Synchronizing Multiagent Plans using Temporal Logic Specifications*

Froduald Kabanza
Université de Sherbrooke
Sherbrooke, Québec, J1K 2R1, Canada
Email: kabanza@dmi.usherb.ca

Abstract

Plan synchronization is a method of analyzing multiagent plans to introduce ordering constraints between them so that their concurrent execution achieves a desired goal. We describe a plan synchronization method for goals expressed using temporal logic specifications. Our goals can involve both qualitative and quantitative time requirements. The requirements expressed by a goal can involve safety constraints as well as liveness ones. The key to our method is a technique for checking goal formulas incrementally over models of concurrent executions of plans. Our approach covers more general goal types than any comparable method. It also promises easy integration with standard AI planning search control and heuristic strategies.

Introduction

Multiagent plan synchronization is a method of analyzing plans of concurrent agents in order to add synchronization constraints between them such that their concurrent execution satisfies a given temporal property (Stuart 1985; Shoham & Tennenholtz 1994). We describe a method for synchronizing multiagent plans from goals described by Metric Temporal Logic (MTL) formulas (Koymans 1990; Alur & Henzinger 1993). MTL is an extension of Linear Temporal Logic (LTL) (Emerson 1990), with explicit time constraints on the temporal modalities *eventually*, *always*, and *until*.

The idea of applying modal temporal logic to synchronize multiagent plans is not new. In particular, Stuart (Stuart 1985) implemented a multiagent plan synchronizer based on a *theorem proving* method developed by Manna and Wolper (Manna & Wolper 1984). In this approach, the goal and the plans to be synchronized are both expressed by an *extended* LTL formula, and then a decision procedure of *extended* LTL is used to obtain a model satisfying this formula and from which a plan can be extracted. As a matter of fact, the planner described in (Stuart 1985) accepts plans specified using an operational language; however, they are replaced with *extended* LTL formulas during the initialization of the planning process. Note that this approach can be easily extended to synchronize plans using MTL specifications by applying the MTL decision procedure.

We describe an alternative method that uses *model checking*. Our method operates explicitly on plans described operationally without translating them into declarative formulas. Specifically, we use plans to generate models that are validated by comparison to the given MTL goal formulas. The processes of generating models from plans and checking MTL goal formulas are concurrent and incremental. Our approach's main advantage is separating the actions of the agents and the goal throughout the planning process. This facilitates plan debugging. Moreover, as we discuss later, this seems more promising in terms of controlling the search process of the planner.

Note that a primitive action can be treated as an atomic plan. If the input of our plan synchronization method consists solely of the possible primitive actions of the agents, then the plan synchronization task reduces to a plan synthesis one. Classical multiagent planning systems are based on a final-state notion. In such systems, an agent executes a sequence of actions in order to reach a given goal state (Lansky 1988; Georgeff 1987; Kabanza 1990). Our framework is very similar to reactive planning (Godefroid & Kabanza 1991; Drummond & Bresina 1990; Shoham & Tennenholtz 1994) in that our agents tend much more to reactively invoke reaction rules based on their current state than to follow a preset action sequence.

In fact, we view each agent in a concurrent system as continuously responding to other agents in the system or external to it. Such agents generally have arbitrary long, ideally nonterminating, executions. For instance, an air-traffic control system can be seen as

*This work was supported by the Canadian Government through their NSERC projects.

formed of software agents that continuously *sense* the arrival of flights in order to schedule the landings. A distributed database system can also be considered as consisting of software agents that continuously communicate between each other or with an external user. The behaviors of these kind of agents are quite often characterized by *liveness goals* which must be satisfied by each infinite execution of the system. Hence there is a need of handling liveness goals appropriately in planners for such multiagent systems. Contrary to the previous approaches for synthesizing multiagent plans, our framework treats liveness requirements that are explicitly specified in the goal.

In this paper, we assume that the coordination between agents in a multiagent system is handled by one central coordinator, that is, the plan synchronizer. This synchronizer has global knowledge of the entire multiagent system from which it generates individual synchronized plans for various agents. It should be noted that such pure central coordination is not, however, a realistic approach. As has been discussed by many authors, including Shoham and Tennenholtz (Shoham & Tennenholtz 1994), a more powerful framework would integrate central coordination with other techniques, such as a negotiation mechanism or safety conventions (also called *social laws* by Shoham and Tennenholtz). We will discuss these extensions in the conclusion.

Although our approach applies to general MTL goals and other real-time temporal logic goals, due to space limitations, we will describe how it works only for a special class of MTL formulas called *bounded-time MTL formulas*. We describe our approach by proceeding in four steps, starting with a brief discussion of our model of plan and action. We then give a brief review of MTL. Finally we describe our plan synchronization method, and conclude with some discussions.

Plans and Actions

We view a multiagent system as consisting of a finite number of concurrent agents, each executing actions that are specified in an individual plan.

Primitive Actions We model an action as a transition between two world states, using a STRIPS description language (Fikes & Nilsson 1972). Specifically, each action is described by a *precondition*, (i.e., the list of propositions that must be true in every world state where the action is applicable), a *delete list* (i.e., the list of propositions that must be deleted when the action is applied), and an *add list* (i.e., the list of propositions that must be added when the action is executed). As in STRIPS, predicates with variables can be used to describe schemas of actions. In addition, our action language allow *interpreted predicates*, that is, predicates for which the truth value is obtained from that of *basic predicates*. Interpreted predicates are only used in the preconditions and in goal formulas. They are not specified in add lists, delete lists, or world-state descriptions, since their truth value is derived from that of basic propositions. Interpreted predicates make it easy, for example, to describe actions that depend on time constraints.

We distinguish between *controllable* actions that are performed by the agents being synchronized and *uncontrollable* (or *exogenous*) actions that are performed by an environment. As is explained later, this classification allows us to model nondeterministic executions.

Plans A plan for an agent is a set of reaction rules of the form

$$S \rightarrow \{a, s(x_1, v_1), \ldots, s(x_n, v_n)\},$$

where S denotes a *state of the agent* consisting of propositional symbols that describe facts about the world or characterize values of internal variables such as synchronization variables; a is a primitive controllable action executable by the agent; and $s(x_i, v_i)$ are *optional* internal instructions for setting variables x_i to values v_i. When the *set* instructions are absent or understood, we will denote the reaction rule by $S \rightarrow a$. We assume that S subsumes the preconditions of a. An agent executes such a plan by continuously sensing the world and internal data to find a reaction rule matching the current situation, and then reacting by performing the action specified by this rule and setting the specified variables to the corresponding values. If many different rules are enabled, one of them is selected nondeterministically.

Hence, a plan only involves controllable actions. However, as is explained below, some of the propositions describing a plan state are derived from the environment's actions. Hence, the environment's actions are only used to generate *events* when constructing synchronized plans. They are not explicitly involved in the execution of plans.

Modeling Plan Execution In order to synchronize plans, we need to predict their possible models of concurrent executions, evaluate these models as to whether or not they satisfy an MTL goal formula, and modify the plans accordingly. Modeling the execution of a plan amounts to modeling its *sense-then-react* execution loop.

We model the *sense-then-react* loop for an agent by introducing a special agent, the *environment*, which performs only *sensing actions*. The execution of a *single* plan is modeled by a sequence of world states, where a transition between two states is labeled with an action, and such that each transition labeled with environment action is followed by one labeled with the agent's action and vice-versa. The *concurrent execution* of a *set of plans* is modeled by interleaving the world-state sequences corresponding to each plan execution. As has been noted by Emerson (Emerson 1990) (page 1017), by picking a sufficiently fine level of granularity for the primitive actions, any behavior that could be produced by true concurrency (i.e., true simultaneity of actions) can be simulated by interleaving. In practice, it is helpful to use as coarse a granularity as possible, so as to reduce the number of interleavings that must be considered.

In order to generate sequences of world states in which the environment and agents' transitions alternate, our search process that generates plan interleavings is run on modified versions of the given original plans. The modifications consist in introducing a binary variable z and the propositions $z0$ ("z has the value 0") and $z1$ ("z has the value 1") into the plans, such that $z0$ holds in every state where an environment's action can be applied, whereas $z1$ holds in every state where an agent's action can be applied. Specifically, we add $z0$ to the antecedent of every environment's reaction rule and $z1$ to the antecedent of every agent's reaction rule. On the other hand, we add $s(z,1)$ to the consequent of every environment's reaction rule and $s(z,0)$ to the consequent of every agent's reaction rule.

Our search process produces a search graph of interleavings. In this graph, a branching point between transitions labeled with environment actions represents a nondeterministic choice, that is, different possible outcomes of the next sensing operations involved in the *sense-then-react* loop after having executed the agent's action leading to this state. A branching point between actions of different agents represents their parallel executions. A branching point between actions of the same agent represents a plan selection between these actions. The exploration of the search space is, however, limited by a combinatorial explosion problem. We will discuss strategies to deal with this limitation in the conclusion.

Example 1 Figure 1 shows a graph of interleavings of the plans given in Table 2 using the actions provided in Table 1. The initial world state is $\{p_1, q_1\}$. We have two controllable agents: one performing the actions starting with an a; the other, those starting with a b; and an environment performing the actions starting with an e. The goal, which is formally described later (see Example 2), requires that we never reach the state $\{p_2, q_2\}$. Moreover, each time p_1 becomes true, we must have p_2 within 3 time units and, similarly, each time q_1 becomes true, we must have q_2 within 3 time units. Assuming that each *sense-then-react* step takes 1 time unit, the problem is to synchronize the given plans. Our approach is to describe the goal using MTL formulas, to check these formulas on interleavings of the plans, and to synchronize the plans accordingly.

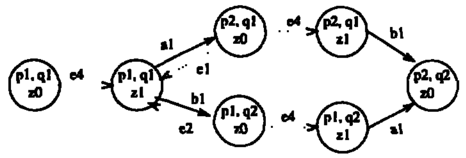

Figure 1: A search graph

Action	Pre	Del	Add
a1	{p1}	{p1}	{p2}
a2	{p3}	{p3}	{p1}
b1	{q1}	{q1}	{q2}
e1	{p2,q1}	{p2}	{p1}
e2	{p1,q2}	{q2}	{q1}
e3	{p2,q1}	{p2}	{p3}
e4	{}	{}	{}

Table 1: Actions for an artificial domain

This example is a version of the following tileworld traffic problem, which we have simplified for the sake of clarity. We have three agents in Figure 2 (A, B, and C) that are controllable and associated with plans for reacting to the traffic of an environment that controls the move of the tiles E and F. The tile F moves continuously clockwise along the rectangle formed by the corner cells $(1,0)$, $(1,4)$, $(4,4)$, and $(4,0)$, at a speed solely controlled by the environment. Similarly, E also moves continuously clockwise, but along the smaller rectangle. Both F and E always stop at least two time units before moving in the cell $(1,1)$ or $(1,3)$. Each

Plan	Reaction rules
P_A	{p1}→ a1 {p3}→ a2
P_B	{q1}→ b1
P_E	{p2,q1}→ e1 {p1,q2}→ e2 {p2,q1}→ e3 {}→ e4

Table 2: Plans for an artificial domain

agent in the tileworld also has some sensing capabilities. In our example, agents F and E can only sense agents of type A, B, or C that are in an adjacent cell. Hence, F and E might collide in the cells $(1,1)$ or $(1,3)$, since they cannot sense each other. On the other hand, agents A, B and C can sense any other agent within a distance of three cells.

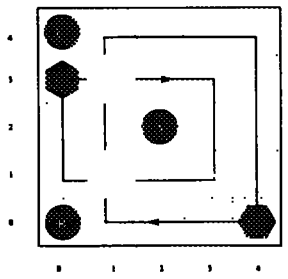

Figure 2: A traffic problem in a tileworld

Let us denote a cell by its (x,y) coordinates. The fact "agent Z is in the cell (x,y)" is denoted by the propositional symbol Zxy; the fact "the cell (x,y) is clear" is denoted by the propositional symbol cxy; the action "move agent Z in the direction d" is denoted by mdZ; the grid is oriented such that north is up. We are given plans P_A, P_B, P_C, and P_E, respectively, for A, B, C, and the environment, and a goal; the problem is to synchronize them so that their concurrent execution satisfies the goal:

- P_A: $\{E31, A00\} \rightarrow meA$; $\{c31, c21, c11, A10\} \rightarrow mwA$.
- P_B: $\{F20, B22\} \rightarrow msB$; $\{c20, c10, c11, B21\} \rightarrow mnB$.
- P_C: $\{E02, C04\} \rightarrow msC$; $\{c11, c12, c13, C04\} \rightarrow mnC$.
- P_E: $\{\} \rightarrow e$, where e describes the operation of sensing the presence of agent E or F, respectively, in cells $(3,1), (2,1), (1,1)$ and $(0,2)$, or $(2,0), (1,0), (1,1), (1,2)$ and $(1,3)$. In other words, if applied to a world state, e produces many different possible successor world states representing the possible outcomes of the corresponding sensing operations.
- The synchronization goal formula is similar to that of the previous artificial problem. It simply requires that no collision or deadlock occur. A deadlock occurs when either E or F is blocked indefinitely by A, B, or C.

Note that while plans P_A, P_B, and P_C can control the traffic of E and F without collision, they might lead to deadlocks ■

MTL

In this section, we briefly define the MTL formulas that we use to describe goals. The reader is referred to (Koymans 1990; Alur & Henzinger 1993) for a more comprehensive discussion of MTL.

Syntax MTL formulas are constructed from an enumerable collection of propositional symbols; the boolean connectives $\wedge$ (and) and $\neg$ (not); and the temporal modalities $U_{\sim x}$ (until) and $\Box_{\sim x}$ (always), where $\sim$ denotes $=$, $\leq$, or $\geq$, and x is an integer. The formula formation rules are (1) every propositional symbol p is a formula and (2) if f_1 and f_2 are formulas, then so are $\neg f_1$, $f_1 \wedge f_2$, $\Box_{\sim x} f_1$, and $f_2 \, U_{\sim x} \, f_1$.[1] The following abbreviations are standard: $\Diamond_{\sim x} f \equiv true \, U_{\sim x} \, f$ (eventually f), and $f_1 \rightarrow f_2 \equiv \neg f_1 \vee f_2$ (f_1 implies f_2).

Semantics MTL formulas are interpreted over models of the form $M = \langle S, \pi, \mathcal{T} \rangle$, where S is an infinite sequence of states $s_0, s_1, \ldots$; π is a function that evaluates propositional symbols in a state;[2] and $\mathcal{T}$ is a function that associates a time stamp with each state. As usual, we write $\langle M, s \rangle \models f$ if state s in model M satisfies formula f. In addition to the standard rules for the boolean connectives, we have that for a state s_i in a model M, a propositional symbol p, and formulas f_1 and f_2:

- $\langle M, s_i \rangle \models p$ *iff* $\pi(p, s_i) = true$.
- $\langle M, s_i \rangle \models \Box_{\sim x} f_1$ *iff* $\forall j$ such that $\mathcal{T}(s_j) \sim \mathcal{T}(s_i) + x$, $\langle M, s_j \rangle \models f_1$.
- $\langle M, s_i \rangle \models f_1 \, U_{\sim x} \, f_2$ *iff* $\exists j$ such that $\mathcal{T}(s_j) \sim \mathcal{T}(s_i) + x$, $\langle M, s_j \rangle \models f_2$, and $\langle M, s_k \rangle \models f_1$ for all k, $i \leq k < j$.

Finally, we say that the model M (or sequence of states $S \in M$) satisfies a formula f if $\langle M, s_0 \rangle \models f$.

Example 2 The formula $\Diamond_{\leq 10} \Box_{\geq 0} \, p$ means that p holds within 10 time units and remains true thereafter. The conjunction of the following formulas is used to express the goal involved in the previous artificial example: (1) $\Box_{\geq 0}(p_1 \rightarrow \Diamond_{\leq 3} \, p_2)$ (i.e., each time p_1 becomes true, p_2 must also become true within 3 time units); (2) $\Box_{\geq 0}(q_1 \rightarrow \Diamond_{\leq 3} \, q_2)$; and (3) $\Box_{\geq 0} \neg (p_2 \wedge q_2)$ ■

Bounded-time Formulas As was mentioned in the introduction, we describe our planning algorithm only for *bounded-time MTL formulas*. To define them, we also need a notion of *normalized MTL formulas*.

[1]For the sake of simplicity, we do not use the *congruence modulo constraint* and the *next* modality defined in the standard MTL (Alur & Henzinger 1993).

[2]The set of propositions true in state s is defined as $\{p | \pi(p, s) = true\}$.

Definition 1 A *normalized MTL formula* is one in which only propositional symbols are negated.[3] A *bounded-time MTL formula* is a normalized MTL formula in which every *until* modality is of the form $U_{\leq x}$ (remember that $\Diamond_{\leq x}$ is an abbreviation of $true\, U_{\leq x} f$).

All the formulas in Example 2 are bounded-time ones. Note that a bounded-time response such as $\Box_{\geq 0}(p_1 \rightarrow \Diamond_{\leq 3} p_2)$ conveys a liveness requirement that must be checked on each infinite execution.

Plan Synchronization Method

Our planner accepts as input a set of plans, a set of primitive actions, an initial world state, a *normalized bounded-time MTL* goal formula, and a function π, which interprets propositions in world states. It returns a set of synchronized plans. Note that if we had many initial states, the planner would be run on each of them separately.

The planner proceeds by *searching* for sequences of world states formed from plan interleavings and simultaneously checking that they satisfy the goal. Synchronized plans are extracted from satisfactory sequences. The search for satisfactory sequences is based on a notion of *progressing an MTL formula* through a world state. In fact, each MTL formula can always be seen as conveying a *present* requirement that must be satisfied in the current state of a sequence of world states and a *future* requirement that must be satisfied by the rest of the sequence. Intuitively, progressing an MTL formula through a world state is to check that the present requirement of the formula is satisfied by this state and, if it is, to return the future part; otherwise *false* is returned.

Formula Progression Algorithm

The formula progression algorithm is described in Figure 3. The function *Prog* accepts as input an MTL formula f, a world state s, a real number t, and a function $\pi(p, s)$ that returns *true* if the proposition p holds in the world state s; otherwise *false* is returned; *Prog* returns an MTL formula representing the progression of f through s. The functions $Neg(f_1)$, $Conj(f_1, f_2)$ and $Disj(f_1, f_2)$ return, respectively, $\neg f_1$, $f_1 \wedge f_2$, and $f_1 \vee f_2$. This algorithm is characterized by the following theorem.

***Theorem 1** Let $M = \langle S, \pi, \mathcal{T}\rangle$ be any MTL model, and let s_i be the i-th state in the sequence of states S. Then, we have for any MTL formula f, $\langle M, s_i\rangle \models f$ if and only if $\langle M, s_{i+1}\rangle \models Prog(f, s_{i+1}, \mathcal{T}(s_{i+1}) - \mathcal{T}(s_i), \pi)$.*

[3]We obtain a normalized formula equivalent to any MTL formula by using De Morgan laws and logical equivalences to propagate the negation symbol inwards.

```
Algorithm Prog($f$,$s$,$t$,$\pi$)
1. case $f$
2.    $p$ ($p$ a proposition): $\pi(p, s)$;
3.    $\neg f_1$ : $\neg Prog(f_1, s, t, \pi)$;
4.    $f_1 \wedge f_2$: $Prog(f_1, s, t, \pi) \wedge Prog(f_2, s, t, \pi)$;
5.    $f_1 \vee f_2$: $Prog(f_1, s, t, \pi) \vee Prog(f_2, s, t, \pi)$;
6.    $\Box_{=x} f_1$: if $x - t > 0$ then $\Box_{=(x-t)} f_1$
7.                            else $Prog(f_1, s, t, \pi)$;
8.    $\Box_{\leq x} f_1$: if $x - t > 0$ then $Prog(f_1, s, t, \pi) \wedge \Box_{\leq(x-t)} f_1$
9.                            else $Prog(f_1, s, t, \pi)$;
10.   $\Box_{\geq x} f_1$: if $x - t > 0$ then $\Box_{\geq(x-t)} f_1$
11.                           else $Prog(f_1, s, t, \pi) \wedge \Box_{\geq 0} f_1$;
12.   $f_1 U_{=x} f_2$:
13.     if $x - t > 0$ then $Prog(f_1, s, t, \pi) \wedge f_1 U_{=(x-t)} f_2$
14.                    else $Prog(f_2, s, t, \pi)$;
15.   $f_1 U_{\leq x} f_2$:
16.     if $x - t > 0$ then $Prog(f_2, s, t, \pi) \vee$
17.                         $(Prog(f_1, s, t, \pi) \wedge f_1 U_{\leq(x-t)} f_2)$
18.                    else $Prog(f_2, s, t, \pi)$;
end
```

Table 3: Progression algorithm for bounded-time formulas.

The proof for this theorem follows easily from the semantics of MTL formulas by observing that the formula progression algorithm implements interpretation rules for the semantics axioms in the previous section.

Search Process Algorithm

We search for sequences of world states that satisfy a bounded-time MTL formula by progressing it through these world states. The main effect of this goal progression is to prune those sequences that violate maintained subgoals or eventuality deadlines. More precisely, we generate a graph of *search states*, where each search state is an MTL formula of the form $zi \wedge w \wedge m$: zi is either $z0$ or $z1$ (i.e., the propositions that were used to extend plans in order to interleave them as was indicated in Section); w is a conjunction of propositions denoting a world state; and f an MTL formula.

The initial search state is $z0 \wedge w_0 \wedge m_0$ where w is the input initial world state and $m_0 = Prog(m, w, 0, \pi)$, with m the input goal. Note that progressing m through w using the time difference 0 amounts to checking that the present part of m is consistent with w.

The successor of a search state $zi \wedge w_1 \wedge m_1$ under the application of a reaction rule $S \rightarrow o$ is the search state $zj \wedge w_2 \wedge m_2$, where $zj = z0$ if $zi = z1$, otherwise $zj = z1$; w_2 =*apply-action*(S, o, w_1) and $m_2 = Prog(m, w, t, \pi)$, with $t = 0$ if $zi = 0$ (i.e., if o is an environment's action), otherwise $t = 1$. The use of time duration 0 to progress formulas over environment transitions and 1 over agent transitions is bolstered by the fact that we have assumed that each *sense-then-react* step takes 1 time unit. The function *apply-action*(S, o, w_1) proceeds as follows: if S holds in w, then the function returns the world state obtained from w_1 by deleting the delete list of o, adding the add list of o, and updating internal variables as indicated by the instructions in the consequent of the reaction rule; otherwise, it returns *false*.

The search state *false* is a *sink*. A search state $z0 \wedge w \wedge m$ that has a transition leading to a sink also becomes a *sink*. Each time we reach a search state $z0 \wedge w \wedge f$ that has no corresponding action enabled by a plan, other than the environment *idle* action (i.e., one which does not change the world, such as e_4 in Example 1), we check if f involves any eventuality modality. If it does, the state becomes a *sink*. Otherwise, this state is successful since we can satisfy its requirement by staying in it forever: there is no risk of being moved out by the environment since no action, other than *idle*, is enabled. Each time we reach a search state $z1 \wedge w \wedge f$ that has no corresponding action enabled by a plan, or for which all enabled actions lead to *sinks*, we apply a dummy action $it1$ (i.e., increment time by 1), which has the time duration 1, and empty add and delete lists. This action only has the effect of progressing the goal and replacing $z1$ with $z0$ so that the environment transitions can be applied in the resulting state. This search procedure is characterized by the following theorem.

Theorem 2 *If the goal formula contains no unbounded-time eventuality, each path of the search graph terminated by a cycle (and not involving sink states) corresponds to an infinite sequence of world states satisfying the goal formula.*

We give a proof sketch for this theorem. First, note that each path in the search graph is terminated by a cycle or a sink. The most trivial cycles correspond to a self loop with the environment *idle* action. The proof is trivial for these cycles since they do not involve any eventuality modality. For the more general case, assume that we have a cycle not containing a sink. The only problem is that we might have an eventuality formula in some state in the cycle that is progressed all along the cycle without being achieved. This problem is ruled out by the following three observations. First, we always expand states by alternating the applications of controllable and uncontrollable actions. Second, each application of an uncontrollable transition yields a successor state different from the current one because the z propositions are changed. Third, the progression of a timely-bounded formula through a controllable transition yields a formula different from the current one because the timing constraint of every bounded-time formula (in particular, every bounded-time eventuality) is decremented by the duration of the controllable action, which is strictly positive. This means that the progression of the formula in the current state cannot yield a formula identical to that of an ancestor state unless all the bounded-time formulas that have been progressed between this state and the current one have disappeared (i.e., made *true* or *false*). The conjunction of these three observations entails that we can have a cycle only if all bounded-time eventualities have been made *true* in this cycle (they cannot have been made *false* because this would have generated a sink).

The previous theorem thus indicates that our search process will be successfully terminated once each environment transition that is reachable from the initial world state is in a cycle not involving sink states. Note that this theorem only holds for bounded-time formulas. Indeed, the progression of an unbounded-time eventuality might yield the same formula as an earlier progression without having the eventuality satisfied anywhere between the two progressions. Our method can be extended to handle unbounded-time eventualities by introducing additional mechanisms in the search process. This extension is, however, beyond the scope of this paper.

$$\begin{tikzcd}
f0 \arrow[r, "e4"] & f1 \arrow[r, "a1"] & f2 \arrow[r, "c1"] & f3 \arrow[r, "a1"] & f4 \arrow[d, "c1"] \\
 & & f7 \arrow[u, "a1"] & f6 \arrow[l, "e2"'] & f5 \arrow[l, "h1"']
\end{tikzcd}$$

- goal= $f = \Box_{\geq 0} \neg(p2 \wedge q2) \wedge \Box_{\geq 0}(p1 \rightarrow \Diamond_{\leq 2} p2)$
  $\wedge \Box_{\geq 0}(q1 \rightarrow \Diamond_{\leq 2} q2)$
- $f_0 = z_0 \wedge p1 \wedge q1 \wedge f \wedge \Diamond_{\leq 2}; p2 \wedge \Diamond_{\leq 2} q2;$
- $f_1 = z_1 \wedge p1 \wedge q1 \wedge f \wedge \Diamond_{\leq 2} p2 \wedge \Diamond_{\leq 2} q2;$
- $f_2 = z_0 \wedge p2 \wedge q1 \wedge f \wedge \Diamond_{\leq 1} q2;$
- $f_3 = z_1 \wedge p1 \wedge q1 \wedge f \wedge \Diamond_{\leq 1} q2 \wedge \Diamond_{\leq 2} p2;$
- $f_4 = z_0 \wedge p2 \wedge q1 \wedge f \wedge \Diamond_{\leq 0} q2;$
- $f_5 = z_1 \wedge p1 \wedge q1 \wedge \wedge \Diamond_{\leq 0} q2 \wedge \Diamond_{\leq 2} p2;$
- $f_6 = z_0 \wedge p1 \wedge q2 \wedge f \wedge \Diamond_{\leq 1} p2 \wedge f;$
- $f_7 = z_1 \wedge p1 \wedge q1 \wedge f \wedge \Diamond_{\leq 1} p2 \wedge \Diamond_{\leq 2} q2.$

Figure 3: A satisfactory cycle for the artificial problem

Example 3 Figure 3 shows a satisfactory cycle for the problem in Example 1. In this figure, each formula f_{j+1} is obtained by progressing the modal part of f_j through the world state part of f_{j+1}. Moreover, we have applied the following simplification: $\Diamond_{\leq k} p \equiv \Diamond_{\leq k} p \wedge \Diamond_{\leq l}$, $\forall k, l$ such that $l > k$. Notice the correspondence between the transitions in this cycle and those in the interleavings of Figure 1 ∎

Extracting Plans

We extract synchronized plans from satisfactory paths terminated by cycles. The extraction process involves a replacement of the modal part of search states having equal world states and being in a cycle, with propositions ranging over values of shared synchronization variables (remember that a search state consists of a world state and an MTL formula). The explanation of this extraction process is beyond the scope of this paper. For the sequence in Figure 3, we will obtain the reaction rules $\{p1, q1, x0\} \rightarrow \{a_1, s(x,1)\}$, $\{p1, q1, x1\} \rightarrow \{a_1, s(x,2)\}$, and $\{p1, q1, x2\} \rightarrow \{b_1, s(x,0)\}$; the proposition xi means "the variable x has the value i."

It can be shown that if the plan synchronization problem has a solution, then our algorithm will sooner or later terminate successfully. However, the size of the state space can be very large such that an effective termination would require heuristic search strategies. This is further discussed below.

As has been explained so far, our approach only generates sequential plans. We can generate parallel plans by considering simultaneous transitions explicitly. For instance, we have to consider the transition labeled with $\{a_1, b_1\}$ from the state f_1 in Figure 3, in addition to those labeled solely with a_1 or b_1. This must be done because the simultaneous execution of a_1 and b_1 lasts 1 time unit. Although it leads to the same world state as the interleavings of a_1 and b_1, it has a different time duration, hence a different progressed subgoal. In other words, we have two search states having the same world state but different goals. However, such an approach requires us to consider an exponential number of possible transitions in each search state.

We are currently finalizing a more practical approach that avoids this combinatorial explosion. In this new approach, we can still reason about simultaneous executions solely using interleavings, that is, without applying subsets of actions. With this new approach the search space reduces to $S \times G$, with S being the number of world states reachable from the initial state, and G the number of different subgoals that can be progressed. It can be shown that G is $O(2^{n \times k})$, with n being the number of different modal operators in the input goal and k the greatest action duration. This worst-case complexity bound is, however, overly pessimistic since a considerable number of search states will consist of sinks. Results of this approach will appear in a subsequent paper.

Although the new approach reduces the combinatorial explosion considerably, the number of possible states that are not sinks remains tremendous even for medium-scaled problems. This means that an exhaustive exploration of the states of $S \times G$ that are not sinks is not practical in reactive planning in which the synchronizer must generate solutions within very short deadlines. Exhaustive exploration of search spaces has been proven effective in hardware and protocol verification using efficient implementation techniques (e.g., see (Emerson 1990) for a starting point). However, it does not seem realistic to expect similar effectiveness for exhaustive exploration in reactive planning. Indeed, an acceptable output delay for verification tools is on the order of minutes or hours for many practical problems, whereas it is on the order of milliseconds or seconds for many reactive planning problems. Below, we conclude with some strategies that we think could enhance our method, making it effective in planning applications.

Conclusions

We have described a method for synchronizing plans based on model checking of MTL goal specifications. Our method handles qualitative as well as quantitative time specifications. We think that our method has significant potential for real applications. This will require, however, appropriate mechanisms to deal with the combinatorial state explosion. We are investigating the integration of some available AI planning and multiagent planning techniques for this purpose. For instance, it seems natural to extend our planning algorithm using search control-strategies expressed with temporal logic formulas. Such strategies were proven to be effective in different classical planning problems (Bacchus & Kabanza 1995). Along similar lines, we are interested in adapting classical heuristic exploration algorithms, such as *best-first* or A^*. In these algorithms, one specifies a heuristic function assigning a merit to each state in the search graph depending on how it looks close to a given goal state. Using these merits, search states are selected for expansion in the order of their promise. We are studying a generalization of similar approaches using the observation that, instead of a notion of final goal state, we have a notion of final goal cycle. We are also investigating partial order search techniques similar to those in (Godefroid & Kabanza 1991) that could moderate the tradeoffs incurred by a model of concurrent executions with nondeterministic interleaving of actions. Another interesting extension would be to use modular representations of actions that abstract over large sets of states.

Plan synchronization is complementary to other techniques in multiagent planning. In particular, it can be combined with negotiation and the *social laws* of Shoham and Tennenholtz (Shoham & Tennenholtz 1994). The idea behind *social laws* is to design agents that obey a number of basic safety conventions such that to reduce the number of possible conflicts when they interact. These conflicts can then be resolved using synchronization techniques or negotiation mechanisms. For example, in a domain of mobile robots, we could adopt that each robot should keep to the right of the path much like cars do on streets. In this way, possible conflicts should occur mostly at crossings. We could exploit this knowledge to explore the search space using descriptions of actions that abstract over states without potential conflicts. These are some of the extensions that we believe could make our method useful in practical applications.

References

Alur, R., and Henzinger, T. 1993. Real-time logics: Complexity and expressiveness. *Information and Computation* 104(1):35–77.

Bacchus, F., and Kabanza, F. 1995. Control strategies in planning. *AAAI Spring Symposium Series. Extending Theories of Action: Formal Theory and Practical Applications.*

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximazing probability of goal satisfaction. In *Proc. of Eighth National Conference on Artificial Intelligence*, 138–144.

Emerson, E. A. 1990. Temporal and modal logic. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science*, volume B. MIT Press/Elsevier. 995–1072.

Fikes, R., and Nilsson, N. J. 1972. Learning and executing generalized robots. *Artificial Intelligence* 3(4):251–288.

Georgeff, M. P. 1987. Actions, processes, and causality. In Georgeff, M. P., and Lansky, A., eds., *Reasonning about Actions and Plans, Proceedings of the 1986 Workshop, Timberline, Oregon*, 123–159. Morgan Kaufmann.

Godefroid, P., and Kabanza, F. 1991. An efficient reactive planner for synthesizing reactive plans. In *Proc. of Ninth National Conference on Artificial Intelligence*, 640–645.

Kabanza, F. 1990. Synthesis of reactive plans for multi-path environments. In *Proc. of Eighth National Conference on Artificial Intelligence*, 164–169.

Koymans, R. 1990. Specifying real-time properties with metric temporal logic. *Real-time Systems* 2(4):255–299.

Lansky, A. 1988. Localized event-based reasoning for multiagent domains. *Computational Intelligence Journal, Special Issue on Planning* 4(4).

Manna, Z., and Wolper, P. 1984. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 6(1):68–93.

Shoham, Y., and Tennenholtz, M. 1994. On social laws for artificial agent societies: off-line design. *Artificial Intelligence* 72(1-2):231–252. Part 2.

Stuart, C. 1985. An implemantation of a multiagent plan synchronizer using a temporal logic theorem prover. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1031–1033.