Search Control in Planning for Temporally Extended Goals

Froduald Kabanza
Département d'informatique
Université de Sherbrooke
Sherbrooke, Qc JIK2R1, Canada
kabanza@usherbrooke.ca

Sylvie Thiébaux
National ICT Australia &
The Australian National University
Canberra 0200, Australia
sylvie.thiebaux@anu.edu.au

Abstract

Current techniques for reasoning about search control knowledge in AI planning, such as those used in TLPlan, TALPlanner, or SHOP2, assume that search control knowledge is conditioned upon and interpreted with respect to a fixed set of goal states. Therefore, these techniques can deal with reachability goals but do not apply to temporally extended goals, such as goals of achieving a condition whenever a certain fact becomes true. Temporally extended goals convey several intermediate reachability goals to be achieved at different point of execution, sometimes with cyclic executions; that is, the notion of goal state becomes dynamic. In this paper, we describe a method for reasoning about search control knowledge in the presence of temporally extended goals. Given such a goal, we generate an equivalent Büchi automaton—an automaton recognising the language of the executions satisfying the goal—and interpret control knowledge over this automaton and the world state trajectories generated by a forward search planner. This method is implemented and experimented with as an extension of the TLPlan planner, which incidentally becomes capable of handling cyclic goals.

Introduction

Motivation

One of the most powerful approaches to coping with state-space explosion in planning is to provide the planner with knowledge of how to plan in specific application domains. Such search control knowledge can be acquired from human experts of the domain, much like models of primitive actions are. Interestingly, there is hope that both action models and search control knowledge will ultimately be learnt, removing the hurdle of relying on human experts to provide them, see e.g. (Pasula, Zuttlemoyer, & Pack Kaelbling 2004; Fern, Yoon, & Givan 2004) for recent works on these topics.

Over the last decade, a number of planners such as TLPlan (Bacchus & Kabanza 2000), TALPlanner (Kvarnström & Magnusson 2003), and SHOP2 (Nau *et al.* 2003), have successfully exploited the idea of using domain-specific control knowledge to guide a planner's search for a sequence of actions leading to a goal state. For instance, the forward search planner TLPlan provides a logic-based platform facilitating reasoning about search control knowledge in the form of temporal logic properties that promising plan prefixes must not violate. TLPlan prunes from the search any path leading to a plan prefix which violates these properties.

A key component of TLPlan's reasoning about search control knowledge is the GOAL modality: GOAL(f) means that the first-order formula f is true of all goal states. TALPlanner and SHOP2 have similar mechanisms to refer to goal properties. Using the GOAL modality and first-order bounded quantification, one can specify generic search control knowledge which is applicable to *any* problem in a given domain, without change. For instance, consider a health care robot which assists ederly or disabled people by achieving simple goals such as reminding them to do important tasks (e.g. taking a pill), entertaining them, checking or transporting objects for them (e.g. checking the stove's temperature or bringing coffee), escorting them, or searching (e.g. for glasses or for the nurse) (Cesta *et al.* 2003). In this domain, a useful piece of search control knowledge might be: "if a goal is to bring some object to some room, then once you grasp this object, keep it until you are in that room".

It is not possible to write control strategies that are both efficient and generic without using the GOAL modality, as such strategies need to be conditioned upon the goals. Yet, an important limit of current approaches is that the GOAL modality is interpreted with respect to a fixed set of goal states. This makes these approaches inapplicable to anything but simple goals of reaching a desired state. Such goals are called reachability goals.

The Problem

In this paper, we address the problem of extending the search control framework to planning with temporally extended goals (Bacchus & Kabanza 1998; Dal Lago, Pistore, & Traverso 2002). While reachability goals are properties of the final state of a finite sequence, the temporally extended goals we consider are general properties of potentially cyclic execution sequences. Examples include maintaining a property, achieving a goal periodically or within a number of steps of the request being made, and achieving several goals in sequence. In the health care robot domain for instance, we may want to specify goals such as "walk Mrs Smith to the bathroom, then get her a coffee, and if you meet the nurse on the way, tell her that I need her", or "get the house clean every morning, and during each meal remind me to take my pills". Although planning for temporally extended goals and planning for reachability goals are both PSPACE-complete in the deterministic fully observable setting we consider (De Giacomo & M. Y. Vardi 1999), in practice the former is often computationally more demanding than the latter. Therefore, good search control is crucial.

As it turns out, TLPlan uses the same formalism and technique to describe and process temporally extended goals and search control knowledge. This should not cause confusion: while the formalism and technique are the same, temporally extended goals and search control knowledge are fundamentally different in nature. The first is a property of the plans we want to generate, and the second is a property of the search (which incidentally in TLPlan, happens to be describing which plans the search should prune). In fact, TLPlan disables the use of search control when planning for temporally extended goals. The reason is, as we mentioned before, that when planning with such goals, there is no fixed set of goal states with respect to which the GOAL modality can be evaluated. Our purpose is to extend TLPlan's mechanism for reasoning about search control knowledge to make it operational even in the presence of temporally extended goals.

Assumptions

For simplicity, we remain in the framework of deterministic actions and, consequently, of temporally extended goals expressible in linear temporal logic (LTL). This is essentially the framework adopted by TLPlan with however one significant difference. TLPlan can only handle problems that can be solved by a finite, i.e. acyclic sequence of actions, while in the general case, temporally extended goals may require cyclic sequences. For instance, consider the goal "whenever the kitchen is dirty, get it clean, and when you are finished cleaning, get the next meal ready". This requires a cyclic plan of the form "do-for-ever: clean ; cook", since after the meal is cooked, the kitchen will be dirty again. Our approach will handle this general case.

For simplicity, we also remain in the framework of LTL to describe search control knowledge. In particular, we do not consider branching temporal logics such as CTL and variants (Pistore & Traverso 2001; Dal Lago, Pistore, & Traverso 2002; Baral & Zhao 2004), even though, as will become apparent, CTL would be useful to reason about what definitely is or just possibly is a useful goal to work on. Handling non-deterministic domains and more complex planning strategies is a topic for future work.

Overview of the Approach

Even under these assumptions, there are a number of ways we could go about specifying and handling control knowledge for planning under temporally extended goals. At one extreme, we could allow the GOAL modality to apply to temporal formulae. $\mathsf{GOAL}(f)$ would mean that the temporal formula f is derivable (in the LTL sense of the term) from the temporally extended goal g under consideration. One advantage of this approach is that the search control becomes highly expressive. It is possible to explicitly condition search control upon correct logical consequences of the temporally extended goal, and to prescribe farsighted strategies that take into account the behavior to be achieved in its globality. A drawback is that this approach requires performing LTL theorem proving for each formulae in the scope of the GOAL modality in turn, prior to invoking the planner.[1]

At another extreme, we could keep the language for specifying search control knowledge, and even the control knowledge itself, the same as in the reachability case, and analyse the temporal goal to dynamically generate, while planning, a series of reachability goals to be achieved in turn. The GOAL modality would then be evaluated with respect of the current reachability goal to be achieved. To guarantee completeness, backtracking on the reachability goals generated may be necessary. For instance, when analysing the temporally extended goal "walk Mrs Smith to the bathroom, then get her a coffee, and if you meet the nurse on the way, tell her that I need her", one could first generate the goal of getting Mrs Smith to the bathroom, then the goal of getting her the milk, temporarily preempting such goals with that of talking to the nurse when she is met on the current plan prefix. This option defers part of the search control decisions to the planner, since it chooses the successive reachability goals. This makes the task of the domain modeller much easier, but may lead to shortsighted search control strategies. In the above example, search control would not be able to exploit the fact that it is best to take Mrs Smith to the east-side bathroom which is nearer to the kitchen where the robot will need to go later to get the coffee.

As a middle ground, we could consider allowing temporal formulae inside the GOAL modality, and analyse the temporally extended goal to dynamically generate entire *sequences* of reachability goals, with respect to which the modality would be evaluated. That is, given such a sequence Γ, $\mathsf{GOAL}(f)$ would be true iff Γ models f in the LTL sense of the term. Again backtracking on the generated sequences would be necessary to ensure completeness. The main difference between this option and the first one is that search control decisions are not necessarily based on correct logical sequences of the original temporally extended goal.[2]

While all three approaches have their benefits and shortcomings, this paper focuses on the second. We describe a method which analyses the temporally extended goal and generates, during planning, successive reachability goals which are passed onto the traditional search control process. Backtracking occurs when an alternative goal appears more promising or when the current strategy does not result in a plan. The method is based on the construction of what is known in the verification literature as a Büchi automaton equivalent to the temporally extended goal (Wolper 1987). Such an automaton recognises the language of (cyclic) execution sequences that satisfy the goal. It suggests a range of possibilities for interpreting the GOAL modality, including but not limited to the three described above. Our method is incorporated to the TLPlan planner and evaluated on a number of example domains.

---

[1]A variant of this is to evaluate the GOAL modality with respect, not to the original temporally extended goal, but to the part of it that remains to be achieved. This affects the modelling style for search control. In that case, theorem proving would need to be performed at each node of the search space, and the computational cost would very likely outweigh the benefits of search control.

[2]Again, a variant exists where f only needs to be true of a suffix of the given goal sequence, as the current partial plan has completed the goals in the prefix.

Plan of the paper

The rest of the paper is organised as follows. The first section provides some background on LTL, temporally extended goals, search control knowledge, the formula progression technique at the core of the TLPlan planner, and Büchi automata. In the next section, we explain how the reachability goals are extracted from the Büchi automaton as planning proceeds, and give the extended TLPlan algorithm. We go on by presenting experimental results, and conclude with remarks on related and future work.

Background

Notations

We start with some notations. Given a possibly infinite sequence Γ, we let Γ_i, $i = 0, 1 \ldots$, denote the element of index i in Γ, and $\Gamma(i)$ the suffix $(\Gamma_i, \Gamma_{i+1}, \ldots)$ of Γ. $\Gamma'; \Gamma$ denotes the concatenation of the finite sequence Γ' and the sequence Γ. The infinite sequences we will consider will often be cyclic, of the form $\Gamma = \gamma; \gamma(j); \gamma(j); \ldots$ where γ is a finite sequence and j an index. That is, they consist of the finite sequence γ and then some suffix of γ repeated indefinitely. We will represent such cyclic sequences using the pair (γ, j).

First-Order Linear Temporal Logic

The language we use to express goals and search control knowledge is a first-order version of linear temporal logic. It starts with a first-order language containing collections of constant, function, and predicate symbols, along with variables, quantifiers, propositional constants $\top$ and $\bot$, and the usual connectives $\neg$ and $\wedge$. To these are added the unary GOAL modality, the unary temporal modality $\bigcirc$ (next) and the binary temporal modality U (until). Compounding under all those operators is unrestricted (in particular quantifying into temporal contexts is allowed), except that $\bigcirc$, U, and GOAL may not occur inside the scope of GOAL. In the following, we assume that all variables are bound.

Whereas, in the TLPlan planner, LTL is interpreted over finite[3] sequences of world states and a fixed set of goal states (Bacchus & Kabanza 2000), here we interpret the language over possibly infinite sequences Γ of pairs $\Gamma_i = (s_i, G_i)$ consisting each of a current world state s_i and a set of current goal states G_i. The idea is that at stage i of the sequence we are in world state s_i and are trying to reach one of the states in G_i. The temporal modalities allow the expression of general properties of such sequences. Intuitively, an atemporal formula f means that f holds now (that is, at the first stage Γ_0 of the sequence). $\bigcirc f$ means that f holds next (that is, f, which may itself contain other temporal operators, is true of the suffix $\Gamma(1)$). $f_1 \mathsf{U} f_2$ means that f_2 will be true at some future stage and that f_1 will be true until then (that is f_2 is true of some suffix $\Gamma(i)$ and f_1 must be true of the all the suffixes starting before Γ_i). $\mathsf{GOAL}(f)$ means that f is true in all the goal states G_i at the current stage, while an atemporal formula outside the scope of GOAL refers to the world state s_i at the current stage.

[3] More precisely, an infinite sequence of world states consisting of a finite sequence and an idling final state.

Formally, let D be the domain of discourse, which we assume constant across all world and goal states, and let Γ a sequence defined as above, with $\Gamma_i = (s_i, G_i)$. We say that a formula f is true of Γ, noted $\Gamma \models f$ iff $(\Gamma, 0) \models f$ where truth is defined recursively as follows:

- if f is an atomic formula, $(\Gamma, i) \models f$ iff $s_i \models f$ according to the standard interpretation rules for first-order logic
- $(\Gamma, i) \models \mathsf{GOAL}(f)$ iff for all $g_i \in G_i$, $g_i \models f$ according to the standard interpretation rules for first-order logic
- $(\Gamma, i) \models \forall x\, f$ iff $(\Gamma, i) \models f[x/d]$ for all $d \in D$, where $f[x/d]$ denotes the substitution of d's name for x in f.
- $(\Gamma, i) \models \neg f$ iff $(\Gamma, i) \not\models f$
- $(\Gamma, i) \models f_1 \wedge f_2$ iff $(\Gamma, i) \models f_1$ and $(\Gamma, i) \models f_2$
- $(\Gamma, i) \models \bigcirc f$ iff $(\Gamma, i+1) \models f$
- $(\Gamma, i) \models f_1 \mathsf{U} f_2$ iff there exists $j \geq i$ such that $(\Gamma, j) \models f_2$ and for all $k, i \leq k < j$, $(\Gamma, k) \models f_1$

In the following, where f is a formula not containing the GOAL modality and Γ is a sequence of world states rather than a sequence of pairs, we will abusively use the notation $\Gamma \models f$, since in that case the particular sequence of sets of goal states presented is irrelevant. We will make use of the usual abbreviations $\vee, \rightarrow, \exists$, as well as of the temporal abbreviations $\Diamond f \equiv \top \mathsf{U} f$ (eventually f) meaning that f will become true at some stage, and $\Box f \equiv \neg\Diamond\neg f$ (always f), meaning that f will always be true from now on. Where f is any formula and b is any atomic formula or any atomic formula inside a goal modality, we will also use the bounded quantifiers: $\forall[x : b(x)] f \equiv \forall x(b(x) \rightarrow f)$ and $\exists[x : b(x)] f \equiv \exists x(b(x) \wedge f)$.

Temporally Extended Goals

We represent temporally extended goals as LTL formulae which do not involve the GOAL modality or quantifiers (Bacchus & Kabanza 1998). These includes classical reachability goals of ending up in a state satisfying f as a special case; they can be expressed as $\Diamond\Box f$. Various useful types of temporally extended goals include goals of tidying up the environment to restore whatever property f was true at the start of the plan, i.e., $f \rightarrow \Diamond\Box f$, goals of non-disturbance of some property f that was true at the start of the plan, i.e. $f \rightarrow \Box f$, guards goals expressing that whenever condition c becomes true, property f must be achieved, i.e., $\Box(c \rightarrow \Diamond f)$, and goals of achieving properties f_1 and f_2 in sequence: $\Diamond(f_1 \wedge \bigcirc\Diamond f_2)$. For instance,

$$\begin{array}{l}(in(\mathsf{robot}, \mathsf{c1}) \wedge in(\mathsf{nurse}, \mathsf{c1}) \rightarrow \bigcirc talkto(\mathsf{nurse})) \\ \mathsf{U}\,(\Box with(\mathsf{Smith}) \wedge \bigcirc\Diamond(in(\mathsf{Smith}, \mathsf{bathroom}) \\ \quad \wedge \bigcirc\Diamond has(\mathsf{Smith}, \mathsf{coffee})))\end{array}$$

means that the robot should find Mrs Smith and stay with her while getting her to the bathroom and then arranging for her to get a cup of coffee, and that he should delivers a message to the nurse if he meets her on corridor c1 on the way to finding Mrs Smith.

$$\begin{array}{l}\Box(in(\mathsf{robot}, \mathsf{r1}) \rightarrow \Diamond in(\mathsf{robot}, \mathsf{r2})) \;\wedge \\ \Box(in(\mathsf{robot}, \mathsf{r2}) \rightarrow \Diamond in(\mathsf{robot}, \mathsf{r1}))\end{array}$$

means that the robot should continually move between room r1 and room r2, e.g., to check that everything is alright in those rooms.

We consider plans that are infinite, i.e., cyclic, sequences of actions—a finite sequence can easily be accommodated using a special *wait* action which does not change the world state. Any such a plan, Π, executed in world state s_0, induces an infinite sequence of world states Γ such that $\Gamma_0 = s_0$ and $\Gamma_{i+1} = \text{result}(\Pi_i, \Gamma_i)$. We say that a plan achieves a temporally extended goal f whenever $\Gamma \models f$. For instance, starting with $s_0 = \{in(\mathsf{robot}, \mathsf{r4})\}$, the second goal above would be satisfied by the cyclic plan:

$$((move(\mathsf{r4}, \mathsf{c2}), move(\mathsf{c2}, \mathsf{c1}), move(\mathsf{c1}, \mathsf{r2}), move(\mathsf{r2}, \mathsf{c1}), move(\mathsf{c1}, \mathsf{r1}), move(\mathsf{r1}, \mathsf{c1})), 2)$$

The TLPlan planner (Bacchus & Kabanza 1998) is originally aimed for non-cyclic plans, that is, plans of the form $(\gamma; wait, |\gamma|)$ where $|\gamma|$ is the length of γ. If a goal, such as the one above, can only be achieved by a cyclic plan, TLPlan reports failure to find a plan.

Search Control Knowledge

While temporally extended goals belong to the problem description and are properties of the plans we want to generate, search control knowledge belongs to the domain description and indicates to the search process how to plan in the domain. We also represent search control knowledge as LTL formulae, but this time involving the GOAL modality and quantifiers. For instance,

$$\Box\forall[(x, y) : \mathsf{GOAL}(in(x, y))]$$
$$holding(\mathsf{robot}, x) \rightarrow holding(\mathsf{robot}, x) \,\mathsf{U}\, in(\mathsf{robot}, y)$$

means that whenever getting object x at location y is part of the current goal, the robot, if it happens to be holding x, should hold on to it until he is at y. Note the use of bounded quantification here.

Here, as in the TLPlan planner (Bacchus & Kabanza 2000), the purpose of search control is to enable early pruning of finite plan prefixes. A plan prefix can be pruned as soon as it violates the control knowledge. For instance, with the above piece of search control knowledge, we would never expand a prefix in which the robot has released an object before getting to its goal location. Search control knowledge essentially consists of so-called *safety* properties, which are those expressing conditions that must be maintained, or in other words, conditions that are only violated by a finite prefix. Technically, such conditions are conveyed by formulae on the left-hand side of an U modality or formulae in the scope of a $\Box$ modality. In contrast, *liveness* properties express conditions that must be eventually made true, or in other words, conditions that can only be violated by a cyclic sequence. Technically, they are conveyed by formulae on the right-hand side of an U modality or formulae in the scope of a $\Diamond$ modality. While liveness properties are interesting from the purpose of specifying temporally extended goals—and we should indeed check that any cyclic plan we generate satisfy them—they are useless as far as control knowledge is concerned because they cannot be used for early pruning.

Our syntax allows arbitrary use of the GOAL modality inside the scope of temporal operators. So in principle, we could condition current search control decisions upon goals. However, this, or equivalently authorizing temporal modalities inside the scope of GOAL, is something we conceptually want to avoid with the particular approach described in this paper. Indeed, recall that we explore an approach whereby search control knowledge for temporally extended goals is simply implemented by means of interpreting traditional search control knowledge for reachability goals with respect to a suitable sequence of reachability goals generated by the planner.

We could easily restrict the language to both prevent the expression of liveness properties (e.g., by using the well-known 'weak' version of U), and conditioning of present decision on future goals (see e.g. (Slaney 2004) for both syntactic and semantic restrictions). However, we refrain doing so in order to stick to the general syntax adopted in TLPlan, and more importantly to keep the language open to more sophisticated reasoning about search control along the lines mentioned in the introduction of this paper.

Formula Progression

To use search control knowledge effectively, TLPlan must be able to check plan prefixes on the fly and prune them as soon as violation is detected. The key to achieve this is an incremental technique called formula *progression*, because it progresses or pushes the formula through the sequence Γ induced by the plan prefix. The idea behind formula progression is to decompose an LTL formula f into a requirement about the present Γ_i and which can be checked straight away, and a requirement about the future that will have to hold of the yet unavailable suffix. That is, formula progression looks at Γ_i and f, and produces a new formula fprog(Γ_i, f) such that $(\Gamma, i) \models f$ iff $(\Gamma, i+1) \models$ fprog(Γ_i, f). If Γ_i violates the part of f that refers to the present then fprog$(\Gamma_i, f) = \bot$ and the plan prefix can be pruned, otherwise, the prefix will be extended and the process will be repeated with Γ_{i+1} and the new formula fprog(Γ_i, f). In our framework, where world states are paired with sets of goal states in Γ, i.e., $\Gamma_i = (s_i, G_i)$, fprog(Γ_i, f) is defined as follows:[4]

Algorithm 1 Progression

fprog(Γ_i, f)	=	$\top$ iff $s_i \models f$ else $\bot$, for f atomic
fprog$(\Gamma_i, \mathsf{GOAL}(f))$	=	$\top$ iff $g_i \models f$ for all $g_i \in G_i$ else $\bot$
fprog$(\Gamma_i, \forall x\, f)$	=	fprog$(\bigwedge_{d \in D} f[x/d])$
fprog$(\Gamma_i, \neg f)$	=	$\neg$fprog(Γ_i, f)
fprog$(\Gamma_i, f_1 \wedge f_2)$	=	fprog$(\Gamma_i, f_1) \wedge$ fprog(Γ_i, f_2)
fprog$(\Gamma_i, \bigcirc f)$	=	f
fprog$(\Gamma_i, f_1 \,\mathsf{U}\, f_2)$	=	fprog$(\Gamma_i, f_2) \vee$ (fprog$(\Gamma_i, f_1) \wedge f_1 \,\mathsf{U}\, f_2$)

fprog runs in linear time in $|f| \times |G_i|$. Progression can be seen as a delayed version of the LTL semantics, which is useful when the elements of the sequence Γ become available one at a time, in that it defers the evaluation of the part

[4]Again, later on we abusively use simple world states as argument to fprog when it is clear that f does not contain GOAL.

of the formula that refers to the future to the point where the next element becomes available. However, since progression only applies to finite prefixes, it is only able to check safety properties, such as those involved in search control knowledge. In particular, it is unable to detect violation of liveness properties involved in temporally extended goals, as these can only be violated by an infinite sequence. Such properties will progress to $\top$when satisfied, but will never progress to $\bot$. For that reason, when dealing with temporally extended goals, we need an extra mechanism to check that liveness properties are satisfied.

Büchi Automaton Equivalent to a Goal

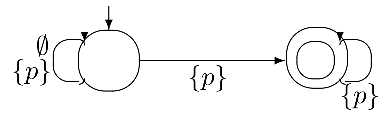
A Büchi automaton equivalent to the temporally extended goal (Wolper 1987) provides us with such a mechanism. A Büchi automaton is a nondeterministic automaton over infinite words. The main difference with an automaton over finite words is that accepting a word requires an accepting state to be reached infinitely often, rather than just once.

Formally a Büchi automaton is a tuple $B = (\Sigma, Q, \Delta, q_0, Q_F)$, where Σ is an alphabet, Q is a set of automaton states, $\Delta : Q \times \Sigma \mapsto 2^Q$ is a nondeterministic transition function, q_0 is the start state of the automaton, and Q_F the set of accepting states. A run of B over an infinite word $w = w_0 w_1 \ldots$ is an infinite sequence of automaton states $(q_0, q_1, \ldots)$ where $q_{i+1} = \Delta(q_i, w_i)$. The run is accepting if some accepting state occurs infinitely often in the sequence, that is, if for some $q \in Q_F$, there are infinitely many i's such that $q_i = q$. Word w is accepted by B if there exists an accepting run of of B over w, and the language recognised by B is the set of words it accepts.

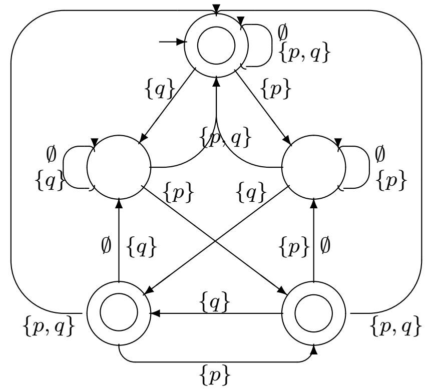
Given a temporally extended goal as an LTL formula f without GOAL modality and quantifiers, one can build a Büchi automaton recognising exactly the set of infinite world state sequences that satisfy f. The transitions in this automaton are labelled with world states, i.e., $\Sigma = 2^P$ where P is the set of atomic propositions in f. In the worst case, the number of states in the automaton is exponential in the length of f. It is often small however for many practical cases as illustrated in Figure 1. The many methods to build such automata differ by the size of the result and the time needed to produce it, see e.g. (Fritz 2003).

Formula progression can be seen as simulating parallel runs in the Büchi automaton. A useful image for a single run is that of the 'active' state of the automaton being updated when the next world state is processed; if no transition is possible at some point, no state becomes active, and the run is non-accepting. Now since the automaton is nondeterministic, imagine that several states are active at a time as a result of taking all transitions than can be taken at any given step. Each progression step can be viewed as updating the set of active states of the automaton when a new world state is processed, and progression to $\bot$ amounts to having no state left active. Formally, we can define a Büchi automaton version of progression, bprog, which looks at the current world state Γ_i and at the set of currently active automaton states Q_i and returns a new set of active automaton states, i.e. $\text{bprog}(\Gamma_i, Q_i) = \cup_{q \in Q_i} \Delta(q, \Gamma_i)$. Let f be a temporally extended goal, and B a Büchi automaton equivalent to f. Let furthermore $(f_0, f_1, \ldots)$ be the sequence of formulae obtained by successive progressions of f through the world state sequence Γ, i.e. $f_0 = f$ and $f_{i+1} = \text{fprog}(\Gamma_i, f_i)$ for all i, and let $(Q_0, Q_1, \ldots)$ be the sequence of active states obtained by Büchi progression through Γ in B, i.e., $Q_0 = \{q_0\}$ and $Q_{i+1} = \text{bprog}(\Gamma_i, Q_i)$ for all i. We have that for all i $Q_i = \emptyset$ iff $f_i = \bot$.

Figure 1 Simple Büchi automata examples



automaton for $f \equiv \Diamond\Box p$



automaton for $\Box(p \rightarrow \Diamond q) \wedge \Box(q \rightarrow \Diamond p)$

The Büchi automaton additionally makes it possible to check that a plan satisfies the liveness properties required by the temporally extended goal. Intuitively, for these liveness conditions to be satisfied, executing the plan must induce a sequence of world states that leads us to cycle through an accepting state of the automaton infinitely often. Formally, the cyclic plans $\Pi = (\pi, k)$ we will consider will induce a cyclic sequence of world states $\Gamma = (\gamma, k)$, with $|\pi| = |\gamma|$, and furthermore the finite sequence of active automaton states $(Q_0, \ldots, Q_{|\pi|})$ obtained by Büchi progression through γ in B will be such that $Q_{|\pi|} = Q_k$. A plan Π with these properties satisfies the goal iff 1) no Q_i is empty, and 2) there exists a $j \in \{k, \ldots, |\pi| - 1\}$ such that $Q_j \cap Q_F \neq \emptyset$. The first condition ensures that the safety properties are not violated, and the second that the liveness properties are satisfied.

Whereas the Büchi automaton is more powerful than formula progression, each have their preferred usage in our planner. For processing search control knowledge, which is a large collection of safety formulae, the linear-time formula progression procedure given in algorithm 1 is best suited. For full processing of temporally extended goals, which are rather short in comparison and additionally include liveness properties, it is worth investing time in building the Büchi automaton. This only needs to be done once, at the start of the planning episode. One can then resort to Büchi progression and checking for accepting cycles as described above. TLPlan (Bacchus & Kabanza 1998) never needs to resort to the Büchi automaton because it only handles the subset of temporally extended goals that admit finite plans. For such plans, a variant of formula progression can check for satisfaction of liveness properties. Below, we describe a method which also relies on the Büchi automaton to generate successive reachability goals with respect to which the GOAL modality is interpreted. These are taken as input by formula progression when processing search control knowledge.

Method

Intuition

Given a description of the domain (including the available actions and search control knowledge), an initial world state, and a temporally extended goal, our planner searches forward for a finite sequence of actions from which a cyclic plan satisfying the goal can be formed. In doing so, it uses Büchi progression to prune sequence prefixes which violate the safety requirements of the goal. It also uses formula progression to prune prefixes violating the control knowledge. In the latter, search control decisions are conditioned upon a current reachability goal, via the GOAL modality. We now explain how we use the Büchi automaton to dynamically generate, while planning, the successive reachability goals.

We view the states of the Büchi automaton as representing intermediate stages of satisfaction of the temporally extended goal, and the labels of the transitions as conditions to be reached to move from one stage to another. At any step of the search, our current search control *strategy* is represented by a transition[5] originating at an active state of the Büchi automaton. The label of that transition is our current reachability goal. We will often refer to that transition as the strategy's transition and to its origin state as the strategy's state. Because, in the automaton, the plan we are seeking must induce a cyclic run that includes an accepting state, our idea is to to guide the search towards such accepting states. The most direct way to achieve this, is to prefer strategies that are part of an *acyclic* path to an accepting state. That is, there should be a path to an accepting state starting with the strategy's transition, and that path should not go through the same state twice.

When the planner generates the next action and the new current world state, we update our strategy by choosing between the transitions available at the newly active automaton states, with the following preferences. Since when a strategy is selected, its label is used as the reachability goal to control search, we are hoping for the planner to produce a new world state satisfying this goal. If this happens, we prefer taking the strategy's transition in the Büchi automaton and selecting a new strategy originating at the the target state of that transition, again preferring strategies that are part of an acyclic path. Otherwise, as long as the world state produced by the planner matches the label of a transition back to the strategy's state, we prefer sticking to the current strategy, thereby giving more time to the planner to achieve the current goal. Otherwise we opportunistically choose a new strategy by selecting a transition originating at a new active state, again preferring strategies that are part of acyclic paths. We backtrack on the choice of the current strategy when it fails to lead to a plan.

For instance, consider the automaton for the cyclic goal $\Box(p \rightarrow \Diamond q) \wedge \Box(q \rightarrow \Diamond p)$ in Figure 1. Suppose that the only active state is the non-accepting state on the right-hand side. There are two transitions that are part of an acyclic path from that state to an accepting state: those labelled with $\{q\}$ and $\{p, q\}$, respectively. Suppose that we select $\{q\}$, this means that search control knowledge is evaluated with respect to the set of current goal states $G_i = \{s_i \mid s_i \models q \wedge \neg p\}$. As long as the current world state does not contain q, we stick to the current strategy. If the planner produces a world state that contains q but not p, our current strategy has succeeded and we take the transition to the bottom left-hand side accepting state. That state becomes our new strategy state. If on the other hand the planner had produced a world state that contains both p and q, we would opportunistically have taken the transition to the top accepting state. Naturally, when reaching an accepting state, things are not over yet, as the planner must still generate the other part of the cycle.

Algorithm

We now describe the details of our algorithm. In doing so, we use the macro CHOOSE to denote a backtracking choice point. We assume that this choice takes into account the preferences we mentioned before.

Our planning procedure is shown in Algorithm 2. The function PLAN takes as parameters a set A of planning operator descriptions, a search control formula f, an initial world state s_0, and a temporally extended goal g. It returns a cyclic plan achieving g or FAILURE to do so. Its main task is to initialise the data structures required by the search. In particular, it builds the Büchi automaton B for the goal g (line 2), and marks transitions in B that are part of some acyclic path to an accepting state (lines 3). This marking will be used to prefer marked transitions when choosing between strategies. Initially, the set $closed$ of closed nodes of the search is empty (line 4). The initial strategy (lines 5-6) consists of a transition t_0 chosen among the set $\text{TRANS}(B, q_0)$ of all transitions available at the initial state q_0 in B.

The function SEARCH takes as parameters the set A of operator descriptions, the Büchi automaton B, the search control formula f, the current plan prefix P_i (initially empty), and the current search node, characterised by the current world state s_i (initially s_0), the current formula f_i obtained by progression of the search control formula (initially f), the set Q_i of currently active automaton states obtained by Büchi progression (initially $\{q_0\}$), and the transition t_i representing the current strategy (initially t_0). The plan prefixes we consider are sequences of triplets (s_i, Q_i, a_i) consisting of the current world state, the currently active automaton states (this is often called the plan context (Pistore & Traverso 2001)), and the action to be performed.

After closing the current node (line 9), the first task of the SEARCH function is to check whether the current plan prefix can form a cyclic plan achieving the goal (lines 10-13). For this, it calls the function CYCLICPLAN, which takes the Büchi automaton B and the current plan prefix as parame-

[5] A triple: origin state, label, target state.

Algorithm 2 Planning Procedure

1. function PLAN(A, f, s_0, g)
2. $B \leftarrow$ BUILDAUTOMATON(g)
3. $B \leftarrow$ MARKACYCLIC(B)
4. $closed \leftarrow \emptyset$
5. $q_0 \leftarrow$ INITIAL(B)
6. $t_0 \leftarrow$ CHOOSE a transition in TRANS(B, q_0)
7. return SEARCH$(A, B, f, (), s_0, f, \{q_0\}, t_0)$
8. function SEARCH$(A, B, f, P_i, s_i, f_i, Q_i, t_i)$
9. $closed \leftarrow closed \cup \{(s_i, f_i, Q_i, t_i)\}$
10. $(cyclic, accepting, plan) \leftarrow$ CYCLICPLAN(B, P_i)
11. if $cyclic$ then
12. if $accepting$ then return $plan$
13. else return FAILURE
14. $f_{i+1} \leftarrow$ PROGRESSFORMULA$(s_i, f_i,$ LABEL$(B, t_i))$
15. if $f_{i+1} = \bot$ then return FAILURE
16. $Q_{i+1} \leftarrow$ PROGRESSAUTOMATON(s_i, B, Q_i)
17. if $Q_{i+1} = \emptyset$ then return FAILURE
18. $update \leftarrow$ UPDATE(B, Q_{i+1}, t_i)
19. if $update = \emptyset$ then return FAILURE
20. else $t_{i+1} \leftarrow$ CHOOSE a strategy in $update$
21. if $t_{i+1} \neq t_i$ then $f_{i+1} \leftarrow f$
22. $successors \leftarrow$ EXPAND$(s_i, A \cup \{wait\})$
23. if $successors = \emptyset$ then return FAILURE
24. $(a_i, s_{i+1}) \leftarrow$ CHOOSE a successor from $successors$
25. if $(s_{i+1}, f_{i+1}, Q_{i+1}, t_{i+1}) \in closed$ then
26. return FAILURE
27. $P_{i+1} \leftarrow P_i; (s_i, Q_i, a_i)$
28. return SEARCH$(A, B, f, P_{i+1}, s_{i+1}, f_{i+1}, Q_{i+1}, t_{i+1})$
29. function CYCLICPLAN$(B, ((s_0, Q_0, a_0), \ldots, (s_n, Q_n, a_n)))$
30. for $k = 0$ to $n - 1$
31. if $(s_k = s_n) \wedge (Q_k = Q_n) \wedge (a_k = a_n)$ then
32. for $j = k$ to $n - 1$
33. if $\exists q \in Q_j$ such that ISACCEPTING(B, q) then
34. return $(true, true, ((a_0, \ldots, a_{n-1}), k))$
35. return $(true, false, ((a_0, \ldots, a_{n-1}), k))$
36. return $(false, false, ((), 0))$

ters, and returns two booleans $cyclic$ and $accepting$ as well as a cyclic plan. $cyclic$ is true when a cyclic plan can be formed from the current plan prefix, i.e., when the last element of the plan prefix is identical to the the k^{th} element, for some k (lines 30-31). $accepting$ is true when the plan leads to an accepting cycle in the Büchi automaton, that is, when some active set Q_j in the loop ($j \geq k$) includes an accepting state (lines 32-35). If the current plan prefix is cyclic and accepting, then the search returns the corresponding cyclic plan (lines 10-12). If it is cyclic but not accepting, then it is a dead-end and FAILURE is returned (line 13). Otherwise, the search needs to expand the current prefix further.

This expansion can only take place if the prefix does not violate the control knowledge (lines 14-15). The function PROGRESSFORMULA takes as parameters the current world state s_i, the current search control formula f_i, and the label l_i of the current strategy's transition t_i, and checks that f_i successfully progresses through $\Gamma_i = (s_i, G_i)$ where $G_i = \{s \mid s \models l_i\}$. That is, it computes $f_{i+1} = \text{fprog}(\Gamma_i, f_i)$ and checks that $f_{i+1} \neq \bot$. It is also required that the prefix does not violate the safety requirements of the plan, i.e. the new set Q_{i+1} of active states of the Büchi automaton B must not be empty (lines 16-17). This is checked by the function PROGRESSAUTOMATON which takes as parameters the current world state s_i, the automaton B, and the current state of active states Q_i and computes the Büchi progression $Q_{i+1} = \text{bprog}(s_i, Q_i)$ in B.

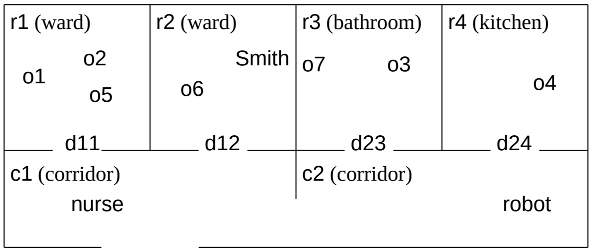
The next step is to update the strategy (lines 18-21). The function UPDATE takes as parameters the Büchi automaton B, the new set of active automaton states Q_{i+1}, and the current strategy t_i. It returns the strategies available at a state in Q_{i+1}. We assume that these are ordered in decreasing order of preference. As mentioned earlier, in our experiments, we have adopted the following preferences. Any strategy whose origin state is the target state of t_i, if any, should be ranked before strategies originating at any other state. Strategy t_i should be ordered next. Furthermore, subject to those constraints, marked strategies should rank before unmarked ones. Many other orderings and complementary preferences may make sense. When the newly selected strategy differs from the current one, we reset the search control (line 21). The reason is that we are using search control for reachability goals, and that the strategy change will result in guiding the planner towards a new reachability goal.

Finally, the current prefix is expanded (lines 22-28). The function EXPAND computes the set $successors$ pairs consisting of an action a_i applicable in the current world state s_i together with the resulting world state $s_{i+1} = \text{result}(a_i, s_i)$. (line 22). EXPAND explicitly considers the action of waiting without changing the world state, which is necessary to handle goals achievable by non-cyclic plans. One of the successors is chosen (lines 23-24). Provided that the resulting node has not already been closed (line 25-26), the plan prefix is expanded (line 27), and the search recurses with the new prefix and the new node (line 28).

Completeness

The algorithm is 'complete' in the following sense. If there exists a sequence of strategies $(t_0, \ldots)$ forming an accepting run of the Büchi automaton and a sequence of world states $(s_0, \ldots)$ such that the sequence $\Gamma = ((s_0, G_0), \ldots) \models f$, where f is the search control knowledge and G_i is the set of goal states satisfying t_i's label, then the algorithm will return a plan satisfying the temporally extended goal. This is because we potentially generate all such sequences, and these cover all possible runs of the automaton, including accepting ones. In particular, if the search control formula is $f \equiv \top$, the planner will find a plan satisfying the goal if one exists because we will not prune any solution from the search space. A slightly more useful completeness result would only assume the existence of the sequence $(G_0, \ldots)$ as a premise, instead of insisting that this sequence corresponds to a sequence of strategies extracted from the automaton. While such strategies have the nice property of representing successive reachability goals that the planner needs to achieve in order to satisfy the temporally extended goal, we have no guarantee that the control knowledge will not progress to $\bot$ with all of them while not progressing to $\bot$ with some other totally unmotivated goal sequence.

Figure 2 Health Care Domain Floor Map



With any approach relying on search control (and this is true of the original TLPlan algorithm), it is difficult to present more useful completeness results because the search control written by the domain modeller could well prevent finding any plan. The approach investigated in this paper exacerbates this somewhat, since it uses search control knowledge designed for reachability goals to solve problems with temporally extended goals. There is tension between the requirements of the temporally extended goals, and the requirement that search control knowledge for reachability goals prune as much of the search space as possible. There is no guarantee that these requirements be compatible.

We offer the experimental results in the next section as evidence that our approach, despite this tension, is useful in practice. For these experiments, we took domains from the TLPlan suite and did not alter the original search control knowledge at all. We took genuine examples of temporally extended goals we wanted to plan for, and in most cases the algorithm found a sensible plan in under a second. Without search control knowledge, the same algorithm and where applicable the original TLPlan algorithm were unable to find a plan in reasonable time for any but the smallest problems.

Experimental Results

We incorporated the above algorithm to the Scheme implementation of TLPlan. TLPlan has two modes. The *classic* mode handles reachability goals with search control knowledge, and the *temporal* mode handles non-cyclic temporally extended goals without search control knowledge. Our extension adds a *comprehensive* mode which handles both general temporally extended goals and search control knowledge. We performed experiments in the health care robot and blocks world domains, using MIT Scheme version 7.7 running on a Pentium 4, 2GHZ, 512MB RAM processor. All time performance results reported below are in CPU seconds (sec). The LTL to Büchi automaton translation relies on a library mapping typical formulas to corresponding Büchi automata, and causes negligible overhead. The experiments aim at demonstrating (1) the feasibility of the comprehensive mode, that is, the usability in the temporally extended goals context of search control knowledge designed for reachability goals, (2) its increased expressivity even compared to the temporal mode, i.e. in handling cyclic goals, and (3) its efficiency, that is, where such comparisons make sense, the performance gain compared to the temporal mode, and the overhead compared to the classic mode.

Domains

The blocks world domain we consider is the version with 4 operators (stack, unstack, pickup, putdown) and TLPlan's standard search control knowledge which conveys, as a function of the goal, most ways of avoiding building bad towers.

The health care robot domain is isomorphic to the robot room domain introduced in (Bacchus & Kabanza 1998), where a robot moves within the floor plan shown in Figure 2, carrying objects from room to room. The atomic propositions indicate the location of the robot and the objects, whether the robot is holding an object, and the positions (closed/open) of the doors. The planning operators are moving, opening or closing a door, and grasping or releasing an object. The control knowledge, taken as is from the domain specification for the classic mode, specifies (1) to keep door opened unless the goal states otherwise, (2) not to grasp objects unless they need to move or be held, (3) not to release objects until their destination is reached. The health care domain can be simulated by having objects play person roles (Mrs Smith, nurse) and having rooms with special functions (e.g., kitchen or bathroom). Since the domain is deterministic, people can only move when accompanied (held) by the robot. For instance, we can simulate $with(\mathsf{Smith})$ by $holding(\mathsf{Smith})$, $has(\mathsf{Smith}, \mathsf{Coffee})$ by $in(\mathsf{Smith}, \mathsf{r4})$, and $talkto(\mathsf{nurse})$ by $closed(\mathsf{d11})$.

Reachability Goals

From the initial state in Figure 2, we set the goal to be: $\Diamond\Box in(\mathsf{o1}, \mathsf{r2})$. This corresponds to the reachability goal $in(\mathsf{o1}, \mathsf{r2})$ and to the top Büchi automaton in Figure 1. In comprehensive mode, the planner generates the following plan in 0.04 sec:

$$((move(\mathsf{c2}, \mathsf{c1}), move(\mathsf{c1}, \mathsf{r1}), grasp(\mathsf{o1}), move(\mathsf{r1}, \mathsf{c1}), move(\mathsf{c1}, \mathsf{r2}), \mathsf{release}(\mathsf{o1}), \mathsf{wait}), 6)$$

The classic mode obtains the same plan in the same time, using the same control knowledge. This shows that our approach leads to little or no overhead with reachability goals. As expected, control knowledge leads to dramatic gains: the temporal mode, which has no control knowledge, was only able to generate a 2448 steps plan after 5.88 sec. When varying the size (number of rooms and objects) of the problem, the comprehensive and classic modes yield similar performances in all cases. These outweigh those of the temporal mode by an amount increasing with the problem size and ranging between one to two orders of magnitude for the above configuration with 1 to 10 objects.

Similar observations were made in the Blocks World domain. To illustrate, with the initial configuration

$$clear(\mathsf{g}) \wedge clear(\mathsf{e}) \wedge clear(\mathsf{c}) \wedge clear(\mathsf{a}) \wedge ontable(\mathsf{f}) \wedge ontable(\mathsf{e}) \wedge ontable(\mathsf{b}) \wedge ontable(\mathsf{a}) \wedge on(\mathsf{g}, \mathsf{d}) \wedge on(\mathsf{d}, \mathsf{f}) \wedge on(\mathsf{c}, \mathsf{b}) \wedge handempty$$

and the goal $on(\mathsf{d}, \mathsf{a}) \wedge on(\mathsf{c}, \mathsf{e}) \wedge on(\mathsf{e}, \mathsf{f}) \wedge on(\mathsf{f}, \mathsf{b})$ we obtain a plan in 0.16 sec both in the classic and comprehensive mode; the plan contains only the 12 actions needed to accomplish the task. This goal is beyond the capability of the temporally extended mode without search control.

Sequential Goals

Because there are few interactions among subgoals in the health care robot domain, it is much faster to plan for a sequence of deliveries of separate objects than for the union of these deliveries formulated as a single reachability goal. For example, it only takes 0.11 sec to the comprehensive mode to return a plan for the sequential goal:

$$\Diamond(in(\mathsf{o1},\mathsf{r2}) \wedge \bigcirc\Diamond(in(\mathsf{o2},\mathsf{r4}) \wedge \bigcirc(\Diamond in(\mathsf{o4},\mathsf{r2})))))$$

(the temporal mode, faced with the same problem, runs out of memory after 163 sec). In contrast, it took both the comprehensive and classic modes 0.19 sec, that is nearly twice as long, to generate a plan for the reachability goal:

$$\Diamond(in(\mathsf{o1},\mathsf{r2}) \wedge in(\mathsf{o2},\mathsf{r4}) \wedge in(\mathsf{o4},\mathsf{r2}))$$

Experiments with a range of delivery problems from the above initial configuration showed similar performances. Therefore, it seems that not only temporally extended goals allow additional expressivity of practical use (one often wants to specify in which order tasks need to be accomplished), but also can be more efficient in domains with few subgoal interactions.

In the blocks world, which has much richer subgoal interactions, it is no longer the case that subgoal sequences are easier to treat than the conjunction of the subgoals. In fact, sequenced subgoals are not even equivalent to their conjunction. For instance, the goal $\Diamond(on(\mathsf{a},\mathsf{b}) \wedge \bigcirc\Diamond on(\mathsf{c},\mathsf{a}))$ yields the same (optimal) plan as the goal $\Diamond(on(\mathsf{a},\mathsf{b}) \wedge on(\mathsf{c},\mathsf{a}))$ in the same 0.06 sec, but no less. Furthermore, the goal $\Diamond(on(\mathsf{c},\mathsf{a}) \wedge \bigcirc\Diamond on(\mathsf{a},\mathsf{b}))$ is not equivalent, since the plan, which is returned in 0.07 sec, is unable to preserve the first subgoal when achieving the second.

Reactive and Cyclic Goals

We experimented with more complex goals combining nested U and $\Box$ modalities. For instance, it takes 0.93 sec for the comprehensive mode to generate a plan for our goal example:

$$\begin{array}{l}(in(\mathsf{robot},\mathsf{c1}) \wedge in(\mathsf{nurse},\mathsf{c1}) \rightarrow \bigcirc talkto(\mathsf{nurse}))\\ \mathsf{U}\,(\Box with(\mathsf{Smith}) \wedge \bigcirc\Diamond(in(\mathsf{Smith},\mathsf{bathroom})\\ \quad \wedge\bigcirc\Diamond has(\mathsf{Smith},\mathsf{coffee})))\end{array}$$

The temporal mode, without search control, runs out of memory after 192 sec of computation.

We also experimented with cyclic plans in the comprehensive mode. Recall that those plans cannot be generated at all in temporal mode. For instance, consider the same initial configuration as in the figure except that the robot starts in $\mathsf{r1}$, and the goal:

$$\begin{array}{l}\Box(in(\mathsf{robot},\mathsf{r1}) \rightarrow \Diamond in(\mathsf{robot},\mathsf{r3}))\wedge\\ \Box(in(\mathsf{robot},\mathsf{r3}) \rightarrow \Diamond in(\mathsf{robot},\mathsf{r1}))\end{array}$$

The corresponding Büchi automaton is shown in Figure 1. The following plan is returned in 0.31 sec.

$$\begin{array}{l}((close(\mathsf{d11}), open(\mathsf{d11}), move(\mathsf{r1},\mathsf{c1}), move(\mathsf{c1},\mathsf{c2}),\\ move(\mathsf{c2},\mathsf{r3}), close(\mathsf{d23}), open(\mathsf{d23}), move(\mathsf{r3},\mathsf{c2}),\\ move(\mathsf{c2},\mathsf{c1}), move(\mathsf{c1},\mathsf{r1})), 0)\end{array}$$

Without search control (i.e., with search control set to $\top$), the comprehensive mode runs out of memory even in a simpler problem where no objects are present. As the presence in the plan of $close$ and $open$ actions illustrate, our heuristic for switching between strategies in the Büchi automaton is suboptimal. What is happening here is that a transition labelled with $\emptyset$ ends up being selected in the automaton. This, in turn, means that the control knowledge will progress to $\top$ (because GOAL modalities will evaluate to $\bot$), resulting in useless actions being allowed.

In a nutshell, based on these results from the blocks world and health care robot domains, our approach seems very promising. We generally obtain plans for reasonably complex goals in a matter of seconds, yet with an implementation in Scheme. Our approach of using search control knowledge originally designed for reachability goals worked well. Because the search control knowledge was quite conservative, there was never any conflict between this knowledge and the goal.

Conclusion, Future and Related Work

In this paper, we have argued that search control knowledge is an important component of the planning domain description. To work, every planning system already needs to be given some knowledge of the domain, e.g. that of preconditions and effects of primitive actions, and we consider it as natural and indeed useful to also provide rules of thumb on how to choose between those actions. Current planning systems condition search control decisions on properties of a desired final state and can therefore only use search control in conjunction with reachability goals. We have described one of the many possible approaches to the use of search control in conjunction with temporally extended goals.

This approach consists in interpreting search control with respect to successive reachability goals which are chosen from the labels of the transitions of a Büchi automaton for the temporally extended goal. One of its strengths is that search control knowledge originally written for simple reachability goals can be reused without change. Its main weakness is that such search control knowledge fails to take into account important aspects of temporally extended goals, such as the interaction between sequential reachability goals, and the interaction between reachability goals and safety properties.

This paper is by no means the final answer to the difficult problem of specifying and exploiting search control knowledge for temporally extended goals. There is potential for improving the simple method described here by investigating better heuristics for the selection of the reachability goals. An important aspect of future work is to experiment with alternative ways of evaluating the GOAL modality, including but not limited to those mentioned in the introduction of this paper. In particular, we would like to investigate the possibility of allowing CTL operators around the GOAL modality. The Büchi automaton for the LTL goal would then be the Kripke structure with respect to which to search control goals are evaluated. This would enable more complex conditioning of search control decisions, e.g., on the possibility that one reachability goal becomes our current reachability goal before another does. From then, another natural extension is to consider nondeterministic domains, for which CTL is needed in all aspects of search control.

Perhaps the most important item on our future work agenda is the design of a search control specification language which explicitly refers to temporally extended goals, while enabling a modular specification much as in the reachability case. The language of the Gapps compiler (Kaelbling 1988) can be seen as a primitive form of what we could aim at. Gapps considers symbolic reduction rules that map primitive goals such as $\Diamond p(x)$ and $\Box p(x)$ or composite goals (disjunction, conjunctions) to subgoals or actions. In our case, we need to consider more complex composite goals allowing the interleaving of temporal modalities.

Another aspect of future work is determining whether special forms or generalisations of Büchi automata could help reducing the complexity of search. For instance, a deterministic automaton would reduce the branching factor for our algorithm. While deterministic Büchi automata are strictly weaker than non-deterministic Büchi automata, it would be possible to build a deterministic ω-automaton with more complex acceptance conditions (e.g. a Rabin automaton) (Safra 1988). The worst-case complexity of doing so being explonential in the number of states, we need to investigate the tradeoff in using those or one of the many other generalisations of Büchi automata proposed in the literature as an alternative to our current approach.

Our work is largely orthogonal to previous planning research. While approaches for planning with temporally extended goals resort to similar mechanisms to those we use here (progression, variants of the Büchi automaton) (Bacchus & Kabanza 1998; Kabanza, Barbeau, & St-Dennis 1997; Dal Lago, Pistore, & Traverso 2002), and while the latter two also handle cyclic goals, they are not concerned at all with search control. In particular, TLPlan is unable to exploit search control when dealing with temporally extended goals and is restricted to acyclic plans.

Similar remarks apply to recent work on compiling LTL formulae into the classical planning framework (Rintanen 2000; Cresswell & Coddington 2004). The aim is to enable classical planners to solve problems involving either temporally extended goals, or search control knowledge. The compilation in (Rintanen 2000) for instance, results in classical planners effectively exploiting generic search control. However, the scheme is only adequate for reachability goals because the GOAL modality is still interpreted with respect to a final goal state. Our method could be applied in the framework of this compilation scheme to lift classical planners to handling search control for temporally extended goals. The compilation in (Cresswell & Coddington 2004), consists in letting the planning operators track the state of an automaton representing all reachable progressions of the temporally extended goal. This automaton, which is known as the 'local automaton' in the verification literature (Wolper 1987), does not deal with liveness properties. The approach is clearly aimed at (acyclic) temporally extended goals rather than search control. In particular the translation does not cope with quantification and with the GOAL modality which are essential components of search control.

Acknowledgements

The authors thank John Slaney for helpful discussions. This work was initiated while Froduald Kabanza was visiting the Australian National University. He is grateful for the support received from ANU for this visit. He is also supported by the Canadian Natural Sciences and Engineering Research Council (NSERC). Sylvie Thiébaux thanks National ICT Australia (NICTA) and the Australian Research Council (ARC) for their support. NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the ARC.

References

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22:5–27.

Bacchus, F., and Kabanza, F. 2000. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence* 116(1-2).

Baral, C., and Zhao, J. 2004. Goal specification in presence of nondeterministic actions. In *Proc. ECAI*.

Cesta, A.; Bahadori, S.; G, C.; Grisetti, G.; Giuliani, M.; Loochi, L.; Leone, G.; Nardi, D.; Oddi, A.; Pecora, F.; Rasconi, R.; Saggase, A.; and Scopelliti, M. 2003. The robocare project. cognitive systems for the care of the elderly. In *Proc. International Conference on Aging, Disability and Independence (ICADI)*.

Cresswell, S., and Coddington, A. 2004. Compilation of LTL goal formulas into PDDL. In *Proc. ECAI*.

Dal Lago, U.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *Proc. AAAI*.

De Giacomo, G., and M. Y. Vardi. 1999. Automata-theoretic approach to planning for temporally extended goals. In *Proc. ECP*.

Fern, A.; Yoon, S.; and Givan, R. 2004. Learning domain-specific knowledge from random walks. In *Proc. ICAPS*.

Fritz, C. 2003. Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In *Proc. International Conference on Implementation and Application of Automata*.

Kabanza, F.; Barbeau, M.; and St-Dennis, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence* 95:67–113.

Kaelbling, L. 1988. Goals as parallel program specifications. In *Proc. AAAI*.

Kvarnström, J., and Magnusson, M. 2003. TALplanner in IPC-2002: Extensions and Control Rules. *Journal of Artificial Intelligence Research* 20:343–377.

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdoch, J.; Wu, D.; and Yaman, F. 2003. SJOP2, an HTN Planning System. *Journal of Artificial Intelligence Research* 20:379–404.

Pasula, H.; Zuttlemoyer, L.; and Pack Kaelbling, L. 2004. Learning probabilistic relational planning rules. In *Proc. ICAPS*.

Pistore, M., and Traverso, P. 2001. Planning as model-checking for extended goals in non-deterministic domains. In *Proc. IJCAI*.

Rintanen, J. 2000. Incorporation of temporal logic control into plan operators. In *Proc. ECAI*.

Safra, S. 1988. On the complexity of w-automata. In *Proc. FOCS*.

Slaney, J. 2004. Semi-positive LTL with an uninterpreted past operator. *Logic Journal of the IGPL*. To appear.

Wolper, P. 1987. On the relation of programs and computations to models of temporal logic. In *Proc. Temporal Logic in Specification, LNCS 398*, 75–123.