Synthesizing Probabilistic Composers[⋆]

Sumit Nain and Moshe Y. Vardi

Department of Computer Science,
Rice University, Houston, TX 77005, USA
{nain,vardi}@cs.rice.edu

Abstract. Synthesis from components is the automated construction of a composite system from a library of reusable components such that the system satisfies the given specification. This is in contrast to classical synthesis, where systems are always "constructed from scratch". In the *control-flow* model of composition, exactly one component is in control at a given time and control is switched to another when the component reaches an exit state. The composition can then be described implicitly by a transducer, called a *composer*, which *statically* determines how the system transitions between components.

Recently, Lustig, Nain and Vardi have shown that control-flow synthesis of deterministic composers from libraries of probabilistic components is decidable. In this work, we consider the more general case of probabilistic composers. We show that probabilistic composers are more expressive than deterministic composers, and that the synthesis problem still remains decidable.

Keywords: synthesis, temporal logic, probabilistic components.

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. 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 [11], and de Alfaro and Henzinger's work on "interface-based design" [7]. Furthermore, other situations, such as web-service orchestration [3], 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

[⋆] Work supported in part by NSF grants CCF-0728882 and CNS 1049862, and BSF grant 9800096. The authors are grateful to Orna Kupferman for suggesting the study of probabilistic composers.

(LTL) synthesis [10]. In LTL synthesis, the specification is given in LTL and the system constructed is a finite-state transducer modeling a reactive 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 [9] 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 [11].

Our model is based on the *control-flow composition* model of [9]. In this model, a component is just 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 contrast to [9], we allow components to be probabilistic, i.e., the transducers have a probabilistic transition function. The use of probabilistic transducers is a common 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. [5,6,12,13], 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.

In control-flow composition, control is held by a single component at every point in time. When the current component reaches an exit state, the execution passes to the start state of another component. The flow of control between components can itself be modeled by a supervisory transducer called a *composer*. Given the name of the current component and exit state, the composer gives the name of the next component in control. In essence, the composer describes how the composite system is put together and can be viewed as an implicit description of the composite system. The goal of the synthesis problem is then to find a suitable composer such that the resulting system satisfies the specification.

Here we consider two kinds of specification formalism: *embedded parity* and *deterministic parity automaton* (DPW). An embedded parity specification is given by associating a natural number called a priority to each state of each component in the given set of components. The specification is satisfied if the run of the system satisfies the parity condition with probability 1. A DPW specification is satisfied if the input-output behavior of the system is accepted by the DPW with probability 1.

In previous work [8], the synthesis problem for probabilistic components was shown to be decidable for both DPW and embedded parity specifications. However, while the components there were probabilistic, the composer, and thus the control-flow of the composite system, was still deterministic. Since the composer is itself a transducer that describes the flow of control between components in a composition, it is natural to consider the more general case of allowing the composer to also be a probabilistic transducer. In this case, not just the behavior of individual components, but also the flow of control between components is probabilistic. Does this allow the composite system to satisfy more specifications? That is, we would like to know whether probabilistic composers are more *expressive* than deterministic composers. And if so, how do we synthesize them? These questions are the focus of this paper.

We have two main goals: to investigate the expressiveness of probabilistic composers and to solve the synthesis problem for probabilistic composers. We show that expressive power depends on the type of specification. For embedded parity specifications, allowing the composer to be probabilistic gives no advantage. In particular, if a suitable probabilistic composer exists then a suitable deterministic composer also exists. In contrast, for the more general case of a DPW specification, we find that probabilistic composers are more expressive. We give an instance of the DPW synthesis problem such that there exists a suitable probabilistic composer that solves it, but no suitable deterministic composer exists. We note that expressiveness is not just a theoretical concern. The fact that probabilistic composers are more expressive means that some systems can only be constructed using a probabilistic composition. As a result the DPW synthesis problem for probabilistic composers becomes important in its own right and not just as an extension of the deterministic case. It is interesting that probabilistic composers only gain expressive power in the presence of specifications with memory (DPW specifications). We view this as another example of a memory vs. randomness trade-off that is well-known in the game-theoretic literature [4].

In [8], the DPW synthesis problem for deterministic composers is solved by reducing it to the embedded parity version. But for probabilistic composers, a similar approach does not work because their expressive power differs for embedded parity and DPW specifications. Instead, we solve the DPW synthesis problem for probabilistic composers by a reduction to deterministic composers. The key insight is that it is possible to equip the library with an additional component with a specific structure, called M_{rand}, such that the probabilistic choices made by a composer can be simulated by the probabilistic transitions within M_{rand}. The idea is that whenever a probabilistic composer C makes a probabilistic transition, the equivalent deterministic composer C' can instead call an instance of M_{rand} to simulate the moves of C. The result is that the DPW synthesis problem for probabilistic composers is decidable.

Finally, the embedded-parity version of our synthesis problem can also be viewed as a partial information stochastic game between two players, the composer C, which chooses components, and the environment E, which chooses paths through the components chosen by C. The partial information arises because our model of composition is static, which means the components chosen by the composer cannot depend on the inputs selected by the environment. In contrast to standard parity games, partial information stochastic games are known to be undecidable even for co-Büchi objectives (and thus for parity objectives) [2]. Thus, in the framework of games, our result can be viewed as a rare positive result for partial-information stochastic games. We explain this game-theoretic view of the problem in more detail in Section 7.

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

2 Preliminaries

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.

A *strongly connected component* of a directed graph $G = (V, E)$ 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: $U_1 \le U_2$ if there exists $u_1 \in U_1$ and $u_2 \in U_2$ such that u_1 is reachable from u_2. An *ergodic set* of G is a minimal element of this partial order.

A probability distribution on a finite set X is a function $\mu : X \to [0,1]$ such that $\sum_{x\in X} \mu(x) = 1$. The *support* of μ, denoted *supp*(μ), is the set $\{x \in X : \mu(x) > 0\}$. $Dist(X)$ denotes 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 is 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.

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, and a set of states X, we denote by $\alpha(X)$ the highest priority assigned by α in X.

3 Control-Flow Composition

Let $\mathcal{L}$ be a library with width D. We first informally describe our notion of probabilistic control-flow composition of components from $\mathcal{L}$. Each library component can be used multiple times in a composition, and we treat these occurrences as distinct *component instances*. Thus, the size of a composition, a priori, is not bounded. The component instances in a composition take turns interacting with the environment, and at each point in time, exactly one component instance is active. When the active component instance reaches an exit state, control is transferred probabilistically to the start state of some other component instance.

Given a set of component instances to be composed, a control flow composition can be defined by giving, for each exit state of each component instance, the probability distribution that determines the probability of transitioning from that exit state to another component instance. This information can be represented naturally by a probabilistic transducer, called a (probabilistic) *composer*, whose set of states corresponds to the set of component instances and whose input alphabet corresponds to the set of exits D. Then, for each component instance and exit, the transition function of the composer gives the probability distribution that determines which component instance will be in control next.

Formally, a *composer* over a library $\mathcal{L}$ with width D is a probabilistic tranducer $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$. Here $\mathcal{M}$ is an arbitrary finite set of states. In particular, there is no a priori 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. The transition function $\Delta : \mathcal{M} \times D \to Dist(\mathcal{M})$ gives a distribution on the set of instance names. We say a composer is a *deterministic composer* if its transition function is deterministic.

Note that while each component name M_i is distinct, the corresponding component instances $\lambda(\mathsf{M}_i)$ 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: $\Delta(\mathsf{M}, i)(\mathsf{M}') = p$ denotes that when the composition is in the ith exit state of component instance $\lambda(\mathsf{M})$ it moves to the start state of component instance $\lambda(\mathsf{M}')$ with probability p. A composer can be viewed as an implicit representation of a composition. We give the 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 $q = q_x^i \in F_i$, then $\delta(\langle q, i\rangle, \sigma)(\langle q', j\rangle) = \Delta(\mathsf{M}_i, x)(\mathsf{M}_k)$.*

When the composition is in a state $\langle q, i\rangle$ corresponding to a non-exit state q of component instance $\lambda(\mathsf{M}_i)$, it behaves like $\lambda(\mathsf{M}_i)$. When the composition is in a state $\langle q_f, i\rangle$ corresponding to an exit state q_f of component instance $\lambda(\mathsf{M}_i)$, the control is transferred to the start state of another component instance 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. 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.

Definition 2. *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 $(i, \mathsf{M}') \in R$. Thus, each element of R can be viewed as a constraint on how the composer is allowed to connect components.*

4 Defining the Synthesis and Realizability Problems

The goal of synthesis from components is to find a composer C over library $\mathcal{L}$ such that $\mathcal{T}_C$, the composition defined by C, satisfies the given specification, which is typically some ω-regular property. In general, a specification property is usually defined as a subset of $(\Sigma \times Q)^\omega$ where Σ is the input alphabet and Q is the set of states of the system. However, here we assume, without loss of generalization, that the states of the transducers 'remember' the input, so defining a property as a subset of Q^ω is sufficient. In this paper, we focus on two different but related formalisms for describing ω-regular properties:

- *embedded parity specification*: This is a simple specification given by an *index function* α, which assigns a *priority* to each state of the system. We say a run of the system satisfies the *parity condition* if the highest priority visited infinitely often is even. Then the ω-regular property defined by the specification is just the set of all runs that satisfies the parity condition.
- *deterministic parity word automata* (DPW): This is a more powerful formalism that can express all ω-regular specifications. Given a DPW A, the ω-regular property defined by A is simply the language of A.

While the two formalisms differ in power, they ultimately both use the parity condition to check whether a run of the system is accepting or not. As we see in Section 5, the contrast between the relatively weak embedded parity specification and the more general DPW specification is a useful tool to study the expressiveness of probabilistic composers. In particular, we note that a DPW specification has memory while an embedded parity specification is memoryless. As we see in the next section, this difference illuminates the issue of expressiveness of probabilistic composers.

Now, in order to formalize the synthesis problem, we need to define an appropriate notion of correctness for a reactive probabilistic system w.r.t. a given specification. We assume the presence of an adversarial environment that controls the input, while the system, a probabilistic transducer, controls the output. We require that the executions of the system satisfy the specification with probability 1 irrespective of the inputs selected by the environment. Our notion of correctness is *qualitative* [12].

Definition 3. *Let M be a probabilistic transducer, Q be the state space of M, and $P \subseteq Q^\omega$ be an ω-regular property. Let f be a strategy for M. Then f is* winning *for the environment if $\mu_f(P) < 1$. We say that M* satisfies *P if there exists no winning strategy for the environment. If M satisfies P, and the property P is given by index function α or DPW A, we say M satisfies α or, respectively, M satisfies A.*

Let $\mathcal{L}$ be a library, α be an index function, and A be a DPW. Let C be a composer over $\mathcal{L}$. We say C satisfies α or C satisfies A, if $\mathcal{T}_C$ satisfies α or, respectively, $\mathcal{T}_C$ satisfies A.

The two types of specifications give rise to two related synthesis problems.

Definition 4. *The* embedded parity realizability problem for probabilistic composers *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 probabilistic composer C over $\mathcal{L}$, such that $\mathcal{T}_C$ satisfies α and C is compatible with R. If such a composer exists we say $\mathcal{L}$* realizes α *under R. The* embedded parity synthesis problem for probabilistic composers *is to find such a composer C if it exists.*

Definition 5. *The* DPW realizability problem for probabilistic composers *is: Given a library $\mathcal{L}$ and a DPW specification A, decide whether there exists a probabilistic 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 synthesis problem for probabilistic composers *is to find such a composer C if it exists.*

We can obtain weaker versions of these problems by restricting ourselves to deterministic composers. The resulting problems are then known to be decidable:

Theorem 1. [8] *The embedded parity realizability and synthesis problems for deterministic composers are decidable.* □

Theorem 2. [8] *The DPW realizability and synthesis problems for deterministic composers are decidable.* □

We observe that while one might hope to solve the more general case of probabilistic composers by using the methods of [8], there are two main difficulties with that approach. First, in [8], the DPW version of the problem is solved by reducing it to the embedded parity version. However, as we show in the next section, while deterministic and probabilistic composers have the same expressive power for embedded parity specifications, probabilistic composers are strictly more expressive than deterministic composers for DPW specifications. Thus it is not possible to reduce the DPW realizability of probabilistic composers to the embedded parity version. Second, the automata theoretic techniques used in [8] crucially depend on the fact that every deterministic composer can be represented as a regular D-tree (a tree with constant branching degree D) where D is the width of the library. This is because when the control-flow is deterministic, there is exactly one successor component for each exit of a component, and so the number of outgoing edges from each component is always the same as the number of exits. When the composer is probabilistic, the branching degree of its unfolding can be as large as the number of its states, which is not a priori bounded.

5 The Expressive Power of Probabilistic Composers

Given a specification formalism, it is natural to ask the following question: Do we gain any additional power for solving the synthesis problem by allowing composers to be probabilistic? That is, are there instances of the synthesis problem (i.e. a library and a specification) such that some probabilistic composer satisfies the specification, but no deterministic composer does? If the answer is yes, then we say that probabilistic composers are more expressive than deterministic composers for that class of specifications. Otherwise, we say they are equally expressive.

5.1 Embedded Parity Specifications

We first consider the issue of expressiveness for embedded parity specifications. Our main result here is that deterministic and probabilistic composers are both equally expressive for embedded parity specifications. To prove this, we need to show that, for every library $\mathcal{L}$, exit control relation R and index function α, if there is a probabilistic composer over $\mathcal{L}$ that satisfies α under R, then we can also find a deterministic composer over $\mathcal{L}$ that satisfies α under R. We first recall the following useful characterization of winning strategies.

Theorem 3. [8] *Let M be a probabilistic transducer and α be an index function.*

1. *If there exists a winning strategy for the environment then there exists a pure and memoryless winning strategy.*
2. *Let 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.* $\square$

The key idea behind our approach is that instead of directly comparing probabilistic composers to deterministic composers, we should instead compare a probabilistic composer C to a composer C' which is 'less probabilistic' than it. We formalize what it means for one composer to be less probabilistic then another, by introducing a partial order, denoted $\leq_{\text{prob}}$, on the set of all composers.

Definition 6. *Let $\mathsf{COMP}(\mathcal{L})$ be the set of all probabilistic composers over $\mathcal{L}$. We define the partial order $\leq_{prob}$ on $\mathsf{COMP}(\mathcal{L})$: for $C_1 = (D, \mathcal{L}, \mathcal{M}_1, \mathsf{M}_0^1, \Delta_1, \lambda_1)$ and $C_2 = (D, \mathcal{L}, \mathcal{M}_2, \mathsf{M}_0^2, \Delta_2, \lambda_2)$, we have $C_1 \leq_{prob} C_2$ if*

- *$\mathcal{M}_1 = \mathcal{M}_2$, $\mathsf{M}_0^1 = \mathsf{M}_0^2$, and $\lambda_1 = \lambda_2$*
- *$\forall i \in D, \mathsf{M} \in \mathcal{M}_1$, $supp(\Delta_1(\mathsf{M}, i)) \subseteq supp(\Delta_2(\mathsf{M}, i))$*

If $C_1 \leq_{prob} C_2$ and $C_2 \leq_{prob} C_1$, we say that C_1 and C_2 are qualitatively equivalent*. We denote by $<_{prob}$ the strict partial order corresponding to $\leq_{prob}$.*

Thus $C_1 <_{\text{prob}} C_2$ if they have the same set of states and output functions and the set of possible transitions of C_1 is a proper subset of the set of possible transitions of C_2. We note that $<_{\text{prob}}$ is a well-founded relation, and deterministic composers are the minimal elements of the relation. The well-foundedness of $<_{\text{prob}}$ is crucial because it allows the use of induction.

Next we show that if all the composers obtained by removing transitions from a composer C, fail to satisfy α under R, then C itself also fails to satisfy α under R. The intuition here is that the behavior of C can be determined by looking at the behavior of composers that make fewer probabilistic choices than C but have the same underlying structure.

Lemma 1. *Let $\mathcal{L}$ be a library with width D, R be an exit control relation and α be an index function for $\mathcal{L}$, and let $C \in \mathsf{COMP}(\mathcal{L})$ be such that C is not deterministic. Suppose that for all $C' \in \mathsf{COMP}(\mathcal{L})$ and $C' <_{prob} C$, we have that C' does not satisfy α under R. Then C also does not satisfy α under R.*

Proof. Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda) \in \mathsf{COMP}(\mathcal{L})$. If C is not compatible with R, then C trivially does not satisfy α under R. So we assume that C is compatible with R. We first arbitrarily choose $\mathsf{M}_a \in \mathcal{M}$ and $i_0 \in D$ such that

$|supp(\Delta(\mathsf{M}_a, i_0))| > 1$. That is, we require that the transition out of state M_a on input i_0 is not deterministic. Since it is given that C is not a deterministic composer, there is at least one such state and input pair. Consider M_a and i_0 to be fixed for the rest of this proof.

Let $supp(\Delta(\mathsf{M}_a, i_0)) = \{\mathsf{M}_1, \ldots, \mathsf{M}_k\} \subseteq \mathcal{M}$. For $1 \le j \le k$, we define probabilistic composers $C_j = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta_j, \lambda)$, where $\Delta_j(\mathsf{M}_a, i_0) = \mathsf{M}_j$ and for all $\mathsf{M} \in \mathcal{M}$, $i \in D$, $\Delta_j(\mathsf{M}, i) = \Delta(\mathsf{M}, i)$ when $i \neq i_0$ or $\mathsf{M}_a \neq \mathsf{M}$. Thus each C_j is deterministic at state M_a and behaves exactly like C at other states. Further, for each possible choice of next state available to C at state M_a on input i_0, there is exactly one of the C_j's that makes that choice deterministically. By construction, for all $1 \le j \le k$, we have $C_j \in \mathsf{COMP}(\mathcal{L})$, $C_j < C$, and C_j is compatible with R. So, by the assumption in the theorem statement, C_j does not satisfy α for all $1 \le j \le k$. Then, by Definition 3 and Theorem 3, for each $1 \le j \le k$, there exists memoryless strategy f_j for $\mathcal{T}_{C_j}$ that is winning for the environment. Note that each f_j is also a memoryless strategy for $\mathcal{T}_C$.

Let Q be the set of states of $\mathcal{T}_C$, q_0 be the start state of $\lambda(\mathsf{M}_0)$ and q be the exit state of $\lambda(\mathsf{M}_a)$ in direction i_0. For $1 \le j \le k$, let $G_j = G_{\mathcal{T}_C, f_j}$ and $G'_j = G_{\mathcal{T}_{C_j}, f_j}$. Now, by construction, we have $\Delta_j(\mathsf{M}, i) = \Delta(\mathsf{M}, i)$ for $i \neq i_0$ or $\mathsf{M} \neq \mathsf{M}_a$. That is, C and C_j differ in their choices only when the composition is in exit state q of component $\lambda(\mathsf{M}_a)$. Further, G_j and G'_j have the same set of vertices Q and G'_j is a subgraph of G_j. Thus all edges that lie in G_j but not in G'_j must have q as their source. Let X_j be a reachable ergodic set of G'_j such that $\alpha(X_j)$ is odd. Such an X_j must exist because f_j is winning for the environment for $\mathcal{T}_{C_j}$. Then, since G'_j is a subgraph of G_j, X_j is also reachable and strongly connected in G_j. Also q is the source of all the edges that lie in G_j but not in G'_j. So if q does not lie in X_j, then no edges can leave X_j in G_j and X_j is also a reachable ergodic set of G_j. In this case, f_j is also a winning strategy for the environment for $\mathcal{T}_C$. Note that this argument does not depend on the particular value of j. For the rest of the proof we can assume that $q \in X_j$ for all $1 \le j \le k$.

Let $X = \bigcup_{j=1}^{k} X_j$ and $x \in X$ be such that $\alpha(x) = \alpha(X)$. Since, by definition, $\alpha(X_j)$ is odd for all $1 \le j \le k$, therefore $\alpha(x)$ is also odd. We assume, without loss of generalization, that $x \in X_1$ and define a memoryless strategy f for $\mathcal{T}_C$ such that: $f(x) = f_1(x)$ for $x \in (Q - X) \cup X_1$, and $f(x) = f_j(x)$ for $x \in X_j - \bigcup_{i=1}^{j-1} X_i$. Let $G = G_{\mathcal{T}_C, f}$. Since f takes the same values as f_1 on X_1, then X_1 must be strongly connected in G. We first show that no edges in G leave X. Since X_1 is an ergodic set of G_1, therefore q is the source of all the edges that leave X_1 in G. By construction of the C_j, each of these edges goes to some X_j. Thus no edges from X_1 leave X in G. Similarly, any edge leaving a vertex $x \in X_j - \bigcup_{i=1}^{j-1} X_i$ must stay in X_j, because f and f_j agree on those vertices and $q \notin X_j - \bigcup_{i=1}^{j-1} X_i$. Thus there are no edges leaving X in G. Thus X must contain at least one ergodic set of G. Let Y be this ergodic set of G.

We next show that X_1 is reachable in G from every vertex in X. Clearly X_1 is reachable in G from X_1. Assume that X_1 is reachable in G from every vertex in $X_m - \bigcup_{i=1}^{m-1} X_i$, for all $1 < m \le j$, for some $j < k$. Let $X' = X_{j+1} - \bigcup_{i=1}^{j} X_i$ and let $x \in X'$. We claim that there is a path in G that starts from x and leaves

X'. If there is no such path, then some $Y' \subseteq X'$ must be an ergodic set of G. Since f and f_{j+1} agree on vertices in X', if Y' is an ergodic set of G, then Y' is also an ergodic set of G_{j+1}. But this contradicts the fact that X_{j+1} is an ergodic set of G'_{j+1}, and so is strongly connected in G_{j+1}. Thus there is a path in G that starts from x and leaves X'. Further, any outgoing edge from a vertex in X' cannot leave X_{j+1}, because f and f_{j+1} agree on X'. Thus there is a path from x to some vertex y in $X_{j+1} - X' = \bigcup_{i=1}^{j} X_i$. Now, by the inductive hypothesis, X_1 is reachable in G from y. Thus X_1 is also reachable in G from x. Since x was chosen arbitrarily, X_1 is reachable in G from every vertex in $X_{j+1} - \bigcup_{i=1}^{j} X_i$. By induction, X_1 is reachable in G from every vertex in X. In particular, X_1 is reachable in G from the ergodic set $Y \subseteq X$ of G. Thus, X must be contained in Y. Then we have $\alpha(x) \leq \alpha(X_1) \leq \alpha(Y) \leq \alpha(X) = \alpha(x)$. Thus $\alpha(Y)$ is also odd.

Finally, all that remains is to show that Y is reachable from the start state q_0. Since f_1 is winning for the environment for $\mathcal{T}_{C_1}$, we know that X_1 is reachable in G_1 from the start state q_0. Since $X_1 \subseteq X$, X is also reachable in G_1 from q_0. Consider the shortest path in G_1 that starts from q_0 and reaches X. Then all vertices, except the last one, in this path are in $Q - X$. Since f and f_1 agree on vertices in $Q - X$, this path is also present in G. Then X is reachable from q_0 in G and so Y is reachable from q_0 in G. Thus f is a pure and memoryless winning strategy for the environment. □

Finally, we show that, for embedded parity specifications, probabilistic composers are not more expressive than deterministic composers.

Theorem 4. *Let $\mathcal{L}$ be a library with width D and α be an index function for $\mathcal{L}$. There is a probabilistic composer $C \in \mathsf{COMP}(\mathcal{L})$ that satisfies α if and only if there is a deterministic composer $C' \in \mathsf{COMP}(\mathcal{L})$ that satisfies α.*

Proof. Follows immediately from Lemma 1 using transfinite induction on the well-founded strict partial order $<_{\text{prob}}$. □

As a consequence of Theorem 4, together with Theorem 1, we obtain:

Theorem 5. *The embedded parity synthesis problem for probabilistic composers is decidable.* □

5.2 DPW Specifications

While embedded parity specifications are memoryless, DPW specifications have an associated memory (the state of the DPW). This difference turns out to be crucial in determining the expressive power of probabilistic composers. As we see below, the ability to make random transitions allows a probabilistic composer to successfully deal with memoryful specifications where deterministic composers fail. We find that, in contrast to the embedded parity case, there are instances of the DPW realizability problem where a suitable probabilistic composer exists but no suitable deterministic composer exists. Thus, probabilistic composers are strictly more expressive than deterministic composers for DPW specifications.

We describe a suitable problem instance, consisting of a library $\mathcal{L}$ and DPW A. Let $\Sigma = \{a, b, c, b', c'\}$ and A be a DPW that accepts a word over Σ iff it contains at least one occurrence of bab' or cac'. The language of A is $\Sigma^*(bab' + cac')\Sigma^\omega$. Consider the library $\mathcal{L} = \{M_1, M_2, M_3\}$ consisting of the three components M_1, M_2 and M_3, which are shown in Fig. 1 (the figure depicts a composition built using a single instance of each component). Each component in the library has a single exit state. The input alphabet of each component is $\{0, 1\}$ and the output alphabet is Σ. The components M_2 and M_3 each consist of a single state, which serves as both the start and exit state. As a result, they have no internal transitions. They are only distinguished by the output in their single state. M_2 outputs b' and M_3 outputs c'. The component M_1 has four states and its transition function is such that every run from its start state to its exit state always deterministically produces an output of either aba or aca, depending solely on the input selected by the environment in the start state.

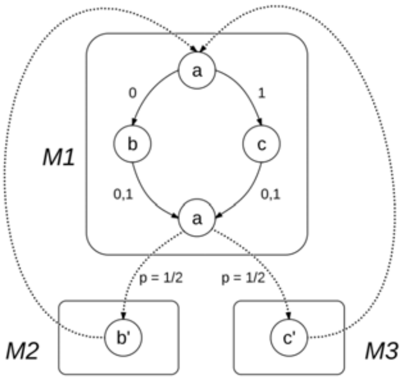

Fig. 1. A composition with probabilistic control-flow that satifies $\Sigma^*(bab' + cac')\Sigma^\omega$

Now consider a composition built from components in $\mathcal{L}$ that is defined by a deterministic composer. Since each component in $\mathcal{L}$ has exactly one exit, if the composer is deterministic, then the composition can be viewed as a linear sequence of components, i.e., a pipeline. We claim that in this situation, the environment can always prevent bab' or cac' from occurring anywhere in the output. This is because bab' can only be output if an instance of M_2 occurs immediately after an instance of M_1 in the pipeline, but in any such case, the environment can always force that particular instance of M_1 to output aca instead of aba. Similarly, the environment can always prevent cac' from being output. The result is that no deterministic composer over $\mathcal{L}$ can satisfy A.

Theorem 6. *Let $\mathcal{L}$ and A be as defined above. Then there does not exist a deterministic composer over $\mathcal{L}$ that satisfies A.* □

In contrast to the deterministic case, there is a probabilistic composer over $\mathcal{L}$ that satisfies A. Intuitively, the composer needs to overcome the fact that the environment has complete control over the output of M_1. It can do this by probabilistically connecting each instance of M_1 to instances of both M_2 and M_3. Then the control that the environment has over the output of M_1 becomes irrelevant. One such composition is shown in Fig. 1, where control is transfered from the single exit state of M_1 to either M_2 or M_3 with equal probability.

Theorem 7. *Let $\mathcal{L}$ and A be as defined above. Then there exists a probabilistic composer over $\mathcal{L}$ that satisfies A.* □

6 Synthesizing Probabilistic Composers

In the previous section, we saw that probabilistic composers are more expressive than deterministic composers for DPW specifications, but both have the same expressive power for embedded parity specifications. This unfortunately rules out following the approach of [8] to solve the DPW synthesis problem for probabilistic composers by reducing it to the embedded parity version. Instead, we show that the DPW synthesis problem for probabilistic composers can be reduced to the DPW synthesis problem for deterministic composers. Since, by Theorem 2, the latter problem is decidable, this suffices to solve the probabilistic version too.

The key idea behind our reduction is that probabilistic choices made by a composer can be simulated with the help of a component with a specific structure. Consider a component, called M_{rand}, which ignores the environment's input and transitions uniformly at random from its start state to each of its two exit states. Now suppose that a composer C over $\mathcal{L}$ probabilistically calls two different components, say M_1 and M_2. Then this behavior can be simulated by a deterministic composer, say C', that first calls M_{rand}, and then calls M_1 and M_2 from the two exits of M_{rand}. In this way, we can replace a probabilistic composer over $\mathcal{L}$ by a deterministic composer over the larger library $\mathcal{L} \cup \{M_{\text{rand}}\}$. We first formally define the special component M_{rand}.

Definition 7. *Let Σ_I and Σ_O be the input and output alphabets of every component in $\mathcal{L}$. Let b be a fresh output symbol not contained in Σ_O. We define the probabilistic transducer $M_{rand} = \{\Sigma_I, \{b\}, Q, q_0, \delta, F, L\}$, where $F = \{q_1, q_2, \ldots, q_{|D|}\}$, $Q = \{q_0\} \cup F$, $L(q) = b$ for all $q \in Q$, and $\delta(q_0, a)(q) = 1/|D|$ for all $a \in \Sigma_I$ and $q \in F$.*

So M_{rand} has $|D|$ final states and $|D| + 1$ total states, it outputs b in every state, and it transitions with uniform probability from the start state to a final state irrespective of the input. Note that M_{rand} is not a component in $\mathcal{L}$ since $b \notin \Sigma_O$ by construction. If we add M_{rand} to $\mathcal{L}$ to obtain a larger library, we also have to translate DPW specifications for $\mathcal{L}$ into specifications for the larger library. The idea is to modify the DPW to ignore the output of M_{rand}.

Definition 8. *Let $A = (\Sigma_O, Q_A, s_0, \delta_A, \alpha_A)$ be a DPW specification for $\mathcal{L}$. We define the DPW $A^b = (\Sigma_O \cup \{b\}, Q_A \times \{0, 1\}, (s_0, 0), \delta^b_A, \alpha^b_A)$, where $\alpha^b_A(q, 0) = \alpha_A(q)$ and $\alpha^b_A(q, 1) = 1$, and δ^b_A is defined as follows:*

– *For* $a \in \Sigma_O$*,* $i \in \{0,1\}$ *and* $s \in Q_A$*,* $\delta^b_A((s,i),a) = (\delta_A(s),0)$
– *For* $i \in \{0,1\}$ *and* $s \in Q_A$*,* $\delta^b_A((s,i),b) = (s,1)$

Thus A^b ignores b and behaves like A on other inputs. A^b accepts a word w iff A accepts w' where w' is the result of removing all occurences of b in w. Note that A^b is a DPW specification for the library $\mathcal{L}' = \mathcal{L} \cup \{M_{\text{rand}}\}$.

We now reduce the problem of finding a probabilistic composer over $\mathcal{L}$ that satisfies A to the problem of finding a deterministic composer over $\mathcal{L}'$ that satisfies A^b. We give a mapping that transforms a deterministic composer C over $\mathcal{L}'$ to a probabilistic composer *prob*(C) over $\mathcal{L}$. The intuition behind the mapping is that C uses multiple instances of M_{rand} to simulate the probabilistic choices made by *prob*(C), such that C and *prob*(C) have the same behaviour.

Definition 9. *Let* $\mathcal{L}' = \mathcal{L} \cup \{M_{rand}\}$ *and* $C = (D, \mathcal{L}', \mathcal{M}, \mathsf{M}_1, \Delta, \lambda)$ *be a deterministic composer over* $\mathcal{L}'$*. Let* $\mathcal{M} = \mathcal{M}_0 \cup \mathcal{M}_{rand}$ *where for all* $\mathsf{M} \in \mathcal{M}_0$*,* $\lambda(\mathsf{M}) \neq M_{rand}$ *and for all* $\mathsf{M} \in \mathcal{M}_{rand}$*,* $\lambda(\mathsf{M}) = M_{rand}$*. Then* *prob*$(C) = (D, \mathcal{L}, \mathcal{M}_0, \mathsf{M}_1, \Delta', \lambda)$ *is the probabilistic composer over* $\mathcal{L}$*, whose probabilistic transition function* Δ' *is defined as follows: For all* $\mathsf{M} \in \mathcal{M}_0$ *and* $i \in D$*,*

– $\Delta'(\mathsf{M}, i)$ *is a uniform distribution on its support*
– *For all* $\mathsf{M}' \in \mathcal{M}_0$*,* $\Delta'(\mathsf{M}, i)(\mathsf{M}') > 0$ *if there is a finite run of* C *that starts in* M*, ends in* M'*, and visits only states in* $\mathcal{M}_{rand}$*.*

Note that the mapping *prob* is not reversible, and given a probabilistic composer C over $\mathcal{L}$, there might not be a deterministic composer C' over $\mathcal{L}'$ such that C = *prob*(C'). However, if we partition the set of all composers over $\mathcal{L}$ by qualitative equivalence (see Defn. 6), then we obtain a reversible mapping. We use this reversible mapping to show that the synthesis of probabilistic composers is reducible to the synthesis of deterministic composers over a larger library.

Lemma 2. *Let* C' *be a deterministic composer over* $\mathcal{L} \cup \{M_{rand}\}$ *and let* C *be a probabilistic composer over* $\mathcal{L}$ *such that* C *is qualitatively equivalent to* *prob*(C')*. Then* C *satisfies* A *iff* C' *satisfies* A^b*.* □

Theorem 8. *The DPW synthesis problem for probabilistic composers is decidable.* □

7 Discussion

In the framework of parity games, the embedded parity version of our synthesis problem can be viewed as a 2-player stochastic game with partial information; that is, one player cannot see the moves of the other player in full. Informally, the game is the following: We are given a library $\mathcal{L}$ of n components each with D exits with index function α. The two players are the composer C and the environment E. Player C chooses components and player E chooses paths through the components chosen by C. However, C cannot see the moves E makes inside a component. At the start C chooses a component M from $\mathcal{L}$. The turn passes to E, who chooses a sequence of inputs, inducing a path in M from its start state to some exit x of M. The turn then passes to C, which must choose some component M' in $\mathcal{L}$ and pass the turn to E. 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 problem. If C has no winning strategy or if every winning strategy requires infinite memory, then α is not realizable from the library $\mathcal{L}$.

In contrast to standard parity games, partial information 2-player stochastic parity games are known to be undecidable in general [2]. Thus, when viewed in the framework of games, our result is a rare positive result for partial-information stochastic games. Since the general problem is undecidable, the best result one can hope for is to show that some restricted but useful class of partial information parity games is decidable. Our result on the embedded parity synthesis problem can be viewed as just such a result.

References

1. Baier, C., Größer, M., Leucker, M., Bollig, B., Ciesinski, F.: Controller synthesis for probabilistic systems. In: Proc. TCS 2004, pp. 493–506. Kluwer (2004)
2. Baier, C., Bertrand, N., Größer, M.: On Decision Problems for Probabilistic Büchi Automata. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 287–301. Springer, Heidelberg (2008)
3. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic Composition of *E*-services That Export Their Behavior. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 43–58. Springer, Heidelberg (2003)
4. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: Trading memory for randomness. In: QEST 2004, pp. 206–217. IEEE Computer Society (2004)
5. Courcoubetis, C., Yannakakis, M.: Markov Decision Processes and Regular Events. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 336–349. Springer, Heidelberg (1990)
6. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. Journal of the ACM 42, 857–907 (1995)
7. de Alfaro, L., Henzinger, T.A.: Interface-based design. In: Engineering Theories of Software-Intensive Systems, pp. 83–104. Springer, Heidelberg (2005)
8. Lustig, Y., Nain, S., Vardi, M.Y.: Synthesis from probabilistic components. In: Proc. CSL 2011. LIPICS, vol. 12, pp. 412–427 (2011)
9. Lustig, Y., Vardi, M.Y.: Synthesis from Component Libraries. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 395–409. Springer, Heidelberg (2009)
10. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. POPL 1989, pp. 179–190 (1989)

11. Sifakis, J.: A framework for component-based construction extended abstract. In: Proc. SEFM 2005, pp. 293–300. IEEE Computer Society (2005)
12. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: Proc. 26th FOCS 1985, pp. 327–338 (1985)
13. Vardi, M.Y.: Probabilistic Linear-Time Model Checking: An Overview of the Automata-Theoretic Approach. In: Katoen, J.-P. (ed.) AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999. LNCS, vol. 1601, pp. 265–276. Springer, Heidelberg (1999)