Using Linear Temporal Logic to Model and Solve Planning Problems

Serenella Cerrito[1] and Marta Cialdea Mayer[2]*

[1] Université de Paris-Sud, LRI, Bât 490,
F-91405 Orsay Cedex, France.
e-mail: serena@lri.fr

[2] Dipartimento di Informatica e Automazione, Università di Roma Tre,
via Vasca Navale 79, 00146 Roma, Italia.
e-mail: cialdea@dia.uniroma3.it

Abstract. In this work we investigate the use of propositional linear temporal logic LTL as a specification language for planning problems and the use of analytic tableaux as a tool for plan search, following the "planning as satisfiability" approach [11]. We claim that LTL can be a good specification language for planning problems, because of its rich expressive power and the underlying simple model of time and actions. We propose the use of Tabplan, a tableau calculus for bounded model search in LTL (fully described in [7]), as a system for plan synthesis. We show how to code a given planning problem by means of different LTL theories, each encoding making the model construction procedure simulate a different search strategy, namely planning by progression and partial order regression planning in the style of [14].

1 Introduction

Recently, increasing attention has been devoted to the problem of the expressive power of planning languages. In order to address more ambitious tasks, some attempt has been done to use formalisms different from classical STRIPS-like ones. For example, [9, 15, 8] address in different ways the same problem: endowing a planner of powerful modelling capabilities. The motivation behind this work is the design of a logical language to model planning problems and an associated calculus for plan construction, integrating the declarativity, expressiveness and flexibility typical of the logical languages with the capability of embedding different search strategies and techniques. Importance is given to the fact that the language is based on a natural model of time and actions and the calculus allows the user to follow and understand its execution, so that it is suited to be the base for mixed-initiative systems.

We investigate the use of propositional linear temporal logic (LTL, that is based on a discrete, linear model of time) as a specification language for planning

* This work has been partially supported by MURST, ASI (Agenzia Spaziale Italiana) and CNR (SCIxSIA Project).

problems, keeping inside the "planning as satisfiability" approach [11]: a plan corresponds to a finite model of the problem specification. The semantics of the modal temporal language used to represent planning problems is based on finite temporal structures and the execution mechanism is based upon model construction techniques related to analytic tableaux. The tableau system we propose is fully described in [7], where it is introduced as a system for finite model search in LTL, suggesting its possible application to planning. The system has been given a prototype implementation, TabPlan, that is presently being tested. The calculus is based on the use of prefixes that, intuitively, label formulae as true in a particular time point or interval. Time points are partially ordered, according to a set of temporal constraints that is built up step by step as the construction of a tableau branch proceeds. The tableau construction tests whether a set of LTL formulae in negation normal form admits models whose underlying temporal structure does not exceed a given size and returns one if such exists. Hence, it can be used to solve *bounded* planning problems, whose solution must not exceed a given length k, for some fixed k. [1]

The main advantage of the use of LTL as a planning language derives from its rich expressive power and the underlying simple model of time and actions. LTL can easily been used to express domain restrictions in the style of [8] as well as domain knowledge useful to guide the search, as shown by [1]. In this work we show how to represent action centered planning problems in the style of STRIPS like languages, with the addition of intermediate tasks to be accomplished. Planning problems can be represented in different forms, each encoding making the model construction procedure simulate a different search strategy, namely planning by progression and partial order planning in the style of [14]. Although the objection that tableau calculi are in principle far less efficient than other model search methods might be raised, the study of suitable control strategies and techniques can lead to obtain usable systems, still keeping the benefits deriving from the fact that tableau construction procedures mirror a natural and understandable way of reasoning.

2 Bounded Model Construction in LTL

We consider the full language of propositional linear temporal logic. Table 1 is a sketchy description of the basic temporal operators and some definable ones we shall use in the sequel.

The semantics of the language is restricted to finite models: a *temporal structure* T is a finite initial segment of the natural numbers: $\langle 0, ..., k\rangle$; its elements are called *time points*. If $\mathcal{L}$ is an LTL temporal language and P the set of propositional letters in $\mathcal{L}$, an *$\mathcal{L}$-interpretation* $\mathcal{M}$ is a pair $\langle T, \sigma\rangle$, where:

- T is a temporal structure;
- $\sigma : T \rightarrow \mathcal{P}(P)$ is a function on time points, i.e. if $i \in T$, then $\sigma(i) \subseteq P$ is the set of propositional letters true at i.

[1] It can also, in principle, decide whether a planning problem has any solution at all.

$\Box A$	A will always be true (but it is not necessarily true now)
$\bar{\Box} A$	A is true now and will always be true
$\bar{\Diamond} A$	A is either true now or sometime in the future
$\bigcirc A$	If there exists a next point, then A is true there
$A \mathcal{U} B$	B will be true sometime in the future and A holds from now till then (the present point and the point where B is true are not included)
$A \mathcal{W} B$	Either A will always be true, or it will be true until B occurs
$\overleftarrow{\Box} A$	A has always been true in the past
$A \mathcal{S} B$	B occurred sometime in the past and A has been true since then (the present point and the point where B was true are not included)
$A \mathcal{Z} B$	Either A has always been true or it has been true since B occurred

Table 1. The meaning of some temporal operators

The satisfiability relation $\mathcal{M}_i \models A$, for $i \in T$, is defined as usual (the detailed definition can be found in [7]; it differs from some traditional approaches because of the strong semantics for the future time operators, excluding the present time point). Truth is satisfiability in the initial state: a formula A is true in $\mathcal{M}$ (and $\mathcal{M}$ is a model of A) iff $\mathcal{M}_0 \models A$. Truth of sets of formulae is defined as usual. A is valid in $\mathcal{M}$ ($\mathcal{M} \models A$) iff for every time point $i \in T$, $\mathcal{M}_i \models A$. Two formulae A and B are equivalent iff $\mathcal{M} \models A \equiv B$ for all $\mathcal{M}$.

A formula is in *negation normal form* (nnf) iff no logical operator is in the scope of a negation. It can easily be shown that every formula can be transformed into an equivalent formula in nnf.

In the rest of this section the tableau system introduced in [7] is briefly described, that allows one to test whether a set of LTL formulae in nnf admits models whose underlying temporal structure does not exceed a given size.[2] In case of a positive answer, a model can be easily extracted from the tableau construction. A tableau is a tree whose nodes are labelled either by expressions of the form $[s,t]A$ (logical nodes) or by temporal constraints $s \leq t$, where A is a formula in nnf and s,t are *states*, whose form is $c+n$, where c is a time constant and n an integer. States intend to denote time points, $[s,t]A$ means that A is true at all time points in the interval between s and t (included), $s \leq t$ is a constraint on the relative order between s and t. $[s,s]A$ is abbreviated by $[s]A$.

Among the time constants, *start* and *finish* denote, respectively, the initial and final time point of the searched models. If S is a finite set of formulae in nnf and $K = \{finish \leq start + k\}$ for some integer $k \geq 0$ (representing the maximal size of the searched models), then tableaux for $S \cup K$ are initialized with the set $\{[start]A \mid A \in S\} \cup K$ (i.e. such nodes are the first ones created) and expanded by application of expansion rules: new nodes are added (below the line, in the rule figures), as "expansions" of existing nodes (above the line), according to the rules. Some of such rules are shown in Table 2, where c,d denote elements of the

[2] The restriction to nnf allows for a more compact description of the system.

Logical rules			
α-rule	**β-rule**	**simple β-rule**	**$\Box$-rule**
$\dfrac{\begin{array}{c}[s,t]\,A\wedge B\\ s\leq t\end{array}}{\begin{array}{c}[s,t]\,A\\ [s,t]\,B\end{array}}$	$\dfrac{\begin{array}{c}[s,t]\,A\vee B\\ s\leq t\end{array}}{\begin{array}{cc}[s,t]\,A & s\leq c,\ c\leq t\\ & [s,c-1]\,A\\ & [c]\,B\\ & [c+1,t]\,A\vee B\\ & c\in C\ \text{fresh}\end{array}}$	$\dfrac{[c]\,A\vee B}{[c]\,A\quad [c]\,B}$	$\dfrac{\begin{array}{c}[s,t]\,\Box A\\ s\leq t\end{array}}{[s+1,finish]\,A}$

$\bar{\Diamond}$-rule	**simple $\mathcal{W}$-rule**	**simple $\mathcal{Z}$-rule**
$\dfrac{\begin{array}{c}[s,t]\,\bar{\Diamond}A\\ s\leq t\end{array}}{\begin{array}{c}[c]\,A\\ t\leq c\\ c\in C\ fresh\end{array}}$	$\dfrac{[c]\,A\mathcal{W}B}{\begin{array}{cc}[c+1,finish]\,A & [d]\,B\\ & c+1\leq d\\ & [c+1,d-1]\,A\\ & d\in C\ fresh\end{array}}$	$\dfrac{[c]\,A\mathcal{Z}B}{\begin{array}{cc}[start,c-1]\,A & [d]\,B\\ & d\leq c-1\\ & [d+1,c-1]\,A\\ & d\in C\ fresh\end{array}}$

Interval rule	**Conflict resolution rule**
$\dfrac{[s,t]\,A}{t\leq s-1\qquad s\leq t}$	$\dfrac{\begin{array}{c}s\leq t\\ s'\leq t'\\ [s,t]\,p\\ [s',t']\,\neg p\end{array}}{t\leq s'-1\qquad t'\leq s-1}$

Table 2. Some of the tableau expansion rules

set C of time constants, and $s, t, s', t', ..$ are states. Some rules (the β-rules, for instance) create a branching in the tree. The "simple" β-rule is just a shortcut of the general β-rule, handling the particular case where the time interval consists of a single time point. The simple $\mathcal{W}$ and $\mathcal{Z}$ rules are similar shortcuts for the corresponding general rules.

A tableau branch is *closed* iff the set of its temporal constraints has no finite solution over the integers (TabPlan incrementally checks branch closure by use of Bellman-Ford shortest path algorithm). A branch is called *canonical* iff (intuitively) no wasteful expansions are ever performed. In particular, a closed canonical branch is never expanded. A branch $\mathcal{B}$ is *complete* iff there exists no canonical expansion of $\mathcal{B}$.

It can be shown that the construction of any canonical tableau terminates and that the calculus is complete and sound, i.e., if a given set of formulae S has a model $\mathcal{M}$ whose size is not greater than k, then any tableau for $S \cup \{finish \leq start + k\}$ has an open branch that is satisfiable in $\mathcal{M}$ (completeness) and any complete and open branch, in any tableau for $S \cup \{finish \leq start + k\}$, is satisfiable in some model whose size is bounded by k (soundness). Complete proofs can be found in [6].

Since the rules of the calculus are invertible, the search for a model can be done by construction of a single (canonical) tableau, following any strategy to choose the nodes to be expanded. It can be shown that the length of a branch in a tableau for $S \cup \{finish \leq start + k\}$ is polynomial in $n \times k$, where n is the size of S. Since testing whether a branch is open or not can be done in polynomial time, a non deterministic algorithm searching for models via the proposed tableau system has polynomial time complexity in $n \times k$. From a theoretical standpoint, the calculus can be used to check satisfiability in models of any finite size, but it is clearly highly unpractical.

The calculus is significantly different from other systems for linear temporal logic, for example [16, 5, 3], that are essentially based on the equivalences $\Diamond A = A \vee \bigcirc\Diamond A$ and $A\mathcal{U}B = B \vee (A \wedge \bigcirc(A\mathcal{U}B))$, where $\Diamond$ and $\mathcal{U}$ have the weak semantics, including the present time point. Rewriting a formula of the form $\Diamond A$ as $A \vee \bigcirc\Diamond A$ leads to choose whether "executing" A at the present time point or postponing its execution, the same problem passing on to the next time point. Consequently, in the model description that results from the construction, the ordering of time points is always total and the model is built up in a point-to-point manner: the initial state is completely expanded by means of the classical rules, its expansion (non deterministically) describing which formulae hold in the next state; such a next state is expanded, and so on. Moreover, such systems must be equipped with some mechanism to check whether all "eventualities" (formulae in the scope of a $\Diamond$ or $\overleftarrow{\Diamond}$, or in the right scope of a $\mathcal{U}$ or $\mathcal{S}$ operator) are sooner or later satisfied.

On the contrary, the proposed system (that bears strong similarities with [13], where however only the unary temporal operators are considered) immediately "executes" an existential temporal formula (for example $\bar{\Diamond}A$ or $B\mathcal{W}A$), by addition of a fresh constant (time point) $c \in C$ where A holds, and suitable temporal constraints over c. The conflict resolution rule introduces new temporal constraints in order to make those intervals, where contradictory literals hold, disjoint.

3 Plans and Temporal Models

We consider planning problems with deterministic actions that may have conditional effects. Problems may require intermediate tasks to be accomplished as well as final goals to reach. Moreover, actions are not necessarily mutually exclusive, the contemporary or parallel execution of two or more actions being allowed. This requires that the problem specifications account for the compatibility/incompatibility relation on actions.

We assume that the elements of a problem description are built out of ground atoms (*fluents*) in a language $\mathcal{L}_\Pi$, i.e. $\mathcal{L}_\Pi$ is essentially propositional, although we shall use a first order syntax in order to enhance readibility. Among the ground terms of the language, the distinguished set *Actions* contains terms denoting actions. We shall use the unary predicate symbol *do*, that applies to terms in *Action*: $do(a)$ means that action a is performed. Such expressions allow the description of intermediate tasks (e.g. $at(Rome) \wedge do(eat)$), and are assumed to occur only in the encoding of tasks.

The $\mathcal{L}_\Pi$-specification of a planning problem Π consists of a complete specification of the initial state, S_0, a formula *Goal* specifying the final goals to be achieved, a set of intermediate tasks to be accomplished, each of them having the form $do(a) \wedge C$, where C is any $\mathcal{L}_\Pi$-formula, and a description of the available actions, completely specifying the preconditions and effects of each action and the compatibility/incompatibility relation on actions: if $a \in A$, π_a denotes the *precondition formula for* a; the *effect specification for* a consists of a set of pairs of either forms $C \Rightarrow R$ or $C \Rightarrow \neg R$, where C is an $\mathcal{L}_\Pi$-formula and R a fluent, specifying that the execution of the action a leads R to become true (resp. false), whenever C holds; and the *incompatibility of* a is a set of ground terms in *Action*, denoting actions that are incompatible with a.

A *partial plan* P solving a planning problem Π is a sequence of sets of actions $A_0, ..., A_n$ such that the execution of all the actions in A_0 (either in parallel or in any order), followed by the execution of the plan $A_1, ..., A_n$ leads from the initial state 0 to a state $n+1$ where all the final goals are achieved. Note that since some A_i may be empty, there is no need to introduce the special action "wait", with no effects.

In order to characterize plans in terms of temporal models, we use the temporal language $\mathcal{T}_\Pi$, that extends $\mathcal{L}_\Pi$ by means of the temporal operators. The interpretations of $\mathcal{T}_\Pi$ are candidates for representing plans solving Π. The notion of a temporal interpretation satisfying all the constraints of a planning problem, thus representing a plan that solves the problem, is quite straightforward (basically, the initial and final state of the model satisfy S_0 and *Goal*, respectively; for each task τ, there exists a time point satisfying τ; and classical effect and frame conditions for actions are satisfied by any time point). A plan can be extracted from a temporal interpretation as follows: an interpretation $\mathcal{M} = \langle T, \sigma \rangle$ of $\mathcal{T}_\Pi$ determines the plan $P = \langle A_0, ..., A_{n-1} \rangle$ iff $T = \langle 0, ..., n \rangle$ and for every $i = 0, ..., n-1$ and action a, $\mathcal{M}_i \models do(a)$ iff $a \in A_i$. Since we assume that the initial state description is complete, it can be shown that $P = \langle A_0, ..., A_n \rangle$ is a plan solving the planning problem Π iff there exists an interpretation $\mathcal{M}$ for $\mathcal{T}_\Pi$, satisfying all the constraints of Π, that determines P. Hence, plans actually solving a problem can be characterized in terms of temporal models.

Finally, a set of LTL formulae S is a correct and complete encoding of a planning problem Π iff for every temporal interpretation $\mathcal{M}$ of $\mathcal{T}_\Pi$, $\mathcal{M}$ is a model of S if and only if $\mathcal{M}$ satisfies the constraints of Π. As a consequence, any model of a correct and complete encoding S of a planning problem Π determines a plan solving Π and, conversely, every plan solving Π is represented by some model of S.

4 Representation of Planning Problems

In this section different encodings of planning problems into LTL are presented, determining different search strategies to be induced by the tableau construction. Plans can be extracted from open and complete tableau branches for the problem encoding. An extraction function ε can be defined, that applies to tableau branches (that are partial descriptions of temporal models) and returns the description of a partial plan: such a description consists of the set of the temporal constraints and all the nodes of the form $[c]do(a)$ occurring in the branch. A plan $P = \langle A_1, ..., A_n \rangle$ is an instance of $\varepsilon(\mathcal{B})$ if there exists a solution $\mathcal{I} : C \to Z$ of the constraints in $\mathcal{B}$ (hence a linearization of the partial order described by $\mathcal{B}$) such that $a \in A_i$ iff $[c]do(a) \in \mathcal{B}$ and $\mathcal{I}(c) = i$. The extraction function clearly enjoys the following property: if a branch $\mathcal{B}$ is satisfiable in $\mathcal{M}$, then there exists a plan P that is an instance of $\varepsilon(\mathcal{B})$ such that $\mathcal{M}$ determines P.

All the proposed encodings share some axioms: the specification of the initial state, goal and tasks, action precondition axioms and the encoding of incompatibilities between actions. Precisely, the basic encoding of the planning problem Π, $Basic(\Pi)$, consists of:

1. The specification of the initial state S_0.
2. The formula $\bar{\Diamond}(Goal \wedge \Box\bot)$: "at the last time point (either now or sometime in the future) $Goal$ is true".
3. For every $\tau \in Tasks$, the formula $\bar{\Diamond}\tau$: "τ holds either now or sometime in the future".
4. For every action a, the formula $\bar{\Box}(do(a) \to \pi_a)$: "at any time, a is performed only if its preconditions hold" (action precondition axioms).
5. For every action a, the formula: $\bar{\Box}(do(a) \to \neg do(a_1) \wedge \cdots \wedge \neg do(a_n))$, where $a_1, ..., a_n$ are all the incompatibilities of a: "at any time, a is performed only if no contemporary execution of an incompatible action is performed" (action incompatibility axioms).

Note that if two actions a and b are incompatible because they have conflicting preconditions or effects, this need not be represented explicitly. In fact, no model of a complete encoding can have $do(a)$ and $do(b)$ true at the same time point. But if the reason of the incompatibility is different, it must be explicitly coded. In particular, this is the case if a deletes a precondition of b (or *vice versa*). In fact, if $\mathcal{M}_i \models do(a) \wedge do(b)$, then the preconditions of the actions must be true at i, while their effects are true at $i+1$: the conflict cannot be determined by logic alone.

In the rest of this section we shall further make use of the following definition, that borrows the notation from [12]. Let R be a fluent and $\mathcal{A}$ the set of actions. Then:

$$\Gamma_R^+ = \{do(a) \land C \mid a \in \mathcal{A} \text{ and } C \Rightarrow R \text{ is in the effect specification of } a\}$$

$$\Gamma_R^- = \{do(a) \land C \mid a \in \mathcal{A} \text{ and } C \Rightarrow \neg R \text{ is in the effect specification of } a\}$$

If Γ_R^+ (resp. Γ_R^-) are not empty, then γ_R^+ (resp. γ_R^-) is the disjunction of all the formulae in Γ_R^+ (resp. γ_R^-), otherwise $\gamma_R^+ = \bot$ and $\gamma_R^- = \bot$. γ_R^+ and γ_R^- represent the *fluent positive and negative preconditions*: they specify all the conditions that can lead to change the truth value of R (*completeness assumption*). We assume here that γ_R^+ and γ_R^- can never hold contemporarily.

The encodings presented in the rest of this section are distinguished into two classes: the progression and regression encodings. Progression encodings induce a form of data driven search: new time points at which actions are executed are introduced in the model, under the guide of what holds in the previous points, and the time points that follow are determined as a consequence of such actions. Regression encodings induce a form of goal driven search: the expansion of the goal node $[start]\bar{\Diamond}(Goal \land \Box\bot)$ introduces in the branch a time point c_f, such that $c_f = finish$ is implied by the temporal constraints, and the node $[c_f]Goal$. Such a final node serves as a guide to decide which actions have to be inserted in the model, according to a backward reasoning of the form: in order to have A true, it must be the case that previously B has been true (note, however, that the goal obviously affects also the case of progression encodings, and the initial state may be taken into accout in the backward search).

The different encodings are characterized by two axioms for each fluent R, a positive and a negative one. We shall explicitly state only the positive ones, the corresponding negative forms being obtained by replacing R by $\neg R$, γ_R^+ by γ_R^- and γ_R^- by γ_R^+. Each encoding can obviously be enriched by addition of other axioms describing any kind of constraints on the planning domain. Here, we restrict the description to the kernel theories.

4.1 Progression Encodings

The *linear progression encoding with explanatory frame axioms*, $Lin_{prog}(\Pi)$, is the LTL counterpart of the corresponding encoding described in [10], except for the fact that there is no commitment on the number of actions executed at each time point. It extends $Basic(\Pi)$ with the following effect-and-frame axioms (and their negative counterparts), for every fluent R:

$$\bar{\Box}(\gamma_R^+ \lor (R \land \neg\gamma_R^-) \rightarrow \bigcirc R)$$

"At any time, if either some action having R as effect is accomplished, or R holds and no action destroying R is made, then R holds in the next time point (if any)".

The encoding $Lin_{prog}(\Pi)$, being directly related to the specification of the planning problem Π, can easily be shown to be a correct and complete specification of Π. The models obtained from such linear encodings are linear plans, built up by a step-by-step forward mechanism: the effect-and-frame axioms lead to decide, from *start* onwards, which actions are performed and, consequently, what holds in the next time point, and so on till the model (including the goal state) is completely constructed. Hence, $Lin_{prog}(\Pi)$ makes TabPlan simulate a sort of linear progression planning.

The expressive power of temporal logic can be exploited to give progression encodings a different form. In fact, Lin_{prog} is equivalent to the extension of $Basic(\Pi)$ with the following axioms and the corresponding negative forms, for every fluent R:

$$\bar{\Box}[\gamma_R^+ \vee (R \wedge \neg\gamma_R^-) \rightarrow R\mathcal{W}(R \wedge \gamma_R^-)]$$

"At any time, if either some action having R as effect is accomplished, or R holds and no action destroying R is made, then R will hold until (possibly) some action destroys it".

The use of the temporal operator $\mathcal{W}$ allows the axiom to refer to a whole time interval where R holds, instead of the single next time point. The above axioms act in a very similar way with respect to decisions about which actions are to be performed and when. But their role as effect and frame axioms is somewhat different: whenever a node of the form $[c]\gamma_R^+$ is present on the branch, a whole interval is immediately forced to hold R: either $[c+1, finish]$ or $[c+1, d]$, where d is a time point where γ_R^- holds. I.e. R is propagated in one step to the whole interval where it holds.

4.2 Partial-Order Regression Encodings

The main novelty of TabPlan with respect to most systems for LTL is that the search for models may keep only a partial order of time points. This fact, together with the high expressive power of the binary temporal operators, makes it possible to encode planning problems in such a way that plan space search, in the style of [14], is simulated. Namely, partial order regression planning is obtained by means of the regression encoding $Part_{reg}(\Pi)$ of a planning problem Π, that extends $Basic(\Pi)$ with the following axioms for every fluent R (positive regression axiom), and the corresponding negative ones:

$$\Box[R \rightarrow (\neg\gamma_R^- \wedge R)\mathcal{Z}\gamma_R^+]$$

"If R holds at any time point, then either R has always been true and no action destroying it has ever been performed, or sometime in the past an action yelding R has been made and R has never been destroyed since then".

It can be shown that for any model $\mathcal{M}$, $\mathcal{M} \models Lin_{prog}(\Pi)$ iff $\mathcal{M} \models Part_{reg}(\Pi)$ – assuming that $\mathcal{M} \models \bar{\Box}(\neg\gamma_R^+ \vee \neg\gamma_R^-)$. Hence, $Part_{reg}(\Pi)$ is a correct and complete encoding of Π.

The search in the tableau resulting from the expansion of the regression axiom for a fluent R is goal driven. Let us assume, for example, that $[c]R$ is in the branch (R is a goal holding at c), with $c > start$, and that $[start]\neg R$ is in the branch. Then the leftmost expansion of the positive axiom is a closed branch (it contains $[start+1, finish]\neg R$), hence the other subtree is explored. Here, new time points e (with $start+1 \leq e \leq finish$) and $d < e$ are introduced, and nodes $[e](\neg\gamma_R^- \wedge R)\mathcal{Z}\gamma_R^+$, $[start+1, e-1]\neg R$, $[d]\gamma_R^+$ and $[d+1, e-1]\neg\gamma_R^- \wedge R$: at time point d some action producing R is performed, with the constraint that the effect R is not destroyed since then to $e-1$. The conflict resolution rule, applied to $[c]R$ and $[start+1, e-1]\neg R$ yelds $e \leq c$, hence $d < c$ (the other branch is closed).

In general, in the description of a model that results by executing TabPlan on a such regression encodings, time points may be only partially ordered. However, in many cases, since the first model output by a depth first search is the smallest one (the leftmost expansion of the interval rule forcing intervals to be empty), the first result is in fact a totally ordered set of time points. Obviously, whenever the tableau branch contains nodes $[c]do(a)$ and $[s,t]\neg do(a)$, then the conflict resolution rule decides whether $c < s$ or $t < c$ (this mechanism is similar to the resolution of *threats* in [14]). The same happens with $[c]do(a)$ and $[d]do(b)$ if a and b are incompatible. In such cases, in fact, the two time points necessarily have to be distinguished.

We conclude this section observing that the plan description that is output by the search contains enough information to distinguish actions that *must* be performed simultaneusly, in order to obtain a given effect, from actions that *may* be performed contemporarily but also sequentialized in any order. In fact, in any tableau branch for any of the considered encodings:

- every node of the form $[s,t]\gamma_R^+$ is such that $s = t$ is a constant;
- there are no pairs of nodes of the form $[c]\gamma_R^+$ and $[c]\gamma_Q^+$ for distinct R and Q, labeled by the same constant;
- every node of the form $[s,t]do(a)$ has $s = t = c$ for some constant c and it derives from the expansion of a node $[c]\gamma_R^+$;

Hence, if $[c]do(a)$ and $[c]do(b)$ occur in a branch for distinct a and b, this is because the problem specification explicitly requires a and b to be executed contemporarily in order to obtain a given effect. Conversely, any pair of actions that may, possibly but not necessarily, be executed contemporarily (or in parallel) occur in the branch with different constant labels. Thus, the calculus allows to express the difference between necessarily simultaneous actions (the corresponding constants are syntactically equal) and actions that may but do not need occur simultaneously (the corresponding constants are syntactically distinct, even if they may be denotationally equal).

5 Conclusions and Related Works

In this work, the use of linear temporal logic as a planning language is investigated, in the view that a plan is a finite model of the specification of the problem. With respect to formalisms used in the search based approach to planning, the use of LTL shares the advantages of any logical approach: a formal semantics, generality and expressivity. In particular, the language is independent from any specific search strategy. The tableau calculus by means of which model search is performed allows the embedding of different strategies and techniques, depending on the encoding of the problem. This allow one to choose a representation that is best suited to the domain, depending for example on the presence of heuristic information to guide a forward search.

The "planning as satisfiability" paradigm has been proposed by Kautz and Selman [11], the logical language used to encode planning problems being propositional logic. With respect to the use of propositional logic, in an LTL specification time is implicitly represented and the binary temporal operators "since" and "until" give the language a richer expressive power. They make it possible to give natural encodings of planning problems, in such a form that partial order planning in the style of [14] is simulated by the tableau construction. The approach proposed in this work aims in fact at designing a reasoning system that may be "open" to the user. The naturalness of semantically based methods makes it possible to conceive mixed-initiative planning systems, where proofs are expected to convey understandable information.

The application of LTL to planning has been considered in other works. For example, [4] apply the executable temporal language METATEM [3] to planning and scheduling. F. Bacchus and F. Kabanza [2] use a version of temporal logic to specify temporally extended goals as sets of acceptable sequences of states, i.e. temporal models, and define correct plans as those whose execution results in one of such sequences. In that work, however, temporal logic is not used to encode the planning problem entirely: actions are described in an ADL format, and an *Expand* operation is used, that, applied to a state s, generates all the successors of s that are produced by performing any allowed action. In both cited works, only a form of linear, data driven planning is obtained.

This work lays the theoretical grounds for the use of TabPlan to solve planning problems. In order to have a practical system, one of the first issues to be addressed is the extension to a (restricted) first order language. In fact, although from a theoretical standpoint propositional logic suffices to represent planning problems over finite domains, needless to say that treating an existential quantified formula like the finite disjunction of its instances would be a gross violation of the *least commitment* principle.

Furthermore, the class of formulae involved in different encodings of planning problems must be analysed, aiming at defining refinements of the calculus and control search strategies, hence improving TabPlan performance.

References

1. F. Bacchus and F. Kabanza. Using temporal logic to control search in a forward chaining planner. In *Proc. of the TIME-95 International Workshop on Temporal Representation and Reasoning*, Melbourne, Florida, April 1995.
2. Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1215–1222. AAAI Press / The MIT Press, 1996.

3. H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: a framework for programming in temporal logic. In *Proc. of REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *LNCS*. Springer, 1989.
4. H. Barringer, M. Fisher, D. Gabbay, and A. Hunter. Meta-reasoning in executable temporal logic. In *Proc. of the Second Int. Conf. on Principles of Knowledge Representation and Reasoning*, 1991.
5. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
6. S. Cerrito and M. Cialdea Mayer. TabPlan: Planning in linear temporal logic. Technical Report 1141, Université de Paris Sud, Laboratoire de Recherche en Informatique, 1997.
7. S. Cerrito and M. Cialdea Mayer. Bounded model search in linear temporal logic and its application to planning. In H. De Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, number 1397 in LNAI, pages 124–140. Springer, 1998.
8. A. Cesta and A. Oddi. DDL.1: a formal description of a constraint representaton language for physical domains. In M. Ghallab and A. Milani, editors, *New Direction in AI Planning*, pages 341–352. IOS Press, 1996.
9. K. Currie and A. Tate. O-Plan: the open planning architecture. *Journal of Artificial Intelligence*, 52:49–86, 1991.
10. H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proc. of the 5th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 374–384, 1996.
11. H. Kautz and B. Selman. Planning as satisfiability. In B. Neumann, editor, *10th European Conference on Artificial Intelligence (ECAI)*, pages 360–363. Wiley & Sons, 1992.
12. R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and mathematical theory of computation: Papers in honor of John McCarthy*, pages 359–380. Academic Press, 1991.
13. P.H. Schmitt and J. Goubault-Larrecq. A tableau system for linear-time temporal logic. In E. Brinksma, editor, *3rd Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*, LNCS. Springer Verlag, 1997.
14. J. Scott Penberthy and D.S Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. of the Third Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 103–114. Morgan Kauffman Publ., 1992.
15. B. Stephan and S. Biundo. Deduction based refinement planning. In B. Drabble, editor, *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 213–220. AAAI Press, 1996.
16. P. Wolper. The tableau method for temporal logic: an overview. *Logique et Analyse*, 28:119–152, 1985.