Planning for Temporally Extended Goals*

Fahiem Bacchus
Dept. of Computer Science
University of Waterloo
Waterloo, Ontario
Canada, N2L 3G1

Froduald Kabanza
Dept. de Math et Informatique
Universite de Sherbrooke
Sherbrooke, Quebec
Canada, J1K 2R1

Abstract

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 as they 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.

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 is necessary if the agent is to be able to accomplish a variety of tasks under a range of 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 many important constraints we might wish to place on the agent's behavior that simply cannot be expressed using these semantics for goals. The importance of specifying such constraints on the agent's plans has been recognized. For example, 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*.

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 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. This latter is important, as without a precise semantics for our goals we will not be able to analyze and verify exactly what it is our agents will be accomplishing.

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]. Plan-

* This work was supported by the by the Canadian government through their NSERC and IRIS programs. Fahiem Bacchus also wishes to thank the University of Toronto for hosting his sabbatical leave during which much of this work was accomplished.

ning 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 main difference between these previous works and what we present here, lies in our use of 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 shown in our previous work [BK95], 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 temporal planners have utilized complex constraint management techniques to deal with temporal information.

In [Kab90, GK91, BKSD95] similar temporal logics and similar notions of formula progression have been utilized. In this case the main difference is that here we address classical plans, i.e., finite sequences of actions, while these works have concentrated on generating reactive plans, i.e., mappings from states to actions (sometimes called universal plans). 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 uses the logical mechanism of formula progression. This notion was previously utilized in our TLPLAN system [BK95]. In fact we have implemented the planning algorithm by extending the TLPLAN system. TLPLAN is planning system whose key feature is that it is able to utilize domain dependent search control information.. This control is expressed in a temporal logic that is a limited form of the logic presented here, and it is utilized by the planner via the mechanism of formula progression.

The planning algorithm we develop is sound and complete and as we will demonstrate it is able to generate a range of interesting plans. Further work is required, however, to evaluate the planner's performance on realistic planning problems.

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, provide examples to demonstrate the range of goals that our system can cope with, and discuss the heuristic adequacy of our approach to planning. Finally, we close with some conclusions and discussion of future work.

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 allow first-order quantification.

Syntax

We start with a collection of n-ary predicate (including equality and the predicate constants TRUE and FALSE) function and constant symbols, variables, and the connectives $\neg$ (not) and $\wedge$ (and). We add the 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 those 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); and (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 in our language we use *bounded* quantification. The atomic formula α is used to specify the range over which the quantified variable ranges. The precise semantics are 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 ∞. 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 $\Box_{\leq 4}$. Finally, we will often omit writing the interval $[0, \infty]$, e.g., we will write $\phi_1 \,\mathsf{U}_{[0,\infty]}\, \phi_2$ as $\phi_1 \,\mathsf{U}\, \phi_2$.[1]

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.

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

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 of states, 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 without bound.

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.

- $\langle M, s_i, V \rangle \models \phi$, when ϕ is atemporal (i.e., contains no temporal modalities) and quantifier free, iff $\langle s_i, V \rangle \models \phi$.[2]
- $\langle M, s_i, V \rangle \models \bigcirc \phi$ iff $\langle M, s_{i+1}, V \rangle \models \phi$.
- $\langle M, s_i, V \rangle \models \phi_1 \mathsf{U}_I \phi_2$ iff there exists s_j 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 $\langle M, s_k, V \rangle \models \phi_1$.
- $\langle M, s_i, V \rangle \models \forall [x{:}\alpha(x)]\, \phi$ iff for all $d \in D$ such that $\langle s_i, V(x/d) \rangle \models \alpha(x)$ we have $\langle M, s_i, V(x/d) \rangle \models \phi$.
- $\langle M, s_i, V \rangle \models \exists [x{:}\alpha(x)]\, \phi$ 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 of MITL, 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.

Discussion

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 do 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. 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 thus takes 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.

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 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 we do planning.

Planning

Planning Assumptions and Restrictions

Now we turn to the problem of generating plans for goals expressed in the language MITL. First we specify the assumptions we making. (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 as made in classical planning. For example, the assumption that actions preserve completeness is implied by the standard STRIPS assumption.

It is possible to weaken our assumptions of completeness. 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 is paid however is efficiency, instead of a database lookup, theorem proving may be required to determine the truth of these two items. However, 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.

Plan Correctness

Given a goal g expressed as a sentence of MITL we want to develop a method for generating plans that satisfy g. Sentences of MITL are satisfied by the timed state sequences 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.

[2] 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.

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 $s_i \models \phi$	ϕ^+	:= TRUE
	else	ϕ^+	:= FALSE
2.	$\phi = \phi_1 \wedge \phi_2$:	ϕ^+	:= *Progress*$(\phi_1, s_i, \Delta) \wedge$ *Progress*(ϕ_2, s_i, Δ)
3.	$\phi = \neg\phi_1$:	ϕ^+	:= $\neg$*Progress*(ϕ_1, s_i, Δ)
4.	$\phi = \bigcirc\phi_1$:	ϕ^+	:= ϕ_1
5.	$\phi = \phi_1 \mathsf{U}_I \phi_2$:		
	if $I < 0$	ϕ^+	:= FALSE
	else if $0 \in I$	ϕ^+	:= *Progress*(ϕ_2, s_i, Δ) $\vee$ (*Progress*$(\phi_1, s_i, \Delta) \wedge \phi_1 \mathsf{U}_{I-\Delta} \phi_2$)
	else		*Progress*$(\phi_1, s_i, \Delta) \wedge \phi_1 \mathsf{U}_{I-\Delta} \phi_2$
6.	$\phi = \forall[x{:}\alpha]\,\phi_1$:	ϕ^+	:= $\bigwedge_{\{c:\, s_i \models \alpha(c)\}}$ *Progress*$(\phi_1(x/c), s_i, \Delta)$
7.	$\phi = \exists[x{:}\alpha]\,\phi_1$:	ϕ^+	:= $\bigvee_{\{c:\, s_i \models \alpha(c)\}}$ *Progress*$(\phi_1(x/c), s_i, \Delta)$

Table 1: The progression algorithm.

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 greater than or equal to 0. 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. 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 $\mathcal{T}(s_i) = \mathcal{T}(s_{i-1}) + \mathit{duration}(a_i)$, $0 < i \leq n$, $\mathcal{T}(s_0) = 0$, 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.

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 $M \models g$.

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 $\langle M, s_{i+1}\rangle \models$ Progress$(\phi, s_i, \mathcal{T}(s_{i+1}) - \mathcal{T}(s_i))$.*

This proposition can easily be proved by utilizing the definition of MITL semantics.

Say that we label the start state, s_0, with the formula ϕ, and we generate new labels 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*

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

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*(ϕ_2, s)
3.	$\phi = \neg\phi_1$:	**return** $\neg$*Idle*(ϕ_1, s)
4.	$\phi = \bigcirc\phi_1$:	**return** *Idle*(ϕ_1, s)
5.	$\phi = \phi_1 \mathsf{U}_I \phi_2$:	
	if $I < 0$	**return** FALSE
	else if $0 \in I$	**return** *Idle*(ϕ_2, s)
	else	**return** *Idle*$(\phi_1, s) \wedge$ *Idle*(ϕ_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.

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

The proof of this proposition follows directly from Proposition 3.

Since $\ell(s_0)$ is a formula specifying the goal, this proposition shows that 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 soundness of the algorithm. We will describe an appropriate method below.

Furthermore, as long as the search procedure used by the algorithm eventually examines 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.*

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.

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 $\cup \{(s^+, \phi^+)\}$.

Table 3: The planning 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 optimize those cycles using the standard techniques.[3] 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.

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 correct plan if one exists.*

This proposition follows from the soundness of our termination test (Proposition 4).

We have implemented the planning algorithm as an extension of the TLPLAN system [Bac95]. This allowed us to utilize many of the features already built into the TLPLAN system, including full support of the ADL formalism [Ped89] for specifying actions.

Example and Empirical Results

Types of Goals

The domain we used 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

[3] 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$((\phi^+, s^+))$ in step 2.4.2.1

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.

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 literals of the form *connects*$(corridor, C1, C4)$, and *opened*$(corridor)$, must also be present in the initial state description.

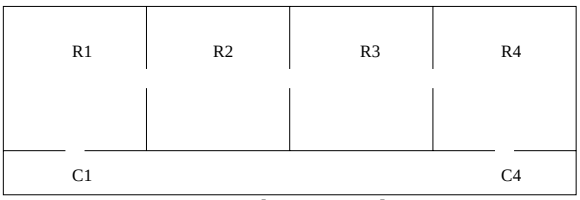

Figure 1: Robot Room domain

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 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. Such conditions are easily specified by conjoining the formula $\forall x.\phi(x) \Rightarrow \Box\phi(x)$ to the original goal.[4] 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 disturb any open doors.

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

[4] We must appropriately rewrite $\forall x.\phi(x)$ in terms of bounded quantification. Also it is not difficult to see that multiple variables in ϕ can be handled by additional quantifiers. Similar remarks hold for encoding *restore*.

in the initial state.[5] 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 we can use the formula

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

This formula specifies that if the robot opens a closed door ($\mathit{closed}(z) \wedge \bigcirc(\mathit{open}(z))$), then it must go through the door ($\bigcirc\bigcirc \mathit{at}(robot, y)$) and then it must close the door ($\bigcirc\bigcirc\bigcirc \mathit{closed}(z)$). 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.

Empirical Results

We have tested different goals from each of the categories mentioned above. Most of the plans were generated from the initial state in which *at*$(\mathit{obj1}, R1)$, *at*$(\mathit{obj2}, R2)$, *at*$(\mathit{robot}, C1)$, *handempty*, *object*$(\mathit{obj1})$, *object*$(\mathit{obj2})$, and all of the doors are opened.

G1: From this initial state we set the goal to be $\Diamond\Box(\mathit{at}(\mathit{robot}, C1) \wedge \mathit{at}(\mathit{obj1}, R2))$. This corresponds to the classical goal $\{\mathit{at}(\mathit{robot}, C1), \mathit{at}(\mathit{obj1}, R2)\}$. The planner generates the plan: *move*$(C1, R1)$, *grasp*$(\mathit{obj1})$, *move*$(R1, R2)$, *release*$(\mathit{obj1})$, *move*$(R2, R1)$, *move*$(R1, C1)$. It took the planner 22 sec., expanding 636 worlds to find this plan.[6]

G2: From the same initial state we set the goal to be $\Diamond\Box(\exists[x{:}\mathit{object}(x)]\, \mathit{at}(x, R3) \wedge \mathit{handempty})$. Now the planner generates the plan: *move*$(C1, R1)$, *move*$(R1, R2)$, *grasp*$(O2)$, *move*$(R2, R3)$, *release*$(O2)$. In this case it has generated a plan for a quantified goal. This plan takes the planner 3 sec., expanding 126 worlds to find the plan.

[5] 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.

[6] Timings are taken on a SPARC station 20, and a breadth first strategy was used so as to find the shortest plans.

G3: Now we change the initial state so all of the doors are closed. We set the goal to be $\Diamond\Box(\mathit{at}(robot, C1) \wedge \mathit{at}(\mathit{obj1}, R2))$ conjoined with Formula 1. 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 *open*$(D1)$, *move*$(C1, R1)$, *close*$(D1)$, *grasp*$(O1)$, *open*$(D12)$, *move*$(R1, R2)$, *close*$(D12)$, *release*$(O1)$, *open*$(D12)$, *move*$(R2, R1)$, *close*$(D12)$, *open*$(D1)$, *move*$(R1, C1)$, *close*$(D1)$. This plan took the planner 77 sec., expanding 1571 worlds, to find.

G4: We reset the initial state to one where all of the doors are open and set the goal to be $\Box_{\geq 20}\mathit{at}(\mathit{obj1}, R4) \wedge \Box_{\geq 5}\mathit{at}(\mathit{obj2}, R3) \wedge \forall[x{:}\mathit{opened}(x)]\, \Box\mathit{opened}(x)$. 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 a 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: *move*$(C1, R1)$, *move*$(R1, R2)$, *grasp*$(O2)$, *move*$(R2, R3)$, *release*$(O2)$, *move*$(R3, R2)$, *move*$(R2, R1)$, *grasp*$(O1)$, *move*$(R1, R2)$, *move*$(R2, R3)$, *move*$(R3, R4)$. This plan took the planner 8 sec., expanding 284 worlds, to find.

G5: If we change the time deadlines in the previous goal and set the goal it to be $\Box_{\geq 9}\mathit{at}(\mathit{obj1}, R4) \wedge \Box_{\geq 20}\mathit{at}(\mathit{obj2}, R3) \wedge \forall[x{:}\mathit{opened}(x)]\, \Box\mathit{opened}(x)$ The planner generates the plan: *move*$(C1, R1)$, *grasp*$(O1)$, *move*$(R1, R2)$, *move*$(R2, R3)$, *move*$(R3, R4)$, *release*$(O1)$, *move*$(R4, R3)$, *move*$(R3, R2)$, *grasp*$(O2)$, *move*$(R2, R3)$. It took the planner 120 sec. to find this plan, expanding 1907 worlds on the way.

Search Control

Although our planner can generate an interesting range of plans, by itself it is not efficient enough for practical 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.

The 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. We have explored this approach to search control in [BK95] where we demonstrate that is often possible to construct *polynomial time* planners using quite simple search control knowledge. We know of no other approach to increasing the efficiency of planners

that has been able to produce polynomial time behavior in these domains.

As a simple illustration of the power of this using search control consider the following trivial search control formula:

$$\Box\Big(\forall[x{:}\mathit{at}(\mathit{robot},x)]\,\neg(\bigcirc\neg\mathit{at}(\mathit{robot},x)\wedge\bigcirc\bigcirc\mathit{at}(\mathit{robot},x))$$
$$\wedge\,\forall[z{:}\mathit{object}(z)]\,\neg(\neg\mathit{holding}(z)\wedge\bigcirc\mathit{holding}(z)$$
$$\wedge\bigcirc\bigcirc\neg\mathit{holding}(z))\Big)$$

If we conjoin this formula with any other goal, the planner will prune sequences in which (1) the robot grasps an object and then immediately releases it, and (2) the robot moves away from a location and then immediately moves back. For this domain these sequences serve no purpose even in plans where the robot must visit the same state more than once.[7]

Conjoining this formula with the example goals given above we obtain the following speedups.

Example	Time	World	New-Time	New-Worlds
1	22	636	12	405
2	3	126	2	93
3	77	1571	18	304
4	8	284	1	38
5	120	1907	7.75	199

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

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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[7] In general, in order to achieve some timed goals we may need to allow the robot to wait. But, in that case it is more effective to introduce a specific wait action and still outlaw pointless cycles.