Planning for Temporally Extended Goals*

Fahiem Bacchus[1] and Froduald Kabanza[2]

[1] *Dept. of Computer Science*
University of Waterloo
Waterloo, Ontario
Canada, N2L 3G1
E-mail: fbacchus@logos.uwaterloo.ca

[2] *Dept. de Math et Informatique*
Universite de Sherbrooke
Sherbrooke, Quebec
Canada, J1K 2R1
E-mail: kabanza@dmi.usherb.ca

In planning, goals have been traditionally been viewed as specifying a set of desirable final states. Any plan that transforms the current state to one of these desirable states is viewed to be correct. Goals of this form are limited in what they can specify, and they also do not allow us to constrain the manner in which the plan achieves its objectives.

We propose viewing goals as specifying desirable sequences of states, and a plan to be correct if its execution yields one of these desirable sequences. We present a logical language, a temporal logic, for specifying goals with this semantics. Our language is rich and allows the representation of a range of temporally extended goals, including classical goals, goals with temporal deadlines, quantified goals (with both universal and existential quantification), safety goals, and maintenance goals. Our formalism is simple and yet extends previous approaches in this area.

We also present a planning algorithm that can generate correct plans for these goals. This algorithm has been implemented, and we provide some examples of the formalism at work. The end result is a planning system which can generate plans that satisfy a novel and useful set of conditions.

Keywords: planning, temporal logic.

1 Introduction

One of the features that distinguishes *intelligent* agents is their flexibility: generally they have the ability to accomplish a task in a variety of ways. Such flexibility

*This work has been supported by the Canadian Government through their NSERC and IRIS programs. A preliminary version of this paper appears in *Proceedings of AAAI '96*, pp. 1215-1222.

is, of course, necessary if the agent is to be able to accomplish a range of tasks under varying conditions. Yet this flexibility also poses a problem: how do we communicate to such an agent the task we want accomplished in a sufficiently precise manner so that it does what we *really* want.

In the area of planning, methods and algorithms are studied by which, given information about the current situation, an intelligent agent can compose its primitive abilities so as to accomplish a desired task or goal. The afore mentioned problem then becomes the problem of designing sufficiently expressive and precise ways of specifying goals.

Much of the work in planning has dealt with goals specified as conditions on a final state. For example, we might specify a goal as a list of literals. The intent of such goals is that the agent should find a plan that will transform the current situation to a configuration that satisfies all of the literals in the goal. Any plan that achieves such a satisfying final state is deemed to be correct. However, there are two major limitations in the expressive power of such goals. First, not all goals are "final state" goals. For example, we might have goals of maintenance or reaction, where the agent must try to maintain a condition or respond within a limited time frame to a condition. Such goals can be important, but they cannot be specified as commands to reach a certain final state. Second, there are many important constraints we might wish to place on the agent's behavior that similarly cannot be expressed using final state semantics for goals. For example, if the agent's goal is to minimize disk usage, we might want to constrain it from deleting arbitrary files [WE94]. The importance of specifying such "safety" constraints on the agent's plans has been recognized. In particular, Weld and Etzioni [WE94] present strong arguments for looking beyond the simple achievement of a final state, and suggest two additional constraints on plans, a notion of *don't-disturb* and *restore*. However, these notions are only the tip of the iceberg; in general there are many other types of constraints that we may need to specify.

In this paper we present a richer formalism for specifying goals that borrows from work in verification [MP92], and develop a planning algorithm for generating plans to achieve such goals. Our formalism suggests a different way of viewing goals in planning. Instead of viewing goals as characterizing some set of acceptable final states and a plan as being correct if it achieves one of these states, we will view a goal as specifying a set of acceptable *sequences* of states and a plan as being correct if its execution results in one of these sequences. As we will show our formalism for goals subsumes the suggestions of Weld and Etzioni, except that instead of viewing *don't-disturb* and *restore* as constraints on plans, we view them as simply being *additional goals*.

Our formalism allows us to specify a wide range of *temporally extended goals*. This range includes, but is not limited to, classical goals of achieving some final state; goals with temporal deadlines; safety and maintenance goals like those

discussed by Weld and Etzioni and others [HH93]; and quantified goals (both universally and existentially quantified). Furthermore, our formalism is a logical language that carries with it a precise, and quite intuitive, semantics. The latter is important, as without a precise semantics for our goals we will not be able to analyze and verify exactly what task our agents will be accomplishing.

Logic has, of course, been previously used in work on planning. Included among the works using logic for planning has been Green's use of the situation calculus [Gre69], Rosenschein's use of dynamic logic [Ros81], and Bauer et al.'s use of temporal logic [BBD+91].[1] However, all of this work has viewed planning as a theorem proving problem. In this approach the initial state, the action effects, and the goal, are all encoded as logical formulas. Then, following Green, plans are generated by attempting to prove (constructively) that a plan exists. Planning as theorem proving has to date suffered from severe computational problems, and this approach has not yet yielded an effective planner.

Logic is used here in a completely different manner. In particular, we use the standard STRIPS approach of representing the initial state as a database of facts, and we use either the STRIPS or the ADL [Ped89] representations for action effects. Plan generation is accomplished by searching in spaces that are much more closely related to the structure of plans than is the more abstract space of proofs searched by theorem proving approaches. The logic is used solely to express goals and, as we will see later, search control information that can be used to guide search. Our approach uses tractable mechanisms for model checking logical formulas, rather than intractable mechanisms for generating proofs.

Temporally extended goals have previously been examined in the literature. Haddawy and Hanks [HH93] have provided utility models for some types of temporally extended goals. Kabanza et al. [Kab90, GK91, BKSD95] have developed methods for generating reactive plans that achieve temporally extended goals, as has Drummond [Dru89]. Planning systems and theories specifically designed to deal with temporal constraints (and sometimes other metric resources) have also been developed [Ver83, Wil88, AKRT91, CT91, Lan93, PW94].

The major difference between these previous works and what we present here, lies in our use of a logical representation. In particular, we use a temporal logic that supports a unique approach to computing plans, an approach based on formula progression. The method of formula progression lends itself naturally to the specification and utilization of domain dependent search control knowledge. As we will argue later, the approach of domain dependent search control offers considerable promise, and has motivated our approach to dealing with temporally extended goals. The other works that have constructed planners capable of dealing with temporally extended conditions have either utilized complex constraint

[1] In fact, the temporal logic used by Bauer et al. shares some of the features of the logic used here.

management techniques to deal with temporal information, e.g., the ZENO planner of Pemberthy and Weld [PW94], or permit only a limited range of temporally extended constructs, e.g., the specific extensions to SNLP suggested by Weld and Etzioni [WE94]. These mechanisms, unlike our logic, are not compositional. That is, special purpose modifications must be made to the underlying planning algorithm to accommodate each of the primitive "temporally extended" constructs used, and these constructs cannot be composed as can logical expressions. In our approach we develop a general purpose algorithm that can take as input any temporally extended goal expressible as a logical formula and, subject to computational limitations, find a plan satisfying that goal. The compositional features of our logical language allow us to express a much wider range of temporally extended goals than previous approaches.

The works by Kabanza cited above, are closer to our approach. In particular, he and his co-authors have utilized similar logics and similar notions of formula progression in their work. In this case the main difference is that here we address classical plans, i.e., finite sequences of actions, while Kabanza has concentrated on generating reactive plans, i.e., mappings from states to actions (sometimes called universal plans [Sch87]). Reactive plans have to specify an on-going interaction between an agent and its environment, and thus pose a quite distinct set of problems.

To generate plans that achieve the goals expressed in our formalism we present a planning algorithm that, as alluded to above, relies on the logical notion of goal progression. This notion has been utilized in our previous work [BK95]. In that work we examined the generation of plans for classical goals and used a simpler temporal logic to express search control information. We have implemented our algorithm as an extension to the TLPLAN system developed in that work. The planning algorithm is sound and complete, and it is able to generate a range of interesting plans.

In the rest of the paper we will first provide the details of the logic we propose for expressing goals. This logic is a temporal logic that is based on previous work by Alur et al. [AFH91]. We then present our approach to planning and provide examples to demonstrate the range of goals that our system can cope with. Finally, we close with some conclusions and discussion of future work.

2 Expressing goals in MITL

We use a logical language for expressing goals. The logic is based on Metric Interval Temporal Logic developed by Alur et al. [AFH91], but we have extended it to allow first-order quantification.

2.1 Syntax

We start with a collection of n-ary predicates (including equality and the 0-ary predicate constants TRUE and FALSE) and function symbols (with constants viewed as 0-ary functions), variables, and the Boolean connectives $\neg$ (not) and $\wedge$ (and). We add the universal and existential quantifiers $\forall$ and $\exists$ and the modal operators $\bigcirc$ (next) and U (until). From this collection of symbols we generate MITL, the language we use to express goals. MITL is defined by the traditional rules for generating terms, atomic formulas, and Boolean combinations, taken from ordinary first-order logic. In addition to these standard formula formation rules we add:

1. If ϕ is a formula then so is $\bigcirc\phi$.
2. If ϕ_1 and ϕ_2 are formulas and I is an interval then so is $\phi_1 \mathsf{U}_I \phi_2$. (The syntax of intervals is defined below).
3. If $\alpha(x)$ is an *atomic* formula in which the variable x is free, and ϕ is a formula then so are $\forall[x{:}\alpha(x)]\,\phi$, and $\exists[x{:}\alpha(x)]\,\phi$.

Notice that we have defined a special syntax for quantification. In particular, in our language we will use *bounded* quantification. The atomic formula α is used to specify the range over which the quantified variable ranges. Bounded quantification is used so as to make our implementation computationally feasible. The precise semantics of bounded quantifiers is given below.

The syntax of intervals is as one would expect. The allowed intervals are all intervals over the non-negative real line, and we specify an interval by giving its two endpoints, both of which are required to be non-negative numbers. To allow for unbounded intervals we allow the right endpoint to be ∞. Intervals can be either open, closed, or half-open half-closed, and we use the traditional symbols '[' and ']' to denote closed bounds and '(' and ')' to denote open bounds. For example, $[0, \infty)$ specifies the interval of numbers x such that $0 \leq x$, $(5.1, 6.1]$ specifies the interval $5.1 < x \leq 6.1$, and $[5, 5]$ specifies the interval $5 \leq x \leq 5$ (i.e., the point $x = 5$).

Although non-negative intervals are the only ones allowed in the formulas of MITL, in the semantics and algorithms we will need to utilize shifted intervals and to test for negative intervals. For any interval I let $I + r$ be the set of numbers x such that $x - r \in I$, $I - r$ be the set of numbers x such that $x + r \in I$, and $I < 0$ be true iff all numbers in I are less than 0. For example, $(5, \infty) + 2.5$ is the new interval $(7.5, \infty)$, $(0, 2) - 2.5$ is the new interval $(-2.5, -0.5)$, and $(-2.5, -0.5) < 0$ is true.

Finally, we introduce $\Rightarrow$ (implication), and $\vee$ (disjunction) as standard abbreviations. We also introduce the temporal modalities eventually $\Diamond$ and always

$\Box$ as abbreviations with $\Diamond_I \phi \equiv \text{TRUE}\ \mathsf{U}_I\ \phi$, and $\Box_I \phi \equiv \neg\Diamond_I \neg\phi$. We will also abbreviate intervals of the form (r, ∞) and $[0, r)$; e.g., $\Diamond_{(r,\infty)}$ will be written as $\Diamond_{>r}$ and $\Box_{[0,4]}$ as $\Box_{\leq 4}$. And we will often omit writing the interval $[0, \infty]$; e.g., we will write $\phi_1\ \mathsf{U}_{[0,\infty]}\ \phi_2$ as $\phi_1\ \mathsf{U}\ \phi_2$.[2]

2.2 The Intuitive Meaning of the Temporal Modalities

Intuitively, the temporal modalities can be explained as follows. The "next" modality $\bigcirc$ simply specifies that something must be true in the next state. Its semantics will not depend on the time of the states. It is important to realize, however, that what it requires to be true in the next state may itself be a formula containing temporal modalities. MITL gets its expressive power from its ability to nest temporal modalities.

The "until" modality is more subtle. In its untimed form $\phi_1\,\mathsf{U}\,\phi_2$ specifies that ϕ_1 must hold until ϕ_2 is achieved. When we introduce a timing constraint the condition to be achieved, ϕ_2, must additionally be achieved during the interval specified. The formula $\phi_1\ \mathsf{U}_{[5,7]}\ \phi_2$, for example, requires that ϕ_2 be true in some state whose time is between 5 and 7 units into the future, and that ϕ_1 be true in all states until we reach a state where ϕ_2 is true. The "eventually" modality will thus take on the semantics that $\Diamond_I \phi$ requires that ϕ be true in some state whose time lies in the interval I, and $\Box_I \phi$ requires that ϕ be true in all states whose time lies in I.

Now we present the formal semantics for these modalities.

2.3 Semantics

We intend that goals be expressed as sentences of the language MITL. As hinted in the introduction such formulas are intended to specify sets of sequences of states. Hence, it should not be surprising that the underlying semantics we assign to the formulas of MITL be in terms of state sequences.

A model for MITL is a *timed sequence of states*, $M = \langle s_0, \ldots, s_n, \ldots \rangle$. In particular, a model is an infinite sequence, and each state is a first-order model over a fixed domain D. That is, each state s_i assigns a denotation for each predicate and function symbol over the domain D. Furthermore, there is a timing function $\mathcal{T}$ that maps each state s_i in M to a point on the non-negative real line such that for all i, $\mathcal{T}(s_i) \leq \mathcal{T}(s_{i+1})$ and for all real numbers r there exists an i such that $\mathcal{T}(s_i) > r$. This means that time is only required to be non-decreasing, not strictly increasing. Time can stall at a single point for any finite number of states. Eventually, however time must increase beyond any fixed bound. The ability to have a sequence of states in which time is not advancing allows us to

[2]The temporal modalities with the interval $[0, \infty]$ correspond precisely to the traditional untimed modalities of Linear Temporal Logic [Eme90].

deal with concurrent actions.

Let V be a variable assignment, i.e., a mapping from the variables to elements of D; ϕ, ϕ_1, and ϕ_2 be formulas of MITL; and M be an MITL model. The semantics of MITL are then defined by the following clauses.

1. $\langle M, s_i, V\rangle \models \phi$, when ϕ is atemporal (i.e., contains no temporal modalities) and quantifier free, iff $\langle s_i, V\rangle \models \phi$.[3]
2. $\langle M, s_i, V\rangle \models \bigcirc\phi$ iff $\langle M, s_{i+1}, V\rangle \models \phi$.
3. $\langle M, s_i, V\rangle \models \phi_1 \mathsf{U}_I \phi_2$ iff there exists s_j, $j \geq i$ with $\mathcal{T}(s_j) \in I + \mathcal{T}(s_i)$ such that $\langle M, s_j, V\rangle \models \phi_2$ and for all s_k with $i \leq k < j$ we have $\langle M, s_k, V\rangle \models \phi_1$.
4. $\langle M, s_i, V\rangle \models \forall[x{:}\alpha(x)]\,\phi$ iff for all $d \in D$ such that $\langle s_i, V(x/d)\rangle \models \alpha(x)$ we have $\langle M, s_i, V(x/d)\rangle \models \phi$.
5. $\langle M, s_i, V\rangle \models \exists[x{:}\alpha(x)]\,\phi$ iff there exists $d \in D$ such that $\langle s_i, V(x/d)\rangle \models \alpha(x)$ and $\langle M, s_i, V(x/d)\rangle \models \phi$.

It is not difficult to show that any formula of MITL that has no free variables, called a sentence, has a truth value that is independent of the variable assignment V. Given a sentence ϕ of MITL we say it is true in a model M, $M \models \phi$, iff $\langle M, s_0\rangle \models \phi$.

Since sentences of MITL are either true or false on any individual timed sequence of states, we can associate with every sentence a set of sequences: those sequences on which it is true. We express goals as sentences of MITL, hence we obtain our desired semantics for goals: a set of acceptable sequences.

2.4 Discussion

An important property of these semantics is that they are what could be called "snapshot" semantics. They require that a state (a snapshot) exists that witnesses a particular property. Such a witness could fail to exist for two reasons: either the snapshot was taken at the right time but showed that the property was false, or no snapshot was taken (i.e., no state with the appropriate time exists in the sequence). For example, say that in the model M, $\mathcal{T}(s_0) = 0$ and $\mathcal{T}(s_1) = 2$, then it cannot be the case that $\langle M, s_0\rangle \models \Diamond_{[1,1]}\phi$ nor is it the case that $\langle M, s_0\rangle \models \Diamond_{[1,1]}\neg\phi$: no state exists with time 1.

Turning to the clauses for the bounded quantifiers we see that the range of the quantifier is being restricted to the set of domain elements that satisfy α. If α is true of all domain individuals, then the bounded quantifiers become equivalent to ordinary quantification. Similarly, we could express bounded quantification with

[3]Note that s_i is a first-order model, so the relationship "$\langle s_i, V\rangle \models \phi$" is defined according to the standard rules for first-order semantics.

ordinary quantifiers using the syntactic equivalences $\forall[x{:}\alpha(x)]\,\phi \equiv \forall x.\alpha(x) \Rightarrow \phi$ and $\exists[x{:}\alpha(x)]\,\phi \equiv \exists x.\alpha(x) \wedge \phi$. We have defined MITL to use bounded quantification because we will need to place finiteness restrictions on quantification when computing plans.

3 Planning

3.1 Planning Assumptions and Restrictions

Now we turn to the problem of generating plans for goals expressed in the language MITL. First we specify our assumptions.

1. We have as input a complete description of the initial state.
2. Actions preserve this completeness. That is, if an action is applied to a completely described state, then the resulting state will also be completely described.
3. Actions are deterministic; that is, in any world they must produce a unique successor world.
4. Plans are finite sequences of actions.
5. Only the agent who is executing the plan changes the world. That is, there are no other agents nor any exogenous events.
6. All quantifier bounds, i.e., the atomic formulas $\alpha(x)$ used in the definition of quantified formulas, range over a *finite* subset of the domain.

These assumptions allow us to focus on a particular extension of planning technology. They are essentially the same assumptions made in classical planning. For example, the assumption that actions preserve completeness is implied by the standard STRIPS assumption.

It is possible, however, to weaken some of these assumptions. For example, our approach could accommodate some degree of incompleteness. Incomplete state descriptions will suffice as long as they are complete enough to (1) determine the truth of the preconditions of every action and (2) determine the truth of all atemporal subformulas of the goal formula. The price that may have to be paid however is efficiency; instead of database lookup, theorem proving may be required to determine the truth of these two items. More conservative notions of incompleteness like locally closed worlds [EGW94] could be utilized in our framework without imposing a large computational burden.

Also, it should be made clear that restricting ourselves to deterministic actions does not mean actions cannot have conditional effects. In fact, the planner

we implemented handles full ADL conditional actions [Ped89] including actions with disjunctive and existentially quantified preconditions and actions that update metric quantities (functions).

3.2 Plan Correctness

Given a goal g expressed as a sentence of MITL we need to develop a method for generating plans that satisfy g. Sentences of MITL are satisfied by timed state sequences as described above. Hence, to determine whether or not a plan satisfies g we must provide a semantics for plans in terms of the models of MITL.

Since actions map states to new states, any finite sequence of actions will generate a finite sequence of states: the states that would arise as the plan is executed. Furthermore, we will assume that part of an action's specification is a specification of its duration, which is constrained to be non-negative. This means that if we consider s_0 to commence at time 0, then every state that is visited by the plan can be given a time stamp. Hence, a plan gives rise to a finite timed sequence of states—almost a suitable model for MITL.

The only difficulty is that models of MITL are infinite sequences. Intuitively, we intend to control the agent for some finite time, up until the time the agent completes the execution of its plan.[4]

Since we are assuming that the agent is the only source of change, once it has completed the plan the final state of the plan *idles*, i.e., it remains unchanged. Formally, we define the MITL model corresponding to a plan as follows:

Definition 1
Let plan P be the finite sequence of actions $\langle a_1, \ldots, a_n \rangle$. Let $S = \langle s_0, \ldots, s_n \rangle$ be the sequence of states such that $s_i = a_i(s_{i-1})$, and s_0 is the initial state. S is the sequence of states visited by the plan. Then the MITL model corresponding to *P and s_0 is defined to be $\langle s_0, \ldots, s_n, s_n, \ldots \rangle$, i.e., S with the final state s_n idled, where $\mathcal{T}(s_i) = \mathcal{T}(s_{i-1}) + \text{duration}(a_i)$, $0 < i \leq n$, $\mathcal{T}(s_0) = 0$, and the time of the copies of s_n increases without bound.*

Therefore, every finite sequence of actions we generate corresponds to a *unique* model in which the final state is idling. Given a goal expressed as a sentence of MITL we can determine, using the semantics defined above, whether or not the plan satisfies the goal.

[4]Work on reactive plans [BKSD95] and policies [DKKN93, TR94, BD94] has concerned itself with on-going interactions between the agent and its environment. However, there are still many applications where we only want the agent to accomplish a task that has a finite horizon, in which case plans that are finite sequences of actions can generally suffice.

Inputs: A state s_i, with formula label ϕ, and a time duration Δ to the successor state.
Output: A new formula ϕ^+ representing the formula label of the successor state.

Algorithm *Progress*(ϕ,s_i,Δ)
Case

1.	ϕ contains no temporal modalities:	
	if $s_i \models \phi$	$\phi^+ :=$ TRUE
	else	$\phi^+ :=$ FALSE
2.	$\phi = \phi_1 \wedge \phi_2$:	$\phi^+ := \mathit{Progress}(\phi_1, s_i, \Delta) \wedge \mathit{Progress}(\phi_2, s_i, \Delta)$
3.	$\phi = \neg\phi_1$:	$\phi^+ := \neg\mathit{Progress}(\phi_1, s_i, \Delta)$
4.	$\phi = \bigcirc\phi_1$:	$\phi^+ := \phi_1$
5.	$\phi = \phi_1 \mathsf{U}_I \phi_2$:	
	if $I < 0$	$\phi^+ :=$ FALSE
	else if $0 \in I$	$\phi^+ := \mathit{Progress}(\phi_2, s_i, \Delta) \vee (\mathit{Progress}(\phi_1, s_i, \Delta) \wedge \phi_1 \mathsf{U}_{I-\Delta} \phi_2)$
	else	$\mathit{Progress}(\phi_1, s_i, \Delta) \wedge \phi_1 \mathsf{U}_{I-\Delta} \phi_2$
6.	$\phi = \forall[x{:}\alpha]\,\phi_1$:	$\phi^+ := \bigwedge_{\{c:s_i \models \alpha(c)\}} \mathit{Progress}(\phi_1(x/c), s_i, \Delta)$
7.	$\phi = \exists[x{:}\alpha]\,\phi_1$:	$\phi^+ := \bigvee_{\{c:s_i \models \alpha(c)\}} \mathit{Progress}(\phi_1(x/c), s_i, \Delta)$

Table 1: The progression algorithm.

Definition 2
Let P be a plan, g be a goal expressed as a formula of MITL, s_0 be the initial state, and M be the model corresponding to P and s_0. P is a correct plan *for g given s_0 iff $M \models g$.*

3.3 Generating Plans

We will generate plans by adopting the methodology of our previous work [BK95]. In particular, we have constructed a forward-chaining planning engine that generates linear sequences of actions, and thus linear sequences of states. As these linear sequences of states are generated we *incrementally* check them against the goal. Whenever we can show that achieving the goal is impossible along a particular sequence, we can prune that sequence and all of its possible extensions from the search space. And we can stop when we find a sequence that satisfies the goal. The incremental checking mechanism is accomplished by the logical progression of the goal formula.

Formula Progression.
The technique of formula progression works by labeling the initial state with the sentence representing the goal, call it g. For each successor of the initial state,

generated by forward chaining, a new formula label is generated by *progressing* the initial state's label using the algorithm given in Table 1. This new formula is used to label the successor states. This process continues. Every time a state is expanded during planning search each of its successors is given a new label generated by progression.

Intuitively a state's label specifies a condition that we are looking for. That is, we want to find a sequence of states starting from this state that satisfies the label. The purpose of the progression algorithm is to update this label as we extend the state sequence. It takes as input the current state and the duration of the action that yields the successor state.

The logical relationship between the input formula and output formula of the algorithm is characterized by the following proposition:

Proposition 3
Let $M = \langle s_0, s_1, \ldots \rangle$ *be any MITL model. Then, we have for any formula* ϕ *of MITL,* $\langle M, s_i \rangle \models \phi$ *if and only if* $\langle M, s_{i+1} \rangle \models \mathit{Progress}(\phi, s_i, \mathcal{T}(s_{i+1}) - \mathcal{T}(s_i))$.

Proof
The proof of this proposition follows almost directly from the semantics of MITL formulas. To see how the argument proceeds consider case 1, when ϕ contains no temporal modalities. By the similar clause for the semantics of ϕ (Section 2.3) we see that $\langle M, s_i \rangle \models \phi$ iff $s_i \models \phi$. The progression algorithm produces as the progressed formula, ϕ^+, TRUE if $s_i \models \phi$ and FALSE otherwise. Since $\langle M, s_{i+1} \rangle \models$ TRUE and $\langle M, s_{i+1} \rangle \not\models$ FALSE, we see that the proposition holds in this case.

The only more complicated case is when $\phi = \phi_1 \mathsf{U}_I \phi_2$. Let $\Delta = \mathcal{T}(s_{i+1}) - \mathcal{T}(s_i)$. If $I < 0$ then we see that ϕ cannot be satisfied by any sequence starting at s_i: there can be no s_j with $j \geq i$ and $\mathcal{T}(s_j) \in I + \mathcal{T}(s_i)$ as time is non-decreasing along any sequence. Hence, time has expired and $\phi^+ =$ FALSE validates the proposition. If $0 \in I$ then ϕ will be true if ϕ_2 is true at the current time (which is always 0). That is, the current state falls into the interval where ϕ_2 must be true. Hence, by the semantics of U, ϕ will be true iff ϕ_2 is true now or ϕ_1 is true now and a time shifted version of ϕ is true in the next state. By induction ϕ_2 will be true now (i.e., in s_i) iff $\mathit{Progress}(\phi_2, s_i, \Delta)$ is true in the next state, and similarly for ϕ_1. To see the need for a time shifted version of ϕ, observe that if ϕ_2 must be true in the interval I, then when we move to the next state, which is Δ time units into the future, ϕ_2 must be true in the interval $I - \Delta$ Putting this together we obtain the formula ϕ^+ that must be satisfied in the next state. Finally, if $I > 0$ then whether or not ϕ_2 is true in the current state becomes irrelevant, as the current time does not fall into the required interval. In this final case, what we require is that ϕ_1 be true in the current state (i.e., $\mathit{Progress}(\phi_1, s_i, \Delta)$ be true in the next state), and a time shifted version of ϕ be true in the next state.

Inputs: A state s, and a formula ϕ.
Output: True if the state sequence $\langle s, s, \ldots \rangle$, where time increases without bound, satisfies ϕ. False otherwise.

Algorithm *Idle*(ϕ,s)
Case

1.	ϕ contains no temporal modalities:	
	if $s \models \phi$	**return** TRUE
	else	**return** FALSE
2.	$\phi = \phi_1 \wedge \phi_2$:	**return** $Idle(\phi_1, s) \wedge Idle(\phi_2, s)$
3.	$\phi = \neg\phi_1$:	**return** $\neg Idle(\phi_1, s)$
4.	$\phi = \bigcirc\phi_1$:	**return** $Idle(\phi_1, s)$
5.	$\phi = \phi_1 \mathsf{U}_I \phi_2$:	
	if $I < 0$	**return** FALSE
	else if $0 \in I$	**return** $Idle(\phi_2, s)$
	else	**return** $Idle(\phi_1, s) \wedge Idle(\phi_2, s)$
6.	$\phi = \forall[x{:}\alpha]\, \phi_1$:	**return** $\bigwedge_{\{c:s\models\alpha(c)\}} Idle(\phi_1(x/c), s)$
7.	$\phi = \exists[x{:}\alpha]\, \phi_1$:	**return** $\bigvee_{\{c:s\models\alpha(c)\}} Idle(\phi_1(x/c), s)$

Table 2: The idling algorithm.

The final two cases of the algorithm show that we handle quantification by rewriting it as a formula over the possible instantiations of the quantified variable. This is where we use our assumption that all quantification ranges over only a finite subset of the domain. □

Say that we label the start state, s_0, with the formula ϕ, and we generate new labels for the sequence of states $\langle s_0, s_1, \ldots, s \rangle$ up to the state s, using the progression algorithm. Furthermore, say we find a sequence of states, $S = \langle s, s^1, s^2, \ldots \rangle$, starting at state s that satisfies s's label. Then a simple induction using Proposition 3 shows that the sequence leading from s_0 to s followed by the sequence S, i.e., $\langle s_0, \ldots, s, s^1, s^2, \ldots \rangle$, satisfies ϕ. The progression algorithm keeps the labels up to date: they specify what we are looking for given that we have arrived where we are.

From this insight we can identify two important features of the formula progression mechanism. First, if we find any state whose idling satisfies its label, we have found a correct plan.

Proposition 4
Let $\langle s_0, s_1, \ldots, s_n \rangle$ be a sequence of states generated by forward chaining from the initial state s_0 to s_n. For each state s_i let its label be $\ell(s_i)$. Let the labels of the states be computed via progression, i.e., for each state s_i in the sequence

$$\ell(s_{i+1}) = \mathit{Progress}(\ell(s_i), s_i, \mathcal{T}(s_{i+1}) - \mathcal{T}(s_i)).$$

Then $M = \langle s_0, , \ldots, s_n, s_n, \ldots \rangle \models \ell(s_0)$ iff $\langle s_n, s_n, \ldots \rangle \models \ell(s_n)$.

Proof
By induction from Proposition 3. □

Since $\ell(s_0)$ is a formula specifying the goal, this proposition shows that if $\langle s_n, s_n, \ldots \rangle$ satisfies s_n's label then the plan leading to s_n satisfies the goal. Hence, if we have a method for testing for any state s and any formula $\phi \in$ MITL whether or not $\langle s, s, \ldots \rangle \models \phi$, we have a termination test for the planning algorithm that guarantees the soundness of the algorithm. We will describe an appropriate method below.

Furthermore, as long as the search procedure used by the algorithm eventually considers all finite sequences of states the planning algorithm will also be complete.

The second feature of formula progression is that it allows us to prune the search space without losing completeness. As we compute the progressed label we simplify it by processing all TRUE and FALSE subformulas. For example, if the label $\phi \wedge$ TRUE is generated we simplify this to ϕ. If any state receives the label FALSE we can prune it from the search space, thus avoiding searching any of its successors. From Proposition 3 we know that this label specifies a requirement on the sequences that start at this state. No sequence can satisfy the requirement FALSE; hence no sequences starting from this state can satisfy the goal, and this state and its successors can be safely pruned from the search space.

Termination.
As indicated above, we can detect when a plan satisfies the goal if we can detect when an idling state satisfies its label. This computation is accomplished by the algorithm given in Table 2.

Proposition 5
Idle(ϕ, s) returns TRUE *if and only if $\langle s, s, \ldots \rangle \models \phi$. That is, Idle detects if an idling state satisfies a formula.*

Proof
Again this proposition is easily proved from the definition of the semantics of MITL. For example, consider an until formula $\phi = \phi_1 \, \mathsf{U}_I \, \phi_2$. If it has not yet

Inputs: An initial state s_0, and a sentence $g \in$ MITL specifying the goal.
Returns: A plan P consisting of finite sequence of actions.
Algorithm *Plan*(g,s)

1. Open $\leftarrow \{(g, s_0)\}$.
2. **While** Open is not empty.
  2.1 $(\phi, s) \leftarrow$ Remove an element of Open.
  2.2 **if** *Idle*(ϕ, s) **Return** $((\phi, s))$.
  2.3 Successors $\leftarrow$ Expand(s).
  2.4 **For** all $(s^+, a) \in$ Successors
    2.4.1 $\phi^+ \leftarrow$ *Progress*$(\phi, s,$ *duration*$(a))$.
    2.4.2 **if** $\phi^+ \neq$ FALSE
      2.4.2.1 Parent$((\phi^+, s^+)) \leftarrow (\phi, s)$.
      2.4.2.2 Open $\leftarrow$Open $\cup \{(\phi^+, s^+)\}$.

Table 3: The planning algorithm.

timed out, it requires that ϕ_2 become true. Since the state s is idling, this means that s must itself satisfy ϕ_2. By induction this can be tested by calling *Idle*(ϕ_2, s). Similarly, if there is some time gap until the time that ϕ_2 must be satisfied then s must also satisfy ϕ_1, and this can be tested by calling *Idle*(ϕ_1, s). □

The Planning Algorithm.
Given the pieces developed in the previous sections we specify the planning algorithm presented in Table 3. The algorithm labels the initial state with the goal and searches among the space of state-formula pairs. We test for termination by running the *Idle* algorithm on the state's formula. To expand a state-formula pair we apply all applicable actions to its state component, returning all pairs containing a successor state and the action that produced that state (this is accomplished by Expand(s)). We then compute the new labels for those successor states using the *Progress* algorithm.

It should be noted that we *cannot* treat action sequences that visit the same state as being cyclic. If we are only looking for a path to a final state, as in classical planning, we could eliminate such cycles. Goals in MITL, however, can easily require visiting the same state many times. Nevertheless, we can view visiting the same state-formula pair as a cycle, and, using the standard techniques, we can optimize the search so as not to perform redundant expansions of the same state-

formula pair.[5] Intuitively, when we visit the same state-formula node we have arrived at a point in the search were we are searching for the same set of extensions to the same state. The planner we implemented performs this optimization.[6]

Proposition 6
The planning algorithm is sound and complete. That is, it produces a plan that is correct for g given s_0 (Definition 2), and so long as nodes are selected from Open in such a manner that every node is eventually selected, it will find a plan if one exists.

Proof
The soundness of our algorithm follows directly from the soundness of our termination test (Propositions 4 and 5). Completeness is also obvious given the completeness of the underlying search algorithm. The argument that we can optimize by eliminating state-formula cycles can also be proved formally. □

3.4 The implementation

We have implemented the planning algorithm as an extension of the TLPLAN system [Bac95]. This system provides a general forward chaining engine with a number of useful features. It requires some set of *described* predicates. All positive instances of the described predicates must be provided in the initial state, and the completeness of this description must be preserved by the actions.

With the described predicates in place the system implements an efficient first-order formula evaluator (utilizing a closed world assumption over the described predicates). By utilizing the complete descriptions of the described predicates, the evaluator is able to test if a state satisfies an arbitrary first-order formula without using theorem proving. Instead it employs model checking [HV91], in a process similar to query evaluation in databases. The formula evaluator is then used to provide a number of useful features.

First, additional predicates can be defined by recursive first-order formulas over the described predicates. For example, in the blocks world domain an *above* predicate can be defined by a recursive formula over the described *on* predicates. Furthermore, predicates that invoke arbitrary computations can be defined. These additional defined predicates can be used in the preconditions of actions and in

[5] For example, we can eliminate that node or search from it again if the new path we have found to it is better than the old path. These considerations will determine how we decide to set Parent((ϕ^+, s^+)) in step 2.4.2.1

[6] We have specified the algorithm so that *Progress* is run for each successor of the current state, as each successor might be generated by an action with a different duration. However, the implementation is able to avoid this and run *Progress* only once by producing a parameterized version of the progressed formula that is appropriate for labeling all of the successor states.

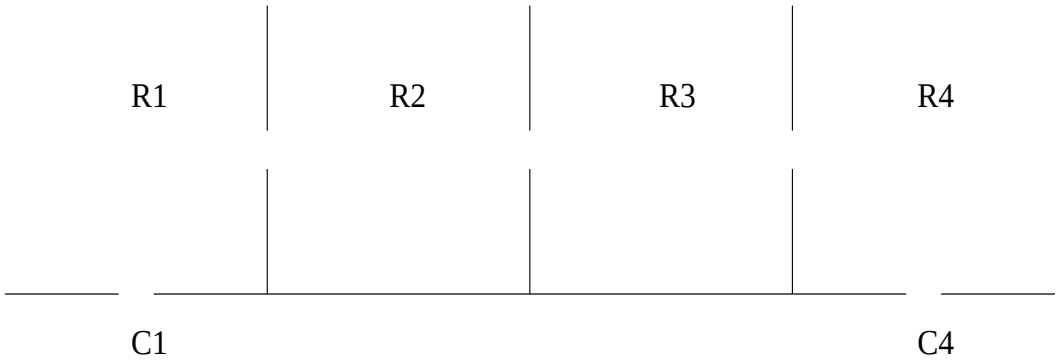

Figure 1: Robot Room domain

the goal formulas. Second, actions can be specified using the full ADL formalism. In particular, unlike UCPOP [PW92], the preconditions of actions can use disjunction and existential quantification. Finally, functions are fully supported. In particular, described functions can be defined that are updated by the ADL actions (see [Ped89] for details about function updates), and functions that invoke arbitrary computations can be defined and used in formulas.

The TLPLAN system was designed to plan for classical "achieve a final state" goals. A key component of this system is the ability to express search control knowledge as formulas of a temporal language. These formulas are progressed using an algorithm similar to *Progress*. However, the temporal language used is simpler and does not include the ability to specify timing constraints.

To implement our algorithm in the TLPLAN system, we altered the progression mechanism to make it suitable for the MITL language, and we implemented the *Idle* algorithm to test for termination. Both of these algorithms utilized TLPLAN's formula evaluator to allow the testing of an arbitrary first-order formula on a state (the progression and idle algorithms both bottom out on testing if a formula is true in the current state).[7]

[7]This is where the completeness assumptions mentioned in Section 3.1 come into play. Since we have complete descriptions TLPLAN can evaluate first-order formulas on a state using model checking. Less complete state descriptions might require some form of theorem proving.

Operator	Precondition	Adds	Deletes
$open(?d)$	$at(robot, ?x)$ $connects(?d, ?x, ?y)$ $closed(?d)$ $door(?d)$	$opened(?d)$	$closed(?d)$
$close(?d)$	$at(robot, ?x)$ $connects(?d, ?x, ?y)$ $opened(?d)$ $door(?d)$	$closed(?d)$	$opened(?d)$
$grasp(?o)$	$at(robot, ?x)$ $at(?o, ?x)$ $handempty$ $object(?o)$	$holding(?o)$	$handempty(?d)$
$release(?o)$	$holding(?o)$	$handempty$	$holding(?o)$
$move(?x, ?y)$	$at(robot, ?x)$ $connects(?d, ?x, ?y)$ $opened(?d)$	$at(robot, ?y)$ $holding(?o)$ $\Rightarrow at(?o, ?y)$	$at(robot, ?x)$ $holding(?o)$ $\Rightarrow at(?o, ?x)$

Table 4: Operators for Robot Room domain.

4 Example and Empirical Results

In this section we examine a simple example domain to illustrate the range and types of goals that can be expressed and planned for in our approach.

The domain we use is a variant of the classical STRIPS robot rooms domain [FN71]. The configuration of the rooms is illustrated in Figure 1. In this domain there are objects and a robot, which can be located at any of the 2 locations in the corridor, $C1$ or $C4$, or any of the 4 rooms $R1, \ldots, R4$. The robot can move between connected locations, it can open and close doors (indicated as gaps in the walls), and it can grasp and carry one object at a time. The operators that characterize its capabilities are shown in Table 4. In this table variables are preceded by a question mark "?". Also, the *move* operator is an ADL operator with conditional effects. For all objects that the robot is holding it updates their position. This is indicated in Table 4 by the notation $f_1 \Rightarrow \ell$ in the add and delete columns: the literal ℓ is added or deleted if f_1 holds. The duration of most of the actions is set to 1. Our implementation allows us to set the duration of an action to be dependent on the instantiation of its parameters. In particular, we set the duration of $move(x, y)$ to be 1, except for $move(C1, C4)$ which has duration 3.

Any initial state for this domain must specify the location of the robot and the existence and location of any objects in the domain. It must also specify whether each door is opened or closed. The doors connect the rooms to each other and to the corridor locations, and a set of *connects* relations must be specified,

e.g., $connects(D1, C1, R1)$. Door $D1$ connects the corridor location $C1$ and $R1$, door $D4$ connects $C4$ and $R4$, and the doors Dij connect rooms Ri and Rj $(i, j \in \{1, 2, 3\})$.

Finally, the two corridor locations are connected by a "*corridor*" which is always "open". So the literals $connects(corridor, C1, C4)$, and $opened(corridor)$, are also present in the initial state description.

4.1 Types of Goals

Classical Goals: Classical goals can easily be encoded as untimed eventualities that hold forever. For example, the classical goal $\{at(robot, C1), at(obj1, R4)\}$ expressed as a set of literals, can be encoded as the MITL formula

$$\Diamond\Box(at(robot, C1) \wedge at(obj1, R4)).$$

Any classical goal can be encoded in this manner. Given the semantics of plans as idling their final state, this formula will be satisfied by a plan if and only if the final state satisfies the goal.

More generally we can specify a classical "achieve a final state" goal by enclosing any atemporal formula of our language in an eventuality. We can specify disjunctive goals, negated conditions, quantified goals, etc. The formula

$$\Diamond(\exists[x{:}object(x)]\, at(x, R4) \vee at(robot, R4)),$$

for example, specifies the goal state where some object or the robot is in room $R4$.

Safety and Maintenance Goals: In [WE94] Weld and Etzioni discuss the need for safety conditions in plans. Such conditions have also been studied in the verification literature [MP92]. MITL can express a wide range of such conditions. Maintenance goals (e.g., [HH93]) which involve keeping some condition intact, are very similar.

Weld and Etzioni propose two specific constructions, *don't-disturb* and *restore*, as a start towards the general goal of expressing safety conditions. Both of these constructions are easily encoded as goals in MITL.

Don't-disturb specifies a condition $\phi(x)$. A plan is defined to satisfy a *don't-disturb* condition if during its execution no instantiation of $\phi(x)$ changes truth value. For example, we might constrain the robot to not disturb the open status of doors by setting $\phi(x)$ to $opened(x)$. This would prohibit plans that closed any doors.

Such conditions are easily specified by conjoining the formula $\forall x.\phi(x) \Rightarrow$

$\Box\phi(x)$ to the original goal.[8] For example, the goal

$$\Diamond\Box(at(robot, C1) \wedge at(obj1, R4)) \wedge \forall[x{:}opened(x)]\,\Box opened(x),$$

can only be satisfied by a plan that does not close any doors that were open in the initial state.

Restore also specifies a condition $\phi(x)$. A plan satisfies a *restore* condition if it tidies up after it has finished. That is, at the end of its plan it must append a new plan to restore the truth of all instantiations of $\phi(x)$ that held in the initial state.

We can specify *restore* goals in MITL by conjoining the formula $\forall x.\phi(x) \Rightarrow \Diamond\Box\phi(x)$, which specifies that the final state of the plan must satisfy all instantiations of ϕ that held in the initial state.[9] Notice that the semantic distinction between *restore* and *don't-disturb* goals is made clear by our formalism. *Restore* goals use $\Diamond\Box$ while *don't-disturb* goals use $\Box$. That is, *restore* goals allow the violation of ϕ during the plan, as long as these conditions are eventually restored in the final state.

Both of these conditions are limited special cases. MITL can express much more than this. For example, say that we want to constrain the robot to close doors that it opens. We cannot place a *don't-disturb* condition $closed(x)$, as this would prohibit the robot from moving into rooms where the doors are closed. If we specify this as a *restore* condition, the robot might leave a door opened for a very long time until it has finished the rest of its plan. In MITL, however in this domain we can use the formula

$$\begin{aligned}\Box(\forall[x, y, z{:}connects(z, x, y)]\, at(robot, x) \wedge closed(z) \wedge \bigcirc open(z) \\ \Rightarrow \bigcirc\bigcirc at(robot, y) \wedge \bigcirc\bigcirc\bigcirc closed(z))\end{aligned} \qquad (1)$$

This formula specifies that if the robot opens a closed door ($closed(z) \wedge \bigcirc(open(z))$), then it must go through the door ($\bigcirc\bigcirc at(robot, y)$) and then it must close the door ($\bigcirc\bigcirc\bigcirc closed(z)$). Hence, the robot is forced to be tidy with respect to doors: it only opens doors for the purpose of moving through them, and it closes the doors it opens behind it.

Timing Deadlines: MITL is also capable of expressing goals with timing conditions. For example $\Box_{\geq 10}\phi$ requires the condition ϕ be achieved within ten time units. Timed reactions are also possible. For example, $\Box(\phi \Rightarrow \Diamond_{\leq 5}\psi)$ requires that the condition ψ be achieved within five time units after condition ϕ becomes true. Note that ϕ and ψ can themselves be temporally extended conditions.

[8] We must appropriately rewrite $\forall x.\phi(x)$ in terms of bounded quantification, and use additional quantifiers if ϕ has multiple free variables. Similar remarks hold for encoding *restore*.

[9] When we add this formula as a conjunct to the original goal we force the planner to find a plan that satisfies the restore. If we want to give *restore* conditions lower priority, as discussed in [WE94], we could resort to the techniques of replanning suggested there.

4.2 Empirical Results

We have tested different goals from each of the categories mentioned above. All the plans were generated from similar initial states. In particular, in this state we have $at(obj1, R1)$, $at(obj2, R2)$, $at(robot, C1)$, $handempty$, $object(obj1)$, $object(obj2)$, and all of the doors are opened.

G1) From this initial state we set the goal to be

$$\Diamond\Box(at(robot, C1) \wedge at(obj1, R2)).$$

This corresponds to the classical goal $\{at(robot, C1), at(obj1, R2)\}$. The planner generates the following plan:

Time	Action	Time	Action
0	$move(C1, R1)$	3	$release(obj1)$
1	$grasp(obj1)$	4	$move(R2, R1)$
2	$move(R1, R2)$	5	$move(R1, C1)$

It took the planner 8.5 seconds, expanding 149 worlds to find this plan.[10]

G2) From the same initial state we set the goal to be

$$\Diamond\Box(\exists[x{:}object(x)]\, at(x, R3) \wedge handempty).$$

Now the planner generates the plan:

Time	Action	Time	Action
0	$move(C1, R1)$	3	$move(R2, R3)$
1	$move(R1, R2)$	4	$release(obj2)$
2	$grasp(obj2)$		

In this case it has generated a plan for a quantified goal by making an convenient choice for the object to place in $R3$. This plan took the planner 4.2 sec., expanding 89 worlds to find the plan.

G3) Now we change the initial state so all of the doors are closed, and we set the goal to be

$$\begin{aligned}&\Diamond\Box(at(robot, C1) \wedge at(obj1, R2))\\ \wedge\ &\Box(\forall[x, y, z{:}connects(z, x, y)]\, at(robot, x) \wedge closed(z) \wedge \bigcirc open(z)\\ &\qquad \Rightarrow \bigcirc\bigcirc at(robot, y) \wedge \bigcirc\bigcirc\bigcirc closed(z)\end{aligned}$$

[10] Timings were taken on a SPARC station 20, and a best first search strategy (exploring least cost plans first) was used so as to find the shortest plan. Finally, our planner tested for duplicate state-formula pairs during search.

This is simply a classical goal with an additional constraint on the robot to ensure it closes doors behind it. For this goal the planner generates the plan:

Time	Action	Time	Action
0	$open(D1)$	7	$release(obj1)$
1	$move(C1, R1)$	8	$open(D12)$
2	$close(D1)$	9	$move(R2, R1)$
3	$grasp(obj1)$	10	$close(D12)$
4	$open(D12)$	11	$open(D1)$
5	$move(R1, R2)$	12	$move(R1, C1)$
6	$close(D12)$	13	$close(D1)$

This plan took the planner 36 sec., expanding 313 worlds, to find.

G4) We reset the initial state to one where all of the doors are open and set the goal to be

$$\Box_{\geq 20}\, at(obj1, R4) \wedge \Box_{\geq 5}\, at(obj2, R3) \wedge \forall[x{:}\, opened(x)]\, \Box\, opened(x).$$

This is a goal with a tight deadline. The robot must move directly to $R2$ and move *obj2* to $R3$. If it stops to grasp *obj1* along the way it will fail to get *obj2* into $R3$ on time. Also we conjoin at subgoal of not closing any open doors. As we will discuss below this safety constraint acts as a form of search control, it stops the planner pursing useless (for this goal) *close* actions. The planner generates the plan:

Time	Action	Time	Action
0	$move(C1, R1)$	6	$move(R2, R1)$
1	$move(R1, R2)$	7	$grasp(obj1)$
2	$grasp(obj2)$	8	$move(R1, R2)$
3	$move(R2, R3)$	9	$move(R2, R3)$
4	$release(obj2)$	10	$move(R3, R4)$
5	$move(R3, R2)$		

This plan took the planner 2.9 sec., expanding 48 worlds, to find.

G5) If we change the time deadlines in the previous goal and set the goal to be

$$\Box_{\geq 9}\, at(obj1, R4) \wedge \Box_{\geq 20}\, at(obj2, R3) \wedge \forall[x{:}\, opened(x)]\, \Box\, opened(x).$$

The planner generates the plan:

Time	Action	Time	Action
0	$move(C1, R1)$	5	$release(obj1)$
1	$grasp(obj1)$	6	$move(R4, R3)$
2	$move(R1, R2)$	7	$move(R3, R2)$
3	$move(R2, R3)$	8	$grasp(obj2)$
4	$move(R3, R4)$	9	$move(R2, R3)$

It took the planner 15.0 sec. to find this plan, expanding 165 worlds on the way.

G6) Working from the same initial state we set the next goal to be

$$\Diamond_{[5,6]} at(obj1, R4) \wedge \Diamond\Box(at(obj1, R1) \wedge at(robot, C1).$$

The planner generates the plan:

Time	Action	Time	Action
0	$move(C1, R1)$	5	$move(R4, R3)$
1	$grasp(obj1)$	6	$move(R3, R2)$
2	$move(R1, R2)$	7	$move(R2, R1)$
3	$move(R2, R3)$	8	$release(obj1)$
4	$move(R3, R4)$	9	$move(R1, C1)$

It took the planner 132.0 sec. to find this plan, expanding 777 worlds on the way. Note that *obj1* is in fact in $R4$ at least once during the required time period $[5, 6]$, and that the final state is the same configuration as the initial state. This is an example of a temporally extended goal that requires visiting the same state twice.

Search Control

Although our planner can generate an interesting range of plans, problems remain. By itself it is not heuristically adequate for most planning problems. For example, when it is only given the goal of achieving some final state, it has to resort to blind search to find a plan. Similarly, it has no special mechanisms for planning for quantified goals, it simply searches until it finds a state satisfying the goal. Safety goals offer better performance, as such goals prune the search space of sequences that falsify them. This is why we included safety conditions on open doors in the fourth and fifth tests above: they allow the planner to find a plan faster. Again for goals with complex timing constraints, the planner does not utilize any special temporal reasoning.

As mentioned in the introduction, a major advantage of our approach lies in the ability of the planner to utilize domain dependent search control information. Such information can be expressed as formulas of MITL and conjoined with the goal. This approach to search control has been explored in our previous work [BK95], where we demonstrate some impressive results. In particular, we were able to construct *polynomial time* domain dependent planners for a range of domains, by adding domain specific control knowledge expressed in a temporal logic that was similar but simpler to MITL. No other approach to increasing the efficiency of planners, that we know of, has been able to produce polynomial time behavior in these domains.

To see how such control information could be utilize consider the following examples.

G1) Say that we modify **G1**, described above, by conjoining two clauses:

$$\begin{aligned}\Diamond\Box(&at(robot, C1) \wedge at(obj1, R2))\\ &\wedge \forall[o{:}object(o)]\, o \neq obj1 \Rightarrow \Box\neg holding(o)\\ &\wedge \forall[d{:}opened(d)]\, \Box\, opened(d)\end{aligned}$$

This new goal simply takes advantage of the fact that to achieve the goal of moving *obj1* into $R2$ it is not necessary to pickup any other object, nor is it necessary to close any doors.
The planner takes 0.65 sec. to generate the same plan as before, expanding only 20 worlds.

G2) Modifying **G2** by conjoining the clause

$$\forall[d{:}opened(d)]\, \Box\, opened(d),$$

since this goal does not require closing any doors, we generate the same plan as before in 0.64 sec., expanding 19 worlds.

G3) Modifying **G3** by conjoining the clause

$$\forall[o{:}object(o)]\, o \neq obj1 \Rightarrow \Box\neg holding(o),$$

since this goal also does not require picking up (or moving) any object besides *obj1*, we generate the same plan as before in 22 sec., expanding 217 worlds.

G6) Modifying **G6** by conjoining the clauses

$$\begin{aligned}&\forall[o{:}object(o)]\, o \neq obj1 \Rightarrow \Box\neg holding(o)\\ &\quad\wedge \forall[d{:}opened(d)]\, \Box\, opened(d),\end{aligned}$$

since this goal does not require moving any object besides *obj1* nor closing any doors, we generate the same plan as before in 5.2 sec., expanding 72 worlds.

To summarize, these simple control formulas generate the following speedups:

Example	Time	Worlds	New-Time	New-Worlds
1	8.5	149	0.65	20
2	4.2	89	0.64	19
3	36	313	22	217
6	132	777	5.2	77

The columns give the planning time and the number of worlds expanded, before and after we add the search control clauses. Note in particular, the speedups obtained on the harder problem **G6**. Furthermore, it should be noted that these are only the simplest and most obvious of control formulas for this domain.

5 Conclusions and Future Work

We have presented a clean and expressive formalism for expressing temporally extended goals. This formalism can be useful simply as a representation, no matter how one wants to compute plans. We have also demonstrated that the formalism has a direct computational interpretation, upon which we have constructed and implemented a planning algorithm.

As we have discussed, the key to making practical the approach to planning for temporally extended goals that we have advanced lies in utilizing search control information. This information, as shown in our examples, is domain dependent. Nevertheless, there are certain cases where control information can be automatically derived from the domain description. It will be the ability to automatically derive control formulas that will make our approach feasible.

Hence, a key component of our future work is to examine mechanisms for automatically generating and analyzing search control knowledge. There are a lot of interesting questions that arise when dealing with search control for quantified goals and goals with complex quantitative temporal constraints. One avenue we are pursuing is to statically analyze the goal formula and employ more sophisticated temporal reasoning *prior* to planning to construct control formulas. We hope to experiment with this approach and compare its efficiency with the approach of performing temporal constraint reasoning at every stage of plan generation (as is done in many other temporal planners, e.g., the ZENO system [PW94]).

References

[AFH91] Rajeev Alur, Tomas Feder, and Thomas Henzinger. The benefits of relaxing punctuality. In *Tenth Annual ACM Symposium on Principles of Distributed Computing (PODC 1991)*, pages 139–152, 1991.

[AKRT91] J. Allen, H. Kautz, Pelavin R., and J. Tenenberg. *Reasoning about Plans*. Morgan Kaufmann, San Mateo, CA, 1991.

[Bac95] Fahiem Bacchus. Tlplan (version 2.0) user's manual. Available via the URL ftp://logos.uwaterloo.ca:/pub/bacchus/tlplan-manual.ps.Z, 1995.

[BBD+91] M. Bauer, S. Biunod, D. Dengler, M. Hecking, J. Koehler, and Merziger G. Integrated plan generation and recognition—a logic-based approach. Technical Report RR-91-26, DFKI, 1991.

[BD94] Craig Boutilier and Richard Dearden. Using abstractions for decision-theoretic planning with time constraints. In *Proc. National Conference on Artificial Intelligence (AAAI '94)*, pages 1016–1022, Seattle, 1994.

[BK95] Fahiem Bacchus and Froduald Kabanza. Using temporal logic to control search in a forward chaining planner. In *Proceedings of the 3rd European Workshop on Planning*, 1995. Available via the URL ftp://logos.uwaterloo.ca:/pub/tlplan/tlplan.ps.Z.

[BKSD95] M. Barbeau, F. Kabanza, and R. St-Denis. Synthesizing plant controllers using real-time goals. In *Proc. Thirteenth International Joint Conference on Artificial Intelligence (IJCAI '95)*, pages 791–798, 1995.

[CT91] K. Currie and A. Tate. O-plan: the open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.

[DKKN93] T. Dean, L. P. Kaelbling, J. Kerman, and A. Nicholson. Planning with deadlines in stochastic domains. In *Proc. National Conference on Artificial Intelligence (AAAI '93)*, pages 574–579, 1993.

[Dru89] M. Drummond. Situated control rules. In *Proc. First International Conference on Principles of Knowledge Representation and Reasoning (KR '89)*, pages 103–113. Morgan Kaufmann, 1989.

[EGW94] O. Etzioni, K. Golden, and D. Weld. Tractable closed world reasoning with updates. In *Principles of Knowledge Representation and Reasoning: Proc. Forth International Conference (KR '94)*, pages 178–189, 1994.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, chapter 16, pages 997–1072. MIT, 1990.

[FN71] Richard Fikes and Nils Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[GK91] P. Godefroid and F. Kabanza. An efficient reactive planner for synthesizing reactive plans. In *Proc. National Conference on Artificial Intelligence (AAAI '91)*, pages 640–645, 1991.

[Gre69] Cordell Green. Application of theorem proving to problem solving. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 219–239, 1969.

[HH93] P. Haddawy and S. Hanks. Utility models for goal-directed decision-theoretic planners. Technical Report 93–06–04, University of Washington, 1993. Technical Report.

[HV91] J. Y. Halpern and M. Y. Vardi. Model checking vs. theorem proving: a manifesto. In J. A. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proc. Second International Conference (KR '91)*, pages 325–334. Morgan Kaufmann, San Mateo, CA, 1991.

[Kab90] F. Kabanza. Synthesis of reactive plans for multi-path environments. In *Proc. National Conference on Artificial Intelligence (AAAI '90)*, pages 164–169, 1990.

[Lan93] A. Lansky. Localized planning with diversified plan construction methods. Technical Report T.R. FIA-93-17, NASA Ames Research Center, 1993. Technical Report.

[MP92] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems:*

Specication. Springer-Verlag, New York, 1992.

[Ped89] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. First International Conference on Principles of Knowledge Representation and Reasoning (KR '89)*, pages 324–332, 1989.

[PW92] J. Scott Penberthy and Daniel Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Principles of Knowledge Representation and Reasoning: Proc. Third International Conference (KR '92)*, pages 103–114. Morgan Kaufmann, 1992.

[PW94] J. Scott Penberthy and Daniel Weld. Temporal planning with continuous change. In *Proc. National Conference on Artificial Intelligence (AAAI '94)*, pages 1010–1015. Morgan Kaufmann, 1994.

[Ros81] Stanley J. Rosenschien. Plan synthesis: A logical perspective. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 115–119, 1981.

[Sch87] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proc. Tenth International Joint Conference on Artificial Intelligence (IJCAI '87)*, pages 1039–1046, 1987.

[TR94] Jonathan Tash and Stuart Russell. Control strategies for a stochastic planner. In *Proc. National Conference on Artificial Intelligence (AAAI '94)*, pages 1079–1085, Seattle, 1994.

[Ver83] S. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 5, 1983.

[WE94] Daniel Weld and Oren Etzioni. The first law of robotics (a call to arms). In *Proc. National Conference on Artificial Intelligence (AAAI '94)*, pages 1042–1047, 1994.

[Wil88] D. Wilkins. *Practical Planning*. Morgan Kaufmann, San Mateo, CA, 1988.