The Main Features of a Planner Fully Based on LTL

M. Cialdea Mayer and **C. Limongelli** and **A. Orlandini** and **G. Balestreri**

Dipartimento di Informatica e Automazione, Università Roma Tre,
via della Vasca Navale 79, 00146 Roma, Italia.
{cialdea,limongel,orlandin,gippo}@dia.uniroma3.it

Abstract

In this work we describe a prototype system, PADOK (Planning with Domain Knowledge), where the whole planning domain is modelled in Linear Time Logic (LTL) and planning is reduced to model search. The problem specification accepted by the system can include problem specific information, in a declarative format. The work briefly resumes and extends (Cialdea Mayer *et al.* 2000), showing, through a complete example, how LTL is effectively used as a planning language.

Introduction

Modal temporal logics have been used in different perspectives in the context of automatic planning. For example, (Stephan & Biundo 1996; Koehler & Treinen 1995) view planning as theorem proving in interval-based temporal logics; (Cimatti *et al.* 1997; Cimatti & Roveri 1999) propose planning via model checking. Linear time temporal logic (LTL) has also been used (Bacchus & Kabanza 1996a; 2000; Barringer *et al.* 1991), since it is based on a simple model of time and allows an easy and natural representation of a world that changes over time. In this work we propose the use LTL as a specification language for planning, keeping inside the "planning as satisfiability" approach (Kautz & Selman 1992). Differently from (Bacchus & Kabanza 1996a; 2000), however, LTL is used here to model the whole planning problem (including the initial state, goal and operators), and planning is reduced to model search. The advantages of using LTL instead of classical propositional logic, like in (Kautz & Selman 1992), stem from its greater expressiveness. For example, the representation of a planning problem in classical propositional logic can be done only when a limit to the length of the plan is fixed in advance, and the search proceeds by iteratively deepening such a limit. The use of the binary temporal operators ("until" and "since") allows one to represent the fundamental qualitative relations between time intervals (Allen & Koomen 1983). As a consequence, LTL can express problems where actions have a duration, as well as domain restrictions in the style of (Cesta & Oddi 1996). Moreover, intermediate tasks can easily be represented, like in (Bacchus & Kabanza 1996a), or in the style of the *Hierarchical Task Networks* approach (Erol, Hendler, & Nau 1994; Yang 1990). Most important, LTL can express domain knowledge useful to guide the search, as shown in (Bacchus & Kabanza 1996b; 2000).

We believe that the possibility of specifying extra problem dependent information, such as the domain expert is often aware of, is of great importance. Such additional knowledge can be of great help both in reducing the search space and finding a better plan, especially in complex problems. The statement of domain specific information can include control knowledge and even become a sort of high level, non sequential program that guides the planner. Obviously, with the addition of new restrictions, completeness may be lost. If we want to minimize the risk that additional information spoils the possibility of finding a plan when one exists, the specification itself must be given in some language with a well defined formal semantics, so that meta-level tools can be designed, in order to check, for instance, consistency of the specification. A logical language is obviously the best choice in this respect.

In (Cialdea Mayer *et al.* 2000), we have reported the results of the first experiments with a planner fully based on LTL, PADOK (Planning with Domain Knowledge), built to check the effective feasibility of the approach. We compared PADOK with three other planning systems, on some planning problems in the "classical" style, characterized by an initial state, a goal and a set of instantaneous actions, with preconditions and effects. In general, PADOK, when given suitable domain knowledge, is comparable to the other systems, and its behaviour shows a smaller time growth rate w.r.t. the other planners: when the problem instances increases, the behaviour of PADOK improves.

In this work we consider in further extent one of the planning domains used in the above experiments, as a leading example in order to demonstrate how a planning problem is fully encoded in LTL, what kind of domain knowledge we have found useful and in which form it is represented. Moreover, we point out some of the potentialities of the full encoding of planning problems in LTL, that are not apparent when only problems that can be expressed in ADL-like languages are considered.

PADOK is implemented in Objective Caml (Leroy 1999) and its model search mechanism relies on the system `ptl`, developed in C by G. Janssen at Eindhoven University (Janssen 1999). The system `ptl` is an efficient implementation of proof search in LTL, thanks also to the use of Ordered Binary Decision Diagrams (OBDD).

Planning in LTL

The leading example we are going to consider is the teacup domain with a single robot: a robot has to deliver tea to the inhabitants of a given number of rooms. Each room is connected to the hallway and possibly to other rooms. One of the rooms contains a cup-stack, another the tea-machine. This is a simplification of the teacup domain with two robots, from IPP and BlackBox repositories. The problem kernel is represented in a multi-sorted first-order language over finite, fixed domains, by specifying:

1. The object domains, consisting, in our example, of locations only:

   ```
   type loc: hallway, room1, room2, ...;
   ```

2. The fluents involved in the problem, and the "static predicates", i.e. predicates whose meaning does not change over time (where the tea-machine and cup-stack are located, the connections between locations):

   ```
   fluents = at(loc), fullcup, hascup,
             ordered(loc);
   preds = teamachine(loc), cupstack(loc),
           connected(loc,loc);
   ```

   (`fullcup` is true when the cup is full of tea and `hascup` if the robot is holding the cup; `at(x)` is true when the robot is at the location `x` and `ordered(x)` when the inhabitant of room `x` has ordered tea and has not been served yet.)

3. A specific section contains knowledge about static predicates (assuming a closed world, what is not derivable from such a *background theory*, is false):

   ```
   theory = cupstack(room2), teamachine(room1),
         All x:loc (-(x=hallway)
                       -> connected(x,hallway)),
         connected(room1,room3), ... ;
   ```

   (where `-` is negation and `->` is implication; the universal formula above asserts that every location different from the hallway is connected to the hallway). In general, the background theory contains formulae without temporal operators. Such formulae are meant to be true throughout time, i.e. they represent facts that do not vary over time.

4. The initial state (the robot is in room 1 and every location different from the hallway has ordered tea) and goal:

   ```
   init = at(room1),
       All x:loc (-(x=hallway) -> ordered(x));
   goal = All x:loc -ordered(x);
   ```

5. The operators:

   ```
   op getcup x:loc
     PRE at(x), cupstack(x), -hascup
     POST hascup
   op fillcup x:loc
     PRE at(x), teamachine(x), -fullcup, hascup
     POST fullcup
   op deliver x:loc
     PRE at(x), ordered(x), fullcup, hascup
     POST -fullcup, -hascup, -ordered(x)
   op go x:loc, y:loc
     PRE at(x), connected(x,y) | connected(y,x)
     POST at(y), -at(x);
   ```

   Preconditions can have any form. For instance, the second formula in the precondition of `go` is a disjunction (| denotes $\vee$). Such a precondition dispenses us to explicitly declare that `connected` is a symmetric relation. Conditional and universally quantified effects are also allowed.

The kernel specification is translated into a set of propositional LTL formulae. The model of time underlying LTL is a countably infinite sequence of states (or *time points*), that can be identified with $\mathbb{N}$. An interpretation $\mathcal{M}$ is a function mapping each time point k to the set of propositional letters true at k. Truth in a model is truth in its initial state. In this work we use only the unary future time operators: $\Box A$ ("always A") is true at k iff A is true at any time point $j \geq k$, $\Diamond A$ ("eventually A") is true at k iff A is true at some time point $j \geq k$, and $\circ A$ ("next A") is true at k iff A is true at $k+1$.

A temporal interpretation describes how the world evolves in time, hence a set of formulae constrains the behaviour of the world and can be taken as the specification of a planning domain. A planning problem can be represented by a set of LTL formulae according to different encodings, as shown in (Cerrito & Cialdea Mayer 1998). PADOK uses a simple form of progression encoding, that recalls the encoding of planning problems in the situation calculus (Reiter 1991) and the linear encoding of (Kautz, McAllester, & Selman 1996).

Each ground atom (fluent, static predicate or operator instance) is mapped to a propositional letter. We shall get on using a first order syntax in order to enhance readability, where quantifiers are abbreviations of finite conjunctions or disjunctions. If an operator instance a (an *action*) is true at a given state k of a temporal interpretation $\mathcal{M}$, then $\mathcal{M}$ represents a time sequence where a is performed at state k. Therefore, any initial segment $\langle 0, ..., n\rangle$ of a temporal interpretation $\mathcal{M}$ that satisfies all the constraints of the planning domain corresponds to a (possibly parallel) plan: it determines the sequence of sets of actions $\Pi = \langle S_0, ..., S_n\rangle$ such that for every $k = 0, ..., n$ and action a, $a \in S_k$ if and only if a is true at the time point k of the interpretation.

The LTL encoding of the problem is built by PADOK as follows.

1. From the background theory a set of formulae is constructed, containing, for each ground instance P of a static predicate, either P or $\neg P$. Such formulae are assumed to hold at any time point. They are not explicitly asserted, however, but rather used to simplify all the rest of the encoding: if P always holds, it is replaced everywhere by $True$, otherwise by $False$. The encoding is then simplified in the standard way.

   Moreover, the theory is used to filter operator instances, by elimination of those actions whose preconditions or effects are inconsistent with the theory. Obviously, there are often operator instances which are self-contradictory and should not be taken into consideration. For example, any instance of `go(x,y)` where `x=y` is contradictory: its post-condition, in fact, can never be satisfied, and PADOK automatically rules out such actions. But there are also cases where an operator instance can be ruled out because it is inconsistent with the background knowledge about the domain. For example, if we know that room 2 is not connected to room 3 (and it will never be), then the actions `go(room2,room3)` and `go(room3,room2)` are impossible. For the same reason, only one `getcup` action is actually created, if the cup-stack is in one location only. Action filtering by use of the background theory often dramatically reduces the search space. In our sample domain, 105 contradictory actions (out of 154) are ruled out in the problem instance with 10 rooms.

   The background theory does not necessarily contain only *knowledge about the domain*: sometimes (classical) formulae expressing control knowledge can be used also for simplification purposes.

2. The explicit specification of the initial state, S, is completed according to a closed world assumption w.r.t. fluents. In our example:

   $$\begin{array}{l} at(room1), ordered(room1), ordered(room2), ... \\ \neg at(hallway), \neg at(room2), ... \\ \neg fullcup, \neg hascup, \neg ordered(hallway) \end{array}$$

3. The specification of the goal produces a formula of the form $\Diamond Goal$ (the goal will eventually be true):

   $$\Diamond \forall r : loc\ \neg ordered(r)$$

4. For every action a, a formula of the form $\Box(a \rightarrow \pi_a)$ represents the preconditions for the executability of a: "at any time, a is performed only if its preconditions π_a hold".

   $$\begin{array}{l} \Box(getcup(room2) \rightarrow at(room2) \wedge \neg hascup), \\ \Box(fillcup(room1) \rightarrow \\ \qquad at(room1) \wedge \neg fullcup \wedge hascup), \\ \forall x : loc\ \Box(deliver(x) \rightarrow \\ \qquad at(x) \wedge ordered(x) \wedge fullcup \wedge hascup), \\ \Box(go(room1, hallway) \rightarrow at(room1)), \\ \Box(go(room1, room3) \rightarrow at(room1)), \\ ... \end{array}$$

   Note that static predicates disappear: if `x` is not connected to `y`, the action `go(x,y)` has been filtered out, so that, whenever `go(x,y)` exists, there is no need to state that the two locations are connected.

5. A set of formulae describes incompatibility relations between actions. Obviously, 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 a and b true at the same time point. But if a deletes a precondition of b (or *vice versa*), then the incompatibility must be explicitly coded. For every action a, the encoding contains a formula of the form $\Box(a \rightarrow \neg a_1 \wedge ... \wedge \neg a_n)$, where $a_1, ..., a_n$ are all the actions whose incompatibility with a must be made explicit (in case of conditional effects, incompatibility axioms are weaker).

6. For every ground instance R of a fluent, two formulae are computed from the operators descriptions: G_R^+ specifies all the conditions that can lead to change the truth value of R from false to true, and G_R^- specifies all the conditions that can lead to change the truth value of R from true to false. The encoding includes the following formula, for any fluent R:

   $$\Box(\circ R \equiv G_R^+ \vee (R \wedge \neg G_R^-))$$

   "At any non-initial time point (say $n + 1$), R holds iff at the previous state (n) some action having R as effect is accomplished, or else R holds and no action having $\neg R$ as effect is performed". This is a paraphrase of Reiter's *successor state axiom* (Reiter 1991) into LTL.

   For instance:

   $$\begin{array}{l} \forall x : loc\ \Box(\circ at(x) \equiv \exists y : loc\ go(y, x) \\ \qquad\qquad \vee (at(x) \wedge \neg \exists y : loc\ go(x, y))) \\ \Box(\circ fullcuploaded \equiv fillcup(room1) \\ \qquad \vee (fullcuploaded \wedge \neg \exists r : loc\ deliver(r))) \end{array}$$

Control Knowledge

The encoding schema described above is provably correct and complete (Cerrito & Cialdea Mayer 1998): every model of the encoding represents a valid plan and every plan corresponds to a model of the encoding. However, in many cases, the human expert is often aware of knowledge that can be of precious help for the planner. A section in problem specifications accepted by PADOK is devoted to the declarative representation of *control knowledge*, that can help to reduce the search space and possibly find a better plan.

Control knowledge can be specified by means of any logical formula, possibly containing temporal operators. Such formulae (called *control formulae*) are required to hold at any time point: for each control formula A, the formula $\Box A$ is added to the LTL encoding.

The fact that actions are represented by formulae allows us to explicitly talk of actions in control formulae. In the sample problem we are considering, two main categories of control formulae have been used. The first one represents knowledge of the kind "whenever you are in a situation such that ..., then your action must be one among ...". This form of "positive guide" is similar to a high level programming instruction. The second form of useful knowledge concerns restrictions on the execution of some action, in the form: "do not perform action x unless ...".

In the teacup domain we are considering, the following control formulae have been used:

$$(T1) \quad \forall r : loc\,(\neg(r = hallway) \wedge \exists x : loc\ go(x, r) \rightarrow \circ(deliver(r) \vee fillcup(r) \vee getcup(r)))$$

$$(T2) \quad \forall x : loc\,(at(x) \wedge ordered(x) \wedge fullcup \wedge hascup \rightarrow deliver(x))$$

$$(T3) \quad \forall x : loc\,(at(x) \wedge teamachine(x) \wedge \neg fullcup \wedge hascup \rightarrow fillcup(x))$$

$$(T4) \quad \forall x : loc\,(at(x) \wedge cupstack(x) \wedge \neg hascup \rightarrow getcup(x))$$

$T1$ allows the robot to enter a room only if there is something to do there, other than going away immediately (in the topology of the domain every room is connected to the hallway, so that no shortest length plan is lost). This is a control formula restricting the execution of *go* actions. The other three formulae are strong positive guides for *deliver*, *fillcup* and *getcup* actions: they must be done as soon as possible (the conditions forcing their executions are actually the same as their respective preconditions).

Control knowledge helps not only in reducing the search space, but also to possibly find a better plan. In fact, in the sample domain we are considering, the length of the plan is always reduced and the importance of control knowledge grows with the size of the instances.

	with guide		without guide	
rooms	secs	length	secs	length
8	2.46	68	4.67	168
10	3.79	87	6.84	253
12	7.26	104	13.39	355
14	12.48	123	21.85	474
16	15.26	140	38.32	610
18	23.27	159	48.28	763
20	30.59	177	59.05	933
22	40.97	195	75.34	1120
24	54.02	213	–	–

In (Cialdea Mayer *et al.* 2000), the results of a comparison with three other planners, BlackBox, IPP and STAN, on some sample problems are reported. The aim was to check whether the approach fully based on LTL and a general purpose proof method is actually feasible, when the expressive power of the language is exploited so as to encode rich problem specific knowledge.

In general, the behaviour of PADOK is quite poor without a background theory and control knowledge, and it highly improves with the encoding of suitable knowledge. Even under the guide of control knowledge, on easy problems the system is usually less efficient than the other planners we have considered. However, its performances improve with the difficulty of the problems, in comparison with the other planners and, in some domains, in the more complex problem instances it is faster than all the other planners and it can solve problems that other planners cannot.

In the teacup domain, provided with the control formulae $T1-T4$, PADOK has solved the problem up to 24 rooms. The other planners have not solved the problem with more than 12 rooms. The table below shows the solution times (in seconds) of PADOK, IPP and STAN.

The teacup domain (1 robot)

rooms	PADOK	STAN	IPP 4.0
4	0.48	0.1	0.04
6	1.18	0.39	0.35
8	2.46	1.78	1.63
10	3.79	25.15	16.65
12	7.26	683.7	106.27
14	12.48	–	–

The intuition that the importance of domain specific and guiding knowledge grows with the size of the addressed problems is confirmed by our experiments: in general, we could note that the time growth rate in PADOK behaviour is considerably smaller than the other planners, i.e. it gains in relative efficiency when the size of the problem increases. We are optimistic also with respects to more complex domains: the implementation of our system can still be improved and, if the slower growth rate we have observed in PADOK execution time w.r.t. other planners is confirmed, the approach can aim at real problems, where we usually have a high degree of guiding knowledge.

Exploiting the Expressivity of LTL

PADOK has not been directly compared with TLplan (Bacchus & Kabanza 2000), a planning system based on a forward chaining algorithm and exploiting control knowledge expressed in LTL. According to the results reported in (Bacchus & Kabanza 2000), comparing TLplan with some of the same planners we have considered, TLplan is considerably more efficient than PADOK. However, PADOK uses LTL to represent the whole planning domain, including the execution of actions, and this makes it possible to represent in a natural way control knowledge explicitly referring to actions, as well as new kinds of domain constraints.

Obviously, in order to compare PADOK with the other planners, we had to run the system on domains whose kernel can be expressed in ADL-like langugages. In order to appreciate the flexibility of LTL, let us consider the following example, borrowed from (Allen 1991), that is not easily and naturally representable in ADL-like planning languages. In order to open the door to the Computer Science Building at Rochester, both hands must be used: a spring lock must be held open with one hand, while the door is pulled open with the other hand. Unless the lock is held open, it snaps shut.

This is an example where the effect of two actions performed together is different from the sum of their effects. Let us consider the (propositional) language with the single fluent `open` (the door is open) and the two actions `pull_door` and `hold_lock`. The following specification is accepted by PADOK, which correctly opens the door by executing the two actions together:

```
fluents = open;
op pull_door
   POST hold_lock => open
op hold_lock
   POST pull_door => open;
goal = open;
```

This is an example with conditional effects, where conditions are actions. In general, any formula can be used to express conditions, with no syntactical restrictions.

Intermediate tasks can also be easily encoded: the specification of the initial state may contain formulae with temporal operators, and, in particular, formulae of the form $\Diamond Task$ ("eventually, $Task$ is accomplished"). Furthermore, all the basic relations on time intervals considered for instance in (Allen & Koomen 1983) can easily be represented in full LTL, and most of them can be expressed without use of the past time operators. We are presently working at an extension of PADOK that allows the specification of temporal relations of this kind in the description of operators.

A further important advantage deriving from the full use of a logical language is the possibility of developing domain analysis tools, such as for instance off-line consistency checking. Finally, the full encoding of planning problems into LTL makes the system potentially parametric w.r.t. the model search algorithm, similarly to Satplan and BlackBox. A planner fully based on LTL can gain immediate benefits from the development of new, more efficient model search algorithms. The above points are important issues that we intend to address.

In this work we have not considered the positive effect of simplification and control knowledge in detail, by means of an internal analysis of our system. This is the subject of a next work, where we also intend to further analyse the different forms of control formulae that can be included in a specification, and their respective effects.

Acknowledgments. This work has been partially supported by Italian MURST, ASI and CNR. Moreover, the authors wish to thank Geert Janssen, who kindly made his system `ptl` available to us and never denied his help, and the anonymous referees, for their useful comments and suggestions.

References

Allen, J. F., and Koomen, J. A. 1983. Planning using a temporal world model. In *Proc. of IJCAI-83*, 741–747.

Allen, J. F. 1991. Temporal reasoning and planning. In Allen, J. F.; Kautz, H. A.; Pelavin, R. N.; and Tenenberg, J. D., eds., *Reasoning about Plans*. Morgan Kaufmann Publishers, Inc. 1–68.

Bacchus, F., and Kabanza, F. 1996a. Planning for temporally extended goals. In *Proc. of AAAI-96*, 1215–1222.

Bacchus, F., and Kabanza, F. 1996b. Using temporal logic to control search in a forward chaining planner. In Ghallab, M., and Milani, A., eds., *New Directions in AI Planning*. IOS Press (Amsterdam). 141–156.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *J. of Artificial Intelligence*. To appear.

Barringer, H.; Fisher, M.; Gabbay, D.; and Hunter, A. 1991. Meta-reasoning in executable temporal logic. In *Proc. of KR-91*.

Cerrito, S., and Cialdea Mayer, M. 1998. Using linear temporal logic to model and solve planning problems. In Giunghiglia, F., ed., *Proc. of the 8th Int. Conf. on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98)*, 141–152.

Cesta, A., and Oddi, A. 1996. DDL.1: a formal description of a constraint representaton language for physical domains. In Ghallab, M., and Milani, A., eds., *New Direction in AI Planning*, 341–352. IOS Press.

Cialdea Mayer, M.; Orlandini, A.; Balestreri, G.; and Limongelli, C. 2000. A planner fully based on linear time logic. In *Proc. of AIPS-2000*.

Cimatti, A., and Roveri, M. 1999. Conformant planning via model checking. In *Proc. of ECP-99*.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via model checking: a decision procedure for $\mathcal{AR}$. In *Proc. of ECP-97*, 130–142.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Complexity results for HTN planning. In *Proc. of AAAI-94*.

Janssen, G. L. J. M. 1999. *Logics for Digital Circuit Verification. Theory, Algorithms and Applications.* CIP-DATA Library Technische Universiteit Eindhoven.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proc. of ECAI-92*, 360–363.

Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proc. of KR'96*, 374–384.

Koehler, J., and Treinen, R. 1995. Constraint deduction in an interval-based temporal logic. In Fisher, M., and Owens, R., eds., *Executable Modal and Temporal Logics, (Proc. of the IJCAI'93 Workshop)*, 103–117.

Leroy, X. 1999. The Objective Caml system, documentation and user's guide, release 2.02. Available at `http://caml.inria.fr/`.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and mathematical theory of computation: Papers in honor of John McCarthy*. Academic Press. 359–380.

Stephan, B., and Biundo, S. 1996. Deduction based refinement planning. In *Proc. of AIPS-96*, 213–220.

Yang, Q. 1990. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence* 6:12–24.