Synthesis from Probabilistic Components*

Yoad Lustig, Sumit Nain, and Moshe Y. Vardi

Department of Computer Science
Rice University, Houston, TX 77005, USA
yoad.lustig@gmail.com , nain@cs.rice.edu , vardi@cs.rice.edu

Abstract

Synthesis is the automatic construction of a system from its specification. In classical synthesis algorithms, it is always assumed that the system is "constructed from scratch" rather than composed from reusable components. This, of course, rarely happens in real life, where almost every non-trivial commercial software system relies heavily on using libraries of reusable components. Furthermore, other contexts, such as web-service orchestration, can be modeled as synthesis of a system from a library of components. Recently, Lustig and Vardi introduced *dataflow* and *control-flow* synthesis from libraries of reusable components. They proved that dataflow synthesis is undecidable, while control-flow synthesis is decidable. In this work, we consider the problem of control-flow synthesis from libraries of *probabilistic components*. We show that this more general problem is also decidable.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Temporal synthesis, probabilistic components

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.412

1 Introduction

Hardware and software systems are rarely built from scratch. Almost every non-trivial system is based on existing components. A typical component might be used in the design of multiple systems. Examples of such components include function libraries, web APIs, and ASICs. Consider the mapping application in a typical smartphone. Such an application might call the location service provided by the phone's operating system to get the user's co-ordinates, then call a web API to obtain the correct map image tiles, and finally call a graphics library to display the user's location on the screen. None of these components are exclusive to the mapping application and all of them are commonly used by other applications.

The construction of systems from reusable components is an area of active research. Some examples of important work on the subject can be found in Sifakis' work on component-based construction [15], and de Alfaro and Henzinger's work on "interface-based design" [7]. Furthermore, other situations, such as web-service orchestration [2], can be viewed as the construction of systems from libraries of reusable components.

Synthesis is the automated construction of a system from its specification. In contrast to model checking, which involves verifying that a system satisfies the given specification, synthesis aims to automatically construct the required system from its formal specification. The modern approach to temporal synthesis was initiated by Pnueli and Rosner who introduced linear temporal logic (LTL) synthesis [13]. In LTL synthesis, the specification is given in LTL and the system constructed is a finite-state transducer modeling a reactive

* Work supported in part by NSF grants CCF-0728882, and CNS 1049862, by BSF grant 9800096, and by gift from Intel.

system. In this setting it is always assumed that the system is "constructed from scratch" rather than "composed" from existing components. Recently, Lustig and Vardi [11] introduced the study of synthesis from reusable components. The use of components abstracts much of the detailed behavior of a sub-system, and allows one to write specifications that mention only the aspects of sub-systems relevant for the synthesis of the system at large.

A major concern in the study of synthesis from reusable components is the choice of a mathematical model for the components and their composition. The exact nature of the reusable components in a software library may differ. One finds in the literature many different types of components; for example, function libraries (for procedural programming languages) or object libraries (for object-oriented programming languages). Indeed, there is no single "right" model encompassing all possible facets of the problem. The problem of synthesis from reusable components is a general problem to which there are as many facets as there are models for components and types of composition [15].

As a basic model for a component, following [11], we abstract away the precise details of the component and model a component as a *transducer*, i.e., a finite-state machine with outputs. Transducers constitute a canonical model for reactive components, abstracting away internal architecture and focusing on modeling input/output behavior. In [11], two models of composition were studied. In *dataflow* composition, the output of one component is fed as input to another component. The synthesis problem for dataflow composition was shown to be undecidable. In *control-flow* composition control is held by a single component at every point in time. The synthesis problem can then be viewed as constructing a supervisory transducer that switches control between the component transducers. Control-flow composition is motivated by software (and web services) in which a single function is in control at every point during the execution. LTL synthesis in this setting was shown in [11] to be 2EXPTIME-complete, just like classical LTL synthesis [13].

In this paper, we extend the control-flow synthesis model of [11] to probabilistic components, which are transducers with a probabilistic transition function. This is a well known approach to modeling systems where there is probabilistic uncertainty about the results of input actions. Intuitively, we aim at constructing a reliable system from unreliable components. There is a rich literature about verification and analysis of such systems, cf. [16, 5, 6, 17], as well about synthesis in the face of probabilistic uncertainty [1]. The introduction of probability requires us to use a probabilistic notion of correctness; here we choose the *qualitative* criterion that the specification be satisfied with probability 1, leaving the study of *quantitative criteria* to future work.

Here, our focus is on proving decidability, rather than on establishing precise complexity bounds, leaving the study of precise bounds to future work. Consequently, we abstract away from the details of the specification formalism and assume that the specification is given in terms of deterministic parity word automata (DPW). This allows us to consider all ω-regular properties. We define and study the *DPW probabilistic realizability* and synthesis problems, where the input is a library $\mathcal{L}$ of probabilistic components and a DPW $\mathcal{A}$, and the question is whether one can construct a *finite* system S from the components in $\mathcal{L}$, such that, regardless of the external environment, the traces generated by the system S are accepted by $\mathcal{A}$ with probability 1. Each component in the library can be used an arbitrary number of times in the construction and there is no apriori bound on the size of the system obtained. The technical challenge here is dealing with the finiteness of the system under construction. In [11], as well as in [13], one need not deal with finiteness from the start. In fact, one can test realizability without being concerned with finiteness of the constructed system, as finiteness is a *consequence* of the construction. This is not the case here, where we need to deal with finiteness from the start. Nevertheless, we are able to show that the problem is in 2EXPTIME.

Before tackling the full problem, we first consider a restricted version of the problem, where the specification is given in the form of a parity index on the states of the components, and the composed system must satisfy the parity condition. We call this the *embedded parity realizability* problem. We solve this problem and then show how solving the embedded parity realizability problem directly allows us to solve the more general DPW probabilistic realizability problem as well. The key idea here is that by taking the product of the specification DPW with each of the components, we can obtain larger components each of whose states has a parity associated with it. The challenge in completing the reduction is the need to generate a static composition, which does not depend on the history of the computation. Here we use ideas about synthesis with incomplete information from [10].

The paper is self-contained, except for certain technical proofs that have been omitted to save space; a longer version is posted on the authors' home pages.

2 Preliminaries

Given a set D of directions, a *D-tree* is a set $T \subseteq D^*$ such that if $x \cdot c \in T$, where $x \in D^*$ and $c \in D$, then also $x \in T$. For every $x \in T$, the words $x \cdot c$, for $c \in D$, are the *successors* of x. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or there exists a unique $c \in D$ such that $x \cdot c \in \pi$. The *full* D-tree is D^*. Given an alphabet Σ, a *Σ-labeled D-tree* is a pair $\langle T, \tau \rangle$, where T is a tree and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ. A *subtree* of $\langle D^*, \tau \rangle$, is a Σ-labeled D-tree $\langle T, \tau' \rangle$, where $\tau'(x) = \tau(x)$, for all $x \in T$. For a node $x \in D^*$, the *full subtree* at x is the subtree whose set of nodes is $x \cdot D^*$.

A *deterministic transducer* is a tuple $B = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, L \rangle$, where: Σ_I is a finite input alphabet, Σ_O is a finite output alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $L : Q \to \Sigma_O$ is an output function labeling states with output letters, and $\delta : Q \times \Sigma_I \to Q$ is a transition function. We define $\delta^* : \Sigma_I^* \to Q$ as follows: $\delta^*(\epsilon) = q_0$ and for $x \in \Sigma_I^*$ and $a \in \Sigma_I$, $\delta^*(x \cdot a) = \delta(\delta^*(x), a)$. We denote by $tree(B)$, the Σ_O-labeled Σ_I-tree $\langle \Sigma_I^*, \tau \rangle$, where for all $x \in \Sigma_I^*$, we have $\tau(x) = L(\delta^*(x))$. We say $tree(B)$ is the *unwinding* of B. A Σ-labeled D-tree T is called *regular*, if there exists a deterministic transducer C such that $T = tree(C)$.

Given a directed graph $G = (V, E)$, a *strongly connected component* of G is a subset U of V, such that for all $u, v \in U$, u is reachable from v. We can define a natural partial order on the set of maximal strongly connected components of G as follows: $U_1 \leq U_2$ if there exists $u_1 \in U_1$ and $u_2 \in U_2$ such that u_1 is reachable from u_2. Then $U \subseteq V$ is an *ergodic set* of G if it is a minimal element of the partial order.

A probability distribution on a finite set X is a function $f : X \to [0, 1]$ such that $\sum_{x \in X} f(x) = 1$. We use $Dist(X)$ to denote the set of all probability distributions on set X. A *probabilistic transducer*, is a tuple $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, F, L \rangle$, where: Σ_I is a finite input alphabet, Σ_O is a finite output alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : (Q - F) \times \Sigma_I \to Dist(Q)$ is a probabilistic transition function, $F \subseteq Q$ is a set of exit states, and $L : Q \to \Sigma_O$ is an output function labeling states with output letters. Note that there are no transitions out of an exit state. If F is empty, we say $\mathcal{T}$ is a probabilistic transducer without exits.

Given a probabilistic transducer $M = (\Sigma_I, \Sigma_o, Q, q_0, \delta, F, L)$, a *strategy* for M is a function $f : Q^* \to Dist(\Sigma_I)$ that probabilistically chooses an input for each sequence of states. A strategy is memoryless if the choice depends only on the last state in the sequence. A memoryless strategy can be written as a function $g : Q \to Dist(\Sigma_I)$. A strategy is *pure* if the choice is deterministic. A pure strategy is a function $h : Q^* \to \Sigma_I$, and a memoryless and pure strategy is a function $h : Q \to \Sigma_I$.

A strategy f along with a probabilistic transducer M, with set of states Q, induces a probability distribution on Q^ω, denoted μ_f. By standard measure theoretic arguments, it suffices to define μ_f for the cylinders of Q^ω, which are sets of the form $\beta \cdot Q^\omega$, where $\beta \in Q^*$. First we extend δ to exit states as follows: for $a \in \Sigma_I$, $q \in F$, $q' \in Q$, $\delta(q,a)(q) = 1$ and $\delta(q,a)(q') = 0$ when $q' \neq q$. Then we define $\mu_f(q_0 \cdot Q^\omega) = 1$, and for $\beta \in Q^*$, $q, q' \in Q$, $\mu_f(\beta q q' \cdot Q^\omega) = \mu_f(\beta q)(\sum_{a \in \Sigma_I} f(\beta q)(a) \times \delta(q,a)(q'))$. These conditions say that there is a unique start state, and the probability of visiting a state q', after visiting βq, is the same as the probability of the strategy picking a particular letter multiplied by the probability that the transducer transitions from q to q' on that input letter, summed over all input letters.

Let M be a probabilistic transducer, Q be its set of states, and f be a memoryless strategy for M. We define the graph induced by f on Q, denoted by $G_{M,f}$, as the directed graph (Q, E), where $(q_1, q_2) \in E$ if $\sum_{a \in \Sigma_I} f(q_1)(a)\, \delta(q_1, a)(q_2) > 0$. That is, there is an edge from q_1 to q_2 if the transducer can transition from the state q_1 to the state q_2 on an input letter that the strategy chooses with positive probability. Given $q_1, q_2 \in Q$, we say that q_2 is reachable from q_1 if there is a path from q_1 to q_2 in $G_{M,f}$. We say a state is ergodic if it belongs to some ergodic set of $G_{M,f}$. An ergodic set is reachable if there is a path from the start state to some state in the ergodic set. A state q of M is *reachable under* f, if there is a path in $G_{M,f}$ from q_0 to q.

A *library* is a set of probabilistic transducers that share the same input and output alphabets. Each transducer in the library is called a *component.* Given a finite set of directions D, we say a library $\mathcal{L}$ has width D, if each component in the library has exactly $|D|$ exit states. Since we can always add dummy unreachable exit states to any component, we assume, w.l.o.g., that all libraries have an associated width, usually denoted D. In the context of a particular component, we often refer to elements of D as exits, and subsets of D as sets of exits. Given a component M from library $\mathcal{L}$, and a strategy f for M, we say that the exit $i \in D$ is *selected* by f, if the ith exit state of M is reachable under f.

An *index function* for a transducer is a function that assigns a natural number, called a priority index, to each state of the transducer. An index function for a library is a function that assigns a priority to every state of every component in the library. Given an index function α for a library $\mathcal{L}$, we define $\max(\alpha)$ to be the highest priority assigned by α. We can assume, w.l.o.g., that $\max(\alpha)$ is not larger than twice the maximal number of states in the components of the library. Given a transducer M, index function α, and a strategy f for M, we say f *visits* priority p if there exists a state q of M such that $\alpha(q) = p$ and q is reachable under f.

3 Control-flow Composition from Libraries

We first informally describe our notion of control-flow composition of components from a library. The components in the composition take turns interacting with the environment, and at each point in time, exactly one component is active. When the active component reaches an exit state, control is transferred to some other component. Thus, to define a control flow composition, it suffices to name the components used and describe how control should be transferred between them. We use a deterministic transducer to define the transfer of control. Each library component can be used multiple times in a composition, and we treat these occurrences as distinct *component instances.* We emphasize that the composition can contain potentially arbitrarily many repetitions of each component inside it. Thus, the size of the composition, a priori, is not bounded. Note that our notion of composition is *static*, where the components called are determined before run time, rather than *dynamic*, where the components called are determined during run time.

Let $\mathcal{L}$ be a library with width D. A *composer* over $\mathcal{L}$ is a deterministic tranducer $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$. Here $\mathcal{M}$ is an arbitrary finite set of states. There is no bound on the size of $\mathcal{M}$. Each $\mathsf{M}_i \in \mathcal{M}$ is the name of an instance of a component from $\mathcal{L}$ and $\lambda(\mathsf{M}_i) \in \mathcal{L}$ is the type of M_i. We use the following notational convention for component instances and names: the upright letter M always denotes component names (i.e. states of a composer) and the italicized letter M always denotes the corresponding component instances (i.e. elements of $\mathcal{L}$). Further, for notational convenience we often write M_i directly instead of $\lambda(\mathsf{M}_i)$. Note that while each M_i is distinct, the corresponding components M_i need not be distinct. Each composer defines a unique composition over components from $\mathcal{L}$. The current state of the composer corresponds to the component that is in control. The transition function Δ describes how to transfer control between components: $\Delta(\mathsf{M}, i) = \mathsf{M}'$ denotes that when the composition is in the ith final state of component M it moves to the start state of component M'. A composer can be viewed as an implicit representation of a composition. We give an explicit definition of composition below.

▶ **Definition 1** (Control-flow Composition). Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ be a composer over library $\mathcal{L}$ with width D, such that $\mathcal{M} = \{\mathsf{M}_0, \ldots, \mathsf{M}_n\}$, $\lambda(\mathsf{M}_i) = (\Sigma_I, \Sigma_O, Q_i, q_0^i, \delta_i, F_i, L_i)$ and $F_i = \{q_x^i : x \in D\}$. The composition defined by C, denoted $\mathcal{T}_C$, is a probabilistic transducer $\langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \emptyset, L\rangle$, where $Q = \bigcup_{i=0}^{n}(Q_i \times \{i\})$, $q_0 = \langle q_0^0, 0\rangle$, $L(\langle q, i\rangle) = L_i(q)$, and the transition function δ is defined as follows: For $\sigma \in \Sigma_I$, $\langle q, i\rangle \in Q$ and $\langle q', j\rangle \in Q$,

1. If $q \in Q_i \setminus F_i$, then
$$\delta(\langle q, i\rangle, \sigma)(\langle q', j\rangle) = \begin{cases} \delta_i(q, \sigma)(q') & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$
2. If $q = q_x^i \in F_i$, where $\Delta(\mathsf{M}_i, x) = \mathsf{M}_k$, then
$$\delta(\langle q, i\rangle, \sigma)(\langle q', j\rangle) = \begin{cases} 1 & \text{if } j = k \text{ and } q' = q_0^k \\ 0 & \text{otherwise} \end{cases}$$

Note that the composition is a probabilistic transducer without exits. When the composition is in a state $\langle q, i\rangle$ corresponding to a non-exit state q of component M_i, it behaves like M_i. When the composition is in a state $\langle q_f, i\rangle$ corresponding to an exit state q_f of component M_i, the control is transferred to the start state of another component as determined by the transition function of the composer. Thus, at each point in time, only one component is active and interacting with the environment.

4 Synthesis for Embedded Parity

In this section we consider a simplified version of the general synthesis problem, where each state of a component in the library has a priority associated with it and the specification to be satisfied is that the highest priority visited i.o. must be even with probability 1.

Let M be a probabilistic tranducer and α be an index function. A strategy f for M is *winning* for the environment if with positive probability the highest priority visited infinitely often (i.o.) is odd. We say that M *satisfies* α if there exists no winning strategy for the environment. Given a composer C over library $\mathcal{L}$, we say that C *satisfies* α if $\mathcal{T}_C$ satisfies α.

Given a library $\mathcal{L}$ with width D, an *exit control relation* is a set $R \subseteq D \times \mathcal{L}$. We say that a composer $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ over $\mathcal{L}$ is *compatible* with R, if the following holds: for all $\mathsf{M}, \mathsf{M}' \in \mathcal{M}$ and $i \in D$, if $\Delta(\mathsf{M}, i) = \mathsf{M}'$ then $(i, M') \in R$. Thus, each element of R can be viewed as a constraint on how the composer is allowed to connect components.

▶ **Definition 2.** The *embedded parity realizability problem* is: Given a library $\mathcal{L}$ with width D, an exit control relation R for $\mathcal{L}$, and an index function α for $\mathcal{L}$, decide whether there exists a composer C over $\mathcal{L}$, such that C satisfies α and C is compatible with R. If such a composer exists, we say that $\mathcal{L}$ *realizes* α under R. The *embedded parity synthesis problem* is to find such a composer C if it exists.

The following theorem allows us to restrict attention to memoryless strategies. It states that if a winning strategy exists, then a memoryless winning strategy must also exist. Here we give a direct combinatorial proof, but we note that the result can also be obtained by adapting the methods in [4], where a similar result was proved for 2–1/2 player stochastic parity games by Chatterjee et al.

▶ **Theorem 3.** *Given a probabilistic transducer M, and index function α, if there exists a winning strategy for the environment then there exists a pure and memoryless winning strategy.*

Memoryless strategies are important because they induce an ergodic structure on the set of states. Ergodic sets are useful because they enable us to replace probabilistic reasoning with combinatorial reasoning. In particular, they have the following crucial properties: (a) the suffix of a path is contained in some ergodic set with probability 1, and (b) the suffix of a path is contained in a proper subset of an ergodic set with probability zero [9]. This allows us to define the winning strategy condition in terms of graph reachability.

▶ **Lemma 4.** *Let M be a probabilistic transducer and f be a memoryless strategy for M. Then f is winning for the environment iff $G_{M,f}$ has a reachable ergodic set whose highest priority is odd.*

When the underlying probabilistic transducer is a composition, ergodic sets acquire additional structure. Given a composer C and a memoryless strategy f for $\mathcal{T}_C$, if a reachable ergodic set X of $G_{\mathcal{T}_C,f}$ contains some state from a component M of $\mathcal{T}_C$, then either X is contained in M or all the reachable states of M are contained in X. Formally:

▶ **Lemma 5.** *Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ be a composer over $\mathcal{L}$ and f be a memoryless strategy for $\mathcal{T}_C$. Let $\mathsf{M}_i \in \mathcal{M}$ and Q_i be the state space of M_i. Let X be a reachable ergodic set of $G_{\mathcal{T}_C,f}$ such that $X \cap (Q_i \times \{i\}) \neq \emptyset$. Then either $X \subseteq Q_i \times \{i\}$ or $(Q_i \times \{i\}) \cap Y \subseteq X$, where Y is the set of states of $\mathcal{T}_C$ that are reachable under f.*

Proof. Assume that $X \cap (Q_i \times \{i\}) \neq \emptyset$ and X is not contained in $Q_i \times \{i\}$. Let $(q, i) \in X \cap (Q_i \times \{i\})$ and $(q', j) \in X - (Q_i \times \{i\})$, for some $j \neq i$. Since X is ergodic, there is a path π in $G_{\mathcal{T}_C,f}$ from (q', j) to (q, i). Let s be the first state along π such that $s = (q'', i) \in Q_i \times \{i\}$. We claim that $q'' = q_0^i$, where q_0^i is the start state of M_i. Let $s' = (q''', k)$, where $k \neq i$, be the predecessor of s in π. By the definition of $G_{\mathcal{T}_C,f}$, there is an edge from s' to s only if $\mathcal{T}_C$ can transition from s' to s on some input with positive probability. By Definition 1, $\mathcal{T}_C$ can transition from (q''', k) to (q'', i) only if q''' is a final state of M_k and q'' is the initial state of M_i. Thus (q_0^i, i) is in X.

Since X is an ergodic set, if it contains a state s of $\mathcal{T}_C$, then it also contains all states reachable under f from s. By definition, every state in $(Q_i \times \{i\}) \cap Y$ is reachable under f from (q_0^i, i). Since X contains (q_0^i, i), it also contains all states in $(Q_i \times \{i\}) \cap Y$. ◀

Given a graph G, each of whose vertices is assigned a priority, we say that G has the *odd ergodic property* if it has a reachable ergodic set whose highest priority is odd. Consider a composer C and a memoryless strategy f for $\mathcal{T}_C$. Then, by Lemma 4, f is winning for the environment iff $G_{\mathcal{T}_C,f}$ has the odd ergodic property. So the probabilistic notion of winning strategy is reduced to a combinatorial one. However, the graph $G_{\mathcal{T}_C,f}$ is very large as it contains all the internal states of each component explicitly. Further, to show that C satisfies α, we have to consider every possible memoryless strategy for C. We tackle this complexity by simplifying the description of a strategy f and graph $G_{\mathcal{T}_C,f}$ so as to abstract away the inner states of components and the choices that f makes on those inner states. Let $\mathcal{M}$ be the state space of C. We aim to replace $G_{\mathcal{T}_C,f}$ by a simpler graph G', whose set of vertices is $\mathcal{M}$, such that the odd ergodic property is preserved. We first discuss this transformation informally, and then give formal definitions and proofs.

Let M be a component of $\mathcal{T}_C$. If some reachable ergodic set of $G_{\mathcal{T}_C,f}$ lies entirely within M, we say M is a *sink*. When the highest priority in the ergodic set is odd (resp. even) we say M is an *odd* (resp. *even*) sink for f. Note that a component can be both an odd and an even sink for a given strategy. Intuitively, we aim to replace the subgraph of $G_{\mathcal{T}_C,f}$ that corresponds to states of M by a single new vertex x_M to obtain a new graph G' and assign a suitable priority to x_M such that the odd ergodic property is preserved by the transformation. Now if M is not a sink, then, by Lemma 5, x_M lies in a reachable ergodic set of G' iff all reachable states of M lie in a reachable ergodic set of $G_{\mathcal{T}_C,f}$. In this case, we can simply assign the highest reachable priority in M to x_M and the odd ergodic property is preserved. If, however, M is a sink, then the collapse of M to a single vertex might introduce new ergodic sets in the graph. That is, x_M might lie in an ergodic set of G' which has no analogue in $G_{\mathcal{T}_C,f}$. We then have to choose the priority of x_M such that the odd ergodic property is still preserved. There are two cases to consider:

- M is an odd sink for f. Then, by Lemma 4, f is winning for the environment. Let f_M denote f restricted to the states in M. Then f_M is a memoryless strategy for M that is winning for the environment, and in every composition involving M, the environment can simply play f_M on the states in M to win. So a component that is an odd sink is not useful for synthesizing compositions. We note that it is easy to check for and remove any odd sinks from $\mathcal{L}$ in a preprocessing step before attempting synthesis. Checking whether a particular component is a sink is equivalent to model checking Markov decision processes and can be done in polynomial time [16]. In the rest of the paper, we assume that the given library $\mathcal{L}$ does not contain components that are odd sinks.
- M is an even sink for f but not an odd sink for f. Then, by Lemma 5, every reachable state in M either lies in an even sink or does not lie in an ergodic set. So no reachable state in M is part of an ergodic set with odd highest priority. Thus collapsing M to x_M does not remove any ergodic sets with odd highest priority. It only remains to consider the possibility that the transformation can introduce a new ergodic set whose highest priority is odd. We can avoid this by assigning a priority of $2\max(\alpha)$ to x_M, where $\max(\alpha)$ is the highest parity assigned by the index function α. Then if x_M is part of a reachable ergodic set X' in G', then X' has highest priority $2\max(\alpha)$, which is even. Thus the odd ergodic property is preserved.

In formalizing the approach given above, instead of explicitly transforming $G_{\mathcal{T}_C,f}$ into a more abstract graph, it is simpler to directly define a suitable graph on the state space $\mathcal{M}$ of the composer C such that the odd ergodic property is preserved. Just as a memoryless strategy f applied to the composition $\mathcal{T}_C$ gives rise to the graph $G_{\mathcal{T}_C,f}$, we define a combinatorial object, called a *choice function*, such that choice function g together with composer C gives rise to a graph $G_{C,g}$.

▶ **Definition 6** (Choice Function). Given a library $\mathcal{L}$ with width D and index function α, we define the set $LABELS(\mathcal{L}) \subseteq 2^D \times \{1, \ldots, 2\max(\alpha)\} \times \mathcal{L}$ as follows: $(X, j, M) \in LABELS(\mathcal{L})$ iff there exists a memoryless strategy f for M such that

- $X \subseteq D$ is the set of exits of selected by f in M.
- If M is an even sink for f, then $j = 2\max(\alpha)$.
- Otherwise j is the highest priority visited by f in M.

Given a composer $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ over $\mathcal{L}$, a *choice function* for C, is a function $g : \mathcal{M} \to 2^D \times \{1, \ldots, 2\max(\alpha)\}$, such that, for all $\mathsf{M}_i \in \mathcal{M}$, $(g(\mathsf{M}_i), M_i) \in LABELS(\mathcal{L})$. The graph induced by g on C, denoted $G_{C,g}$, is the directed graph $(\mathcal{M}, \mathcal{E})$, where $(\mathsf{M}_1, \mathsf{M}_2) \in \mathcal{E}$ if $\Delta(\mathsf{M}_1, i) = \mathsf{M}_2$ for some $i \in D$ such that $i \in X$ where $g(\mathsf{M}_1) = (X, j)$. The priority of a vertex $\mathsf{M} \in \mathcal{M}$ of $G_{C,g}$ is j where $g(\mathsf{M}) = (X, j)$. We say that g has *rank* r, if $G_{C,g}$ has a reachable ergodic set whose highest priority is r.

The size of the set $LABELS(\mathcal{L})$ is at most $\max(\alpha)|\mathcal{L}|2^{|D|}$. For an arbitrary triple (X, j, M), we can check whether $(X, j, M) \in LABELS(\mathcal{L})$ in time polynomial in $|M|$ using standard techniques for solving Markov decision processes [16]. Thus $LABELS(\mathcal{L})$ can be computed in time exponential in the size of $\mathcal{L}$.

▶ **Theorem 7.** *Let C be a composer over $\mathcal{L}$. Then there exists a strategy for $\mathcal{T}_C$ that is winning for the environment iff there exists a choice function for C that has an odd rank.*

Proof. Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$. Let Q_i be the state space of $M_i = \lambda(\mathsf{M}_i)$, for $\mathsf{M}_i \in \mathcal{M}$, and let $Q = \bigcup(Q_i \times \{i\})$ be the state space of $\mathcal{T}_C$.

Only If: Assume there exists a strategy for $\mathcal{T}_C$ that is winning for the environment. Then, by Theorem 3, there exists a memoryless winning strategy f. We construct a choice function g for C as follows: for all $\mathsf{M}_i \in \mathcal{M}$, $g(\mathsf{M}_i) = (X, p)$, where X is the set of exits of M_i selected by f, and $p = 2\max(\alpha)$ if M_i is an even sink for f and otherwise p is the highest priority in M_i visited by f. Since f is winning, $G_{\mathcal{T}_C,f}$ has a reachable ergodic set H with odd highest priority r. Consider the set $\mathcal{H} \subseteq \mathcal{M}$ defined as follows: for all $\mathsf{M}_i \in \mathcal{M}$, $\mathsf{M}_i \in \mathcal{H}$ if $(Q_i \times \{i\}) \cap H \neq \emptyset$. Thus, $\mathcal{H}$ contains a state of the composer C if the corresponding component of $\mathcal{T}_C$ overlaps with the ergodic set H. Since $\mathcal{L}$ contains no components that are odd sinks, and even sinks can not be a part of an ergodic set whose highest priority is odd, H must contain all the reachable states in each component named in $\mathcal{H}$.

We claim that $\mathcal{H}$ is an ergodic set of $G_{C,g}$. We first show that $\mathcal{H}$ is strongly connected. Let M_i and M_k be in $\mathcal{H}$. Since all the reachable states of M_i and M_k are contained in H, in particular their start states are also contained in H. Let these be q_i and q_k respectively. Then there is a path in $G_{\mathcal{T}_C,f}$ from (q_i, i) to (q_k, k) because H is an ergodic set of $G_{\mathcal{T}_C,f}$. Consider the path π from (q_i, i) to (q_k, k) that contains the least number of exit states. Let the length of π be n and let (q'_i, i) be the first exit state along π. Suppose $\Delta(\mathsf{M}_i, x) = \mathsf{M}_j$, where q'_i is the exit state of M_i in direction x, and let q_j be the start state of M_j. Then, if $g(\mathsf{M}_i) = (X, p)$, we have $x \in X$, so there is an edge from M_i to M_j in $G_{C,g}$, and the immediate next state after (q'_i, i) in π is (q_j, j). The suffix of π starting from (q_j, j) is a path π' from (q_j, j) to (q_k, k) of length less than n. Further, by construction, among all such paths it has the least number of exit states. Assume, by the induction hypothesis, there is a path from M_j to M_k in $G_{C,g}$. Since $(\mathsf{M}_i, \mathsf{M}_j)$ is also an edge in $G_{C,g}$, therefore, by induction, there is a path from M_i to M_k in $G_{C,g}$. M_i and M_k were chosen arbitrarily in $\mathcal{H}$. So $\mathcal{H}$ is strongly connected.

Next, we show that there are no edges that leave $\mathcal{H}$. Assume there is some edge in $G_{C,g}$ from a vertex $\mathsf{M}_i \in \mathcal{H}$ to a vertex $\mathsf{M}_j \in \mathcal{M} - \mathcal{H}$. Let $g(\mathsf{M}_i) = (X, p')$. Then there exists $x \in X$ such that $\Delta(\mathsf{M}_i, x) = \mathsf{M}_j$. Let (q', i) be the exit state of M_i in direction x. Then (q', i) is reachable under f and so is (q_j, j), where q_j is the start state of M_j. Therefore, there is an edge in $G_{\mathcal{T}_C,f}$ from $(q', i) \in H$ to $(q_j, j) \notin H$, which contradicts that H is an ergodic set. Thus no edges leave $\mathcal{H}$ in $G_{C,g}$ and $\mathcal{H}$ is ergodic.

Finally, we show that the highest priority in $\mathcal{H}$ is r. By construction of g, since H does not contain any even sinks, the priority of a vertex M_i in $\mathcal{H}$ is the highest priority visited in M_i by f. Thus, the highest priority in $\mathcal{H}$ is at most the highest priority in H, which is r. Let $(q, j) \in H$ be such that q has priority r. Then the highest priority visited by f in M_j is r, so $g(\mathsf{M}_j) = (X, r)$ for some $X \subseteq D$. Since $\mathsf{M}_j \in \mathcal{H}$, the highest priority in $\mathcal{H}$ is r, and g has rank r.

If: Now assume that g is a choice function for C with rank p, for some odd $p \leq \max(\alpha)$. Then, by the definition of choice function, for all $\mathsf{M}_i \in \mathcal{M}$, there exists a memoryless strategy f_i for M_i, such that $g(\mathsf{M}_i) = (X_i, p_i)$ where X_i is the set of exit directions of M_i under f_i, and $p_i = 2\max(\alpha)$ if M_i is an even sink for f_i and otherwise p_i is the highest priority visited by f_i.

We define a memoryless strategy f for $\mathcal{T}_C$ as follows: for all $q \in Q_i$, $f(q, i) = f_i(q)$. Since g has rank p, there exists a reachable ergodic set $\mathcal{H} \subseteq \mathcal{M}$ of $G_{C,g}$ with highest priority p. Consider the set $H = \{(q, i) : q \in Q_i, \mathsf{M}_i \in \mathcal{H}\}$, which consists of all states in all components corresponding to the set $\mathcal{H}$. Let H_f be the subset of H that is reachable under f from the start state of $\mathcal{T}_C$. We first show that H_f is strongly connected. Let (q_i, i) and (q_k, k) be two arbitrary states in H_f. Then q_i is a state of M_i and q_k is a state of M_k. Further, M_i and M_k are both in $\mathcal{H}$. We have the following two cases:

1. *q_i is the start state of M_i.* Consider the shortest path in $G_{C,g}$ from M_i to M_k. Such a path exists because $\mathcal{H}$ is an ergodic set of $G_{C,g}$. Let the length of the path be n and let M_j be the successor of M_i in this path. So there is path of length $n-1$ in $G_{C,g}$ from M_j to M_k. Now, by the definition of $G_{C,g}$, there exists $x \in D$ such that $\Delta(\mathsf{M}_i, x) = \mathsf{M}_j$ and the exit state in direction x is reachable from the start state of M_i under f_i. Thus there is a path in $G_{\mathcal{T}_C,f}$ from (q_i, i) to (q_j, j) where q_j is the start state of M_j. By induction, there is a path in $G_{\mathcal{T}_C,f}$ from (q_i, i) to (q_k, k).
2. *q_i is not the start state of M_i.* Let $g(\mathsf{M}_i) = (X, p')$, where $X \subseteq D$. Since p is the highest priority in $\mathcal{H}$ and $\mathsf{M}_i \in \mathcal{H}$, we have $p' \leq p \leq \max(\alpha)$. Thus $p' \neq 2\max(\alpha)$ and so M_i is not an even sink for f. Also, the library $\mathcal{L}$ is assumed to have no components that are odd sinks. Thus, some exit of M_i must be reachable from q_i under f_i. Let this exit be in direction $x \in D$, and let $\Delta(\mathsf{M}_i, x) = \mathsf{M}_j$. Then there is a path in $G_{\mathcal{T}_C,f}$ from (q_i, i) to (q_j, j) where q_j is the start state of M_j. Now, since q_j is a start state, by the previous case, there is a path from (q_j, j) to (q_k, k) in $G_{\mathcal{T}_C,f}$. So there is a path from (q_i, i) to (q_k, k) and therefore H_f is strongly connected.

Assume that some edge in $G_{\mathcal{T}_C,f}$ leaves H_f. Let there be an edge between $(q, i) \in H_f$ and $(q', j) \in Q - H_f$. Now M_j can not belong to $\mathcal{H}$ because otherwise (q', j) would be in H_f. So we have $i \neq j$ and (q, i) must be an exit state of M_i. Therefore there is an edge in $G_{C,g}$ from $\mathsf{M}_i \in \mathcal{H}$ to $\mathsf{M}_j \in \mathcal{M} - \mathcal{H}$, which contradicts that $\mathcal{H}$ is ergodic. Thus H_f is also an ergodic set.

By Lemma 4, it suffices to show that the highest priority in H_f is odd. Now p is the highest priority in $\mathcal{H}$, and p is odd, which means $p \neq 2\max(\alpha)$. So there must exist $\mathsf{M}_i \in \mathcal{H}$ such that some state q in M_i has priority p and is reachable under f_i. Then (q, i) is in H_f and so H_f has highest priority at least p. Assume some state (q', j) in H_f has priority $p' > p$. Since q' is reachable under f_j, therefore, we have $g(\mathsf{M}_j) = (X, p'')$, for some $X \subseteq D$ and $p'' \geq p' > p$. This contradicts the fact that $\mathsf{M}_j \in \mathcal{H}$. Thus the highest priority in the ergodic set H_f is p, which is odd. ◀

Let $\Gamma = LABELS(\mathcal{L})$. A composer and choice function pair has a natural representation as a regular Γ-labeled D-tree. Given a composer $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ over $\mathcal{L}$, and a choice function g for C, we denote by $tree(C, g)$, the regular Γ-labeled full D-tree $\langle D^*, \tau\rangle$, where for all $x \in D^*$, we have that $\tau(x) = (g(\Delta^*(x)), \lambda(\Delta^*(x)))$. Thus $tree(C, g)$ is the tree obtained as a result of adding labels to $tree(C)$ such that a node x corresponding to $\mathsf{M}_i \in \mathcal{M}$ that is labeled with M_i in $tree(C)$ is labeled with (X, j, M_i) where $(X, j) = g(\mathsf{M}_i)$. As we show in the next lemma, the mapping is reversible, in the sense that given a regular Γ-labeled D-tree, we can obtain a composer and choice function in a natural way.

▶ **Lemma 8.** *Let T be a regular Γ-labeled full D-tree. Then there exist a composer C over $\mathcal{L}$ and a choice function g for C such that $tree(C, g) = T$.*

In light of Lemma 8, we can represent an arbitrary regular Γ-labeled full D-tree as $tree(C, g)$ for some composer C over $\mathcal{L}$ and some choice function g for C. Similarly, we can represent an arbitrary regular $\mathcal{L}$-labeled full D-tree as $tree(C)$ for some composer C over $\mathcal{L}$.

Since the question of whether a given composition satisfies α boils down to whether its composer has a choice function that has an odd rank, we find it useful to characterize regular trees that correspond to choice functions having a particular rank (see [14] for related results). First, we inductively define the set of *marked* nodes of a Γ-labeled D-tree as follows: the root is always marked, and a node $y \cdot i$, where $i \in D$ and $y \in D^*$, is marked if y is marked and $i \in X$, where (X, j, M) is the label on $y \cdot i$.

▶ **Lemma 9.** *Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ be a composer over library $\mathcal{L}$ with width D, α be an index function for $\mathcal{L}$, g be a choice function for C, and $p \leq \max(\alpha)$. Then g has rank p iff $tree(C, g)$ has a full subtree T such that:*

1. *The root of T is marked.*
2. *Every node in T that is marked has priority label at most p.*
3. *From each marked node in T there is a path in T to a marked node with priority label p.*

The conditions given by Lemma 9 can be checked by a suitable tree automaton as follows:

▶ **Lemma 10.** *Let $\mathcal{L}$ be a library with width D and let $p \leq k$. Then there exists an nondeterministic Büchi tree automaton (NBT) $\mathcal{A}_p$ such that $\mathcal{A}_p$ accepts a Γ-labeled regular D-tree T iff $T = tree(C, g)$ for some composer C over $\mathcal{L}$ and choice function g with rank p.*

Proof. By Lemma 8 and 9, it suffices to construct an NBT $\mathcal{A}_p$ such that $\mathcal{A}_p$ accepts a tree T' iff T' has a full subtree T that satisfies the three conditions in Lemma 9. For simplicity, the automaton is defined over binary trees, where $D = \{0, 1\}$, but the definition can be easily extended to n-ary trees.

Let $\mathcal{A}_p = (\Gamma, Q, q_0, \delta, \beta)$. We define $Q = \{\mathsf{search}, \mathsf{cut}, \mathsf{wait}, \mathsf{reach}, \mathsf{visit}, \mathsf{err}\}$, $q_0 = \mathsf{search}$ and $\beta = \{\mathsf{visit}, \mathsf{wait}, \mathsf{cut}\}$. The states of the automaton can then be described as follows:

- search: In this state the automaton is searching for the root of the special subtree.
- cut: This represents a branch not taken.

- wait and reach: In these states the automaton has entered the subtree and is looking for nodes labeled with p.
- visit: In this state the automaton has just visited a node with label p in the subtree.
- err: This is an error state that is entered if there is a label higher than p in the subtree.

The transition function δ is defined as follows: For all $\rho = (X, j, M_i) \in \Gamma$,

1. For $q \in \{\mathsf{cut}, \mathsf{err}\}$, $\delta(q, \rho) = \{(q, q)\}$.
2. For $q = \mathsf{search}$

$$\delta(q,\rho) = \begin{cases} \{(\mathsf{search}, \mathsf{cut}), (\mathsf{wait}, \mathsf{cut})\} & \text{if } X = \{0\} \\ \{(\mathsf{cut}, \mathsf{search}), (\mathsf{cut}, \mathsf{wait})\} & \text{if } X = \{1\} \\ \{(\mathsf{search}, \mathsf{cut}), (\mathsf{cut}, \mathsf{search}), (\mathsf{wait}, \mathsf{wait})\} & \text{if } X = \{0, 1\} \end{cases}$$

3. For $q \in \{\mathsf{wait}, \mathsf{reach}, \mathsf{visit}\}$, if $j > p$ then $\delta(q, \rho) = \{(\mathsf{err}, \mathsf{err})\}$, if $j = p$ then

$$\delta(q,\rho) = \begin{cases} \{(\mathsf{visit}, \mathsf{cut})\} & \text{if } X = \{0\} \\ \{(\mathsf{cut}, \mathsf{visit})\} & \text{if } X = \{1\} \\ \{(\mathsf{visit}, \mathsf{visit})\} & \text{if } X = \{0, 1\} \end{cases}$$

and if $j < p$ then

$$\delta(q,\rho) = \begin{cases} \{(\mathsf{reach}, \mathsf{cut})\} & \text{if } X = \{0\} \\ \{(\mathsf{cut}, \mathsf{reach})\} & \text{if } X = \{1\} \\ \{(\mathsf{reach}, \mathsf{wait}), (\mathsf{wait}, \mathsf{reach})\} & \text{if } X = \{0, 1\} \end{cases}$$

In the first stage, $\mathcal{A}_p$ guesses the location of the root of the special subtree T. While searching for this root, $\mathcal{A}_p$ remains in the state search. When it encounters the root, it enters the state wait for the first time. This starts the second stage, where $\mathcal{A}_p$ considers only marked nodes in T. In directions that correspond to a non-marked node, $\mathcal{A}_p$ moves to the state cut and remains there perpetually. From every marked node in T, $\mathcal{A}_p$ guesses a path to another marked node with label p, using the states wait and reach. It starts this search in state wait, moves to state reach immediately, remains there until it encounters a marked node with label p, and then moves to state visit. If there is no path from some node to another node with label p, all runs corresponding to the choice of T as subtree will eventually get stuck in reach. Thus, some run corresponding to T as the required subtree is accepting iff T satisfies the required conditions. ◀

▶ **Theorem 11.** *Let $\mathcal{L}$ be a library with width D, R be an exit control relation for $\mathcal{L}$, and α be an index function for $\mathcal{L}$. There exists a non-deterministic parity tree automaton (NPT) $\mathcal{B}$ such that, for all composers C over $\mathcal{L}$, $\mathcal{B}$ accepts $tree(C)$ iff C satisfies α and C is compatible with R. Consequently, $\mathcal{B}$ is non-empty iff $\mathcal{L}$ realizes α under R.*

Proof. We define $\mathcal{B} = \mathcal{B}_R \cap \mathcal{B}_\alpha$, where $\mathcal{B}_R$ is a safety tree automaton that accepts $tree(C)$ iff C is compatible with R, and $\mathcal{B}_\alpha$ is an NPT that accepts $tree(C)$ iff C satisfies α. Since the intersection of a safety automaton and an NPT is again an NPT, $\mathcal{B}$ is also an NPT.

Construction of $\mathcal{B}_R$: For simplicity, we define the automaton for the case $D = \{0, 1\}$, and note that the definition can be easily extended for arbitrary D. $\mathcal{B}_R = \{\mathcal{L}, \{\mathsf{start}\} \cup D, \mathsf{start}, \delta_R\}$, where δ_R is defined as follows: For all $M \in \mathcal{L}$,

- $\delta_R(\mathsf{start}, M) = \{(0, 1)\}$

- For $q \in D$, if $(q, M) \in R$ then $\delta_R(q, M) = \{(0, 1)\}$

Note that $\mathcal{B}_R$ has no transitions out of the states 0 and 1 iff the exit control relation R is violated. Thus $\mathcal{B}_R$ accepts $tree(C)$ iff C is compatible with R.

Construction of $\mathcal{B}_\alpha$: Let $\Gamma = LABELS(\mathcal{L})$ and let $\mathcal{A}_p = (\Gamma, Q, q_0, \delta, \beta)$ be the NBT defined in Lemma 10. We define $\mathcal{A}'_p = (\mathcal{L}, Q, q_0, \delta', \beta)$, where

$$\delta'(q, M_i) = \bigvee_{(X,j,M_i) \in LABELS(\mathcal{L})} \delta(q, (X, j, M_i))$$

While $\mathcal{A}_p$ accepts Γ-labeled D-trees, $\mathcal{A}'_p$ accepts $\mathcal{L}$-labeled D-trees. $\mathcal{A}'_p$ simply simulates $\mathcal{A}_p$ by using its larger transition function to guess the missing portion of the labels. We can characterize the regular trees accepted by $\mathcal{A}'_p$ as follows: for a composer C over $\mathcal{L}$, $\mathcal{A}'_p$ accepts $tree(C)$ iff there exists a choice function for C which has rank p.

Consider the automaton $\mathcal{A}'_\alpha$ whose language is the union of the language of each $\mathcal{A}'_p$, for all odd $p \leq \max(\alpha)$. Let C be a composer over $\mathcal{L}$. Then $\mathcal{A}'_\alpha$ accepts $tree(C)$ iff there exists a choice function for C that has an odd rank. Thus, by Theorem 7, $\mathcal{A}'_\alpha$ accepts $tree(C)$ iff C does not satisfy α. Finally, consider the automaton $\mathcal{B}_\alpha = \overline{\mathcal{A}'_\alpha}$, which is the complement of $\mathcal{A}'_\alpha$. Then $\mathcal{B}_\alpha$ accepts $tree(C)$ iff C satisfies α.

Since an NPT is nonempty iff it accepts a regular tree, and $\mathcal{L}$ realizes α under R iff some composer C over $\mathcal{L}$ satisfies α and C is compatible with R, therefore $\mathcal{B}$ is non-empty iff $\mathcal{L}$ realizes α under R. ◀

The NBT $\mathcal{A}'_p$ accepts $|D|$-ary trees and has $O(1)$ states, with an alphabet of size $|\mathcal{L}|$, so $\mathcal{A}'_\alpha$ is an NBT with $O(k)$ states, where $k = \max(\alpha)$. It follows that $\mathcal{B}_\alpha$ is a nondeterministic parity tree automaton (NPT) with $k^{O(k)}$ states and parity index $O(k)$ [12]. Also, $\mathcal{B}_R$ is a safety automaton with $O(|D|)$ states. Thus, their intersection $\mathcal{B}$ is an NPT with $|D|k^{O(k)}$ states and parity index $O(k)$, whose nonemptiness can be tested in time $|\mathcal{L}||D|^{O(k+|D|)}k^{O(k^2+k|D|)}$ [12]. We thus obtain the following:

▶ **Theorem 12.** *The embedded parity realizability problem is in EXPTIME.*

If an alternating tree automaton is nonempty, then it must accept some regular tree [12]. Given a regular tree accepted by $\mathcal{B}$, we can obtain a finite transducer that generates that tree. This transducer is a composer that realizes α under R. Thus, we also obtain a solution to the embedded parity synthesis problem.

▶ **Theorem 13.** *The embedded parity synthesis problem is in EXPTIME.*

The complexity of our solution is exponential in both k^2, where k is the highest parity index, as well as $|D|$, which is the number of exit states in each component. The exponential dependence on k is expected, as typical algorithms for solving parity games are exponential in the parity index, cf. [8]. Improving k^2 to k is an open challenge. It is also an open question whether the exponential dependence on $|D|$ can be avoided.

We remark that the embedded parity synthesis problem can be viewed as a 2-player partial information stochastic parity game. Informally, the game can be described as follows: The two players are the composer C and the environment E. The C player chooses components and the E player chooses paths through the components chosen by C. C cannot see the moves E makes inside a component. At the start C chooses a component M from the library $\mathcal{L}$. The turn passes to E, who chooses a sequence of inputs, inducing a path in M from its start state to some exit x in D. The turn then passes to C, which must choose some component M' in L and pass the turn to E and so on. As C cannot see the moves made by

E inside M, C cannot base its choice on the run of E in M, but only on the exit induced by the inputs selected by E and previous moves made by C. So C must choose the same next component M' for different runs that reach exit x of M. In general, different runs will visit different priorities inside M. This is a two-player stochastic parity game where one of the players does not have full information. If C has a winning strategy that requires a finite amount of memory, then we can use such a strategy to obtain a suitable finite composer that satisfies the index function α, thus solving the embedded parity synthesis problem. If C has no winning strategy or if every winning strategy requires infinite memory, then α is not realizable from the library L.

We also note that, when viewed in the framework of games, our result is a rare positive result for partial-information stochastic games. In general, 2-player partial information stochastic games are known to be undecidable even for co-Buchi objectives (and thus for parity objectives) [3].

5 Synthesis for DPW Specifications

Let A be a deterministic parity automaton (DPW), M be a probabilistic transducer and $\mathcal{L}$ be a library of components. We say A is a *monitor* for M (resp. $\mathcal{L}$) if the input alphabet of A is the same as the output alphabet of M (resp. $\mathcal{L}$). Let A be a monitor for M and let L_A be the language accepted by A. We say a strategy f for M is *winning* for the environment iff $\mu_f(L_A) < 1$, i.e., the output of M is rejected by A with positive probability. We say that M *satisfies* A if there exists no winning strategy for the environment.

▶ **Definition 14.** The *DPW probabilistic realizability problem* is: Given a library $\mathcal{L}$ and a DPW A that is a monitor for $\mathcal{L}$, decide whether there exists a composer C over $\mathcal{L}$, such that $\mathcal{T}_C$ satisfies A. If such a composer exists, we say that $\mathcal{L}$ *realizes* A. The *DPW probabilistic synthesis problem* is to find such a composer C if it exists.

We transform this problem into a version of the embedded parity problem solved in the previous section. Let $A = (\Sigma_O, Q_A, s_0, \delta_A, \alpha_A)$ be a DPW and $M = (\Sigma_I, \Sigma_O, Q_M, q_0, \delta_M, F, L)$ be a probabilistic transducer. For $s \in Q_A$, we denote by $M \times A_s$, the probabilistic transducer $(\Sigma_I, \Sigma_O, Q_M \times Q_A, (q_0, s), \delta, F \times Q_A, L')$, where $\delta((q, s'), a)(q', s'') = \delta_M(q, a)(q')$ if $s'' = \delta_A(s', L(q))$ and 0 otherwise. Given a library $\mathcal{L}$ with width D, we define the *augmented library* $\mathcal{L}_A = \{M \times A_s : M \in \mathcal{L}, s \in Q_A\}$. The width of $\mathcal{L}_A$ is $D \times Q_A$. We define the exit control relation $R_A \subseteq D \times Q_A \times \mathcal{L}_A$ for $\mathcal{L}_A$ as follows: for all $i \in D$, $s \in Q_A$, $M \in \mathcal{L}$, we have $(i, s, M \times A_s) \in R_A$. We also extend α_A to $\mathcal{L}_A$ as follows: for $(q, s') \in Q_M \times Q_A$, $\alpha_A(q, s') = \alpha_A(s')$. Thus α_A is an index function for $\mathcal{L}_A$.

Our first step is to treat this augmented library as a new library and solve the embedded parity synthesis problem for $\mathcal{L}_A$ with α_A as the index function and R_A as the exit control relation. This gives us a tree automaton that accepts $\mathcal{L}_A$-labeled $(D \times Q_A)$-trees and that is empty iff $\mathcal{L}_A$ does not realize α_A under R_A. Later, we show how to transform this automaton into another that accepts $\mathcal{L}$-labeled D-trees and is empty iff $\mathcal{L}$ does not realize A. Since, by definition, $\mathcal{L}_A$ bijectively maps to $\mathcal{L} \times Q_A$, we find it convenient to use labels from $\mathcal{L} \times Q_A$ in place of $\mathcal{L}_A$. We now define a composer for the augmented library. The states of the composer are pairs of the form (M, s), where s is a monitor state and M represents an instance of a component from $\mathcal{L}$. A *composer* for $\mathcal{L}_A$, is a deterministic transducer $C = (D \times Q_A, \mathcal{L} \times Q_A, \mathcal{M} \times Q_A, (\mathsf{M}, s), \Delta, \lambda)$. The following lemma follows directly from

Theorem 11[1].

▶ **Lemma 15.** *Let $\mathcal{L}$ be a library and A be a DPW that is a monitor for $\mathcal{L}$. There exists an NPT $\mathcal{B}$ that accepts a regular tree T iff $T = tree(C)$ for some composer C over $\mathcal{L}_A$ such that $\mathcal{T}_C$ satisfies α_A and C is compatible with R_A.*

Given a composer C over a library $\mathcal{L}$ and a monitor A for $\mathcal{L}$, we can extend C to a composer over the augmented library $\mathcal{L}_A$.

▶ **Definition 16** (Augmented Composer). Let $\mathcal{L}$ be a library and A be a monitor for $\mathcal{L}$. Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ be a composer over $\mathcal{L}$. The *augmentation* of C by A, denoted C_A, is a composer over $\mathcal{L}_A$ such that $C_A = (D \times Q_A, \mathcal{L} \times Q_A, \mathcal{M} \times Q_A, (\mathsf{M}_0, s_0), \Delta', \lambda')$, where

1. For all $s \in Q_A$, $\mathsf{M} \in \mathcal{M}$, $\lambda'(\mathsf{M}, s) = (\lambda(\mathsf{M}), s)$.
2. For all $i \in D$, $\mathsf{M} \in \mathcal{M}$ and $s, s' \in Q_A$, $\Delta((\mathsf{M}, s), (i, s')) = (\Delta(\mathsf{M}, i), s')$.

We say C_A is an augmented composer. While a composer only keeps track of the transfer of control between components, the augmented composer also keeps track of the state of the monitor before and after the control is transferred. To go from augmented composers to composers, we use techniques from synthesis with incomplete information [10]. We start by describing a relation between $tree(C)$ and $tree(C_A)$. First we need to introduce some convenient notation.

Let X, Y and Z be finite sets. For a Z-labeled $(X \times Y)$-tree $\langle T, V\rangle$, we denote by $xray(Y, \langle T, V\rangle)$, the $(Z \times Y)$-labeled $(X \times Y)$-tree $\langle T, V'\rangle$ in which each node is labeled by both its direction in Y and its labeling in $\langle T, V\rangle$. We define operators $hide_Y$ and $wide_Y$. The operator $hide_Y : (X \times Y)^* \to X^*$ replaces each letter $x \cdot y$, where $x \in X$ and $y \in Y$, by the letter x. The operator $wide_Y$ maps Z-labeled X-trees to Z-labeled $(X \times Y)$-trees as follows: $wide_Y(\langle X^*, V\rangle) = \langle (X \times Y)^*, V'\rangle$, where for each node $w \in (X \times Y)^*$, we have $V'(w) = V(hide_Y(w))$.

▶ **Lemma 17.** *Let $\mathcal{L}$ be a library and A be a monitor for $\mathcal{L}$. Let C be a composer over $\mathcal{L}$ and C_A be the augmentation of C by A. Then $tree(C_A) = xray(Q_A, wide_{Q_A}(tree(C)))$.*

▶ **Theorem 18.** *Let $\mathcal{L}$ be a library and A be a monitor for $\mathcal{L}$. Let C be a composer over $\mathcal{L}$ and C_A be the augmentation of C by A. Then C satisfies A iff C_A satisfies α_A.*

Given a library $\mathcal{L}$ and monitor A, we can solve the embedded realizability problem for the augmented library $\mathcal{L}_A$ to obtain a regular tree T, where $T = tree(C)$ for some composer C over $\mathcal{L}_A$ such that C satisfies α_A. Then the tree $T' = xray(Q_A, wide_{Q_A}(tree(C)))$ is also regular, so $T' = tree(C')$ for some composer C' over $\mathcal{L}$. Now we would like to use C' to solve the DPW realizability problem, but C' is only guaranteed to satisfy A if C is the augmentation of C' by A. Therefore, to solve the DPW realizability problem, we have to obtain an automaton that accepts a tree $T' = tree(C')$ if the augmentation of C' by A satisfies α_A.

▶ **Theorem 19.** *Let X, Y and Z be finite sets. Given an alternating automaton $\mathcal{B}$ over $(Z \times Y)$-labeled $(X \times Y)$-trees, we can construct an alternating automaton $\mathcal{B}'$ over Z-labeled X-trees such that $\mathcal{B}'$ accepts a labeled tree $\langle X^*, V\rangle$ iff $\mathcal{B}$ accepts $xray(Y, wide_Y(\langle X^*, V\rangle))$. Further, $\mathcal{B}$ and $\mathcal{B}'$ have the same acceptance condition and $|\mathcal{B}'| = O(|\mathcal{B}|)$.*

[1] Note that even with the slightly modified definition of composer, the results of the previous section still apply because a pair $(M, s) \in \mathcal{L} \times Q_A$ still uniquely identifies an element of $\mathcal{L}_A$.

Given an alternating automaton $\mathcal{B}$, let $narrow_Y(\mathcal{B})$ denote the corresponding automaton constructed in Theorem 19.

▶ **Theorem 20.** *Let $\mathcal{L}$ be a library and A be a monitor for $\mathcal{L}$. Then there exists an alternating parity tree automaton (APT) $\mathcal{B}$ such that, for all composers C over $\mathcal{L}$, $\mathcal{B}$ accepts $tree(C)$ iff C satisfies A. Consequently, $\mathcal{B}$ is non-empty iff $\mathcal{L}$ realizes A.*

Proof. Let $A = (\Sigma_O, Q_A, s_0, \delta_A, \alpha_A)$. Let $\mathcal{B}'$ be the NPT that accepts $tree(C')$ iff C' satisfies α_A and C' is compatible with R_A, for all composers C' over $\mathcal{L}_A$. Such a $\mathcal{B}'$ exists by Lemma 15. Let $\mathcal{B} = narrow_{Q_A}(\mathcal{B}')$. We show that $\mathcal{B}$, which is an APT, is the required automaton.

Let C be a composer over $\mathcal{L}$. By Theorem 18, C satisfies A iff C_A satisfies α_A. Therefore, $\mathcal{B}'$ accepts $tree(C_A)$ iff C satisfies A. By Lemma 17, $tree(C_A) = xray(Q_A, wide_{Q_A}(tree(C)))$, and by Theorem 19, $\mathcal{B}$ accepts a tree T iff $\mathcal{B}'$ accepts $xray(Q_A, wide_{Q_A}(T))$. Thus, $\mathcal{B}$ accepts $tree(C)$ iff C satisfies A. Since an APT is nonempty iff it accepts a regular tree, and $\mathcal{L}$ realizes A iff some composer C over $\mathcal{L}$ satisfies A, hence $\mathcal{B}$ is non-empty iff $\mathcal{L}$ realizes A. ◀

Each transducer in the augmented library $\mathcal{L}_A$ has a set of final states of size $|D||Q_A|$. Thus the automaton $\mathcal{B}'$ has size exponential in both $|D|$ and $|Q_A|$. The translation from $\mathcal{B}'$ to $\mathcal{B}$ adds no blowup, but $\mathcal{B}$ is an APT, while $\mathcal{B}'$ is an NPT. Since emptiness for an alternating parity tree automaton can be checked in time exponential in the size of the automaton [12], therefore $\mathcal{B}$ can be be checked for emptiness in time doubly exponential in $|D|$ and $|Q_A|$.

▶ **Theorem 21.** *The DPW probabilistic realizability problem is in 2EXPTIME.*

Again, if an alternating tree automaton is nonempty, then it must accept some regular tree [12], and given a regular tree accepted by $\mathcal{B}$, we can obtain a finite transducer that generates that tree. This transducer is a composer that realizes A. Thus, we also obtain a solution to the DPW probabilistic synthesis problem.

▶ **Theorem 22.** *The DPW probabilistic synthesis problem is in 2EXPTIME.*

The doubly exponential upper bound for our solution can be viewed as follows: we inherit one exponential from the embedded parity solution and the second exponential is introduced by the use of an APT to deal with incomplete information. It is an open question whether the second exponential can be avoided.

6 Discussion and Future Work

Component-based synthesis seeks to build systems that satisfy a given specification using pre-existing components. This contrasts with classical synthesis, where the aim is to build a system from scratch. The component-based approach is closer in spirit to how systems are built in the real world. In this paper, we generalize the component-based synthesis problem to a probabilistic setting. Our components are modeled as probabilistic transducers and the specification is given as a deterministic parity automaton. The composition itself is described by a deterministic transducer, called a *composer*, which governs the transitions between components.

We break the problem down in two stages. First we solve a simpler version, which we call the *embedded parity synthesis problem*, where the specification is embedded as parities in the components themselves. Our solution combines techniques from Markov chain analysis and automata theoretic verification. Then we show how to solve the more general case of a separate specification, which we call the *DPW probabilistic synthesis problem*, by reducing it to the simpler case using techniques from synthesis with incomplete information.

We show that the embedded parity synthesis problem is in EXPTIME and the DPW probabilistic synthesis problem is in 2EXPTIME. The question of tighter lower and upper bounds we leave for future work. In particular, it is an open question whether the DPW probabilistic synthesis problem is in EXPTIME. Another line of work is suggested by the possibility of probabilistic composers. While we do not know how to synthesize probabilistic composers, we do know that a direct reduction to the deterministic case will not work as probabilistic composers are more expressive.

References

1 C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski. Controller synthesis for probabilistic systems. In *Proc. IFIP TCS'04*, pages 493–506. Kluwer, 2004.

2 D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. ICSOC'03*, LNCS 2910, pages 43–58. Springer, 2003.

3 K. Chatterjee and L. Doyen. The complexity of partial-observation parity games. In *Proc. LPAR'10*, LNCS 6397. Springer, 2010.

4 K. Chatterjee, M. Jurdzinski, and T. A. Henzinger. Simple stochastic parity games. In *Proc. CSL'03*, LNCS 2803, pages 100–113. Springer, 2003.

5 C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. In *Proc. ICALP'90*, LNCS 443, pages 336–349. Springer, 1990.

6 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42:857–907, 1995.

7 L. de Alfaro and T.A. Henzinger. Interface-based design. In *Engineering Theories of Software-intensive Systems*, NATO Science Series: Mathematics, Physics, and Chemistry 195, pages 83–104. Springer, 2005.

8 M. Jurdzinski. Small progress measures for solving parity games. In *Proc. STACS'00*, LNCS 1770, pages 290–301. Springer, 2000.

9 J.G. Kemeny and J.L. Snell. *Finite Markov Chains.* Van Nostrad, 1960.

10 O. Kupferman and M.Y. Vardi. Synthesis with incomplete informatio. In *2nd Int. Conf. on Temporal Logic*, pages 91–106. Kluwer, 1997.

11 Y. Lustig and Moshe Y. Vardi. Synthesis from component libraries. In *Proc. FOSSACS'09*, LNCS 5504, pages 395 – 409. Springer, 2009.

12 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.

13 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

14 S. Schewe. Synthesis for probabilistic environments. In *Proc. ATVA'06*, LNCS 4218. Springer, 2006.

15 J. Sifakis. A framework for component-based construction extended abstract. In *Proc. 3rd Int. Conf. on Software Engineering and Formal Methods*, pages 293–300. IEEE, 2005.

16 M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. FOCS'85*, pages 327–338. IEEE, 1985.

17 M.Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *Formal Methods for Real-Time and Probabilistic Systems*, LNCS 1601, pages 265–276. Springer, 1999.