A Method for the Synthesis of Controllers to Handle Safety, Liveness, and Real-Time Constraints

Michel Barbeau, *Member, IEEE,* Froduald Kabanza, *Member, IEEE,* and Richard St.-Denis, *Member, IEEE*

***Abstract*—This paper describes a synthesis method that automatically derives controllers for timed discrete-event systems with nonterminating behavior modeled by timed transition graphs and specifications of control requirements expressed by metric temporal logic (MTL) formulas. Synthesis is performed by using 1) a forward-chaining search that evaluates the satisfiability of MTL formulas over sequences of states generated by occurrences of actions and 2) a control-directed backtracking technique that takes into consideration the controllability of actions. This method has several interesting features. First, the issues of controllability, safety, liveness, and real time are integrated in a single framework. Second, the synthesis process does not require explicit storage of an entire transition structure over which formulas are checked and can be stopped at any moment, giving an approximate but useful result. Third, search and control mechanisms allow circumvention of the state explosion problem.**

***Index Terms*— Discrete-event systems, metric temporal logic, supervisory control, synthesis algorithm, ω-languages.**

I. Introduction

A CONTROLLER can be viewed as a program that restrains the behavior of a process in order to satisfy given constraints on sequences of actions executed by the process. *Supervisory control theory,* initiated by Ramadge and Wonham [36], addresses the problem of synthesizing controllers for discrete-event systems (DES) by focusing on the formulation of conditions for the solvability of different control problems and on the investigation of algorithms for computing controllers from formal specifications. One of the main issues of this theory concerns the *controllability* of a specification, which has similarities with the issue of *realizability* [1], [34]. In *open* systems, the process to be controlled interferes with other processes in its environment. This interaction is essentially of a reactive nature. A controller can be realized by taking into account changes caused by uncontrollable events generated by the environment.

Specifying the dynamics of a process and control requirements represents a challenge for engineers who want to apply formal methods such as controller synthesis. This task can be accomplished by using specification languages that are expressive and readable. Expressiveness deals with complex properties, while readability facilitates the explanation of specifications. One method of writing down specifications is to use state machines. In fact, most synthesis methods for supervisory control theory have been done in the context in which both the unrestrained and legal behaviors of a process are modeled with automata [16], [24], [37], [40]. Besides, temporal logics have long been recognized as a useful formalism for specifying properties of reactive systems [19], [30]. One key characteristic of such logics is that they are declarative and involve simple syntax and semantics. Although mostly used in the verification of concurrent systems, temporal logics have been applied to supervisory control theory (e.g., [21], [26], [27], [32], [33], [41] for linear temporal logic frameworks and [5] for a branching temporal logic framework). Real-time interval logics have also been used in the verification of control systems (e.g., [38]).

The synthesis method advocated in this paper uses specification formalisms and integrates temporal aspects by associating durations to transitions and time constraints to modal operators. More specifically, the dynamics of processes and specifications of control requirements are represented by *timed transition graphs* (TTG's) and *metric temporal logic* (MTL) formulas [4], [23], respectively. Such an approach is very attractive. On the one hand, the dynamics of a process is more understandable from a state machine because it explicitly shows the atomic actions, the states in which they are enabled, and their effects. On the other hand, constraints on a process are often more understandable from declarative statements.

Our synthesis method is closely related to the recent work by Brandin and Wonham [15]. In their model, both the dynamics of processes and specification of control requirements are described by TTG's. Transitions represent instantaneous events and time progresses in states that represent actions. In our case, transitions represent actions with durations. As discussed in [11], both models are dual but lead to different synthesis methods. However, our approach can also handle *liveness* constraints, that is, constraints over nonterminating behaviors or behaviors that have a very remote or indefinite termination point. Thus, the control requirements expressed by an MTL formula refer to infinite behaviors.

Our method is also closely related to recent works by Thistle and Wonham [42], [43]. We adopt the same model for the nonterminating behavior of the closed-loop system which is essentially due to Ramadge [35]. Like Brandin and Wonham's approach, they use only transition structures to realize effective controllers: processes are represented by deterministic Büchi automata and specifications of control requirements by deter-

Manuscript received February 7, 1997; revised November 27, 1997. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

The authors are with the Département de Mathématiques et d'Informatique, Université de Sherbrooke, Sherbrooke, Québec, J1K 2R1 Canada.

Publisher Item Identifier S 0018-9286(98)08480-3.

ministic Rabin automata.[1] This representation also allows the expression of liveness constraints, but it does not deal with time constraints.

The most substantial difference with these two previous works and many others that consider the problem of finding a winning strategy for finite or infinite games (e.g., [7], [29]) lies in the synthesis algorithms. As in the original method proposed by Wonham and Ramadge [47], they promote synthesis methods based on a fixpoint characterization of the supremal controllable sublanguage of a given legal language. In addition, they include an induction on the automaton structure to compute a controller that generates the supremal controllable sublanguage. In contrast, our approach simply consists of seeing sequences of actions as paths on a TTG. By searching through the space of possible paths, MTL formulas representing constraints are verified over these paths to determine points at which controllable actions must be disabled. This is done incrementally in a single phase so that a controller can be obtained without exploring the entire state space because unsatisfactory paths are pruned and most of the vertices on these paths are not expanded further [22]. A depth-first exploration obviates storing the entire graph in memory. Furthermore, heuristics and search control mechanisms, reminiscent of familiar techniques in the field of artificial intelligence search, can be used to control the state explosion problem. One can reasonably expect that our algorithm is less greedy for memory and performs better on average.

The rest of the paper is organized as follows. Section II summarizes Thistle and Wonham's framework and situates the control problem addressed in this paper with regard to their model. Section III describes the syntax and semantics of MTL, gives a characterization of safety and liveness constraints, and introduces basic properties of temporal operators that allow transformations of formulas into appropriate forms. Section IV presents the foundation of our synthesis method by abstracting over implementation details. Section V contains some simple examples illustrating the method's most important aspects. Section VI introduces fundamental properties that will be used to show the correctness of a new synthesis algorithm detailed in Section VII. Section VIII presents a simple application to an antenna rotor control system. Finally, Section IX discusses related works from a more technical point of view and concludes the paper.

II. Supervisory Control of DES

The atomic actions of a DES are represented by a nonempty set of symbols A, called an alphabet. Let A^* and A^ω be the set of finite words and the set of infinite words over A, respectively. The empty word is noted ϵ. An ω-word over A is written as $\alpha = \alpha[0]\alpha[1]\cdots$ and represents an infinite execution of actions. Let $A^\infty := A^* \cup A^\omega$. For any two words $k \in A^*$ and $u \in A^\infty$, the expression $k \leq u$ means that k is a prefix of u. Given $K \subseteq A^*$, $U \subseteq A^\infty$, and $W \subseteq A^\omega$, we have the following operations:

$$\begin{aligned} \mathrm{pre(U)} &:= \{k \in A^*\colon (\exists u \in U)(k \leq u)\} \\ \lim(K) &:= \{w \in A^\omega\colon \mathrm{pre}(\{w\}) \subseteq K\} \\ \mathrm{clo}(W) &:= \lim(\mathrm{pre}(W)). \end{aligned}$$

Following Ramadge [35], a DES G is modeled by a pair of languages $L \subseteq A^*$ and $L_\omega \subseteq A^\omega$, such that $L = \mathrm{pre}(L)$ (L is $*$-closed) and $\mathrm{pre}(L_\omega) \subseteq L$. The languages L and L_ω are used to describe *transient* and *persistent* traces of actions that the process can execute. If $\mathrm{pre}(L_\omega) = L$, then G is deadlock-free.

Let $\{A_c, A_{uc}\}$ be a partition of A, where A_c and A_{uc} denote the set of *controllable* actions and set of *uncontrollable* actions, respectively. Let $\Gamma := \{\gamma \in 2^A\colon A_{uc} \subseteq \gamma\}$. A supervisor is a function $S\colon A^* \rightarrow \Gamma$ that maps each finite sequence of actions to a set of enabled actions.

A controlled DES is one constrained by a supervisor. Given a DES $G = (L, L_\omega)$ and a supervisor S, the corresponding controlled DES is noted $G^S = (L^S, L^S_\omega)$, where:

1) L^S is defined recursively as $\epsilon \in L^S$ and for all $k \in A^*$ and $a \in A$, $ka \in L^S$ *iff* $k \in L^S$, $ka \in L$, and $a \in S(k)$;
2) $L^S_\omega := \lim(L^S) \cap L_\omega$.

We assume that the supervisor S is complete, that is, L^S is a subset of the domain of S. A supervisor S is said to be deadlock-free for G if $\mathrm{pre}(L^S_\omega) = L^S$. The control problem addressed herein can now be formalized as follows.

Problem 1: Given a DES $G = (L, L_\omega)$ and $W \subseteq A^\omega$ such that $W \subseteq L_\omega$, construct a complete deadlock-free supervisor S for G such that $L^S_\omega \subseteq W$.

Thistle and Wonham [43] give necessary and sufficient conditions for the existence of a maximal solution to this problem. Their result is mainly based on ω-controllability and ω-closed properties. If the ω-closed property is not satisfied for a particular instance of Problem 1, the maximal solution does not exist because of the open-ended nature of liveness properties [43]. Besides, our goal is not to derive the maximally permissive controller, but a useful controller.

In this paper, we provide a solution for a particular case of Problem 1. We assume that $L_\omega = \lim(L)$, that is, L_ω is completely determined by L. We also suppose that the control requirements are given by an MTL formula f. Thus, the legal language W can be interpreted as transforming f into a nondeterministic Büchi automaton by using the *tableau method* [45], then taking the intersection of the language accepted by the Büchi automaton with the language L_ω. In reality, we do not construct the Büchi automaton. Rather, our method works incrementally on f and a representation of L so that only the part of the Büchi automaton relevant to f is built. Finally, we assume that the DES is modeled as a TTG $G = (X, \mathcal{P}, \lambda, A, \tau, \xi, x_0)$, where X is a finite set of states; $\mathcal{P}$ is a finite set of propositional symbols; $\lambda\colon X \rightarrow 2^{\mathcal{P}}$ is a labeling function that assigns to each state the set of propositional symbols true at that state; A is a finite set of actions partitioned into A_c and A_{uc}; $\tau\colon A \rightarrow \mathbb{R}^+$ is the time duration function such that $\tau(a) > 0$ for all $a \in A$; $\xi\colon X \times A \rightarrow X$ is the transition function; and $x_0 \in X$ is the initial state. The $*$-language generated by G is $\mathcal{L}(G) := \{k \in A^*\colon \xi(x_0, k) \text{ is defined}\}$

[1] Büchi and Rabin automata are finite automata equipped with an acceptance condition that is appropriate for infinite words [44].

and the ω-language accepted by G is $\mathcal{L}_\omega(G) = \lim(\mathcal{L}(G))$. Therefore, $L = \mathcal{L}(G)$ and $L_\omega = \lim(\mathcal{L}(G))$.

Given a sequence of states σ, we note $\sigma[i]$, the ith state on the sequence. A trajectory of G on an ω-word $\alpha \in L_\omega$ is an infinite sequence of states σ such that $\sigma[0] = x_0$ and $\sigma[i+1] = \xi(\sigma[i], \alpha[i])$ for $i \geq 0$. Since the execution of a process never terminates and X is finite, successive applications of ξ introduce simple cycles with distinct states on them (except one that begins and ends the cycle). Nevertheless, finite executions can be simulated by using a terminal state in which the processes continually execute a *wait* action that lasts, for example, one time unit. If this action is controllable, selfloops labeled by *wait* can be used in conjunction with some control requirements to introduce specific delays at the process level.

A realization of a supervisor S for a DES G is a pair $(\mathcal{M}, \phi)$, where $\mathcal{M} = (Q, A, \delta, q_0)$ is a transition structure and ϕ: $Q \rightarrow \Gamma$ a feedback function such that for each $k \in L^S$, $\phi(\delta(q_0, k)) = S(k)$. In this paper, a realization of a supervisor is called a controller. The combination of a DES and a controller constitutes a closed-loop system. As usual, the transition structure $\mathcal{M}$ mimics the behavior of G and function ϕ determines the set of permissible actions for G in each step of the execution of the closed-loop system.

III. Control Requirements

The temporal logic that we have adopted to specify the control requirements is MTL [4], [23]. In this logic, time constraints are associated with modal operators. It allows expression of various properties such as "eventually, within t time units, property p will be satisfied" or "property p must always be satisfied after t time units."

A. Syntax

MTL formulas are constructed from a finite set of propositional symbols $\mathcal{P}$; the Boolean connectives $\wedge$ (*and*) and $\neg$ (*not*); and the temporal connectives $\bigcirc_{\sim t}$ (*next*), $\Box_{\sim t}$ (*always*), and $U_{\sim t}$ (*until*), where $\sim$ denotes $\leq$, $<$, $\geq$, or $>$ and $t \in \mathbb{R}^+$. The formula formation rules are:

1) every propositional symbol $p \in \mathcal{P}$ is a formula;
2) if f, f_1, and f_2 are formulas, then so are $\neg f$, $f_1 \wedge f_2$, $\bigcirc_{\sim t} f$, $\Box_{\sim t} f$, and $f_1 U_{\sim t} f_2$.

In addition to these basic rules, we use the abbreviations $f_1 \vee f_2 \equiv \neg(\neg f_1 \wedge \neg f_2)$ (f_1 or f_2), $f_1 \rightarrow f_2 \equiv \neg f_1 \vee f_2$ (f_1 implies f_2), and $\Diamond_{\sim t} f \equiv true\ U_{\sim t} f$ (eventually f). The language also includes the constant propositional symbols *true* and *false*, which denote valid $(\neg p \vee p)$ and inconsistent $(\neg p \wedge p)$ formulas, respectively.

The intuitive meaning of MTL formulas is captured by using the natural language interpretation for connectives and by noting that, when a time constraint "$\sim t$" is associated with a temporal connective, the modal formula must hold within a time period that satisfies the relation $\sim t$. For example, $\bigcirc_{\geq t} f$ is read as "the next state is in the semi-open time interval $[t, \infty)$ and satisfies f"; $\Box_{\leq t} f$ as "always f on the closed time interval $[0, t]$"; and $\Diamond_{<t} f$ as "eventually f on the semi-open time interval $[0, t)$."

Although positive real numbers are used for specifying time constraints, the control requirements will be sampled only at time points that interact with discrete transitions.

B. Semantics

MTL formulas are interpreted over models of the form $M = \langle \sigma, \pi, T \rangle$, where

1) σ is a trajectory;
2) π: $\mathbb{N} \times \mathcal{P} \rightarrow \{\texttt{true}, \texttt{false}\}$ is a binary function that evaluates a propositional symbol p at $\sigma[i]$, that is, $\pi(i, p)$ returns `true` if p holds at $\sigma[i]$, `false` otherwise;
3) T: $\mathbb{N} \rightarrow \mathbb{R}^+$ is a function that assigns the time stamp $T(i)$ to position i.

We write $\langle M, i \rangle \models f$ if formula f holds at position i in the trajectory σ of M. When the model is understood, we simply write $\sigma[i] \models f$. In addition to the standard rules for Boolean connectives, we use the following rules for temporal connectives. For a position i, $i \geq 0$, a propositional symbol p, formulas f, f_1, and f_2:

1) $\sigma[i] \models p$ *iff* $\pi(i, p)$ returns `true`;
2) $\sigma[i] \models \bigcirc_{\sim t} f$ *iff* $T(i+1) \sim T(i) + t$ and $\sigma[i+1] \models f$;
3) $\sigma[i] \models \Box_{\sim t} f$ *iff* for all j, $j \geq i$, $\sigma[j] \models f$ whenever $T(j) \sim T(i) + t$;
4) $\sigma[i] \models f_1 U_{\sim t} f_2$ *iff* there exists j, $j \geq i$, such that $T(j) \sim T(i) + t$ and $\sigma[j] \models f_2$, and for all k, $i \leq k < j$, $\sigma[k] \models f_1$ whenever $T(k) \sim T(i) + t$.

Finally, we say that model M (or trajectory σ) satisfies a formula f if $\sigma[0] \models f$.

C. Safety and Liveness Properties

In general, the control requirements include interconnected safety and liveness properties [25]. A safety property is expressed by formulas of the form $\bigcirc_{\sim t} f$, $\Box_{\sim t} f$, $f_1 U_{<t} f_2$, or $f_1 U_{\leq t} f_2$. It is characterized by the fact that, when it is violated, the violation occurs on a finite prefix of a trajectory. For example, the violation of a deadline, conveyed by the *until* connective with a constraint "$\leq t$," occurs when a finite number of transitions, for which the sum of durations is greater than t, has been traversed without satisfying f_2, while f_1 was satisfied. A liveness property is expressed by formulas of the form $f_1 U_{>t} f_2$ or $f_1 U_{\geq t} f_2$. It is characterized by the fact that it can only be violated on an infinite trajectory [2]. For such a formula with a constraint "$> t$," there are no bounds on the time when f_2 should occur after t time units. In other words, it must be checked over the infinite open time interval (t, ∞). Of course, such formulas also involve the safety property of maintaining f_1 true as long as f_2 is not made true.

This characterization of safety and liveness properties is important in the description of the synthesis algorithm. In fact, the difficult part of the algorithm deals with the management of formulas that express liveness properties.

D. Positive Normal Form

Safety and liveness properties can be syntactically determined by checking their main temporal connectives. One must, however, take into account the fact that the *not* connective changes the temporal modalities as indicated by the following equivalences:

$$\neg(\Box_{\sim t} f) \Leftrightarrow \Diamond_{\sim t} \neg f \tag{E1}$$

$$\neg(f_1 U_{\sim t} f_2) \Leftrightarrow (\Box_{\sim t} \neg f_2) \vee (\neg f_2 U_{\sim t}(\neg f_1 \wedge \neg f_2)). \tag{E2}$$

To avoid checking these equivalences, we assume that the initial formula representing the control requirements is written in *positive normal form*.[2] This can be done by using the usual De Morgan laws, equivalences (E1) and (E2), and the following equivalence:

$$\neg \bigcirc_{\sim t} f \Leftrightarrow (\bigcirc_{\sim t} \neg f) \vee \bigcirc_{\overline{\sim} t} true \tag{E3}$$

where $\overline{\sim}$ denotes the converse of the ordering relation $\sim$.

E. Decomposition of Formulas

The assessment of an MTL formula over a trajectory is based on the observation that a formula specifies a *present* requirement that must be satisfied in the current state and a *future* requirement that must be satisfied in the next state in the trajectory. In order to formalize this observation, the $\bigcirc_d$*-formula* is introduced, where d is a strictly positive real number representing the duration of a transition between two consecutive states in a trajectory. A subformula having $\bigcirc_d$ as main operator represents a future requirement that must hold in the next state. This operator is not included in the requirements specification language, but it helps to explain how a formula is decomposed. The semantic rule for this formula is

$$\sigma[i] \models \bigcirc_d f \text{ iff } T(i+1) - T(i) = d \text{ and } \sigma[i+1] \models f.$$

The decomposition of an MTL formula is based on the following equivalences[3]:

$$\bigcirc_{\sim t} f \Leftrightarrow \begin{cases} \bigcirc_d f, & \text{if } d \sim t \\ false, & \text{otherwise} \end{cases} \tag{E4}$$

$$\Box_{\leq t} f \Leftrightarrow \begin{cases} f \wedge \bigcirc_d \Box_{\leq t-d} f, & \text{if } d \leq t \\ f, & \text{otherwise} \end{cases} \tag{E5}$$

$$\Box_{\geq t} f \Leftrightarrow \begin{cases} \bigcirc_d \Box_{\geq t-d} f, & \text{if } d \leq t \\ \bigcirc_d \Box_{\geq 0} f, & \text{if } d > t \text{ and } t \neq 0 \\ f \wedge \bigcirc_d \Box_{\geq 0} f, & \text{if } t = 0 \end{cases} \tag{E6}$$

$$f_1 U_{\leq t} f_2 \Leftrightarrow \begin{cases} f_2 \vee (f_1 \wedge \bigcirc_d f_1 U_{\leq t-d} f_2), & \text{if } d \leq t \\ f_2, & \text{otherwise} \end{cases} \tag{E7}$$

$$f_1 U_{\geq t} f_2 \Leftrightarrow \begin{cases} \bigcirc_d f_1 U_{\geq t-d} f_2, & \text{if } d \leq t \\ \bigcirc_d f_1 U_{\geq 0} f_2, & \text{if } d > t \text{ and } t \neq 0 \\ f_2 \vee (f_1 \wedge \bigcirc_d f_1 U_{\geq 0} f_2), & \text{if } t = 0. \end{cases} \tag{E8}$$

It should be noted that, when equivalences (E4)–(E8) are applied recursively[4] to a formula in positive normal form, the result is an equivalent formula in the same form because no negation is introduced by these rules.

F. Disjunctive Normal Form

The goal of the decomposition is to obtain formulas of the form $\bigvee_i(\mathrm{present}_i \wedge \bigcirc_d \mathrm{future}_i)$, where $\mathrm{present}_i$ is a conjunction of literals[5] and future_i a conjunction of literals and formulas for which the main connective is $\bigcirc$, $\Box$, or U. This is done by 1) recursively using equivalences (E4)–(E8) and 2) transforming formulas in positive normal form into equivalent formulas in *disjunctive normal form*.[6] This transformation, which also preserves the positive normal form, requires the usual distributive laws between the connectives $\wedge$ and $\vee$ and the following equivalence:

$$\bigcirc_d(f_1 \wedge f_2) \Leftrightarrow (\bigcirc_d f_1 \wedge \bigcirc_d f_2). \tag{E9}$$

In the sequel, the decomposition of a formula f with respect to a transition of duration d will be noted as follows:

$$\bigvee_{l=1}^{n} (\mathrm{pre}_l^f \wedge \bigcirc_d \, \mathrm{fut}_l^f).$$

IV. Synthesis Method

Let $G = (X, \mathcal{P}, \lambda, A, \tau, \xi, x_0)$ be a TTG describing the behavior of a process and f an MTL formula in disjunctive normal form representing the control requirements. The process of synthesizing a controller $(\mathcal{M}, \phi)$ for DES G and formula f involves simultaneous operations that are performed incrementally based on a forward-chaining search and a control-directed backtracking mechanism.

The basic operation is the expansion of a finite labeled directed graph that represents a combination of f and trajectories of G. This expansion involves a verification of f over trajectories of G. During this operation, dead ends or bad cycles may be detected according to the nature of the formula to be checked. Violations of safety properties lead to dead ends, while violations of liveness properties lead to bad cycles. When a dead end or a bad cycle is detected, a backtracking mechanism goes further back on an uncontrollable path of arbitrary but finite length to select an alternate path. Finally, a controller is obtained by extracting a subgraph, representing the transition structure $\mathcal{M}$, from satisfactory trajectories of G and by updating, for some vertices, the value of the feedback function ϕ during the backtracking operation.

A. Expansion of a Graph

Let $D = (V, E, A)$ be a labeled directed graph, where V is a finite set of vertices, E is a finite set of directed edges, and A is the set of actions labeling the edges. Every vertex $v \in V$ is labeled with a state of G, a formula, and a set of unbounded-time eventualities. These labels are denoted $v.\mathcal{X}$, $v.\mathcal{F}$, and $v.\mathcal{E}$, respectively. The first label is used to record trajectories of G. The second label is a subformula of f that must be satisfied over trajectories of G starting from $v.\mathcal{X}$. The last label allows the verification of liveness properties.

[2] In this form, only propositional symbols are negated.

[3] We only give the equivalences for temporal connectives with time constraints $\leq$ and $\geq$. The equivalences for $<$ and $>$ are similar.

[4] The recursion is applied to subformulas not in the scope of the connective $\bigcirc_d$.

[5] A literal is a propositional symbol or the negation of a propositional symbol.

[6] A formula in disjunctive normal form is a disjunction $g_1 \vee \cdots \vee g_n$ such that each disjunct g_i is a conjunction $h_1 \wedge \cdots \wedge h_{m_i}$, where each conjunct h_j is a literal or a formula whose main connective is $\bigcirc$, $\Box$, or U.

A formula in the set of unbounded-time eventualities is the second operand of a formula of the form $gU_{\geq 0}h$[7] that must be eventually satisfied from $v.\mathcal{X}$. Initially, a vertex v_0 labeled with x_0, f_0 a disjunct of f, and an empty set of eventualities ($v_0.\mathcal{X} = x_0$, $v_0.\mathcal{F} = f_0$, and $v_0.\mathcal{E} = \emptyset$) is created and inserted into V. There are as many initial vertices as there are disjuncts in f. Only one is selected at a time. Let v_i be a vertex of the graph such that $v_i.\mathcal{F} \neq false$. For an action a such that $\xi(v_i.\mathcal{X}, a)$ is defined, a successor v_{i+1} is generated and an edge from v_i to v_{i+1} labeled a is added in E. The state labeling v_{i+1} is $\xi(v_i.\mathcal{X}, a)$. The two other labels are obtained by progressing $v_i.\mathcal{F}$ and $v_i.\mathcal{E}$ with respect to information contained in the edge from v_i to v_{i+1} (this is further discussed in the next section). The vertex v_{i+1} is inserted into V if it is not already there.

B. Progression of Formulas and Sets of Eventualities

Let a be the action labeling an edge from v_i to v_{i+1}. Let us suppose that $v_i.\mathcal{X} = x_i$, $v_i.\mathcal{F} = f_i$, where f_i is a disjunct of the form $f_1^i \wedge \cdots \wedge f_{m_i}^i$, $v_i.\mathcal{E} = E_i$, $v_{i+1}.\mathcal{X} = x_{i+1}$, and $\tau(a) = d_i$. The truth value of f_i on trajectory $x_i x_{i+1} \cdots$ is established by evaluating a present requirement at x_i and postponing a future requirement to be checked at x_{i+1}. This is accomplished by:

1) applying recursively equivalences (E4)–(E8) on f_i;
2) transforming the obtained formula to get a formula of the form (see Section III-F)

$$\bigvee_{l=1}^{n_i}(\mathrm{pre}_l^{f_i} \wedge \bigcirc_{d_i}\mathrm{fut}_l^{f_i}) \qquad \text{(A1)}$$

3) selecting the disjunct $\mathrm{pre}_r^{f_i} \wedge \bigcirc_{d_i}\mathrm{fut}_r^{f_i}$ in the previous formula, for some r ($1 \leq r \leq n_i$).

To check f_i at state x_i, the present requirement $\mathrm{pre}_r^{f_i}$ is assessed at x_i. If the present requirement is violated, then $v_{i+1}.\mathcal{F} = false$ (since $false \wedge f \Leftrightarrow false$, for any f), otherwise $v_{i+1}.\mathcal{F} = \mathrm{fut}_r^{f_i}$. If there is no future requirement, then $v_{i+1}.\mathcal{F} = true$. A vertex labeled with *false* is defined as a dead end during the expansion of D.

The set of unbounded-time eventualities E_{i+1} labeling v_{i+1} is defined as follows:

$$E_{i+1} := \begin{cases} \overline{\mathcal{E}}(\{f_1^i, \cdots, f_{m_i}^i\}), & \text{if } E_i = \emptyset \\ E_i - \mathcal{E}(E_i), & \text{otherwise} \end{cases} \qquad \text{(A2)}$$

where $\overline{\mathcal{E}}(\{f_1^i, \cdots, f_{m_i}^i\})$ returns the set of formulas h such that there exists f_p^i ($1 \leq p \leq m_i$) of the form $gU_{\geq 0}h$ with h *not locally entailed* by the edge from v_i to v_{i+1}; and $\mathcal{E}(E_i)$ returns the set of formulas h included in E_i and *locally entailed* by the edge from v_i to v_{i+1}. A formula h is locally entailed by an edge from v_i to v_{i+1} if there exists $1 \leq q \leq n$ such that pre_q^h is true at $v_i.\mathcal{X}$ and fut_q^h is *locally entailed* by the vertex v_{i+1} whenever $h = \bigvee_{l=1}^{n}(\mathrm{pre}_l^h \wedge \bigcirc_d \mathrm{fut}_l^h)$.

Establishing the fact that a formula f in disjunctive normal form is locally entailed by a vertex v is based on an inference procedure applied to a set of premises formed from all the conjuncts of $v.\mathcal{F}$. The *modus ponens*, "*and introduction*," and following axiom schemata are used by the inference procedure:

$$f_1 \to f_1 \vee f_2 \qquad \text{(S1)}$$

$$\bigcirc_{\sim t} f \to \bigcirc_{\sim t'} f, \quad \text{if } t \sim t' \qquad \text{(S2)}$$

$$\Box_{\sim t} f \to \Box_{\sim t'} f, \quad \text{if } t' \sim t \qquad \text{(S3)}$$

$$f_1 U_{\leq t} f_2 \to f_1 U_{\leq t'} f_2, \quad \text{if } t \leq t' \qquad \text{(S4)}$$

$$f_1 U_{\geq t} f_2 \to f_1 U_{\geq t} f_2. \qquad \text{(S5)}$$

Intuitively, the inference procedure replaces each conjunct of f, whose main connective is $\bigcirc$, $\Box$, or U, by *true* or *false* according to the fact that one of the schemata (S2)–(S5) can be applied to a conjunct of $v.\mathcal{F}$ (left pattern) and the temporal subformula of f (right pattern). A conjunct of f that is a propositional symbol is replaced by *true* if it is a conjunct of $v.\mathcal{F}$. Otherwise, it is replaced by *false*. Then, the obtained propositional formula is evaluated. If it is true, then f is locally entailed by v. This inference procedure is sound but incomplete because temporal subformulas are not decomposed further and a limited number of axiom schemata are used. However, as it is proven later, this completeness property is not required for the completeness of the synthesis algorithm.

A cycle is satisfactory if it contains at least one vertex labeled with an empty set of eventualities. A cycle that does not meet this criterion is defined as a bad cycle. In other words, a cycle is satisfactory when its infinite execution does not lead to pending unsatisfied eventualities.

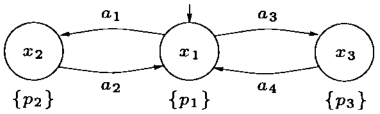


Fig. 1. The process: $L = (a_1a_2 + a_3a_4)^*(\epsilon + a_1 + a_3)$; $L_\omega = (a_1a_2 + a_3a_4)^\omega$.

V. Examples

Let us consider a process modeled as the TTG of Fig. 1. Every action lasts one time unit, that is, $\tau(a_i) = 1$ ($1 \leq i \leq 4$). There are three states, namely, x_1, x_2, and x_3 with $\lambda(x_i) = \{p_i\}$ ($1 \leq i \leq 3$); x_1 is the initial state.

A. Stability Property

In this example, the control requirements are formally specified by the following bounded-time MTL formula in disjunctive normal form:

$$f = \Box_{\geq 0}(p_1 \to \Diamond_{\leq 3} p_2) \wedge \Box_{\geq 0}(p_1 \to \Diamond_{\leq 3} p_3).$$

By recursively applying (E6) and (E7) on f, we obtain successively the following formulas:

$$(p_1 \to \Diamond_{\leq 3} p_2) \wedge (p_1 \to \Diamond_{\leq 3} p_3) \wedge \bigcirc_1 f$$

$$(\neg p_1 \vee p_2 \vee \bigcirc_1 \Diamond_{\leq 2} p_2) \wedge (\neg p_1 \vee p_3 \vee \bigcirc_1 \Diamond_{\leq 2} p_3) \wedge \bigcirc_1 f.$$

During the application of equivalences (E4)–(E8), literals can be evaluated at the current state to make trivial simplifications possible earlier without decomposing further temporal subformulas. Since $p_1 \in \lambda(x_1)$, $p_2 \notin \lambda(x_1)$, and $p_3 \notin \lambda(x_1)$,

[7] We discuss only the case for the *until* connective with time constraints $\geq$. The case for $>$ is similar.

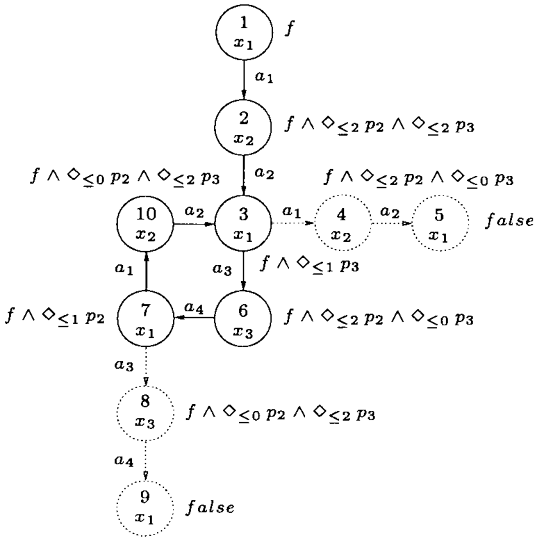

1 x_1
f
a_1
2 x_2
$f \wedge \Diamond_{\leq 2} p_2 \wedge \Diamond_{\leq 2} p_3$
$f \wedge \Diamond_{\leq 0} p_2 \wedge \Diamond_{\leq 2} p_3$
a_2
$f \wedge \Diamond_{\leq 2} p_2 \wedge \Diamond_{\leq 0} p_3$
10 x_2
a_2
3 x_1
a_1
4 x_2
a_2
5 x_1
$false$
a_1
a_3
$f \wedge \Diamond_{\leq 1} p_3$
$f \wedge \Diamond_{\leq 1} p_2$
7 x_1
a_4
6 x_3
$f \wedge \Diamond_{\leq 2} p_2 \wedge \Diamond_{\leq 0} p_3$
a_3
8 x_3
$f \wedge \Diamond_{\leq 0} p_2 \wedge \Diamond_{\leq 2} p_3$
a_4
9 x_1
$false$

Fig. 2. A part of a graph developed from safety properties.

we obtain only one valid disjunct with the present requirement *true* and a future requirement $f_1 = f \wedge \Diamond_{\leq 2} p_2 \wedge \Diamond_{\leq 2} p_3$. This means that, over the next states, formula f must be satisfied; p_2 and p_3 must be satisfied before two units of time have elapsed. Note that f must always be satisfied from one state to another because its main operator is $\Box_{\geq 0}$. Furthermore, the sets of eventualities are empty because f involves only safety properties.

The progression of f_1 from x_2 to x_1 gives the future requirement $f_2 = f \wedge \Diamond_{\leq 1} p_3$. In fact, since $p_2 \in \lambda(x_2)$, the eventual satisfaction of p_2 is met and erased from the future requirement; time has progressed one unit and p_3 is not satisfied. Finally, the progression with four units of time of f in the sequence of states $x_1 x_2 x_1 x_2 x_1$ yields *false*, because the time limit allowed for satisfying p_3 has expired.

Fig. 2 gives a part of the graph developed from f. Dotted arrows and circles correspond to edges and vertices created during the verification process but rejected because they lead to dead ends. If we assume that actions a_1 and a_3 are controllable, then the controller must prohibit these actions at states 3 and 7, respectively.

B. Unbounded-Time Eventualities

Let us consider the formula in the example in Section V-A, making the time constraint for the *eventually* connectives "≥ 0." In this example, the formula labeling a given vertex can be identical to that labeling an ancestor vertex while not achieving an unbounded-time eventuality. The graph in Fig. 3 illustrates an infinite trajectory that is not satisfactory. The formula $\Diamond_{\geq 0} p_3$, which is a subformula for every vertex in the cycle, is never satisfied. This example shows that equivalences (E4)–(E8) are not sufficient to assess the satisfiability of an unbounded-time formula.

C. Reachability Property

For a more comprehensive example, Fig. 4 shows a description of the graph obtained by progressing the sets of

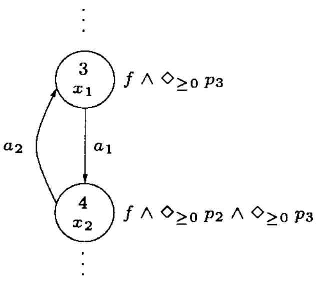

3 x_1
$f \wedge \Diamond_{\geq 0} p_3$
a_2
a_1
4 x_2
$f \wedge \Diamond_{\geq 0} p_2 \wedge \Diamond_{\geq 0} p_3$

Fig. 3. A bad cycle.

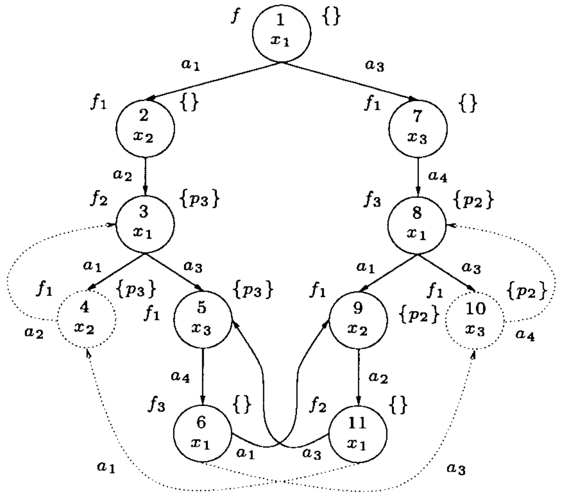


Fig. 4. A graph developed from liveness properties: $f_1 = f \wedge \Diamond_{\geq 0} p_2 \wedge \Diamond_{\geq 0} p_3$; $f_2 = f \wedge \Diamond_{\geq 0} p_3$; $f_3 = f \wedge \Diamond_{\geq 0} p_2$; $W = ((a_1 a_2)^+ a_3 a_4 + (a_3 a_4)^+ a_1 a_2)^\omega$; $W_0 = a_1 a_2 a_3 (a_4 a_1 a_2 a_3)^\omega + a_3 a_4 a_1 (a_2 a_3 a_4 a_1)^\omega$.

eventualities and the formula

$$f = \Box_{\geq 0}(p_1 \rightarrow \Diamond_{\geq 0} p_2) \wedge \Box_{\geq 0}(p_1 \rightarrow \Diamond_{\geq 0} p_3)$$

through the TTG in Fig. 1. It can be checked that any cycle containing a vertex labeled with an empty set of eventualities is satisfactory.

If the graph of Fig. 4 is interpreted as a transition graph of an automaton, then a vertex labeled with an empty set of eventualities represents an accepting state in the sense of Büchi automata. Let us assume again that a_1 and a_3 are controllable. In that case, the legal language W, defined by the formula f, is $*$-controllable w.r.t. L and ω-controllable w.r.t. (L, L_ω) (see Fig. 1 for the definition of L and L_ω) [24], [43]. However, a maximally permissive controller cannot be extracted from this graph because the supremal ω-controllable sublanguage of W (which is itself W in this example) is not ω-closed with respect to L_ω.[8] A family of useful controllers can, however, be extracted by unwinding cycles 3-4-3 and 8-10-8 a finite number of times. In particular, a controller that corresponds to W_0 (see Fig. 4 for the definition of W_0) can be obtained by removing dotted arrows and circles because they correspond to edges and vertices created during the verification process but rejected as constituting bad cycles.

[8] A language L_1 is ω-closed with respect to a language L_2 if $L_1 = \mathrm{clo}(L_1) \cap L_2$.

VI. Properties

In this section, correspondences between good paths of D and trajectories of G that satisfy formula f are established. It is shown how the progression of a formula through good paths is related to its semantic interpretation with respect to the corresponding trajectories. A path of D is good if it contains neither a dead end nor a bad cycle. For any infinite path $v_0 v_1 \cdots$, this is formalized by the following two properties:

$$(\forall\, i \geq 0)(v_i.\mathcal{F} \neq false) \tag{P1}$$

$$(\forall\, i \geq 0)(\exists j \geq i)(v_j.\mathcal{E} = \emptyset). \tag{P2}$$

Since graph D has a finite number of vertices, infinite paths are represented by paths terminated by a cycle. Let us start with a restricted version of the main result.

Theorem 1: For any path $v_0 v_1 \cdots v_j \cdots v_j$ terminated by a cycle and produced by (A1) and (A2), if for all $i \geq 0$, $v_i.\mathcal{F} \neq false$ and there exists $k \geq j$ such that $v_k.\mathcal{E} = \emptyset$, then for any vertex v on the trajectory obtained from the path by unwinding the cycle, $v.\mathcal{X} \models v.\mathcal{F}$.

In order to simplify the proof, we introduce some notations based on the following observations. First, a vertex is always labeled with a conjunction. Second, the satisfaction of a conjunction depends on the satisfaction of all its conjuncts. Third, the decomposition of a conjunct gives a disjunction. Finally, the satisfaction of a disjunction depends on the satisfaction of one of its disjuncts. Therefore, the satisfaction of a conjunction is equivalent to the satisfaction of the conjunction of disjuncts (at least one per conjunct) obtained from the decomposition of every conjunct. Nevertheless, one must take into account the time intervals associated with temporal connectives.

For any vertex v_i, we note d_i the duration associated with the edge from v_i to v_{i+1} and $v_i.\mathcal{F} = f_i = f_1^i \wedge \cdots \wedge f_{m_i}^i$. In order to focus on a particular conjunct, f_i is broken into two parts: $f_i = F_p^i \wedge f_p^i$, where $F_p^i = \bigwedge_{l \neq p} f_l^i$, and $1 \leq p \leq m_i$. The decomposition of $v_i.\mathcal{F}$ at v_i gives

$$\left(\bigvee_{l=1}^{n_i} \mathrm{pre}_l^{F_p^i} \wedge \bigcirc_{d_i} \mathrm{fut}_l^{F_p^i}\right) \wedge \left(\bigvee_{l=1}^{n_i'} \mathrm{pre}_l^{f_p^i} \wedge \bigcirc_{d_i} \mathrm{fut}_l^{f_p^i}\right). \tag{F1}$$

If (P1) holds, then only the disjuncts for which the present requirement is true are relevant and formula (F1) simplifies to

$$\bigvee_{l} \bigvee_{l'} \bigcirc_{d_i} \left(\mathrm{fut}_l^{F_p^i} \wedge \mathrm{fut}_{l'}^{f_p^i}\right). \tag{F2}$$

The proof of Theorem 1 is based on the following six lemmas. The first five lemmas consider the different cases for f_p^i. The last lemma expresses essentially the same property as Theorem 1, but for every conjunct f_p^i.

Lemma 1: If $v_0 v_1 \cdots$ satisfies properties (P1) and (P2), and $f_p^i = \bigcirc_{\sim t} g$ for some p $(1 \leq p \leq m_i)$, then 1) $d_i \sim t$ and 2) $v_{i+1}.\mathcal{F}$ has a conjunct that is a disjunct of the disjunctive normal form of g.

Proof: From (E4), $\bigcirc_{\sim t} g$ is equivalent to $\bigcirc_{d_i} g$ (and $d_i \sim t$) or *false*. From (P1), $v_{i+1}.\mathcal{F} \neq false$. Thus, the only possible case is the first one. By using (A1) and (F2)

$$v_{i+1}.\mathcal{F} = \mathrm{fut}_q^{F_p^i} \wedge \mathrm{fut}_r^{f_p^i}$$

for some q $(1 \leq q \leq n_i)$ and r $(1 \leq r \leq n_i')$, where $\mathrm{fut}_r^{f_p^i}$ is a disjunct of the disjunctive normal form of g. Then the conclusion follows. ■

Lemma 2: If $v_0 v_1 \cdots$ satisfies properties (P1) and (P2), and $f_p^i = gU_{\geq t} h$, for some p $(1 \leq p \leq m_i)$, then 1) there exists a vertex v_j such that $j = i$ and $t = 0$ or $j > i$ and $d_i + \cdots + d_{j-1} \geq t$ and $v_{j+1}.\mathcal{F}$ has conjuncts that imply a disjunct of the future part obtained from the decomposition of h at v_j and 2) for all k $(i \leq k < j)$ such that $k = i$ and $t = 0$ or $k > i$ and $d_i + \cdots + d_{k-1} \geq t$, $v_{k+1}.\mathcal{F}$ has a conjunct that is a disjunct of the future part obtained from the decomposition of g at v_k.

Proof: For a formula of the form $gU_{\geq t} h$, the satisfaction of g must be established for every state after a delay of t time units and until h holds (after the same delay). If $t = 0$, this means from state $v_i.\mathcal{X}$; otherwise, from a state $v_l.\mathcal{X}$ such that $d_i + \cdots + d_{l-1} \geq t$, but $d_i + \cdots + d_{l-2} < t$. The progression of the formula is accomplished step-by-step by repeatedly considering the current duration until the delay is expired. There are three possibilities: 1) $d_i \leq t$; 2) $d_i > t$ and $t \neq 0$; and 3) $t = 0$.

Case I: If $d_i \leq t$, then the delay is not expired. From (E8), $gU_{\geq t} h$ is equivalent to $\bigcirc_{d_i} gU_{\geq t - d_i} h$. Without loss of generality, let us assume that $p = 1$. By using (A1) and (F2)

$$v_{i+1}.\mathcal{F} = (gU_{\geq t - d_i} h) \wedge \mathrm{fut}_q^{F_1^i}$$

for some q $(1 \leq q \leq n_i)$. We can repeat this reasoning with vertices v_{i+1}, v_{i+2}, and so on. Since all the action durations are strictly positive, it follows that there exists a vertex v_j $(j > i)$ such that $t' = t - d_i - \cdots - d_{j-1} \geq 0$, $d_j > t'$, and

$$v_j.\mathcal{F} = (gU_{\geq t'} h) \wedge \mathrm{fut}_{q'}^{F_1^{j-1}}$$

for some q' $(1 \leq q' \leq n_{j-1})$. Then, either $t' \neq 0$ or $t' = 0$. If $t' \neq 0$, then v_j is in the same situation as v_i in Case II; otherwise, v_j is in the same situation as v_i in Case III. Either way, we continue with the argumentation of Cases II or III by replacing v_i with v_j.

Case II: If $d_i > t$ and $t \neq 0$, then the delay is residual. From (E8), $gU_{\geq t} h$ is equivalent to $\bigcirc_{d_i} gU_{\geq 0} h$. By using (A1) and (F2)

$$v_{i+1}.\mathcal{F} = (gU_{\geq 0} h) \wedge \mathrm{fut}_q^{F_1^i}$$

for some q $(1 \leq q \leq n_i)$. Thus, v_{i+1} is in the same situation as v_i in Case III. Hence, we continue with the argumentation of Case III by replacing v_i with v_{i+1}.

Case III: If $t = 0$, then the delay is expired and the progression of g must be considered until the progression of h or the progression of a formula that implies h. From (E8), $gU_{\geq t} h$ is equivalent to $h \vee (g \wedge \bigcirc_{d_i} gU_{\geq 0} h)$. By using (A1) and (P1), formula (F2) becomes

$$\left(\bigvee_{l} \bigvee_{l'} \bigcirc_{d_i} \left(\mathrm{fut}_l^{F_1^i} \wedge \mathrm{fut}_{l'}^{h^i}\right)\right) \vee \left(\bigvee_{l} \bigvee_{l'} \bigcirc_{d_i} \left(\mathrm{fut}_l^{F_1^i} \wedge \mathrm{fut}_{l'}^{g^i} \wedge gU_{\geq 0} h\right)\right).$$

There are two possibilities.

1) $v_{i+1}.\mathcal{F} = \mathrm{fut}_r^{h^i} \wedge \mathrm{fut}_q^{F_1^i}$, for some q and r.
2) $v_{i+1}.\mathcal{F} = (gU_{\geq 0}h) \wedge \mathrm{fut}_r^{g^i} \wedge \mathrm{fut}_q^{F_1^i}$, for some q and r.

Case III-1): Since $\mathrm{fut}_r^{h^i}$ is a disjunct of the future part obtained from the decomposition of h at v_i, part 1) is satisfied. Moreover, since $j = i$, part 2) is trivially satisfied.

Case III-2): Since $v_{i+1}.\mathcal{F}$ contains $gU_{\geq 0}h$ as a conjunct, we can resume the reasoning from the beginning of Case III, but applied to v_{i+1}. Similarly, we can repeat this for v_{i+2}, and so on. It follows that, for any vertex v_k $(k \geq i)$ before a vertex v_j (assuming that it exists) satisfying part 1), $v_{k+1}.\mathcal{F}$ must be of the form

$$(gU_{\geq 0}h) \wedge \mathrm{fut}_{r'}^{g^k} \wedge \mathrm{fut}_{q'}^{F_1^k}$$

for some q' and r'. This means that all the vertices before any such a vertex v_j satisfy part 2). Thus, to complete the proof, it remains to show that a vertex v_j satisfying part 2) effectively exists.

From (P2), there exists v_l after v_i such that $v_l.\mathcal{E} = \emptyset$. Either there exists a vertex v_j between v_i and v_l (inclusively) such that v_j satisfies part 1) or no such vertex exists. If such a vertex v_j exists, then this trivially ends the proof. Now, if no such vertex v_j exists, this means that all the vertices between v_i and v_l fall in Case III-2). Hence, $v_l.\mathcal{F}$ is of the form

$$(gU_{\geq 0}h) \wedge \mathrm{fut}_{r''}^{g^{l-1}} \wedge \mathrm{fut}_{q''}^{F_1^{l-1}}$$

for some q'' and r''. From (A2), $v_{l+1}.\mathcal{E}$ must contain h (unless h is locally entailed by the edge from v_l to v_{l+1}). But then, from (P2), there must exist a vertex $v_{l'}$ after v_{l+1} such that $v_{l'}.\mathcal{E} = \emptyset$. From (A2), this means that there exists a vertex $v_{j'}$ between v_l and $v_{l'}$ such that $v_{j'+1}.\mathcal{E}$ is obtained by removing h from $v_{j'}.\mathcal{E}$ because h is locally entailed by the edge from $v_{j'}$ to $v_{j'+1}$. Then, propositional symbols in $\lambda(v_{j'}.\mathcal{X})$ and conjuncts of $v_{j'+1}.\mathcal{F}$ logically imply h. More precisely:

1) $\mathrm{pre}_{r'''}^{h^{j'}}$ (a disjunct of the present part obtained from the decomposition of h at $v_{j'}$) is true in $v_{j'}.\mathcal{X}$;
2) $\mathrm{fut}_{r'''}^{h^{j'}}$ (the corresponding disjunct of the future part obtained from the decomposition of h at $v_{j'}$) is implied by $v_{j'+1}.\mathcal{F}$ for some r'''. Hence, $v_{j'}$ satisfies part 1). ■

The proofs of the next three lemmas are based on arguments developed in Lemma 2. In fact, the first line of (E7) is like the last line of (E8), and (E6) is like (E8) if f_2 in (E8) is replaced by *false*, that is, $\Box_{\geq t} g$ is almost equivalent to $gU_{\geq t}\mathit{false}$.

Lemma 3: If $v_0v_1 \cdots$ satisfies properties (P1) and (P2), and $f_p^i = gU_{\leq t}h$, for some p $(1 \leq p \leq m_i)$, then 1) there exists a vertex v_j such that $j = i$ or $j > i$ and $d_i + \cdots + d_{j-1} \leq t$ and $v_{j+1}.\mathcal{F}$ has a conjunct that is a disjunct of the future part obtained from the decomposition of h at v_j and 2) for all k $(i \leq k < j)$, $v_{k+1}.\mathcal{F}$ has a conjunct that is a disjunct of the future part obtained from the decomposition of g at v_k.

Proof: The proof is similar to that for Lemma 2 in Case III since, for t greater than or equal to the duration of the current action, the progression of $gU_{\leq t}h$ is similar to the progression of $gU_{\geq 0}h$. From (E7) and (A1), sooner or later, a vertex, for which the time constraint for the *until* connective is less than the duration of the current action, will be ultimately reached. From (P1), h must be progressed from v_i or v_j $(j > i$ and $d_i + \cdots + d_{j-1} \leq t)$. ■

Lemma 4: If $v_0v_1 \cdots$ satisfies properties (P1) and (P2), and $f_p^i = \Box_{\geq t} g$, for some p $(1 \leq p \leq m_i)$, then for $j = i$ whenever $t = 0$ and all $j > i$ whenever $d_i + \cdots + d_{j-1} \geq t$, $v_{j+1}.\mathcal{F}$ has a conjunct that is a disjunct of the future part obtained from the decomposition of g at v_j.

Lemma 5: If $v_0v_1 \cdots$ satisfies properties (P1) and (P2), and $f_p^i = \Box_{\leq t} g$, for some p $(1 \leq p \leq m_i)$, then for $j = i$ and all $j > i$ such that $d_i + \cdots + d_{j-1} \leq t$, $v_{j+1}.\mathcal{F}$ has a conjunct that is a disjunct of the future part obtained from the decomposition of g at v_j.

Proof: The proofs of Lemmas 4 and 5 are similar to those for Lemmas 2 and 3. In fact, $\Box_{\sim t} g$ is almost equivalent to $gU_{\sim t}\mathit{false}$ and is progressed like an *until* formula, except that we do not have to check that *false* is eventually satisfied. ■

Lemma 6: If $v_0v_1 \cdots$ satisfies properties (P1) and (P2), then for all $i \geq 0$ and for all p $(1 \leq p \leq m_i)$, $v_i.\mathcal{X} \models f_p^i$.

Proof: The proof is by induction on the structure of formulas. We first prove the case for $f_p^i = \Box_{\leq t} g$. The basis is when g is a propositional symbol. Given (P1) and (E5), then $v_j.\mathcal{X} \models g$ for $j = i$ and all $j > i$ whenever $d_i + \cdots + d_{j-1} \leq t$. From the semantic definition of MTL, $v_i.\mathcal{X} \models \Box_{\leq t} g$. The inductive hypothesis is: if $v_0v_1 \cdots$ satisfies properties (P1) and (P2), then for all $k \geq 0$ and all $(f')_q^k$ that are conjuncts of $v_k.\mathcal{F}$ with simpler structure that f_p^i, then $v_k.\mathcal{X} \models (f')_q^k$. From Lemma 5, $v_{j+1}.\mathcal{F}$ has a conjunct that is a disjunct of the future part obtained from the decomposition of g at v_j for $j = i$ and all $j > i$ such that $d_i + \cdots + d_{j-1} \leq t$. This means that the corresponding disjunct of the present part holds at $v_j.\mathcal{X}$. By using the inductive hypothesis, the conjunct of $v_{j+1}.\mathcal{F}$ corresponding to the disjunct of the future part obtained from the decomposition of g at v_j is satisfied at $v_{j+1}.\mathcal{X}$. Therefore, $v_j.\mathcal{X} \models g$ for $j = i$ and all $j > i$ whenever $d_i + \cdots + d_{j-1} \leq t$. From the semantic definition of MTL, $v_i.\mathcal{X} \models \Box_{\leq t} g$. The proof for the other cases are similar. The case $f_p^i = gU_{\geq t}h$ requires, however, more explanation.

The progression of a formula of the form $gU_{\geq t}h$ decreases t by the action duration d, only as long as $d \leq t$. Then, the progression keeps the formula invariant as long as only g is satisfied, but not h. From (P1) the safety property is satisfied. The only problem is that cycles violating the liveness property may be formed. We show that this is impossible. From Lemma 2, $v_{j+1}.\mathcal{F}$ has conjuncts that imply a disjunct of the future part obtained from the decomposition of h at v_j for some j. From the inductive hypothesis, these conjuncts hold at $v_{j+1}.\mathcal{X}$. Thus, $v_j.\mathcal{X} \models h$. From the semantic definition of MTL, $v_i.\mathcal{X} \models gU_{\geq t}h$. ■

Now, we can prove Theorem 1.

Proof: Let $v_0v_1 \cdots v_j \cdots v_j$ be a path terminated by a cycle and produced by (A1) and (A2). If for all $i \geq 0$, $v_i.\mathcal{F} \neq \mathit{false}$, and there exists $k \geq j$ such that $v_k.\mathcal{E} = \emptyset$, then the trajectory obtained by unwinding the cycle satisfies (P1) and (P2). From Lemma 6, for any vertex v_i on that trajectory and for any f_p^i that is a conjunct of $v_i.\mathcal{F}$ $(1 \leq p \leq m_i)$, $v_i.\mathcal{X} \models f_p^i$. But, $v_i.\mathcal{F} = f_1^i \wedge \cdots \wedge f_{m_i}^i$. Hence, $v_i.\mathcal{X} \models v_i.\mathcal{F}$. ■

```
procedure $Expand(v \in V)$
  for each $a \in A$ such that $\xi(v.\mathcal{X}, a)$ is defined do
    $f' := \bigvee_{l=1}^{n}(\mathrm{pre}_l^{v.\mathcal{F}} \wedge \bigcirc_{\tau(a)} \mathrm{fut}_l^{v.\mathcal{F}})$;

    for each disjunct $\mathrm{pre}_l^{v.\mathcal{F}} \wedge \bigcirc_{\tau(a)} \mathrm{fut}_l^{v.\mathcal{F}}$ of $f'$ do
      create a new vertex $v'$;
      $v'.\mathcal{X} := \xi(v.\mathcal{X}, a)$;
      $v'.\mathcal{F} := \mathrm{fut}_l^{v.\mathcal{F}}$;
      $v'.\mathcal{E} := A2(v, a, v')$;
      create a new edge $(v, a, v')$.
```

Fig. 5. The expansion of a vertex.

The next theorem generalizes Theorem 1. It is based on the procedure *Expand* (see Fig. 5) that generates successors of a vertex v. For every action a such that $\xi(v.\mathcal{X}, a)$ is defined, it decomposes formula $v.\mathcal{F}$ at v [items 1 and 2 of (A1)]. Then for every disjunct of the decomposition of $v.\mathcal{F}$ at v, it creates a new vertex v', assigns values to its labels $v'.\mathcal{X}$, $v'.\mathcal{F}$, and $v'.\mathcal{E}$, and creates the corresponding edge. The notation $A2(v, a, v')$ means that (A2) is applied by replacing v_i by v and v_{i+1} by v'. It should be noted that if there is no future requirement, then $v'.\mathcal{F} = true$. In addition, if the present requirement is violated at v, then $v'.\mathcal{F} = false$.

Theorem 2: For any trajectory of G terminated by a cycle $x_0 x_1 \cdots x_l \cdots x_l$, the trajectory obtained by unwinding the cycle satisfies an MTL formula f if and only if there exists a graph produced by *Expand* that contains a path terminated by a cycle $v_0 v_1 \cdots v_j \cdots v_j$ such that 1) for all $i \geq 0$, $v_i.\mathcal{F} \neq false$; 2) there exists $k \geq j$ such that $v_k.\mathcal{E} = \emptyset$; and 3) $v_0.\mathcal{X} = x_0$ and, for any $v_{i'}$ and x_i, if $v_{i'}.\mathcal{X} = x_i$, then $v_{i'+1}.\mathcal{X} = x_{i+1}$.

Proof: ($\Leftarrow$) This follows trivially from Theorem 1.

($\Rightarrow$) The idea of the proof is to build paths from trajectory $x_0 x_1 \cdots x_l \cdots x_l$ by progressively adding labels to vertices, starting with states, then disjunctions, then disjuncts (or conjunctions), and finally sets of eventualities. Thereafter, two vertices are equal if their labels match.

Let us build a first path $u_0 u_1 \cdots u_l \cdots u_l$, where $u_i.\mathcal{X} = x_i$ $(0 \leq i \leq l)$. This path satisfies trivially part 3). Let us build another path $v_0 v_1 \cdots v_l \cdots v_{l'} \cdots v_{l'}$ from the previous one[9] by unwinding its cycle a finite number of times as follows: $v_0.\mathcal{F} = f_0 = f$ and $v_i.\mathcal{F} = f_i = \bigvee_l \mathrm{fut}_l^{f_{i-1}}$ (only the future parts of disjuncts for which the corresponding present parts are true at $v_{i-1}.\mathcal{X}$ appear in this disjunction) $(1 \leq i \leq l')$. The cycle of the first path is unwound a finite number of times because items 1 and 2 of (A1) generate only finitely many different formulas. From (E4)–(E8), $v_1.\mathcal{X} \models f_1$, since $v_0.\mathcal{X} \models f_0$ (by hypothesis). Similarly, since $v_1.\mathcal{X} \models f_1$, then $v_2.\mathcal{X} \models f_2$ and so on. In particular, $f_i \neq false$ for all i. Hence, this path satisfies part 1).

For any f_i, let us note $f_1^i \vee \cdots \vee f_{n_i}^i$ (if $i \neq 0$, $f_p^i = \mathrm{fut}_q^{f_{i-1}}$) its disjunctive normal form. For every vertex v_i, there exists at least one f_q^i $(1 \leq q \leq n_i)$ such that $v_i.\mathcal{X} \models f_q^i$. Then, for every f_q^i such that $v_i.\mathcal{X} \models f_q^i$, there exists a disjunct f_r^{i+1} $(1 \leq r \leq n_{i+1})$ such that $x_{i+1}.\mathcal{X} \models f_r^{i+1}$ and f_r^{i+1} is a disjunct of the future part obtained from the decomposition f_q^i at v_i, since the decomposition of a formula is join-preserving.

Again, let us build another path $v_0' v_1' \cdots v_{l'}' \cdots v_{l''}' \cdots v_{l''}'$ from the previous one by unwinding its cycle a finite number of times and replacing the formulas of vertices by corresponding disjuncts as follows: $v_0'.\mathcal{F} = f_{p_0}^0$, $v_i'.\mathcal{F} = f_{p_i}^i$ if $f_{p_i}^i$ is a disjunct of the future part obtained from the decomposition of $f_{p_{i-1}}^{i-1}$ at v_{i-1}' $(1 \leq i \leq l'')$, and $v_i'.\mathcal{X} \models f_{p_i}^i$ $(0 \leq i \leq l'')$. The cycle of the second path is unwound a finite number of times because there is a finite number of disjuncts. It should be noted that the third path satisfies parts 1) and 3).

Let us build a last path $w_0 w_1 \cdots w_{l''} \cdots w_{l'''} \cdots w_{l'''}$ from the previous one by unwinding its cycle a finite number of times and extending each vertex with a set of eventualities as follows: $w_0.\mathcal{E} = \emptyset$ and $w_i.\mathcal{E} = E_i$, accordingly to (A2) $(1 \leq i \leq l''')$. The cycle of the third path is unwound a finite number of times because there is a finite number of different possible sets of unbounded-time eventualities obtained from subformulas of f having the form $gU_{\geq 0}h$.

Let us now consider all such paths [still satisfying parts 1) and 3)] that can be derived from the initial trajectory. At least one of them satisfies part 2), which is proven by contradiction. Let us suppose that there exists no path satisfying part 2). This means that, on the cycle of every path, all the sets of eventualities contain at least one formula h. Then, every path has a vertex w, before the entry in the cycle, such that $w.\mathcal{E} = \emptyset$ and $w.\mathcal{F}$ has a conjunct of the form $gU_{\geq 0}h$ (and h is not locally entailed by the edge having w as head). Since all the paths satisfy 1), there exists a vertex w' after w such that $w'.\mathcal{X} \models h$ and $w'.\mathcal{F}$ has a conjunct of the form $gU_{\geq 0}h$. In that case, the decomposition of $w'.\mathcal{F}$ can be done in two ways according to (E8) (see also Case III of Lemma 2). We consider only the first way, which corresponds to a particular path still satisfying parts 1) and 3). Let us focus on a particular set of eventualities. In the worst situation, h is not locally entailed by an edge before the edge having w' as head, which means that it cannot be detected earlier that the formula h will be effectively satisfied and therefore be removed from the set of eventualities. The successor vertex w'' of w' has the following characteristics: $w''.\mathcal{F}$ has a disjunct of the future part obtained from the decomposition h at w' and the corresponding present part is true at w'. Then, h is locally entailed by the edge from w' to w''. From (A2), $w''.\mathcal{E}$ does not contain h. The same reasoning can be done with the other formulas in the set of eventualities appearing on the cycle to conclude that this set will become empty. This contradicts the hypothesis.

Finally, let us construct a family of graphs by grouping and interleaving all the paths having the same initial vertex. Each graph is a subgraph of a graph that can be generated by *Expand*. In at least one graph, there is a path that satisfies parts 1)–3). Then, the conclusion follows. ■

VII. The Synthesis Algorithm

The algorithm described in this section provides more details on (A1) and (A2) in Section IV-B. It includes elements concerning the use of a search technique for the expansion of

[9] This means that $v_0 = u_0$ and $v_i = \mathrm{succ}(u_j)$, if $v_{i-1} = u_j$. Hence, part 3) is still satisfied.

```
    procedure $Synthetize()$
 1. create a linked list of initial vertices;
 2. $head$ := first element of this list;
 3. $Expand(head)$; $\phi(head) := \emptyset$; $e := head.first$;
 4. $DFS :=$ true; $safe :=$ true;
 5. while $DFS$ do
 6.    while $safe$ and $e \neq$ nil do
 7.       $tail := e.first$; $success :=$ false;
 8.       while not $success$ and $tail \neq$ nil do
 9.          $Examine()$;
10.       if not $success$ then
11.          if $e.action \in A_{uc}$ then $safe :=$ false;
12.                                        $head.unc :=$ true
13.                                  else $e := e.next$;
14.    $Backtrack()$;
15. return $((Q, A, \delta, head), \phi)$.
```

Fig. 6. The synthesis algorithm.

the graph D, selection of the disjunct labeling a vertex, and backtracking operation whenever a dead end or a bad cycle is encountered. It can, however, be improved to prevent some bad vertices from being processed more than once.

A. Description of the Algorithm

The algorithm performs a depth-first search (DFS). The set Q and transition function δ correspond to the set of vertices V and set of edges E of D, respectively. In addition to the labels $v.\mathcal{X}$, $v.\mathcal{F}$, and $v.\mathcal{E}$, a vertex v has a linked list of outgoing edges. The reference to the first edge of this list is denoted $v.$*first*. Each item in the list contains three elements: an action ($e.$*action*), a reference to the next edge ($e.$*next*), and a reference to the first element of a linked list of its possible tails ($e.$*first*). For a given action, there are as many tails as there are disjuncts in the decomposition of $v.\mathcal{F}$. Thus, each vertex also has a reference to the next vertex ($v.$*next*). The vertices in this linked list differ in their formula and set of unbounded-time eventualities. Finally, a Boolean ($v.unc$) indicates that vertex v has an outgoing edge that leads to an illegal situation on an uncontrollable action. The procedure *Expand* of Fig. 5 is augmented in order to create the two previous lists and set $v.unc$ to false. A stack is maintained throughout the DFS. It contains all the vertices on the current path, except the last vertex that represents the current vertex which is denoted *head*. The detection of a cycle in the graph is done by examining the contents of the stack. Between every two vertices v and v', the stack also contains a reference to the edge from v to v' that the DFS examines when it backtracks from v'.

The algorithm (see Fig. 6) begins with some initial operations (lines 1–4). The main loop (starting at line 5) represents the recursive character of the DFS. The first embedded loop (starting at line 6) scans edges outgoing from the head. The second embedded loop (starting at line 8) scans tails of the current edge e. Three Boolean variables control these loops. The variable *success* controls the second embedded loop. Tails are examined until one of them verifies its disjunct (*success* is then set to true) or the list is exhausted (*tail* = nil). Then, some postprocessing is performed on e. If the examination of tails terminates with *success* equals false and the action associated with the current edge is uncontrollable, then *safe* is set to false. This causes termination of the first embedded loop, which is continued by backtracking actions (line 13). The variable DFS becomes false if a solution is obtained causing the termination of the main loop.

```
    procedure $Examine()$;
 1. if dead end or bad cycle then
 2.    $tail := tail.next$
 3. else $success :=$ true;
 4.      if $tail \in Q$ or good cycle then
 5.         $\phi(head) := \phi(head) \cup \{e.action\}$;
 6.         $\delta(head, e.action) := tail$; $e := e.next$
 7.      else
 8.         push $head$ and $e$ on the stack;
 9.         $head := tail$; $Expand(head)$;
10.         $\phi(head) := \emptyset$; $e := head.first$.
```

Fig. 7. The examination of a vertex.

```
    procedure $Backtrack()$;
 1. $safe :=$ true;
 2. if $\phi(head) = \emptyset$ or $head.unc$ then
 3.    if $head.next \neq$ nil then
 4.       $head := head.next$; $Expand(head)$;
 5.       $\phi(head) := \emptyset$; $e := head.first$
 6.    else if the stack is empty then
 7.          "there are no solutions"; stop
 8.       else
 9.          remove $e$ and $head$ from the stack;
10.          if $e.action \in A_{uc}$ then $safe :=$ false;
11.                                         $head.unc :=$ true
12.                                   else $e := e.next$
13. else
14.    $Q \leftarrow Q \cup \{head\}$;
15.    if the stack is not empty then
16.       $tail := head$;
17.       remove $e$ and $head$ from the stack;
18.       $\phi(head) := \phi(head) \cup \{e\}$; $\delta(head, e) := tail$;
19.       $e := e.next$
20.    else $DFS :=$ false.
```

Fig. 8. The backtracking operation.

Tails are processed by the procedure *Examine* in Fig. 7 according to three cases. First, the tail causes a dead end or closes a bad cycle. In this case, the DFS resumes with the next tail (line 2). Second, the tail represents a good state or closes a good cycle. In this case, the action associated with the edge is enabled from the head, the transition function δ is updated, and the DFS resumes with the next edge (lines 5 and 6). Otherwise, the DFS proceeds with the current tail by redefining *head* as *tail* (lines 8–10).

The procedure *Backtrack* (see Fig. 8) is invoked when all edges of the head have been examined or one of them, having an uncontrollable action, leads to a dead end or a bad cycle regardless of its tail. If the head represents a deadlock (because $\phi(\mathit{head}) = \emptyset$) or cannot prevent an illegal situation (because $\mathit{head}.\mathit{unc} = \mathtt{true}$), then the DFS resumes with the next tail of the current edge if it exists (lines 3–5). If the list of tails has been exhausted, then the algorithm backtracks to the ancestor unless the stack is empty, which means that there are no solutions (lines 6–11). Otherwise, the head is a good vertex and the algorithm backtracks to the ancestor unless the stack is empty (lines 13–18). If the stack is empty, then the main loop terminates because there is a solution (line 19).

B. Proof of Correctness

Theorem 3: Let $G = (X, \mathcal{P}, \lambda, A, \tau, \xi, x_0)$, its associated pair of languages (L, L_ω), control requirements represented by an MTL formula f and a controller $S = (\mathcal{M}, \phi)$ calculated by the procedure *Synthetize*. The corresponding controlled DES $G^S = (L^S, L^S_\omega)$ is such that, if an ω-word $\alpha \in L^S_\omega$ and σ is the trajectory of G^S on α, then $\sigma \models f$.

Proof: If $\alpha \in L^S_\omega$, then $\alpha \in \lim(L^S)$ and $\alpha \in L_\omega$. By construction, $\lim(L^S) \subseteq \lim(L) = L_\omega$. Therefore, it remains to be proved that if $\alpha \in \lim(L^S)$ and σ is the trajectory of G^S on α, then $\sigma \models f$. If $\alpha \in \lim(L^S)$, then $\mathrm{pre}(\{\alpha\}) \subseteq L^S$. In other words, the trajectory of G^S on α is $\sigma' = q_0\delta(q_0, \alpha[0])\delta(q_0, \alpha[0]\alpha[1]) \cdots$. Since Q is finite, the trajectory must be of the form $\sigma'[0]\sigma'[1] \cdots \sigma'[j] \cdots \sigma'[j]$. Note that the procedure *Synthetize* ensures that 1) for all $i \geq 0$, $\sigma'[i].\mathcal{F} \neq false$ and 2) there exists $k \geq j$ such that $\sigma'[k].\mathcal{E} = \emptyset$. The trajectory of G on α is $\sigma = \sigma'[0].\mathcal{X}\sigma'[1].\mathcal{X} \cdots$. From Theorem 2, $\sigma \models f$. ■

C. Maximality of Solutions

In general, there may be many trajectories σ of G such that $\sigma \models f$ that are disabled by the controller S calculated by the procedure *Synthetize*. The procedure keeps only enough of them to obtain a controller. There are three explanations for this. First, the procedure eliminates bad cycles without unwinding them. This is related to the ω-closed property. Second, a trajectory σ of G such that $\sigma \models f$ and a trajectory σ' of G such that $\sigma' \not\models f$ intersect and the trajectory σ' can only be prevented before the intersection point. This is normal and is related to the controllability property. Third, the formula f explicitly or implicitly includes some forms of nondeterminism. This results in several tails for a given edge. They may be all good, but the procedure *Synthetize* retains just one of them. It is interesting to look closer at what are the causes of nondeterminism. In fact there are two.

First, there may be Boolean connectives $\vee$ in the disjunctive normal form of f (e.g., $f = f_1 \vee f_2$). Since the procedure *Synthetize* establishes satisfiability by proving only one of the disjuncts, trajectories satisfying the other disjuncts may be overlooked.

The second source of nondeterminism results from the decomposition of formulas having the *until* connective because they generate disjunctions [see (E7) and (E8)]. However, in this particular case, it seems that if one of the disjuncts is satisfiable, it does not disable trajectories enabled by the other disjunct, and vice versa.

One may therefore conjecture that, if the supremal ω-controllable sublanguage of the language defined by f is ω-closed and nondeterminism is caused solely by the decomposition of formulas and not because there are Boolean connectives $\vee$ in f, the procedure *Synthetize* computes a maximal solution (because of Theorem 2).

In fact, even when disjunctions are involved in f, a maximal solution is computed for some formulas. This depends on the interconnection of temporal connectives. In particular, a maximal solution is computed for any conjunction of formulas in the following form: literals, $\Box_{\sim t} f_1$, and $\Box_{\sim t}(f_1 \rightarrow \Diamond_{\sim t} f_2)$, where f_1 and f_2 do not involve temporal connectives.

D. Computational Complexity

A vertex of the directed graph D consists of a state, a formula, and a set of eventualities. Hence, the maximum number of vertices in D is given by $n \times |F| \times |E|$, where n is the number of states in G (i.e., $n := |X|$), $|F|$ is the number of different possible subformulas of f, and $|E|$ is the number of different possible sets of unbounded-time eventualities obtained from subformulas of f having the form $gU_{\geq 0}h$. By abstracting over the action durations, the number of different subformulas that can be produced for a formula f using equivalences (E4)–(E8) is $2^{|\mathrm{closure}(f)|}$, where $|\mathrm{closure}(f)|$ is the set of subformulas of f. It can be easily checked that $|\mathrm{closure}(f)| \leq 2N$, where N is the number of Boolean and temporal connectives in f. Since $|E| \leq |F|$, the state space is $\mathcal{O}(n2^{4N})$.

In order to take action durations into account, let T be the maximum of the different constants that occur in a time constraint associated with temporal connectives, d the minimum of the different action durations, and C the maximum of 1 and $\lceil T/d \rceil$. It can be shown that there can be at most C different time arguments. Hence, the state space is $\mathcal{O}(n2^{2N(C+1)})$ since $|\mathrm{closure}(f)| \leq 2NC$. The worst case computational complexity is doubly exponential in the size of the formula (but exponential in the size of D) since the algorithm searches for simple cycles in a state space that grows exponentially with the size of the formula (but linearly with the size of D).

This complexity analysis concerns, however, the worst case. In fact, it has been proven that the time complexity for verifying many interesting temporal formulas over concurrent systems is polynomial and sometimes linear [20]. This suggests that the average complexity of our algorithm is much better than the worst case. Actually, many formula combinations are mutually inconsistent, so that they are never generated, or are inconsistent with some states so that their decomposition yields *false*, which causes a pruning of the state space.

E. Comparison with Other Related Algorithms

Several algorithms for synthesizing controllers have been proposed in the literature to achieve or approximate the supremal ω-controllable sublanguage of W $(\sup C^\omega(W))$. Three of them are briefly introduced hereinafter and compared with our synthesis algorithm.

The algorithm proposed by Thistle and Wonham synthesizes the maximally permissive controller when $\sup C^{\omega}(W)$ is ω-closed. Synthesis of controllers is performed for the case where L, L_{ω}, and W are represented by a deterministic $*$-automaton, a deterministic Büchi automaton, and a deterministic Rabin automaton, respectively [43]. Their algorithm includes three steps: 1) computation of controllability prefixes of W; 2) computation of $\sup C^{\omega}(W)$; and 3) computation of the supervisor S. Let us focus on the first step, which is the most significant in terms of computational complexity. If we assume that $L_{\omega} = \lim(L)$ (as is the case in this paper), the first step reduces to the computation of the controllability subset of a deterministic Rabin automaton, which is the set of states from which the automaton can be controlled to the satisfaction of its own acceptance condition. In this particular case, the computational complexity of the first step is $\mathcal{O}(kl(mn)^{3m})$, where k is the number of control patterns (the subsets of A to which one can restrict, at any point in the operation of the automaton, the set of actions that it may execute), l is the size of alphabet A, m is the number of state subset pairs in the Rabin acceptance condition, and n is the number of states [42].

In comparing with our approach, let us assume that W is represented by an MTL formula f instead of a Rabin automaton. When the time domain is dense, the problem has no solution in Thistle and Wonham's framework, since there is no decision procedure for MTL with dense-time domains (i.e., one cannot obtain a Rabin automaton from an MTL formula). In contrast, when the time domain is discrete, one can construct a nondeterministic Büchi automaton for f, whose number of states is exponential in the size of f [46]. By using a determinization procedure defined in [39], one can obtain a deterministic Rabin automaton for f, which has an exponential number of states, but a linear number of state subset pairs in the size of the nondeterministic Büchi automaton. Based on Thistle and Wonham's approach, the computational complexity is therefore triply exponential in the size of f.

The advantage of our approach is that it circumvents the problem with dense-time domains by using a *model-checking* approach as opposed to the *Rabin automaton synthesis approach*. We check the formula directly on all paths that can be generated by the TTG. Since a TTG contains finitely many states and finitely many transition durations, it turns out that there are finitely many states that can be distinguished by an MTL formula, even with dense-time domains. That follows from the above complexity analysis (see Section VII-D). From another perspective, the problem of deciding whether a TTG satisfies a given MTL formula is decidable, while that of deciding if an MTL formula has a model is undecidable for dense-time domains. Another advantage is that it uses forward-chaining state exploration. On average, this yields better computational complexity because it implements a control-directed backtracking technique that can be combined with the use of heuristics to better control the state explosion problem. Finally, one may use it by sacrificing maximally permissiveness, when $\sup C^{\omega}(W)$ is not ω-closed.

Kumar and Garg [24] proposed an algorithm for computing $\sup C^{\omega}(W)$ when $W = \lim(K)$ and K is a regular language recognized by a deterministic automaton. The result is a deterministic automaton from which it is possible to extract controllers that approximate $\sup C^{\omega}(W)$ if this language is not ω-closed with respect to L_{ω}; no specific algorithm for this extraction is given. In particular, the bad cycles are not detected. Their algorithm for computing $\sup C^{\omega}(W)$ is polynomial in the cardinalities of state spaces of deterministic automata modeling the unrestrained and legal behaviors of the process, but it is limited since they implicitly assume that W is recognized by a deterministic Büchi automaton, yet deterministic Büchi automata are strictly weaker than nondeterministic Büchi automata, as explained in [44].

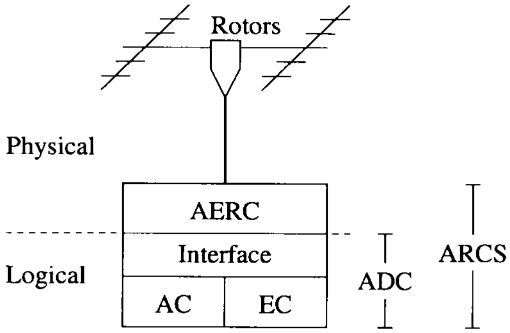


Fig. 9. System architecture.

Antoniotti [5] also proposed a controller synthesis approach based on a model-checking paradigm with *computational tree logic* (CTL) formulas [18]. This model-checking paradigm is, however, significantly different from ours. The idea is still to label states with formulas that they satisfy based on the input formula and CTL semantics. What differs truly from our approach is that the model-checking procedure traverses the state transition structure over which the CTL formula is verified backward, considering innermost subformulas, then iteratively, outermost ones. This requires the whole state transition structure to reside in memory. In contrast, our approach, which goes forward, does not require explicit storage of the entire transition structure. This means that one can exploit standard heuristics to cope with the state explosion problem. Antoniotti also implemented a restricted version of CTL in order to obtain an efficient version of his method. In our case, there is no need to restrict the specification language to enhance efficiency. The actual efficiency depends on the complexity of formulas (e.g., nesting of temporal connectives). In fact, it can be verified that, for simple formulas, our algorithm is polynomial in the size of the formula.

VIII. Application: Antenna Rotor Control System

The application presented in this section is a simplified version of an *antenna rotor control system* (ARCS) used in a laboratory for experimenting with satellite telecommunications [31]. It is responsible for tracking antennas on a moving telecommunications satellite. As illustrated in Fig. 9, it includes two main components:

1) an *azimuth-elevation rotor controller* (AERC), which is a piece of equipment that monitors two rotors that move the antennas;

Fig. 10. Antennas for a satellite tracking system.

2) an *antenna direction controller* (ADC), which determines when to start/stop moving the antennas and the direction of their movement.

In this system, the antennas point in a direction defined by an azimuth and an elevation, both in degrees. There are separate sensing and control processes for azimuth and elevation. Therefore, the ADC comprises three modules: an *azimuth controller* (AC), an *elevation controller* (EC), and an *interface* that maps the physical part of the system onto its logical part.

As explained in [10], in most cases, a system is not readily available as a logical model. It needs to be brought from the physical level to a logical level that is suitable for behavioral language specification. With this application, we deal with voltages and polarities at the physical level and with events or actions at the logical level. On the one hand, the interface extracts information about the position of the antennas by using AERC sensing operations, converts this information into events with respect to an azimuth target and an elevation target, and sends events to the AC or EC. On the other hand, the interface receives actions from the AC or EC, interprets them, and performs the corresponding control operations on the AERC. This architecture allows responsibilities to be separated between several specialized controllers and specific methods to be used depending on the nature of the control. The AERC perceives the rotors as a physical continuous system, while the EC and AC view the antennas as a logical discrete-event system. Finally, it should be noted that the ADC is also responsible for synchronizing the controller operations with the operations of a larger satellite tracking system (the antennas are pictured in Fig. 10).

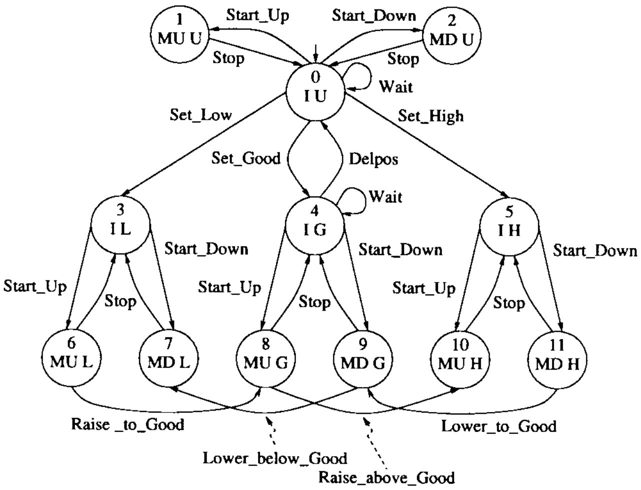


Fig. 11. The TTG: L: Low; H: High; G: Good; U: Unknown; I: Idle; MU: Moving_Up; MD: Moving_Down.

In the sequel, we detail only elevation behavior and control; azimuth behavior and control are analogous. The interface maintains two control variables: *current* and *target*. The former represents the current position of antennas, whereas the latter represents their target position. The antennas are considered on target when the distance between *current* and *target* is less than or equal to a constant d. The domain of *current* and *target* is continuous, from 0 to 180°. The EC reasons about an abstract model of the continuous behavior in which only the relations between the variables *current* and *target* are relevant. Propositions *Low*, *High*, and *Good* refer to the current position of the antennas with respect to the target and they hold when the conditions ($\mathit{target} - \mathit{current} > d$), ($\mathit{current} - \mathit{target} > d$), and $|\mathit{target} - \mathit{current}\ | \leq d$ are, respectively, true. Initially, the relation between *target* and *current* is unknown, which is conveyed by the proposition *Unknown*. The propositions *Idle*, *Moving_Up*, and *Moving_Down* refer to the state of the antennas with respect to their movement. Therefore, the set of propositional symbols $\mathcal{P}$ contains *Low*, *High*, *Good*, *Unknown*, *Idle*, *Moving_Up*, and *Moving_Down*. The TTG that represents the logical system is given in Fig. 11. Every action has a duration of one time unit and the actions *Set_Low*, *Set_High*, *Set_Good*, *Delpos*, and *Wait* are uncontrollable. In order to illustrate the application of the method described in Section IV with short examples, we have omitted some details in this model.

Consider the following formula that specifies the property that when a target is entered in the system (i.e., the relation between the target and current antenna positions is unknown), the antenna must eventually reach the good position

$$f = \Box_{\geq 0}[\neg \mathit{Unknown} \rightarrow \Diamond_{\geq 0}\mathit{Good}].$$

Instead of constructing the graph D step-by-step, let us consider all formulas and sets of unbounded-time eventualities that can be generated from f. Let us suppose that there exists a vertex v such that $v.\mathcal{F} = f$. According to (A1) and equivalences (E6) and (E8)[10] the decomposition of f is

$$[\mathit{Unknown} \vee \mathit{Good} \vee \bigcirc_1(\Diamond_{\geq 0}\mathit{Good})] \wedge \bigcirc_1 f.$$

This formula is equivalent to the following formula in disjunctive normal form:

$$[\mathit{Unknown} \wedge \bigcirc_1 f] \vee [\mathit{Good} \wedge \bigcirc_1 f] \vee [\bigcirc_1(\Diamond_{\geq 0}\mathit{Good}) \wedge \bigcirc_1 f].$$

The first disjunct is relevant only if $\mathit{Unknown} \in \lambda(v.\mathcal{X})$; otherwise successors of v are labeled with *false*. Similarly, the second disjunct is relevant only if $\mathit{Good} \in \lambda(v.\mathcal{X})$. In both cases, successors of v are labeled with the future part of the disjunct, that is, f. Since the present part of the last disjunct is *true*, successors of v are labeled with $g = (\Diamond_{\geq 0}\mathit{Good}) \wedge f$ if this disjunct is selected.

Formula g is new. By using similar arguments, its decomposition is

$$\begin{aligned}
&[\mathit{Unknown} \wedge (\bigcirc_1 f) \wedge \mathit{Good}]\\
&\quad \vee [\mathit{Unknown} \wedge (\bigcirc_1 f) \wedge \bigcirc_1(\Diamond_{\geq 0}\mathit{Good})]\\
&\quad \vee [\mathit{Good} \wedge (\bigcirc_1 f) \wedge \mathit{Good}]\\
&\quad \vee [\mathit{Good} \wedge (\bigcirc_1 f) \wedge \bigcirc_1(\Diamond_{\geq 0}\mathit{Good})]\\
&\quad \vee [\bigcirc_1(\Diamond_{\geq 0}\mathit{Good}) \wedge (\bigcirc_1 f) \wedge \mathit{Good}]\\
&\quad \vee [\bigcirc_1(\Diamond_{\geq 0}\mathit{Good}) \wedge (\bigcirc_1 f) \wedge \bigcirc_1(\Diamond_{\geq 0}\mathit{Good})].
\end{aligned}$$

After trivial simplifications and the elimination of the first disjunct, which is inapplicable [there exists no vertex v such that $\{\mathit{Good}, \mathit{Unknown}\} \subseteq \lambda(v.\mathcal{X})$], we obtain the following disjuncts:

$$\begin{aligned}
&[\mathit{Unknown} \wedge \bigcirc_1(\Diamond_{\geq 0}\mathit{Good}) \wedge (\bigcirc_1 f)]\\
&[\mathit{Good} \wedge (\bigcirc_1 f)]\\
&[\mathit{Good} \wedge \bigcirc_1(\Diamond_{\geq 0}\mathit{Good}) \wedge (\bigcirc_1 f)]\\
&[\bigcirc_1(\Diamond_{\geq 0}\mathit{Good}) \wedge (\bigcirc_1 f)].
\end{aligned}$$

This completes the calculation of the closure of the progression of f. Therefore, successors of v are labeled with the future part of the selected disjunct, that is, either f or g.

In this example, there is only one set of unbounded-time eventualities, that is, $\{\mathit{Good}\}$. In fact, there is no conjunct in f of the form $\Diamond_{\geq 0}h$ or $h_1 U_{\geq 0} h_2$, but g includes the conjunct $\Diamond_{\geq 0}\mathit{Good}$. So, according to (A2), a successor v' of a vertex v is labeled with $\{\mathit{Good}\}$ whenever $v.\mathcal{E} = \emptyset$, $v.\mathcal{F} = g$, and $\mathit{Good} \notin \lambda(v.\mathcal{X})$ (i.e., *Good* is not locally entailed by the edge from v to v'). Furthermore, a successor v' of a vertex v is labeled with the empty set of eventualities whenever $v.\mathcal{E} = \{\mathit{Good}\}$ and $\mathit{Good} \in \lambda(v.\mathcal{X})$ (i.e., *Good* is locally entailed by the edge from v to v'). In the other cases, $v'.\mathcal{E} = v.\mathcal{E}$.

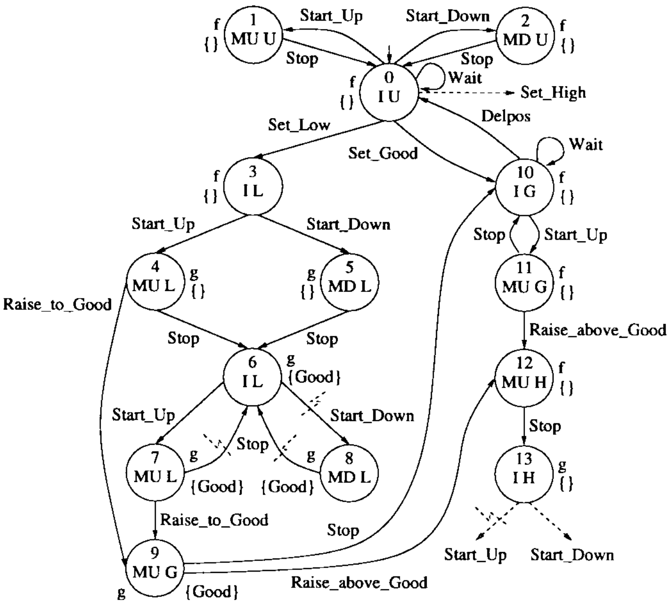


Fig. 12. A part of the graph.

According to the observations above, the graph D can be easily expanded (see Fig. 12 for a part of it). Since f includes only a liveness property, the synthesis algorithm searches only for bad cycles (e.g., 6-8-6) and tries to exclude them from system behavior by prohibiting controllable actions (e.g., *Stop* from state 8). In addition, it removes paths that do not close cycles because they cause a deadlock (e.g., *Start_Down* from state 6). The final solution is a controller with 18 states after minimalization (e.g., states 9 and 11 are equivalent apart from formulas and sets of unbounded-time eventualities).

This first example shows that the solution is not optimal in the sense of maximal permissiveness (the legal language is not ω-closed). In addition, the solution is not efficient. Let us illustrate this point with the sequence of actions

Set_Low Start_Down Stop Start_Up Raise_to_Good $\cdots$.

This sequence is legal, but it is less efficient than the sequence

Set_Low Start_Up Raise_to_Good $\cdots$.

The solution can be refined by the introduction of more constraints. Let us consider the case where the formal specification of control requirements is the conjunction of f and the following seven formulas:

$$\begin{aligned}
f_1 &= \Box_{\geq 0}[\mathit{Unknown}\\
&\qquad \rightarrow \neg \bigcirc_{\geq 0} (\mathit{Moving\_Up} \vee \mathit{Moving\_Down})]\\
f_2 &= \Box_{\geq 0}[(\mathit{Idle} \wedge \mathit{Good}) \rightarrow \bigcirc_{\geq 0}\mathit{Idle}]\\
f_3 &= \Box_{\geq 0}[(\mathit{Idle} \wedge \mathit{High}) \rightarrow \bigcirc_{\geq 0}\mathit{Moving\_Down}]\\
f_4 &= \Box_{\geq 0}[(\mathit{Idle} \wedge \mathit{Low}) \rightarrow \bigcirc_{\geq 0}\mathit{Moving\_Up}]\\
f_5 &= \Box_{\geq 0}[(\mathit{Moving\_Down} \wedge \mathit{High}) \rightarrow \bigcirc_{\geq 0}\mathit{Moving\_Down}]\\
f_6 &= \Box_{\geq 0}[(\mathit{Moving\_Up} \wedge \mathit{Low}) \rightarrow \bigcirc_{\geq 0}\mathit{Moving\_Up}]\\
f_7 &= \Box_{\geq 0}[((\mathit{Moving\_Down} \vee \mathit{Moving\_Up}) \wedge \mathit{Good})\\
&\qquad \rightarrow \bigcirc_{\geq 0}(\mathit{Idle} \wedge \mathit{Good})].
\end{aligned}$$

[10] Equivalence (E8) is reduced to the following equivalence for the *eventually* connective:

$$\Diamond_{\geq t} f \Leftrightarrow \begin{cases} \bigcirc_d \Diamond_{\geq t-d} f, & \text{if } d \leq t\\ \bigcirc_d \Diamond_{\geq 0} f, & \text{if } d > t \text{ and } t \neq 0\\ f \vee \bigcirc_d \Diamond_{\geq 0} f), & \text{if } t = 0. \end{cases}$$

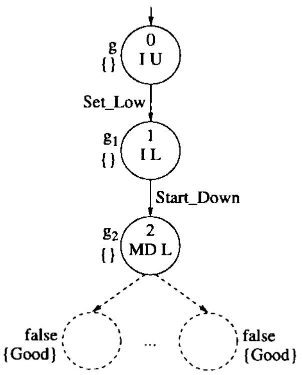


Fig. 13. Progression of g.

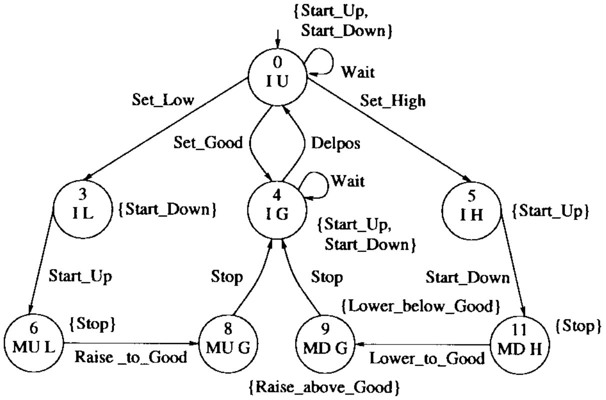


Fig. 14. A controller.

Formulas f_1 to f_7 represent safety properties. Formula f_1 states that whenever the relation between the target position and the current position is unknown, the antennas must not be moved. Formula f_2 specifies that whenever the antennas are idle and their position is good, the antennas must remain idle. Formula f_3 (respectively, f_4) specifies that if the antennas are idle and too high (low), then they must be moved in the down (up) direction. Formula f_5 (respectively, f_6) specifies that if the antennas are moving in the down (up) direction and are too high (low), then they must keep moving in the down (up) direction. Finally, formula f_7 states that whenever the antennas are moving and they are at the good position, they must be stopped.

Fig. 13 shows the progression of $g = f \wedge f_1 \wedge \cdots \wedge f_7$ from the initial vertex. Since the decomposition of f_1 is

$$[\neg \mathit{Unknown} \wedge \bigcirc_1 f_1] \vee [\bigcirc_1(\neg \mathit{Moving\_Down} \wedge \neg \mathit{Moving\_Up}) \wedge \bigcirc_1 f_1]$$

then the progression of g from vertex 0 is

$$g_1 = \neg \mathit{Moving\_Down} \wedge \neg \mathit{Moving\_Up} \wedge g.$$

Similarly, since the decomposition of f_4 is

$$[\neg \mathit{Idle} \wedge \bigcirc_1 f_4] \vee [\neg \mathit{Low} \wedge \bigcirc_1 f_4] \vee [\bigcirc_1 \mathit{Moving\_Up} \wedge \bigcirc_1 f_4]$$

then the progression of g_1 from vertex 1 is

$$g_2 = (\Diamond_{\geq 0} \mathit{Good}) \wedge \mathit{Moving\_Up} \wedge g.$$

The progression of g_2 from vertex 2 gives *false* whatever the successors are, since the present part of g_2 contains *Moving_Up*, yet $\mathit{Moving\_Up} \notin \lambda(2.\mathcal{X})$. The controller must disable action *Start_Down* at vertex number 1 since this vertex causes a deadlock.

Using similar developments for the other formulas, the result is a controller shown in Fig. 14 (the sets of disabled actions are indicated to the right of states).

It should be noted that this controller is much more efficient in terms of number of actions performed to achieve the goal.

IX. Conclusion

In this paper, we have considered a synthesis method based mainly on a temporal logic framework. This work draws from a number of different but related sources on control theory, concurrency theory, and artificial intelligence planning. In these fields, the terms *controller, reactive module,* and *plan* are analogous.

In the artificial intelligence community, the idea of progressing temporal formulas through a trajectory was originally introduced by Bacchus and Kabanza [8]. They first applied this idea to progress search control formulas. Recently, they extended their approach to the problem of synthesizing classical plans when the goals include only safety constraints [9]. The problem of synthesizing reactive plans for safety and liveness constraints has also been addressed by generalizing this technique [22]. The idea of using a similar technique in the context of supervisory control theory has also been investigated earlier by the authors [12]. The present control-theoretic synthesis approach produces controllers that differ from plans; maximal sublanguages, controllable events, and the controllability property are considered. In our planning approach, maximal permissiveness is not a criterion of optimality; uncontrollable events are represented by nondeterminism; and control problems such as controllability, observability, and stability do not arise naturally and, hence, cannot be easily investigated as in the framework of *supervisory control theory*.

Our approach is reminiscent of the decision procedure for linear temporal logic using the *tableau method* [46]. This decision procedure proceeds by constructing a Büchi automaton accepting trajectories satisfying a temporal formula. It involves three steps: 1) construction of a *local automaton* accepting trajectories that satisfy safety properties; 2) construction of an *eventuality automaton* accepting trajectories satisfying liveness properties; and 3) combination of the two automata. Originally developed solely for formulas without time constraints, the technique was later generalized to formulas with time constraints [4]. In technical terms, our method is related to this decision procedure as follows. On the one hand, the progression of formulas, which is based on the property that a temporal formula is decomposable into a present part and a future part, is like the construction of a local automaton. On the other hand, the progression of eventualities, which keeps track of unbounded-time eventualities that must be satisfied, is similar to the construction of an eventuality automaton. Our progression techniques are, however, done with respect to transitions of the process. Intuitively, this amounts to a composition *on the fly* of the local automaton, eventuality automaton, and transition structure of the process.

Another difference between our approach and the construction of Büchi automata from temporal logic specifications is that such automata do not embody the notion of uncontrollable events. Hence, they are not appropriate to the synthesis of controllers for reactive systems. To take into account uncontrollable events, one needs a generalization of Büchi automata to Büchi tree automata. From a tree automaton point of view, our approach is related to approaches for synthesizing reactive modules that satisfy given temporal properties [1], [34]. A reactive module is essentially the same as a controller, except that it is computed by constructively proving that there exists a tree automaton accepting infinite trees that satisfy the desired temporal property. Reactive modules are then simply obtained as representations of satisfactory infinite trees. In fact, a graph generated by *Expand* can also be viewed as an acceptor of infinite trees corresponding to controllers. Rather than trying to obtain a controller from a trace of a proof that shows the validity of a specification, our algorithm searches for a useful controller in the graph. In this way, the state explosion problem can be more easily circumvented by using heuristic techniques.

The work reported in this paper could be extended in several ways to adapt or combine existing synthesis algorithms based on a forward-chaining search (contrary to a fixpoint calculation). For example, adjustments to variable lookahead policy with state information (VLP-S) [13] and variable lookahead policies under partial observation (VLP-PO) [14] algorithms could be realized to express the specification as a temporal formula instead of a state machine as is now the case in supervisory control of DES using limited lookahead [17]. Our algorithm could also be improved to better handle the state explosion problem for instances of control problems consisting of large processes with many similar components. An algorithm that avoids an exhaustive search of the state space by using a symmetry specification already exists [28]. In this approach, processes are described by using colored Petri nets with symmetry specifications; control requirements are expressed as sets of forbidden markings. These two algorithms could be combined to strengthen the development process of controllers based on a synthesis approach. The former could benefit from the symmetry specification, while the latter could benefit from the expressiveness of temporal logic.

As a matter of fact, the application presented in Section VIII is a typical example of a hybrid system in which the controller is a discrete-event system and the process is a continuous system [6]. The discrete-event system model, represented by the TTG in Fig. 11, is an abstraction of the continuous system. In this model, a state of the graph corresponds to more than one state of the continuous system. The control problem addressed in this paper has been solved in the context of *supervisory control theory* for which the theoretical results cannot be applied directly to hybrid systems. Generally, hybrid systems are described by transition structure diagrams in which states represent continuous activities and transitions discrete state changes. They include variables and their behavior is governed in each state by a set of differential equations. Our synthesis approach could be extended to hybrid system specifications by adapting model-checking techniques that are more suitable for such systems [3]. These issues remain to be addressed in future work.

Acknowledgment

The authors would like to thank J. G. Thistle for his discussion on ω-controllability.

References

[1] M. Abadi, L. Lamport, and P. Wolper, "Realizable and unrealizable specifications of reactive systems," in *Proc. 16th Int. Colloquium on Automata, Languages, and Programming,* Stresa, Italy, G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, Eds. Berlin, Germany: Springer-Verlag, 1989, Lecture Notes in Computer Science, vol. 372, pp. 1–17.

[2] B. Alpern and F. B. Schneider, "Defining liveness," *Information Processing Lett.,* vol. 21, no. 4, pp. 181–185, 1985.

[3] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Sci.,* vol. 138, no. 1, pp. 3–34, 1995.

[4] R. Alur and T. Henzinger, "Real-time logics: Complexity and expressiveness," *Inform. and Computation,* vol. 104, no. 1, pp. 35–77, 1993.

[5] M. Antoniotti, "Synthesis and verification of discrete controllers for robotics and manufacturing devices with temporal logic and the control-D system," Ph.D. dissertation, Dept. Computer Science, New York Univ., 1995.

[6] P. J. Antsaklis, J. A. Stiver, and M. Lemmon, "Hybrid system modeling and autonomous control systems," in *Hybrid Systems,* Lecture Notes in Computer Science, vol. 736, R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds. Berlin, Germany: Springer-Verlag, 1993, pp. 366–392.

[7] E. Asarin, O. Maler, and A. Pnueli, "Symbolic controller synthesis for discrete and timed systems," in *Hybrid Systems II,* Lecture Notes in Computer Science, vol. 999, P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, Eds. Berlin, Germany: Springer-Verlag, 1995, pp. 1–20.

[8] F. Bacchus and F. Kabanza, "Using temporal logic to control search in a forward chaining planner," in *Proc. 3rd European Workshop on Planning,* Assisi, Italy, Sept. 1995, pp. 157–169.

[9] F. Bacchus and F. Kabanza, "Planning for temporally extended goals," in *Proc. 13th Nat. Conf. on Artificial Intelligence,* Portland, OR, Aug. 1996, pp. 1215–1222.

[10] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE Trans. Automat. Contr.,* vol. 38, pp. 1040–1059, July 1993.

[11] M. Barbeau, F. Kabanza, and R. St.-Denis, "A comparison of two synthesis methods for timed discrete-event systems," in *Proc. 8th Canadian Conf. Electrical and Computer Engineering,* Montreal, Canada, Sept. 1995, pp. 809–812.

[12] ——, "Supervisory control synthesis from metric temporal logic specifications," in *Proc. 33th Allerton Conf.,* Univ. Illinois, Urbana, Oct. 1995, pp. 96–105.

[13] N. Ben Hadj-Alouane, S. Lafortune, and F. Lin, "Variable lookahead supervisory control with state information," *IEEE Trans. Automat. Contr.,* vol. 39, pp. 2398–2410, Dec. 1994.

[14] ——, "Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation," *J. Discrete Event Dynamic Systems: Theory and Appl.,* vol. 6, no. 4, pp. 379–427, 1996.

[15] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Trans. Automat. Contr.,* vol. 39, pp. 329–342, Feb. 1994.

[16] C. G. Cassandras, S. Lafortune, and G. J. Olsder, "Introduction to the modeling, control and optimization of discrete event systems," in *Trends in Control—A European Perspective,* A. Isidori, Ed. London, U.K.: Springer-Verlag, 1995, pp. 217–291.

[17] S. L. Chung, S. Lafortune, and F. Lin, "Limited lookahead policies in supervisory control of discrete-event systems," *IEEE Trans. Automat. Contr.,* vol. 37, pp. 1921–1935, Dec. 1992.

[18] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. Program. Lang. Syst.,* vol. 8, no. 2, 1986, pp. 244–263.

[19] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics,* J. van Leeuwen, Ed. Cambridge, MA: MIT Press, 1990, pp. 995–1072.

[20] E. A. Emerson, T. Sadler, and J. Srinivasan, "Efficient temporal reasoning," in *Proc. 16th Annual ACM Symp. Principles of Programming Languages,* Austin, TX, Jan. 1989, pp. 166–178.

[21] A. Fusaoka, H. Seki, and K. Takahashi, "A description and reasoning of plant controllers in temporal logic," in *Proc. 8th Int. Joint Conf. Artificial Intelligence,* Karlsruhe, Germany, Aug. 1983, pp. 405–408.

[22] F. Kabanza, M. Barbeau, and R. St.-Denis, "Planning control rules for reactive agents," *Artificial Intelligence,* vol. 95, no. 1, pp. 67–113, 1997.

[23] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Syst.,* vol. 2, no. 4, pp. 255–299, 1990.

[24] R. Kumar and V. K. Garg, *Modeling and Control of Logical Discrete Event Systems*. Boston, MA: Kluwer, 1995.

[25] L. Lamport, "Proving the correctness of multiprocess programs," *IEEE Trans. Soft. Eng.,* vol. SE-3, no. 2, pp. 125–143, 1977.

[26] F. Lin, "Analysis and synthesis of discrete event systems using temporal logic," *Control Theory and Advanced Technology,* vol. 9, no. 1, pp. 341–350, 1993.

[27] J.-Y. Lin and D. Ionescu, "Optimization of controller design for discrete event systems in a temporal logic framework," in *Proc. American Control Conf.,* Chicago, IL, June 1992, pp. 2819–2823.

[28] M. Makungu, R. St.-Denis, and M. Barbeau, "A colored Petri net-based approach to the design of controllers," in *Proc. 35th IEEE Conf. Decision and Contr.*, Kobe, Japan, Dec. 1996, pp. 4425–4432.

[29] O. Maler, A. Pnueli, and J. Sifakis, "On the synthesis of discrete controllers for timed systems," in *Proc. 12th Annual Symp. on Theoretical Aspects of Computer Science,* Munich, Germany, Mar. 1995, E. W. Mayr and C. Puech, Eds. Berlin, Germany: Springer-Verlag, 1995, Lecture Notes in Computer Science, vol. 900, pp. 229–242.

[30] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems—Specification*. New York: Springer-Verlag, 1992.

[31] M. Normandeau, S. Bernier, J.-M. Desbiens, and M. Barbeau, "WATOO: An Internet access software to a satellite tracking station," in *Proc. AMSAT-NA Space Symp. and Annual Meeting,* Toronto, Canada, Oct. 1997, pp. 24–28.

[32] J. S. Ostroff, "Formal methods for the specification and design of real-time safety critical systems," *The J. Syst. Software,* vol. 18, no. 1, pp. 33–60, 1992.

[33] J. S. Ostroff and W. M. Wonham, "A framework for real-time discrete event control," *IEEE Trans. Automat. Contr.*, vol. 35, pp. 386–397, Apr. 1990.

[34] A. Pnueli and R. Rosner, "On the synthesis of an asynchronous reactive module," in *Proc. 16th Int. Colloquium on Automata, Languages and Programming,* Stresa, Italy, July 1989, G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, Eds. Berlin, Germany: Springer-Verlag, 1989, Lecture Notes in Computer Science, vol. 372, pp. 652–671.

[35] P. J. G. Ramadge, "Some tractable supervisory control problems for discrete-event systems modeled by Büchi automata," *IEEE Trans. Automat. Contr.,* vol. 34, pp. 10–19, Jan. 1989.

[36] P. J. G. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete-event processes," *SIAM J. Contr. and Optimization,* vol. 25, no. 1, pp. 206–230, 1987.

[37] ——, "The control of discrete event systems," *Proc. IEEE,* vol. 77, pp. 81–98, 1989.

[38] A. P. Ravn, H. Rischel, and K. M. Hansen, "Specifying and verifying requirement of real-time systems," *IEEE Trans. Soft. Eng.,* vol. 19, no. 1, pp. 41–55, 1993.

[39] S. Safra, "On the complexity of ω-automata," in *Proc. 29th IEEE Symp. on Foundations of Computer Science,* White Plains, NY, Oct. 1988, pp. 319–327.

[40] J. G. Thistle, "Supervisory control of discrete event systems," *Math. Comput. Modeling,* vol. 23, nos. 11/12, pp. 25–53, 1996.

[41] J. G. Thistle and W. M. Wonham, "Control problems in a temporal logic framework," *Int. J. Contr.,* vol. 44, no. 4, pp. 943–976, 1986.

[42] ——, "Control of infinite behavior of finite automata," *SIAM J. Contr. Optim.,* vol. 32, no. 4, pp. 1075–1097, 1994.

[43] ——, "Supervision of infinite behavior of discrete-event systems," *SIAM J. Contr. Optim.*, vol. 32, no. 4, pp. 1098–1113, 1994.

[44] W. Thomas, "Automata on infinite objects," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics,* J. van Leeuwen, Ed. Cambridge, MA: MIT Press, 1990, pp. 135–191.

[45] P. Wolper, "The tableau method for temporal logic: An overview," *Logique et Analyse,* vol. 28, nos. 110/111, pp. 119–136, 1985.

[46] ——, "On the relation of programs and computations to models of temporal logic," in *Proc. Colloquium on Temporal Logic in Specification,* Altrincham, U.K., Apr. 1987, B. Banieqbal, H. Barringer, and A. Pnueli, Eds. Berlin, Germany: Springer-Verlag, 1989, Lecture Notes in Computer Science, vol. 398, pp. 75–123.

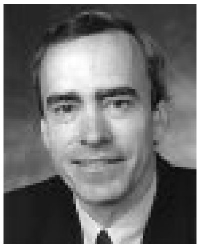
[47] W. M. Wonham and P. J. G. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM J. Contr. Optim.,* vol. 25, no. 3, pp. 637–659, 1987.

Michel Barbeau (M'97) received the B.S. degree from the Université de Sherbooke, Canada, in 1985 and the M.S. and Ph.D. degrees from the Université de Montréal, Canada, in 1987 and 1991, respectively, all in computer science.

From 1987 to 1988 he was a Research Assistant at INRS-Télécommunications, Verdun, Canada. Since 1991 he has been a Professor in the Department of Mathematics and Computer Science at the Université de Sherbrooke. His research interests include software of telecommunications systems.

Froduald Kabanza (M'96) received the Ph.D. degree in 1992 and the B.Sc. degree in 1988, both in computer science, from the Université de Liège, Belgium.

He is an Associate Professor at the Université de Sherbrooke, Canada, and the author of several journal and conference papers on planning and control in the fields of artificial intelligence and control theory.



Richard St.-Denis (S'86–M'88) received the B.Sc. and M.Sc. degrees in computer science from the Université de Montréal in 1975 and 1977, respectively, and the Ph.D. degree in Applied Sciences from École Polytechnique de Montréal in 1992.

He is currently a Professor in the Department of Mathematics and Computer Science at the Université de Sherbrooke, Canada, where his research interests include control theory, software engineering, and artificial intelligence. He has published more then 35 journal and conference papers in these areas.