Open Systems in Reactive Environments: Control and Synthesis

Orna Kupferman[1], P. Madhusudan[2], P.S. Thiagarajan[3], and Moshe Y. Vardi[4]

[1] Hebrew University orna@cs.huji.ac.il
[2] The Institute of Mathematical Sciences madhu@imsc.ernet.in
[3] Chennai Mathematical Institute pst@smi.ernet.in
[4] Rice University vardi@cs.rice.edu

Abstract. We study the problems of synthesizing open systems as well as controllers for open systems. We deal with specifications given as formulas of the branching temporal logic CTL* and its sub-logic CTL. A key aspect of our work is that we deal with *reactive* environments. These are environments that can disable some of their responses along the interaction with the system.

We show tight complexity bounds for the solutions to these problems with the upper bounds being established using automata-theoretic techniques. When the answer to a problem is positive, it is possible to synthesize the desired finite-state system or controller. For CTL specifications, the problem of synthesizing an open system as well as that of synthesizing a controller are 2EXPTIME-complete in the size of the specification. For CTL* specifications, both problems are 3EXPTIME-complete. Thus, in a sense, reactive environments constitute a provably harder setting for the synthesis of open systems and the synthesis of controllers for them.

1 Introduction

A *closed* system is a system whose behavior is completely determined by its state. An *open* system is a system that interacts with its environment and whose behavior crucially depends on this interaction [HP85]. In an open system, we are required to distinguish between output signals (generated by the system) over which we have control and input signals (generated by the environment) over which we have no control. Given a specification of an open system, say as a temporal logic formula, satisfiability of the formula only guarantees that we can synthesize a system for *some* environment, whereas we need to synthesize a system that meets the specification for *all* environments.

To make this intuition more precise, suppose we are given finite sets I and O of input and output signals. A *program* can be viewed as a *strategy* $f : (2^I)^* \to 2^O$ that maps finite sequences of input signal sets into an output signal set. When f interacts with an environment that generates infinite input sequences what results is an infinite computation over $2^{I \cup O}$. Though f is deterministic, it produces a computation tree. The branches of the tree correspond to external non-determinism caused by the different possible inputs. One can now specify an open system by a linear or branching temporal logic formula (over $I \cup O$). Unlike linear temporal logics, in branching temporal logics one can specify possibility requirements such as "every input sequence can be extended so that the output signal v eventually becomes true" ([DTV99]). This is achieved via existential and universal quantification provided in branching temporal logics [Lam80,Eme90]. In this paper, we concentrate on branching temporal logics.

The *realizability problem* for a branching-temporal logic is to determine, given a branching-time specification φ, whether there exists a program $f : (2^I)^* \to 2^O$ whose computation tree satisfies φ [ALW89,PR89]. Realizing φ boils down to synthesizing such a function f. An important aspect of the computation tree associated with f is that it has a fixed branching degree $|2^I|$. It reflects the assumption that at each stage, all possible input signals are provided by the environment. Such environments are referred to as *maximal environments*. Intuitively, these are static environments in terms of the branching possibilities they contribute to the associated computation trees. Equivalently, as we have noted already, each program has just one computation tree capturing its behavior in the presence of a maximal environment. In a more general setting, however, we have to consider environments that are, in turn, open systems. We term such environments *reactive*. They might offer different subsets of 2^I as input possibilities at different stages in the computation.

As an illustration, consider $I = \{r_1, r_2, ..., r_n\}$ and $O = \{t_1, t_2, ..., t_n\}$ where I represents n different types of resources and O represents n different types of tasks with the understanding that, at each stage, the system needs to receive r_i from the environment in order to execute t_i. In the case of the maximal environment, the specification "it is always possible to reach a stage where t_i is executed" ($AGEF(t_i)$ in CTL parlance) is realizable. This is so because at each stage in the computation, the maximal environment presents all possible combinations of the resources. In the case of the reactive environment, the above specification is *not* realizable. This is so because there could be an environment driven by an open system that can produce only a a proper subset of the set I of resources. In the resulting computation tree, each path eventually reaches a node in which the environment stop offering r_i. From then on, t_i cannot be executed.

So, a reactive environment associates a *set* of computation trees with a program. Consequently, in the presence of reactive environments, the realizability problem must seek a program *all* of whose computation trees satisfy the specification.

A closely related problem concerns the controllability of open systems. Here we are given an open system, often called a *plant* in this context, typically consisting of system states and environment states. We can control the moves made from the system states (i.e. disable some of the possible moves), but not the moves made from the environment states. Given a branching-time specification φ, the *control problem* is to come up with a strategy for controlling the moves made from the system sates so that the resulting computation tree satisfies φ. Here again, assuming a reactive environment requires the controller to function correctly no matter how the environment disables some of its moves; thus correctness should be checked with respect to a whole set of computation trees.

We study control problems for both CTL* and CTL specifications. We show that the realizability problem can be reduced to the control problem. We prove that both these problems are 3EXPTIME-complete and 2EXPTIME-complete for CTL* and CTL, respectively. As established in [KV99a,KV99b], these problems are EXPTIME-complete and 2EXPTIME-complete for maximal environments, respectively. In this sense, reactive environments make it more difficult to realize open systems and synthesize controllers for them.

In the literature, a variety of realizability and control problems have been studied for open systems. These studies have been mainly in linear-time settings where the nature of the environment (maximal or reactive) does not play a role (this is since a maximal environment is the most challenging one for the system). In branching-time settings, the emphasis has been mainly on the realizability problem (often referred to as the synthesis problem) in the presence of maximal environments. For the linear time case, the literature goes back to a closely related realizability problem due to Church [Chu63], which was solved in [BL69] but more elegantly dealt with later using tree automata [Rab70,PR89]. In branching-time settings, the realizability problem for CTL* and CTL has been solved for maximal environments as cited above and more recently also for the μ-*calculus* [KV00].

As for controller synthesis, as mentioned above, most of the settings considered in the literature are linear time ones and often involve dealing with incomplete information [KG95,KS95,PR89,Var95]. As for branching-time settings, control problems for settings with maximal environment is studied in [Ant95]. Also, for maximal environments, the control problem can be transformed (by flipping the role of the system and the environment) into *module-checking* problems solved in [KV96,KV97a]. Yet another work, but in a different framework is [MT98] where the specification is also modeled by a transition system and the correctness criterion is that there should be a behavior preserving simulation from the plant to the specification. This has been later extended to the case of bisimulation in [MT00].

2 The Problem Setting

We assume familiarity with the branching temporal logic CTL* and its sub-logic CTL. The syntax and semantics of these logics are detailed in Appendix A. Here we just fix the notation for (Kripke) structures, which provide the semantics.

A *(Kripke) structure* is a tuple $S = \langle AP, W, R, w_0, L\rangle$, where AP is the set of atomic propositions, W is a set of states, $R \subseteq W \times W$ is a transition relation that must be total (i.e., for every $w \in W$ there exists $w' \in W$ such that $R(w, w')$), w_0 is an initial state, and $L : W \to 2^{AP}$ maps each state to a set of atomic propositions true in this state. For w and w' with $R(w, w')$, we say that w' is a successor of w. A *path* of S is an infinite sequence $\pi = w^0, w^1, \ldots$ of states such that for every $i \geq 0$, we have $R(w^i, w^{i+1})$. The suffix $w^i, w^{i+1}, \ldots$ of π is denoted by π^i. We use $w \models \varphi$ to indicate that a state formula φ holds at state w, and we use $\pi \models \varphi$ to indicate that a path formula φ holds at path π (assuming a structure S). We say that S is a model of φ to mean that $w_0 \models \varphi$.

For studying controller synthesis, we model a plant as $P = \langle AP, W_s, W_e, R, w_0, L\rangle$, where AP, R, w_0, and L are as in a structure with $W = W_s \cup W_e$. Here W_s is a set of *system states* and W_e is a set of *environment states*. Throughout what follows we are concerned only with *finite* plants; AP and W are both finite sets. We also assume that $W_s \cap W_e = \emptyset$. The *size* of the plant is $|P| = |W| + |R|$.

Given a finite set Υ, an Υ-tree is a set $T \subseteq \Upsilon^*$ such that, if $v.c \in T$ with $v \in \Upsilon^*$ and $c \in \Upsilon$, then $v \in T$. When Υ is clear from the context we call T a tree. The elements of T are called nodes and the empty word ε is the root of T. For every $v \in T$, the set $succ_T(v) = \{v.c \mid c \in \Upsilon \text{ and } v.c \in T\}$ is the set of children (successors) of v. Where T is clear from the context, we drop the subscript T and write $succ(v)$. In what follows, every tree T we consider is assumed to satisfy $succ(v) \neq \emptyset$ for every v in T. We associate a direction $dir(v) \in \Upsilon$ with each node $v \in T$. A designated element $c_0 \in \Upsilon$ is the direction of ε. For each non-root node $v.c$ with $c \in \Upsilon$, we set $dir(v.c) = c$. A path π of a tree is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for each $v \in \pi$, we have $|\pi \cap succ(v)| = 1$. Finally, given a set Σ, a Σ-labeled Υ-tree is a pair (T, V) where T is a Υ-tree and $V : T \to \Sigma$ is a labeling function. Of special interest to us are 2^{AP}-labeled trees. We call such trees *computation trees* and we sometimes interpret CTL* formulas with respect to them. Formally, a computation tree (T, V) satisfies a CTL* formula φ if φ is satisfied in the infinite-state structure $\langle AP, T, R_T, \varepsilon, V\rangle$, where $R_T(v, v.c)$ iff $v.c \in succ_T(v)$.

Let $P = \langle AP, W_s, W_e, R, w_o, L\rangle$ be a plant. Then P can be unwound into a W-tree T_P in the obvious manner; thus $\varepsilon \in T_P$, and we set $dir(\varepsilon) = w_0$, and for all $v \in T_P$, we have that $v.c \in T_P$ iff $R(dir(v), c)$. The tree T_P induces the 2^{AP}-labelled tree (T_P, V_P) where for each $v \in T_P$, we have $V_P(v) = L(dir(v))$. A *restriction* (T, V) of (T_P, V_P) is a tree $T \subseteq T_P$ such that V is V_P restricted to T. We denote by $T_P^s = \{v \mid v \in T_P \text{ and } dir(v) \in W_s\}$ the set of nodes of T_P

that correspond to system states, and by $T_P^e = T_P \setminus T_P^s$ the set of states that correspond to environment states.

Consider a plant P and a restriction $\langle T, V \rangle$ of $\langle T_P, V_P \rangle$. We can think of $\langle T, V \rangle$ as a computation tree generated by an interaction of the system with its environment: each position in this interaction corresponds to a node $v \in T$. When $v \in T_P^s$, the system chooses a nonempty subset of $succ_{T_P}(v)$. This subset corresponds to successors of $dir(v)$ to which the system enables the transition from $dir(v)$. Similarly, when $v \in T_P^e$, the environment chooses a nonempty subset of $succ_{T_P}(v)$, corresponding to successors of $dir(v)$ the transition to which is enabled. Note we can have v and v' with $dir(v) = dir(v')$ and still $succ_T(v) \neq succ_T(v')$. Indeed, the decisions made by the system and the environment depend not only in the current state of the interaction (that is, $dir(v)$), but also in the entire interaction between the system and the environment so far (that is, v). Intuitively, given P and a CTL* formula φ, the control problem is to come up with a strategy for the system so that no matter how the environment responds, the resulting restriction of $\langle T_P, V_P \rangle$ satisfies φ.

We now make this intuition formal. A *strategy* for the system is a function g that assigns to each $v \in T_P^s$, a non-empty subset of $succ_{T_P}(v)$. A *g-respecting execution* of P is a restriction $\langle T, V \rangle$ of $\langle T_P, V_P \rangle$ such that for every $v \in T \cap T_P^s$, we have $succ_T(v) = g(v)$. Note that a strategy g determines only the successors of nodes in T_P that correspond to system states. Thus, P may have many g-respecting executions, each corresponding to a different strategy of the environment. Given a CTL* formula φ, the strategy g is *winning* for φ if all the g-respecting executions satisfy φ. The *control problem* is then to decide, given a plant P and a CTL* specification φ, whether there the system has a strategy winning for φ, denoted *controllable*(P, φ).

The synthesis problem for programs with reactive environments can be tackled along very similar lines: we consider a program f interacting with its environment via two finite sets I and O of input and output signals respectively . We can view f as a strategy $f : (2^I)^* \rightarrow 2^O$ that maps finite sequences of input signal sets into an output signal set. The interaction starts by the program generating the output $f(\varepsilon)$. The environment replies with some $i_1 \subseteq I$. In general, $f(i_1, i_2, \ldots, i_j)$, is the response of f for the input sequence $i, i_2, \ldots i_j$. This (infinite) interaction can be represented by a computation tree. The branches of the tree correspond to external non-determinism caused by different input signal sets chosen by the environment. Thus f can be viewed as the full $2^{I \cup O}$- labeled 2^I-tree $\langle T_f, V_f \rangle$ with $T_f = (2^I)^*$ and $V_f(v) = dir(v) \cup f(v)$ for each $v \in T_f$. Given a CTL* formula φ, the *program-synthesis problem* is to find a a strategy f so that φ is satisfied in f no matter how the environment disables (in a non-blocking manner) its possible responses at different stages. Formally, let $f : (2^I)^* \rightarrow 2^O$ be a strategy and let $\langle T, V \rangle$ be a $2^{I \cup O}$-labeled 2^I-tree. We say that $\langle T, V \rangle$ is an *f-respecting execution* iff $V(v) = f(v) \cup dir(v)$ for each $v \in T$. (In other words, $\langle T, V \rangle$ is a restriction of the computation tree $\langle T_f, V_f \rangle$). We say that f *realizes* φ if every f-respecting execution satisfies φ. Also, φ is *realizable* iff there a program f that realizes φ.

Using a universal plant, which embodies all the possible assignments to I and O, the program-synthesis problem can be reduced to the supervisory control problem. Formally, we have the following (see Appendix B for the proof):

Lemma 1. *Let φ be a CTL* (CTL) formula over $AP = I \cup O$. We can effectively construct a finite plant P and a CTL* (resp. CTL) formula φ' such that $|P| = O((2^{AP})^2)$, $|\varphi'| = O(|\varphi| + 2^{AP})$, and φ is realizable iff controllable(P, φ).*

3 Upper bounds

Tree automata

Since our decision procedure uses tree automata, we first introduce Büchi and parity tree automata (see [Tho90] for more details). Tree automata run on Σ-labeled Υ-trees. We assume that the set Υ of directions is ordered. For an Υ-tree T and a node $v \in T$, we use $o\_succ(v)$ to denote an ordered list of $succ(v)$. Let $|\Upsilon| = k$, and let $[k] = \{1, \ldots, k\}$. A *nondeterministic tree automaton* over Σ-labeled Υ-trees is $A = \langle \Sigma, Q, \delta, q_0, \mathcal{F} \rangle$ where Σ is a finite alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \times [k] \to 2^{Q^*}$ is a transition function that maps a state, a letter, and a branching degree $i \in [k]$ to a set of i-tuples of states, $q_0 \in Q$ is the initial state, and $\mathcal{F}$ is an acceptance condition that depends on the kind of automata we consider. For *Büchi automata*, $\mathcal{F}$ is a subset of Q and for *parity automata*, $\mathcal{F}$ is a function $\mathcal{F} : Q \to \{0, \ldots, h\}$ for some $h \in \mathbb{N}$ (the set {0,…, h} are called the set of *colors*).

Let (T, V) be a Σ-labeled tree. A *run* of A over (T, V) is a Q-labeled tree (T, ρ) such that $\rho(\varepsilon) = q_{in}$, and for all $v \in T$ with $o\_succ(v) = \langle x_1, \ldots, x_l \rangle$, we have $\langle \rho(x_1), \ldots, \rho(x_l) \rangle \in \delta(q, V(x), l)$. For a path π of T, let

$$Inf_\rho(\pi) = \{q \in Q \mid \text{there are infinitely many } x \in \pi \text{ such that } \rho(x) = q\}.$$

If A is a Büchi automaton, then ρ is a *accepting* if for every path π of T, $Inf_\rho(\pi) \cap \mathcal{F} \neq \emptyset$. For parity automata, ρ is *accepting* if for every path π of T, $min(\mathcal{F}(Inf_\rho(\pi)))$ is an even number. A Σ-labeled tree (T, V) is accepted by A iff there is a accepting run of A over (T, V). The language accepted by A is the set of all Σ-labeled trees accepted by A. We say that A is *universal* iff it accepts all Σ-labeled trees.

There are well-known connections relating branching temporal logics and tree automata. For our purposes, it suffices to know the following.

Theorem 2.

(1) [VW86] *Given a CTL formula φ over AP and a set Υ, we can construct a nondeterministic Büchi tree automaton $A_{\Upsilon,\varphi}$ with $2^{O(|\varphi|)}$ states that accepts exactly the set of 2^{AP}-labeled Υ-trees that satisfy φ.*

(2) [EJ88,Saf88,Tho97] *Given a CTL* formula φ over AP and a set Υ, one can construct a nondeterministic parity tree automaton $A_{\Upsilon,\varphi}$ with $2^{2^{O(|\varphi|)}}$ states and $2^{O(|\varphi|)}$ colors that accepts exactly the set of 2^{AP}-labeled Υ-trees that satisfy φ.*

The decision procedure

Recall that in the control problem we are given a plant $P = \langle AP, W_s, W_e, R, w_0, L \rangle$ and a CTL (or CTL*) formula φ over AP, and we have to decide whether there is a strategy g for the system so that all the g-respecting executions of P satisfy φ.

Recall that a strategy g for the system assigns to each $v \in T_P^s$ a nonempty subset of $succ(v)$. We can associate with g a $\{\perp, \top, d\}$-labeled W-tree (T_P, V_g), where for every $v \in T_P$, the following hold:

- If $v \in T_P^s$, then the children of v that are members of $g(v)$ are labeled by $\top$, and the children of v that are not members of $g(v)$ are labeled by $\perp$.
- If $v \in T_P^e$, then all the children of v are labeled by d.

Intuitively, nodes $v.c$ are labeled by $\top$ if g enables the transitions from $dir(v)$ to c (given that the execution so far has traversed v), they are labeled by $\perp$ if g disables the transition from $dir(v)$ to c, and they are labeled by d if $dir(v)$ is an environment state, where the system has no control about the transition from $dir(v)$ to c being enabled. We call the tree (T_P, V_g) the *strategy tree* of g. Note that not every $\{\perp, \top, d\}$-labeled W-tree (T_P, V) is a strategy tree. Indeed, in order to be a strategy tree, V should label all the successors of nodes corresponding to environment states by d, and it should label all the successors of nodes corresponding to system states by either $\top$ or $\perp$, with at least one successor being labeled by $\top$. Formally, we have the following.

Theorem 3. *Given a plant P with states space W, there is a nondeterministic Büchi automaton A_{stra} over $\{\perp, \top, d\}$-labeled W-trees, such that A_{stra} has W states and it accepts exactly all the strategy trees of P.*

Our algorithm proceeds as follows. Given a formula φ, we contsruct a tree automaton A such that A accepts a strategy tree (T_P, V_g) iff there is a g-respecting executions of P that does not satisfy φ. Then, *controllable*(P, φ) iff the automaton A is not universal with respect to strategy trees (that is, the language of A_{stra} is not contained in that of A). Indeed, a strategy tree that is not accepted by A is induced by a strategy g all of whose g-respecting executions satisfy φ.

Theorem 4. *Given a plant P with state space W and a branching-time formula φ, we can construct a nondeterministic tree automaton A over $\{\perp, \top, d\}$-labeled W-trees such that the following hold:*

1. *A accepts a strategy tree (T_P, V_g) iff there is a g-respecting execution of P that does not satisfy φ.*
2. *If φ is a CTL formula, then A is a Büchi automaton with $|W| \cdot 2^{O(\varphi)}$ states.*
3. *If φ is a CTL* formula, then A is a parity automaton with $|W| \cdot 2^{2^{O(|\varphi|)}}$ states and $2^{O(|\varphi|)}$ colors*

Proof. Let $P = \langle AP, W_s, W_e, R, w_0, L \rangle$, and let $A_{W,\neg\varphi} = \langle 2^{AP}, Q, \delta, Q_0, \mathcal{F} \rangle$ be the automaton that accepts exactly all 2^{AP}-labeled W-trees that satisfy $\neg\varphi$, as described in Theorem 2. We define $A = \langle \{\perp, \top, d\}, Q', \delta, Q'_0, \mathcal{F}' \rangle$ as follows.

– $Q' = (W \times Q \times \{\top, \bot\}) \cup \{q_{acc}\}$. The state q_{acc} is an accepting sink. Consider a state $(w, q, m) \in W \times Q \times \{\top, \bot\}$. The last component m is the *mode* of the state. When $m = \top$, it means that the transition to the current node is enabled (by either the system or the environment). When $m = \bot$, it means that the transition to the current node is disabled.
When A is at a state $(w, q, \top)$ as it reads a node v, it means that $dir(v) = w$, and that v to participate in the g-respecting execution. Hence, A can read $\top$ or d, but not $\bot$. If v is indeed labeled by $\top$ or d, the automaton A guesses a subset of successors of w of some size $l \geq 1$. It then moves to states corresponding to the successors of w and q, with an appropriate update of the mode ($\top$ for the successors in the guessed subset $\bot$ for the rest).
When A is in a state $(w, q, \bot)$ and it reads a node v, it means that $dir(v) = w$ and that v does not take part in a g-respecting execution. Then, A expects to read $\bot$ or d, in which case it goes to the accepting sink.
– $Q'_0 = \{w_0\} \times Q_0 \times \{\top\}$.
– The transition function δ' is defined, for all $w \in W$ $q \in Q$, and $l = |succ_{T_P}(w)|$ as follows.
  - $\delta((w, q, \top), \bot, l) = \delta((w, q, \bot), \top, l) = \emptyset$
  - If $x \in \{\bot, d\}$, then $\delta'((w, q, \bot), x, l) = \{\langle q_{acc}, \ldots, q_{acc}\rangle\}$,
  - If $x \in \{\top, d\}$, then $\delta((w, q, \top), x, l)$ is defined as follows. Let $o\_succ(w) = \langle w_1, \ldots, w_l\rangle$ and let $Y = \{w_{y_1}, \ldots, w_{y_n}\}$ be a nonempty subset (of size n) of $succ(w)$. Then, $\delta'((w, q, \top), x, l)$ contains all tuples $\langle (w_1, s_1, m_1), \ldots, (w_l, s_l, m_l)\rangle$ such that there is $\langle q_1, \ldots, q_n\rangle \in \delta(q, L(w), n)$ and for all $1 \leq i \leq l$, the following hold:
    * If $w_i \in Y$, namely $w_i = w_{y_j}$ for some $1 \leq j \leq n$, then $s_i = q_j$ and $m_i = \top$.
    * If $w_i \notin Y$, then $w_i = w$ (in fact, w_i can be an arbitrary state) and $m_i = \bot$.

    Intuitively, δ' propagates the requirements imposed by $\delta(q, L(w), n)$ among the successors of w to which the transition from w is enabled.
    Note that δ' is independent of w being a system or an environment state. The type of w is taken into consideration only in the definition of A_{stra}.
– The final states are inherited from the formula automaton. Thus, if φ is in CTL, then $\mathcal{F}' = (W \times \mathcal{F} \times \{\top, \bot\}) \cup \{q_{acc}\}$. If φ is in CTL*, let $\mathcal{F} : Q \to \{0, \ldots, h\}$. Then, $\mathcal{F}' : Q' \to \{0, \ldots, h\}$ is such that $\mathcal{F}(q_{acc}) = 0$ and for all $w \in W$, $q \in Q$, and $m \in \{\bot, \top\}$, we have $\mathcal{F}'((w, q, m)) = \mathcal{F}(q)$.

□

It is left to check whether the language of A_{stra} is contained in the language of A. Since tree automata are closed under complementation [Rab69,Tho97], we can complement A, get an automaton $\tilde{A}$, and then check the nonemptiness of the intersection of A_{stra} with $\tilde{A}$. Hence the following theorem.

Theorem 5. *Given a plant P and a formula φ in CTL, the control problem for φ is in 2EXPTIME. More precisely, it can be solved in time $O(\exp(|P|^2 \cdot 2^{O(|\varphi|)}))$. For φ in CTL*, the problem is in 3EXPTIME. More precisely, it can be solved in time $O(\exp(|P|^2 \cdot 2^{2^{O(|\varphi|)}}))$.*

Proof. For the complexity of this procedure, it is easy to see that if φ is in CTL, the automaton A has a state-space size of $O(|P| \cdot 2^{O(|\varphi|)})$. Though A runs on k-ary trees (where k depends on P), it can be complemented as easily as automata on binary trees — the complemented automaton $\tilde{A}$ (as well as its intersection with A_{stra}) is a parity automaton with $O(\exp(|P| \cdot 2^{O(|\varphi|)}))$ states and $O(|P| \cdot 2^{O(|\varphi|)})$ colors[1] ([Tho97]). Since emptiness of parity tree automata can be done in time polynomial in the state-space and exponential in the number of colors [EJ88,PR89], we can check emptiness of this intersection in time $O(\exp(|P|^2 \cdot 2^{O(|\varphi|)}))$. For CTL* specifications, the analysis is similar only that the complexity contributed by the formula increases by one exponential. □

By [Rab69], if there is indeed a strategy that is winning for the system, then the automaton that is the product of A_{stra} and the complement of the automaton constructed on in Theorem 4 accepts it and when we test it for emptiness, we can get a *regular* tree accepted by the automaton. This then provides a finite-memory winning strategy that can be realized as a finite state controller for the system.

4 Lower bounds

For two 2^{AP}-labeled trees (T, V) and (T', V'), and a set $Q = \{q_1, \ldots, q_k\} \subseteq AP$, we say that (T, V) and (T', V') are Q-*different* if they agree on everything except possibly the labels of the propositions in Q. Formally, $T = T'$ and for all $x \in T$, we have $V(x) \setminus Q = V'(x) \setminus Q$. The logic AQCTL* extends CTL* by universal quantification on atomic proposition: if ψ is a CTL* formula and $q_1, \ldots, q_k$ are atomic propositions, then $\forall q_1, \ldots, q_k \psi$ is an AQCTL* formula. The semantics of $\forall q_1, \ldots, q_k \psi$ is given by $S \models \forall q_1, \ldots, q_k \psi$ iff for all trees (T, V) such that (T, V) and the unwinding (T_S, V_S) of S are $\{q_1, \ldots, q_k\}$-different, $(T, V) \models \psi$. The logics AQLTL and AQCTL are defined similarly as the extensions of LTL and CTL with universal quantification on atomic propositions.

The following Theorem is taken from [SVW87]. We describe here the full proof, as our lower-bound proofs are based on it.

Theorem 6. [SVW87] *The satisfiability problem for AQLTL is EXPSPACE-hard.*

Proof. We do a reduction from the problem whether an exponential-space deterministic Turing machine T accepts an input word x. That is, given T and x, we construct an AQLTL formula formula $\forall q \varphi$ such that T accepts x iff $\forall q \varphi$ is satisfiable. Below we describe the formula φ informally. The formal description of φ and of the function *next* we use below are given in Appendix C.1.

Let $T = \langle \Gamma, Q, \rightarrow, q_0, F \rangle$, where Γ is the alphabet, Q is the set of states, $\rightarrow \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ is the transition relation (we use $(q, a) \rightarrow (q', b, \Delta)$ to indicate that when T is in state q and it reads the input a in the current tape cell, it moves to state q', writes b in the current tape cell, and its reading head moves one cell to the left or to the right, according to Δ), q_0 is the initial state, and $F \subseteq Q$ is the set of accepting states. Let $n = a \cdot |x|$, for some constant a, be

[1] $exp(x)$ stands for $2^{O(x)}$

such that the working tape of T has 2^n cells. We encode a configuration of T by a word $\gamma_1\gamma_2\ldots(q,\gamma_i)\ldots\gamma_{2^n}$. That is, all the letters in the configuration are in Γ, except for one letter in $Q\times\Gamma$. The meaning of such a configuration is that the j's cell of T, for $1\leq j\leq 2^n$, is labeled γ_j, the reading head points on cell i, and T is in state q. For example, the initial configuration of T is $(q_0,x_1)x_2\cdots x_n\#\#\cdots$, where # stands for the empty cell. We can now encode a computation of T by a sequence of configurations.

Let $\Sigma=\Gamma\cup(Q\times\Gamma)$. We can encode letters in Σ by a set $AP(T)=\{p_1,\ldots,p_m\}$ (with $m=\lceil log|\Sigma|\rceil$) of atomic propositions). We define our formulas over the set $AP=AP(T)\cup\{b,c,d,e,q\}$ of atomic propositions. The task of the last five atoms will be explained shortly. Since T is fixed, so is Σ, and hence so is the size of AP.

Consider an infinite sequence π over 2^{AP}. For an atomic proposition $p\in AP$ and a node u in π, we use $p(u)$ to denote the truth value of p at u. That is, $p(u)$ is 1 if p holds at u and is 0 if p does not hold at u. We divide the sequence π to blocks of length n. Every such block corresponds to a single tape cell of the machine T. Consider a block $u_1,\ldots,u_n$ that corresponds to a cell ρ. We use the node u_1 to encode the content of cell ρ. Thus, the bit vector $p_1(u_1),\ldots,p_m(u_1)$ encodes the letter (in $\Gamma\cup(Q\times\Gamma)$) that corresponds to cell ρ. We use the atomic proposition b to mark the beginning of the block; that is, b should hold on u_1 and fail on $u_2,\ldots,u_n$ (see **C1** in Appendix C.1).

Recall that the letter with which cell ρ is labeled is encoded at the node u_1 of the block $u_1,\ldots,u_n$ that corresponds to ρ. Why then do we need a block of length n to encode a single letter? The block also encodes the location of the cell ρ on the tape. Since T is an exponential-space Turing machine, this location is a number between 0 and 2^n-1. Encoding the location eliminates the need for exponentially many X operators when we attempt to relate two successive configurations. Encoding is done by the atomic proposition c, called *counter*, Let $c(u_n),\ldots,c(u_1)$ encode the location of ρ. Note that, for technical convenience, the least significant bit of the counter is in u_1. A sequence of 2^n blocks corresponds to 2^n cells and encodes a configuration of T. The value of the counters along this sequence goes from 0 to 2^n-1, and then start again from 0. This is enforced by φ. Since we want the size of φ to be $O(n)$, we need also an atomic proposition d that acts as a "carry" bit (see **C2** and **C3**).

An atomic proposition e marks the last block of a configuration, that is, e holds in a node u_1 of a block $u_1,\ldots,u_n$ iff c holds on all nodes in the block (see **C4**).

Let $\sigma_1\ldots\sigma_{2^n},\sigma'_1\ldots\sigma'_{2^n}$ be two successive configurations of T. For each triple $\langle\sigma_{i-1},\sigma_i,\sigma_{i+1}\rangle$ with $1\leq i\leq 2^n$ (taking σ_{2^n+1} to be σ'_1 and σ_0 to be the label of the last letter in the configuration before $\sigma_1\ldots\sigma_{2^n}$, or some special label when $\sigma_1\ldots\sigma_{2^n}$ is the initial configuration), we know, by the deterministic transition relation of T, what σ'_i should be. Let $next(\langle\sigma_{i-1},\sigma_i,\sigma_{i+1}\rangle)$ denote our expectation for σ'_i. The formal definition of $next$ can be found in Appendix C.1.

Consistency with $next$ gives us a necessary condition for a word to encode a legal computation. In addition, the computation should start with the initial configuration. Finally, if the computation ends with an accepting configuration (that is, one with (q,γ_i) with $q\in F$), then T accepts x. It is easy to specify in

LTL the requirements about the initial and accepting configurations. For a letter $\sigma \in \Sigma$, let $\eta(\sigma)$ be the propositional formula over AP that encodes σ. That is, $\eta(\sigma)$ holds in node u_1 of a block that encodes cell ρ iff the cell ρ is labeled σ. Then, φ contains conjuncts (see **C5** and **C6**) that require the computation to start with the initial configuration and to eventually reach an accepting configuration.

The difficult part in the reduction is in guaranteeing that the sequence of configurations is indeed consistent with $next$. To enforce this, we have to relate σ_{i-1}, σ_i, and σ_{i+1} with σ'_i for any i in any two successive configurations $\sigma_1 \ldots \sigma_{2^n}, \sigma'_1 \ldots \sigma'_{2^n}$. One natural way to do so is by a conjunction of formulas like "whenever we meet a cell with counter $i-1$ and the labeling of the next three cells forms the triple $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle$, then the next time we meet a cell with counter i, this cell is labeled $next(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle)$". The problem is that as i can take any value from 1 to 2^n, there are exponentially many such conjuncts. This is where the universal quantification of the AQLTL comes into the picture. It enables us to relate $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle$ with σ'_i, for all i.

To understand how this is done, consider the atomic proposition q, and assume that the following hold. **(1)** q is true at precisely two points, both are points in which a block starts, **(2)** there is exactly one point between them (possibly in exactly one of them) in which e holds (thus, the two points are in successive configurations), and **(3)** the value of the counter at the blocks starting at the two points is the same. Then, it should be true that **(4)** if the labels of the three blocks starting one block before the first q are σ_{i-1}, σ_i, and σ_{i+1}, then the block starting at the second q is labeled by $next(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$.

Using b, c, and e, we can easily express **(1)**-**(4)** with formulas of length polynomial in n (recall that as the set Σ is fixed, scanning all the possible labels of a cell can be done with a formula of a fixed length). Note that for expressing **(3)**, we need to compare the value of the two counters bit by bit (see **C7** - **C10**). The formula φ contains the conjunct $((\mathbf{1}) \wedge (\mathbf{2}) \wedge (\mathbf{3}))$. Since the condition $(\mathbf{4})$ is checked in $\forall q\varphi$ for all the assignments to q that satisfy $(\mathbf{1}) \wedge (\mathbf{2}) \wedge (\mathbf{3})$, it follows that φ is satisfied only in computations consistent with $next$. Hence, $\forall q\varphi$ is satisfiable iff there is an accepting computation of T on x. ☐

We now show that AQCTL is also strong enough to describe an exponential-space Turing machine with a formula of polynomial length. Moreover, since CTL has both universal and existential path quantification, AQCTL can describe an alternating exponential-space Turing machine, implying a 2EXPTIME lower bound for its satisfiability problem.

Theorem 7. *The satisfiability problem for AQCTL is 2EXPTIME-hard.*

Proof. We do a reduction from the problem whether an exponential-space alternating Turing machine T accepts an input word x. That is, given T and x, we construct an AQCTL formula $\forall q\psi$ such that T accepts x iff $\forall q\psi$ is satisfiable.

Let $T = \langle \Gamma, Q_u, Q_e, \mapsto, q_0, F \rangle$, where the sets Q_u and Q_e of states are disjoint, and contain the universal and the existential states, respectively. We denote their union (the set of all states) by Q. Our model of alternation prescribes that $\mapsto \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ has a binary branching degree. When a universal or an existential state of T branches into two states, we distinguish between the left and the right branches. Accordingly, we use $(q, a) \mapsto \langle(q_l, b_l, \Delta_l), (q_r, b_r, \Delta_r)\rangle$ to indicate that when T is in state $q \in Q_u \cup Q_e$ reading input symbol a, it branches to the left with (q_l, b_l, Δ_l) and to the right with (q_r, b_r, Δ_r). (Note that the directions left and right here have nothing to do with the movement direction of the head; these are determined by Δ_l and Δ_r.)

For a configuration c of T, let $succ_l(c)$ and $succ_r(c)$ be the successors of c when applying to it the left and right choices in $\mapsto$, respectively. Given an input x, a computation tree of T on x is a tree in which each node corresponds to a configuration of T. The root of the tree corresponds to the initial configuration. A node that corresponds to a universal configuration c has two successors, corresponding to $succ_l(c)$ and $succ_r(c)$. A node that corresponds to an existential configuration c has a single successor, corresponding to either $succ_l(c)$ or $succ_r(c)$. The tree is an accepting computation tree if all its branches reach an accepting configuration.

The formula ψ will describe accepting trees. As in the linear case, we encode a configuration of T by a sequence $\gamma_1\gamma_2 \ldots (q, \gamma_i) \ldots \gamma_{2^n}$, and we use a block of length n to describe each letter $\sigma_i \in \Gamma \cup (Q \times \Gamma)$ in the sequence. The construction of ψ is similar to the construction described for φ in the linear case. For an LTL formula ξ, let ξ_A be the CTL formula obtained from ξ by preceding each temporal operator by the path quantifier A. For example, $(G(p \to Fq))_A = AG(p \to AFq)$, and $(GFp)_A = AGAFp$. As in the linear case, the atomic propositions c and d are used to count, b is used to mark the beginning of blocks, and e is used to mark the last letter in a configuration. Note that the conjuncts ξ in φ that impose the desired behavior of b, c, d, and e are such that ξ_A impose the desired behavior of b, c, d and e in the branching case. Also, the conjuncts ξ used in φ in order to check that the first configuration is the initial one and that the computation is accepting are such that ξ_A do the same for the branching case. Our formula ψ has all these conjuncts.

The difficult part is to check that the $succ_l$ and $succ_r$ relations are maintained. For that, we add two atomic propositions, e_E and e_U, that refine the proposition e and indicate whether the configuration just ended has been existential or universal. Formally, ψ contains the conjuncts

- $AG(e \leftrightarrow (e_U \vee e_E)) \wedge AG(\neg(e_U \wedge e_E))$,
- $AG(\bigwedge_{q \in Q_e, \gamma \in \Gamma}(\eta(q, \gamma) \to A(\neg e)U(e \wedge e_E)))$, and
- $AG(\bigwedge_{q \in Q_u, \gamma \in \Gamma}(\eta(q, \gamma) \to A(\neg e)U(e \wedge e_U)))$.

In addition, we use an atomic proposition l to indicate whether the nodes belong to a left or a right successor. For clarity, we denote $\neg l$ by r. Formally, ψ contains the conjunct

$$AG(l \to (AlUe)) \wedge AG(r \to (ArUe)).$$

Since a universal configuration c has both $succ_l(c)$ and $succ_r(c)$ as successors, and an existential c has only one of them, ψ also contains the conjunct

$$AG(e_E \to (EXl \vee EXr)) \wedge AG(e_U \to (EXl \wedge EXr)) \wedge AG(e \to AX(l \vee r)).$$

We can now use universal quantification over atomic propositions in order to check consistency with $succ_l$ and $succ_r$. Note that $succ_l(c)$ and $succ_r(c)$ are uniquely defined. Thus, we can define functions, $next_l$ and $next_r$, analogous to function $next$ of the linear case. Given a sequence $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle$ of letters in c, the function $next_l(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle)$ returns the expectation for the i'th letter in $succ_l(c)$. We denote this letter by σ_i^l, and similarly for $next_r$ and σ_i^r.

In the linear case, we considered assignments to q in which q holds at exactly two points in the computation. Here, we look at assignments where q holds at exactly two points in each branch. The first point is a node where a block of σ_i starts, and the second point is a node where a block of σ_i^l or σ_i^r starts (note that each assignments to q may check consistency with $succ$ along different branches)[2]

Consider the atomic proposition q, and assume that the following hold: **(1)** q is labeled as described above (in particular, in each branch of the tree, q is true at precisely two nodes, both are nodes in which a block starts), **(2)** in every branch with two occurrences of q, there is exactly one node between them in which e holds (thus, the two nodes are in successive configurations), and **(3)** the value of the counter at the blocks starting at the two points is the same. Then, it should be true that **(4)** if the labels of the three blocks starting one block before the first q are σ_{i-1}, σ_i, and σ_{i+1}, then the blocks starting at the second q are either left branches in which case they are labeled by $next_l(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$, or they are right branches, in which case they are labeled by $next_r(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$. Formally, ψ contains the conjunct $((\mathbf{1}) \wedge (\mathbf{2}) \wedge (\mathbf{3})) \rightarrow (\mathbf{4})$, where **(1)**-**(4)** are as described in Appendix C.2. Since in $\forall q \psi$, the condition **(4)** is checked for all the assignments to q that satisfy $(\mathbf{1}) \wedge (\mathbf{2}) \wedge (\mathbf{3})$, it follows that ψ is satisfied only in a computation tree consistent with $succ_l$ and $succ_r$. Hence, $\forall q \psi$ is satisfiable iff there is an accepting computation tree of T on x. □

The satisfiability problem for CTL⋆ is exponentially harder than the one for CTL. We now show that this computational difference is preserved when we look at the extensions of these logics with universal quantification over atomic propositions.

Theorem 8. *The satisfiability problem for A*Q*CTL⋆ is 3EXPTIME-hard.*

Proof. We do a reduction from the problem whether a doubly-exponential-space alternating Turing machine T accepts an input word x. That is, given T and x, we construct an AQCTL⋆ formula $\forall q \psi$ such that T accepts x iff ψ is satisfiable.

In [VS85], the satisfiability problem of CTL⋆ is proved to be 2EXPTIME-hard by a reduction from an exponential-space alternating Turing machine. Below we explain how universal quantification can be used to "stretch" the length of the tape that a polynomial CTL⋆ formula can describe by another exponential. As in the proof of Theorem 6, the formula in [VS85] maintains an n-bit counter, and each cell of T's tape corresponds to a block of length n.

In order to point on the letters σ_i and σ_i^l simultaneously (that is, the letters that the atomic proposition q point on in the proof of Theorem 6), [VS85] adds to each node of the tree a branch such that nodes that belong to the original tree

[2] It is convenient to think of a satisfying tree for ψ as a tree that has branching degree 1 everywhere except for nodes labeled by e_U, where the branching degree is 2. Our reduction, however,makes no assumption about such a structure.

are labeled by some atomic proposition p, and nodes that belong to the added branches are not labeled by p. Every path in the tree has a single location where the atom p stops being true. [VS85] uses this location in order to point on σ' and in order to compare the values of the n-bit counter in the current point (where σ is located) and in the point in the computation where p stops being true.

On top of the method in [VS85], we use the universal quantification in order to maintain a 2^n-bit counter and thus count to 2^{2^n}. Typically, each bit of our 2^n-bit counter is kept in a block of length n, which maintains the index of the bit (a number between 0 to $2^n - 1$). For example, when $n = 3$, the counter looks as follows.

000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111	$\leftarrow$	n-bit counter
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	$\leftarrow$	2^n-bit counter
000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111 ...		
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1 ...		

To check that the 2^n-bit counter proceeds properly, we use a universally quantified proposition q and we check that if q holds at exactly two points (say, last points in a block of the n-bit counter), with the same value to the n-bit counter, and with only one block between them in which the n-bit counter has value 1^n, then the bit of the 2^n-bit counter that is maintained at the block of the second q is updated properly (we also need to relate and update carry bits, but the idea is the same). $\square$

Note that the number of atomic propositions in ψ in the proofs of both Theorems 7 and 8 is fixed. Note also that if ψ is satisfiable, then it is also satisfied in a tree of a fixed branching degree (a careful analysis can show that for CTL the sufficient branching degree is 2, and for CTL* it is 3).

The logic EAQCTL* extends AQCTL* by adding existential quantification on atomic propositions: if $\forall q_1, \ldots, q_k \psi$ is an AQCTL* formula and $p_1, \ldots, p_m$ are atomic propositions, then $\exists p_1, \ldots, p_m \forall q_1, \ldots, q_k \psi$ is an EAQCTL* formula. The semantics of $\exists p_1, \ldots, p_m \forall q_1, \ldots, q_k \psi$ is given by $S \models \exists p_1, \ldots, p_m \forall q_1, \ldots, q_k \psi$ iff there is a tree (T, V) such that (T_S, V_S) and (T, V) are $\{p_1, \ldots, p_m\}$-different and $(T, V) \models \forall q_1, \ldots, q_k \psi$. The logic EAQCTL is the subset of EAQCTL* corresponding to CTL. For $n \geq 1$, let $[n] = \{1, \ldots, n\}$, and let $S_n = \langle \emptyset, [n], [n] \times [n], 1, L \rangle$ be the structure whose transition relation is the n-state clique (note that since S_n has no atomic propositions, its labeling function L is redundant). For an AQCTL* formula ψ, let $width(\psi)$ be the *sufficient branching degree* for ψ; that is, $width(\psi)$ is such that if there is some tree that satisfies ψ, then there is also a tree with branching degree $width(\psi)$ that satisfies ψ. Recall that the semantics of EAQCTL* formulas is defined with respect to the unwinding (T_S, V_S), for structures S. Hence, as detailed in [Kup97], the satisfiability problem for the AQCTL* formula $\forall q_1, \ldots, q_k \psi$ can be reduced to the model-checking problem of the EAQCTL* formula $\exists p_1, \ldots, p_m \forall q_1, \ldots, q_k \psi$ in $S_{width(\psi)}$. Since the formulas used in the proof of Theorems 7 and 8 have fixed widths, the following Theorem follows immediately from Theorems 7 and 8.

Theorem 9. *The model-checking problems for EAQCTL and EAQCTL* are 2EXPTIME-hard and 3EXPTIME-hard in the size of the specification, respectively.*

Intuitively, the model-checking problem for EAQCTL* asks whether we can find an assignment to the propositions that are existentially quantified so that no matter how we assign values to the propositions that are universally quantified, the formula is satisfied. Recall that in the control problem we ask a similar question, namely whether we can find a strategy for the system so that no matter which strategy the environment uses, the formula is satisfied. In the theorem below we make the relation between existential and universal quantification over atomic propositions and supervisory control formal. The relation is similar to the relation between existential quantification and the module-checking problem, as described in [KV96].

Theorem 10. *Given a structure S and an EAQCTL* formula $\exists p_1, \ldots, p_m \forall q_1, \ldots, q_k \psi$, there is a plant P and a CTL* formula ψ' such that $|P| = O((1+k+m) \cdot |S|)$, $|\psi'| = O(|S| + |\psi|)$, and $S \models \exists p_1, \ldots, p_m \forall q_1, \ldots, q_k \psi$ iff controllable(P, ψ).*

Proof. See Appendix C.3. □

Since the number of atomic propositions in the formulas used in the reductions in Theorems 7 and 8 is fixed, and since in the case P is fixed the size of ψ' in Theorem 10 is $O(|\psi|)$, we can conclude with the following.

Theorem 11. *The supervisory control problems for CTL and CTL* are 2EXPTIME-hard and 3EXPTIME-hard in the size of the specification, respectively.*

5 Discussion

Our results shed an additional light on the discussion regarding the relative merits of linear versus branching temporal logics, cf. [Lam80,Pnu85]. We mainly refer here to the linear temporal logic LTL and the branching temporal logic CTL. One of the beliefs dominating this discussion has been "while specifying is easier in LTL, model checking is easier for CTL". As is argued in [KV96,KV97a,KV99b], the computational advantage of CTL over LTL (cf. [CGP99]), disappears once one considers reactive environments. Our results here show that the same phenomenon occurs in the context of the synthesis problem and the control problem: once one considers reactive environments, the problems for LTL and CTL are equally hard (2EXPTIME-complete).

In a setting with incomplete information the system (resp., the environment) may not be able to observe all the signals generated by the environment (resp., the system), so it strategy needs to depend only on the observed signals. The effect that incomplete information has on the complexity of the synthesis and control problems can vary dramatically, from having no impact [KV97b] to causing undecidability [PR90]. An interesting question that deserves further study is whether we can handle incomplete information in our setting within the same complexity bounds.

References

[ALW89] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 16th Int. Colloquium on Automata, Languages and Programming*, volume 372, pages 1–17. Lecture Notes in Computer Science, Springer-Verlag, July 1989.

[Ant95] M. Antoniotti. *Synthesis and verification of discrete controllers for robotics and manufacturing devices with temporal logic and the Control-D system.* PhD thesis, New York University, New York, 1995.

[BL69] J.R. Büchi and L.HG. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[Chu63] A. Church. Logic, arithmetics, and automata. In *Proc. International Congress of Mathematicians, 1962*, pages 23–35. institut Mittag-Leffler, 1963.

[DTV99] M. Daniele, P. Traverso, and M.Y. Vardi. Strong cyclic planning revisited. In S. Biundo and M. Fox, editors, *5th European Conference on Planning*, pages 34–46, 1999.

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symposium on Foundations of Computer Science*, pages 328–337, White Plains, October 1988.

[Eme90] E.A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, pages 997–1072, 1990.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 477–498. Springer-Verlag, 1985.

[KG95] R. Kumar and V.K. Garg. *Modeling and control of logical discrete event systems.* Kluwer Academic Publishers, 1995.

[KG96] O. Kupferman and O. Grumberg. Buy one, get one free!!! *Journal of Logic and Computation*, 6(4):523–539, 1996.

[KS95] R. Kumar and M.A. Shayman. Supervisory control of nondeterministic systems under partial observation and decentralization. *SIAM Journal of Control and Optimization*, 1995.

[Kup97] O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. *Journal of Logic and Computation*, 7:1–14, 1997.

[KV96] O. Kupferman and M.Y. Vardi. Module checking. In *Computer Aided Verification, Proc. 8th Int. Conference*, volume 1102 of *Lecture Notes in Computer Science*, pages 75–86. Springer-Verlag, 1996.

[KV97a] O. Kupferman and M.Y. Vardi. Module checking revisited. In *Computer Aided Verification, Proc. 9th Int. Conference*, volume 1254 of *Lecture Notes in Computer Science*, pages 36–47. Springer-Verlag, 1997.

[KV97b] O. Kupferman and M.Y. Vardi. Synthesis with incomplete informatio. In *2nd International Conference on Temporal Logic*, pages 91–106, Manchester, July 1997.

[KV99a] O. Kupferman and M.Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245 – 263, June 1999.

[KV99b] O. Kupferman and M.Y. Vardi. Robust satisfaction. In *Proc. 10th Conference on Concurrency Theory*, Lecture Notes in Computer Science. Springer-Verlag, August 1999.

[KV00] O. Kupferman and M.Y. Vardi. μ-calculus synthesis. Submitted, January 2000.

[Lam80] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proc. 7th ACM Symposium on Principles of Programming Languages*, pages 174–185, January 1980.

[MT98] P. Madhusudan and P. S. Thiagarajan. Controllers for discrete event systems via morphisms. In *In Davide Sangiorgi and Robert de Simone, editors, CONCUR'98, Concurrency Theory, 9th International Conference, Proceedings*, volume 1466 of *Lecture Notes in Computer Science*, pages 18–33, Nice, France, September 1998. Springer-Verlag.

[MT00] P. Madhusudan and P. S. Thiagarajan. Branching time controllers for discrete event systems. *To appear in CONCUR'98 Special Issue, Theoretical Computer Science*, 2000.

[Pnu85] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Proc. Advanced School on Current Trends in Concurrency*, pages 510–584, Berlin, 1985. Volume 224, LNCS, Springer-Verlag.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symposium on Principles of Programming Languages*, Austin, January 1989.

[PR90] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. 31st IEEE Symposium on Foundation of Computer Science*, pages 746–757, 1990.

[Rab69] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

[Rab70] M.O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Math. Logic and Foundations of Set Theory*, pages 1–23. North Holland, 1970.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symposium on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.

[SVW87] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.

[Tho97] W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1997.

[Var95] M.Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In P. Wolper, editor, *Computer Aided Verification, Proc. 7th Int'l Conf.*, volume 939 of *Lecture Notes in Computer Science*, pages 267–292. Springer-Verlag, Berlin, 1995.

[VS85] M.Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc 17th ACM Symp. on Theory of Computing*, pages 240–251, 1985.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

A Branching-time logics CTL and CTL*

In the branching time logic CTL*, a path quantifier, E denoting "for some path" (or A "for all paths"), can prefix an assertion composed of an arbitrary combination of the linear time operators X ("next") and U ("until"). There are two types of formulas: *state formulas*, whose satisfaction is related to a specific state, and *path formulas*, whose satisfaction is related to a specific path. Let AP be a set of atomic propositions. A CTL* state formula is either:

- p, for $p \in AP$.
- $\neg\varphi$ or $\varphi \vee \psi$, where φ and ψ are CTL* state formulas.
- $E\varphi$ where φ is a CTL* path formula.

A path formula is either:

- A CTL* state formula.
- $\neg\varphi$ or $\varphi \vee \psi$ or $X\varphi$, or $\varphi U \psi$, where φ and ψ are CTL* path formulas.

The logic CTL* consists of the set of state formulas generated by the above rules.

The logic CTL is a restricted subset of CTL*. In CTL, the temporal operators X, and U must be immediately preceded by a path quantifier. Formally, it is the subset of CTL obtained by restricting the path formulas to be $X\varphi$ or $\varphi U \psi$, where φ and ψ are CTL state formulas.

The semantics of CTL* (and its sub-logic CTL) is defined with respect to a *(Kripke) structure* $S = \langle AP, W, R, w_0, L\rangle$, where AP is the set of atomic propositions, W is a set of states, $R \subseteq W \times W$ is a transition relation that must be total (i.e., for every $w \in W$ there exists $w' \in W$ such that $R(w, w')$), w_0 is an initial state, and $L : W \to 2^{AP}$ maps each state to a set of atomic propositions true in this state. For w and w' with $R(w, w')$, we say that w' is a successor of w . A *path* of S is an infinite sequence $\pi = w^0, w^1, \ldots$ of states such that for every $i \geq 0$, we have $R(w^i, w^{i+1})$. The suffix $w^i, w^{i+1}, \ldots$ of π is denoted by π^i. We use $w \models \varphi$ to indicate that a state formula φ holds at state w, and we use $\pi \models \varphi$ to indicate that a path formula φ holds at path π (assuming a structure S). The relation $\models$ is inductively defined as follows.

- For an atomic proposition $p \in AP$, we have $w \models p$ iff $p \in L(w)$
- $w \models \neg\varphi$ iff $w \not\models \varphi$.
- $w \models \varphi \vee \psi$ iff $w \models \varphi$ or $w \models \psi$.
- $w \models E\varphi$ iff there exists a path $\pi = w_0, w_1, \ldots$ such that $w_0 = w$ and $\pi \models \varphi$.
- $\pi \models \varphi$ for a state formula φ iff $w^0 \models \varphi$.
- $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$.
- $\pi \models \varphi \vee \psi$ iff $\pi \models \varphi$ or $w \models \psi$.
- $\pi \models X\varphi$ iff $\pi^1 \models \varphi$.
- $\pi \models \varphi U \psi$ iff there exists $j \geq 0$ such that $\pi^j \models \psi$ and for all $0 \leq i < j$, we have $\pi^i \models \varphi$.

In the paper, we also use the following abbreviations.

- $\varphi \wedge \psi = \neg((\neg\varphi) \vee (\neg\psi))$ ("and").
- $F\varphi = \mathbf{true} U \varphi$ ("eventually").
- $G\varphi = \neg F \neg\varphi$ ("always").
- $A\varphi = \neg E \neg\varphi$ ("for all paths").

B Proof of Lemma 1

We define $P = (AP', W_s, W_e, R, w_0, L)$ as follows:

- $AP' = AP \cup \{p_e\}$ with $p_e \notin AP$. (The role of p_e will become clear soon).
- $W_e = 2^I \times 2^O$
- $W_s = \{w_0\} \cup 2^I$ with $w_0 \notin 2^I \cup W_e$
- $R = R_0 \cup R_1 \cup R_2$ where $R_0 = \{w_0\} \times (\{\emptyset\} \times 2^O)$, $R_1 = W_e \times 2^I$, and $R_2 = \{(X, (X, Y)) | X \subseteq I \text{ and } Y \subseteq 0\}$.
- $L((X, Y)) = X \cup Y \cup \{p_e\}$ for each $(X, Y) \in W_e$ and $L(w) = \emptyset$ for each $w \in W_s$.

Next we construct the CTL formula φ' over AP' by setting $\varphi' = \varphi'_1 \wedge \varphi'_2$. The basic idea is that φ'_1 ensures that the truthhood of φ matters only at the states in W_e. The conjunct φ'_1 is the formula $EX(\|\varphi\|)$ where $\|\varphi\|$ is given inductively via: $\|p\| = p$ for $p \in AP$, $\|\neg\varphi\| = \neg\|\varphi\|$ and $\|\varphi_1 \vee \varphi_2\| = \|\varphi_1\| \vee \|\varphi_2\|$. Further, $\|EX\varphi\| = EXEX(\|\varphi\|)$ and $\|E(\varphi_1 U \varphi_2)\| = E((p_e \rightarrow \|\varphi_1\|)U(p_e \wedge \|\varphi_2\|))$. Finally, $\|A(\varphi_1 U \varphi_2)\|$ is defined by replacing E by A in the clause for $\|E(\varphi_1 U \varphi_2)\|$.

The conjunct φ'_2 ensures that the system chooses only one move at states in W_s (since the 2^O labelling required must be unique). It is given by $\varphi'_2 = AG(\neg p_e \rightarrow (\bigwedge_{z \in O}(EXz \rightarrow AXz))$ It is easy to check that P and φ' have the required properties.

C Lower bound

C.1 AQLTL satisfiability is EXPSPACE-hard

Below we specify the conjuncts described informally in the proof of Theorem 6.

C1. b should hold on u_1 and fail on $u_2, \ldots, u_n$:

$$b \wedge X(\neg b \wedge X(\neg b \wedge \cdots \wedge X\neg b) \cdots) \wedge G(b \leftrightarrow X^n b)$$

C2. The counter starts at 0:

$$\neg c \wedge X(\neg c \wedge X(\neg c \ldots \wedge X\neg c) \cdots)$$

C3. The counter is increased properly. Note that as we always want to increase it by 1 we take b as a carry to the least significant bit.
- $G(((b \vee d) \wedge \neg c) \rightarrow (X(\neg d) \wedge X^n c))$.
- $G((\neg(b \vee d) \wedge \neg c) \rightarrow (X(\neg d) \wedge X^n \neg c))$.
- $G(((b \vee d) \wedge c) \rightarrow (Xd \wedge X^n \neg c))$.
- $G((\neg(b \vee d) \wedge c) \rightarrow ((X\neg d) \wedge X^n c))$.

C4. e holds in a node u_1 of a block with counter 1^n:

$$G(e \leftrightarrow (b \wedge c \wedge X(c \wedge X(c \wedge X(c \cdots))).$$

C5. The computation starts with the initial configuration:

$$\eta(q_0, x_1) \wedge X^n(\eta(x_2) \cdots \wedge X^n(\eta(x_n) \wedge X^n(\eta(\#) \wedge [(\eta(\#) \rightarrow X^n \eta(\#)) U(\eta(\#) \wedge e)])) \cdots)).$$

C6. The computation reaches an accepting configuration:

$$F(b \wedge \bigvee_{q \in F, \gamma \in \Gamma} \eta(q, \gamma)).$$

C7. q is true at precisely two points, both are points in which a block starts:

$$(\neg q)U(b \wedge q \wedge X((\neg q)U(b \wedge q \wedge XG\neg q)))$$

C8. There is exactly one point between them in which e holds:

$$(\neg q)U(q \wedge ((\neg e)U(e \wedge X((\neg e)Uq)))).$$

C9. The value of the counter at the blocks starting at the two points is the same:

$$\begin{gathered}(\neg q)Uq \wedge [(c \to XF(q \wedge c)) \wedge ((\neg c) \to XF(q \wedge (\neg c)))\wedge \\ X((c \to F(q \wedge Xc)) \wedge ((\neg c) \to F(q \wedge X(\neg c)))\wedge \\ X((c \to F(q \wedge XXc)) \wedge ((\neg c) \to F(q \wedge XX(\neg c)))\wedge \\ \vdots \\ X((c \to F(q \wedge X^{n-1}c)) \wedge ((\neg c) \to F(q \wedge X^{n-1}(\neg c)))) \cdots))].\end{gathered}$$

C10. if the labels of the three blocks starting one block before the first q are σ_{i-1}, σ_i, and σ_{i+1}, then the block starting at the second q is labeled by $next(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$:

$$\bigvee_{\sigma_1, \sigma_2, \sigma_3 \in \Sigma} (\neg q)U(\eta(\sigma_1) \wedge X^n(q \wedge \eta(\sigma_2) \wedge X^n(\eta(\sigma_3) \wedge F(q \wedge \eta(next(\sigma_1, \sigma_2, \sigma_3)))))).$$

The function ***next*****.**
$next(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle)$, which denote our expectation for σ'_i is defined as follows [3].

- $next(\langle \gamma_{i-1}, \gamma_i, \gamma_{i+1} \rangle) = \gamma_i$.
- $next(\langle (q, \gamma_{i-1}), \gamma_i, \gamma_{i+1} \rangle) = \begin{cases} \gamma_i & \text{If } (q, \gamma_{i-1}) \to (q', \gamma'_{i-1}, L). \\ (q', \gamma_i) & \text{If } (q, \gamma_{i-1}) \to (q', \gamma'_{i-1}, R). \end{cases}$
- $next(\langle \gamma_{i-1}, (q, \gamma_i), \gamma_{i+1} \rangle) = \gamma'_i$ where $(q, \gamma_i) \to (q', \gamma'_i, \Delta)$.
- $next(\langle \gamma_{i-1}, \gamma_i, (q, \gamma_{i+1}) \rangle) = \begin{cases} \gamma_i & \text{If } (q, \gamma_{i+1}) \to (q', \gamma'_{i+1}, R). \\ (q', \gamma_i) & \text{If } (q, \gamma_{i+1}) \to (q', \gamma'_i, L). \end{cases}$

C.2 AQCTL satisfiability is 2EXPTIME-hard

(1) q is labeled as described above:

$$A(\neg q)U(b \wedge q \wedge AXA((\neg q)U(b \wedge q \wedge AXAG\neg q))).$$

[3] We assume that T's head does not "fall" from the right or the left boundaries of the tape. Thus, the case where $i = 1$ and $(q, \gamma_i) \to (q', \gamma'_i, L)$ and the dual case where $i = 2^n$ and $(q, \gamma_i) \to (q', \gamma'_i, R)$ are not possible.

(2) In every branch with two occurrences of q, there is exactly one node between them in which e holds (thus, the two nodes are in successive configurations):

$$A(\neg q)U(q \wedge A((\neg e)U(e \wedge AXA((\neg e)Uq)))).$$

(3) The value of the counter at the blocks starting at the two points is the same:

$$\begin{array}{c} A(\neg q)Uq \wedge [(c \rightarrow AXAF(q \wedge c)) \wedge ((\neg c) \rightarrow AXAF(q \wedge (\neg c)))\wedge \\ AX((c \rightarrow AF(q \wedge AXc)) \wedge ((\neg c) \rightarrow AF(q \wedge AX(\neg c)))\wedge \\ AX((c \rightarrow AF(q \wedge AXAXc)) \wedge ((\neg c) \rightarrow AF(q \wedge AXAX(\neg c)))\wedge \\ \vdots \\ AX((c \rightarrow AF(q \wedge (AX)^{n-1}c)) \wedge ((\neg c) \rightarrow AF(q \wedge (AX)^{n-1}(\neg c)))) \cdots))]. \end{array}$$

(4) if the labels of the three blocks starting one block before the first q are σ_{i-1}, σ_i, and σ_{i+1}, then the blocks starting at the second q are either left branches in which case they are labeled by $next_l(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$, or they are right branches, in which case they are labeled by $next_r(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$:

$$\begin{array}{l} A(\neg q)U(\bigvee_{\sigma_1,\sigma_2,\sigma_3 \in \Sigma}(\eta(\sigma_1) \wedge (AX)^n(q \wedge \eta(\sigma_2) \wedge (AX)^n(\eta(\sigma_3)\wedge \\ AF[(q \wedge l \wedge \eta(next_l(\sigma_1, \sigma_2, \sigma_3))) \vee (q \wedge r \wedge \eta(next_r(\sigma_1, \sigma_2, \sigma_3)))])))). \end{array}$$

C.3 Proof of Theorem 10

For technical convenience, let us first assume that a plant has a third type of states, W_n, which belong to neither the system nor the environment (that is, all the successors of states in W_n are always taken). Let $E = \{p_1, \ldots, p_m\}$ and $U = \{q_1, \ldots, q_k\}$ be the sets of universally and existentially quantified propositions, and let $S = \langle AP, W, R, w_0, L\rangle$. We define $P = \langle AP \cup \{yes, dummy\}, W_n, W_s, W_e, R', w_0, L'\rangle$, where

- $W_n = W \cup \{q_{yes}, q_{no}\}$.
- $W_s = W \times E$.
- $W_e = W \times U$.
- $R' = R \cup \{\langle w, \langle w, r\rangle\rangle : w \in W \text{ and } r \in E \cup U\} \cup ((W \times (E \cup U)) \times \{q_{yes}, q_{no}\}) \cup \{\langle q_{yes}, q_{yes}\rangle, \langle q_{no}, q_{no}\rangle\}$.
- For all $w \in W$ and $r \in E \cup U$, we have $L'(w) = L(w) \setminus (E \cup U)$, $L'(\langle w, r\rangle) = \{r, dummy\}$, $L'(q_{yes}) = \{yes\}$, and $L'(q_{no}) = \emptyset$.

That is, the plant P contains the structure S. Each state w in S is duplicated $k + m$ times. Each copy of w is associated with a quantified proposition. States associated with existentially quantified propositions are system states. States associated with universally quantified propositions are environment states. Each duplicated state is labeled by the proposition it corresponds to and by a new proposition *dummy*. In addition, there are two states q_{yes} and q_{no} that all the duplicated states go to. The new atomic proposition *yes* is true in q_{yes}.

We define ψ' in two steps. First, path quantification in ψ' should be restricted to computations of S. That is, to paths that never meet a duplicated state. To do this, we use a function f : CTL* formulas $\rightarrow$ CTL* formulas that restricts path quantification to paths that never visit a state labeled with *dummy*. For example, $f(EqU(AFp)) = E((G\neg dummy) \wedge (qU(A((F dummy) \vee Fq))))$. The full definition of f and a proof that when ψ is a CTL formula, there is also a CTL formula equivalent to $f(\psi)$, can be found in [KG96,KV96]. We can now define ψ' as $f(\psi)$ with $EX(r \wedge EX yes)$ replacing each occurrence of a quantified proposition r. So, if r is existentially quantified, the system chooses whether it holds or not (by enabling/disabling the transition from the state $\langle w, r\rangle$ to the state q_{yes}), and dually for universal quantification. Note that we first apply f and only then do the replacement. The length of the formula ψ' is linear in the length of ψ.

Finally, we remove the assumption about a plant having a third type of states by adding to ψ' a conjunct that disables the pruning of transitions from W_n. Formally, this is done by adding W to AP, and having formulas like $AG(w_1 \rightarrow (EXw_2 \wedge EXw_3)$ that describe R. This is why the length of ψ' is $O(|\psi| + |P|)$.