftrightarrow \bigvee_j \beta_j \vee (\neg f \wedge \neg[a]f)).) \tag{3}$$

Notice that, for each action a and fluent f which is nonframe with respect to a, i.e. $(f, a) \notin Frame$, axioms (2) and (3) above are not added in $Comp(D)$.

In the special case in which Π contains (for all actions a and fluents f) action laws and causal rules of the form:

action laws	$\Box(\alpha_i \rightarrow [a]f)$	$\Box(\beta_j \rightarrow [a]\neg f)$
causal laws	$\Box([a]\gamma_k \rightarrow [a]f)$	$\Box([a]\delta_h \rightarrow [a]\neg f)$

with no occurrence of modalities in α_i, β_j, γ_k and δ_h, from the two axioms above we can obtain the following successor state axioms:

$$\Box(\langle a\rangle\top \rightarrow ([a]f \leftrightarrow (\bigvee_i \alpha_i \vee \bigvee_k [a]\gamma_k) \vee (f \wedge \bigwedge_j \neg\beta_j \wedge \bigwedge_h \neg[a]\delta_h)))$$
$$\Box(\langle a\rangle\top \rightarrow ([a]\neg f \leftrightarrow (\bigvee_j \beta_j \vee \bigvee_h [a]\delta_h) \vee (\neg f \wedge \bigwedge_i \neg\alpha_i \wedge \bigwedge_k \neg[a]\gamma_k))).$$

Observe that these axioms are very similar to those that have been introduced by Sheila McIlraith in [18] for providing a closed form solution to the ramification problem in the situation calculus.

We will now define the extensions of a domain description as temporal models of $Comp(D)$.

Definition 4 Let $D = (\Pi, Frame, \mathcal{C})$ be a domain description. An *extension of* D is a model $M = (\sigma, V)$ of DLTL(Σ) such that $M, \varepsilon \models Comp(D)$.

Notice that, in general, a domain description may have more than one extension even for the same action sequence σ: the different models of $Comp(D)$ with the same σ account for the different possible initial states (when the initial state in incompletely specified) as well as for the different possible effects of nondeterministic actions.

In order to give a formalization of the temporal projection problem and of the planning problem, let us define what we mean by a solution to a query in a domain description.

Definition 5 Let $D = (\Pi, Frame, \mathcal{C})$ be a domain description and $\langle\pi\rangle\alpha$ a query. A *solution to the query* $\beta\mathcal{U}^\pi\alpha$ *in* D is a model $M = (\sigma, V)$ of DLTL(Σ) such that $M, \varepsilon \models Comp(D)$ and $M, \varepsilon \models \beta\mathcal{U}^\pi\alpha$.

[8] We are assuming that a causal law $\Box(\bigwedge_{a\in\Sigma}([a]\alpha \rightarrow [a]\beta))$ is represented by the finite conjunction of formulas $\Box([a]\alpha \rightarrow [a]\beta)$, for all $a \in \Sigma$.

Let us now consider the simple robot domain description in Example 3.

Example 6 We want to determine if there is a possible execution of the robot program π (the program describing the behaviour of the robot) terminating in a state in which the conjunction $has\_box(1) \wedge has\_box(2) \wedge has\_box(3)$ holds. The answer can be found as a solution of the query $\langle\pi\rangle(has\_box(1) \wedge has\_box(2) \wedge has\_box(3))$ which succeeds. In fact, there is a model $M = (\sigma, V)$, with σ starting with an execution τ of the program π and such that $has\_box(1) \wedge has\_box(2) \wedge has\_box(3)$ is true in $V(\tau)$. For instance, we can take τ as the action sequence (in which we have omitted the test actions for readability) $goto(1,2)$, $getin(2)$, $pickup(2)$, $get\_out(2)$, $goto(2,3)$, $getin(3)$, $pickup(3)$.

Given a domain description $D = (\Pi, Frame, \mathcal{C})$, the *temporal projection problem* "is it possible to execute a given action sequence $a_1, \ldots, a_n$ to reach a state in which the condition α holds?" can be solved by checking if the query $\langle a_1; \ldots; a_n\rangle\alpha$ has a solution in D, that is, if there is a model satisfying the query and the completion $Comp(D)$ of the domain description.

Example 7 Referring to the Russian turkey problem above (example 1) assume we want to determine if it is possible to execute the action sequence $load, spin, shoot$ so to reach a state in which $alive$ holds. We expect the answer is yes, provided the nondeterministic action $spin$ has the effect of making the gun unloaded. The query $\langle load, spin, shoot\rangle alive$ has a solution. In fact, there is a model $M = (\sigma, V)$ satisfying the domain description, and such that the sequence *load spin shoot* is a prefix of σ and V is defined in such a way that after the (nondeterministic) action of spinning, the gun is unloaded, i.e. $\neg loaded \in V(load\ spin)$. Observe, that the query $\langle load, spin, shoot\rangle alive$ has also a solution, since there is another extension in which the gun remains loaded after spinning.

The *planning problem* "is there a sequence of actions leading to a state where a given formula α holds?" can be solved by checking if the query $\Diamond\alpha$ has a solution in D, that is, if there is a model $M = (\sigma, V)$ satisfying the query and the completion of the domain description $Comp(D)$. Notice that, in such a case, the action sequence (the plan) which makes α true can be extracted from the model M[9].

The definition of extensions given above is less general than the definition of extensions as abductive solutions as proposed in [6], since it assumes a restricted class of theories in which only fluent literals are allowed in the consequent of action laws and causal laws. However, the action language is more general, as it allows several kinds of constraints to be introduced in a domain description, including program expressions which describe complex actions.

We already pointed out that in the formalization of causal rules proposed here reasoning about cases is not allowed, while it is allowed in [6]. As a consequence, it is possible to find a domain description which has an abductive extension, according to the definition in [6], but does not have an extension according to the definition above. Furthermore, it is possible to find a domain description (restricting to the

[9] As nondeterministic actions are allowed in our action theory, the linear plan which is extracted from a model relies on some assumptions on the outcome of nondeterministic actions. In this sense, following the terminology in [9], we can call it a "weak plan".

common language) which has an extension according to the definition above, but does not have an extension according to the definition in [6]. This is due to the fact that here we have adopted a different solution form [6] to deal with the frame problem. More precisely, the adoption of the completion solution above to deal with the frame problem leads to unexpected extensions when action laws and causal laws in Π contain *cyclic dependencies*.

The problem is in essence similar to the one emerging with Lin's approach [16]. As observed by Lin (see [16], pp.1989) the persence of cyclic dependencies has the consequence that a *successor state axiom* may not be computable for some fluent. In [6] an example of domain description has been provided (modelling the behaviour of a flip-flop set-reset) on which Lin's approach gives an unexpected solution (while no unexpected abductive solutions are obtained). The same unexpected solution can be obtained with the formalization above. To avoid this problem, in [18] it is assumed that action laws and causal laws are stratified. We make the same assumption here.

While in the paper we have proposed a theory for reasoning about actions in DLTL and we have mainly focused on the temporal projection problem and on the planning problem, we want to mention that the action theory we have defined is also well suited to verify the correctness of a conditional plan π. In fact, verifying the correctness of π with respect to a condition ϕ in a domain description D amounts to verify that all terminating executions of π in the domain description lead to a state in which ϕ holds. This ammounts to show that, for all models $M = (\sigma, V)$ of DLTL(Σ), if $M, \varepsilon \models Comp(D)$ then $M, \varepsilon \models [\pi]\phi$.

5 Conclusions

In this paper we have proposed an approach for reasoning about actions and change in a temporal logic. Our action theory extends the action theory in [6] with the introduction of complex constraint formulas (including program expressions) and it allows the temporal projection problem and the planning problem to be modelled as a satisfiability problems in Dynamic Linear Time Temporal Logic.

Our action theory addresses both the frame and the ramification problem by introducing successor state axioms which are obtained by a completion construction and which are similar to those introduced by Sheila McIlraith in [18] for dealing with ramifications in the situation calculus. A similar kind of transformation has also been used by Enrico Giunchiglia in [8] to translate action theories to first order logic theory to model planning as satisfiability in first order logic. The action theory we consider in this paper is more general than the one considered in [18] and in [8], as it allows both constraints (which are arbitrary temporal formulas) and queries to include program expressions. Under this respect, the language is related to the Golog language [14], in which complex actions (plans) can be formalized as Algol-like programs. As a difference, here we define complex actions as regular expressions in the style of dynamic logic.

The formalization of the temporal projection problem and the planning problem as a satisfiability problems in a temporal logic opens the possibility of using the tools that have been developed for automating the verification task. In particular, model checking techniques can be adopted to this purpose. As described in [11], the satisfiability problem for DLTL can be solved in deterministic exponential time by constructing for each formula $\alpha \in DLTL(\Sigma)$ a Buchi automaton $\mathcal{B}_\alpha$ such that the language of ω-words accepted by $\mathcal{B}_\alpha$ is non-empty if and only if α is satisfiable. As usual the decision procedure can be used to solve the associated model-checking problem, which, for a linear time temporal logic, is formulated as follows: "Given a program Pr and a temporal formula β determine whether $Pr \models \beta$ (Pr meets the specification α), that is, whether $M, \varepsilon \models \beta$ for all models M induced by a computation of Pr". In the case $Pr \models \beta$ is not true, there is a model M which provides a counterexample, that is a model M such that $M, \varepsilon \models \neg\beta$.

A planning problem, that above we have modelled as the problem of verifying if a query $\Diamond\alpha$ has a solution in a domain description D, can be formalized as a model checking problem as follows: the program Pr, whose computations have to be checked, is provided by the domain description $Comp(D)$, through the Buchi automaton associated with $Comp(D)$; the formula to be checked is $\neg\Diamond\alpha$. If $Pr \models \neg\Diamond\alpha$ does not hold, than any counterexample provides a solution to the planning problem.

A similar idea of using model checking for modelling the planning problem has been developed by de Giacomo and Rossati [4] in the context of a formalism which combines the modal μ-calculus with minimal knowledge modalities. They address the problem of modelling incomplete knowledge and knowledge producing actions (or sensing actions). In [4] frame axioms schemas are introduced for sensing actions, while no special treatment of the frame problem is considered for other actions, nor the ramification problem is addressed. They deal with a branching time structure and with deterministic atomic actions. In that context, the temporal projection problem and the planning problem are modelled by verifying that a query formula is true in all models of the domain description (where a model is defined as a maximal set of interpretations). An algorithm is introduced to compute a transition graph from an action specification, which can be used for verifying properties of the possible executions through model checking.

References

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. in *Annals of Mathematics and AI*, 22:5–27, 1998.

[2] M. Baldoni, L. Giordano, A. Martelli and V. Patti. Modal programming language for representing complex actions. In *Proc. DYNAMICS'98: Transactions and Change in Logic Databases. Technical Report MPI-9808*, pages 1–15, 1998.

[3] G. Brewka and J. Hertzberg. How to do things with worlds: on formalizing action and plans. In *J. Logic and Computation*, vol.3, no.5, pages 517–532, 1993.

[4] G. De Giacomo and R. Rosati. Minimal knowledge approach to reasoning about actions and sensing. In *Proc. of NRAC'99*, Stockholm, Sweden, August 1999.

[5] G. De Giacomo and X.J.Chen. Reasoning about nondeterministic and concurrent actions: A process algebra approach. AIJ 107:63-98,1999.

[6] L. Giordano, A. Martelli, and Camilla Schwind. Ramification and causality in a modal action logic. *Journal of Logic and Computation*, 1999. to appear.

[7] E. Giunchiglia, G.C. Kartha and V. Lifschitz. Representing Actions: Indeterminacy and Ramifications. In *Artificial Intelligence*, vol. 95, N. 2, 409–438, 1997.

[8] E. Giunchiglia. Planing as satisfiability with expressive action languages: Concurrency, Constraints and Nondeterminism. In Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'00). Breckenridge, Colorado, USA 12-15 April 2000.

[9] F. Giunchiglia and P. Traverso. Planning as Model Checking. In Proc. The 5th European Conf. on Planning (ECP'99), pp.1–20, Durham (UK), 1999.

[10] D. Harel. First order dynamic logic in *Extensions of Classical Logic, Handbook of Philosophical Logic II*, pp. 497–604, 1984.

[11] J.G. Henriksen and P.S. Thiagarajan Dynamic Linear Time Temporal Logic. in *Annals of Pure and Applied logic*, vol.96, n.1-3, pp.187–207, 1999

[12] F. Kabanza, M. Barbeau and R.St-Denis Planning control rules for reactive agents. in *Artificial Intelligence*, 95(1997) 67-113.

[13] G.N. Kartha and V. Lifschitz. Actions with Indirect Effects (Preliminary Report). In *Proc. KR'94* , pages 341–350, 1994.

[14] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *J. of Logic Prog.*, 31, 1997.

[15] V. Lifschitz. Frames in the Space of Situations. *Artificial Intelligence Journal*, Vol. 46, pages 365–376, 1990.

[16] F. Lin. Embracing Causality in Specifying the Indirect Effects of Actions. In *Proc. IJCAI'95*, pages 1985–1991, 1995.

[17] N. McCain and H. Turner. A Causal Theory of Ramifications and Qualifications. In *Proc. IJCAI'95*, pages 1978–1984, 1995.

[18] S. Mc Ilraith Representing Actions and State Constraints in Model-Based Diagnosis. AAAI'97, pp. 43–49, 1997.

[19] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of Jhon McCarthy*, V. Lifschitz, ed.,pages 359–380, Academic Press, 1991.

[20] Munindar P. Singh. Applying the Mu-Calculus in Planning and Reasoning about Actions. in *Journal od Logic and Computation*, 1998.

[21] M. Thielscher. Ramification and Causality. Artificial Intelligence Journal, vol. 89, No. 1-2, pp. 317-364, 1997.