# Compositional Methods for Probabilistic Systems[⋆,⋆⋆]

Luca de Alfaro Thomas A. Henzinger Ranjit Jhala

Electrical Engineering and Computer Sciences, University of California, Berkeley
{dealfaro,tah,jhala}@eecs.berkeley.edu

**Abstract.** We present a compositional trace-based model for probabilistic systems. The behavior of a system with probabilistic choice is a stochastic process, namely, a probability distribution on traces, or "bundle." Consequently, the semantics of a system with both nondeterministic and probabilistic choice is a set of bundles. The bundles of a composite system can be obtained by combining the bundles of the components in a simple mathematical way. Refinement between systems is bundle containment. We achieve assume-guarantee compositionality for bundle semantics by introducing two scoping mechanisms. The first mechanism, which is standard in compositional modeling, distinguishes inputs from outputs and hidden state. The second mechanism, which arises in probabilistic systems, partitions the state into probabilistically independent regions.

## 1 Introduction

A system model is *compositional* if the model of a composite system can be obtained by composing the models of the components. Compositionality comes in two flavors: *shallow* and *deep*. Shallow compositionality is essentially a syntactic notion: given two components $P$ and $Q$, we can construct their composition $P\|Q$, but the semantics of this composition is not directly related to that of $P$ and $Q$. On the other hand, deep compositionality relates not only the syntax, but also the semantics: not only can we combine $P$ and $Q$ into $P\|Q$, but the semantics $[\![P\|Q]\!]$ of $P\|Q$ can be obtained by combining $[\![P]\!]$ and $[\![Q]\!]$. A simple model with deep compositionality is that of transition systems with trace semantics [Dil89,Lam93,Lyn96,AH99]. In the variable-based version of this model, a state is an assignment of values to a set of variables, a trace is a sequence of states, and the semantics $[\![P]\!]$ of a component $P$ consists of the set of all traces that correspond to behaviors of $P$. If the variables written by $P$ are read by another component $Q$, and vice versa, and components interact synchronously, then

⋆ This research was supported in part by the SRC contract 99-TJ-683.003, the AFOSR MURI grant F49620-00-1-0327, the MARCO GSRC grant 98-DT-660, the NSF Theory grant CCR-9988172, and the DARPA SEC grant F33615-C-98-3614.

⋆⋆ A preliminary version of this paper will appear in the *Proceedings of the 12th International Conference on Concurrency Theory* (CONCUR 2001).

composition corresponds to the intersection of trace sets: $[\![P \| Q]\!] = [\![P]\!] \cap [\![Q]\!]$. If each component has also private variables, which are invisible to the other component, then we obtain the observable traces of $P \| Q$ via projection from the behaviors of $P$ and $Q$ that agree on the mutually visible variables.

The chief advantage of deep over shallow compositionality is that deep compositionality enables not only the composition of systems, but also the composition of properties. In particular, it becomes possible to prove properties of systems by proving properties of their components. Since each component is simpler than the composite system, such a compositional approach can be markedly more efficient. A basic application of property composition consists in proving a refinement relation $P \| Q \preceq P' \| Q'$ between a composite implementation $P \| Q$ and a composite specification $P' \| Q'$ by proving independently the two component refinements $P \preceq P'$ and $Q \preceq Q'$. In practice, a more powerful *assume-guarantee* rule is preferred, where the proofs of each component refinement rely on the hypothesis that the other component refinement holds, yielding the proof obligations $P \| Q' \preceq P' \| Q'$ and $P' \| Q \preceq P' \| Q'$. Such a circular assume-guarantee rule is available, for example, for [MC81,AL95,McM97,AH99]. In spite of the advantages of deeply compositional models, no such model has thus far been presented for systems with both probability and nondeterminism. The difficulty, as we will detail in Section 2, lies in the interaction between the resolution of nondeterministic choice, mediated by *schedulers,* and composition.

We introduce a deeply compositional model for systems with both probabilistic and nondeterministic choice, and we show how the model leads to the first assume-guarantee rule for checking refinement between probabilistic systems. The model is based on a synchronous, variable-based view of systems, as in *reactive modules* [AH99]. The semantics of a component is obtained by generalizing trace semantics: instead of a trace, our basic semantical unit is a probability distribution on traces —i.e., a stochastic process over the state space— which we call a “bundle.” A bundle represents a single (probabilistic) behavior of a component, once all nondeterminism has been resolved by a scheduler. Thus, the semantics $[\![P]\!]$ of a component $P$ consists of a set of bundles. This is very similar to the semantics for the *probabilistic I/O automata* of [SL94,Seg95]. Unlike the models of [SL94,Seg95], however, our models are deeply compositional. In our models, the semantics of composition is essentially intersection, as in the nonprobabilistic case: for two components $P$ and $Q$ that share the same variables, we have $[\![P \| Q]\!] = [\![P]\!] \cap [\![Q]\!]$; this is now an intersection of bundles, rather than traces. This relationship will be generalized also to components with private variables.

Our deeply compositional semantics opens the way to the use of assume-guarantee methods for probabilistic systems. In the trace-based semantics of nondeterministic systems, refinement is defined as trace containment. In analogy, we define refinement as bundle containment: $P \preceq P'$ iff $[\![P]\!] \subseteq [\![P']\!]$. This definition, together with deep compositionality, ensures that refinement can be proven in a compositional fashion: $P \preceq P'$ and $Q \preceq Q'$ imply $P \| Q \preceq P' \| Q'$. Furthermore, we show that a circular assume-guarantee rule for refinement can be applied: $P \| Q' \preceq P' \| Q'$ and $P' \| Q \preceq P' \| Q'$ imply $P \| Q \preceq P' \| Q'$. This does not follow immediately from deep compositionality, but requires inductive reasoning, as in the nonprobabilistic case. Arguably, the ability of studying systems in a compositional fashion is even more beneficial for probabilistic than for purely nondeterministic systems, due to the greater complexity of the verification algorithms and symbolic data structures [dAKN+00]. We therefore believe that our deeply compositional semantics, together with the assume-guarantee rule for proving refinement, represent a step forward in the analysis and verification of complex probabilistic systems.

## 2 Motivational Examples

In systems with both probabilistic and nondeterministic choice, the resolution of nondeterministic choice is mediated by *schedulers,* which specify how to choose between nondeterministic alternatives [Der70,Var85,SL94,BdA95]. Once a scheduler is fixed, the behavior of a system is a stochastic process, namely, a bundle. Following [SL94,Seg95], we define the semantics $[\![P]\!]$ of a component $P$ as the set of bundles that arise from all possible schedulers for $P$. While *deterministic* schedulers resolve each choice in a deterministic manner [Var85], we opt for *randomized* schedulers, which select probability distributions of outcomes [SL94,BdA95], thus resolving nondeterminism in a randomized fashion, similarly to Markov decision processes [Der70]. Our preference for randomized schedulers is motivated by refinement: under randomized scheduling, if we replace probability by nondeterminism in a component $P$, obtaining the component $P'$, then $P$ refines $P'$. Hence, nondeterminism can be seen as "unspecified probability," and it can be used to encode imprecise probability values [JL91,dA98]. To see this, consider the following example.

**Example 1** Assume that $P$ and $P'$ are two components, each writing once to a variable $x$ that can take the two values 0 or 1. In $P$, the variable $x$ is set to 0 or 1 with probability $\frac{1}{2}$ each; in $P'$, the choice between setting $x$ to 0 or 1 is entirely nondeterministic. Since there is no nondeterminism in $P$, there is a single scheduler for $P$ (taking no choices), which gives rise to the behavior (bundle) with the two one-step traces $x\!:\!0$ and $x\!:\!1$, each with probability $\frac{1}{2}$. There are two deterministic schedulers for $P'$: the first sets $x$ to 0 and yields the bundle with the single trace $x\!:\!0$; the second sets $x$ to 1 and yields the bundle with the single trace $x\!:\!1$. Therefore, with deterministic scheduling, $[\![P]\!] \cap [\![P']\!] = \emptyset$. There are infinitely many randomized schedulers for $P'$, one for each real number $\delta \in [0,1]$: the scheduler $\sigma_\delta$ sets $x$ to 0 with probability $\delta$, and sets $x$ to 1 with probability $1-\delta$, and thus yields the bundle with the two traces $x\!:\!0$ (probability $\delta$) and $x\!:\!1$ (probability $1-\delta$). Choosing $\delta = \frac{1}{2}$, we see that using randomized schedulers, $[\![P]\!] \subseteq [\![P']\!]$, as desired. ∎

We adopt a purely variable-based view of systems: each state is a valuation of a set of typed variables. Following *reactive modules* [AH99], our components, called *probabilistic modules,* have a set of *private* variables, visible to the module

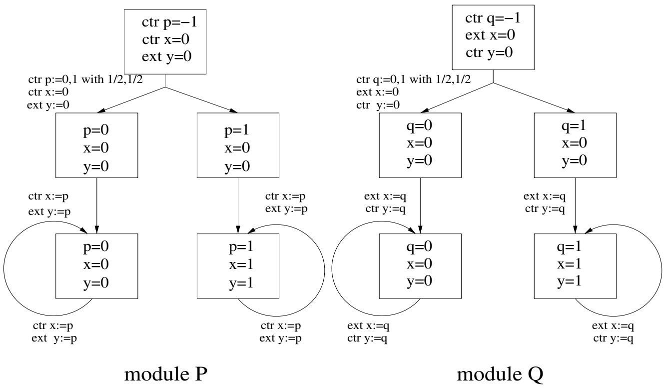


**Fig. 1.** Bundle of $P$ and $Q$, but not of $P \| Q$

alone, a set of *interface* variables, which are the outputs of the module, and a set of *external* variables, which are the inputs of the module. Together, the interface and external variables are called *observable*, and the private and interface variables are called the *controlled* variables of the module. Here, we justify some of the definitions that are necessary to achieve a deeply compositional semantics.

**Example 2** The module $P$ has private variable $p$, interface variable $x$, and external variable $y$. All variables are modified repeatedly, in a sequence of discrete steps. The initial values of $p$ and $x$ are $-1$ and $0$, respectively. When $p = -1$, the module $P$ updates $p$ to 0 or 1 with equal probability $\frac{1}{2}$, and updates $x$ to 0. When $p \neq -1$, the module $P$ leaves $p$ unchanged, and updates $x$ to $p$. The initial value and updates of the external variable $y$ are entirely nondeterministic. Let $\sigma_P$ be the scheduler for $P$ that initially sets $y$ to 0, and then updates $y$ to 0 when $p = -1$, and updates $y$ to $p$ when $p \neq -1$. The module $Q$, and its scheduler $\sigma_Q$, are defined symmetrically, with $q$ in place of $p$, and $x, y$ interchanged. The behavior of these two modules, under their respective schedulers, is illustrated in Figure 1. Under the scheduler $\sigma_P$, the observable part of the behavior of $P$ is a bundle $\mathbf{b}$ consisting of the two infinite traces $(x:0, y:0), (0,0), (0,0), \ldots$ and $(x:0, y:0), (0,0), (1,1), \ldots$ with probability $\frac{1}{2}$ each. The bundle $\mathbf{b}$ also results from $Q$ under the scheduler $\sigma_Q$. However, $\mathbf{b}$ is not a bundle of $P \| Q$, under any scheduler. In fact, there is no nondeterminism in $P \| Q$: the values for $p$ and $q$ are chosen independently, and the unique observable behavior of $P \| Q$ is the bundle that, for each $i, j \in \{0, 1\}$, contains the trace $(x:0, y:0), (0,0), (i,j), \ldots$ with probability $\frac{1}{4}$. ∎

Thus, in order to obtain a compositional semantics, the values of the external variables must be chosen without looking at the values of private variables. On the other hand, the choice of values for the controlled variables should be able to depend on the values of private variables. Hence, we need at least two schedulers for each module: one for the external variables, which cannot look at the values of the private variables, and one for the controlled variables, which can.

**Example 3** Consider a module $P$ with an interface variable $x$ and an external variable $y$, and a module $Q$ with the interface variable $y$ and the external variable $x$. Both variables can have value 0 or 1, and are updated completely nondeterministically. The behavior of $P$ is thus determined by two schedulers: a module scheduler $\pi_P$ that provides a probability distribution for the choice between values 0 and 1 for $x$, and an environment scheduler $\eta_P$ that provides a probability distribution for the choice between values 0 and 1 for $y$. Symmetrically, the behavior of $Q$ is determined by the two schedulers $\pi_Q$ and $\eta_Q$. In the compositon $P \| Q$, the variables $x$ and $y$ are both controlled variables. If we postulate that all controlled variables are controlled by the same scheduler, then there is a module scheduler $\pi_{P\|Q}$ for $P \| Q$ that chooses for $(x, y)$ the values $(0, 0)$ with probability $\frac{1}{2}$, and $(1, 1)$ with probability $\frac{1}{2}$. This scheduler gives rise to the bundle that contains the one-step trace $(x : 0, y : 0)$ with probability $\frac{1}{2}$, and the one-step trace $(x : 1, y : 1)$ with probability $\frac{1}{2}$. This bundle is neither a bundle of $P$, nor a bundle of $Q$, however, because in $P$ and $Q$ the values for $x$ and $y$ are chosen independently. ∎

In previous models of probabilistic systems, a single scheduler is used to resolve all nondeterminism in the choice of values for the controlled variables [Var85,SL94,Seg95,BdA95]. The above example indicates that in order to achieve deep compositionality, we must abandon this restriction, and allow multiple schedulers for the resolution of the nondeterminism within a module. For each scheduler, we must specify the set of variables that it affects, as well as the set of variables at which it can look. To this end, we partition the controlled variables of a module into *atoms:* each atom represents a set of controlled variables that are scheduled together, by a single scheduler. Each atom has also a set of *read* variables, which are the variables visible to the scheduler, on which the choice of values for the controlled variables may depend. When we compose modules, we take the union of their sets of atoms, thus ensuring that the scheduling dependencies between variables remain unchanged. In the example above, $P$ would contain an atom for scheduling $x$, and $Q$ an atom for scheduling $y$ (there are no read variables). The composite system $P \| Q$ then inherits the atoms of $P$ and $Q$, and has two schedulers, one for $x$, the other for $y$. Each pair of schedulers for $P \| Q$ corresponds to both a scheduler of $P$ and a scheduler of $Q$, yielding $[\![P]\!] \cap [\![Q]\!] = [\![P \| Q]\!]$.

Our atoms are derived directly from the atoms of *reactive modules* [AH99]. However, while in [AH99] the atoms indicate which variables are updated jointly (i.e., interdependently), and dependent on which other variables, here atoms acquire additional meaning: they indicate which variables are scheduled jointly, and dependent on which other variables. In particular, while in the nonprobabilistic case the merging of atoms never changes the behaviors (traces) of a module, in the probabilistic case, the merging of atoms may increase the behaviors (bundles) by permitting strictly more probabilistic dependencies between variable values, as the previous example illustrates. This is because it is the atom structure of a module that determines probabilistic dependence and, importantly, *independence* between variables values.

# 3 Probabilistic Modules and Composition

## 3.1 Definition of probabilistic modules

**Definition 1.** [**States and moves**] *Let $X$ be a set of typed variables. An $X$-*state *$s$ is a function that maps each variable in $X$ to a value of the appropriate type. We write* $\mathrm{Val}(X)$ *for the set of $X$-states. An $X$-*move **m** *is a probability distribution on $X$-states. The move* **m** *is* nonprobabilistic *if the support of $m$ is a single state. Given two $X$-moves $\mathbf{m}_1$ and $\mathbf{m}_2$, and a real number $\delta \in [0,1]$, we write $\delta \cdot \mathbf{m}_1 + (1-\delta) \cdot \mathbf{m}_2$ for the $X$-move $\mathbf{m}$ such that $\mathbf{m}(s) = \delta \cdot \mathbf{m}_1(s) + (1-\delta) \cdot \mathbf{m}_2(s)$ for all $X$-states $s$.*

While a nonprobabilistic transition $(s, s')$ consists of a source state $s$ and a target state $s'$, a probabilistic transition $(s, \mathbf{m})$ consists of a source state $s$ and a probability distribution $\mathbf{m}$ of target states. A nondeterministic collection of transitions (probabilistic or not) is called an "action." Consider, for example, the action $F = \{f_1, f_2\}$ with the two transitions $f_1 = (s, \mathbf{m}_1)$ and $f_2 = (s, \mathbf{m}_2)$. Every action is resolved by a scheduler, which, given a sequence of actions, produces a sequence of states. Given the action $F = \{f_1, f_2\}$ in state $s$, a deterministic scheduler may choose either the transition $f_1$, whose outcome is determined by the probability distribution $\mathbf{m}_1$, or the transition $f_2$, whose outcome is determined by the probability distribution $\mathbf{m}_2$. A randomized scheduler may choose any convex combination of $f_1$ and $f_2$, say, $f_1$ with probability $\delta$ and $f_2$ with probability $1-\delta$.

**Definition 2.** [**Transitions and actions**] *Let $X$ and $Y$ be two sets of typed variables. A* probabilistic transition $(s, \mathbf{m})$ from $X$ to $Y$ *consists of an $X$-state $s$ and a $Y$-move $\mathbf{m}$. The transition $(s, \mathbf{m})$ is* nonprobabilistic *if the move $\mathbf{m}$ is nonprobabilistic. A* probabilistic action $F$ from $X$ to $Y$ *is a set of probabilistic transitions from $X$ to $Y$. The action $F$ is* deterministic *if for every $X$-state $s$, there is at most one $Y$-move $\mathbf{m}$ such that $(s, \mathbf{m}) \in F$. The action $F$ is* nonprobabilistic *if all transitions in $F$ are nonprobabilistic. The action $F$ is* convex-closed *if for all $X$-states $s$, all $Y$-moves $m_1$ and $m_2$, and all real numbers $\delta \in [0,1]$, if $(s, \mathbf{m}_1) \in F$ and $(s, \mathbf{m}_2) \in F$, then $(s, \delta \cdot \mathbf{m}_1 + (1-\delta) \cdot \mathbf{m}_2) \in F$. The* convex-closure $\mathrm{ConvexClosure}(F)$ *is the least superset of $F$ that is a convex-closed action from $X$ to $Y$.*

A system proceeds in a sequence of discrete rounds. In the first round, all system variables are initialized in accordance with initial actions; in the subsequent rounds, all system variables are updated in accordance with update actions. Dependencies between variables are expressed by clustering the variables into "atoms." If two variables are controlled (i.e., initialized and updated) by the same atom, then their behaviors are interdependent. Consequently, if the behaviors of two variables are desired to be independent, then the variables must be put into different atoms. Consider, for example, two boolean variables $x$ and $y$. First, suppose that $x$ and $y$ are jointly controlled by a single atom. The deterministic initial action $\frac{1}{2}(x, y := 0, 0) + \frac{1}{2}(x, y := 1, 1)$ with probability $\frac{1}{2}$ initializes both variables to 0, and with probability $\frac{1}{2}$ initializes both variables to 1. There are two possible initial states, (0,0) and (1,1). Second, suppose that $x$ and $y$ are independently controlled by different atoms. The deterministic initial actions $\frac{1}{2}(x := 0) + \frac{1}{2}(x := 1)$ and $\frac{1}{2}(x := 0) + \frac{1}{2}(x := 1)$ initialize each variable with equal probability to 0 or 1. There are four possible initial states, (0,0), (0,1), (1,0), and (1,1). If $x$ is controlled by one atom, and $y$ by another atom, then $x$ may still depend on $y$, because the atom controlling $x$ may "read" the value of $y$ at the beginning of each round. All such read dependencies must be declared explicitly; the absence of read dependencies (or transitively implied read dependencies) between different atoms means true independence, in the probabilistic sense.

**Definition 3.** [**Atoms**] *Let $X$ be a set of typed variables. A* probabilistic $X$-atom *$A$ consists of a set* $\mathrm{readX}(A) \subseteq X$ *of* read variables, *a set* $\mathrm{ctrX}(A) \subseteq X$ *of* controlled variables, *a probabilistic* initial action $\mathrm{initF}(A)$ *from $\emptyset$ to* $\mathrm{ctrX}(A)$*, and a probabilistic* update action $\mathrm{updateF}(A)$ *from* $\mathrm{readX}(A)$ *to* $\mathrm{ctrX}(A)$*. The atom $A$ is* deterministic *if both* $\mathrm{initF}(A)$ *and* $\mathrm{updateF}(A)$ *are deterministic actions. The atom $A$ is* nonprobabilistic *if both* $\mathrm{initF}(A)$ *and* $\mathrm{updateF}(A)$ *are nonprobabilistic.*

In addition to its atoms, which provide the initial and update actions for variables, an open probabilistic system —or "module"— also provides the capability to view variables that are not initialized and updated by the module, and the capability to hide variables from the view of other modules. The former variables are called "external"; the latter, "private." The variables that are neither external nor private —i.e., the variables that are initialized and updated by the module and can be viewed by other modules— are called "interface" variables.

**Definition 4.** [**Modules**] *A* probabilistic module *$P$ consists of a declaration and a body. The module declaration is a finite set of typed variables $X(P)$ that is partitioned into three sets: the set* $\mathrm{extlX}(P)$ *of* external variables, *the set* $\mathrm{intfX}(P)$ *of* interface variables, *and the set* $\mathrm{privX}(P)$ *of* private variables. *The module body is a finite set* $\mathrm{Atoms}(P)$ *of probabilistic $X(P)$-atoms such that (1)* $\mathrm{intfX}(P) \cup \mathrm{privX}(P) = \bigcup_{A \in \mathrm{Atoms}(P)} \mathrm{ctrX}(A)$*, and (2) for all atoms $A_1$ and $A_2$ in* $\mathrm{Atoms}(P)$*, the sets* $\mathrm{ctrX}(A_1)$ *and* $\mathrm{ctrX}(A_2)$ *are disjoint. The module $P$ is* deterministic *if all atoms in* $\mathrm{Atoms}(P)$ *are deterministic. The module $P$ is* nonprobabilistic *if all atoms in* $\mathrm{Atoms}(P)$ *are nonprobabilistic.*

Given a module $P$, we call $\mathrm{intfX}(P) \cup \mathrm{extlX}(P)$ the set of *observable* variables of $P$, and we call $\mathrm{privX}(P) \cup \mathrm{intfX}(P)$ the set of *controlled* variables of $P$. The nonprobabilistic modules are exactly the *reactive modules* of [AH99] without await dependencies. We have omitted await dependencies, which are instrumental for synchronous communication, for simplicity; they can be added without changing the results of this paper. Modules without external variables are called "closed." Every closed module defines a Markov decision process; every closed deterministic module defines a Markov chain.

### 3.2 Operations on probabilistic modules

We define three operations on probabilistic modules: hiding, composition, and opening. The hiding (or abstraction) operation makes some interface variables private.

**Definition 5.** [**Hiding**] *Let $P$ be a probabilistic module, and let $Y \subseteq \mathrm{intfX}(P)$ be a set of interface variables. By* hiding $Y$ in $P$ *we obtain the probabilistic module $P \backslash Y$ with the set* $\mathrm{extlX}(P \backslash Y) = \mathrm{extlX}(P)$ *of external variables, the set* $\mathrm{intfX}(P \backslash Y) = \mathrm{intfX}(P) \backslash Y$ *of interface variables, the set* $\mathrm{privX}(P \backslash Y) = \mathrm{privX}(P) \cup Y$ *of private variables, and the set* $\mathrm{Atoms}(P \backslash Y) = \mathrm{Atoms}(P)$ *of atoms.*

The (parallel) composition operation puts together two modules which control the behaviors of disjoint sets of variables. The composition operation can be applied only when the observable —i.e., external and interface— variables of two modules coincide. This constraint is necessary for a compositional semantics in the presence of probabilities. If a module has a private variable $p$ and an external variables $y$, then the module semantics insists on the independence between $p$ and $y$, because the environment, which controls $y$, cannot observe $p$. It is therefore illegal to compose a module with private $p$, but without external $y$, and an environment that controls $y$, because the module, which does not know about the existence of $y$, has no way of noting that $p$ and $y$ must be independent. We will illustrate this in Example 4, presented below after the necessary terminology has been introduced. This underlines how the scoping of variables in the probabilistic case is considerably more delicate than in the nonprobabilistic case, where incidental dependencies between variables cause no harm.

**Definition 6.** [**Composition**] *Two probabilistic modules $P_1$ and $P_2$* can be composed *if (1)* $\mathrm{extlX}(P_1) \cup \mathrm{intfX}(P_1) = \mathrm{extlX}(P_2) \cup \mathrm{intfX}(P_2)$*, (2)* $\mathrm{intfX}(P_1) \cap \mathrm{intfX}(P_2) = \emptyset$*, and (3)* $\mathrm{privX}(P_1) \cap X(P_2) = \emptyset$ *and* $X(P_1) \cap \mathrm{privX}(P_2) = \emptyset$*. Two* composition *of two probabilistic modules $P_1$ and $P_2$ that can be composed is the probabilistic module $P_1||P_2$ with the set* $\mathrm{extlX}(P_1||P_2) = (\mathrm{extlX}(P_1) \cup \mathrm{extlX}(P_2)) \backslash \mathrm{intfX}(P_1||P_2)$ *of external variables, the set* $\mathrm{intfX}(P_1||P_2) = \mathrm{intfX}(P_1) \cup \mathrm{intfX}(P_2)$ *of interface variables, the set* $\mathrm{privX}(P_1||P_2) = \mathrm{privX}(P_1) \cup \mathrm{privX}(P_2)$ *of private variables, and the set* $\mathrm{Atoms}(P_1||P_2) = \mathrm{Atoms}(P_1) \cup \mathrm{Atoms}(P_2)$ *of atoms.*

The opening operation adds external variables to a module, and is unique to probabilistic modules. It is used to ensure that two modules have the same set of observable variables before they are composed.

**Definition 7.** [**Opening**] *Let $P$ be a probabilistic module, and let $Y$ be a set of typed variables disjoint from the set $X(P)$ of module variables. By* opening $P$ to $Y$ *we obtain the probabilistic module $P \uplus Y$ with the set* $\mathrm{extlX}(P \uplus Y) = \mathrm{extlX}(P) \cup Y$ *of external variables, the set* $\mathrm{intfX}(P \uplus Y) = \mathrm{intfX}(P)$ *of interface variables, the set* $\mathrm{privX}(P \uplus Y) = \mathrm{privX}(P)$ *of private variables, and the set* $\mathrm{Atoms}(P \uplus Y) = \mathrm{Atoms}(P)$ *of atoms.*

### 3.3 Trace semantics of probabilistic systems

**Definition of probabilistic languages.** While the behavior of a deterministic and closed nonprobabilistic system is an infinite state sequence —called a "trace"— the behavior of a deterministic and closed probabilistic system (Markov chain) is a probability distribution on traces —called a "bundle." Consequently, the possible behaviors of a nondeterministic or open probabilistic system form a *set* of traces, and the possible behaviors of a nondeterministic or open probabilistic system (in the nondeterministic and closed case, a Markov decision process) form a *set* of bundles. We restrict ourselves to safe systems, where it suffices to consider *finite* behaviors, albeit of arbitrary length; this restriction is particularly technically convenient in the probabilistic case, as probability distributions on finite traces can be defined in a straightforward manner.

**Definition 8.** [**Traces and bundles**] *Let $X$ be a set of typed variables, and let $n$ be a nonnegative integer. An $X$-*trace $t$ of length $n$ *is a sequence of $X$-states with $n$ elements. We write $\varepsilon$ for the empty sequence, and given $1 \le i \le n$, we write $t(i)$ for the $i$-th element of $t$. We write* $\mathrm{Val}^n(X)$ *for the set of $X$-traces of length $n$. An $X$-*bundle of length $n$ *is a probability distribution over $X$-traces of length $n$. The unique $X$-bundle of length 0, which assigns the probability 1 to $\varepsilon$, is called the* empty bundle. *The bundle* $\mathbf{b}$ *is* nonprobabilistic *if the support of* $\mathbf{b}$ *is a single trace. If $n > 0$, then the* prefix *of* $\mathbf{b}$ *is the $X$-bundle* $\mathbf{b}'$ *of length $n-1$ such that* $\mathbf{b}'(t) = \sum_{s \in \mathrm{Val}(X)} \mathbf{b}(t \cdot s)$ *for all $X$-traces $t$ of length $n-1$. The $X$-bundle* $\mathbf{b}''$ *of length $n+1$ is an* extension *of* $\mathbf{b}$ *if* $\mathbf{b}$ *is the prefix of* $\mathbf{b}''$.

Each bundle records the outcome of a particular sequence of nondeterministic or randomized choices made by a system. Such a sequence of choices is called a "scheduler" (to be defined later). The set of bundles that result from all possible schedulers are collected forms a probabilistic languages.

**Definition 9.** [**Languages**] *Let $X$ be a set of typed variables. A set $L$ of $X$-bundles is* prefix-closed *if for every bundle* $\mathbf{b}$ *in $L$, the prefix of* $\mathbf{b}$ *is also in $L$. The set $L$ of bundles is* extension-closed *if (1) the empty bundle is in $L$, and (2) for every bundle* $\mathbf{b}$ *in $L$, some extension of* $\mathbf{b}$ *is also in $L$. A* probabilistic $X$-language $L$ *is a prefix-closed and extension-closed set of $X$-bundles. The language $L$ is* deterministic *if for all nonnegative integers $n$, there is a single bundle of length $n$ in $L$. The language $L$ is* nonprobabilistic *if all bundles in $L$ are nonprobabilistic.*

The nonprobabilistic languages are precisely the prefix-closed and extension-closed trace sets; that is, the languages generated by safe and deadlock-free discrete systems.

**Operations on probabilistic languages.** We define two operations on bundles and on probabilistic languages: projection and product. Properties of these operations are needed to prove the compositionality of hiding and composition for probabilistic systems.

**Definition 10.** [**Projection**] *Let $X$ and $X' \subseteq X$ be two sets of typed variables. The* $X'$-projection *of an $X$-state $s$ is the $X'$-state $s[X']$ such that $(s[X'])(x) = s(x)$ for all variables $x \in X'$. The* $X'$-projection *of an $X$-move $\mathbf{m}$ is the $X'$-move $\mathbf{m}[X']$ such that $(\mathbf{m}[X'])(s') = \sum_{s \in \mathrm{Val}(X) \text{ with } s[X']=s'} \mathbf{m}(s)$ for all $X'$-states $s'$. The* $X'$-projection *of an $X$-trace $t$ of length $n$ is the $X'$-trace $t[X']$ of length $n$ such that $(t[X'])(i) = (t(i))[X']$ for all $1 \le i \le n$. The* $X'$-projection *of an $X$-bundle $\mathbf{b}$ of length $n$ is the $X'$-bundle $\mathbf{b}[X']$ of length $n$ such that $(\mathbf{b}[X'])(t') = \sum_{t \in \mathrm{Val}^n(X) \text{ with } t[X']=t'} \mathbf{b}(t)$ for all $X'$-traces $t$ of length $n$. The* $X'$-projection *of an $X$-language $L$ is the $X'$-language $L[X'] = \{\mathbf{b}[X'] \mid \mathbf{b} \in L\}$.*

**Definition 11.** [**Product**] *Let $X_1$ and $X_2$ be two sets of typed variables. An $X_1$-state (resp. move; trace; bundle) $s_1$ and a $X_2$-state (resp. move; trace; bundle) $s_2$* can be multiplied *if $s_1[X_1 \cap X_2] = s_2[X_1 \cap X_2]$. The* product *of an $X_1$-state $s_1$ and an $X_2$-state $s_2$ that can be multiplied is the $(X_1 \cup X_2)$-state $s_1 \times s_2$ such that $(s_1 \times s_2)(x_1) = s_1(x_1)$ for all variables $x_1 \in X_1$, and $(s_1 \times s_2)(x_2) = s_2(x_2)$ for all $x_2 \in X_2$. The* product *of an $X_1$-move $\mathbf{m}_1$ and an $X_2$-move $\mathbf{m}_2$ that can be multiplied is the $(X_1 \cup X_2)$-move $\mathbf{m}_1 \times \mathbf{m}_2$ such that $(\mathbf{m}_1 \times \mathbf{m}_2)(s) = \mathbf{m}_1(s[X_1]) \cdot \mathbf{m}_2(s[X_2]) / \mathbf{m}_1(s[X_1 \cap X_2])$ for all $(X_1 \cup X_2)$-states $s$. The* product *of an $X_1$-trace $t_1$ and an $X_2$-trace $t_2$ that have length $n$ and can be multiplied is the $(X_1 \cup X_2)$-trace $t_1 \times t_2$ of length $n$ such that $(t_1 \times t_2)(i) = t_1(i) \times t_2(i)$ for all $1 \le i \le n$. The* product *of an $X_1$-bundle $\mathbf{b}_1$ and an $X_2$-bundle $\mathbf{b}_2$ that have length $n$ and can be multiplied is the $(X_1 \cup X_2)$-bundle $\mathbf{b}_1 \times \mathbf{b}_2$ of length $n$ such that $(\mathbf{b}_1 \times \mathbf{b}_2)(t) = \mathbf{b}_1(t[X_1]) \cdot \mathbf{b}_2(t[X_2]) / \mathbf{b}_1(t[X_1 \cap X_2])$ for all $(X_1 \cup X_2)$-traces $t$ of length $n$. The* product *of an $X_1$-language $L_1$ and an $X_2$-language $L_2$ is the $(X_1 \cup X_2)$-language $L_1 \times L_2 = \{\mathbf{b}_1 \times \mathbf{b}_2 \mid \mathbf{b}_1 \in L_1 \text{ and } \mathbf{b}_2 \in L_2 \text{ can be multiplied}\}$.*

The product of bundle languages is the probabilistic analogue to the intersection of trace languages. This is captured in the following lemma.

**Lemma 1.** *Let $L_1$ be a probabilistic $X_1$-language, and let $L_2$ be a probabilistic $X_2$-language. Then $(L_1 \times L_2)[X_1 \cap X_2] = L_1[X_1 \cap X_2] \cap L_2[X_1 \cap X_2]$.*

**Containment between probabilistic languages.** Since the set of possible behaviors of a probabilistic system is a set of bundles, the appropriate notion of refinement between probabilistic systems is bundle containment: an implementation refines a specification iff every possible behavior (bundle) of the implementation is a legal behavior (bundle) of the specification.

**Definition 12.** [**Bundle containment**] *Let $X$ and $X' \subseteq X$ be two sets of typed variables. If $L$ is a probabilistic $X$-language, and $L'$ is a probabilistic $X'$-language, then $L$* is bundle-contained in *$L'$ if $L[X'] \subseteq L'$.*

### 3.4 Connecting syntax and semantics

**Bundle semantics of probabilistic modules.** We associate with every probabilistic module a probabilistic language, i.e., a set of bundles. The key concept for doing this is the concept of a "scheduler," which represents a possible sequence of choices taken by the module. Each scheduler, then, gives rise to an infinite bundle that can be represented by all its finite prefixes. We permit randomized schedulers, which in each state can choose probability distributions over the enabled transitions. By contrast, a deterministic scheduler must choose exactly one of the enabled transitions.

**Definition 13.** [**Schedulers**] *Let $X$ and $Y$ be two sets of variables. A* scheduler *$\sigma$ from $X$ to $Y$ is a function that maps every $X$-trace to a probability distribution on $Y$-states. The scheduler $\sigma$ is* nonprobabilistic *if for all $X$-traces $t$, the support of $\sigma(t)$ is a single $Y$-state. If $\sigma$ is a scheduler from $X$ to $X$, then the* 0-outcome *of $\sigma$ is the empty bundle, and for all positive integers $i > 0$, the* $i$-outcome *of $\sigma$ is an inductively defined $X$-bundle $\mathbf{b}_i$ of length $i$: the bundle $\mathbf{b}_i$ is the extension of the bundle $\mathbf{b}_{i-1}$ such that $\mathbf{b}_i(t) = \mathbf{b}_{i-1}(t(1)\cdots t(i-1)) \cdot (\sigma(t(1)\cdots t(i-1)))(t(i))$ for all $X$-traces $t$ of length $i$. We collect the set of $i$-outcomes of $\sigma$, for all $i \geq 0$, in the set* $\mathrm{Outcome}(\sigma)$ *of $X$-bundles.*

Each scheduler for a module consists of a scheduler for the environment, which chooses the initial and updated values for the external variables, together with a scheduler for each atom, which chooses the initial and updated values for the variables controlled by that atom.

**Definition 14.** [**Schedulers for an atom**] *Consider a probabilistic $X$-atom $A$. The set* $\mathrm{atom}\Sigma(A)$ *of* atom schedulers *for $A$ contains all schedulers $\sigma$ from* $\mathrm{readX}(A)$ *to* $\mathrm{ctrX}(A)$ *such that (1)* $(\cdot, \sigma(\varepsilon)) \in \mathrm{ConvexClosure}(\mathrm{initF}(A))$*, and (2)* $(t(n), \sigma(t)) \in \mathrm{ConvexClosure}(\mathrm{updateF}(A))$ *for all nonempty* $\mathrm{readX}(A)$*-traces $t$ of length $n$. A scheduler $\sigma$ in* $\mathrm{atom}\Sigma(A)$ *is* deterministic *if (1)* $(\cdot, \sigma(\varepsilon)) \in \mathrm{initF}(A)$*, and (2)* $(t(n), \sigma(t)) \in \mathrm{updateF}(A)$ *for all nonempty traces $t$ of length $n$. Let* $\mathrm{atom}\Sigma^d(A)$ *be the set of deterministic schedulers in* $\mathrm{atom}\Sigma(A)$.

To compose the schedulers for several atoms, we define the product of schedulers.

**Definition 15.** [**Product of schedulers**] *Two schedulers $\sigma_1$ and $\sigma_2$ are* disjoint *if $\sigma_1$ is a scheduler from $X_1$ to $Y_1$, and $\sigma_2$ is a scheduler from $X_2$ to $Y_2$, and $Y_1 \cap Y_2 = \emptyset$. If $\sigma_1$ is a scheduler from $X_1$ to $Y_1$, and $\sigma_2$ is a scheduler from $X_2$ to $Y_2$, such that $\sigma_1$ and $\sigma_2$ are disjoint, then the* product *is the scheduler $\sigma_1 \times \sigma_2$ from $X_1 \cup X_2$ to $Y_1 \cup Y_2$ such that $(\sigma_1 \times \sigma_2)(t) = \sigma_1(t[X_1]) \times \sigma_2(t[X_2])$ for all $(X_1 \cup X_2)$-traces $t$. If $\sigma_1$ and $\sigma_2$ are two sets of schedulers such that every scheduler in $\Sigma_1$ is disjoint from every scheduler in $\Sigma_2$, then $\Sigma_1 \times \Sigma_2 = \{\sigma_1 \times \sigma_2 \mid \sigma_1 \in \Sigma_1 \text{ and } \sigma_2 \in \Sigma_2\}$.*

The environment scheduler can initialize and update the external variables in arbitrary, interdependent ways.

**Definition 16.** [**Schedulers for a module**] *Consider a probabilistic module $P$. The set* $\mathrm{extl}\Sigma(P)$ *of* environment schedulers for $P$ *contains all schedulers from* $\mathrm{extlX}(P) \cup \mathrm{intfX}(P)$ *to* $\mathrm{extlX}(P)$. *Let* $\mathrm{extl}\Sigma^d(P)$ *be the set of nonprobabilistic schedulers in* $\mathrm{extl}\Sigma(P)$. *The set* $\mathrm{mod}\Sigma(P)$ *of* module schedulers for $P$ *contains the schedulers from* $X(P)$ *to* $X(P)$ *such that* $\mathrm{mod}\Sigma(P) = \mathrm{extl}\Sigma(P) \times \prod_{A \in \mathrm{Atoms}(P)} \mathrm{atom}\Sigma(A)$. *Let* $\mathrm{mod}\Sigma^d(P) = \mathrm{extl}\Sigma^d(P) \times \prod_{A \in \mathrm{Atoms}(P)} \mathrm{atom}\Sigma^d(A)$.

We are finally ready to define the "trace semantics" of a probabilistic module.

**Definition 17.** [**Semantics of a module**] *Given a probabilistic module $P$, we define* $L(P) = \{\mathrm{Outcome}(\sigma) \mid \sigma \in \mathrm{mod}\Sigma(P)\}$ *and* $L^d(P) = \{\mathrm{Outcome}(\sigma) \mid \sigma \in \mathrm{mod}\Sigma^d(P)\}$. *The* trace semantics of the probabilistic module $P$ *is* $[\![P]\!] = L(P)[\mathrm{extlX}(P) \cup \mathrm{intfX}(P)]$. *The* deterministic trace semantics of $P$ *is* $[\![P]\!]^d = L^d(P)[\mathrm{extlX}(P) \cup \mathrm{intfX}(P)]$.

It it not difficult to verify that the bundle semantics of a module is indeed prefix-closed and extension-closed: if $P$ is a probabilistic module, then $[\![P]\!]$ is a probabilistic $(\mathrm{extlX}(P) \cup \mathrm{intfX}(P))$-language. In general, $[\![P]\!]^d \subset [\![P]\!]$. For nonprobabilistic modules, the traditional trace semantics corresponds to bundle semantics with only deterministic schedulers: according to [AH99], the *trace semantics* of a reactive module $P$ is $[\![P]\!]^d$.

**Bundle interpretation of module operations.** The hiding of variables in a module corresponds to a projection on the bundle language of the module: it is easy to check that for every probabilistic module $P$, and every set $Y \subseteq \mathrm{intfX}(P)$ of interface variables, $[\![P \backslash Y]\!] = [\![P]\!][\mathrm{extlX}(P) \cup \mathrm{intfX}(P) \backslash Y]$. The composition of two modules corresponds to a product on the respective bundle languages. This observation, which is stated in the following proposition, will be instrumental to the compositionality properties of probabilistic modules given in the next section.

**Theorem 1.** *If $P_1$ and $P_2$ are two probabilistic modules that can be composed, then* $[\![P_1 || P_2]\!] = [\![P_1]\!] \times [\![P_2]\!] = [\![P_1]\!] \cap [\![P_2]\!]$.

*Proof.* By induction on the length of bundles, we show the following two observations, which rely heavily on the fact that schedulers have restricted visibility. First, $\mathbf{b} \in L(P_1 \| P_2)$ implies that $\mathbf{b}[X(P_1)] \in L(P_1)$ and $\mathbf{b}[X(P_2)] \in L(P_2)$. Moreover, $\mathbf{b} = \mathbf{b}[X(P_1)] \times \mathbf{b}[X(P_2)]$. This in turn also means that every bundle in $[\![P_1 \| P_2]\!]$ can be "factored," via projection, into bundles in $[\![P_1]\!]$ and $[\![P_2]\!]$. Second, for all bundles $\mathbf{b}_1 \in L(P_1)$ and $\mathbf{b}_2 \in L(P_2)$ such that $\mathbf{b}_1[X(P_1) \cap X(P_2)] = \mathbf{b}_2[X(P_1) \cap X(P_2)]$), we have $\mathbf{b}_1 \times \mathbf{b}_2 \in L(P_1 \| P_2)$. These two observations combine to give the first equality. The observation that $X(P_1) \cap X(P_2)$ is the set of observables of both $P_1$ and $P_2$, coupled with Lemma 1, gives the second equality. ∎

The following example illustrates the need for restricting composition to modules with identical sets of observable variables.

**Example 4** Consider two modules $P$ and $Q$ defined as in Example 2, except that the variables $p$ and $q$ are both interface variables, and thus observable. We assume that each controlled variable of $P$ and $Q$ belongs to a different atom. Under the scheduler $\sigma_P$, the behavior of $P$ is a bundle $\mathbf{b}_P$ consisting of the two infinite traces $(p:-1, x:0, y:0), (0,0,0), (0,0,0), \ldots$ and $(p:-1, x:0, y:0), (1,0,0), (1,1,1), \ldots$ with probability $\frac{1}{2}$ each (to be precise, there are infinitely many bundles, each consisting of two finite traces, whose limit is $\mathbf{b}_P$, but in examples, we find it convenient to informally consider bundles of infinite traces). Similarly, the behavior of $Q$ under $\sigma_Q$ is the bundle $\mathbf{b}_Q$ that contains the two traces $(x:0, y:0, q:-1), (0,0,0), (0,0,0), \ldots$ and $(x:0, y:0, q:-1), (0,0,1), (1,1,1), \ldots$, each with probability $\frac{1}{2}$ (see Figure 1). The bundles $\mathbf{b}_P$ and $\mathbf{b}_Q$ can be multiplied, but their product $\mathbf{b}_P \times \mathbf{b}_Q$ is not a bundle of $P \| Q$. In fact, $\mathbf{b}_P \times \mathbf{b}_Q$ consists of the two traces $(p:-1, x:0, y:0, q:-1), (0,0,0,0), (0,0,0,0), \ldots$ and $(p:-1, x:0, y:0, q:-1), (1,0,0,1), (1,1,1,1), \ldots$ On the other hand, since the values of $p$ and $q$ are chosen independently, $[\![P\|Q]\!]$ consists of a single bundle $\mathbf{b}_{P\|Q}$, containing for each $i, j \in \{0,1\}$ the trace $(p:-1, x:0, y:0, q:-1), (i,0,0,j), (i,i,j,j), \ldots$ with probability $\frac{1}{4}$. It follows that $[\![P]\!] \times [\![Q]\!] \supset [\![P\|Q]\!]$. ∎

## 4 Refinement between Probabilistic Modules

### 4.1 Definition of probabilistic refinement

The refinement relation between probabilistic modules is defined essentially as bundle containment. Unlike in the nonprobabilistic case, however, we require an additional constraint on the atom structure of the two modules, which ensures that an implementation cannot exhibit more variable dependencies than a specification. In other words, all variables that are specified to be independent must be implemented independently.

**Definition 18.** [**Refinement between modules**] *Let $P$ and $P'$ be two probabilistic modules. The module $P$* structurally refines *$P'$, written $P \preceq_S P'$, if (1) $\mathrm{intfX}(P) \supseteq \mathrm{intfX}(P')$ and $\mathrm{extlX}(P) \cup \mathrm{intfX}(P) \supseteq \mathrm{extlX}(P')$, (2) for all variables $x_1, x_2 \in \mathrm{intfX}(P')$, if there is an atom $A \in \mathrm{Atoms}(P)$ such that $x_1, x_2 \in \mathrm{ctrX}(A)$, then there is an atom $A' \in \mathrm{Atoms}(P')$ such that $x_1, x_2 \in \mathrm{ctrX}(A')$, and (3) for all variables $x \in \mathrm{intfX}(P')$ and $y \in \mathrm{intfX}(P') \cup \mathrm{extlX}(P')$, if there is an atom $A \in \mathrm{Atoms}(P)$ such that $x \in \mathrm{ctrX}(A)$ and $y \in \mathrm{readX}(A)$, then there is an atom $A' \in \mathrm{Atoms}(P')$ such that $x \in \mathrm{ctrX}(A')$ and $y \in \mathrm{readX}(A')$. The module $P$ (*behaviorally*) refines *$P'$, written $P \preceq P'$, if $P \preceq_S P'$ and (4) $[\![P]\!]$ is bundle-contained in $[\![P']\!]$.*

It is easy to check that the refinement relation $\preceq$ it is a preorder. Furthermore, every probabilistic module refines its nonprobabilistic abstraction. The nonprobabilistic abstraction of a probabilistic action $F$ is the nonprobabilistic action $\{(s, s') \mid (s, \mathbf{m}) \in F$ and $s' \in \mathrm{Support}(\mathbf{m})\}$. The *nonprobabilistic abstraction* $\mathrm{Nonprob}(P)$ of a probabilistic module $P$ is the nonprobabilistic module that results from $P$ by replacing all initial and update actions of $P$ with their nonprobabilistic abstractions. Then, $P \preceq \mathrm{Nonprob}(P)$.

Refinement between nonprobabilistic modules, however, does not quite agree with refinement between *reactive modules*, as defined in [AH99]. The reason is that conditions (2) and (3) are absent from the definition of refinement for reactive modules, which is purely behavioral (namely, trace containment). For example, two atoms of a reactive module specification can be implemented by a single atom. If atoms are viewed structurally, say, as blocks in a block diagram, then such a refinement breaks component boundaries. This is brought to the fore formally in the probabilistic case, where atoms carry meaning as boundaries between independent variables. We submit that it is the definition above, including the structural conditions (2) and (3), which is more sensible also in the nonprobabilistic case of reactive modules. Once conditions (2) and (3) are added to the refinement between reactive modules, then the probabilistic case is a conservative extension: for nonprobabilistic modules $P$ and $P'$, we have $P \preceq P'$ iff $P \preceq_S P'$ and ($4^d$) $[\![P]\!]^d$ is bundle-contained in $[\![P']\!]^d$.

### 4.2 Compositionality of probabilistic refinement

The following theorem summarizes the compositionality properties of the refinement relation between probabilistic modules. In particular, refinement is a congruence with respect to all module operations, and the refinement between composite modules can be decomposed using circular assume-guarantee reasoning.

**Theorem 2. [Compositionality]** *The following statements are true, provided all subexpressions are well-defined:*

- $P \preceq P \backslash Y$.
- $P \uplus Y \preceq P$.
- $P || Q \preceq P$.
- *If* $P \preceq P'$, *then* $P \backslash Y \preceq P' \backslash Y$.
- *If* $P \preceq P'$, *then* $P \uplus Y \preceq P' \uplus Y$.
- *If* $P \preceq P'$, *then* $P || Q \preceq P' || Q$.
- *If* $P || Q' \preceq Q$ *and* $Q || P' \preceq Q'$, *then* $P || P' \preceq Q || Q'$.

The last assertion is an assume-guarantee rule for probabilistic modules. Its proof uses the following lemma, whose proof relies on Theorem 1. Essentially, the lemma states that the observable part of a bundle of length $i$ is obtained from the observable part of its prefix of length $i-1$, the environment scheduler and the "observable" behaviour of the module scheduler, and the last may be written as the product of the observable behaviours of the atom schedulers and the environment scheduler. Given a scheduler $\sigma$ from $X(P)$ to $\mathrm{ctrX}(P)$, an $X(P)$-bundle $\mathbf{b}$, and its projection $\mathbf{b}^* = \mathbf{b}[\mathrm{intfX}(P) \cup \mathrm{extlX}(P)]$, define the *observable scheduler* $\sigma^*$ w.r.t. $\mathbf{b}$ as the scheduler from $\mathrm{intfX}(P) \cup \mathrm{extlX}(P)$ to $\mathrm{intfX}(P)$ such that for every $(\mathrm{intfX}(P) \cup \mathrm{extlX}(P))$-trace $s$ of length $i-1$,

$$\sigma^*(s(1) \cdots s(i-1)) = \sum_{t^*=s} \frac{\mathbf{b}_{i-1}(t(1) \cdots t(i-1))}{\mathbf{b}^*_{i-1}(s(1) \cdots s(i-1))} \cdot \sigma(t(1) \cdots t(i-1))[\mathrm{intfX}(P)],$$

where $t^* = t[\mathrm{intfX}(P) \cup \mathrm{extlX}(P)]$. Recall that if $P = P_1 \| P_2$ is defined, then it must be that $\mathrm{intfX}(P) \cup \mathrm{extlX}(P) = \mathrm{intfX}(P_1) \cup \mathrm{extlX}(P_1) = \mathrm{intfX}(P_2) \cup \mathrm{extlX}(P_2)$.

**Lemma 2.** *Let $P = P_1 \| P_2$ be a probabilistic module, and let $\mathbf{b} \in L(P_1 \| P_2)$ be the outcome of a scheduler $\sigma = \sigma_{Env} \times \sigma_{P_1} \times \sigma_{P_2}$ with $\sigma_{Env} \in \mathrm{extl}\Sigma(P)$ and $\sigma_{P_j} \in \prod_{A \in \mathrm{Atoms}(P_j)} \mathrm{atom}\Sigma(A)$ for $j = 1, 2$. For every $(\mathrm{intfX}(P) \cup \mathrm{extlX}(P))$-trace $t$ of length $i$, we have $\mathbf{b}^*_i(t) = \mathbf{b}^*_{i-1}(t(1) \cdots t(i-1)) \cdot (\sigma_{Env} \times \sigma^*_{P_1} \times \sigma^*_{P_2})(t(i))$, where $\sigma^*_{P_j}$ is the observable scheduler w.r.t. $\mathbf{b}[X(P_j)]$ for $j = 1, 2$.*

Using this lemma, the soundness of the assume-guarantee rule can be proved in a fashion similar to that for nonprobabilistic systems like *reactive modules* [AH99].

## References


[AH99] R. Alur and T.A. Henzinger. Reactive modules. *Formal Methods in System Design* 15:7–48, 1999.

[AL95] M. Abadi and L. Lamport. Conjoining specifications. *ACM Trans. Programming Languages and Systems*, 17:507–534, 1995.

[BdA95] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lect. Notes in Comp. Sci.*, pages 499–513. Springer-Verlag, 1995.

[dA98] L. de Alfaro. Stochastic transition systems. In *Concurrency Theory*, volume 1466 of *Lect. Notes in Comp. Sci.*, pages 423–438. Springer-Verlag, 1998.

[dAKN+00] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lect. Notes in Comp. Sci.*, pages 395–410. Springer-Verlag, 2000.

[Der70] C. Derman. *Finite State Markovian Decision Processes.* Academic Press, 1970.

[Dil89] D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-independent Circuits.* The MIT Press, 1989.

[JL91] B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proc. Symp. Logic in Computer Science*, pages 266–277. IEEE Computer Society Press, 1991.

[Lam93] L. Lamport. Specifying concurrent program modules. *ACM Trans. Progamming Languages and Systems*, 5:190–222, 1993.

[Lyn96] N.A. Lynch. *Distributed Algorithms.* Morgan-Kaufmann, 1996.

[MC81] J. Misra and K.M. Chandy. Proofs of networks of processes. *IEEE Trans. Software Engineering*, SE-7:417–426, 1981.

[McM97] K.L. McMillan. A compositional rule for hardware design refinement. In *Computer-Aided Verification*, volume 1254 of *Lect. Notes in Comp. Sci.*, pages 24–35. Springer-Verlag, 1997.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, 1995. Technical Report MIT/LCS/TR-676.

[SL94] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. In *Concurrency Theory*, volume 836 of *Lect. Notes in Comp. Sci.*, pages 481–496. Springer-Verlag, 1994.

[Var85] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state systems. In *Proc. Symp. Foundations of Computer Science*, pages 327–338. IEEE Computer Society Press, 1985.