Probabilistic Rely-guarantee Calculus *

Annabelle McIver  Tahiry Rabehaja †

Department of Computing, Macquarie University, Australia

Georg Struth

Department of Computer Science, The University of Sheffield

Abstract

Jones’ rely-guarantee calculus \(^1\) for shared variable concurrency is extended to include probabilistic behaviours. We use an algebraic approach which combines and adapts probabilistic Kleene algebras with concurrent Kleene algebra. Soundness of the algebra is shown relative to a general probabilistic event structure semantics \(^21\). The main contribution of this paper is a collection of rely-guarantee rules built on top of that semantics. In particular, we show how to obtain bounds on probabilities by deriving rely-guarantee rules within the true-concurrent denotational semantics. The use of these rules is illustrated by a detailed verification of a simple probabilistic concurrent program: a faulty Eratosthenes sieve.

1 Introduction

The rigorous study of concurrent systems remains a difficult task due to the intricate interactions between the components of a system. A formal framework for concurrent systems ultimately depends on the kind of concurrency considered. Jones’ rely-guarantee calculus provides a mathematical foundation for the correctness of programs with shared variables concurrency \(^1\). This paper extends Jones’ calculus to the quantitative correctness of probabilistic concurrent programs.

Probabilistic programs have become popular due to their ability to express quantitative rather than limited qualitative properties. Probabilities are particularly important for protocols requiring security based on the unpredictability of probabilistic choices. The sequential probabilistic semantics, originating with Kozen \(^2\) and Jones \(^3\), have been extended with nondeterminism \(^1\)\(^5\), to yield

\(^*\)Preprint submitted for TCS-QAPL 2014.
\(^†\)The authors acknowledge the supports by the iMQRES Grant from Macquarie University and the Australia Research Council Discovery Grant DP1092464.
methods for quantitative reasoning using partial orders. We aim to obtain similar reasoning methods for probabilistic programs with shared variable concurrency.

The principle underpinning the rely-guarantee approach is compositionality in a strong sense, that is, the desired property of a system can be expressed as a function of its components. In the presence of probabilistic behaviours, finding a compositional semantics is usually hard. In fact, a problem of congruence (and hence non-compositionality) arises when considering the natural extension of trace-based semantics to probabilistic automata, where a standard workaround is to define a partial order based on simulations. In this paper, we define a similar construct to achieve compositionality. However, simulation-based equivalences are usually too discriminating for program verification. Therefore, we also use a weaker semantics that is essentially based on sequential behaviours. Such a technique has been motivated elsewhere where the sequential order is usually not a congruence. Therefore, the simulation-based order is used for properties requiring composition while the second order provides a tool that captures the sequential behaviours of the system.

Hoare et al. developed an algebraic formalisation of Jones rely-guarantee framework. Algebras provide an abstract view of the studied program and focuses more on control flows rather than data flows. All the rely-guarantee rules described in were proved algebraically from a finite set of axioms. Often, the verification of these axioms on an intended semantics is usually easier than proving the inference rules directly in that semantics. Moreover, every structure satisfying these laws will automatically incorporate a direct interpretation of the rely-guarantee rules, as well as additional rules that can be used for program refinements. Therefore, we also adopt an algebraic approach to the quantitative extension of rely-guarantee, that is, we will establish some basic algebraic properties of a concrete event structure model and derive the rely-guarantee rules using algebraic reasoning.

In summary, the main contribution of this paper is the development of a mathematical foundation for probabilistic rely-guarantee calculi. The rules are expressed algebraically, and we illustrate their use on an example based on Sieve of Eratosthenes which incorporates a probability of failure. We also outline two rules that provide probabilistic lower bounds for the correctness of the concurrent execution of multiple components.

A short summary of the algebraic approach to rely-guarantee calculus and the extension to probabilistic programs are found respectively in Section 2 and Section 3 and 4 are devoted to the construction of a denotational model for probabilistic concurrent programs. Section 7 closes this paper with a detailed verification of the faulty Eratosthenes sieve.

2 Non-probabilistic rely-guarantee calculus

The rely-guarantee approach, originally put forward by Jones, is a compositional method for verifying and developing large concurrent systems. An algebraic formulation was proposed more recently by Hoare et al. using the al-
gebraic treatment of Jones quintuples [1], which in turn is decomposed into a Hoare triple [10] and an order constraint. Notice that the validity of a Hoare triple \( \{P\}E\{Q\} \) is expressed using an inequality \( \leq \), which encodes the fact that all behaviours of \( P\cdot E \) are found in \( Q \), at the trace level ((·) denotes the sequential composition of programs).

Given a suitable definition of the concurrent composition (∥), a Jones quintuple, denoted by \( \{P R\}E\{G Q\} \), holds iff the Hoare triple \( \{P\}R\|E\{Q\} \) holds and \( E \) guarantees \( G \), for a given rely condition \( R \) and guarantee condition \( G \). Intuitively, a rely condition specifies the interference from the environment, while a guarantee condition defines the interference a component will exert on the environment. Thus, if the parallel composition of \( R \) and \( E \) executes from a state satisfying \( P \) terminates, then \( R\|E \) will establish a state satisfying \( Q \), provided that all atomic executions in \( E \) are satisfying \( G \). Algebraically, the quintuple \( \{P R\}E\{G Q\} \) is valid iff

\[
P \cdot (R\|E) \leq Q \quad \text{and} \quad E \leq G
\]

hold [8].

The main ingredients in the definition of a rely-guarantee quintuple are the operation (∥) and the order \( \leq \). Both of these constructs have precise meaning when considering a concrete semantics for concurrent program. Dingel [11] and Coleman and Jones [12] have proven the soundness of rely-guarantee rules with respect to a trace-based semantics, where \( \leq \) is the set inclusion and ∥ interleaves atomic actions taken from different processes to produce a connected sequence of process-indexed pairs of states, namely, Aczel traces [13]. Each element of such a sequence denotes either an action performed by the underlined system or by the environment in which the system is run.

While the approaches taken by Coleman and Jones [12], Dingel [11] and Hoare et al. [8] are all related to non-probabilistic programs, the main difference can be found in the respective proofs of the rely-guarantee rules. Coleman, Jones and Dingel’s proofs are directly based on the semantics of concurrent programs, namely, sets of Aczel traces. In contrast, Hoare et al. provided an algebraic proof which is, perhaps, simpler and is valid for all models satisfying the algebraic axioms. This paper follows Hoare et al.’s development but for probabilistic programs.

An important example of inference rule, that was proven in the trace-based semantics [1, 11] as well as the more abstract algebraic setting [8], demonstrates how to combine the rely-guarantee specifications of components into rely-guarantee conditions for the whole system:

\[
\frac{\{P R\}E\{G Q\} \quad \{P R'\}E'\{G' Q'\} \quad G \leq R' \quad G' \leq R}{\{P R \cap R'\}E\|E'\{G \cup G' \cap Q'\}.}
\]  

This rule implies if \( E \) and \( E' \) satisfy the premises then \( E\|E' \) will satisfy both \( Q \) and \( Q' \) when run in an environment satisfying \( R \cap R' \). Moreover, \( E\|E' \) will guarantee the condition \( G \cup G' \).
In Section 5, we will provide a suitable extension of the rely-guarantee formalism, in particular Rule (2), to probabilistic concurrent programs. The soundness of such a formalism will be shown against a semantic space that allows sequential probabilistic programs to include concurrent behaviours.

3 Sequential probabilistic programs

We start by giving a brief summary of the denotation of sequential probabilistic programs using the powerdomain construction of McIver and Morgan [5]. All probabilistic programs will be considered to have a finite state space denoted by \( \Omega \). A distribution over the set \( \Omega \) is a function \( \mu : \Omega \to [0,1] \) such that \( \sum_{s \in \Omega} \mu(s) = 1 \). The set of distributions over \( \Omega \) is denoted by \( \mathbb{D} \Omega \). Since \( \Omega \) is a finite set, we identify a distribution with the associated measure. For every \( \mu \in \mathbb{D} \Omega \) and \( O \subseteq \Omega \), we write \( \mu(O) = \sum_{s \in O} \mu(s) \). An example of distribution is the point distribution \( \delta_s \), centred at the state \( s \in \Omega \), such that

\[
\delta_s(s') = \begin{cases} 
1 & \text{if } s = s', \\
0 & \text{otherwise.}
\end{cases}
\]

A (nondeterministic) probabilistic program \( r \) is denoted by a map of type \( \Omega \to \mathbb{P} \mathbb{D} \Omega \) such that \( r(s) \) is a non-empty, topologically closed and convex subset of \( \mathbb{D} \Omega \) for every \( s \in \Omega \). The set \( \mathbb{D} \Omega \) is a topological sub-space of the finite product \( \mathbb{R} \Omega \) (endowed with the usual product topology), and the topological closure is considered with respect to the induced topology on \( \Omega \). We denote by \( \mathbb{H}_1 \Omega \) the set of probabilistic programs. Notice that the set \( \mathbb{D} \Omega \) contains only distributions instead of the subdistributions considered by McIver and Morgan [5]. Therefore, we can only model nondeterministic programs that are terminating with probability 1.

The probabilistic combination of two probabilistic programs \( r \) and \( r' \) is defined as (5 Definition 5.4.5)

\[
(r \oplus_p r')(s) = \{ \mu \oplus_p \mu' \mid \mu \in r(s) \land \mu' \in r'(s) \},
\]

where \( (\mu \oplus_p \mu')(s) = p\mu(s) + (1 - p)\mu'(s) \) for every state \( s \in \Omega \).

Nondeterminism is obtained as the set of all probabilistic choices (5 Definition 5.4.6). That is,

\[
(r + r')(s) = \cup_{p \in [0,1]} (r \oplus_p r')(s).
\]

\(^1\)These healthiness conditions are set out and fully explained in the work of McIver and Morgan [5].

---

[5] McIver, A. C., & Morgan, R. L. (2002). Ako Rely-Guarantee Calculus: The Large-Scale System Concurrency Control Cookbook. Chapman & Hall/CRC.

[3] Jones, R. D. (1980). A calculus for concurrent programming. Journal of Computer and System Sciences, 20(2), 200-254.
The sequential composition of \( r \) by \( r' \) is defined as (\cite{lncs} Definition 5.4.7):
\[
(r \cdot r')(s) = \{ f \ast \mu \mid f \sqsubseteq_H r' \land f \in J_1 \Omega \land \mu \in r(s) \}
\]
(5)

where
\[
(f \ast \mu)(s') = \sum_{s'' \in \Omega} f(s'')(s') \mu(s'')
\]
for every state \( s' \in \Omega \).

For \( r \in H_1 \Omega \), the binary Kleene star \( r \ast r' \) is the least fixed point of the function \( f_{r,r'}(X) = r' + r \cdot X \) in \( H_1 \Omega \). It has been shown in \cite{lncs} that the function \( r' \mapsto r' \ast r \) is continuous —it preserves directed suprema. Notice that a topological closure is sometimes needed to ensure that we obtain an element of \( H_1 \Omega \). Hence, the Kleene star \( r \ast r' \) is the program such that
\[
r \ast r'(s) = \bigcup_n f^n_{r,r'}(\bot)(s)
\]
where \( \overline{A} \) is the topological closure of the set \( A \subseteq D \Omega \) and the constant \( \bot \) is defined, as usual, such that \( r'' \cdot \bot = \bot \cdot r'' = \bot, \bot + r'' = r'' \) and \( \bot \sqsubseteq_H r'' \) for every \( r'' \in H_1 \Omega \cup \{ \bot \} \).

We introduce tests, which are used for conditional constructs, following the idea adopted in various algebras of programs. We define a test to be a map \( b : \Omega \to \mathbb{P} \Omega \) such that \( b(s) \subseteq \{ \delta_{s} \} \). Indeed, an “if statement” is modelled algebraically as \( b \cdot r + (\neg b) \cdot r' \). The sub-expression \( b \cdot r(s) \) still evaluates to \( \emptyset \) if \( b(s) \) is empty, but care should be taken to avoid expressions such as \( r \cdot b \) (if \( f \) is a deterministic refinement of \( b \), then \( f(s'')(s') \) may have no meaning if \( b(s'') = \emptyset \)).

A test that is always false can be identified with \( \bot \).

We denote by \( \overline{H} \Omega \) the set of tests together with the set of probabilistic programs. The refinement order \( \sqsubseteq_H \) is extended to \( \overline{H} \Omega \) in a straightforward manner. For every tests \( b \), we have \( b \sqsubseteq_H \delta \); hence, we refer to tests as subidentities. We refer to the elements of \( \overline{H} \Omega \) as programs, unless otherwise specified.

### 4 An event structures model for probabilistic concurrent programs

In this section, we outline a new denotational semantics for probabilistic concurrent programs based on event structures. This construction is necessary to ensure the soundness of the extended rely-guarantee formalism presented in this paper.

Event structures have been thoroughly used to model systems exhibiting true concurrency \cite{weil,weil2}. Though there are many variants, we build our semantics based on Langerak’s bundle event structure \cite{langerak} which has been successfully extended to incorporate quantitative features \cite{weil2,weil}.

**Definition 4.1.** A structure \( \mathcal{E} = (E, \rightarrow, \#, \lambda, \Phi) \) is a bundle event structure with internal probability (i.e. an ipBES) if
• \# is an irreflexive symmetric binary relation on \( E \), called conflict relation.
• \( \mapsto \subseteq P E \times E \) is a bundle relation, i.e., if \( x \mapsto e \) for some \( x \subseteq E \) and \( e \in E \), then \( x \# x \).
• \( \lambda : E \to \mathbb{H}_1 \Omega \), i.e., it labels event with (atomic) probabilistic programs.
• \( \Phi \subseteq PE \) such that \( x \# x \) holds for every \( x \in \Phi \).

The finite state space \( \Omega \) of the programs used as labels is fixed.

Given a set \( E \), a binary relation \( \# \subseteq E \times E \) and two subsets \( x, x' \subseteq E \), the predicate \( x \# x' \) holds iff for every \( (e, e') \in x \times x' \) such that \( e \neq e' \), we have \( e \# e' \). The following definition is an adaptation of Langerak’s definition of bundle event structures [16].

The intuition behind this definition is that events are occurrences of atomic program fragments, i.e., it will happen without interferences from an environment. Hence, we need to distinguish all atomic program fragments when translating a program into a bundle event structure. Atomic programs can be achieved by creating a construct that forces atomicity. Examples of such a technique include the “atomic brackets” [19]. In this paper, we will state which actions are atomic rather than using such a device.

Given an ipBES \( E \), a finite trace of \( E \) is a sequence of events \( e_1 e_2 \ldots e_n \) such for every different \( 1 \leq i, j \leq n \), \( -e_i \# e_j \) and if \( j = i + 1 \) then there exists \( x \subseteq E \) such that \( x \mapsto e_j \) and \( e_i \in x \) [16, 17, 20]. In other words, a trace safe (an event may occur only when it is enabled) and is conflict free. The set of all finite traces of \( E \) is denoted by \( T(E) \).

### 4.1 Schedulers on ipBES

To obtain a sequential equivalence on bundle event structures with internal probability, we define the notion of scheduler that will provide a distribution on states, from “maximal execution traces”.

Firstly, we need the notion of subdistributions [5]. A subdistribution is a map \( \mu : \Omega \to [0, 1] \) such that \( \sum_{s \in \Omega} \mu(s) \leq 1 \). The set of subdistributions over \( \Omega \) is denoted by \( D_{\leq 1} \Omega \).

**Definition 4.2.** A scheduler \( \sigma \) on an ipBES \( E \) is a map

\[
\sigma : T(E) \to [(E \times \Omega) \to \mathbb{D}_{\leq 1}\Omega]
\]

such that for all \( \alpha \in T(E) \):

1. \( \text{dom}(\sigma(\alpha)) = \{(e, s) \mid \alpha e \in T(E) \land s \in \Omega\} \),
2. there exists a function \( w : E \times \Omega \to [0, 1] \) such that, for every \( (e, s) \in \text{dom}(\alpha) \), \( \sigma(\alpha)(e, s) = w(e, s)\mu \) for some \( \mu \in \lambda(e)(s) \).
3. for every \( s \in \Omega \), we have \( \sum_{(e, s) \in \text{dom}(\sigma(\alpha))} w(e, s) = 1 \),
4. For every \((e, s) \in \text{dom}(\sigma(\alpha))\), if \(\lambda(e) = \emptyset\), then \(w(e, s) = 0\) and \(\sigma(\alpha)(e, s) = 0\) (the subdistribution that evaluates to 0 everywhere).

The set of all schedulers on \(\mathcal{E}\) is denoted by \(\text{Sched}(\mathcal{E})\).

Property 1 says that we may schedule an event if it does not depend on unscheduled events.

Property 2 states that, given a trace \(\alpha\), the scheduler will resolve the non-determinism between events enabled after \(\alpha\), using the weight function \(w\). This may include immediate conflicts or interleaving of concurrent events. Moreover, the scheduler has access to the current program state when resolving that non-determinism. That is, \(w(e, s)\) is the probability that the event \(e\) is scheduled, knowing that the program state is \(s\). If the event \(e\) is successfully scheduled, then the scheduler performs a last choice of distribution, say \(\mu\) from \(\lambda(e)(s)\), to generate the next state of the program.

Property 3 ensures that when the state \(s\) is known, then the choice between the events, enabled after the trace \(\alpha\), is indeed probabilistic.

Property 4 says that a scheduler is forced to choose events whose label does not evaluate to the empty set at the current state of the program. This is particularly important when the program contains conditionals and the label of an event is a test. A scheduler is forced to choose the branch whose test holds. If two tests hold at state \(s\), then a branch is chosen probabilistically using the weight function \(w\).

The motivation behind Property 4 is to ensure that, for every trace \(\alpha\) such that \(\text{dom}(\alpha) \neq \emptyset\), and every state \(s \in \Omega\), we have

\[
\sum_{(e, s) \in \text{dom}(\sigma(\alpha))} \sigma(\alpha)(e, s) \in \mathbb{D}\Omega
\]

i.e. that sum is indeed a distribution. To ensure that a scheduler satisfying that condition can be constructed, we restrict ourselves to feasible event structures. Given an element \(r \in \mathbb{P}\Omega\), we write \(\text{dom}(r) = \{s \mid r(s) \neq \emptyset\}\).

**Definition 4.3.** An ipBES \(\mathcal{E}\) is feasible if for every \(\alpha \in \mathcal{T}(\mathcal{E})\) such that \(\text{dom}(\alpha) \neq \emptyset\), we have \(\cup_{\alpha \in \mathcal{T}(\mathcal{E})} \text{dom}(\lambda(e)) = \Omega\).

A consequence of this assumption is that an “if clause” always needs to have a corresponding “else clause”.

**Example 4.4.** Let us consider the program \(r \cdot (\delta + r)\). In this program, \(r\) is atomic deterministic, so the associated event structure is constructed in a straightforward manner. Let us denote by \(r_i\) the event associated to the \(i\)th occurrence of \(r\) \((i \in \{1, 2\})\) in the expression \(r \cdot (\delta + r)\). A scheduler \(\sigma\) on the event structure associated to \(r \cdot (\delta + r)\) is characterised by a weight function \(w: \{r_1, r_2, \delta\} \times \Omega \rightarrow [0, 1]\) resolving the choice \(\delta + r\). In fact we can write \(\sigma(r_1)(r_2, s) = (1 - w(r_2, s))r(s)\).
4.2 Computation function of an ipBES

We define the runs of a bundle event structure $E$ with a given scheduler $\sigma$ as follows. Let $T_n(E)$ be the set of traces of length $n \in \mathbb{N}$. For $\sigma \in \text{Sched}(E)$, the computation sequence of $E$ with respect to $\sigma$ is a sequence of partial functions $\varphi_n : T(E) \to \mathbb{D}_{\leq 1}^\Omega$ such that $\text{dom}(\varphi_n) = \bigcup_{k \leq n} T_k(E)$

1. $\varphi_0(\emptyset) = \delta_s$ where $s$ is the initial state,
2. if $\alpha e \in T_{n+1}(E)$ then
   
   $$\varphi_{n+1}(\alpha e)(s) = \sum_{t \in \Omega} \sigma(\alpha)(e, t)(s)\varphi_n(\alpha)(t)$$
   
   and $\varphi_{n+1}(\alpha e) = \varphi_n(\alpha e)$ otherwise.

If we need to emphasises that this computation function is computed using a specific initial state $t \in \Omega$, then we write $\varphi_{n,t}$ instead of $\varphi_n$.

The complete run of $E$ with respect to $\sigma$ is the limit $\varphi$ of that sequence i.e. $\varphi = \bigcup_n \varphi_n$, which exists because $\varphi_n$ defines a sequence of partial functions such that $\varphi_n$ is the restriction of $\varphi_{n+1}$ on $\text{dom}(\varphi_n)$. Since we consider finite traces only, we have $\text{dom}(\varphi) = T(E)$. The behaviour of $E$ with respect to $\sigma$ from the initial state $s$ is defined by the sum

$$\sigma_s(E) = \sum_{\alpha \in T_{\text{max}}(E)} \varphi(\alpha),$$

where $T_{\text{max}}(E)$ is the set of finite, maximal (with respect to the prefix ordering) traces of $E$.

**Proposition 4.5.** For every bundle event structure $E$, scheduler $\sigma \in \text{Sched}(E)$ and initial state $s$, $\sigma_s(E)$ is a subdistribution.

Let us write $T$ (resp. $T_n$, resp. $T_{\text{max}}$) instead of $T(E)$ (resp. $T_n(E)$, resp. $T_{\text{max}}(E)$) when no confusion may arise.

**Proof.** Let $\varphi$ be the complete run of $E$ with respect to a given scheduler $\sigma$. We show by induction on $n$ that

$$\mu_n(\Omega) = \sum_{\alpha \in T_n \cup (T_{\text{max}} \cap \text{dom}(\varphi_n))} \varphi(\alpha)(\Omega) = \sum_{t \in \Omega} \sum_{\alpha \in T_n \cup (T_{\text{max}} \cap \text{dom}(\varphi_n))} \varphi(\alpha)(t) = 1$$

The set $T_n$ contains all traces of length $n$ and the set $T_{\text{max}} \cap \text{dom}(\varphi_n)$ contains maximal traces of length less than or equal to $n$.

For the base case $n = 0$, we have $\mu_0(\Omega) = \varphi(\emptyset)(\Omega) = \delta_s(\Omega) = 1$, where $s$ is
the initial state. Assume the induction hypothesis \( \mu_n(\Omega) = 1 \). We have

\[
\mu_{n+1}(\Omega) = \sum_{\alpha \in T_{n+1} \cup (T_{\text{max}} \cap \text{dom}(\varphi_n))} \varphi(\alpha)(\Omega)
\]

\[
= \sum_{\alpha \in T_{n+1}} \varphi(\alpha)(\Omega) + \sum_{\alpha \in T_{\text{max}} \cap \text{dom}(\varphi_n)} \varphi(\alpha)(\Omega)
\]

\[
= \sum_{\alpha \in T_{n+1}} \sum_{t \in \Omega} \sigma(\alpha)(e,t)(\Omega) \varphi(\alpha)(t) + \sum_{\alpha \in T_{\text{max}} \cap \text{dom}(\varphi_n)} \varphi(\alpha)(\Omega)
\]

\[
= \sum_{\alpha \in T_n \setminus T_{\text{max}}} \sum_{t \in \Omega} \sigma(\alpha)(e,t)(\Omega) \varphi(\alpha)(t) + \sum_{\alpha \in T_{\text{max}} \cap \text{dom}(\varphi_n)} \varphi(\alpha)(\Omega)
\]

\[
= \sum_{\alpha \in T_n \setminus T_{\text{max}}} \sum_{\alpha \in T_{\text{max}} \cap \text{dom}(\varphi_n)} \varphi(\alpha)(\Omega)
\]

\[
= \mu_{n}(\Omega) = 1.
\]

The square-bracketed term equals 1 because of Properties 2 and 3 of the scheduler \( \sigma \). Therefore, each partial computation \( \varphi_n \) is a probability distribution when restricted on \( T_n \cup (T_{\text{max}} \cap \text{dom}(\varphi_n)) \), and hence the limit \( \nu \) is a subdistribution on \( T_{\text{max}} \). It does not necessarily add up to 1 because elements of \( T_{\text{max}} \) are finite maximal traces only and non-termination will decrease that quantity (we assume that the empty sum is 0. This occurs when there are no maximal traces).

As a consequence of this proposition, we will denote by \( \text{Sched}_1(\mathcal{E}) \) the set of schedulers of \( \mathcal{E} \) such that, for every initial state \( s \), \( \sigma_s(\mathcal{E}) \) is a distribution. A scheduler from \( \text{Sched}_1(\mathcal{E}) \) will generate a sequential behaviour that terminates with probability 1.

### 4.3 Sequential reduction of an ipBES

We construct a semantics map \([\cdot]\) that transforms each event structure to an element of \( \mathbb{H}_1\Omega \). Recall that \( \mathbb{H}_1\Omega \) can express probabilistic programs that terminate with probability 1. Hence, this paper is restricted to partial correctness.

Given a feasible event structure \( \mathcal{E} \), we define

\[
[\mathcal{E}](s) = \text{conv} \{ \sigma_s(\mathcal{E}) \mid \sigma \in \text{Sched}_1(\mathcal{E}) \}
\]

where \( \text{conv}(A) \) (resp. \( \overline{A} \)) is the convex (resp. topological) closure of the set of distributions \( A \) in \( \mathbb{R}^\Omega \). We restrict ourselves to feasible and terminating event structures, i.e., such that \( \text{Sched}_1(\mathcal{E}) \) is non-empty (except for the special element 0 defined below, which has no sequential interpretation).

**Definition 4.6.** Let \( \mathcal{E}, \mathcal{F} \) be two feasible event structures. We say that \( \mathcal{E} \) (sequentially) refines \( \mathcal{F} \), denoted by \( \mathcal{E} \sqsubseteq \mathcal{F} \), if \([\mathcal{E}] \subseteq \mathbb{H}[\mathcal{F}] \) holds in \( \mathbb{H}_1\Omega \).
The relation ⊑ is a preorder on bundle event structures with internal probability. Whilst this order is not a congruence, it is used to specify the desired sequential correctness using event structures.

We now provide interpretations of the operations (+, ·, *, ||) and constants 0, 1 on event structures with disjoint sets of events. These definitions ensure that we can inductively translate program texts into event structure objects.

- The algebraic constant 1 is interpreted as $(e, ∅, ∅, \{(e, 1)\}, \{e\})$ where the 1 in the expression $(e, 1)$ is the identity of $\mathbb{H}_1\Omega$.
- The algebraic constant 0 is interpreted as $(∅, ∅, ∅, ∅, ∅)$.
- Each atomic action $r \in \mathbb{H}_1\Omega$ is associated with $(\{e\}, ∅, ∅, \{(e, r)\}, \{e\})$. This event structure is again denoted by $r$.
- The nondeterministic choice between the event structures $E$ and $F$ is constructed as
  
  $$E + F = (E \cup F, \#E + \#F, \mapsto E \cup \mapsto F, \lambda_E \cup \lambda_F, \{ (x \cup y) \mid x \in \Phi_E \land y \in \Phi_F \})$$

  where $\#E + \#F = \cup_{x \in \Phi_E} \lambda_E \cup \lambda_F \sym(x \times y) \cup \#E \cup \#F \cup \sym(\in(E) \times \in(F))$ and $\sym$ is the symmetric closure of a relation on $E \cup F$. The square-bracketed set ensures that every final event in $E$ is in conflict with every final event in $F$. This ensures that, if $z \in \Phi_{E + F}$, then $z \# z$.

- The sequential composition of $E$ by $F$ is
  
  $$E \cdot F = (E \cup F, \#E \cup \#F, \mapsto E \cup \mapsto F \cup \{ (x \mapsto e) \mid e \in \in(F) \land x \in \Phi_E \}, \lambda_E \cup \lambda_F, \Phi_E \cup \Phi_F)$$

- The concurrent composition of $E$ and $F$ is
  
  $$E \parallel F = (E \cup F, \#E \cup \#F, \mapsto E \cup \mapsto F, \lambda_E \cup \lambda_F, \Phi_E \cup \Phi_F)$$

- The binary Kleene product of $E$ and $F$ is the supremum of the sequence
  
  $$F, F + E \cdot F, F + E \cdot (F + E \cdot F), \ldots$$

  of bundle event structures with respect to the $\omega$-complete sub-BES order [21].

**Example 4.7.** Given a program $r_1, r_2, r_3 \in \mathbb{H}_1\Omega$, the event structure associated to $(r_1 || r_2) + r_3$ is

$$\{(e_1, e_2, e_3), \sym(\{(e_i, e_3) \mid i \in \{1, 2\}\}), 0, \lambda, \{(e_i, e_3) \mid i \in \{1, 2\}\})$$

where $\lambda(e_i) = r_i$. Notice that $e_1 \# e_3$ and $e_2 \# e_3$, but $e_1$ and $e_2$ are concurrent.
convex and topologically closed, we deduce that \( \{ \} \). Hence, \( \conv(E) \subseteq \conv(E + F) \). Moreover, \( 0 \) will disappear in mixed expressions because of these properties. We now show that the operations (+) and (·) are preserved by the map \( \conv \).

The case of the binary Kleene star (\( * \)) is proven in Proposition 4.15.

**Proposition 4.8.** For \( E, F \) non-zero, feasible and terminating event structures, we have \( [E + F] = [E] + [F] \) and \( [E \cdot F] = [E] \cdot [F] \).

**Proof.** For the case of nondeterminism (+), let \( s \in \Omega \) be the initial state and \( \mu \in [E + F](s) \). Let us first assume that \( \mu = \sigma_s(E) \) for some \( \sigma \in \text{Sched}_1(E + F) \). By definition of the sum \( E + F \), the set of events \( E \) and \( F \) are disjoint, so we can define two schedulers \( \sigma^E \in \text{Sched}_1(E) \) and \( \sigma^F \in \text{Sched}_1(F) \) as follows. Let \( \alpha \in T(E + F) \) and \( (e, t) \in \text{dom}(\sigma(\alpha)) \), we define

\[
\sigma^E(\alpha)(e, t) = \begin{cases} 
\sigma(\alpha)(e, t) & \text{if } \alpha \in T(E) \setminus \{\emptyset\}, \\
\frac{\sigma(\alpha)(e, t)}{p^t} & \text{if } \alpha = \emptyset
\end{cases}
\]

where \( p^t = \sum_{\alpha' \in \text{in}(\alpha)} w(e, t) \), \( w \) is the weight function associated to \( \sigma \) at the trace \( \emptyset \) and \( s \) is the initial state. The real number \( p^t \) is just a normalisation constant required by Property 3 in the definition of schedulers. The scheduler \( \sigma^F \) is similarly defined. It follows directly from these definitions of \( \sigma^E \) and \( \sigma^F \) that \( \sigma(\emptyset)(e, t) = p^t \sigma(\emptyset)(e, t) + p^t \sigma(\emptyset)(e, t) \) where \( p^t + p^t = 1 \) because of Property 3. Hence, \( \sigma_s(E) = p^t \sigma_s^E \) and \( \sigma_s^F \) i.e. \( \sigma_s(E) \in [E] \cup [F] \). Since \( [E] \cup [F] \) is convex and topologically closed, we deduce that \( [E + F](s) \subseteq ([E] \cup [F])(s) \).

For the converse inclusion \( ([E] \cup [F])(s) \subseteq [E + F](s) \), notice that \( \conv(A) = \conv(\overline{A}) \) holds for every subset \( A \subseteq \Omega \). If we write \( A = \{ \sigma_s(E) \mid \sigma \in \text{Sched}_1(E) \} \) and \( B = \{ \sigma_s(F) \mid \sigma \in \text{Sched}_1(F) \} \), then

\[
([E] + [F])(s) = \conv(\overline{\text{conv}(A) \cup \text{conv}(B)}) = \conv(A \cup B)
\]

But it is clear that \( A \subseteq [E + F](s) \) (a scheduler that does not choose \( F \) is possible because \( E \) is feasible) and \( B \subseteq [E + F](s) \). Therefore, \( ([E] + [F])(s) = \text{conv}(A \cup B) \subseteq [E + F](s) \) because the last set is convex and topologically closed.

The sequential composition is proven using a similar reasoning. Let \( E, F \) be two bundle event structures satisfying the hypothesis, and \( \mu \in [E \cdot F](s) \) for some initial state \( s \in \Omega \). Firstly, let us assume that there is a scheduler \( \sigma \) on \( E \cdot F \) such that \( \mu = \sigma_s(E \cdot F) \). Since schedulers are inductively constructed, there exists \( \sigma^E \in \text{Sched}_1(E) \) and \( \sigma^F \in \text{Sched}_1(F) \) such that

\[
\sigma(\alpha)(e, t) = \begin{cases} 
\sigma^E(\alpha)(e, t) & \text{if } \alpha \in T(E), \\
\sigma^F(\alpha')(e, t) & \text{if } \alpha = \alpha' \alpha'' \text{ and } (\alpha', \alpha'') \in \text{T}_{\max}(E) \times T(F).
\end{cases}
\]

\[\text{If } p^t = 0, \text{ then } \sigma \in \text{Sched}_1(F)\]
Let us denote by $\varphi_n$ and $\varphi_n^σ$ (resp. $\varphi_n^{σ,e}$) the computation sequences associated to the respective schedulers $σ$ and $σ^e$ (resp. $σ^e$) from the initial state $s$ (resp. $e$). It follows directly that $\varphi_n(α) = \varphi_n^σ(α)$ for every $α \in T_n(\mathcal{E})$. If $α' \in T_{\text{max}}(\mathcal{E}) \cap T_n(\text{max})$ and $e \in \text{in}(F)$ then, for every state $u \in Ω$,

$$\varphi_{n+1}(α'e)(u) = \sum_{t \in Ω} \sigma^F(\emptyset)(e,t)(u)\varphi(α')(t).$$

Similarly, we have

$$\varphi_{n+1}(α'e'e)(u) = \sum_{t \in Ω} \sigma^F(e)(e,t')(u)\left[\sum_{t' \in Ω} \sigma^F(\emptyset)(e,t')(t')\varphi(α')(t)\right]$$

$$= \sum_{t \in Ω} \left[\sum_{t' \in Ω} \sigma^F(e)(e,t')(u)\sigma^F(\emptyset)(e,t')(t')\right]\varphi(α')(t)$$

$$= \sum_{t \in Ω} \varphi_{n+1}(u)\varphi(α')(t).$$

By simple induction on the length of $α''$, we deduce that

$$\varphi(α'α'')(u) = \sum_{t \in Ω} \varphi_n(α'')(u)\varphi(α')(t),$$

where $\varphi_n^F$ is the complete run obtained from the sequence $\varphi_{n,t}$. It follows by definition of the sequential composition on $\mathbb{H}_1Ω$ (Equation 6), that

$$\sigma_s(\mathcal{E})(u) = \sum_{t \in Ω} \sigma_t^F(\mathcal{F})(u)\sigma_t^E(\mathcal{E})(t) ∈ \mathcal{E} \cdot \mathcal{F}(s),$$

for every state $u ∈ Ω$. Secondly, since $\mathcal{E} \cdot \mathcal{F}(s)$ is upclosed and topologically closed, we deduce that $\mathcal{E} \cdot \mathcal{F}(s) ⊆ \mathcal{E} \cdot \mathcal{F}(s)$. Conversely, if $μ ∈ \mathcal{E} \cdot \mathcal{F}(s)$, then either $μ(u) = \sum_{t \in Ω} \sigma_t^F(\mathcal{F})(u)\sigma_t^E(\mathcal{E})(t)$ or $μ$ is in the closure of the set of these distributions. Either way, the closure properties of $\mathcal{E} \cdot \mathcal{F}(s)$ implies that $\mathcal{E} \cdot \mathcal{F}(s) ⊆ \mathcal{E} \cdot \mathcal{F}(s)$.

### 4.4 Simulation for ipBES with tests

The partial order defined in Definition 4.6 provides a mean to compare the functional behaviours of different systems. However, it suffers from a congruence problem, i.e., there exists programs $\mathcal{E}, \mathcal{F}$ and $G$ such that $\mathcal{E} \subseteq \mathcal{F}$ but $\mathcal{F} \parallel G$ is not less than $\mathcal{F} \parallel G$. For probabilistic automata, a known technique to achieve congruence is to construct an order based on a notion of simulation [22]. We use a similar technique in this subsection.

We say that a trace $α$ is weakly maximal if it is maximal or there exist some events $e_1, \ldots, e_n$ such that $αe_1 \cdots e_n ∈ T_{\text{max}}(\mathcal{E})$ and $δ ∈ H λ(e_i)$ for every $1 ≤ i ≤ n$.

3Remind that $ϕ_{n,t}$ is the computation function computed given the initial state $t$. 

12
Definition 4.9. A function \( f : \mathcal{T}(\mathcal{E}) \to \mathcal{T}(\mathcal{F}) \) is called a t-simulation:

- if \( f(\emptyset) = \emptyset \) and \( f^{-1}(\beta) \) is a finite set for every \( \beta \in \mathcal{T}(\mathcal{F}) \),
- if \( \alpha e \in \mathcal{T}(\mathcal{E}) \) then either:
  - \( f(\alpha e) = f(\alpha) \) and \( \lambda(e) \sqsubseteq_H \delta \) holds in \( \mathbb{H}_1\Omega \),
  - or there exists \( e' \in \mathcal{F} \) such that \( \lambda(e) \sqsubseteq_H \lambda(e') \) and \( f(\alpha e) = f(\alpha)e' \).
- if \( \alpha e \) is maximal in \( \mathcal{T}(\mathcal{E}) \) then \( f(\alpha e) = f(\alpha)e' \), for some \( e' \) (with \( \lambda(e) \sqsubseteq_H \lambda(e') \)), and \( f(\alpha e) \) is weakly maximal in \( \mathcal{T}(\mathcal{F}) \).

We say that \( \mathcal{E} \) is simulated by \( \mathcal{F} \), written \( \mathcal{E} \sqsubseteq_{\text{sim}} \mathcal{F} \) if there exists a simulation from \( \mathcal{E} \) to \( \mathcal{F} \). The equivalence generated by this preorder is denoted \( \equiv_{\text{sim}} \).

Notice that if \( f(\alpha) \) is maximal then \( \alpha \) is necessarily maximal.

The notion of t-simulation has been designed to correctly simulate event structures in the presence of tests. For instance, given a test \( b \), the simulation \( \delta \sqsubseteq_{\text{sim}} (b + \neg b) \) fails because a t-simulation is a total function and it does not allow the removal of “internal” events labelled with sub-identities during a refinement step. The finiteness condition on \( f^{-1}(\beta) \) ensures that we do not refine a terminating specification with a diverging implementation. For instance, without that constraint, we would be able write the refinement

\[
\text{if } (0 == 1) \text{ then } s := 0 \text{ else[if } (0 == 1) \text{ then } s := 0 \text{ else[...]] } \sqsubseteq_{\text{sim}} s := 0.
\]

However, this should not hold because the left hand sides is a non-terminating program and cannot refine the terminating assignment \( s := 0 \).

A t-simulation is used to compare bundle event structures without looking in details at the labels of events. It can be seen as a refinement order on the higher level structure of a concurrent program. Once a sequential behaviour has to be checked, we use the previously defined functional equivalence on event structures with internal probabilities.

Example 4.10. A t-simulation from \( b + \neg b \cdot r \) to \( 1 + r \) is given by the dotted arrow in the following diagram:

\[
\begin{array}{c}
\emptyset \\
\vdots \\
e_b \\
\vdots \\
e_{\neg b} \\
\vdots \\
e_\delta \\
\vdots \\
\rightarrow e_r \\
\end{array}
\]

This diagram shows the refinement of a nondeterministic choice with a conditional. Notice that t-simulations allow the introduction of sub-identities.

Proposition 4.11. The t-simulation relation \( \sqsubseteq_{\text{sim}} \) is a preorder.
Proof. Reflexivity follows from the identity function and transitivity is obtained by composing t-simulation which will generate a new t-simulation. Notice that care should be taken with respect to the third property of a t-simulation. If \( f : \mathcal{T}(E) \to \mathcal{T}(F), g : \mathcal{T}(F) \to \mathcal{T}(G) \) are t-simulations, \( \alpha e \in \mathcal{T}_{\text{max}}(E) \) and \( \lambda(e) \subseteq_{\mathbb{H}} \delta \), then \( f(\alpha e) = f(\alpha)e' \) for some \( e' \in F \) such that \( \lambda(e') \subseteq_{\mathbb{H}} \lambda(e') \). If \( \lambda(e') \subseteq_{\mathbb{H}} \delta \), then it is possible that \( g(f(\alpha)e') = g(f(\alpha)) \). However, since \( f(\alpha)e' \) is weakly maximal, \( g(f(\alpha)e') \) is also weakly maximal and we can find an event \( e'' \in G \) such that \( f(\alpha)e'' \) is weakly maximal and \( \lambda(e'') \subseteq_{\mathbb{H}} \lambda(e'') \). We then map \( \alpha e \) to \( g(f(\alpha))e'' \) in the generated t-simulation.

\[ \mathcal{E} \parallel \mathcal{F} \equiv_{\text{sim}} \mathcal{F} \parallel \mathcal{E} \quad (6) \]
\[ \mathcal{E} \parallel (\mathcal{F} \parallel \mathcal{G}) \equiv_{\text{sim}} (\mathcal{E} \parallel \mathcal{F}) \parallel \mathcal{G} \quad (7) \]
\[ X*Y \equiv_{\text{sim}} Y + X \cdot (X*Y) \quad (8) \]
\[ \mathcal{E} \subseteq_{\text{sim}} \mathcal{F} \Rightarrow \mathcal{G} \subseteq_{\text{sim}} \mathcal{G} \parallel \mathcal{F} \quad (9) \]
\[ \mathcal{E} \subseteq_{\text{sim}} \mathcal{F} \Rightarrow \mathcal{G} \subseteq_{\text{sim}} \mathcal{G} \parallel \mathcal{F} \quad (10) \]
\[ \mathcal{E} \subseteq_{\text{sim}} \mathcal{F} \Rightarrow \mathcal{G} \parallel \mathcal{E} \subseteq_{\text{sim}} \mathcal{F} \parallel \mathcal{G} \quad (11) \]

Proof. The constructions \( \mathcal{E} \parallel \mathcal{F} \) and \( \mathcal{F} \parallel \mathcal{E} \) result in the same event structure and similarly for the associativity.

The Unfold Equation (5) is clear because the left and right hand side event structures are exactly the same up to renaming of events.

Implication (6) follows by considering the function \( \text{id}_G \cup f : \mathcal{T}(G + \mathcal{E}) \to \mathcal{T}(G + \mathcal{F}) \). It is indeed a function because the sets of events \( G \) and \( E \) (resp. \( F \)) are disjoint. The property of a t-simulation follows directly because the set of traces \( \mathcal{T}(G + \mathcal{E}) \) is the disjoint union \( \mathcal{T}(G) \cup \mathcal{T}(\mathcal{E}) \) (similarly for \( G + F \)).

For case of sequential composition (10), let \( f \) be a t-simulation from \( \mathcal{E} \) to \( \mathcal{F} \). It is clear that the function \( g : \mathcal{T}(G \cdot \mathcal{E}) \to \mathcal{T}(G \cdot \mathcal{F}) \), such that \( g(\alpha) = \alpha |_{\mathcal{G}} f(\alpha |_{\mathcal{E}}) \) is a t-simulation.

For the Implication (11), let \( f : \mathcal{T}(\mathcal{E}) \to \mathcal{T}(\mathcal{F}) \) be a t-simulation. Let us construct a t-simulation \( g : \mathcal{T}(\mathcal{E} \parallel \mathcal{G}) \to \mathcal{T}(\mathcal{F} \parallel \mathcal{G}) \) inductively. We set \( g(\emptyset) = \emptyset \).

Let \( \alpha \in \mathcal{T}(\mathcal{E} \parallel \mathcal{G}) \) and \( e \in E \cup G \) such that \( \alpha e \) is a trace of \( \mathcal{E} \parallel \mathcal{G} \). We write \( \alpha |_{\mathcal{E}} \) the restriction of \( \alpha \) to the events occurring in \( \mathcal{E} \). The inductive definition of \( g \) is:

\[
g(\alpha e) = \begin{cases} g(\alpha)e & \text{if } e \in G, \\
g(\alpha) & \text{if } e \in E \text{ and } f(\alpha |_{\mathcal{E}} e) = f(\alpha |_{\mathcal{E}}), \\
g(\alpha)e' & \text{if } e \in E \text{ and } f(\alpha |_{\mathcal{E}} e) = f(\alpha |_{\mathcal{E}})e' \\
\end{cases}
\]

Since the set of events of \( \mathcal{E} \) and \( \mathcal{G} \) are disjoint, the cases in the above definition of \( g \) are disjoint. That is, \( g \) is indeed a function and it satisfies the second property of a t-simulation.

The last property is clear because if \( \alpha e \) is maximal in \( \mathcal{T}(\mathcal{E} \parallel \mathcal{G}) \), then either \( \alpha |_{\mathcal{E}} \) is maximal in \( \mathcal{E} \) and \( \alpha |_{\mathcal{G}} e \) is maximal in \( \mathcal{T}(\mathcal{G}) \), or \( \alpha |_{\mathcal{E}} e \) is maximal in \( \mathcal{T}(\mathcal{E}) \) and \( \alpha |_{\mathcal{G}} \) is maximal in \( \mathcal{T}(\mathcal{G}) \). In both cases, \( g(\alpha e) = g(\alpha)e' \) for some \( e' \in E \cup G \) and \( g(\alpha e) \) is weakly maximal in \( \mathcal{T}(\mathcal{F} \parallel \mathcal{G}) \).

\[ \square \]
We now state the main result of this section, which is the backbone of our probabilistic rely-guarantee calculus.

**Theorem 4.13.** Let $\mathcal{E}$ and $\mathcal{F}$ be feasible and terminating ipBESs, if $\mathcal{E} \sqsubseteq_{\text{sim}} \mathcal{F}$ then $\mathcal{E} \sqsubseteq \mathcal{F}$.

**Proof.** Let $f$ be a t-simulation from $\mathcal{E}$ to $\mathcal{F}$, $s \in \Omega$ be the initial state, $\sigma \in \text{Sched}_1(\mathcal{E})$ and $\varphi$ is the complete run of $\sigma$ on $\mathcal{E}$ from $s$. We have to generate a scheduler $\tau \in \text{Sched}_1(\mathcal{F})$ such that the measures $\sigma_s(\mathcal{E})$ and $\tau_s(\mathcal{F})$ are equal i.e. they produce the same value for every state $u \in \Omega$.

For every $\beta \in \mathcal{T}(\mathcal{F})$, we define $f^{-1}_{\min}(\beta)$ to be the set of minimal traces in $f^{-1}(\beta)$, that is,

$$f^{-1}_{\min}(\beta) = \{ \alpha \mid \forall e \in E : \alpha = \alpha' e \in f^{-1}(\beta) \Rightarrow \alpha' \notin f^{-1}(\beta) \}.$$ 

We now construct the scheduler $\tau$. Let $\beta \in \mathcal{T}(\mathcal{F})$. We consider two cases:

- If $f^{-1}(\beta) = \emptyset$ then we set $\tau(\beta)(e, t) = 0 \in \mathbb{D} \leq \Omega$ (the subdistribution that produces 0 on every state), except for some particular maximal traces that are handled in ($\dagger$) below.

- Otherwise, given a state $t \in \Omega$, we define a normalisation factor

$$C_{\beta, t} = \sum_{\alpha \in f^{-1}_{\min}(\beta)} \varphi(\alpha)(t),$$

and we set

$$\tau(\beta)(e, t) = \frac{1}{C_{\beta, t}} \left( \sum_{\alpha \in f^{-1}_{\min}(\beta)} \varphi(\alpha)(t) \sum_{\alpha_{e_1} \cdots \alpha_{e_k} \in f^{-1}_{\min}(\beta)} \prod_{i=1}^{k} w_{i-1}(e_i, t) \mu_k \right)$$

where $w_{i-1}(e_i, t)$ is the quantity such that $\sigma(\alpha e_1 \cdots e_{i-1})(e_i, t) = w_{i-1}(e_i) \mu$, and $\mu \in \lambda(e_i)$ (if $\lambda(e_i)(t)$ is empty then $w_{i-1}(e_i, t) = 0$). The distribution $\mu_k$ is chosen by $\sigma$ from $\lambda(e_k)(t)$, when scheduling $e_k$.

Firstly, we show that $\tau$ is indeed a scheduler on $\mathcal{F}$. The Property ($\ddagger$) of Definition 4.2 is clear. Let us show the other properties. Let $\beta e \in \mathcal{T}(\mathcal{E})$ and let $W : E \times \Omega \to \mathbb{R}$ be the weight function such that

$$W(e, t) = \frac{1}{C_{\beta, t}} \sum_{\alpha \in f^{-1}_{\min}(\beta)} \varphi(\alpha)(t) \sum_{\alpha_{e_1} \cdots \alpha_{e_k} \in f^{-1}_{\min}(\beta)} \prod_{i=1}^{k} w_{i-1}(e_i, t)$$

Indeed, $\mu = \frac{\tau(\beta)(e, t)}{W(e, t)} \in \lambda_e(\tau)(t)$ because $\lambda_e(\tau)(t)$ is convex and for each $\alpha e_1 \cdots e_k \in f^{-1}_{\min}(\beta e)$, $\mu_k \in \lambda(e_k)(t) \subseteq \lambda(e)(t)$. Hence $\tau(\beta)(e) = W_e f_e$ and $\tau$ satisfies the

---

4 Notice if $C_{\beta, t} = 0$ for some $t \in \Omega$ then $\varphi(\alpha)(t) = 0$ for every $\alpha \in f^{-1}_{\min}(\beta)$. In other words, none of these $\alpha$ will be scheduled at all. Hence, $\beta$ need not be scheduled either.

5 The case $W(e, t) = 0$ can be adapted easily because the numerator in the definition of $\tau(\beta)(e)$ is also 0. For instance, we can assume that $\frac{\tau}{\varphi} = 1$. 

---
we have $V(t) = w_0(e_1',t) + w_0(e_1,t)w_1(e_1',t)w_0(e_1,t)w_1(e_2,t) = 1$ because $w_1(e_1',t) + w_1(e_2,t) = 1$ and $w_0(e_1',t) + w_0(e_1,t) = 1$ (Definition 4.2 Property 3).

Figure 1: An example showing that $V(t) = 1$

Property 2 of Definition 4.2 As for Property 3, let $s \in \Omega$ and let us compute the quantity

$$V(t) = \sum_{(e,t) \in \text{dom}(\tau(\beta))} W(e,t),$$

for a fixed $t \in \Omega$. Let us write $\text{dom}(\beta) = \{ e \mid \beta e \in T(F) \}$.

$$V(t) = \sum_{(e,t) \in \text{dom}(\tau(\beta))} \frac{1}{C_{\beta,t}} \sum_{\alpha \in f_{\min}^{-1}(\beta)} \varphi(\alpha)(t) \sum_{\alpha e_1 \cdots e_k \in f_{\min}^{-1}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,t)$$

$$= \frac{1}{C_{\beta,t}} \sum_{\alpha \in f_{\min}^{-1}(\beta)} \varphi(\alpha)(t) \sum_{(e,t) \in \text{dom}(\tau(\beta))} \sum_{\alpha e_1 \cdots e_k \in f_{\min}^{-1}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,t)$$

$$= \frac{1}{C_{\beta,t}} \sum_{\alpha \in f_{\min}^{-1}(\beta)} \varphi(\alpha)(t) \sum_{\alpha e_1 \cdots e_k \in \cup e \in \text{dom}(\beta) f_{\min}^{-1}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,t)$$

From the second to the third expression, the two rightmost sums were merged into a single one because $f_{\min}^{-1}(\beta e) \cap f_{\min}^{-1}(\beta e') = \emptyset$ ($f$ is a function). It follows from Property 3, applied on the weight $w_{i-1}(e_i,t)$ of $\sigma$, that

$$\sum_{\alpha e_1 \cdots e_k \in \cup e \in \text{dom}(\beta) f_{\min}^{-1}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i,s) = 1$$

and hence $V = 1$ (c.f. Figure 1 for a concrete example). The last Property 4 of Definition 4.2 is clear because if $\lambda(e)(t) = \emptyset$, then the coefficient of $\sigma(\alpha e_1 \cdots e_{k-1})(e_k,t)$ is 0 because $\lambda(e_k)(t) = \emptyset$. Hence, the product is also 0.

Secondly, let $\psi$ be the complete run of $F$ with respect to $\tau$. We now show by induction on $\beta$ that

$$\psi(\beta) = \sum_{\alpha \in f_{\min}^{-1}(\beta)} \varphi(\alpha) = C_{\beta,t}, \quad (12)$$
where the empty sum evaluates to the identically zero distribution. The base case is clear because \( \psi(\emptyset) = \delta_s = \phi(\emptyset) \) where \( s \) is the initial state. Let us assume the above identity for \( \beta \in T(\mathcal{F}) \) and let \( e \in F \) such that \( \beta e = T(\mathcal{E}) \) and \( f_{\min}^{-1}(\beta e) \neq \emptyset \). By definition of \( \psi \), if \( u \in \Omega \), we have:

\[
\psi(\beta e)(u) = \sum_{t \in T} \sum_{\alpha \in f_{\min}^{-1}(\beta e)} \varphi(\alpha)(t) \sum_{\alpha_1 \cdots e_k \in f_{\min}^{-1}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i, t) \mu_k(u) \psi(\beta)(t)
\]

\[
= \sum_{t \in \Omega} \sum_{\alpha \in f_{\min}^{-1}(\beta e)} \sum_{\alpha_1 \cdots e_k \in f_{\min}^{-1}(\beta e)} \prod_{i=1}^{k} w_{i-1}(e_i, t) \mu_k(u) \varphi(\alpha)(t)
\]

\[
= \sum_{\alpha \in f_{\min}^{-1}(\beta e)} \sum_{\alpha_1 \cdots e_k \in f_{\min}^{-1}(\beta e)} \sum_{t \in \Omega} \prod_{i=1}^{k} w_{i-1}(e_i, t) \mu_k(u) \varphi(\alpha)(t)
\]

\[
= \sum_{\alpha \in f_{\min}^{-1}(\beta e)} \sum_{\alpha_1 \cdots e_k \in f_{\min}^{-1}(\beta e)} w_0(e_1, t') \delta(t')(t) \prod_{i=2}^{k} w_{i-1}(e_i, t) \mu_k(u) \varphi(\alpha_1)(t')
\]

\[
= \sum_{\alpha \in f_{\min}^{-1}(\beta e)} \sum_{\alpha_1 \cdots e_k \in f_{\min}^{-1}(\beta e)} w_1(e_2, t') \delta(t')(t) \prod_{i=3}^{k} w_{i-1}(e_i, t) \mu_k(u) \varphi(\alpha_2)(t')
\]

\[
= \cdots
\]

By continuing the above reasoning for all \( e_i \) (induction), \( i \leq k - 1 \), we obtain

\[
\psi(\beta e)(u) = \sum_{\alpha \in f_{\min}^{-1}(\beta e)} \sum_{\alpha_1 \cdots e_k \in f_{\min}^{-1}(\beta e)} w_{k-1}(e_k, t) \mu_k(u) \varphi(\alpha e_1 \cdots e_{k-1})(t)
\]

\[
= \sum_{\alpha \in f_{\min}^{-1}(\beta e)} \sum_{\alpha_1 \cdots e_k \in f_{\min}^{-1}(\beta e)} \varphi(\alpha e_1 \cdots e_k)(u)
\]

Hence,

\[
\psi(\beta e)(u) = \sum_{\alpha' \in f_{\min}^{-1}(\beta e)} \varphi(\alpha')(u).
\]
such that $\beta = \beta' e_1 \cdots e_n \in T_{\text{max}}(F)$ and $\delta \in \Pi(e_i)$. We extend $\tau$ such that
\[
\tau(\beta' e_1 \cdots e_i (e_{i+1},t)) = \delta_1.
\]
This implies that $\psi(\beta)(t) = \psi(\beta')(t)$. The other case is that $\beta$ is maximal and belongs to the image of $f$. In both cases, we have
\[
\psi(\beta)(t) = \sum_{\alpha \in A_\beta} \phi(\alpha)(t),
\]
where $A_\beta = f^{-1}_{\text{min}}(\beta)$ if $\beta$ is in the image of $f$, or $A_\beta = f^{-1}_{\text{min}}(\beta')$ if there is such a $\beta'$ as above, otherwise, $A_\beta = \emptyset$. Thus, $A_\beta$ contains maximal traces only (if it is not empty). Since, $f$ is a total function, the set $\{A_\beta \mid \beta \in T_{\text{max}}(F)\}$ is a partition of $T_{\text{max}}(E)$ and we have
\[
\sum_{\beta \in T_{\text{max}}(E)} \psi(\beta)(t) = \sum_{\beta \in T_{\text{max}}(E)} \sum_{\alpha \in A_\beta} \phi(\alpha)(t) = \sum_{\alpha \in T_{\text{max}}(E)} \phi(\alpha)(t),
\]
i.e., we obtain $\tau_s(F) = \sigma_s(E)$.

**Example 4.14.** Let us reconsider the $t$-simulation of Example 4.10. By definition, the unique scheduler $\sigma$ on $b + \neg b \cdot r$ is characterised by the weight function $w$ such that, $w(e_r,t) = 1$ if $\neg b(t)$ is not empty, and $0$ otherwise. From the illustrated $t$-simulation $f$, the corresponding scheduler $\tau$ on $1 + r$, constructed (in the proof of Theorem 4.13) from $\sigma$ using the $t$-simulation $f$ satisfies $\tau(\emptyset)(e_r,t) = w(e_{\neg b},t)\mu$, where $\mu \in r(t)$ is the distribution chosen by $\sigma$. Similarly, $\tau(\emptyset)(e_\delta, t) = w(e_\delta, t)\delta$.

We now show that the binary Kleene star is preserved by the semantics map.

**Proposition 4.15.** For every non-zero, feasible and terminating event structure $E$ and $F$, we have $[E * F] = [E] * [F]$.

**Proof.** For the binary Kleene product, since $[E] * [F]$ is the least fixed point of $f(X) = [F] + [E]$, $X$ in $\Pi_1(F)$ and $E * F$ satisfies
\[
F + E \cdot (E * F) =_{\text{sim}} E * F
\]
by construction of the sequences of bundle event structures defining $E * F$. Therefore, Theorem 4.13 and Proposition 4.8 imply that $[E] * [F] \subseteq \Pi E * F$.

Conversely, let $\mu \in [E * F](s)$ for some initial state $s \in \Omega$. As in the case of Proposition 4.8, we assume that $\mu$ is computed from a scheduler $\sigma$ on $E * F$. We construct a sequence of schedulers $\sigma_n$ that “converges” to $\sigma$ as follows. We set $\sigma_0$ to be any element of $\text{Sched}_1(F)$, $\sigma_1(\alpha) = \sigma(\alpha)$ if $\alpha$ is a trace of $F$ or $E$, otherwise, we set $\sigma_1(\alpha') = \sigma_0(\alpha'')$ where $\alpha' \in T_{\text{max}}(E)$ (notice that $\sigma_0$ is applied to a different copy of $F$ but this is not important as event names can be abstracted.). Inductively, we define
\[
\sigma_n(\alpha) = \begin{cases}  
\sigma(\alpha) & \text{if } \alpha \in T(F + E \cdot (\ldots E \cdot (F + E))) \text{,} \\
\sigma_0(\alpha|_F) & \text{otherwise}
\end{cases}
\]
\footnote{Notice that the least fixed point is in $\Pi_1 \Omega$ but not $\Pi_2 \Omega$. The reason is that $[E]$ and $[F]$ are elements of $\Pi_2 \Omega$ because of feasibility and termination.}
Again, \(\sigma_0\) is applied to the \(n + 1\)th copy of \(\mathcal{F}\). Indeed, we have

\[
\sigma_n \in \text{Sched}_1(\mathcal{F} + \mathcal{E} \cdot (\cdots \mathcal{E} \cdot (\mathcal{F} + \mathcal{E} \cdot \mathcal{F})))
\]

by construction. On the one hand, the sequence of distributions \(\sigma_n(\mathcal{E})\) forms a subset of \([\mathcal{E}] * [\mathcal{F}](s)\). On the other hand, let \(u \in \Omega\) and let us denote

\[
T_{\leq n} = T(\mathcal{F} + \mathcal{E} \cdot (\cdots \mathcal{E} \cdot (\mathcal{F} + \mathcal{E} \cdot \mathcal{F}))).
\]

If we denote by \(\varphi_n\) the complete run of \(\sigma_n\) on \(\mathcal{E} \cdot \mathcal{F}\), then we have

\[
|\sigma_n(\mathcal{E})(u) - \sigma_n(\mathcal{E})(u)| = \left| \sum_{\alpha \in \mathcal{T}_{\max}(\mathcal{E} \cdot \mathcal{F})} \varphi(\alpha)(u) - \sum_{\alpha \in \mathcal{T}_n \cap \mathcal{T}_{\max}(\mathcal{E} \cdot \mathcal{F})} \varphi_n(\alpha)(u) \right|
\]

\[
= \left| \sum_{\alpha \in \mathcal{T}_{\max}(\mathcal{E} \cdot \mathcal{F}) \setminus T_{\leq n-1}} (\varphi(\alpha)(u) - \varphi_n(\alpha)(u)) \right|
\]

\[
\leq \sum_{\alpha \in \mathcal{T}_{\max}(\mathcal{E} \cdot \mathcal{F}) \setminus T_{\leq n-1}} |\varphi(\alpha)(u) - \varphi_n(\alpha)(u)|
\]

The set \(\mathcal{T}_{\max}(\mathcal{E} \cdot \mathcal{F}) \setminus T_{\leq n-1}\) shrinks, when \(n\) increases, because every finite trace of \(\mathcal{E} \cdot \mathcal{F}\) belongs to some set \(T_{\leq k}\). Therefore, the last sum above is decreasing to 0. Hence, since \(\Omega\) is a finite set, the sequence \(\sigma_n(\mathcal{E} \cdot \mathcal{F})\) converges (pointwise) to \(\sigma(\mathcal{E} \cdot \mathcal{F})\) in \(\mathcal{D}\Omega\). Since \([\mathcal{E}] * [\mathcal{F}](s)\) is topologically closed, we deduce that \(\sigma(\mathcal{E}) \in [\mathcal{E}] * [\mathcal{F}](s)\). Therefore, \([\mathcal{E} \cdot \mathcal{F}] \sqsubseteq \mathcal{E} \cdot \mathcal{F}\). \(\square\)

**Proposition 4.16.** Let \(r, r' \in \mathbb{H}_1\Omega\) be two atomic programs and let \(\mathcal{E}, \mathcal{F}\) be two bundle event structures with internal probability, then

\[
r^* || r^* \sqsubseteq_{\text{sim}} r^*
\]

\[
r^* || r' \sqsubseteq_{\text{sim}} r * (r' \cdot r^*)
\]

\[
r^* || (b \cdot \mathcal{E} + c \cdot \mathcal{F}) \sqsubseteq_{\text{sim}} r * (b \cdot (r^* || \mathcal{E}) + c \cdot (r^* || \mathcal{E}))
\]

\[
r^* || (r' \cdot \mathcal{E}) \sqsubseteq_{\text{sim}} r * (r' \cdot (r^* || \mathcal{E}))
\]

where \(r^* = r * 1\).

**Proof.** Let us denote by \(e_1\) and \(e_2\) (resp. \(e\)) the events that are labelled by \(\delta\) in the event structure associated to \(r^* || r^*\) (resp. \(r^*\)). Given a trace \(\alpha\) of \(r^* || r^*\) that does not contain any of the \(e_i\)s, we denote by \(\alpha'\) unique trace corresponding to \(\alpha\) in \(r^*\) (i.e. with the same number of events labelled by \(r\)).

A t-simulation from \(r^* || r^*\) to \(r^*\) is obtained by considering a function \(f\) such that

\[
f(\alpha) = \begin{cases} 
(\alpha \setminus \{e_1, e_2\})' & \text{if } e_1, e_2 \notin \alpha \\
(\alpha \setminus \{e_1, e_2\})' e & \text{if } e_1, e_2 \in \alpha
\end{cases}
\]
The "obvious" arrows, such as an arrow from $r'\delta$ to $r'\delta$, have been left out to keep the picture clear.

Figure 2: The t-simulation from $r^*\parallel r'$ to $r \ast (r' \cdot r^*)$.

The $t$-simulation (14) is constructed as follows. Let us abstract the event names, i.e., $r^k$ would be a trace where each $r$ is the label of a unique event. Every trace of $r^*\parallel r'$ is a prefix of $r^m r' r^n \delta$ or $r^m \delta r'$, for some $m, n \geq 0$. Every prefix of either trace corresponds to a unique trace of $r \ast (r' \cdot r^*)$. For instance, the maximal trace $r^m \delta r'$ is associated to the weakly maximal trace $r^m r' \delta$ of $r \ast (r' \cdot r^*)$. Figure 2 shows an explicit construction of the t-simulation.

The Simulation (15) is similar. Every trace of $r^*\parallel (r' \cdot E + c \cdot F)$ is a prefix of $r^m b \alpha$ or $r^m \delta r' \beta$ or $r^m \delta c \zeta$, where $\alpha \in T(r^*\parallel E)$, $\beta \in T(r^*\parallel F)$, $\gamma \in T(E)$, $\zeta \in F$ and $n \geq 0$. Again, prefixes of the first two traces correspond to a unique trace of $r \ast (b \cdot (r^*\parallel E) + c \cdot (r^*\parallel F))$. The maximal trace $r^m \delta b \gamma$ is again mapped to the weakly maximal trace $r^m b \gamma$. Similarly for the fourth case. This indeed results in a t-simulation.

The Simulation (16) is constructed as follows. Every trace of $r^*\parallel (r' \cdot E)$ is a prefix of $r^m r' \alpha$ or $r^m \delta r' \beta$ for some trace $\alpha \in T(r^*\parallel E)$ and $\beta \in T(E)$. We continue as in the previous case.

Proposition 4.16 is used mainly to interleave the right operand $r^*$ systematically with the internal structure of $E$, while preserving the simulation order. More precisely, these equations are applied to generate algebraic proofs for the reduction of one expression into another where the occurrence of $\parallel$ is pushed deeper into the sub-expressions (and possibly removed).
Probabilistic rely-guarantee conditions

Our first task towards the extension of rely-guarantee to probabilistic systems is to provide a suitable definition of a rely condition that contains sufficient quantitative information about the environment and the components of a system.

From a relational point of view, as in Jones’ thesis [1], a guarantee condition expresses a constraint between a state and its successor by running the relation as a nondeterministic program. Therefore, it is important to know whether some action is executed atomically or whether is split into smaller components. For instance, when run in the same environment, a probabilistic choice between $x := x + 1$ and $x := x - 1$ produced from an if...then...else clause may behave differently from an atomic probabilistic assignment that assigns $x + 1$ and $x - 1$ to $x$ with the exact same probability.

Without probability, a common example of a guarantee condition for a given program is the reflexive transitive closure with respect to $(\parallel)$ of the union of all atomic actions in that program [23] which completely captures all possible “effects” of the program. Such a closure property plays a crucial role in the algebraic proof of Rule 2 and is achieved through Proposition 4.16. This construction was introduced by Jones [1] and later refined by others [11, 19, 23].

Non-probabilistic rely-guarantee conditions usually take the form $\rho^*$ for some binary relation $\rho$, defined on the state space of the studied program. The transitive closure of $\rho$ with respect to the relational composition $(\cdot)$ is usually a desirable property. To obtain a probabilistic guarantee condition from a total relation $\rho \subseteq \Omega \times \Omega$, we construct a probabilistic program $r \in H_1[\Omega]$ such that

$$r(s) = \{ \mu \in D\Omega \mid \mu(\{s' \mid (s, s') \not\in \rho\}) = 0 \},$$

where $D\Omega$ is the set of probability distributions over $\Omega$. Equivalently, $r$ is the convex closure of $\rho$. The following proposition then follows from that construction.

Proposition 5.1. If a relation $\rho \subseteq \Omega \times \Omega$ is transitive, then the convex closure $r$ of $\rho$ satisfies $r \cdot (r + \delta) \sqsubseteq H r$.

Proof. Let $\rho$ be a transitive relation, $r$ its associated probabilistic program, $s \in \Omega$ a state and $\mu \in [r \cdot (r + \delta)](s)$. We need to show that $\mu \in r(s)$. By definition of the sequential composition $(\cdot)$ (Equation 5), there exists $\nu \in r(s)$ and a deterministic program $f \sqsubseteq H (1 + r)$ such that $\mu = f \ast \nu$. Let $u \in \Omega$ such that $(s, u) \not\in \rho$, we are going to show that $\mu(u) = 0$. We have:

$$\mu(u) = \sum_{t \in \Omega} f(t)(s)\nu(t) = \sum_{t \in \Omega \land (s, t) \in \rho} f(t)(u)\nu(t) = \sum_{t \in \Omega \land (s, t) \in \rho \land (t, u) \in \rho} f(t)(u)\nu(t).$$

The second equality follows from $\nu(t) = 0$ for every $(s, t) \not\in \rho$. Similarly, the last equality follows from $f(t)(u) = 0$ for $(t, u) \not\in \rho$. The last expression reduces to $\sum_{t \in \Omega \land (s, u) \not\in \rho} f(t)(u)\nu(t)$, by transitivity of $\rho$, which is an empty sum because $(s, u) \not\in \rho$. Therefore, $\mu(u) = 0$ for every $(u, s) \not\in \rho$, which is equivalent to $\mu \in r(s)$. □
The convex closure of a relation \( \rho \), given in Proposition 5.1, sometimes provides a very general rely condition that is too weak to be useful in the probabilistic case. In practice, a probabilistic assignment is considered atomic and the correctness of many protocols is based on that crucial assumption. That is, the random choice and the writing of the chosen value into \( x \) is assumed to happen instantaneously and no other program can modify \( x \) during and in-between these two operations. Thus, probabilistic rely and guarantee conditions need to capture the probabilistic information in such an assignment.

**Example 5.2.** Let us write \( x := \text{uniform}(0, x) \) for the program that assigns a random integer between two integers 0 and \( x \) to the variable \( x \). A probabilistic guarantee condition for that assignment is obtained from the probabilistic program \( r \) such that

\[
    r(x) = \left\{ \mu \mid \mu(\{0, n\}) \geq \frac{1}{n + 1}\right\}
\]

(17)

The condition \( r \) specifies the convex set of all probabilistic deterministic programs whose atomic actions establish a state in \( \{0, n\} \) with probability at least \( \frac{1}{n+1} \). Notice that the exact probability for the assignment to yield a state in \( \{0, n\} \) is \( \frac{1}{x+1} \). The condition \( r \) is transitive and “completely” probabilistic.

In practice, constructing a useful transitive probabilistic rely-guarantee condition is difficult, but the standard technique is still valid. That is, the strongest guarantee condition of a given program is the nondeterministic choice of all atomic actions found in that program.

**Definition 5.3.** A probabilistic rely or guarantee condition \( R \) is a probabilistic concurrent program such that

\[
    R \parallel R \sqsubseteq_{\text{sim}} R
\]

In particular, the concurrent program \( r^* = r \ast 1 \) is a rely condition because

\[
    r^* \parallel r^* \sqsubseteq_{\text{sim}} r^*
\]

(18)

holds in the event structure model (Proposition 4.16 Equation (13)). This illustrates the idea that a rely condition specifies an environment that can stutter or execute a sequence of actions that are bounded by \( r \).

### 6 Probabilistic rely-guarantee calculus

In this section, we develop the rely-guarantee rules governing programs involving probability and concurrency. An example is given by Rule 2 which allows us to compositionally check the safety properties of the subsystems and infer the correctness of the whole system. We provide a probabilistic version of that rule.

In the previous sections, we have developed the mathematical foundations that are needed for our interpretation of Hoare triple and guarantee relation,
namely, the sequential refinement \(\sqsubseteq\) and simulation-based order \(\sqsubseteq_{\text{sim}}\). Following [23], we only adapt the orders in the algebraic interpretation of rely-guarantee quintuples (Equation \(\{1\}\)). That is, a probabilistic rely-guarantee quintuple \(\{ P \text{ } R \text{ } E \text{ } G \text{ } Q \}\) is valid iff
\[
P \cdot (R \| E) \sqsubseteq Q \quad \text{and} \quad E \sqsubseteq_{\text{sim}} G
\]
hold, where \(P, E, Q\) are probabilistic concurrent programs and \(R\) and \(G\) are rely-guarantee conditions. The first part is seen as a probabilistic instance of the contraction of [7] which specifies the functional behaviour of \(R \| E\) under a precondition \(P\). The second part uses the simulation order which is compositional and is very sensitive to the structural properties of the program.

The terms \(R\) and \(G\) specify how the component \(E\) interacts with its environment. As we have discussed in the previous section, rely and guarantee conditions are obtained by taking \(r^* = r \ast \delta\) for some atomic probabilistic program \(r\). Therefore, \(E \sqsubseteq_{\text{sim}} r^*\) implies that all actions carried by events in \(E\) are either stuttering or satisfying the specification \(r\). This corresponds to the standard approach of Jones [19, 1].

The following rules are probabilistic extensions of the related rely-guarantee rules developed in [8, 19]. These rules are sound with respect to the event structure semantics of Section [4].

**Atomic action**: The rely-guarantee rule an atomic statement \(r'\) is provided by the equation
\[
r^* \| r' \sqsubseteq_{\text{sim}} r \ast (r' \cdot r^*)
\]
where \(r\) is the rely condition. This equation shows that a (background) program satisfying the rely condition \(r\) will not interfere with the low level operations involved in the atomic execution of \(r'\). The programs will be interleaved.

**Conditional statement**: The rely-guarantee rule for conditional statement is provided by the equation
\[
r^* \| (b \cdot E + c \cdot F) \sqsubseteq_{\text{sim}} r \ast (b \cdot (r^* \| E) + c \cdot (r^* \| F)).
\]
This equation shows how a rely conditions \(r^*\) distributes through branching structures. The tests \(b\) and \(c\) are assumed to be atomic and their disjunction is always true (this is necessary for feasibility). This assumption may be too strong in general because \(b\) may involve the reading of some large data that is too expensive to be performed atomically. However, we may assume that such a reading is done before the guard \(b\) is checked and the non-atomic evaluation of the variables involved in \(b\) may be assigned to some auxiliary variable that is then checked atomically by \(b\).

**Prefixing**: the sequential rely-guarantee rule for a probabilistic program expressed using prefixing. We have
\[
r^* \| (r' \cdot E) \sqsubseteq_{\text{sim}} r \ast (r' \cdot (r^* \| E)).
\]
It generalises Rule [19] and tells us that a rely condition \(r^*\) distributes through the prefixing operation. In other words, the program \(r'\) and \(E\) should tolerate
the same rely condition in order to prove any meaningful property of $r \cdot \mathcal{E}$. This is mainly a consequence of our interpretation of $\parallel$ where no synchronisation is assumed.

**Concurrent execution:** In Rule 2, the concurrent composition $\mathcal{E} \parallel \mathcal{E}'$ requires an environment that satisfies $R \cap R'$ to establish the postcondition $Q \cap Q'$. However, such an intersection is not readily accessible at the structural level of event structures. Therefore, the most general probabilistic extension of Rule 2 which applies to our algebraic setting is:

$$\frac{(P \parallel R)\mathcal{E}\{G \parallel Q\}}{(P \parallel R')\mathcal{E}'\{G' \parallel Q'\}} \quad \quad (22)$$

where $R'$ is a rely condition such that $R'' \subseteq R$ and $R'' \subseteq R'$. The proof of this rule is exactly the same as in [8][20]. In fact, we have $R'' \subseteq R$, $\mathcal{E}' \subseteq R$, $R\parallel R \subseteq R$, therefore Equation (10) and Equational Implication (11) imply

$$R'' \parallel (\mathcal{E}' \parallel \mathcal{E}) \subseteq (R \parallel \mathcal{E}) \subseteq R \parallel \mathcal{E},$$

and we obtain $P \cdot R'' \parallel (\mathcal{E}' \parallel \mathcal{E}) \subseteq P \cdot (R \parallel \mathcal{E})$ by Equation (10). It follows from Theorem 4.13 that $P \cdot R'' \parallel (\mathcal{E}' \parallel \mathcal{E}) \subseteq Q$.

The conclusion does not contain any occurrence of $Q'$, but by symmetry, it is also valid if $Q$ is substituted with $Q'$. The combined rely condition $R''$ is constructed such that it is below $R$ and $R'$. Indeed, if $R, R'$ have a greatest lower bound with respect to $\subseteq$, then $R''$ can be that lower bound so that the strengthening of the rely is not too strong.

The above rule can be specialised by considering rely-guarantee conditions of the form $r^*$, where $r$ is an atomic probabilistic program. The following rule is expressed in exactly the same way as in the standard case [8]. This is due to the fact that probabilities are internal.

**Proposition 6.1.** The following rule is valid in BES

$$\frac{(P \parallel r_1)\mathcal{E}_1\{g_1^* \parallel Q_1\}}{(P \parallel r_2)\mathcal{E}_2\{g_2^* \parallel Q_2\}} \quad \quad (23)$$

where $r, r', g, g' \in \mathbb{H}_1\Omega$ and $g + g'$ is the nondeterministic choice on $\mathbb{H}_1\Omega$.

**Proof.** This follows from substituting $R$ and $G$ by respectively $r^*$ and $g^*$ in Rule 22. Moreover $g^* \parallel (g + g')^*$ holds because $(g + g')^* \subseteq (g + g')^*$ (Equation 18).

Recall that the nondeterministic choice of $\mathbb{H}_1\Omega$ is obtained by the pointwise union followed by the necessary closure properties for the elements of $\mathbb{H}_1\Omega$. The intersection $r \cap r'$ is obtained by pointwise intersection.

**Iteration:** A while program is modelled by using the binary Kleene star. The idea is to unfold the whole structure of the loop as far as necessary. The conditional and prefix (sequential) cases can then be applied on the unfolded structure to distribute the rely condition. That is, we write

$$r^* \parallel ((b \cdot \mathcal{E}) \ast c) \subseteq R \ast (c \cdot r^* + b \cdot (r^* \parallel (\mathcal{E} \cdot (b \cdot \mathcal{E} \ast c)))).$$

24
If $E$ is sequential, then $r^*$ can be “interleaved” within the internal structure of $E \cdot (b \cdot E^* c)$ by applying the prefixing and conditional statement rules.

The sequential correctness is achieved by the usual generation of probability distributions, obtained from terminating sequential behaviours, on the “totally” unfolded event structure (assuming that $E$ is sequential). The sequential behaviours are usually obtained by interleaving the rely condition $r^*$ through the internal structure of the unfolded loop. A bounded loop, such as a for loop, should be modelled using a sequence of sequential compositions or prefixing.

### 7 Application: a faulty Eratosthenes sieve

In this section, we show how to use the previously established rely-guarantee rules to verify a probabilistic property of a faulty Eratosthenes sieve.

Let $n \geq 2$ be a natural number and $s_0 = \{2, 3, \ldots, n\}$. We present a variant of the concurrent Eratosthenes sieve that is originally due to Jones [1]. For each integer $i \in [2, \sqrt{n}]$, we consider a program $\text{thd}_i$ that sequentially removes all (strict) multiples of $i$ from the shared set variable $s$ with a fixed probability $p$. More precisely, each thread $\text{thd}_i$ is be implemented as the following program:

```plaintext
for(j = 2 to n/i) 
  u_{i,j} : skip ⊕ p \text{ remove}(i*j from s);
```

where $n/i$ is the integer division of $n$ by $i$. Each $u_{i,j}$ can be seen as a faulty action that removes the product $ij$ from the current value of $s$ with probability $p$. The state space of each atomic deterministic program $u_{i,j}$ is $\Omega = \{s | s \subseteq s_0\}$.

In $H_1 \Omega$, $u_{i,j}$ is defined by $u_{i,j}(s) = (1 - p)\delta_s + p\delta_{s\backslash\{ij\}}$. The whole system is specified by the concurrent execution

$$\text{thd}_2 \parallel \ldots \parallel \text{thd}_{\sqrt{n}} = \parallel_{i=2}^{\sqrt{n}} (u_{i,2} \cdots u_{i,n/i})$$

where, in the sequel, $\sqrt{n}$ is computed without decimals.

Let us denote by $\pi = \{2, 3, \ldots, p\}$ the set of prime numbers less than or equal to $n$. Our goal is to compute a “good” lower bound probability that the final state is $\pi$, after executing the threads $\text{thd}_i$ concurrently, from the initial state $s_0$.

Let us denote by $O_{i,j} = \{s | ij \notin s\} \subseteq \Omega$ and

$$Q_{i,j}(s) = \{\mu \in \mathbb{D}\Omega \mid \mu(O_{i,j}) \geq p \land \mu(\{s' \mid s' \subseteq s\}) = 1\}$$

a specification of a probabilistic program that removes $ij$ from the state $s$ with at least probability $p$ and does not add anything to it. We define $O_i = \cap_{j=2}^{n/i} O_{i,j}$, $Q_i = Q_{i,2} \cdot Q_{i,3} \cdots Q_{i,n/i}$ and $r$ to be the probabilistic program such that $r(s)$ is the convex closure of $\{\delta_{s'} \mid s' \subseteq s\}$.

First, we show that every thread $\text{thd}_i$ guarantees $r^*$. Second, we will show that $\text{thd}_i$ establishes $Q_i$ when run in an environment satisfying $r$, i.e., $r^* \parallel \text{thd}_i \sqsubseteq Q_i$, by mainly using the atomic and prefix rules [19] and [21]. Finally, we will
apply the concurrency rule \[23\] to deduce that the system \(\bigotimes_{i=1}^{\sqrt{\text{thd}_i}}\) establishes all postconditions \(Q_2, Q_3, \ldots, Q_{\sqrt{\text{thd}_i}}\), when run in an environment satisfying \(r\).

**Establishing \(\text{thd}_i \subseteq \text{sim} \ r^*\) and \(r^*\|\text{thd}_i \subseteq Q_i\)**

On the one hand, it is clear that \(u_{i,j} \subseteq \text{H}_1 \ r\), for every \(i, j\), and thus \(\text{thd}_i \subseteq \text{sim} \ r^\star\) follows from the unfold \[25\]. On the other hand, let us show that \(r^*\|\text{thd}_i \subseteq Q_i\).

Multiple applications of the prefix-case give

\[
r^*\|\text{thd}_i \subseteq \text{sim} \ r^* (u_{i,2} \cdot (r^* \cdot (u_{i,3} \cdot (\ldots r^* \cdot (u_{i,n/i} \cdot r^*))))).
\]

Since the right multiplication \(X \mapsto X \cdot r\), by any program \(r' \in \text{H}_1 \Omega\), is the lower adjoint in a Galois connection \[25\], the fixed point fusion theorem \[24\] implies that

\[
r^* (u_{i,2} \cdot (r^* \cdot (u_{i,3} \cdot (\ldots r^* \cdot (u_{i,n/i} \cdot r^*)))) = r^* \cdot u_{i,2} \cdot r^* \cdot u_{i,3} \cdot \ldots \cdot r^* \cdot u_{i,n/i} \cdot r^*
\]

(the equality is in \(\text{H}_1 \Omega\)). Thus,

\[
r^*\|\text{thd}_i \subseteq r^* \cdot u_{i,2} \cdot r^* \cdot u_{i,3} \cdot \ldots \cdot r^* \cdot u_{i,n/i} \cdot r^*
\]

follows from the fact that \(\subseteq\) is weaker than \(\subseteq \text{sim}\) (Theorem 4.13). The right hand side explicitly states the interleaving of the rely condition \(r^\star\) in-between the atomic executions in \(\text{thd}_i\), as in \[19\].

Moreover, since \(r\) is the probabilistic version of a transitive standard relation, Proposition 5.1 implies that \(r \cdot (r + \delta) \subseteq \text{H}_1 \ r\). But \(\text{H}_1 \Omega\) is a probabilistic Kleene algebra \[25\], therefore the right induction law of pKA \[25\] implies that \(r \cdot r^\star \subseteq \text{H}_1 \delta + r\) and therefore \(r^* = \delta + r \cdot r^\star \subseteq \delta + r\). The converse refinement also holds because \(r \subseteq r^\star\), \(\delta \subseteq r^\star\) and \((+)^*\) is idempotent. The reduction of \(r^\star\) to \(\delta + r\) shows the main use of a transitive rely condition. Therefore,

\[
r^*\|\text{thd}_i \subseteq (\delta + r) \cdot u_{i,2} \cdot (\delta + r) \cdot u_{i,3} \cdot \ldots (\delta + r) \cdot u_{i,n/i} \cdot (\delta + r).
\]

where the left hand side is a sequential program (thus Proposition 4.8 enables us to directly use the definition of sequential composition of \(\text{H}_1 \Omega\)). Since \(u_{i,j} \subseteq Q_{i,j}\), it remains to show that \((\delta + r) \cdot Q_{i,j} \cdot (\delta + r) \cdot Q_{i,j} \cdot \ldots (\delta + r) \cdot Q_{i,n/i} \cdot (\delta + r) \subseteq Q_i\).

Let us first show that \(Q_{i,j} \cdot (\delta + r)\) and \((\delta + r) \cdot Q_{i,j}\) are sequentially refined by \(Q_{i,j}\). Let \(s \in \Omega\) and \(\nu \in (Q_{i,j} \cdot (\delta + r))(s)\). By definition of the sequential composition in \(\text{H}_1 \Omega\), there exists a probabilistic deterministic program \(f \subseteq \text{H}_1 \delta + r\) and a distribution \(\mu \in Q_{i,j}(s)\) such that \(\nu(s') = \sum_{t \in \Omega} f(t)(s')\mu(t)\), for every \(s' \in \Omega\). Therefore,

\[
\nu(O_{i,j} \cap \{s' | s' \subseteq s\}) = \sum_{t \in \Omega} f(t)(O_{i,j})\mu(t) = \sum_{t \subseteq s} f(t)(O_{i,j})\mu(t)
\]

where the second equality follows from the fact that for every \(\mu \in Q_{i,j}\), \(\mu(\{t | t \not\subseteq s\}) = 0\). But

\[
\sum_{t \subseteq s} f(t)(O_{i,j})\mu(t) \geq \sum_{i,j \not\subseteq \Omega, t \subseteq s} f(t)(O_{i,j})\mu(t) = \mu(O_{i,j} \cap \{t | t \subseteq s\}) \geq p
\]

26
because \( f(t)(O_{i,j}) = 1 \) for every \( t \) not containing the product \( ij \) and \( \mu(O_{i,j} \cap \{ t \mid t \subseteq s \}) = \mu(O_{i,j}) \) for every \( \mu \in Q_{i,j}(s) \). Consequently, \( Q_{i,j} \cdot (\delta + r) \subseteq Q_{i,j} \). Similarly, we can show that \( (\delta + r) \cdot Q_{i,j} \subseteq Q_{i,j} \) and thus \( r^* \| \text{thd}_i \subseteq Q_i \).

**Establishing the property of \( r^* \| \bigcap_{i=2}^{\sqrt{n}} \text{thd}_i \)**

Applying the rule \([23]\) \( \sqrt{n} - 1 \) times, we obtain, for every \( Q_j \) such that \( 2 \leq j \leq \sqrt{n} \),

\[
r^* \| \bigcap_{i=2}^{\sqrt{n}} \text{thd}_i \subseteq Q_j
\]

**Infering a lower bound for the probability of correctness**

Unfortunately, Rule \([23]\) does not give us an explicit bound in term of probability of correctness. To obtain a lower bound for the probability of removing all composite numbers, let us first study the case of two threads running concurrently. We know from Rule \([23]\) that \( r^* \| \text{thd}_2 \| \text{thd}_3 \subseteq Q_j \), where \( j \in \{2, 3\} \).

Therefore, for every \( \mu \) obtained from a sequential behaviour of \( r^* \| \text{thd}_2 \| \text{thd}_3 \) (generated from a scheduler, Section \([4.1]\)), we have \( \mu(O_2) \geq p^{n/2-1} \) and \( \mu(O_3) \geq p^{n/3-1} \). Therefore, \( \mu(O_1 \cup O_2) + \mu(O_2 \cap O_3) \geq p^{n/2-1} + p^{n/3-1} \) and thus \( \mu(O_2 \cap O_3) \geq p^{n/2-1} + p^{n/3-1} - 1 \).

In the construction of the lower bound above, we have only used the modularity of measure and, therefore, it can be transformed into a more general rely-guarantee rule with explicit probabilities. Let us denote by \([\mathcal{E}]\) the sequential program obtained by considering all possible terminating schedulers on \( \mathcal{E} \), that is, given an initial state \( s_0 \), \([\mathcal{E}](s_0)\) is a set of probability distributions. Given a subset \( O \subseteq \Omega \) and \( p \in \mathbb{R} \), we write \([\mathcal{E}](s_0)(O) \geq p \) if for every \( \mu \in [\mathcal{E}](s_0) \), we have \( \mu(O) \geq p \).

**Proposition 7.1.** For every initial state \( s_0 \) and subsets \( O_1, O_2 \subseteq \Omega \), we have

\[
[r^* \| \bigcap_{i=2}^{\sqrt{n}} \text{thd}_i](s_0)(\bigcap_{i=2}^{\sqrt{n}} O_i) \geq \sum_{i=2}^{\sqrt{n}} p^{n/i-1} - (\sqrt{n} - 2) = f(p, n).
\]

The lower bound \( f(p, n) \) sometimes provides a bad lower-approximation for the probability that the system establishes \( \bigcap_{i=2}^{\sqrt{n}} O_i \), mainly because of the presence of \( r^* \). However, it is clear that \( \lim_{p \to 1} f(p, n) = f(1, n) = 1 \).

In the particular case of \( n = 15 \), \( f(p, 15) = p^9 + p^4 - 1 \). A simple numerical calculation shows that \( f(p, 15) \) gives a positive lower bound when \( p \geq 0.868 \), the exact probability being \( p^{10} + 4p^9(1-p) + 4p^4(1-p)^2 \).

**Refining the lower bound**

We can use other internal properties of the system to obtain a better lower bound. It is clear that \( O_i \) is an invariant for every \( \text{thd}_j \) (for \( j \neq i \)) and that all actions \( u_{i,j} \) (sequentially) commute with each other. Intuitively, we obtain a closer lower bound by noticing that the system is “sequentially better” than considering: \( \text{thd}_2 \) removes all multiples of 2, \( \text{thd}_3 \) removes all multiples of 3.
isolated in the fact that all common multiples of 2 and 3 have been removed by \( \text{thd}_2 \), and so on. Therefore, 
\[
[r^*]|_{i=2}^{\sqrt{n}}\text{thd}_i[(s_0)\cap\cap_{i=2}^{\sqrt{n}}O_i] \geq p^{n/2-1} \left( \frac{p^{n/4-1}}{p^{n/6}} \right) \left( \frac{p^{n/6-1}}{p^{n/10}} \right) \cdots = g(p,n).
\]

In the particular case of \( n = 15 \), \( g(p,15) = p^8 \). A graphical comparison of \( f, g \) and the actual probability is displayed in Figure 3 for this particular value of \( n \).

**Establishing the property of \( \|_{i=2}^{\sqrt{n}}\text{thd}_i \):**

Finally, notice that \( \emptyset \in \cap_{i=2}^{\sqrt{n}}O_i \), which means that \( r^*|_{i=2}^{\sqrt{n}}\text{thd}_i \) can establish \( s = \emptyset \) with a positive probability. This issue is resolved by using a stronger guarantee property such as “\( u_{i,j} \) never removes \( i \)”. Therefore, \( |_{i=2}^{\sqrt{n}}\text{thd}_i \) never removes any prime numbers i.e. any element of \( \cap_{i=2}^{\sqrt{n}}O_i \), that does not contain all the positive prime numbers below \( n \), occurs with probability 0.

### 8 Conclusion

This paper presented an extension of rely-guarantee calculus to account for probabilistic programs running in a shared variable environment. Moreover, the rely-guarantee rules are expressed and developed mostly using the algebraic of the bundle event structure semantics.

In this paper, the specification of a probabilistic concurrent program is expressed with a rely-guarantee quintuple. Each quintuple is algebraically defined through the use of a sequential order \( \sqsubseteq \), which captures all possible sequential behaviors when a suitable definition of the concurrency operation \( \| \) is given, and a simulation order \( \sqsubseteq_{\text{sim}} \), which specifies the level of interference between the specified component and the environment. Various probabilistic rely-guarantee rules have been established and applied on a simple example of faulty concurrent system. We have also shown some rules which provide explicit quantitative properties, such as a lower bound for the probability of correctness. In partic-
ular, a better lower-approximation can be derived if further internal properties of the systems are known.

The framework developed in this paper has its current limitations. Firstly, neither the algebra nor the event structure model support non-terminating probabilistic concurrent programs yet. That is, the rely-guarantee rules of this paper can only be applied in a partial correctness setting. Secondly, the concrete model is restricted to programs with finite state spaces. We will focus, more particularly, on the first limitation in our future work.

References

[1] C. B. Jones. Development Methods for Computer Programs including a Notion of Interference. PhD thesis, Oxford University, June 1981.

[2] Dexter Kozen. Semantics of probabilistic programs. J. Comput. Syst. Sci., 22(3):328–350, 1981.

[3] C. Jones. Probabilistic non-determinism. PhD thesis, Edinburgh, Scotland, UK, 1989. UMI Order No. GAXDX-94930.

[4] Jifeng He, Karen Seidel, and Annabelle McIver. Probabilistic models for the guarded command language. Sci. Comput. Program., 28(2-3):171–192, 1997.

[5] A. K. McIver and C. C. Morgan. Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science). SpringerVerlag, 2004.

[6] R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. Nord. J. Comput., (2):250–273.

[7] A. Armstrong, V. B. F. Gomes, and G. Struth. Algebraic principles for rely-guarantee style concurrency verification tools. In C. B. Jones, P. Pihlajasaari, and J. Sun, editors, FM 2014, pages 78–93, 2014.

[8] C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra and its foundations. J. Log. Algebr. Program., (6):266–296.

[9] A. K. McIver, T. M. Rabehaja, and G. Struth. Probabilistic rely-guarantee calculus. 2014.

[10] Andrzej Tarlecki. A language of specified programs. Sci. Comput. Program., (1):59–81.

[11] J. Dingel. A refinement calculus for shared-variable parallel and distributed programming. Formal Asp. Comput., (2):123–197.

[12] J. W. Coleman and C. B. Jones. A structural proof of the soundness of Rely-guarantee rules. J. Log. Comput., (4):807–841.
[13] Frank S. de Boer, Ulrich Hannemann, and Willem P. de Roever. A compositional proof system for shared variable concurrency. In J. S. Fitzgerald, C. B. Jones, and P. Lucas, editors, FME, LNCS, pages 515–532. Springer.

[14] G. Winskel. Events in Computation. PhD thesis, Edinburgh, Scotland, UK, 1980.

[15] G. Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, Advances in Petri Nets, pages 325–392, 1986.

[16] R. Langerak. Bundle event structures: a non-interleaving semantics for LOTOS. Memoranda informatica.

[17] J. P. Katoen. Quantitative and qualitative extensions of event structures. PhD thesis, University of Twente, 1996.

[18] D. Varacca. Probability, nondeterminism and concurrency: two denotational models for probabilistic computation. BRICS Dissertation Series.

[19] I. J. Hayes, C. B. Jones, and Colvin R. J. Refining rely-guarantee thinking. Technical report, Newcastle University, United Kingdom, 2012.

[20] A. K. McIver, T. M. Rabehaja, and G. Struth. Probabilistic concurrent Kleene algebra. In L. Bortolussi and H. Wiklicky, editors, QAPL, volume 117 of EPTCS, pages 97–115, 2013.

[21] A. K. McIver, T. M. Rabehaja, and G. Struth. An event structure model for probabilistic concurrent Kleene algebra. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, LPAR, LNCS, pages 653–667. Springer.

[22] R. Segala. A compositional trace-based semantics for probabilistic automata. In I. Lee and S. A. Smolka, editors, CONCUR, volume 962 of LNCS, pages 234–248. Springer, 1995.

[23] C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra. In M. Bravetti and G. Zavattaro, editors, CONCUR, LNCS, pages 399–414. Springer.

[24] R. Backhouse. Galois connections and fixed point calculus. In R. Backhouse, R. Crole, and J. Gibbons, editors, ACM/MPC, volume 2297 of LNCS. 2002.

[25] A. K. McIver and T. Weber. Towards automated proof support for probabilistic distributed systems. In G. Sutcliffe and A. Voronkov, editors, LPAR, volume 3835 of LNAI, pages 534–548. Springer, 2005.
A Appendix

This appendix contains the detail of our rely-guarantee rules that incorporate explicit probabilities.

Proposition A.1. For every $s \in P$ and subsets $O_1, O_2 \subseteq \Omega$, we have
\[
\begin{align*}
[r^*_1, \|E_1\|](s)(O_1) & \geq p_1 & [r^*_2, \|E_2\|](s)(O_2) & \geq p_2 & E_1 \sqsubseteq_{\text{sim}} g^* \sqsubseteq_{\text{sim}} r^*_2 & E_2 \sqsubseteq_{\text{sim}} g'^* \sqsubseteq_{\text{sim}} r^*_1 \\
(r_1 \cap r_2)^* \|E_1\| \|E_2\|(s)(O_1 \cap O_2) & \geq p_1 + p_2 - q & \|E_1\| \|E_2\| & \sqsubseteq_{\text{sim}} (g + g')^* 
\end{align*}
\]
where $q \geq \sup_{\mu \in [(r_1 \cap r_2)^* \|E_1\| \|E_2\|]}(s)\mu(O_1 \cup O_2)$.

Proof. Let $\mu \in [(r_1 \cap r_2)^* \|E_1\| \|E_2\|](s)$, we need to show that $\mu(O_1 \cap O_2) \geq p_1 + p_2 - q$ with the above definition of $p_1, p_2$ and $q$.

Let us define $Q_1$ to be the (single event) BES associated to the probabilistic program $u_1$ such that if $s \in P$, then $u_1(s) = \{\mu \mid \mu(O) \geq p\}$ else $u_1(s) = \emptyset$. Similarly, we define $Q_2$. Then the premises imply $P \cdot (r^*_1 \|E_1\|) \subseteq Q_1$ and $P \cdot (r^*_2 \|E_2\|) \subseteq Q_2$. By Proposition 6.1 we have
\[
[P \cdot ((r_1 \cap r_2)^* \|E_1\| \|E_2\|)] \subseteq [Q_1] \quad \text{and} \quad [P \cdot ((r_1 \cap r_2)^* \|E_1\| \|E_2\|)] \subseteq [Q_2].
\]
Therefore $\mu(O_1) \geq p_1$ and $\mu(O_2) \geq p_2$. Modularity of finite measures implies that $\mu(O_1 \cap O_2) + \mu(O_1 \cup O_2) = \mu(O_1) + \mu(O_2) \geq p_1 + p_2$. Hence, $\mu(O_1 \cap O_2) \geq p_1 + p_2 - \mu(O_1 \cup O_2) \geq p_1 + p_2 - \sup_{\mu \in [(r_1 \cap r_2)^* \|E_1\| \|E_2\|]}(s)\mu(O_1 \cup O_2)$.

Proposition A.2. For every $s \in P$ and subsets $O_1, O_2 \subseteq \Omega$, if the sequential refinement $r^* \|E_1\| \|E_2\| \subseteq (r^* \|E_1\|) \cdot (r^* \|E_2\|)$ and, for every $t \in O_1$, $[r^* \|E_2\|](t)(O_1) = 1$ hold, then the rule
\[
\begin{align*}
[r^* \|E_1\|](s)(O_1) & \geq p_1 & \inf_{t \in O_1} [r^* \|E_2\|](t)(O_2) & \geq p_2 & E_1 \sqsubseteq_{\text{sim}} g^* \sqsubseteq_{\text{sim}} r^* & E_2 \sqsubseteq_{\text{sim}} g'^* \sqsubseteq_{\text{sim}} r^* \\
[r^* \|E_1\| \|E_2\|](s)(O_1 \cap O_2) & \geq p_1 p_2 & \|E_1\| \|E_2\| & \sqsubseteq_{\text{sim}} (g + g')^* 
\end{align*}
\]
is valid.
Proof. We reason as follows:
\[
[r^* \parallel \mathcal{E}_1](s)(O_1 \cap O_2) \\
\geq r^* \parallel \mathcal{E}_1 \parallel \mathcal{E}_2 \subseteq (r^* \parallel \mathcal{E}_1) \cdot (r^* \parallel \mathcal{E}_2).
\]
\[
\inf_{\mu \in [r^* \parallel \mathcal{E}_1]} \inf_{f \subseteq [r^* \parallel \mathcal{E}_2]} \sum_{t \in \Omega} f(t)(O_1 \cap O_2) \mu(t)
\]
\[O_1 \subseteq \Omega, \ [r^* \parallel \mathcal{E}_2](t)(O_1) = 1 \text{ for every } t \in O_1 \text{ and modularity of measure.}
\]
\[
\inf_{\mu \in [r^* \parallel \mathcal{E}_1]} \inf_{f \subseteq [r^* \parallel \mathcal{E}_2]} \sum_{t \in O_1} f(t)(O_2) \mu(t)
\]
\[\geq \text{Expectation is monotonic.}\]
\[
\inf_{\mu \in [r^* \parallel \mathcal{E}_1]} \inf_{f \subseteq [r^* \parallel \mathcal{E}_2]} \sum_{t \in O_1} \inf_{t' \in O_1} f(t')(O_2) \mu(t)
\]
\[= \text{The factor } \inf_{t' \in O_1} f(t')(O_2) \text{ is independent of } t.
\]
\[
\inf_{\mu \in [r^* \parallel \mathcal{E}_1]} \inf_{f \subseteq [r^* \parallel \mathcal{E}_2]} \sum_{t' \in O_1} f(t')(O_2) \mu(O_1)
\]
\[= \text{Swapping the second and last inf (†).}\]
\[
\inf_{\mu \in [r^* \parallel \mathcal{E}_1]} \inf_{t' \in O_1} [r^* \parallel \mathcal{E}_2](t')(O_2) \mu(O_1)
\]
\[= \text{Both infs are attained for some value of } t' \text{ and } \mu, \text{ we can separate them.}\]
\[
[r^* \parallel \mathcal{E}_1](s)(O_1) \inf_{t' \in O_1} [r^* \parallel \mathcal{E}_2](t')(O_2)
\]
\[= \text{Definitions of } p_1 \text{ and } p_2.\]
\[p_1 p_2\]

(†) To ensure the replacement of the internal inf with \([r^* \parallel \mathcal{E}_2](t')(O_2)\) in presence of the multiplied constant \(\mu(O_1)\), we can use the fact that the infimum is attained for some deterministic refinement \(f \subseteq H [r^* \parallel \mathcal{E}]\) because \(\Omega\) is a finite set and \([r^* \parallel \mathcal{E}](s)\) is a compact subset of \(\mathbb{R}^\Omega\).

The guarantee part has been established in Proposition 6.1.