A Planner Fully Based on Linear Time Logic

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

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

Abstract

This work aims at verifying the effective possibility of using Linear Time Logic (LTL) as a planning language. The main advantage of such a rich and expressive language is the possibility of encoding problem specific information, that can be of help both in reducing the search space and finding a better plan. To this purpose, we have implemented a planning system, PADOK (Planning with Domain Knowledge), where the whole planning domain is modelled in LTL and planning is reduced to model search.

We briefly describe the components of problem specifications accepted by PADOK, that may include knowledge about the domain and control knowledge, in a declarative format. Some experiments are then reported, comparing the performances of PADOK with some well established existing planners (IPP, BLACKBOX and STAN) on some sample problems. In most cases, our system is guided by additional knowledge that cannot be stated in the languages accepted by the other planners. In general, when the complexity of the problem instances increases, the behaviour of PADOK improves, with respect to the other planners, and it can solve problem instances that other systems cannot.

Introduction

The expressive power of planning languages is an important issue, addressed for example in (Currie & Tate 1991; Stephan & Biundo 1996; Cesta & Oddi 1996). The logical approach to planning, traditionally based on deduction, is certainly more expressive and flexible than specialized planning systems. However, the use of general theorem provers often raises efficiency problems. More effective planners are based on the "planning as satisfiability" approach: a plan corresponds to a model of the problem specification. In (Kautz & Selman 1992) planning is reduced to model search in classical propositional logic, and in (Kautz, McAllester, & Selman 1996) different encodings of planning problems are presented.

Since modal temporal logics are useful tools for modeling and reasoning about time, their application to planning has been proposed in different formats: (Stephan & Biundo 1996; Koehler & Treinen 1995) use interval-based temporal logics, in a deductive view; (Cimatti *et al.* 1997; Cimatti & Roveri 1999) propose planning via model checking; (Barringer *et al.* 1991) and (Bacchus & Kabanza 1996) use model construction in linear time temporal logic; (Cerrito & Cialdea Mayer 1998) analyses the general forms that may take the encoding of a planning problem in Linear Time Logic (LTL).

Similarly to (Bacchus & Kabanza 1996; Barringer *et al.* 1991; Cerrito & Cialdea Mayer 1998) in this work we propose to use LTL as a specification language for planning, keeping inside the "planning as satisfiability" approach. The use of LTL as a planning language has several advantages. A first important aspect of LTL is its underlying simple model of time, allowing an easy and natural representation of a world that changes over time. It is more expressive than classical propositional logic; for instance, even if we identify a plan with a finite time sequence, we do not need to fix a maximal length to such a time segment. Most important, we recall the obvious benefits deriving from its generality and expressiveness, compared with special purpose planning formalisms. In particular, we are not bound to adhere to the model of a single final goal and instantaneous actions: LTL can easily been used to express domain restrictions in the style of (Cesta & Oddi 1996), as well as intermediate tasks, like in (Bacchus & Kabanza 1996), or in the style of the *Hierarchical Task Networks* approach (Erol, Hendler, & Nau 1994; Yang 1990). Furthermore, LTL can express domain knowledge useful to guide the search, as shown in (Bacchus & Kabanza 1995).

This latter issue is the motivation of this work. We believe that the possibility of specifying extra problem specific information, such as the domain expert is often aware of, is of great importance. In fact, such additional knowledge can be of great help both in reducing the search space and finding a better plan, especially in complex problems. The specification of such 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.

The main issue we address in this work is the effective feasibility of an approach to planning based on LTL: can model search in LTL be effective enough in order to constitute the kernel of a planner, when the declarative specification of additional domain knowledge is allowed? In order to address the problem, we have built a prototypal system, PADOK (Planning with Domain Knowledge), run it on a set of examples, and compared its results with some other existing planning systems, i.e. IPP (Koehler *et al.* 1997), BLACKBOX (Kautz & Selman 1998a) and STAN (Long & Fox 1998). For the comparison we only consider planning problems in the "classical" style, characterized by an initial state, a goal and a set of instantaneous actions, with preconditions and effects. However, PADOK is in most examples guided by problem specific information. In many cases its behaviour shows a lower time growth rate w.r.t. the other planners: when the problem instances increases, PADOK behaviour improves. In general, PADOK, when given suitable domain knowledge, is comparable to the other systems.

The Encoding of Planning Problems

We consider the language of propositional LTL containing only unary future time temporal operators: $\Box A$ means that A is true now and will always be true, $\Diamond A$ that A is either true now or sometime in the future, and $\bigcirc A$ that A holds in the next state. The model of time underlying LTL is a countably infinite sequence of states (a *time frame*), that can be identified with $\mathbb{N}$. Its elements are called time points, and an interpretation $\mathcal{M}$ is a function mapping each time point i to the set of propositional letters true at i.

We consider the connectives $\neg$ (negation) and $\vee$ (disjunction) as primitive, and the other propositional operators $\wedge$ (conjunction), $\rightarrow$ (implication) and $\equiv$ (double implication) are defined as usual. The satisfiability relation $\mathcal{M}_i \models A$, for $i \in \mathbb{N}$ (A is true at time point i in the interpretation $\mathcal{M}$), is defined as follows:

1. $\mathcal{M}_i \models p$ iff $p \in \mathcal{M}(i)$, for any propositional letter p in the language.
2. $\mathcal{M}_i \models \neg A$ iff $\mathcal{M}_i \not\models A$.
3. $\mathcal{M}_i \models A \vee B$ iff either $\mathcal{M}_i \models A$ or $\mathcal{M}_i \models B$.
4. $\mathcal{M}_i \models \Box A$ iff for all $j \geq i$, $\mathcal{M}_j \models A$.
5. $\mathcal{M}_i \models \Diamond A$ iff there exists $j \geq i$ such that $\mathcal{M}_j \models A$.
6. $\mathcal{M}_i \models \bigcirc A$ iff $\mathcal{M}_{i+1} \models A$.

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 temporal interpretation describes how the world evolves in time, hence a suitable set of formulae can constrain the behaviour of the world and be taken as the specification of a planning domain.

In the following, we describe how to represent planning problems such as can be specified in ADL-like languages. Such a description can be found also in (Cerrito & Cialdea Mayer 1998), together with alternative forms of encodings in LTL, and recalls the encoding of planning problems in the situation calculus (Reiter 1991) and the linear encoding of (Kautz, McAllester, & Selman 1996). We assume that a propositional language $\mathcal{L}$ is given, containing propositional letters (or ground atoms – we shall use a first order syntax in order to enhance readability) representing all the relevant "fluents", i.e. properties that may change over time. Furthermore, the language contains a distinguished set *Actions* of atoms, representing the execution of actions (i.e. if a is an action and a is true at state i of a given temporal interpretation $\mathcal{M}$, then $\mathcal{M}$ represents a time sequence where a is performed at state i). Hence, any initial segment $\langle 0, ..., n\rangle$ of a temporal interpretation $\mathcal{M}$ of the language $\mathcal{L}$ corresponds to a (possibly parallel) plan: it determines the sequence of sets of actions $P = \langle A_0, ..., A_n\rangle$ such that for every $i = 0, ..., n$ and action a, $a \in A_i$ if and only if $\mathcal{M}_i \models a$.

For simplicity, we assume here that a planning problem Π is specified by a complete description S_0 of the initial state, the description *Goal* of the final state, and a description of the available actions, with their preconditions and (possibly conditional) effects. 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 state satisfies S_0 and there exists a state g satisfying *Goal*; if a is an action holding at time point i then the preconditions of a hold at i; and classical effect and frame conditions for actions are satisfied at any time point). 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 $\mathcal{L}$, satisfying all the constraints of Π, that determines P. Hence, plans can be characterized in terms of temporal models.

Finally, a set of LTL formulae S over $\mathcal{L}$ is a correct and complete encoding of a planning problem Π iff: for every temporal interpretation $\mathcal{M}$ of $\mathcal{L}$, $\mathcal{M}$ 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.

In the rest of this section a simple encoding schema is presented, that is provably correct and complete (Cerrito & Cialdea Mayer 1998). The specification of the initial state, goal and action preconditions are represented as follows: (1) The initial state is represented by a formula S_0, that is assumed to be complete with respect to fluents, i.e. for any fluent R, either $S_0 \models R$ or $S_0 \models \neg R$. (2) The goal is represented by a formula of the form $\Diamond Goal$: "either now or sometime in the future $Goal$ is true". (3) 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" (action precondition axioms).

The encoding contains also a set of axioms describing 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 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 $\mathcal{M}_i \models a \wedge b$, 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 that case the encoding is added an "incompatibility axiom" between a and b, with the form $\Box(\neg a \vee \neg b)$: "at any time, a is performed only if no contemporary execution of b is performed".[1]

In order to express general frame and effect conditions for actions, we assume that for any fluent R two formulae are specified, that can easily be computed from the actions description: 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. We consider a simple form of *progression encoding*, including, for any fluent R, the following formula:

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

"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.

Planning in LTL with PADOK

In order to check whether LTL can be effectively "executed", we have built a prototypal system, PADOK. It 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).

PADOK takes in input the specification of a planning problem given partly in an ADL-like syntax, and translates it into a set of propositional LTL formulae, according to the encoding described in the previous section. The encoding can be enriched with additional problem specific knowledge, included in the specification (as described further on). The system `ptl` is invoked in order to search for a model of the produced axioms and, if some model is found, the corresponding plan is given in output.

The syntax allows universal as well as existential quantification (over finite domains) and conditional effects for actions. Quantifiers are considered as abbreviations for finite disjunctions or conjunctions. Atomic formulae include some simple arithmetical formulae, whose interpretation is the intended one. In particular, the specification of a planning problem includes the declaration of each object type involved in the problem, the set of fluents and static predicates, the initial state (that must not contain positive disjunctions or existential quantifiers), the goal (any syntactical form) and the operators.

An operator is declared by specifying its name, the type of the operands, its pre- and post-conditions. Action preconditions are allowed to have any syntactical form and, when an action has a conditional effect, any formula can be used to express the condition, with no syntactical restrictions. In detail, post-conditions may have one of the following forms:

$$\forall x_1 : t_1 ... \forall x_n : t_n (L_1 \wedge ... \wedge L_k)$$
$$\forall x_1 : t_1 ... \forall x_n : t_n (F \Rightarrow L_1 \wedge ... \wedge L_k)$$

where $n \geq 0$, $k \geq 1$, $L_1, ..., L_k$ are literals and F is any formula. An effect of the second form means that for all x_1 of type t_1, ... for all x_n of type t_n, if F holds in the state where the action is performed, then $L_1 \wedge ... \wedge L_k$ holds in the next state (logical implication is denoted by the distinguished symbol $\rightarrow$).

Let us consider, for example, the `jewelry-box` domain, where knobs are named by natural numbers (0 is the first knob) and there are two operators: $first$, with no parameters and no preconditions, turning the first knob, and $turn$, for the knobs following the first. The preconditions of $turn$ specify that a given knob c can be turned only if it is not the first knob (otherwise $first$ is applied), its predecessor $c-1$ is closed and all the knobs preceding $c-1$ are open.

$$\begin{array}{lll} action & turn & c : knob \\ & PRE & c > 0, \quad \neg open(c-1), \\ & & \forall x : knob(x < c-1 \rightarrow open(x)) \\ & POST & open(c) \Rightarrow \neg open(c), \\ & & \neg open(c) \Rightarrow open(c) \end{array}$$

The following example is the specification of the action $stack$ (move a block on another block) in the `blocks` domain (where the table is an object of different type).

$$\begin{array}{lll} action & stack & b : block, to : block \\ & PRE & free(b), free(to) \\ & POST & \forall b' : block \\ & & \quad (on(b,b') \Rightarrow free(b') \wedge \neg on(b,b')), \\ & & on(b, Table) \Rightarrow \neg on(b, Table), \\ & & \neg free(to), on(b,to) \end{array}$$

Considering the operators declarations, PADOK computes each pair of incompatible actions and adds the necessary axioms to the LTL encoding of the problem.

[1] When actions have conditional effects, such axioms may be weakened. We omit the details here.

Besides these basic information concerning the domain and the problem instance, PADOK exploits the expressive power of LTL by including the specification of both a *background theory* and *control knowledge*, in order to reduce the search space and possibly find a better plan. In other words, the LTL encoding of the basic information about a planning domain (as described above) determines models which are indeed a correct representation of a plan solving the given problem, but, in many cases, the human expert is often aware of knowledge that can be of precious help for the planner. PADOK allows the addition of such knowledge, in a declarative form. The logical format of both the background theory and control knowledge is suited to be checked for consistency off-line, thus providing a precious help to the domain expert.

The domains considered in the experiments and the examples

The `gripper` domain. A robot with two hands has to move a given number of balls from room A to room B (from AIPS-98 planning competition).

The `jewelry-box` domain. In order to open a jewelry-box, its n knobs have to be turned open. The knobs are arranged in a sequence. A knob can be turned (open or closed) only if: either it is the first knob, or it is the knob immediately following the first closed knob (from (Janssen 1999)).

The `blocks` domain. A given number of blocks have to be arranged in a given goal configuration, moving only one block at a time.

The `teacup` domain with one robot. A robot must 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 cupstack, another the tea-machine. This is a simplification of the next problem and a general "delivery" problem.

The `teacup` domain with two robots. In this version of the domain, each of the two robots is allowed only in some of the rooms, so they have to exchange empty or full cups in order to reach the goal (from IPP and BLACKBOX repositories).

The Background Theory

The background theory contains formulae without temporal operators. Such formulae are meant to be true throughout time, i.e. they represent state invariants. Consequently, for each formula A in the theory, the formula $\Box A$ is added to the problem encoding, so that the searched model is required to satisfy A at each time point, for any A in the theory. This is a natural representation of facts that do not vary over time. For example, in the `teacup` domains, the topology of the rooms (which rooms are connected one to the other) is described in the theory.

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, consider the operator *go* in the `teacup` domain with one robot:

$$\begin{array}{lll} \textit{action} & \textit{go} & r1 : \textit{location}, r2 : \textit{location} \\ & \textit{PRE} & \textit{atRobby}(r1), \textit{connected}(r1, r2) \\ & \textit{POST} & \textit{atRobby}(r2),\ \neg \textit{atRobby}(r1) \end{array}$$

Any instance of *go* where $r1 = r2$ 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 should be ruled out not just because it is self contradictory, but because it is inconsistent with some 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. PADOK exploits the knowledge in the background theory to this aim, often dramatically reducing the search space. For example, 105 contradictory actions (out of 154) are ruled out in the problem instance of the `teacup` domain with one robot and 10 rooms.

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

Control Knowledge

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

In the sample problems we have considered, three main categories of formulae have been used to encode control knowledge. The first one is represented by formulae of the form $A \to \Box A$: if something holds at any given stage, then it will be true forever. This kind of suggestion is useful in problems where "good situations" can be identified, i.e. portions of the goal that, when achieved, need never be destroyed and rebuilt again.

Control statements of the second kind say that "whenever you are in a situation such that ..., then your action must be one among ...". This is in fact similar to a high level programming instruction. For example, in the `gripper` domain it is immediate to see that a good strategy would not loose the opportunity to drop a ball at its destination as soon as it were possible:

$$\forall x : \textit{ball}\, \forall g : \textit{gripper}\ (\textit{atRobby}(B) \land \textit{carry}(x, g) \\ \to \textit{drop}(x, B, g))$$

(if the robot is holding something and it is in room B, then it must drop it). Similarly, we can state that if the robot is in room A, it has a free hand and there are still balls in A, then it must do a *pick* action:

$$atRobby(A) \wedge \exists g : gripper\, free(g) \wedge \exists x : ball\, at(x, A) \\ \rightarrow \exists x : ball\, \exists g : gripper\, pick(x, A, g)$$

The same form of "positive guide" is represented in the `teacup` domain with two robots by the following formula, stating that if a robot is in the condition of making a *deliver* action, then it must profit immediately of the occasion:

$$\forall p : robot\, \forall r : room(at(p, r) \wedge ordered(r) \\ \wedge fullcuploaded(p) \rightarrow deliver(p, r))$$

The third form of useful knowledge concerns restrictions on the execution of some action, in the form: "do not perform action x unless ...". For example, in the `teacup` domain with the given topology – as well as in other domains involving some "go" action – we can harmlessly restrict the robot to enter a room only if there is something to do there (i.e. either serve tea, or fill in a cup with tea, or get an empty cup):

$$\forall r : room(\exists x : location\; go(x, r) \rightarrow \\ \bigcirc(deliver(r) \vee fillcup(r) \vee getcup(r)))$$

As can be noticed from the above examples, the guiding knowledge that can be encoded in LTL can easily be viewed as a sort of high level, non sequential program. If we imagine the plan is in fact to be executed by some artificial agent, we can consider LTL as a sort of very high level robot programming language, where maximum freedom is left in the choice of many details, but the agent is guided in the essential steps of its actions, so as to improve its performances.

Control knowledge helps not only in reducing the search space, but also to possibly find a better plan. For example, in the case of the `blocks` domain, under suitable control, a problem instance with 10 blocks is solved in approximately 7.5 seconds and a plan with 11 actions is found. By contrast, if no control knowledge is used, the plan grows to 109 actions and the time to solve the problem is more than 60 seconds (if no simplification is made, the problem just cannot be solved). In other cases, the time required to find a solution does not explode in such a way and it can even be smaller in simple problem instances, because of the overhead due to the treatment of the additional knowledge. However, the length of the plan is always reduced. The next table illustrates the case of the `teacup` domain with one robot (the instance with 24 rooms cannot be solved without guide).

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

Experiments and Comparisons

PADOK has been compared with three other planners, BLACKBOX (v. 3.4), IPP (v. 3.3 and 4.0) and STAN (v. 3.0), on some sample problems. All the systems run on a Pentium II 400 Mhz under Linux. BLACKBOX, IPP and STAN are three of the five contestants participating to the First Planning System Competition at AIPS-98 and IPP v. 3.3 is the planner that won the competition in the ADL track. Recently, the new release of IPP has been delivered, so we have considered it too.

The aim of the comparisons is to check whether the approach 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.

The results of the comparisons are given by tables and diagrams, showing the solution time (in seconds) required by the different systems. Plan lengths are not reported, since they do not differ significantly. We report the results obtained for IPP v. 3.3 only in the case of the `jewelry-box` domain, because in this case the difference in the performances of the two versions is significant. In the other domains the more recent version behaves better, but the results are not so dramatically different as in the case of the `jewelry-box` domain. In particular, IPP 3.3 can solve neither the 12 balls instance of the `gripper` domain, nor the 12 rooms instance of the `teacup` domain with one robot, nor the 10 rooms instance of the `teacup` domain with two robots.

The performances of PADOK depend on the nature of the problem. In some problems the system behaves well even in absence of guiding knowledge. This is the case, for example, of the `jewelry-box` domain, illustrated below. Note that this is an easy domain when measured in terms of number of fluents and propositional actions, but it requires very long solutions: to solve the 10 knobs instances 682 actions are required, and the 16 knobs instance requires 43.690 actions.

Jewelry-box domain					
knobs	PADOK	Blackbox	STAN	IPP 3.3	IPP 4.0
6	0.05	0.41	0.085	18.91	0.01
8	0.13	4.01	–	–	0.08
10	1.02	–	–	–	0.32
12	4.93	–	–	–	2.38
14	24.79	–	–	–	11.64
16	107.28	–	–	–	64.97

However, the `jewelry-box` domain is an exception: in most domains, the behaviour of the system is quite poor without a background theory and control knowledge, and it highly improves with the encoding of suitable knowledge. Considering that PADOK uses a systematic model search mechanism, our experiments confirm the results reported in (Kautz & Selman 1998b), and the positive effect that additional knowledge may have.

There are also problems where PADOK behaves quite poorly, in comparison with other planners. The blocks world seemed to be one of such bad domains: in easy instances PADOK is much slower than other planners and an earlier version of the system could not solve more difficult cases. However, some optimization of the code made it possible to go further and solve more instances of the problem. In such more difficult cases, PADOK shows a better behaviour than the other planners. Specifically, we have considered random configurations for different numbers of blocks. Obviously, the number of blocks alone is not a good representation of the dimension of a problem in the blocks world, but the initial and final configuration should also be taken into account. For the larger problems that have been tried by all planners (i.e. 12 and 14 blocks) we have considered two configurations: an easy one (12a and 14a, respectively, that are essentially reversal problems) and a random one (12b and 14b). With no guide, the configurations with 12 blocks and over have not been solved by PADOK. The results reported below concern runs with knowledge about the destination of blocks and the definition of "good towers", that must never be undone; moreover, a *stack* action is executed only if a good tower is obtained. With this knowledge, PADOK could solve at least problems with 24 blocks. The results are reported in the following table.

The blocks domain				
blocks	PADOK	Blackbox	STAN	IPP 4.0
6	0.49	0.38	0.19	0.12
8	2.94	1.38	0.51	0.44
10	7.5	6.04	1.62	1.50
12a	12.91	2.74	2.57	3.42
12b	23.9	–	26.10	20.28
14a	23.57	7.20	8.10	9.93
14b	51.44	–	–	–

We note that PADOK execution time for the easy configurations with 12 and 14 blocks is approximately half the corresponding more difficult one. It seems to profit less than the other systems of the easy situations 12a and 14a. The graphical representation in Figure 1 shows that in easy problems, PADOK behaves very poorly (the configurations with 12 and 14 blocks are the easier ones).

The knowledge about the `gripper` domain exploited by PADOK consists of heuristics: never drop a ball in the origin room and never pick it back from the destination room, otherwise do pick/drop actions whenever possible. With this help, PADOK can solve problems with at least 30 balls (in approximately 166 seconds).

The gripper domain				
balls	PADOK	Blackbox	STAN	IPP 4.0
4	0.14	0.19	0.09	0.02
6	1.02	5.66	0.13	0.27
8	2.9	–	2.03	3.93
10	5.12	–	39.16	39.87
12	8.32	–	–	–

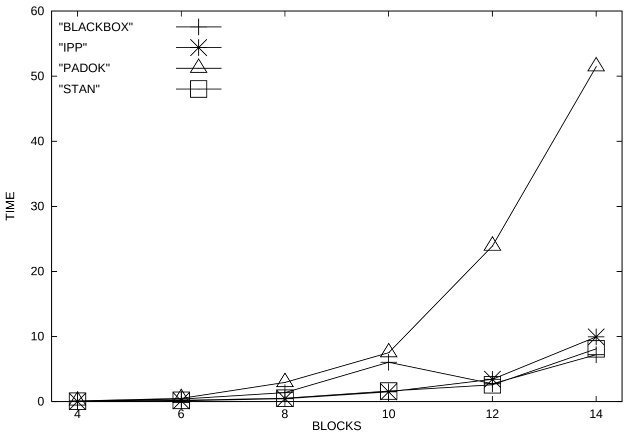

Figure 1. The easier instances in the blocks domain

Figure 2 represents the execution times of the four planners in the gripper domain.

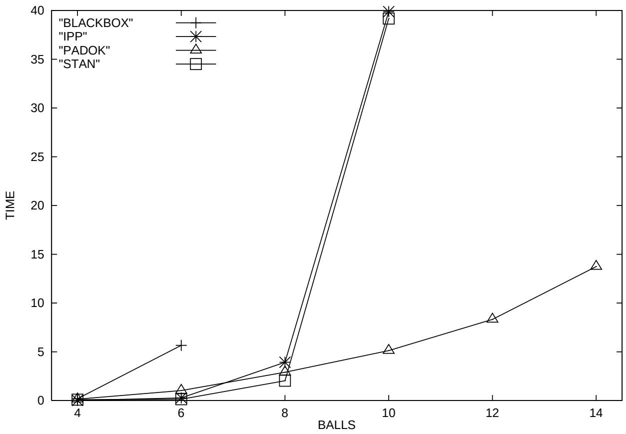

Figure 2. The gripper domain

The knowledge about the domain exploited in the two teacup domains concern essentially the topology of the rooms, the location of the tea-machine and the cups, and, in the two robots case, where each of them is allowed. With such information, PADOK has solved the teacup problem with one robot up to 24 rooms, and the teacup problem with two robot up to 20. In the tables and figures below, we omit data about Blackbox, that could only solve the 4 rooms instance of the teacup domain with one robot.

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 teacup domain (2 robots)			
rooms	PADOK	STAN	IPP 4.0
4	1.56	0.29	0.29
6	6.51	2.72	4.65
8	7.92	32.0	56.36
10	11.99	544.00	369.89
12	26.12	–	–

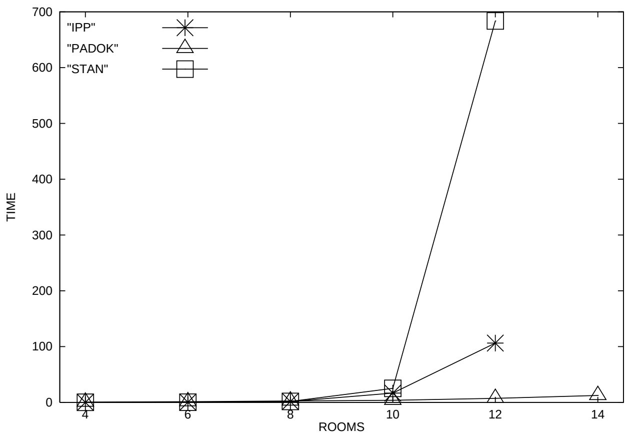

Figure 3. The teacup domain with one robot

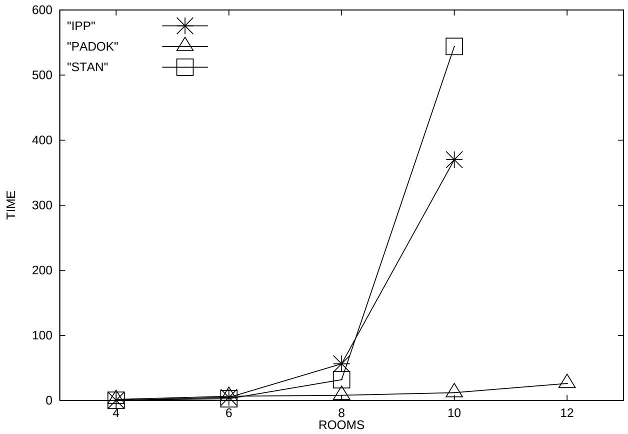

Figure 4. The teacup domain with two robots

In general, we can note that the time growth rate in PADOK behaviour is considerably smaller than the other planners: it gains in relative efficiency when the size of the problem increases. As a consequence, in the more complex problem instances it is faster than all the other planners and it can solve some problem instances that other planners cannot.

Concluding Remarks

In this work we have addressed the problem of checking whether the expressive power of Linear Time Logic as a representation language for planning problems can be effectively exploited, especially considering the benefits that can be gained from the declarative representation of domain specific and control knowledge. We have shown some examples of guiding knowledge that can be expressed in the LTL specification of a planning problem, and presented the first experiments with PADOK, a prototypal planning system based on LTL and strongly exploiting domain specific information.

The experimental results, comparing PADOK with other planners (specifically, BLACKBOX, IPP and STAN), show that in many domains, when we can give the planner meaningful and important information (in a relatively simple form), the time growth rate in PADOK behaviour is smaller, in comparison with the other planners we have considered. As a consequence, in the more complex problem instances of such domains, PADOK is behaves better than the other planners. The intuition that the importance of domain specific and guiding knowledge grows with the size of the addressed problems is confirmed by our experiments. 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.

In general, we believe that the loss in efficiency that often derives from the use of general logical procedures can be recovered with the expressive power of the language and the declarative specification of heuristics. Now, other benefits can be easily obtained, for instance from the possibility to represent actions with a duration and intermediate tasks.

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 effect. Such a work is in fact essential, in order to provide a guide on what knowledge can be of help and how it is better formulated.

In (Cerrito & Cialdea Mayer 1998), beyond the encoding considered in this work, alternative forms of encodings are presented: a "forward" encoding exploiting the *Until* operator, and two forms of "backward" encodings. As a next task, we shall experiment such encodings, in order to check whether either the use of the binary temporal operators or a form of goal driven encoding provide any advantage. In fact, it can happen that, in dependence of the nature of the problem, one or another search strategy is to be preferred. The planning system could allow the user to choose between the corresponding forms of encoding.

A further important step for future research consists in developing domain analysis tools, such as for instance off-line consistency checking, thus exploiting the advantages deriving from the use of a logical language.

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.

References

Bacchus, F., and Kabanza, F. 1995. Using temporal logic to control search in a forward chaining planner. In *Proc. of the TIME-95 International Workshop on Temporal Representation and Reasoning.*

Bacchus, F., and Kabanza, F. 1996. Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1215–1222. AAAI Press / The MIT Press.

Barringer, H.; Fisher, M.; Gabbay, D.; and Hunter, A. 1991. Meta-reasoning in executable temporal logic. In *Proc. of the Second Int. Conf. on Principles of Knowledge Representation and Reasoning.*

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

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.

Cimatti, A., and Roveri, M. 1999. Conformant planning via model checking. In *Proc. of the Fifth European Conference on Planning (ECP-99).*

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via model checking: a decision procedure for $\mathcal{AR}$. In Steel, S., and Alami, R., eds., *Proc. of the Fourth European Conference on Planning (ECP-97)*, 130–142. Springer-Verlag.

Currie, K., and Tate, A. 1991. O-Plan: the open planning architecture. *Journal of Artificial Intelligence* 52:49–86.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Complexity results for HTN planning. In *Proceedings 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 Neumann, B., ed., *10th European Conference on Artificial Intelligence (ECAI)*, 360–363. Wiley & Sons.

Kautz, H., and Selman, B. 1998a. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the AIPS-98 Workshop on Planning as Combinatorial Search*, 58–60.

Kautz, H., and Selman, B. 1998b. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proc. of the Fourth Int. Conf. on Artificial Intelligence Planning Systems (AIPS-98).*

Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proc. of the 5th Int. Conf. on Principles of Knowledge Representation and Reasoning (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)*, volume 897 of *LNAI*, 103–117. Springer.

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In Steel, S., and Alami, R., eds., *Proc. of the Fourth European Conference on Planning (ECP-97)*, 273–285. Springer-Verlag.

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

Long, D., and Fox, M. 1998. Type analysis of planning domain descriptions. In *17th Workshop of UK Planning and Scheduling.*

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 Drabble, B., ed., *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, 213–220. AAAI Press.

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