Synthesis for Probabilistic Environments*

Sven Schewe

Universität des Saarlandes, 66123 Saarbrücken, Germany
schewe@cs.uni-sb.de

Abstract. In synthesis we construct finite state systems from temporal specifications. While this problem is well understood in the classical setting of non-probabilistic synthesis, this paper suggests the novel approach of open synthesis under the assumptions of an environment that chooses its actions randomized rather than nondeterministically. Assuming a randomized environment inspires alternative semantics both for linear-time and branching-time logics. For linear-time, natural acceptance criteria are *almost-sure* and *observable* acceptance, where it suffices if the probability measure of accepting paths is 1 and greater than 0, respectively. We distinguish 0-environments, which can freely assign probabilities to each environment action, from ε-environments, where the probabilities assigned by the environment are bound from below by some $\varepsilon > 0$. While the results in case of 0-environments are essentially the same as for nondeterministic environments, the languages occurring in case of ε-environments are topologically different from the results for nondeterministic and 0-environments (in case of LTL, recognizable by weak alternating automata vs. recognizable by deterministic automata). The complexity of open synthesis is, in both cases, EXPTIME and 2EXPTIME-complete for CTL and LTL specifications, respectively.

1 Introduction

Among the most important developments in verification is the development of model-checking algorithms, which test whether or not a finite-state program satisfies a temporal specification. However, this method suffers from two significant drawbacks: First, it can only be applied *after* much effort has been invested to the (manual) construction of the system. And second, model-checking cannot distinguish unrealizable specifications from erroneous implementations. The natural approach to circumvent these drawbacks is to construct finite-state systems directly from the specification. Such an approach is called synthesis.

Early works consider *closed systems* that do not interact with an environment [3, 18]. Closed synthesis is in this sense a constructive extension of satisfiability checking. This approach is not suitable for *open systems*, which interact with a predefined environment, since the synthesized system cannot restrict the

* This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

behavior of its environment. Later works therefore concentrate on the synthesis of open systems from linear-time specifications [15, 16, 1]. These fundamental works on open synthesis required that the system satisfies its specification for *all* possible behaviors of the environment, i.e., an LTL formula φ is interpreted as the CTL* formula $A\varphi$. Pnueli and Rosner [15] demonstrated that the LTL synthesis problem is 2EXPTIME-complete in this setting. Kupferman and Vardi [10] extended open synthesis to branching-time specifications and incomplete information, and established EXPTIME and 2EXPTIME-completeness results for the CTL and CTL* synthesis problem, respectively.

In the view of the attractiveness of synthesis, it is alluring to extend its applicability as far as possible. A particular interesting extension is the treatment of probabilistic systems. Probabilistic randomization has, e.g., successfully been introduced into protocols (cf. [13]). In synthesis, we want to construct systems which, under reasonable assumptions about the probabilistic behavior of the environment, satisfy a linear-time specification with probability 1 (*almost-surely*) or with probability greater than 0 (*observably*).

System synthesis is more complex than model-checking probabilistic systems (Markov decision processes). There, a probabilistic measure is defined a priori on the set of computations, usually by assigning fixed probabilities to the single transitions. In synthesis, on the other hand, we do not have a transition-system to start with (this situation is comparable with the problem occurring in the treatment of transition fairness in system synthesis, cf. [2]).

When restricting the scope to almost-sure and observable satisfaction of linear-time properties, the concrete probabilities of single transitions play a minor role; in finite systems it is only of interest whether or not a probability is 0 or 1. It turns out that these properties are preserved when the probabilities of the single transitions are uncertain, as long as an (arbitrary) lower bound $\varepsilon > 0$ on their probability is guaranteed. This allows for considering synthesis for environments, which only guarantee the *existence* of some lower bound on the probability of each single action. We call such environments *ε-environments*. They are closely related to probabilistic fair systems [5] (with the distinction that systems discussed in this paper necessarily have a predefined *constant* set of environment actions) and inherit their semantical benefits: they provide a simple way of representing probabilistic choices while abstracting from the numerical value of probability. The LTL synthesis problem remains 2EXPTIME complete in almost-sure and observable semantics for ε-environments.

The decidability of almost-sure and observable acceptance gives rise to a redefinition of the semantics for the branching-time logic CTL*. CTL* allows for universal ($A\pi$) and existential ($E\pi$) path quantification. A natural analogy is to interpret universal path quantification as the property that the probability measure of the paths satisfying π is 1 (i.e., that a path almost-surely satisfies π), and existential path quantification as the property that the probability measure of the paths satisfying π is greater than 0 [8]. This paper provides a constructive method to solve the synthesis problem for CTL* in 3EXPTIME in the length of the specification, whereas a 2EXPTIME lower bound is inherited from the

LTL synthesis problem. While the exact complexity remains open for CTL*, the synthesis problem is EXPTIME-complete for CTL.

Under the assumption of stronger environments, which can reduce the probability of each single event arbitrarily, synthesis for almost-sure/observable semantics is essentially equivalent to synthesis for classical semantics.

2 Preliminaries

Synthesis algorithms automatically construct, for a given class of environments, systems that are correct by construction from a given specification. The environment is an external part of the system, which is not under the control of the synthesis algorithm. Intuitively, the environment provides the system with inputs from a finite input-alphabet Υ. The system reacts on each input by emitting an output symbol from a finite output-alphabet Σ. When the specifications are provided as temporal logics, the input- and output alphabet consist of the possible valuations of boolean input- and output-variables, respectively [15, 10, 11], which also serve as atomic propositions in the specification. A system is modeled as a finite transition-system, which defines a mapping $m : \Upsilon^* \to \Sigma$ from histories of input-signals to output-signals. This paper addresses synthesis for linear- and branching-time specifications for environments with an uncertain probabilistic behavior.

Environments. In general, the concrete behavior of the environment is unknown or too complex to represent. The uncertainty with respect to the concrete behavior of the environment is expressed by the power of the environment to choose, in every step, a probability distribution of its single input letters.

An environment is called an *ε-environment* if, in each step, the probability $p(\upsilon) \in [\varepsilon, 1]$ that the environment chooses a particular input letter $\upsilon \in \Upsilon$ is bound from below by some $\varepsilon > 0$. It is called a *0-environment*, if the probability that the environment chooses a particular input letter $\upsilon \in \Upsilon$ is not bound from below ($p(\upsilon) \in]0, 1]$ or $p(\upsilon) \in [0, 1]$).

Transition Systems. A system is implemented as a finite *Σ-labeled Υ-transition-system* $\mathcal{T} = (S, s_0, \tau, l)$, where S is a set of states with initial state $s_0 \in S$, $\tau : S \times \Upsilon \to S$ is a transition function and $l : S \to \Sigma$ is a labeling function. A Σ-labeled Υ-transition-system is called *input-preserving*, if $\Sigma = \Upsilon \times \Sigma'$ for some Σ' and the Υ-projection of $l(\tau(s, \upsilon))$ is υ for all $s \in S$ (i.e., the Υ-part of the label reflects the previous input from the environment).

Parity Automata. An *alternating automaton* is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, \alpha)$, where Σ denotes a finite set of labels, Q denotes a finite set of states, $q_0 \in Q$ denotes a designated initial state, δ denotes a transition function, and $\alpha : Q \to C \subset \mathbb{N}$ is a coloring function. The transition function $\delta : Q \times \Sigma \to \mathbb{B}^+(Q \times \Upsilon)$ maps a state and an input letter to a positive boolean combination of states and directions. In our context, an alternating automaton runs on Σ-labeled

Υ-transition-systems. The acceptance mechanism of alternating automata is defined in terms of run trees.

As usual, a Ξ-tree is a prefixed closed subset $Y \subseteq \Xi^*$ of the finite words over a predefined set Ξ of directions. For given sets Σ and Ξ, a *Σ-labeled Ξ-tree* is a pair $\langle Y, l\rangle$, consisting of a tree $Y \subseteq \Xi^*$ and a labeling function $l : Y \to \Sigma$ that maps every node of Y to a letter of Σ. If Υ and Σ are not important or clear from the context, $\langle Y, l\rangle$ is called a tree.

A *run tree* $\langle R, r\rangle$ on a given transition-system $\mathcal{T} = (S, s_0, \tau, l)$ is a $Q \times S$-labeled tree where the root is labeled with (q_0, s_0) and where, for each node n with label (q, s), there is a set $\mathfrak{A}_n \subseteq Q \times \Upsilon$ which satisfies $\delta(q, l(s))$ such that $(q', \upsilon) \in \mathfrak{A}_n$ iff a child of n is labeled with $(q', \tau(s, \upsilon))$.

An infinite path fulfills the *parity condition*, if the highest color of the states appearing infinitely often on the path is even. A run tree is *accepting* if all infinite paths fulfill the parity condition. A transition-system is accepted if it has an accepting run tree.

The set of transition-systems accepted by an alternating automaton $\mathcal{A}$ is called its *language* $\mathcal{L}(\mathcal{A})$. An automaton is empty, if its language is empty.

The acceptance of a transition-system can also be viewed as the outcome of a game, where player *accept* chooses, for a pair $(q, \sigma) \in Q \times \Sigma$, a set of atoms of $\delta(q, \sigma)$, satisfying $\delta(q, \sigma)$, and player *reject* chooses one of these atoms, which is executed. The input tree is accepted iff player *accept* has a strategy enforcing a path that fulfills the parity condition. One of the players has a memoryless winning strategy, i.e., a strategy where the moves only depend on the state of the automaton and the state of the transition-system, and, for player *reject*, on the choice of player *accept* in the same move.

A *nondeterministic* automaton is a special alternating automaton, where the image of δ consists only of such formulas that, when rewritten in disjunctive normal form, contain exactly one element of $Q \times \{\upsilon\}$ for all $\upsilon \in \Upsilon$ in every disjunct. For nondeterministic automata, every node of a run tree corresponds to a node in the input tree (the unrolling of the transition-system). Emptiness can therefore be checked with an *emptiness game*, where player *accept* also chooses the letter of the input alphabet. A nondeterministic automaton is empty iff the emptiness game is won by *reject*.

A nondeterministic automaton is called *deterministic* if the image of δ consists only of such formulas that, when rewritten in disjunctive normal form, contain exactly one disjunct. An automaton is called a *word* automaton if Υ is singleton; in this case, Υ is omitted in the notation. An automaton is called *weak* if, for every path on every run tree for every transition-system, the color increases monotonously, i.e, if δ maps each pair (q, σ) of states and input letters to positive boolean combination over pairs of states and directions, where the color of the respective state is not smaller than the color of q. An automaton is called a *Büchi* automaton iff the image of α is contained in $\{1, 2\}$.

The Synthesis Problem. For trace languages, we distinguish *almost-sure* and *observable* acceptance of transition-systems. A transition-system $\mathcal{T}$ satisfies a specification

– *almost-surely* iff the probability measure of the set of infinite paths defined by $\mathcal{T}$ that satisfy the specification is 1, and
– *observably* iff the probability measure of the set of infinite paths defined by $\mathcal{T}$ that satisfy the specification is greater than 0.

In case of temporal logics, the input-alphabet 2^I and output-alphabet 2^O represent the possible assignments to boolean input and output variables, which also serve as atomic propositions in the specification.

For CTL* specifications, all subformulas of the form $A\pi$ and $E\pi$ are interpreted as state formulas with the semantics that the path formula π is satisfied almost-surely and observably, respectively. The *synthesis problem* is to either construct, for a given input-alphabet Υ, a given output-alphabet Σ and a specification φ, an input-preserving $\Upsilon \times \Sigma$-labeled Υ-transition-system which satisfies the specification, or to prove that no such transition-system exists.

3 Synthesis for Trace Languages

Following an automata-theoretic approach to open synthesis, the synthesis problem is decomposed into two parts: finding an automaton, which accepts a transition-system iff it is input-preserving and satisfies the specification, and constructing a transition-system accepted by this automaton (or demonstrating its emptiness). In this section, we consider synthesis for specifications provided as deterministic word automata under the assumption of ε-environments.

Structural Acceptance Criteria. Testing whether a transition-system $\mathcal{T}$ almost-surely (observably) satisfies a deterministic word automaton $\mathcal{D}$ can be reduced to a simple structural argument over the composition of $\mathcal{T}$ and $\mathcal{D}$. The result of their composition is a colored graph, and it suffices to check if the highest color in all (some) reachable strongly connected components of $\mathcal{G}_\mathcal{D}^\mathcal{T}$ that are leaves in the SCC-graph of $\mathcal{G}_\mathcal{D}^\mathcal{T}$ is even.

The composition $\mathcal{G}_\mathcal{D}^\mathcal{T} = \mathcal{T} \| \mathcal{D}$ of a transition-system $\mathcal{T} = (S, s_0, \tau, l)$ and a deterministic word automaton $\mathcal{D} = (\Sigma, Q, q_0, \delta, \alpha)$ is a colored graph $\mathcal{G}_\mathcal{D}^\mathcal{T} = (S \times Q, (s_0, q_0), \tau', \alpha')$ with transition function $\tau' : ((s, q), \upsilon) \mapsto (\tau(s, \upsilon), \delta(q, l(s)))$ and coloring function $\alpha' : (s, q) \mapsto \alpha(q)$.

Lemma 1. *An Υ-transition-system $\mathcal{T}$ almost-surely (observably) satisfies a specification provided as a deterministic word automaton $\mathcal{D}$ iff the highest color in all (some) reachable leaf-SCCs of $\mathcal{G}_\mathcal{D}^\mathcal{T} = \mathcal{T} \| \mathcal{D}$ is even.*

Proof. For all ε-environments, the probability of every single transition is bound from below by some $\varepsilon \in]0, 1]$. This implies the following attributes of the computations:

– Almost-surely almost all states of a computation are in a single leaf of the SCC-tree of $\mathcal{G}_\mathcal{D}^\mathcal{T}$, which is reachable from the initial state of $\mathcal{G}_\mathcal{D}^\mathcal{T}$:
If $\mathcal{G}_\mathcal{D}^\mathcal{T}$ has n states, then, from every state of $\mathcal{G}_\mathcal{D}^\mathcal{T}$, the probability *not* to reach some leaf-SCC within the next n steps is bound from above by $\varepsilon' = 1 - \varepsilon^n < 1$, which implies a probability of 0 to stay forever out of reachable leaf-SCCs.

– Every reachable leaf-SCC of $\mathcal{G}_{\mathcal{D}}^{\mathcal{T}}$ is reached with some positive probability (which is bound from below by ε^n).
– For traces that eventually reach a leaf-SCC L, the highest color occurring infinitely often is almost-surely the highest color of the states of L:
The probability *not* to reach some state s in L within the next n steps is again bound from above $\varepsilon' = 1 - \varepsilon^n < 1$. This implies, for every position in the trace, a probability of 0 that s occurs never again; this holds in particular for a state s whose color is maximal in L.

The first (second) and third attribute imply the claim for almost-sure (observable) satisfaction. □

Game Construction. These structural criteria can be transformed into (weak) *acceptance games* deciding almost-sure and observable acceptance, respectively. These games are played on $\mathcal{G}_{\mathcal{D}}^{\mathcal{T}}$, starting in (s_0, q_0), and consist of three phases. For almost-sure (observable) acceptance the game is played according to the following rules:

– In the first phase, player *reject* (*accept*) either chooses to proceed to the second phase or picks a transition in $\mathcal{G}_{\mathcal{D}}^{\mathcal{T}}$. Picking a transition means that, in a state (s, q), she chooses a direction υ and the game proceeds in $\tau'((s, q), \upsilon)$. Intuitively, she can use this phase to move to a leaf-SCC of her choice.
– In the second phase, player *accept* (*reject*) either picks a transition in $\mathcal{G}_{\mathcal{D}}^{\mathcal{T}}$ or chooses to proceed to the third phase, but with the restriction that he can only move to the third phase if the color of the current node is even (odd). In case he moves to the third phase, the color c of the current node is stored. This phase is to prevent player *reject* (*accept*) from "cheating" by terminating the first phase in a state of $\mathcal{G}_{\mathcal{D}}^{\mathcal{T}}$, which is no element of any leaf-SCC. Player *accept* could, in such a case, move on to a vertex with highest color in a leaf-SCC of his choice (reachable from v), or even pick any arbitrary state reachable from v.
– In the last phase, player *reject* (*accept*) again chooses the transitions. She wins immediately upon reaching a state with an odd (even) color greater than c.

Infinite plays of the game are won by player *accept* (*reject*) if the game always stays in the first phase and if the game eventually stays forever in the third phase, while player *reject* (*accept*) wins otherwise.

Lemma 2. *The acceptance game on $\mathcal{G}_{\mathcal{D}}^{\mathcal{T}}$ is won by player* accept *if, and only if, $\mathcal{T}$ satisfies $\mathcal{D}$ almost-surely (observably).*

Proof. To prove the claim for almost-sure acceptance, first assume that $\mathcal{T}$ does not satisfies $\mathcal{D}$ almost-surely. In this case, the highest color in some reachable leaf-SCC L of $\mathcal{G}_{\mathcal{D}}^{\mathcal{T}}$ is odd by Lemma 1. Player *reject* can direct the game towards such a leaf-SCC L and then let the game proceed to the second phase.

If player *accept* ever moves on to the third phase, he must do so from a state in L. Since L is cyclic, player *reject* can then move to a state with maximal (odd)

color and wins directly. If, on the other hand, player *accept* never moves to the third phase, player reject wins since the third phase is never reached.

To prove the "if" direction, recall that almost-sure satisfaction of $\mathcal{D}$ by $\mathcal{T}$ entails that the highest color in all reachable leaf-SCCs of $\mathcal{G}_{\mathcal{D}}^{\mathcal{T}}$ is even. If player *reject* never leaves the first phase, player *accept* wins due to the winning condition for infinite plays. If player *reject* eventually changes in some state v to the second phase, then player *accept* can move to *some* leaf-SCC L. Since L is cyclic by definition, he can reach a state v' in L, whose (even) color is maximal in L. After having moved on to v', player *accept* changes to the third phase (storing the color of v'). Since the color of v' is maximal in L, player *reject* cannot win directly in the third phase, and consequently loses by the winning condition for infinite plays.

The proof for observable acceptance runs accordingly. □

From Acceptance Games to Automata. It is only a small step from the acceptance games of the previous paragraph to weak alternating automata over transition-systems. A given deterministic word automaton $\mathcal{D}$ can be turned into weak alternating automata, which accept a transition-system iff it satisfies $\mathcal{D}$ almost-surely or observably, respectively. The states of these automata are constructed from the states and colors of $\mathcal{D}$, and the transition function reflects the transitions of the game introduced in the previous paragraph.

Theorem 1. *Given a deterministic word automaton $\mathcal{D} = (\Sigma, Q, q_0, \delta, \alpha)$ we can construct weak alterating tree automata $\mathcal{A}_{\mathcal{D}}$ and $\mathcal{O}_{\mathcal{D}}$ which accept a Σ-labeled Υ-transition-system if it almost-surely and observably satisfies $\mathcal{D}$, respectively. If $\mathcal{D}$ has n states and c colors, $\mathcal{A}_{\mathcal{D}}$ and $\mathcal{O}_{\mathcal{D}}$ have at most $n \cdot \lceil 2 + \frac{c}{2} \rceil$ states.*

Proof. $\mathcal{A}_{\mathcal{D}} = (\Sigma, Q', q_0', \delta', \alpha')$ is defined as follows:

- The set of states is set to $Q' = Q \times (\{f, s\} \cup C_e)$ and initial state $q_0' = (q_0, f)$, where C_e denotes the set of even colors of $\mathcal{D}$.
- The transition function is defined by:
  - $\delta' : ((q, f), \sigma) \mapsto \delta'((q, s), \sigma) \wedge \bigwedge_{\upsilon \in \Upsilon}((\delta(q, \sigma), f), \upsilon)$,
  - $\delta' : ((q, s), \sigma) \mapsto \delta'((q, \alpha(q)), \sigma) \vee \bigvee_{\upsilon \in \Upsilon}((\delta(q, \sigma), s), \upsilon)$ if $\alpha(q)$ is even and
  - $\delta' : ((q, s), \sigma) \mapsto \bigvee_{\upsilon \in \Upsilon}((\delta(q, \sigma), s), \upsilon)$ if $\alpha(q)$ is odd,
  - $\delta' : ((q, c), \sigma) \mapsto$ *false* if $\alpha(q)$ is an odd number greater then c, and
  - $\delta' : ((q, c), \sigma) \mapsto \bigwedge_{\upsilon \in \Upsilon}((\delta(q, \sigma), c), \upsilon)$ otherwise.
- The coloring function α' maps $Q \times \{f\}$ to 0, $Q \times \{s\}$ to 1, and $Q \times C_e$ to 2.

Likewise, $\mathcal{O}_{\mathcal{D}} = (\Sigma, Q'', q_0'', \delta'', \alpha'')$ is defined as follows:

- The set of states is set to $Q'' = Q \times (\{f, s\} \cup C_o)$ and initial state $q_0'' = (q_0, f)$, where C_o denotes the set of odd colors of $\mathcal{D}$.
- The transition function is defined by:
  - $\delta'' : ((q, f), \sigma) \mapsto \delta''((q, s), \sigma) \vee \bigvee_{\upsilon \in \Upsilon}((\delta(q, \sigma), f), \upsilon)$,
  - $\delta'' : ((q, s), \sigma) \mapsto \delta''((q, \alpha(q)), \sigma) \wedge \bigwedge_{\upsilon \in \Upsilon}((\delta(q, \sigma), s), \upsilon)$ if $\alpha(q)$ is odd and
  - $\delta'' : ((q, s), \sigma) \mapsto \bigwedge_{\upsilon \in \Upsilon}((\delta(q, \sigma), s), \upsilon)$ if $\alpha(q)$ is even,
  - $\delta'' : ((q, c), \sigma) \mapsto$ *true* if $\alpha(q)$ is an even number greater then c, and

- $\delta'' : ((q,c),\sigma) \mapsto \bigwedge_{\upsilon\in\Upsilon}((\delta(q,\sigma),c),\upsilon)$ otherwise.

– The coloring function α'' maps $Q\times\{f\}$ to 1, $Q\times\{s\}$ to 2, and $Q\times C_o$ to 3.

The states $Q\times\{f\}$ refer to the first phase of the acceptance game, the states $Q\times\{s\}$ to the second and the remaining states $Q\times C_e$ and $Q\times C_o$, respectively, refer to the third phase of the acceptance game. A winning strategy for either player in the acceptance game on $\mathcal{G}^{\mathcal{T}}_{\mathcal{D}}$ can easily be transformed into a winning strategy in the acceptance game of the respective alternating automaton. □

Efficient Nondeterminization. Weak alternating automata are well suited for model-checking, but synthesis (or its non-constructive equivalent, checking non-emptiness) usually contains an exponential blow-up due to a nondeterminization step. A closer look on the special weak alternating automata of Theorem 1 reveals that this is not the case here: Most decisions can easily be guessed by a nondeterministic automaton. The crucial point in the nondeterminization is the single decision of player *reject* when to proceed from the first to the second phase (in case of almost-sure acceptance) and from the second to the third phase (in case of observable acceptance), respectively. It turns out that this single decision can be left uncertain in the construction of a nondeterministic automaton, avoiding the blow-up.

Theorem 2. *Given deterministic word automaton $\mathcal{D} = (\Sigma, Q, q_0, \delta, \alpha)$ we can construct nondeterministic Büchi tree automata $\mathcal{A}_{\mathcal{D}}{}'$ and $\mathcal{O}_{\mathcal{D}}{}'$ which accept a Σ-labeled Υ-transition-system if it almost-surely and observably satisfies $\mathcal{D}$, respectively. If $\mathcal{D}$ has n states and c colors, $\mathcal{A}_{\mathcal{D}}{}'$ and $\mathcal{O}_{\mathcal{D}}{}'$ have at most $2n\cdot\lfloor 1+\frac{c}{2}\rfloor+1$ and $n\cdot\lfloor 2+\frac{c}{2}\rfloor$ states, respectively.*

Proof. The nondeterministic Büchi tree automaton $\mathcal{O}_{\mathcal{D}}{}' = (\Sigma, Q'', q_0'', \delta'', \alpha'')$ for testing observable acceptance is defined as follows:

– The set of states is set to $Q'' = Q \cup Q\times C_o^-$ and the initial state $q_0'' = q_0$ is the initial state from $\mathcal{D}$. C_o^- denotes the set of odd colors of $\mathcal{D}$, plus an additional color $e_{min} = o_{min} - 1$, where o_{min} denotes the smallest odd color of $\mathcal{D}$.

– The transition function is defined by:

- $\delta'' : (q,\sigma) \mapsto \bigvee_{\upsilon\in\Upsilon}(\delta(q,\sigma),\upsilon) \vee \delta''(q, e_{min}),\sigma)$,
- $\delta'' : ((q,c),\sigma) \mapsto \bigvee_{\upsilon\in\Upsilon}\Big(((\delta(q,\sigma), \max\{c,\alpha(q)\}),\upsilon)$
  $\wedge \bigwedge_{\upsilon\neq\upsilon'\in\Upsilon}((\delta(q,\sigma),e_{min}),\upsilon)\Big)$ if $\alpha(q)$ is odd,
- $\delta'' : ((q,c),\sigma) \mapsto \bigwedge_{\upsilon\in\Upsilon}((\delta(q,\sigma),e_{min}),\upsilon)$
  if $\alpha(q) > c$ is even and greater than c, and
- $\delta'' : ((q,c),\sigma) \mapsto \bigvee_{\upsilon\in\Upsilon}\Big(((\delta(q,\sigma),c),\upsilon) \wedge \bigwedge_{\upsilon\neq\upsilon'\in\Upsilon}((\delta(q,\sigma),e_{min}),\upsilon)\Big)$
  if $\alpha(q) < c$ is even and smaller than c.

– The coloring function α'' maps the states $Q\times\{e_{min}\}$ to 2 and the remaining states to 1.

The states in Q reflect the first phase of the acceptance game on $\mathcal{G}^{\mathcal{T}}_{\mathcal{D}}$: player accept moves to a position of her choice ($\bigvee_{\upsilon\in\Upsilon}(\delta(q,\sigma),\upsilon)$) and eventually moves on to the second phase ($(\delta''(q,e_{min}),\sigma)$). The color 1 for these states reflect the winning condition on infinite plays (player *accept* looses if she stays for ever in the first phase).

In the second phase, the situation is more involved, since rather than guessing the action of player *accept*, the automaton needs to cover all possible actions of player *reject*. Intuitively, the option of player *reject* to stay in the second phase is covered by sending, from a state (q,c), a copy (q',e_{min}) (with $q'=\delta(q,\sigma)$) to each direction. Since player *reject* looses when staying in the second phase indefinitely, the color of these states is 2. Additionally, if $\alpha(q)$ is odd, player *reject* could move to the third phase, which could be reflected by sending a copy $(q,\alpha(q))$ to some direction ($\alpha(q)$ denotes the color to be stored). Concurrently, we must consider the possibility that the game is in the third phase. If $\alpha(q)$ is even and greater than c, then player *accept* wins immediately (no successor send), otherwise (q',c) is sent to some successor. Since player *accept* loses by staying in the third phase indefinitely, the color of a state (q,c) with $c\neq e_{min}$ is 1. Since the situation of player *reject* becomes strictly better when the stored color c increases, we can, instead of sending (q',c) *and* (q',c') into the same direction, send only $(q',\max\{c,c'\})$. This results in the *nondeterministic* automaton $\mathcal{O}_{\mathcal{D}}{}'$.

The nondeterministic Büchi tree automaton $\mathcal{A}_{\mathcal{D}}{}'=(\Sigma,Q',q_0',\delta',\alpha')$ for testing almost-sure acceptance is defined as follows:

- The set of states is set to $Q'=Q\times\mathbb{B}\times C_e^+\cup\{\bot\}$ with initial state $q_0'=(q_0,\mathit{true},e_{max})$, where C_e^+ denotes the set of even colors of $\mathcal{D}$, plus, if the highest color of $\mathcal{D}$ is an odd number o_{max}, $o_{max}+1$. e_{max} denotes the highest number in C_e^+.
- The transition function is defined by:
  - $\delta':((q,*,c),\sigma)\mapsto\bigvee_{\upsilon\in\Upsilon}\Big(((\delta(q,\sigma),\mathit{true},c),\upsilon)\wedge\bigwedge_{\upsilon\neq\upsilon'\in\Upsilon}((\delta(q,\sigma),\mathit{false},c),\upsilon)\Big)\vee\bigwedge_{\upsilon\in\Upsilon}(\delta(q,\sigma),\mathit{false},\min\{c,\alpha(q)\}),\upsilon)$ if $\alpha(q)$ is even,
  - $\delta':((q,*,c),\sigma)\mapsto\bigwedge_{\upsilon\in\Upsilon}(\bot,\upsilon)$ if $\alpha(q)>c$ is odd and greater than c,
  - $\delta':((q,*,c),\sigma)\mapsto\bigvee_{\upsilon\in\Upsilon}\Big(((\delta(q,\sigma),\mathit{true},c),\upsilon)\wedge\bigwedge_{\upsilon\neq\upsilon'\in\Upsilon}((\delta(q,\sigma),\mathit{false},c),\upsilon)\Big)$ otherwise, and
  - $\delta':(\bot,\sigma)\mapsto\bigwedge_{\upsilon\in\Upsilon}(\bot,\upsilon)$.
- The coloring function α' maps $Q\times\{\mathit{true}\}\times C_e^+$ and the error state $\bot$ to 1 and $Q\times\{\mathit{false}\}\times C_e^+$ to 2.

In almost-sure acceptance, the situation is slightly more involved. The states keep three pieces of information: the state of the deterministic word automaton, the information, if the game *could* be in the second phase, and a color, which reflects that the third phase could have been entered from a state in this color. The color is initialized to e_{max}, which is greater than all odd colors. From every point of the computation tree, one or no successor can refer to the second phase: No successor, if player *accept* would move to the third phase, and

one successor otherwise. Player *accept* loses iff there is a trace where he eventually stays indefinitely in the second phase, or if there is a trace where he eventually moves to the third phase in a state $(q, *, *)$ and then reaches a state $(q', *, *)$ with odd color $\alpha(q') > \alpha(q)$. The latter is modelled by moving to the designated error state $\bot$. The remaining information can be handled by storing the (even) color $\alpha(q)$ every time player *accept* would move to the third phase $(\bigwedge_{\upsilon \in \Upsilon}((\delta(q, \sigma), \mathit{false}, \min\{c, \alpha(q)\}), \upsilon))$ *or* by marking the direction player accept would choose when staying in the second phase $(\bigvee_{\upsilon \in \Upsilon}((\delta(q, \sigma), \mathit{true}, c), \upsilon) \wedge \bigwedge_{\upsilon \neq \upsilon' \in \Upsilon}((\delta(q, \sigma), \mathit{false}, c), \upsilon))$.

Obviously, a transition-system is rejected by ${\mathcal{A}_{\mathcal{G}}}'$ iff the acceptance game on $\mathcal{G}_{\mathcal{D}}^{\mathcal{T}}$ is won by player *reject*. □

These automata additionally have the pleasant property that their transition tables are short (at most $|\Upsilon| + 1$ entries for each state/input-letter pair).

The step to input-preserving transition-systems is a small one. The respective automaton can be multiplied with a deterministic safety automaton that checks if the label always agrees with the direction. The small transition table property is preserved by this transformation.

Theorem 3. *[11] Given an alternating tree automaton $\mathcal{A}$ over $\Upsilon \times \Sigma$-labeled Υ-transition-systems, we can construct an alternating tree automaton $\mathcal{A}'$ over $\Upsilon \times \Sigma$-labeled Υ-transition-systems that accepts a transition-system $\mathcal{T}$ iff it is input-preserving and accepted by $\mathcal{A}$. If $\mathcal{A}$ has n states, $\mathcal{A}'$ has at most $n \cdot |\Upsilon| + 1$ states, and if $\mathcal{A}$ is a (non)deterministic, weak or Büchi automaton, so is $\mathcal{A}'$.* □

4 Temporal Logics

While Section 3 provided basic techniques for trace languages and ε-environments, these results are transferred to temporal logics in this section. For the linear-time temporal logic LTL the techniques from the previous section can easily be applied: It suffices to translate an LTL formula into an equivalent deterministic word automaton, and then use the results of Section 3.

For probabilistic systems, the almost-sure/observable semantics for LTL inspire a redefinition of CTL* semantics [8]: Universal path quantification ($A\pi$) can be interpreted as the property that the probability measure of the paths satisfying π is 1, and existential path quantification can be interpreted as the property that the probability measure of the paths satisfying π is greater than 0.

Liner-Time Logic. Converting LTL formulas to deterministic word automata is well established.

Theorem 4. *[15, 7] Given an LTL specification φ, we can construct a deterministic word automaton $\mathcal{D}_\varphi$ that accepts exactly the models of φ. The number of states of $\mathcal{D}_\varphi$ is doubly exponential in the length of φ.* □

Given an LTL specification φ, we can, by the Theorems 4, 2 and 3, construct a nondeterministic Büchi tree automaton $\mathcal{N}_\varphi$ that accepts an input-preserving $2^I \times 2^O$-labeled 2^I-transition-system iff it almost-surely (observably) satisfies φ, such that the number of states of $\mathcal{N}_\varphi$ is doubly exponential in the length of φ. Checking $\mathcal{N}_\varphi$ for emptiness and, if $\mathcal{N}_\varphi$ is non-empty, constructing a transition-system accepted by $\mathcal{N}_\varphi$ reduces to solving a Büchi game, whose states intuitively consist of the states of $\mathcal{N}_\varphi$ and the entries in the transition-table of $\mathcal{N}_\varphi$.

Corollary 1. *Given an LTL specification φ we can, in time doubly-exponential in the length of φ, construct an input-preserving $2^I \times 2^O$-labeled 2^I-transition-system which almost-surely (observably) satisfies φ, or show that no such transition-system exists, in time doubly-exponential in the length of φ.* □

It turns out that this upper bound is sharp.

Theorem 5. *The LTL synthesis problem is 2EXPTIME complete.*

Proof. The upper bound is established by Corollary 1. To establish a matching lower bound, consider the ω-regular trace language

$$\mathcal{L}_n = \{\, \{0,1,2,3\}^* \cdot 3 \cdot \{0,1,2\}^* \cdot 2 \cdot v \cdot 2 \cdot \{0,1,2\}^* \cdot 3 \cdot v \cdot \{0,1,2\}^\omega \mid v \in \{0,1\}^n \}.$$

While $\mathcal{L}_n$ can be expressed by an LTL formula with size quadratic in n, any automaton accepting $\mathcal{L}_n$ necessarily has at least 2^n states [9] (since it must continously update the set of *subsets* of $\{0,1\}$ words of length n that have occurred between two 2 symbols since the last 3).

Consider a system with two boolean input variables i_1 and i_2, and a single output variable o. One can use i_1 and i_2 to encode the letters $0, \ldots, 3$, and represent the language $\mathcal{L}_n$ by a formula φ_n (of length quadratic in n).

The specification $\psi_n = \varphi_n \leftrightarrow FGo$ can only be satisfied by a transition-system with at least $O(2^n)$ states, regardless if in classical, almost-sure or observable semantics, since the transition-system *always* needs to react on an additional 3 (e.g., by setting the value of the output variable to *true* or *false* n steps after a 3 was read and keeping it constant otherwise). □

Branching-Time. In the branching-time case, one can use the fact that $E\psi$ and $A\psi$ are state-formulas. We call the strict subformulas of a CTL* specification φ of this special form the *basic* subformulas of φ, denoted *basic*(φ). Testing if a transition-system $\mathcal{T}$ satisfies a CTL* formula φ can be reduced to testing if the labels of $\mathcal{T}$ can be extended with suitable truth values for the basic subformulas of φ. The correct labels can be guessed on the fly.

Theorem 6. *Given a CTL* specification φ we can construct a weak alternating tree automaton $\mathcal{A}$ which accepts an $2^I \times 2^O$-labeled 2^I-transition-system iff it satisfies φ. The number of states of $\mathcal{A}$ is doubly-exponential in the length of φ.*

Proof. In our construction, the values of the basic formulas are guessed. Let $\mathcal{A}_\psi = (\Sigma^\psi, Q^\psi, q_0^\psi, \delta^\psi, \alpha^\psi)$ denote the weak alternating tree automaton that

accepts the models of a basic formula $\psi = E\psi'$ or $\psi = A\psi'$ of φ (or of φ itself), where the basic subformulas of ψ are provided as atomic proposions. $\mathcal{A}_\psi$ can be constructed by the method introduced in Theorem 1. The number of states of $\mathcal{A}_\psi$ is doubly exponential in the number of states of ψ. Let $\mathcal{A}_{\overline{\psi}} = (\Sigma^\psi, Q^{\overline{\psi}}, q_0^{\overline{\psi}}, \delta^{\overline{\psi}}, \alpha^{\overline{\psi}})$ denote the weak alternating automaton dual to $\mathcal{A}_\psi$.

We assume w.l.o.g. that φ is basic (otherwise we can replace the state formula φ by $A\varphi$ or $E\varphi$ without changing the semantics) and define the weak alternating tree automaton $\mathcal{A} = (2^I \times 2^O, Q, q_0, \delta, \alpha)$ as follows: The states $Q = Q^\varphi \cup \bigcup_{\psi \in basic(\varphi)} (Q^\psi \cup Q^{\overline{\psi}})$ are formed by the states of the single weak alternating automata $\mathcal{A}_\psi$, and the initial state $q_0 = q_0^\varphi$ is the initial state of $\mathcal{A}_\varphi$. The transition function is defined such that

$$\delta(q^\psi, \sigma) = \bigvee_{\Psi \subseteq basic(\psi)} \Big(\delta^\psi(q^\psi, \sigma \cup \Psi) \wedge \bigwedge_{\psi' \in \Psi} \delta(q_0^{\psi'}, \sigma) \wedge \bigwedge_{\psi' \in basic(\psi) \setminus \Psi} \delta(q_0^{\overline{\psi'}}, \sigma) \Big)$$

holds true. The coloring function maps a state q^ψ with even (odd) color $\alpha^\psi(q^\psi)$ in $\mathcal{A}_\psi$ to an even (odd) color, such that the weakness criterion is preserved.

Intuitively, the truth of the single basic subformulas is guessed on the fly. To demonstate that guessing these values is safe, we show that player *accept* has a winning strategy in the acceptance game if, and only if, he as a winning strategy where he always guesses the validity of all basic subformulas correctly. This can be demonstrated by induction along the structure of φ: Assume that player *accept* has a winning strategy where the truth value of some subformula is guessed incorrectly. Then there is a basic subformula ψ whose truth value is eventually guessed *incorrectly*, but the truth values of the basic subformulas of ψ are always guessed correctly. Then, for a state s in the transition-system $\mathcal{T}$ where the truth of ψ was eventually guessed incorrectly (w.l.o.g. to *true*), player accept has a winning strategy from (q_0^ψ, s) in the acceptance game, such that all values of basic subformulas of ψ are guessed correctly. Then player *accept* has a winning strategy in $\mathcal{A}_\psi$ when the labeling of $\mathcal{T}$ are enriched by the correct values for the basic subformulas of ψ (the winning strategy is the winning strategy from $\mathcal{A}$, with the simplification that the correct values need not be guessed). But in this case ψ *is* valid in s. □

The automaton $\mathcal{A}_\varphi$ constructed by Theorem 6 can be turned into an equivalent nondeterministic Büchi tree automaton $\mathcal{N}_\varphi$ [14] with exponentially more states than $\mathcal{A}_\varphi$. The language of $\mathcal{N}_\varphi$ can be restricted to input-preserving transition-systems (Theorem 3). A transition-system accepted by $\mathcal{A}_\varphi$ can be constructed via solving the emptiness game for the resulting automaton.

Corollary 2. *Given a CTL* specification φ we can construct an input-preserving $2^I \times 2^O$-labeled 2^I-transition-system, or proof that no such system exists, in time triply exponential in the length of φ.*

Theorem 5 provides a 2EXPTIME lower bound, which leaves the exact characterization of the complexity of the CTL* synthesis problem open. For its important sub-logic CTL, the complexity coincides with the synthesis complexity for classical semantics.

Theorem 7. *The CTL synthesis problem is EXPTIME complete.*

Proof. In CTL, each path quantifier refers to a path formula of the form $\psi_1 U \psi_2$, $G\psi_1$, or $X\psi_1$, where ψ_1 and ψ_2 are propositional (when basic formulas are viewed as propositions). For such path formulas (and their negations) acceptance of a path can be tested by a deterministic word automaton with three, two, or three states, respectively. The alternating automaton constructed by Theorem 6 is therefore only *linear* in the length of the specification, and emptiness can be checked (via nondeterminization [14] of this automaton) in time exponential in the length of the specification.

To demonstrate EXPTIME-hardness, we reduce solving the two player game PEEK-G_4 [17] to CTL synthesis. An instance of this game is a four-tuple $\langle X, Y, Z, \varphi \rangle$, where X and Y are disjoint sets of boolean variables with the intuition that X is under the control of the system and Y is under the control of the environment. $Z \subseteq X \cup Y$ denotes the variables which initially hold true and φ is a propositional formula over the variables $X \cup Y$. The game is played in rounds where first the system can change the value of at most one variable in X, followed by a decision of the environment to change the value of at most one variable in Y. The system wins the game iff φ is eventually satisfied (after the move of the system). To determine the winner of such games is EXPTIME-hard [17].

An instance of this game can be reduced to the synthesis problem for a system with one input-variable i, two output variables o_1 and o_2, and a CTL specification ψ quadratic in $|X|+|Y|$ and linear in φ. $\psi = \psi_0 \wedge \psi_1 \wedge \psi_2 \wedge \psi_3 \wedge \psi_\varphi$ is a conjunction of the following five CTL formulas:

- ψ_0 requires that the first $|X|$ values of o_1 reflect (on every path) the initial truth value of the variables in X (defined by $X \cap Z$) and the following $|Y|$ values of o_1 reflect the initial truth value of the variables in Y.
- ψ_1 requires that o_2 is *true* exactly every $|X| + |Y|$ steps (and initially) on every path.
- ψ_2 requires that at most one value of the variables o_1 within $|X| - 1$ steps after o_2 was last set to *true* (including the current step) differs from the value of o_1 $|X| + |Y|$ steps earlier.
- ψ_3 states that within $|X|$ to $|X| + |Y| - 1$ steps after o_2 was set true, the value of the variable o_1 is different from its value $|X| + |Y|$ steps earlier iff (1) the value of the input variable is *true* and (2) the values of the previous input variables since $|X|$ steps after o_2 was last set to *true* were all *false*.
- ψ_φ requires that, for all paths, there is eventually a position where o_2 is *true* and along the path where i is *false* for the following $|X| + |Y|$ steps, the following $|X| + |Y|$ values of o_1 (including the current value) satisfy φ.

ψ_2 and ψ_3 refer to the changing of at most one assignment for the variables of X and Y by the system and the environment, respectively, ψ_0 initializes the game and ψ_1 guarantees that o_2 can be used as a flag, indicating that a round starts. ψ_φ reflects the winning condition of the game. An input-preserving transition-system that satisfies ψ (in classical semantics as well as in almost-sure/observable semantics) defines a winning strategy for $\langle X, Y, Z, \varphi \rangle$ and vice versa. □

5 0-Environments

0-environments can "emphasize" each single path by assigning a probability measure of 1 (if the probability of each single action can be chosen from $[0, 1]$) or arbitrarily close to 1 (if the probability of each single action can be chosen from $]0, 1]$). For the latter consider an assignment of the probability $1 - 2^i \cdot \varepsilon$ for staying on the path desired by the environment in the i-th step for some $\varepsilon > 0$[1]. Consequently, the LTL synthesis problem coincides for almost-sure and observable semantics with the LTL synthesis problem for classical semantics, which is 2EXPTIME-complete [15].

For almost-sure/observable CTL* semantics this implies that existential and universal path quantifiers coincide. Consequently, a transition-system $\mathcal{T}$ is a model of a CTL* specification φ iff $\mathcal{T}$ is a model of a specification φ' in classical semantics, where φ' is obtained from φ by replacing all existential path quantifiers by universal path quantifiers. This implies EXPTIME and 2EXPTIME upper bounds for the CTL and CTL* synthesis problem [11], respectively.

On the other hand, in classical semantics each specification ψ can be translated to an equivalent specification ψ' by replacing each occurrence of an existential path quantifier E by the sequence $\neg A \neg$. Since the length of ψ' is linear in the length of ψ and the classical semantics for ψ' coincides with the almost-sure/observable semantics, the matching lower bounds for the CTL and CTL* synthesis problem [11] are preserved as well.

6 Conclusions

This paper suggests constructive decision procedures for the LTL, CTL and CTL* synthesis problems under the assumption of 0-environments and ε-environments. While the semantics for 0-environments essentially reflect the classical semantics and practically all established results trivially carry over, the results for ε-environments provide interesting new insights.

The results of this paper show that the complexity of synthesizing transition-systems satisfying an LTL or CTL specification φ in almost-sure/observable semantics is, under the assumption of ε-environments, equivalent to the complexity in classical semantics. While the complexity coincides, the language classes for LTL are at the same time simpler and more involved than for classical semantics: They are simpler in the sense that the languages are recognizable by *weak* alternating automata, and more involved since they cannot be recognized by deterministic automata.

Two interesting questions deserve further study: the exact complexity of CTL* synthesis in almost-sure/observable semantics, and the influence of incomplete information on the complexity of the LTL[2] synthesis problem. These

[1] The probability measure of the path is, in this case, greater than $1 - \varepsilon$, and can therefore be chosen arbitrarily close to 1 by the 0-environment.

[2] For CTL and CTL* synthesis, incomplete information can be handled using established automata-based techniques [10].

problems may be closely interrelated: In classical semantics, both problems can be solved through the existence of alternating automata that are only exponential in the length of a CTL* formula φ, which accept the models of φ. It does not seem unlikely that similar solutions exist for almost-sure/observable semantics, taking into account that model-checking remains PSPACE-complete (Yannakakis PSPACE result for LTL model-checking [4] trivially extends to CTL*).

An interesting side effect of using an automata-based synthesis algorithm is the possibility to extend the results for single-process synthesis directly to multi-process synthesis [12, 6].

References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. ICALP*, pages 1–17. Springer-Verlag, July 1989.
2. A. Anuchitanukul and Z. Manna. Realizability and synthesis of reactive modules. In *Proc. CAV*, pages 156–168. Springer-Verlag, June 1994.
3. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. IBM Workshop on Logics of Programs*, pages 52–71. Springer-Verlag, 1981.
4. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.
5. L. de Alfaro. From fairness to chance. In *Proc. PROBMIV'98*, 1999.
6. B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Proc. LICS*, pages 321–330. IEEE Computer Society Press, June 2005.
7. Y. Gurevich and L. Harrington. Trees, automata and games. 14:60–65, 1982.
8. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
9. O. Kupferman and M. Vardi. Freedom, weakness, and determinism: From linear-time to branching-time. In *Proc. LICS*, June 1995.
10. O. Kupferman and M. Y. Vardi. Synthesis with incomplete informatio. In *Proc. ICTL*, pages 91–106, Manchester, July 1997.
11. O. Kupferman and M. Y. Vardi. Church's problem revisited. *The bulletin of Symbolic Logic*, 5(2):245–263, June 1999.
12. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proc. LICS'01*, pages 389–398. IEEE Computer Society Press, July 2001.
13. D. Lehmann and M. O. Rabin. On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proc. POPL '81*, pages 133–138. ACM Press, 1981.
14. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theor. Comput. Sci.*, 141(1-2):69–107, 1995.
15. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. POPL*, pages 179–190. ACM Press, 1989.
16. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Automata, Languages and Programming*, pages 652–671. Springer-Verlag, 1989.
17. L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM J. Comput.*, 8(2):151–174, 1979.
18. P. Wolper. *Synthesis of Communicating Processes from Temporal-Logic Specifications*. PhD thesis, Stanford University, 1982.