μ-calculus Synthesis

Orna Kupferman* Hebrew University

Moshe Y. Vardi† Rice University

June 12, 2000

Abstract

In system synthesis, we transform a specification into a system that is guaranteed to satisfy the specification. When the system is open, it interacts with an environment via input and output signals and its behavior depends on this interaction. An open system should satisfy its specification in all possible environments. In addition to the input signals that the system can read, an environment can also have internal signals that the system cannot read. In the above setting, of *synthesis with incomplete information*, we should transform a specification that refers to both readable and unreadable signals into a system whose behavior depends only on the readable signals. In this work we solve the problem of synthesis with incomplete information for specifications in μ-calculus. Since many properties of systems are naturally specified by means of fixed points, the μ-calculus is an expressive and important specification language. Our results and technique generalize and simplify previous work on synthesis. In particular, we prove that the problem of μ-calculus synthesis with incomplete information is EXPTIME-complete. Thus, it is not harder than the satisfiability or the synthesis problems for this logic.

*Address: School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il, URL: http://www.cs.huji.ac.il/~orna

†Address: Department of Computer Science, Rice University, Houston TX 77005-1892, U.S.A. Email: vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi.

1 Introduction

In computer system design, we distinguish between *closed* and *open* systems [HP85]. A *closed system* is a system whose behavior is completely determined by the state of the system. An *open system* is a system that interacts with its environment and whose behavior depends on this interaction. In *system synthesis*, we transform a specification into a system that is guaranteed to satisfy the specification. Earlier work on synthesis considers closed systems. There, a system that meets the specification can be extracted from a constructive proof that the specification is satisfiable [MW80, EC82]. As argued in [Dil89, PR89, ALW89], such synthesis paradigms are not of much interest when applied to open systems. For open systems, we should distinguish between output signals (generated by the synthesized system), over which we have control, and input signals (generated by the environment), over which we have no control. While satisfaction of the specification only guarantees that we can synthesize a system that satisfies the specification for some environment (that is, for some behavior of the input signals), we would like to synthesize a system that satisfies the specification for all environments.

We now make this intuition more formal. We first consider the linear approach to synthesis, where the specification describes the set of correct computations of the system. Given sets I and O of input and output signals, respectively, we can view a system as a *strategy* $P : (2^I)^* \to 2^O$ that maps a finite sequence of sets of input signals into a set of output signals. When P interacts with an environment that generates infinite input sequences, it associates with each input sequence an infinite computation over $2^{I \cup O}$. Given a linear specification ψ over $I \cup O$, *realizability* of ψ is the problem of determining whether there exists a system P all of whose computations satisfy ψ. Synthesis of ψ then amounts to constructing such P [PR89].

The linear approach to synthesis is closely related to *Church's solvability problem* [Chu63]. There, we are given a relation $R \subseteq (2^I)^\omega \times (2^O)^\omega$ that is definable in the sequential calculus (S1S) and we seek a function $f : (2^I)^\omega \to (2^O)^\omega$, generated by a strategy, such that for all $x \in (2^I)^\omega$, we have $R(x, f(x))$. We can view the relation R as a linear specification for the system: it defines all the permitted pairs of input and output sequences. A function f as above then maps every possible input sequence into a permitted output sequence, and can be therefore viewed as a correct system. The solutions to Church's problem and the synthesis problem are similar [Rab70, PR89], and consist of a reduction to the nonemptiness problem of *tree automata* (an earlier and more complicated solution can be found in [BL69]).

Though the system P is deterministic, it induces a computation tree. The branches of the tree correspond to external nondeterminism, caused by different possible inputs. Thus, the tree has a fixed branching degree $|2^I|$, and it embodies all the possible inputs (and hence also computations) of P[1]. When we synthesize P from an LTL specification ψ, we require ψ to hold in all the paths of P's computation tree. Consequently, we cannot impose possibility requirements on P (cf. [DTV99]). For example, while we can require that for every infinite sequence of input, the output signal v is eventually assigned true, we cannot require that every finite sequence of inputs can be extended so that v is eventually assigned true. In order to express possibility properties, we should specify P using *branching temporal logics*, which enable both universal and existential path quantification [Eme90]. Given a branching specification ψ over $I \cup O$, *realizability* of ψ is the problem of determining whether there exists a system P whose computation tree satisfies ψ. Correct synthesis of ψ then amounts to constructing such P.

So far, we considered the case where the specifications refer solely to signals in I and O, both are known to P. This is called synthesis with *complete information*. Often, the system does not have complete information about its environment. That is, there is a set E of signals that the environment generates but the system cannot read. Since P cannot read the signals in E, its activity is independent of them. Hence, it can still be viewed as a strategy $P : (2^I)^* \to 2^O$. Nevertheless, the computations of P are now infinite words

[1] All along this work, we consider synthesis with respect to *maximal environments*, which provide all possible input sequences.

over $2^{I \cup E \cup O}$. Similarly, embodying all the possible inputs to P, the computation tree induced by P now has a fixed branching degree $|2^{I \cup E}|$ and it is labeled by letters in $2^{I \cup E \cup O}$. Note that different nodes in this tree may have, according P's incomplete information, the same "history of inputs" (that is, when we project the labels along the paths from the root to these nodes on $2^{I \cup O}$, we get the same computation).

Often, systems need to satisfy specifications that refer to signals they cannot read. For example, in a distributed setting, each process is an open system and it can read only part of the signals generated by the other processes. Formally, given a specification ψ over the sets I, E, and O of readable input, unreadable input, and output signals, respectively, *synthesis with incomplete information* amounts to constructing a system $P : (2^I)^* \rightarrow 2^O$, which is independent of E, and which realizes ψ (that is, if ψ is linear then all the computations of P satisfy ψ, and if ψ is branching then the computation tree of P satisfies ψ). It is known how to cope with incomplete information in the linear paradigm. In particular, the approach used in [PR89] can be extended to handle synthesis with incomplete information for the linear specifications [KG95, KS95, Var95]. Coping with incomplete information is more difficult in the branching paradigm, where the methods used in the linear paradigm are not applicable [KV97].

In [KV97] we solved the problem of synthesis with incomplete information for specification in the branching temporal logic CTL* and its subset CTL. We proved that independently of the presence of incomplete information, the synthesis problems for CTL* and CTL are complete for 2EXPTIME and EXPTIME, respectively. These results joined the 2EXPTIME-complete bound for LTL synthesis in both settings [PR89, Ros92, Var95]. Keeping in mind that the satisfiability problems for LTL, CTL, and CTL* are complete for PSPACE, EXPTIME, and 2EXPTIME, respectively [Eme90], it follows that while the transition from closed to open systems dramatically increases the complexity of synthesis in the linear paradigm, it does not seem to influence the complexity in the branching paradigm. In both paradigms, incompleteness of the information does not make the synthesis problem more complex.

In this work, we consider the specification language *μ-calculus*. The μ-calculus is a propositional modal logic augmented with least and greatest fixpoint operators. It was introduced in [Koz83], following earlier studies of fixpoint calculi in the theory of program correctness [EC80, Par76, Pra81]. Over the past fifteen years, the μ-calculus has been established as essentially the "ultimate" program logic, as it expressively subsumes all propositional program logics, including dynamic logics such as PDL [FL79], process logics such as YAPL [VW84], and temporal logics such as LTL and CTL* [EH86]. The μ-calculus has gained further prominence with the discovery that its formulas can be evaluated symbolically in a natural way [BCM$^+$92], leading to industrial acceptance of computer-aided verification [BBG$^+$94]. More recently, the μ-calculus has found a new application domain in the theory of *description logics* in Artificial Intelligence [GL94].

As we explain below, the techniques developed in [KV97] do not extend in a straightforward manner to specification expressed in the μ-calculus. The solution described in [KV97] goes through alternating tree automata. In order to cope with incomplete information, we need to transform formulas to alternating tree automata that meet two structural restrictions: the automata are *ε-free* (i.e., they contain no ε-transitions and all the copies of the automaton that are generated during a transition are sent to successors of the node currently read in the input tree) and they are *symmetric* (i.e., the transition function of the automaton does not distinguish between different successors of a node in the input tree, and the automaton proceeds either existentially (a copy of the automaton is sent to some unspecified successor) or universally (copies of the automaton are sent to all successors)). Previous translations of μ-calculus to alternating tree automata result in automata that either contain ε-transitions or are not symmetric[2]. Maintaining both structural requirements involves alternating automata with quadratically many states and transitions of exponential size. The exponential blow-up is not a problem when the specification language is CTL* (there, it is absorbed in the overall

[2]The first translation in [EJ91] does not contain ε-transitions, but it implicitly assumes them (otherwise, it is incorrect). The translations in [BC96, Var98, GW99] have ε-transitions, and the one in [KVW00] is not symmetric.

doubly-exponential complexity), and it does not occur when the specification language is CTL (there, we know how to construct ε-free symmetric alternating automata with no exponential blow-up). As we explain below, for μ-calculus the challenge is to absorb this exponential blow-up in the exponential cost of the final algorithm, rather than have it blow-up the complexity by another exponential.

Technically, the synthesis problem for μ-calculus is reduced to checking the nonemptiness of an alternating automaton with polynomially many states, with respect to trees whose branching degree is exponential. We can translate the alternating automaton to a nondeterministic one with an exponential blow up [MS95]. Since the nonemptiness of this automaton has to be checked with respect to trees whose branching degree is exponential as well, its transition relation can be of a doubly-exponential size, resulting in a doubly-exponential complexity for the synthesis problem. Using the fact that parity games has memoryless strategies [Tho95], we are able to translate the alternating automaton to a deterministic one, running over trees annotated with winning strategies. Being deterministic, the transition relation of the automaton is of exponential size even when it runs on trees with an exponential branching degree. The key to achieving these complexity bounds is a novel, efficient encoding of winning strategies. We believe that our technique will be useful also in other contexts. For example, it implies a translation of alternating parity automata to nondeterministic ones that is exponential, independently of the branching degree of the underlying trees.

Using these automata-theoretic results, we obtain an exponential-time algorithm for the problem of μ-calculus synthesis with incomplete information, implying that the problem is EXPTIME-complete. The exact complexity of the algorithm is $2^{O(n^6)}$, where n is the length of the synthesized formula. This discouraging blow-up rarely appears in practice, and the problem of whether the exponent can be improved is left open. From a theoretical point of view, this result strengthen the observation about satisfiability and synthesis having the same complexity in the branching paradigm [EJ91, FL79]. In addition, using known translations of CTL and CTL* into the μ-calculus, all the results in [KV97] can be obtained as a special case of our result here.

2 Preliminaries

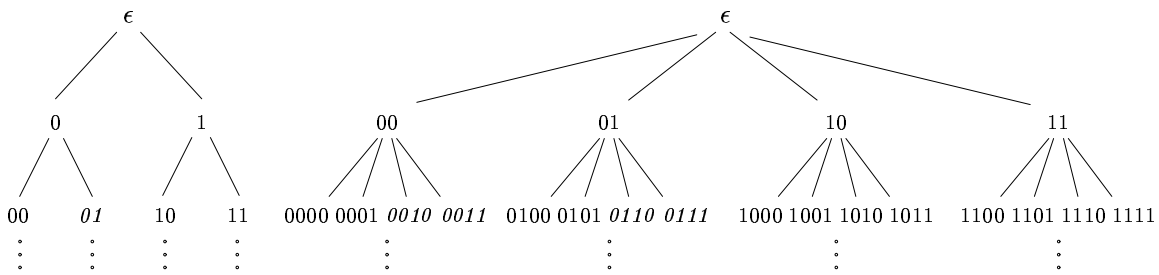
2.1 Trees and Labeled Trees

Given a finite set Υ, an Υ-*tree* is a set $T \subseteq \Upsilon^*$ such that if $x \cdot \upsilon \in T$, where $x \in \Upsilon^*$ and $\upsilon \in \Upsilon$, then also $x \in T$. When Υ is not important or 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 $x \in T$, the nodes $x \cdot \upsilon \in T$, for $\upsilon \in \Upsilon$, are the *children* of x. Each node x of T has a *direction* in Υ. The direction of the root is υ^0, for some designated $\upsilon^0 \in \Upsilon$, called the *root direction*. The direction of a node $x \cdot \upsilon$ is υ. We denote by $dir(x)$ the direction of node x. An Υ-tree T is a *full infinite tree* if $T = \Upsilon^*$. Unless otherwise mentioned, we consider here full infinite trees. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$ there exists a unique $\upsilon \in \Upsilon$ such that $x \cdot \upsilon \in \pi$.

Given two finite sets Υ and Σ, a Σ-*labeled* Υ-*tree* is a pair $\langle T, V \rangle$ where T is an Υ-tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ. When Υ and Σ are not important or clear from the context, we call $\langle T, V \rangle$ a labeled tree. For a Σ-labeled Υ-tree $\langle T, V \rangle$, we define the *x-ray* of $\langle T, V \rangle$, denoted $xray(\langle T, V \rangle)$, as the $(\Upsilon \times \Sigma)$-labeled Υ-tree $\langle T, V' \rangle$ in which each node is labeled by both its direction and its labeling in $\langle T, V \rangle$. Thus, for every $x \in T$, we have $V'(x) = \langle dir(x), V(x) \rangle$. Essentially, the labels in $xray(\langle T, V \rangle)$ contain information not only about the surface of $\langle T, V \rangle$ (its labels) but also about its skeleton (its nodes). We say that a $(\Upsilon \times \Sigma)$-labeled Υ-tree $\langle T, V \rangle$ is Υ-*exhaustive* if for every node $x \in T$, we have $V(x) \in dir(x) \times \Sigma$. Note that for every Σ-labeled Υ-tree $\langle T, V \rangle$, the tree $xray(\langle T, V \rangle)$ is Υ-exhaustive.

Let X and Y be finite sets. Consider the tree $(X \times Y)^*$. We define a function $hide_Y : (X \times Y)^* \to X^*$.

Given a node $w \in (X \times Y)^*$, the node $hide_Y(w) \in X^*$ is obtained from w by replacing each letter $\langle x, y \rangle$ in w by the letter x. For example, if $X = Y = \{0, 1\}$ (see figure next page), then the node 0010 of the 4-ary tree in the figure corresponds, by $hide_Y$, to the node 01 of the binary tree. Note that the nodes $0011, 0110$, and 0111 of the 4-ary tree also correspond, by $hide$, to the node 01 of the binary tree. We extend the hiding operator to paths $\pi \subset (X \times Y)^*$ in the straightforward way. That is, the path $hide_Y(\pi) \subset X^*$ is obtained from π by replacing each node $w \in \pi$ by the node $hide_Y(w)$.



Let Z be a finite set. For a Z-labeled X-tree $\langle X^*, V \rangle$, we define the *Y-widening* of $\langle X^*, V \rangle$, denoted $wide_Y(\langle X^*, V \rangle)$, as the Z-labeled $(X \times Y)$-tree $\langle (X \times Y)^*, V' \rangle$ where for every node $w \in (X \times Y)^*$, we have $V'(w) = V(hide_Y(w))$. Note that for every node $w \in (X \times Y)^*$ and $x \in X$, the children $w \cdot (x, y)$, for all $y \in Y$, agree on their label in $\langle (X \times Y)^*, V' \rangle$. Indeed, they are all labeled with $V(hide_Y(w) \cdot x)$. The essence of widening is that for every path π in $\langle X^*, V \rangle$ and for every path $\rho \in hide_Y^{-1}(\pi)$, the path ρ exists in $\langle (X \times Y)^*, V' \rangle$ and $V(\pi) = V'(\rho)$. In other words, for every two paths ρ_1 and ρ_2 in $\langle (X \times Y)^*, V' \rangle$ such that $hide_Y(\rho_1) = hide_Y(\rho_2) = \pi$, we have $V'(\rho_1) = V'(\rho_2) = V(\pi)$.

2.2 Symmetric automata and the propositional μ-calculus

Let $\Omega = \{\varepsilon, \Box, \Diamond\}$, and let $\mathcal{B}^+(\Omega \times Q)$ be the set of positive Boolean formulas over $\Omega \times Q$; i.e., Boolean formulas built from elements in $\Omega \times Q$ using $\wedge$ and $\vee$, where we also allow the formulas **true** and **false** and, as usual, $\wedge$ has precedence over $\vee$. For a set $S \subseteq \Omega \times Q$ and a formula $\theta \in \mathcal{B}^+(\Omega \times Q)$, we say that S *satisfies* θ iff assigning **true** to elements in S and assigning **false** to elements in $(\Omega \times Q) \setminus S$ makes θ true.

Consider a set $\Upsilon = \{\upsilon_1, \ldots, \upsilon_n\}$ of directions. In a nondeterministic automaton $\mathcal{A}$, over labeled Υ-trees, with a set Q of states, the transition function δ maps an automaton state $q \in Q$ and an input letter $\sigma \in \Sigma$ to a set of n-tuples of states. Each n-tuple suggests a nondeterministic choice for the automaton's next configuration. When the automaton is in a state q and is reading a node x labeled by a letter σ, it proceeds by first choosing a tuple $\langle q_1, \ldots, q_n \rangle \in \delta(q, \sigma)$ and then splitting into n copies, where copy i enters the state q_i and proceeds to the node $x \cdot \upsilon_i$. A *symmetric automaton* [JW95, Wil99] is an alternating tree automaton in which the transition function δ maps q and σ to a formula in $\mathcal{B}^+(\Omega \times Q)$. Atoms of the form $\langle \varepsilon, q \rangle$ are called *ε-transitions*. Intuitively, an atom $\langle \varepsilon, q \rangle$ corresponds to a copy of the automaton in state q sent to the current node of the input tree. An atom $\langle \Box, q \rangle$ corresponds to n copies of the automaton in state q, sent to all the successors of the current node. An atom $\langle \Diamond, q \rangle$ corresponds to a copy of the automaton in state q, sent to some successor of the current node. When, for instance, the automaton is in state q, reads a node x, and $\delta(q, V(x)) = (\Box, q_1) \wedge (\varepsilon, q_2) \vee (\Diamond, q_2) \wedge (\Diamond, q_3)$, it can either send n copies in state q_1 to the nodes $x \cdot \upsilon_1, \ldots, x \cdot \upsilon_n$ and send a copy in state q_2 to x, or send one copy in state q_2 to some node in $x \cdot \upsilon_1, \ldots, x \cdot \upsilon_n$ and send one copy in state q_3 to some node in $x \cdot \upsilon_1, \ldots, x \cdot \upsilon_n$. So, while nondeterministic tree automata send exactly one copy to each successor, symmetric automata can send several copies to the same successor, and can also have ε-transitions. On the other hand, symmetric automata cannot distinguish between left and right and can send copies to successor nodes only in either a universal or an existential manner.

Formally, a symmetric automaton is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$ where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(\Omega \times Q)$ is a transition function, $q_0 \in Q$ is an initial state, and α specifies the acceptance condition (a condition that defines a subset of Q^ω). The automaton $\mathcal{A}$ is *ε-free* iff δ contains no ε-transitions. A *run* of a symmetric automaton $\mathcal{A}$ on an input labeled Υ-tree $\langle T, V \rangle$ is a tree $\langle T_r, r \rangle$ (to be formally defined shortly) in which each node is labeled by an element of $\Upsilon^* \times Q$. Unlike T, in which each node has exactly $|\Upsilon|$ children, the tree T_r may have nodes with many children and may also have *leaves* (nodes with no children). Thus, $T_r \subset \mathbb{N}^*$ and a path in T_r may be either finite, in which case it ends in a leaf, or infinite. Each node of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T; in contrast, in a run of a nondeterministic automaton on $\langle T, V \rangle$ there is a one-to-one correspondence between the nodes of the run and the nodes of the tree. The labels of a node and its children have to satisfy the transition function. Formally, the run $\langle T_r, r \rangle$ is an $(\Upsilon^* \times Q)$-labeled tree that satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_0)$.
2. Let $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$. Then there is a (possibly empty) set $S \subseteq \Omega \times Q$, such that S satisfies θ, and for all $(c, s) \in S$, the following hold:
   - If $c = \varepsilon$, then there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x, s)$.
   - If $s = \Box$, then for each $v \in \Upsilon$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot v, s)$.
   - If $s = \Diamond$, then for some $v \in \Upsilon$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot v, s)$.

For example, if $\langle T, V \rangle$ is a $\{1, 2\}$-tree with $V(\varepsilon) = a$ and $\delta(q_0, a) = \Diamond q_1 \wedge \Box q_2$, then the nodes of $\langle T_r, r \rangle$ at level 1 include one of the labels $(1, q_1)$ or $(2, q_1)$, and include both labels $(1, q_2)$ and $(2, q_2)$. Note that if $\theta =$ **true**, then y need not have children. This is the reason why T_r may have leaves. Also, since there exists no set S as required for $\theta =$ **false**, we cannot have a run that takes a transition with $\theta =$ **false**. Each infinite path ρ in $\langle T_r, r \rangle$ is labeled by a word in Q^ω. Let $inf(\rho)$ denote the set of states in Q that appear in $r(\rho)$ infinitely often. A run $\langle T_r, r \rangle$ is accepting iff all its infinite paths satisfy the acceptance condition. In *parity* automata, α is a partition $\{F_1, F_2, \ldots, F_k\}$ of Q and an infinite path ρ satisfies α iff the minimal index i for which $inf(\rho) \cap F_i \neq \emptyset$ is even. *Co-parity* automata are similar, only that the minimal index i for which $inf(\rho) \cap F_i \neq \emptyset$ is odd. The number k is called the *index* of the automaton. A *safety* automaton is a parity automaton with $\alpha = \{\emptyset, Q\}$. Thus, all the words in Q^ω satisfy α, and we do not specify it.

An automaton accepts a tree iff there exists an accepting run on it. We denote by $\mathcal{L}(\mathcal{A})$ the language of the automaton $\mathcal{A}$; i.e., the set of all labeled trees that $\mathcal{A}$ accepts. We say that $\mathcal{A}$ is *nonempty* iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$. We denote by $\mathcal{A}^q$ the automaton obtained from $\mathcal{A}$ by making q the initial state.

The *propositional μ-calculus* is a propositional modal logic augmented with least and greatest fixpoint operators [Koz83]. For example, the ν-calculus formula $\nu y.p \wedge AXy$, which specifies *invariance*, states that the proposition p holds in all reachable nodes. The formula $\nu y.p \wedge AXAXy$ asserts that the proposition p holds in all the nodes that are reachable with an even number of transitions. Note that while the first formula is expressible by the CTL formula AGp, the second formula is not expressible even in CTL* [Wol81]. In Appendix A, we define the syntax and semantics of the logic in detail, and we also describe a translation of μ-calculus formulas to alternating parity tree automata:

Theorem 2.1 [KVW00] *Given a μ-calculus formula ψ, we can construct a symmetric parity automaton $\mathcal{A}_\psi$ such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of trees satisfying ψ. The automaton $\mathcal{A}_\psi$ has $|\psi|$ states and index at most $|\psi|$.*

Theorem 2.1 enables us to solve the problem of μ-calculus synthesis by means of symmetric parity automata (that is, assuming that the specification is given as a symmetric parity automaton)[3].

3 The Problem

Consider a system P that interacts with its environment. Let I and E be the sets of input signals, readable and unreadable by P, respectively, and let O be the set of P's output signals. We can view P as a *strategy* $P : (2^I)^* \to 2^O$. Indeed, P maps each finite sequence of sets of readable input signals into a set of output signals. Note that the information available to P regarding its environment is incomplete and P does not depend on the unreadable signals in E. We assume the following interaction between P and its environment. The interaction starts by P outputting $P(\varepsilon)$. The environment replies with some $\langle i_1, e_1\rangle \in 2^I \times 2^E$, to which P replies with $P(i_1)$. Interaction then continues step by step, with an output $P(i_1 \cdot i_2 \cdots i_j)$ corresponding to a sequence $\langle i_1, e_1\rangle \cdot \langle i_2, e_2\rangle \cdots \langle i_j, e_j\rangle$ of inputs. Thus, P associates with each infinite input sequence $\langle i_1, e_1\rangle \cdot \langle i_2, e_2\rangle \cdots$, an infinite computation $[P(\varepsilon)] \cdot [i_1 \cup e_1 \cup P(i_1)] \cdot [i_2 \cup e_2 \cup P(i_1 \cdot i_2)] \cdots$ over $2^{I \cup E \cup O}$. We note that our choice of P starting the interaction is for technical convenience only.

Though the system P is deterministic, it induces a *computation tree*. The branches of the tree correspond to external nondeterminism, caused by different possible inputs. Thus, $tree(P)$ is a $2^{I \cup E \cup O}$-labeled $2^{I \cup E}$-tree, where each node with direction $i \cup e \in 2^I \cup 2^E$ is labeled by $i \cup e \cup o$, where o is the set of output signals that P assigns to the sequence of readable inputs leading to the node. Formally, we obtain the computation tree $tree(P)$ by two transformations on the 2^O-labeled tree $\langle (2^I)^*, P\rangle$, which represents P. First, while $\langle (2^I)^*, P\rangle$ ignores the signals in E and the extra external nondeterminism induced by them, the computation tree of P, which embodies all possible computations, takes them into account. For that, we define the 2^O-labeled tree $\langle (2^{I \cup E})^*, P'\rangle = wide_{(2^E)}(\langle (2^I)^*, P\rangle)$. By the definition of the $wide$ operator, each two nodes in $(2^{I \cup E})^*$ that correspond, according to P's incomplete information, to the same input sequence are labeled by P' with the same output. Now, as the signals in I and E are represented in $\langle (2^{I \cup E})^*, P'\rangle$ only in its nodes and not in its labels, we define the computation tree $tree(P)$ as $\langle (2^{I \cup E})^*, P''\rangle = xray(\langle (2^{I \cup E})^*, P'\rangle)$. Note that, as I, E, and O are disjoint, we refer to $wide_{(2^E)}(\langle (2^I)^*, P\rangle)$ as a $2^{I \cup E}$-tree, rather than a $(2^I \times 2^E)$-tree. Similarly, $xray(\langle (2^{I \cup E})^*, P'\rangle)$ is a $2^{I \cup E \cup O}$-labeled tree, rather than a $(2^{I \cup E} \times 2^O)$-labeled tree.

Given a μ-calculus formula ψ over the sets $I \cup E \cup O$ of atomic propositions, the problem of *realizability with incomplete information* is to determine whether there is a system P whose computation tree $tree(P)$ satisfies ψ. The *synthesis problem* requires the construction of such P.

4 Constructions

Often, it is convenient to specify properties with automata containing ε-transitions. In particular, the translation of μ-calculus to symmetric automata results in automata that contain ε-transitions. As we shall see, for synthesis with incomplete information, we need automata that are both symmetric and ε-free. This is taken care of in the following theorem.

Theorem 4.1 [Var98, Wil99] *Given a symmetric parity automaton $\mathcal{A}$, we can construct an equivalent ε-free symmetric parity automaton $\mathcal{A}'$. If $\mathcal{A}$ has n states and index k, then $\mathcal{A}'$ has $O(n^2)$ state and index $O(k)$.*

[3] This means that readers not familiar with μ-calculus can continue to read the paper assuming it studies synthesis of specifications given by symmetric parity automata, keeping in mind that natural applications would start with a μ-calculus formula, automatically transformed to such an automaton.

Note that while Theorem 4.1 provides a quadratic upper bound on the number of states in $\mathcal{A}'$, it does not provide a polynomial bound on the size of $\mathcal{A}'$. In particular, the length of the transitions of $\mathcal{A}'$ could be exponential in n.

Recall that the operator $wide_Y$ maps a Z-labeled X-tree $\langle X^*, V\rangle$ to a Z-labeled $(X \times Y)$-tree $\langle (X \times Y)^*, V'\rangle$ such that for every node $w \in (X \times Y)^*$, we have $V'(w) = V(hide_Y(w))$. We define a variant of the operator $wide_Y$, called fat_Y. Given a Z-labeled X-tree $\langle X^*, V\rangle$, the operator fat_Y maps $\langle X^*, V\rangle$ into a set of $(Z \times Y)$-labeled $(X \times Y)$-trees such that $\langle (X \times Y)^*, V'\rangle \in fat_Y(\langle X^*, V\rangle)$ iff the following hold.

1. $V'(\varepsilon) \in \{V(\varepsilon)\} \times Y$, and
2. for every $w \in (X \times Y)^+$ with $dir(w) \in X \times \{y\}$, we have $V'(w) = \langle V(hide_Y(w)), y\rangle$.

That is, $fat_Y(\langle X^*, V\rangle)$ contains $|Y|$ trees, which differ only on the label of their roots. The trees in $fat_Y(\langle X^*, V\rangle)$ are very similar to the tree $wide_Y(\langle X^*, V\rangle)$, with each node labeled, in addition to its label in $wide_Y(\langle X^*, V\rangle)$, also with the Y-element of its direction. An exception is the root, which is labeled, in addition to its label in $wide_Y(\langle X^*, V\rangle)$, also with some element of Y. Among all the trees in $fat_Y(\langle X^*, V\rangle)$, of special interest to us is the tree with root labeled with $\langle V(\varepsilon), y^0\rangle$, where y^0 is the root direction of Y. We call this tree $wide'_Y(\langle X^*, V\rangle)$.

Theorem 4.2 *Let X, Y, and Z be finite sets. Given an ε-free symmetric parity automaton $\mathcal{A}$, over $(Z \times Y)$-labeled $(X \times Y)$-trees, we can construct an ε-free symmetric parity automaton $\mathcal{A}'$ over Z-labeled X-trees such that the following hold.*

1. *$\mathcal{A}'$ accepts a Z-labeled tree $\langle X^*, V\rangle$ iff $\mathcal{A}$ accepts the $(Z \times Y)$-labeled tree $wide'_Y(\langle X^*, V\rangle)$.*
2. *If $\mathcal{A}$ has n states and index k, then $\mathcal{A}'$ has $O(n)$ states and index k.*

Proof: Let $\mathcal{A} = \langle Z \times Y, Q, \delta, q_0, \alpha\rangle$. Then, $\mathcal{A}' = \langle Z, Q', \delta', q_0, \alpha'\rangle$, where

- $Q' = \{q_0\} \cup (Q \times \{\forall, \exists\})$. Typically, when the automaton is in state $\langle q, \forall\rangle$, it accepts a tree $\langle T, V\rangle$ if all the trees in $fat_Y(\langle T, V\rangle)$ are accepted by $\mathcal{A}^q$. When the automaton is in state $\langle q, \exists\rangle$, it accepts all trees $\langle T, V\rangle$ for which there exists a tree in $fat_Y(\langle T, V\rangle)$ that is accepted by $\mathcal{A}^q$. We call $\forall$ and $\exists$ the *mode* of the state.
- In order to define the transition function δ', we first define the function $adjust : \mathcal{B}^+(Q) \to \mathcal{B}^+(Q')$. Given a formula θ in $\mathcal{B}^+(Q)$, the formula $adjust(\theta)$ is obtained from θ by replacing an atom $\Box q$ by the atom $\Box\langle q, \forall\rangle$ and replacing an atom $\Diamond q$ by the atom $\Diamond\langle q, \exists\rangle$. The transition function $\delta' : Q' \times Z \to \mathcal{B}^+(Q')$ is now defined, for all $q \in Q$ and $z \in Z$ as follows.

$$\delta'(\langle q, \forall\rangle, z) = \bigwedge_{y \in Y} adjust(\delta(q, \langle z, y\rangle)),$$

$$\delta'(\langle q, \exists\rangle, z) = \bigvee_{y \in Y} adjust(\delta(q, \langle z, y\rangle)).$$

  Also, $\delta'(q_0, z) = adjust(\delta(q_0, \langle z, y^0\rangle))$, where y^0 is the root direction of Y. Thus, while from the state $\langle q, \forall\rangle$ we proceed with all possible labeling of roots of trees in $fat_Y(\langle T, V\rangle)$, from the state $\langle q, \forall\rangle$ we guess such a labeling, and from q_0 we take the labeling y^0.

- α' is obtained from α by replacing each set F participating in α by the set $F \times \{\forall, \exists\}$.

□

Note that the automata $\mathcal{A}$ and $\mathcal{A}'$ are symmetric, thus the reference to X-trees and $(X \times Y)$-trees is not reflected in the construction, and is given in the theorem only for its intended application in Section 5. Note also that the ε-freeness of $\mathcal{A}$ is crucial. To see this, consider a symmetric automaton $\mathcal{A}$ for the formula $EX(\varphi_1 \wedge \varphi_2)$. From its initial state, the automaton guesses a direction υ to which it sends a copy in a state q that accepts a tree if it satisfies both φ_1 and φ_2. The automaton $\mathcal{A}'$ should then guess the hidden information (letter in Y) that labels the root of such a tree. For that, we associate with q an existential mode, and proceed to a disjunction of the letters in Y. Suppose that $\mathcal{A}$ does have ε-transitions, say $\delta(q,\sigma) = (\varepsilon,\varphi_1) \wedge (\varepsilon,\varphi_2)$. Then, $\mathcal{A}$ creates two copies that reads the node υ. There is no way to ensure that $\mathcal{A}'$ would guess the same hidden information for the two copies.

Consider an ε-free symmetric automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$. Recall that the transition function δ maps a state and a letter to a formula in $\mathcal{B}^+(\{\Box,\Diamond\} \times Q)$. For a set Υ of directions, let $\Psi_\Upsilon = (\{\Box\} \cup \Upsilon) \times Q$. A subset H of Ψ_Υ can be viewed as an "detailed" description of a set of atoms in $\{\Box,\Diamond\} \times Q$, where instead an atom $(\Diamond, q)$, the set H contains an atom (υ, q), where $\upsilon \in \Upsilon$ is the direction in which the existential requirement $(\Diamond, q)$ is going to be satisfied. We say that H satisfies a formula θ in $\mathcal{B}^+(\{\Box,\Diamond\} \times Q)$ if the set obtained from H by replacing all the atoms in $\Upsilon \times Q$ by the corresponding atoms in $\{\Diamond\} \times Q$ satisfies θ. A *restriction* of δ is a function $\delta' : Q \times \Sigma \to 2^{\Psi_\Upsilon}$ such that for all $q \in Q$ and $\sigma \in \Sigma$ for which $\delta(q,\sigma)$ is satisfiable (i.e., $\delta(q,\sigma)$ is not **false**), the set $\delta'(q,\sigma)$ satisfies $\delta(q,\sigma)$. If $\delta(q,\sigma)$ is not satisfiable, then $\delta'(q,\sigma)$ is undefined. If, for example, $\Upsilon = \{1,2\}$ and $\delta(q,\sigma) = \Box q_1 \vee \Diamond q_2$, then possible $\delta'(q,\sigma)$ are $\{(\Box, q_1)\}$, $\{(1, q_2)\}$, or $\{(2, q_2)\}$. Intuitively, by choosing both the atoms that are going to be satisfied and the directions in which the existential requirements are going to be satisfied, δ' removes all the nondeterminism in δ. Let F_δ be the set of restrictions of δ.

Consider a Σ-labeled tree $\langle \Upsilon^*, V \rangle$. A *running strategy* of $\mathcal{A}$ for $\langle \Upsilon^*, V \rangle$ is an F_δ-labeled tree $\langle \Upsilon^*, f \rangle$. The running strategy $\langle \Upsilon^*, f \rangle$ induces a single run $\langle T_r, r_f \rangle$ of $\mathcal{A}$ on $\langle \Upsilon^*, V \rangle$. Intuitively, whenever the run $\langle T_r, r_f \rangle$ is in state q as it reads a node $x \in \Upsilon^*$, it proceeds according to $f(x)(q, V(x))$. Formally, $\langle T_r, r_f \rangle$ is the $(\Upsilon^* \times Q)$-labeled tree that satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_0)$.
2. Consider a node $y \in T_r$ with $r(y) = (x, q)$. Then, $f(x)(q, V(x))$ is defined and for all $(c, s) \in f(x)(q, V(x))$, the following hold:
   - If $c = \Box$, then for each $\upsilon \in \Upsilon$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot \upsilon, s)$.
   - If $c \in \Upsilon$, then there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot c, s)$.

For a node $x \in \Upsilon^*$ and a state $q \in Q$, we say that x is *obliged to* q by f and V if $x = \varepsilon$ and $q = q_0$, or $x = y \cdot \upsilon$, for $y \in \Upsilon^*$ and $\upsilon \in \Upsilon$, and there is a state q' such that y is obliged to q' and $f(y)(q', V(y))$ contains $(\Box, q)$ or (υ, q). Thus, x is obliged to q if x is visited by q in the run $\langle T_r, r_f \rangle$.

A *will* for the automaton $\mathcal{A}$ is a function $\rho : Q \to 2^Q$. Let G_Q be the set of all wills. A *promise* of $\mathcal{A}$ for a Σ-labeled tree $\langle \Upsilon^*, V \rangle$ is a G_Q-labeled tree $\langle \Upsilon^*, g \rangle$. Intuitively, the promise $\langle \Upsilon^*, g \rangle$ corresponds to a run of $\mathcal{A}$ on $\langle \Upsilon^*, V \rangle$ in which for all nodes $y \cdot \upsilon$ and states $q \in Q$, if y is visited by state q, then $y \cdot \upsilon$ is visited by all the states in $g(y \cdot \upsilon)(q)$. For an infinite sequence $\rho_0, \rho_1, \ldots$ of wills for $\mathcal{A}$ and sequence (either finite or infinite) $q_0, q_1, \ldots$ of states, we say that $q_0, q_1, \ldots$ is a *trace* induced by $\rho_0, \rho_1, \ldots$ if q_0 is the initial state of $\mathcal{A}$ and for every $i \geq 0$, either $\rho_{i+1}(q_i)$ is empty, in which case q_i is the last state in the trace, or $\rho_{i+1}(q_i)$ is not empty, in which case q_{i+1} belongs to $\rho_{i+1}(q_i)$. Note that the trace is independent of ρ_0. We say that

a promise $\langle \Upsilon^*, g\rangle$ is *good* for $\mathcal{A}$ if all the infinite traces induced by paths in $\langle \Upsilon^*, g\rangle$ satisfy the acceptance condition α.

Consider a Σ-labeled tree $\langle \Upsilon^*, V\rangle$, a running strategy $\langle \Upsilon^*, f\rangle$, and a promise $\langle \Upsilon^*, g\rangle$. We say that g *fulfills* f for V if the states promised to be visited by g satisfy the obligations induced by f as it runs on V. Formally, g fulfills f for V if for every node $x \in \Upsilon^*$ and state q such that x is obliged to q by f and V, the following hold:

1. For every atom $(\Box, s) \in f(x)(q, V(x))$, all the successors $x \cdot \upsilon$ of x have $s \in g(x \cdot \upsilon)(q)$.
2. For every atom $(\upsilon, s) \in f(x)(q, V(x))$, the successor $x \cdot \upsilon$ of x has $s \in g(x \cdot \upsilon)(q)$.

Theorem 4.3 *$\mathcal{A}$ accepts $\langle \Upsilon^*, V\rangle$ iff there exist a running strategy $\langle \Upsilon^*, f\rangle$ and a promise $\langle \Upsilon^*, g\rangle$ such that $\langle \Upsilon^*, g\rangle$ is good for $\mathcal{A}$ and g fulfills f for V.*

Annotating input trees with restrictions and wills enables us to transform a symmetric automaton to a deterministic one, with an exponential blow up:

Theorem 4.4 *Consider an ε-free symmetric parity automaton $\mathcal{A}$ such that $\mathcal{A}$ runs on Σ-labeled Υ-trees. There is a deterministic parity tree automaton $\mathcal{A}'$ such that $\mathcal{A}'$ runs on $(\Sigma \times F_\delta \times G_Q)$-labeled Υ-trees and the following hold:*

1. *$\mathcal{A}'$ accepts a tree iff $\mathcal{A}$ accepts its projection on Σ.*
2. *If $\mathcal{A}$ has n states and index k, then $\mathcal{A}'$ has $2^{n(n+k\log nk)}$ states and index nk.*

Proof: Let $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha\rangle$, and let $\Sigma' = \Sigma \times F_\delta \times G_Q$. We refer to a Σ'-labeled Υ-tree as $\langle \Upsilon^*, (V, f, g)\rangle$, where V, f, and g are the projections of the tree on Σ, F_δ, and G_Q, respectively.

The automaton $\mathcal{A}'$ is the intersection of two deterministic automata $\mathcal{A}'_1$ and $\mathcal{A}'_2$. The automaton $\mathcal{A}'_1$ accepts a tree $\langle \Upsilon^*, (V, f, g)\rangle$ iff $\langle \Upsilon^*, g\rangle$ is a good promise for $\mathcal{A}$. The automaton $\mathcal{A}'_2$ accepts a tree $\langle \Upsilon^*, (V, f, g)\rangle$ iff g fulfills f for V. By Theorem 4.3, it follows that $\mathcal{A}'$ accepts $\langle \Upsilon^*, (V, f, g)\rangle$ iff $\mathcal{A}$ accepts $\langle \Upsilon^*, V\rangle$.

In order to define $\mathcal{A}'_1$, we first define a nondeterministic co-parity word automaton $\mathcal{U}$ over the alphabet G_Q such that $\mathcal{U}$ accepts a word if some traces it induces is infinite and violates the acceptance condition α. Thus, $\mathcal{U} = \langle G_Q, Q \cup \{q'_0\}, \eta, q'_0, \alpha\rangle$, where $\eta(q'_0, \rho) = q_0$ and for all $q \in Q$, we have $\eta(q, \rho) = \rho(q)$. Note that $\mathcal{U}$ ignores the first letter in the input, as traces induced by a sequence of wills are independent of the first will. In order to get $\mathcal{A}'_1$, we co-determinize $\mathcal{U}$ and expand it to a tree automaton on Σ'. That is, we first construct a deterministic parity automaton $\tilde{\mathcal{U}}$ that complements $\mathcal{U}$, and then replace a transition $\tilde{\eta}(s, \rho) = s'$ in $\tilde{\mathcal{U}}$ by transitions $\xi(s, (\sigma, \delta', \rho)) = \langle s', \ldots, s'\rangle$, for all $\sigma \in \Sigma$ and $\delta' \in F_\delta$, in $\mathcal{A}'_1$. By [Saf89, Tho97], the automaton $\tilde{\mathcal{U}}$ has $(nk)^{nk}$ states and index nk, thus so does $\mathcal{A}'_1$.

The automaton $\mathcal{A}'_2 = \langle \Sigma', \{q_0\} \cup G_Q, \eta, \rho_0\rangle$ is a deterministic safety tree automaton defined as follows. For a promise $\rho \in G_Q$, let $all(\rho) = \bigcup_{q \in Q} \rho(q)$. For two promises ρ and ρ', we denote by $\rho \subseteq \rho'$ the fact that for all $q \in Q$, we have $\rho(q) \subseteq \rho'(q)$. Intuitively, when $\mathcal{A}'_2$ visits a node x in state $\rho : Q \to 2^Q$, it means that x is obliged to all the states in $all(\rho)$, and for all $q \in Q$, the state q is the last state visited by the copy of $\mathcal{A}$ that made x obliged to the states in $\rho(q)$. For a promise ρ, a restriction δ', a letter $\sigma \in \Sigma$, and a direction $\upsilon \in \Upsilon$, we define $new(\rho, \delta', \sigma, \upsilon)$ as the promise ρ' where for all $q \in all(\rho)$, the set $\rho'(q)$ contains all states q' for which $(\Box, q')$ or (υ, q') is in $\delta'(q, \sigma)$. Let $\Upsilon = \{\upsilon_1, \ldots, \upsilon_m\}$. For every $\rho \in G_Q$ and $(\sigma, \delta', \rho') \in \Sigma'$, we define

$$\eta(\rho, (\sigma, \delta', \rho')) = \begin{cases} \langle new(\rho, \delta', \sigma, \upsilon_1), \ldots, new(\rho, \delta', \sigma, \upsilon_m)\rangle & \text{If } \rho \subseteq \rho' \\ \textbf{false} & \text{Otherwise.} \end{cases}$$

In addition, we define $\eta(q_0, (\sigma, \delta', \rho)) = \langle new(\rho_0, \delta', \sigma, \upsilon_1), \ldots, new(\rho_0, \delta', \sigma, \upsilon_m)\rangle$, where ρ_0 is such that $\rho_0(q) = \{q_0\}$ for all $q \in Q$. Thus, whenever the automaton $\mathcal{A}'_2$ visits a node x in a state ρ, it checks that the promise $g(x)$ covers all the obligations of x, and it learns from $f(x)$ new obligations to be delivered to the successors. In the initial state, there are no obligations to check.

Now, since $\mathcal{A}'_2$ is a safety automaton, taking its product with $\mathcal{A}'_1$ is trivial; thus $\mathcal{A}'$ has $2^{n(n+k\log nk)}$ states and index nk. □

We call $\mathcal{A}'$ the *witnessing* automaton for $\mathcal{A}$. Finally, we need the following construction, which checks an $(\Upsilon \times \Sigma)$-labeled Υ-tree for being Υ-*exhaustive*.

Theorem 4.5 [KV97] *Given finite sets Υ and Σ, there is a deterministic safety tree automaton $\mathcal{A}_{exh}$ on $(\Upsilon \times \Sigma)$-labeled Υ-trees, with $|\Upsilon|$ states, such that $\mathcal{L}(\mathcal{A}_{exh})$ is exactly the set of Υ-exhaustive trees. All the states of $\mathcal{A}_{exh}$ are accepting.*

5 μ-calculus Synthesis

Theorem 5.1 *The synthesis problem for μ-calculus, with either complete or incomplete information, is EXPTIME-complete.*

Proof: Given a specification ψ over $I \cup E \cup O$, we proceed as follows. Let $|\psi| = n$.

1. Construct a symmetric parity automaton $\mathcal{A}_\psi$, such that $\mathcal{A}_\psi$ runs on $2^{I\cup E\cup O}$-labeled $2^{I\cup E}$-trees, and it accepts all trees that satisfy ψ. The automaton $\mathcal{A}_\psi$ has n states and index n. [Theorem 2.1].

2. Construct an ε-free symmetric parity automaton $\mathcal{A}'_\psi$ equivalent to $\mathcal{A}_\psi$. The automaton $\mathcal{A}'_\psi$ has $O(n^2)$ states and index $O(n)$. [Theorem 4.1].

3. Construct an ε-free symmetric parity automaton $\mathcal{A}''_\psi$ that runs on $2^{I\cup O}$-labeled 2^I-trees and accepts $\langle (2^I)^*, V\rangle$ iff $\mathcal{A}'_\psi$ accepts $wide'_{2^E}(\langle (2^I)^*, V\rangle)$. The automaton $\mathcal{A}''_\psi$ has $O(n^2)$ states and index $O(n)$. [Theorem 4.2, with $X = 2^I$, $Y = 2^E$, and $Z = 2^{I\cup O}$].

   The specification ψ is realizable iff $\mathcal{L}(\mathcal{A}''_\psi)$ contains a 2^I-exhaustive tree.

4. Let $\mathcal{A}''_\psi = \langle 2^{I\cup O}, Q, \delta, q_0, \alpha\rangle$. Construct a deterministic parity tree automaton $\mathcal{N}_\psi$ that is a witnessing automaton for $\mathcal{A}''_\psi$. The automaton $\mathcal{N}_\psi$ runs on $(2^{I\cup O} \times F_\delta \times G_Q)$-labeled 2^I-trees, it has $2^{O(n^4)}$ states and index $O(n^2)$ [Theorem 4.4].

5. Construct a deterministic safety tree automaton $\mathcal{A}_{exh}$ with a trivial acceptance condition, that accepts a $(2^{I\cup O} \times F_\delta \times G_Q)$-labeled 2^I-tree iff it is 2^I-exhaustive. Since $|I| \leq |\psi|$, the automaton $\mathcal{A}_{exh}$ has at most 2^n states [Theorem 4.5].

6. Construct the product $\mathcal{A} = \mathcal{N}_\psi \times \mathcal{A}_{exh}$. Since $\mathcal{A}_{exh}$ is a safety automaton, the automaton $\mathcal{A}$ is a deterministic parity tree automaton with $2^{O(n^4)}$ states and index $O(n^2)$. The fact that $\mathcal{A}$ is deterministic guarantees that the size of its transition table is at most exponential in n.

The specification ψ is realizable iff $\mathcal{A}$ is not empty. The nonemptiness problem for a deterministic parity automaton with n states and index k can be solved in time $O(n^k)$ [EJS93], which implies a $2^{O(n^6)}$ complexity for the realizability problem. The nonemptiness algorithm can be extended, within the same complexity bound, to generate a finite-state strategy whose tree is accepted by the automaton [Rab70], thus solving the synthesis problem. □

In order to understand the importance of our construction in Theorem 4.4, recall that the specification ψ is realizable iff the automaton $\mathcal{A}''_\psi$ accepts a 2^I-exhaustive tree. A straightforward approach to check the latter translates $\mathcal{A}''_\psi$ into a nondeterministic automaton, resulting in an automaton with exponentially many states. Since the nonemptiness of this automaton has to be checked with respect to trees whose branching degree is exponential as well, its transition relation can be of a doubly-exponential size, resulting in a doubly-exponential complexity for the synthesis problem. The construction described in Theorem 4.4 transforms $\mathcal{A}''_\psi$ into a deterministic automaton. As such, its transition relation is of exponential size even when it runs on trees with an exponential branching degree.

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.

[BB87] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 62–74. Springer-Verlag, 1987.

[BBG+94] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th Conference on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 182–193, Stanford, June 1994.

[BC96] G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In *Proc. 1996 Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

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

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

[Dil89] D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989.

[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.

[EC80] E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th Int'l Colloq. on Automata, Languages and Programming*, pages 169–181, 1980.

[EC82] E.A. Emerson and E.M. Clarke. Using branching time logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[EH86] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

[EJ91] E.A. Emerson and C. Jutla. Tree automata, Mu-calculus and determinacy. In *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, pages 368–377, San Juan, October 1991.

[EJS93] E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, Proc. 5th Int. Conference*, volume 697, pages 385–396, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

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

[FL79] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Sciences*, 18:194–211, 1979.

[GL94] G. De Giacomo and M. Lenzerini. Concept languages with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 411–415. John Wiley and Sons, 1994.

[GW99] E. Graedel and I. Walukiewicz. Guarded fixed point logic. In *Proc. 14th Symposium on Logic in Computer Science*, July 1999.

[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.

[JW95] D. Janin and I. Walukiewicz. Automata for the modal μ-calculus and related results. In *Proc. 20th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 552–562. Springer-Verlag, 1995.

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

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

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

[KV97] O. Kupferman and M.Y. Vardi. Synthesis with incomplete informatio. In *2nd International Conference on Temporal Logic*, pages 91–106, Manchester, July 1997.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2), March 2000.

[MS95] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

[MW80] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980.

[Par76] D. Park. Finiteness is μ-ineffable. *Theoretical Computer Science*, 3:173–181, 1976.

[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.

[Pra81] V.R. Pratt. A decidable μ-calculus: preliminary report. In *Proc. 22nd IEEE Symposium on Foundation of Computer Science*, pages 421–427, 1981.

[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.

[Ros92] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1992.

[Saf89] S. Safra. *Complexity of automata on infinite objects*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1989.

[Tho95] W. Thomas. On the synthesis of strategies in infinite games. In E.W. Mayr and C. Puech, editors, *Proc. 12th Symp. on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1995.

[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 Verifi cation, Proc. 7th Int'l Conf.*, volume 939 of *Lecture Notes in Computer Science*, pages 267–292. Springer-Verlag, Berlin, 1995.

[Var98] M.Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th International Coll. on Automata, Languages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer-Verlag, Berlin, July 1998.

[VW84] M.Y. Vardi and P. Wolper. Yet another process logic. In *Logics of Programs*, volume 164, pages 501–512. Lecture Notes in Computer Science, Springer-Verlag, 1984.

[Wil99] T. Wilke. CTL$^+$ is exponentially more succinct than CTL. In C. Pandu Ragan, V. Raman, and R. Ramanujam, editors, *Proc. 19th conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 110–121. Springer-Verlag, 1999.

[Wol81] P. Wolper. Temporal logic can be more expressive. In *Proc. 22nd IEEE Symposium on Foundations of Computer Science*, pages 340–348, Nashville, October 1981.

A Propositional μ-calculus

The *propositional μ-calculus* is a propositional modal logic augmented with least and greatest fi xpoint operators [Koz83]. Specifi cally, we consider a μ-calculus where formulas are constructed from Boolean propositions with Boolean connectives, the temporal operators EX and AX, as well as least (μ) and greatest (ν) fi xpoint operators. We describe here a positive normal form for μ-calculus. Given a set AP of atomic proposition constants and a set APV of atomic proposition variables, a μ-calculus formula (in a positive normal form) is either:

- **true**, **false**, p or $\neg p$ for all $p \in AP$;
- y for all $y \in APV$;
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are μ-calculus formulas;
- $AX\varphi$ or $EX\varphi$, where φ is a μ-calculus formula;
- $\mu y.f(y)$ or $\nu y.f(y)$, where $y \in APV$ and $f(y)$ is a μ-calculus formula containing y as a free variable.

A *sentence* is a formula that contains no free atomic proposition variables. We call AX and EX *next modalities*, and we call μ and ν *fixpoint operators* We say that a μ-calculus formula is a *μ-formula* (*ν-formula*), if it is of the form $\mu y.f(y)$ ($\nu y.f(y)$). We use λ to denote a fi xpoint modality μ or ν. For a λ-formula $\lambda y.f(y)$, the formula $f(\lambda y.f(y))$ is obtained from $f(y)$ by replacing each free occurrence of y with $\lambda y.f(y)$. The *closure* $cl(\varphi)$ of a μ-calculus sentence φ is the smallest set of μ-calculus that contains φ and is closed under subformulas (that is, if ψ is in the closure, so do all its subformulas that are sentences) and fi xpoint applications (that is, if $\lambda y.f(y)$ is in the closure, so does $f(\mu y.f(y))$). For example, for $\varphi = \mu y.(q \vee (p \wedge EXy))$, we have $cl(\varphi) = \{\varphi, q \vee (p \wedge EX\varphi), q, p \wedge EX\varphi, p, EX\varphi\}$. As proved in [Koz83], for every μ-calculus formula φ, the number of elements in $cl(\varphi)$ is linear in the length of φ. Accordingly, we defi ne size $|\varphi|$ of φ as the number of elements in $cl(\varphi)$. Note that even though the number of elements in the closure of a formula can be logarithmic in the length of the formula if there are multiple occurrences of identical subformulas, our defi nition of size is reasonable since it corresponds to the number of nodes in a reduced DAG representation of the formula.

A μ-calculus formula is *guarded* if for all $y \in APV$, all the occurrences of y that are in a scope of a fixpoint modality λ are also in a scope of a modality EX or AX that is itself in the scope of λ. Thus, a μ-calculus sentence is *guarded* if for all $y \in APV$, all the occurrences of y are in the scope of a next modality. For example, the formula $\mu y.(p \vee EXy)$ is guarded and the formula $EX\mu y.(p \vee y)$ is not guarded. Given a μ-calculus formula, we can construct, in linear time, an equivalent guarded formula [BB87, KVW00]. Accordingly, we assume that all formulas are guarded.

We define the semantics of μ-calculus formulas with respect to a 2^{AP}-labeled trees. Given a 2^{AP}-labeled tree $\langle T, V\rangle$ and a set $\{y_1, \ldots, y_n\}$ of free variables, a *valuation* $\mathcal{V} = \{\langle y_1, T_1\rangle, \ldots, \langle y_n, T_n\rangle\}$, with $T_i \subseteq T$, is an assignment of subsets of T to the variables $\{y_1, \ldots, y_n\}$. For a valuation $\mathcal{V}$, a variable y, and a set $T' \subseteq T$, we denote by $\mathcal{V}[y \leftarrow T']$ the valuation obtained from $\mathcal{V}$ by assigning T' to y. A formula φ with free variables $\{y_1, \ldots, y_n\}$ is interpreted as a mapping from valuations to subsets of T. Thus, $\varphi(\mathcal{V})$ denotes the set of nodes that satisfy φ with the valuation $\mathcal{V}$. The mapping φ is defined inductively as follows:

- $\mathbf{true}(\mathcal{V}) = T$ and $\mathbf{false}(\mathcal{V}) = \emptyset$.
- For an atomic proposition $p \in AP$, we have $p(\mathcal{V}) = \{x \in T : p \in V(x)\}$ and $(\neg p)(\mathcal{V}) = \{x \in T : p \notin V(x))\}$.
- For an atomic proposition variable $y_i \in APV$, we have $y_i(\mathcal{V}) = W_i$.
- $(\varphi_1 \wedge \varphi_2)(\mathcal{V}) = \varphi_1(\mathcal{V}) \cap \varphi_2(\mathcal{V})$.
- $(\varphi_1 \vee \varphi_2)(\mathcal{V}) = \varphi_1(\mathcal{V}) \cup \varphi_2(\mathcal{V})$.
- $(EX\varphi)(\mathcal{V}) = \{x \in T :$ there is a child y of x such that $y \in \varphi(\mathcal{V})\}$.
- $(AX\varphi)(\mathcal{V}) = \{x \in T :$ all the children y of x satisfy $y \in \varphi(\mathcal{V})\}$.
- $(\mu y.f(y))(\mathcal{V}) = \bigcap\{T' \subseteq T : f(\mathcal{V}[y \rightarrow T']) \subseteq T'\}$.
- $(\nu y.f(y))(\mathcal{V}) = \bigcup\{T' \subseteq T : T' \subseteq f(\mathcal{V}[y \rightarrow T'])\}$.

Note that no valuation is required for a sentence. For a node $x \in T$ and a sentence φ, we say that x satisfies φ iff $x \in \varphi(\emptyset)$. A tree $\langle T, V\rangle$ satisfies φ iff its root satisfies φ.

We now describe the translation of guarded μ-calculus formulas to alternating parity tree automata. The description is a variant of the translation described in [KVW00]. Here, we assume that the automaton has ε-transitions, circumventing the need for the function $split$ used there (and which causes the automaton not to be symmetric).

Theorem A.1 [KVW00] *Given a μ-calculus formula ψ, we can construct a symmetric parity automaton $\mathcal{A}_\psi$ such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of trees satisfying ψ. The automaton $\mathcal{A}_\psi$ has $|\psi|$ states and index $|\psi|$.*

Proof: For a μ-calculus sentence ψ and a subformula $\varphi = \lambda y.f(y)$ of ψ, we define the *alternation level* of φ in ψ, denoted $al_\psi(\varphi)$, as follows [BC96].

- If φ is a sentence, then $al_\psi(\varphi) = 1$.
- Otherwise, let $\xi = \lambda' x.g(x)$ be the innermost μ or ν subformula of ψ that has φ as a strict subformula. Then, if x is free in φ and $\lambda' \neq \lambda$, we have $al_\psi(\varphi) = al_\psi(\xi)+1$. Otherwise, we have $al_\psi(\varphi) = al_\psi(\xi)$.

Intuitively, the alternation level of φ in ψ is the number of alternating fixed-point operators we have to "wrap φ with" in order to reach a sub-sentence of ψ.

Given ψ, we define the parity automaton $\mathcal{A}_\psi = \langle 2^{AP}, cl(\psi), \delta, \psi, F\rangle$, where

- The transition function δ is such that whenever we visit a state associated with a conjunction or a disjunction, we first proceed with ε-transitions until we reach states of the form $p, \neg p, AX\varphi, EX\varphi, \mu y.f(y)$, or $\nu y.f(y)$. The ε-transitions guarantee that when the automaton is following a fixed-point formula φ, it keeps visiting the state φ itself, rather than a Boolean assertion that contains it. The fact that ψ is guarded guarantees that states of the form $y \in APV$ are not reachable.

  For all $\sigma \in 2^{AP}$, we define:

  - $\delta(p,\sigma) =$ **true** if $p \in \sigma$. — $\delta(p,\sigma) =$ **false** if $p \notin \sigma$.
  - $\delta(\neg p,\sigma) =$ **true** if $p \notin \sigma$. — $\delta(\neg p,\sigma) =$ **false** if $p \in \sigma$.
  - $\delta(\varphi_1 \wedge \varphi_2, \sigma) = (\varepsilon, \varphi_1) \wedge (\varepsilon, \varphi_2)$.
  - $\delta(\varphi_1 \vee \varphi_2, \sigma) = (\varepsilon, \varphi_1) \vee (\varepsilon, \varphi_2)$.
  - $\delta(AX\varphi, \sigma) = \Box\varphi$.
  - $\delta(EX\varphi, \sigma) = \Diamond\varphi$.
  - $\delta(\mu y.f(y), \sigma) = \delta(f(\mu y.f(y)), \sigma)$.
  - $\delta(\nu y.f(y), \sigma) = \delta(f(\nu y.f(y)), \sigma)$.

  Our transition relation is very similar to the one suggested in [EJ91], only it contains the ε-transitions implicitly assumed there, and explicitly given in the proof rules in [BC96].

- Let d be the maximal alternation level of subformulas of ψ. Denote by G_i the set of all the ν-formulas in $cl(\psi)$ of alternation level i. Denote by B_i the set of all μ-formulas in $cl(\psi)$ of alternation depth less than or equal to i. Now, let $F_0 = \emptyset$, and for every $1 \leq i \leq d$, let $F_{2i-1} = F_{2i-2} \cup B_i$ and $F_{2i} = F_{2i-1} \cup G_i$. It is easy to see that $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_{2d}$. We define $\alpha = \{F_1, F_2, \ldots, F_{2d}\}$ That is, if the automaton gets stuck in a cycle, it must visit some ν-formula infinitely often and can visit μ-formulas of smaller alternation levels only finitely often.

□