Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Statistical Model Checking QoS Properties of Systems with SBIP ⋆

Ayoub Nouri[1], **Saddek Bensalem**[1], **Marius Bozga**[1],
Benoit Delahaye[2], **Cyrille Jegourel**[3], **Axel Legay**[3]

[1] UJF-Grenoble 1 / CNRS VERIMAG UMR 5104, Grenoble, F-38041, France
[2] LINA/Université de Nantes, France
[3] INRIA/IRISA, Rennes, France

The date of receipt and acceptance will be inserted by the editor

Abstract. BIP is a component-based framework supporting rigorous design of embedded systems. BIP supports incremental design of large systems from atomic components that communicates via connectors and whose interactions can be described with a powerful algebra.

This paper presents SBIP, an extension of BIP for stochastic systems. SBIP offers the possibility to add stochastic information to atomic component's behaviors, and hence to the entire system.

Atomic component's semantics in SBIP is described by Markov Chains. We show that the semantics of the entire system is described by a Markov chain, showing that the non-determinism arising from system interactions is automatically eliminated by BIP. This allows us to verify systems described in SBIP with Statistical Model Checking.

This paper introduces SBIP and illustrates its usability on several industrial case studies.

1 Introduction

Expressive modeling formalism with sound semantical basis and efficient analysis techniques are essential for successful model-based development of embedded systems. While expressivity is needed for mastering heterogeneity and complexity, sound and rigorous models are mandatory to establish and reason meaningfully about system correctness and performance at design time.

The BIP (Behavior-Interaction-Priority) [4] formalism is an example of a highly expressive, component-based framework with rigorous semantical basis. BIP allows the construction of complex, hierarchically structured models from atomic components characterized by their behavior and their interfaces. Such components are transition systems enriched with variables. Transitions are used to move from a source to a destination location. Each time a transition is taken, component variables may be assigned new values, possibly computed by C functions. Atomic components are composed by layered application of interactions and priorities. Interactions express synchronization constraints between actions of the composed components while priorities are used both to select amongst possible interactions and to steer system evolution e.g. to express scheduling policies.

BIP is supported by an extensible toolset [10] which includes tools for checking correctness, for model transformations and for code generation. Correctness can be either formally proven using invariants and abstractions, or tested using simulation. For the latter case, simulation is driven by a specific middleware, the BIP engine, which allows to generate and explore execution traces corresponding to BIP models. Model transformations allow to realize static optimizations as well as special transformations towards distributed implementation of models. Finally, code generation targets both simulation and implementation models, for different platforms and operating systems support (e.g., distributed, multi-threaded, real-time, etc.). The tool has been applied to a wide range of academic case studies as well as to industrial applications [9].

BIP is currently equipped with a series of runtime verification [15] and simulation engines. While those facilities allow us to reason on a given execution, they cannot be used to assess the overall correctness of the entire

⋆ A preliminary version of this paper was published in [5], and the present submission is a special issue extension from [5]. The main differences between this version and the present special issue paper are: the extension of the background section, a complete description of the stochastic semantic of SBIP (including operational rules), a tutorial on how to use SBIP, and new experimental results. Research supported by the European Community's Seventh Framework Program [FP7] under grant agreements no 248776 (PRO3D), no 288917 (DALI), no 287716 (DANSE), no 257414 (ASCENS), the ARTEMIS JU grant agreement 2009-1-100208 (ACROSS), and Regional CREATIVE project ESTASE.

system. This paper presents SBIP, a stochastic extension of the BIP formalism and toolset. Adding stochastic aspects permits to model uncertainty in the design e.g., by including faults or execution platform assumptions. Moreover, it allows to enhance the simulation engine of BIP with statistical inference algorithms in order to reason on properties in a quantitative manner. Stochastic BIP relies on two key features.The first is a stochastic extension of the syntax and the semantics of the BIP formalism. This extension allows us to specify stochastic aspects of individual components and to produce execution traces of the designed system in a random manner.

The second feature is a Statistical Model Checking (SMC) [38,45,26,35,7,48,47,27] engine (SBIP) that, given a randomly sampled finite set of executions/simulations of the system, can decide with some confidence whether the system satisfies a given property. The decision is taken through either a Monte Carlo (that estimates the probability) [19], or an hypothesis testing algorithm [45, 38] (that compares the probability to a threshold). To guarantee termination of each simulation, these properties must be evaluated on bounded executions. Nevertheless, SMC has been recently extended to cover unbounded properties. Extension such as those introduced in [46,39,27,35] rely on an interleaving of estimation of probabilistic operator or a non stochastic exploration of the state space–two techniques known to be costly. In our work, we consider systems with finite life, hence bounded properties, expressed in Bounded Linear Temporal Logic (BLTL) are sufficient. Observe that the techniques in [46, 39,27,35] can be easily implemented in SBIP.

As it relies on sampling executions of a unique distribution, SMC can only be applied to pure stochastic systems i.e., systems without non-determinism. The problem is that most of component-based design approaches exhibit non-determinism due to interleaving semantics, usually adopted for parallel execution of components and their interactions. SBIP allows to specify systems with both non-deterministic and stochastic aspects. However, the semantics of such systems will be purely stochastic, as explained hereafter. Syntactically, we add stochastic behavior to atomic components in BIP by randomizing individual transitions. Indeed, it suffices to randomize the assignments of variables, which can be practically done in the C functions used on transition. Hence, from the user point of view, dealing with SBIP is as easy as dealing with BIP.

We illustrate the SBIP on several case studies that cannot be handled with existing model checkers for stochastic systems [30,25]. The presentation restricts to the analysis of a clock synchronization protocol [1] and an MPEG decoder [36]. Other examples can be found in [2].

Structure of the paper. Section 2 presents a background on stochastic systems, Probabilistic Bounded LTL, and Statistical Model Checking. Section 3 presents some background on BIP. The stochastic extension for BIP and its associated semantics are introduced in Section 4. Section 5 describes the statistical model checking procedure as well as its implementation in SBIP. In section 6 we describe practical utilization of the SBIP tool. Finally, Sections 7 and 8 present experiments and conclusion, respectively.

2 Stochastic Systems

In this section, we first introduce the underlying model used to represent stochastic systems that is, Markov Chains. Second, we discuss the logical formalism used to specify properties of such systems, then we introduce Statistical Model Checking.

2.1 Markov Chains

Let $\mathbb{B}$ be a set of atomic propositions and $\Sigma = 2^{\mathbb{B}}$.

Definition 1. A Labeled Markov Chain (LMC) $\mathcal{S}$ is a tuple $\langle S, Act, \iota, \pi, L^M \rangle$ where,

- S is a finite set of states,
- Act is a finite set of actions,
- $\iota : S \to [0,1]$ the initial states distribution such that $\sum_{s \in S} \iota(s) = 1$,
- $\pi : S \times Act \times S \to [0,1]$ the probability transition function such that for each $s \in S$ and $a \in Act$, $\sum_{s' \in S} \pi(s,a,s') = 1$, and
- $L^M : S \to \Sigma$ a state labeling function.

A labeled Markov chain is deterministic (DLMC) iff:

- $\exists s_0 \in S$ such that $\iota(s_0) = 1$, and
- $\forall s \in S$ and $a \in Act$, there exist at most one $s' \in S$ such that $\pi(s,a,s') > 0$.

Remark 1. We write $\pi(s_i, a, s_j) = \pi_{ij}$, the transition from s_i to s_j as $s_i \xrightarrow{a,\pi_{ij}} s_j$ for $s_i, s_j \in S, \pi_{ij} \in [0,1]$ and $a \in Act$.

2.2 Probabilistic Bounded Linear Time Logic

We use Probabilistic Bounded Linear Temporal Logic (PBLTL) as a formalism for describing stochastic temporal properties. We first recap Bounded Linear Temporal Logic and then define its probabilistic extension. The Bounded LTL formulas that can be defined from a set of atomic propositions $\mathbb{B}$ are the following.

- $\mathbf{T}$, $\mathbf{F}$, p, $\neg p$, for all $p \in \mathbb{B}$;
- $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, where ϕ_1 and ϕ_2 are BLTL formulas;
- $\bigcirc \phi_1$, $\phi_1 \mathcal{U}^t \phi_2$, where ϕ_1 and ϕ_2 are BLTL formulas, and t is a positive integer.

As usual, $\Diamond^t \phi = \mathbf{T}\mathcal{U}^t \phi$ and $\Box^t \phi = \neg(\mathbf{T}\mathcal{U}^t(\neg\phi))$. A Probabilistic BLTL formula is a BLTL formula preceded by a probabilistic operator P.

The semantics of a BLTL formula is defined with respect to an execution $\pi = s_0 s_1 \dots$ in the usual way [14]. Roughly speaking, an execution $\pi = s_0 s_1 \dots$ satisfies $\bigcirc\phi_1$, which we denote $\pi \models \bigcirc\phi_1$, if the sub-trace starting in state s_1 ($s_1 s_2 \dots$) satisfies ϕ_1. The execution π satisfies $\phi_1 \mathcal{U}^t \phi_2$ iff there exists a state s_i with $i{\leq}t$ that satisfies ϕ_2 and all the states in the prefix from s_0 to s_{i-1} satisfy ϕ_1.

Definition 2. The probability for a Markov Chain $\mathcal{S}$ to satisfy a BLTL formula ϕ is given by $\mu\{\pi \mid \pi \models \phi\}{\geq}\theta$, where π are executions of $\mathcal{S}$ and μ is its underlying probability measure. Such probability is always well-defined.

2.3 Statistical Model Checking

Consider a Markov Chain $\mathcal{S}$ and a BLTL property ϕ. *Statistical model checking* refers to a series of simulation-based techniques that can be used to answer two questions : (1) **Qualitative :** Is the probability for $\mathcal{S}$ to satisfy ϕ greater or equal to a certain threshold θ ? and (2) **Quantitative :** What is the probability for $\mathcal{S}$ to satisfy ϕ ? Let B_i be a discrete random variable with a Bernoulli distribution of parameter p. Such a variable can only take 2 values 0 and 1 with $Pr[B_i = 1] = p$ and $Pr[B_i = 0] = 1 - p$. In our context, each variable B_i is associated with one simulation of the system. The outcome for B_i, denoted b_i, is 1 if the i^{th} simulation satisfies ϕ and 0 otherwise.

Qualitative Answer using Statistical Model Checking The main approaches [45,38] proposed to answer the qualitative question are based on *hypothesis testing.* Let $p = Pr(\phi)$, to determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$. A test-based solution does not guarantee a correct result but it is possible to bound the probability of making an error. The *strength* (α, β) of a test is determined by two parameters, α and β, such that the probability of accepting K (respectively, H) when H (respectively, K) holds, called a Type-I error (respectively, a Type-II error) is less or equal to α (respectively, β). A test has *ideal performance* if the probability of the Type-I error (respectively, Type-II error) is exactly α (respectively, β). However, these requirements make it impossible to ensure a low probability for both types of errors simultaneously (see [45,41] for details). A solution is to use an *indifference region* $[p_1, p_0]$ (with θ in $[p_1, p_0]$) and to test $H_0 : p{\geq}p_0$ against $H_1 : p{\leq}p_1$. We now very briefly sketch an hypothesis testing algorithm that is called the *sequential probability ratio test (SPRT in short)* [41].

In SPRT, one has to choose two values A and B ($A > B$) that ensure that the strength (α, β) of the test is respected. Let m be the number of observations that have been made so far. The test is based on the following quotient:

$$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{Pr(B_i = b_i \mid p = p_1)}{Pr(B_i = b_i \mid p = p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}}, \tag{1}$$

where $d_m = \sum_{i=1}^{m} b_i$. The idea behind the test is to accept H_0 if $\frac{p_{1m}}{p_{0m}} \geq A$, and H_1 if $\frac{p_{1m}}{p_{0m}} \leq B$. The SPRT algorithm computes $\frac{p_{1m}}{p_{0m}}$ for successive values of m until either H_0 or H_1 is satisfied; the algorithm terminates with probability 1 [41]. This has the advantage of minimizing the number of simulations. In his thesis [45], Younes proposed a logarithmic based algorithm SPRT that given p_0, p_1, α and β implements the sequential ratio testing procedure. When one has to test $\theta{\geq}1$ or $\theta{\geq}0$, it is better to use *Single Sampling Plan* (SSP) (see [45, 32,38] for details) that is another hypothesis testing algorithm whose number of simulations is pre-computed in advance. In general, this number is higher than the one needed by SPRT, but is it known to be optimal for the above mentioned values. More details about hypothesis testing algorithms and a comparison between SSP and SPRT can be found in [32].

Quantitative Answer using Statistical Model Checking In [21,31] Peyronnet et al. propose an estimation procedure to compute the probability p for $\mathcal{S}$ to satisfy ϕ. Given a *precision* δ, Peyronnet's procedure, which we call PESTIMATION, computes a value for p' such that $|p' - p|{\leq}\delta$ with *confidence* $1 - \alpha$. The procedure is based on the *Chernoff-Hoeffding bound* [22]. Let $B_1 \dots B_m$ be m discrete random variables with a Bernoulli distribution of parameter p associated with m simulations of the system. Recall that the outcome for each of the B_i, denoted b_i, is 1 if the simulation satisfies ϕ and 0 otherwise. Let $p' = (\sum_{i=1}^{m} b_i)/m$, then Chernoff-Hoeffding bound [22] gives $Pr(|p' - p| > \delta) < 2e^{-\frac{m\delta^2}{4}}$. As a consequence, if we take $m{\geq}\frac{4}{\delta^2}\log(\frac{2}{\alpha})$, then $Pr(|p' - p|{\leq}\delta) \geq 1 - \alpha$. Observe that if the value p' returned by PESTIMATION is such that $p'{\geq}\theta - \delta$, then $\mathcal{S} \models Pr_{\geq\theta}$ with confidence $1 - \alpha$.

2.3.1 Playing with Statistical Model Checking Algorithms

The efficiency of the above algorithms is characterized by the number of simulations needed to obtain an answer. This number may change from executions to executions and can only be estimated (see [45] for an explanation). However, some generalities are known. For the qualitative case, it is known that, except for some situations, SPRT is always faster than SSP. PESTIMATION can also be used to solve the qualitative problem, but it is always slower than SSP [45]. If θ is unknown, then a good strategy is to estimate it using PESTIMATION with a low confidence and then validate the result with SPRT and a strong confidence.

3 BIP

The BIP framework, introduced in [4], supports a methodology for building systems from *atomic components*. It uses *connectors*, to specify possible interactions between components, and *priorities*, to select amongst possible interactions.

Atomic components are finite-state automata that are extended with variables and ports. Variables are used to store local data. Ports are action names, and may be associated with variables. They are used for interaction with other components. States denote control locations at which the components await for interaction. A transition is a step, labeled by a port, from a control location to another. It has associated a guard and an action that are, respectively, a Boolean condition and a computation defined on local variables. In BIP, data and their related computation are written in C. Formally:

Definition 3 (Atomic Component in BIP). An atomic component is a transition system extended with data $B = (L, P, T, X, \{g_\tau\}_{\tau\in T}, \{f_\tau\}_{\tau\in T})$, where:

– (L, P, T) is a transition system, with $L = \{l_1, l_2, \ldots, l_k\}$ a set of control locations, P a set of ports, and $T \subseteq L \times P \times L$ a set of transitions,
– $X = \{x_1, \ldots, x_n\}$ is a set of variables over domains $\{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$ and for each $\tau \in T$ respectively, $g_\tau(X)$ is a guard, a predicate on X, and $X' = f_\tau(X)$ is a deterministic update relation, a predicate defining X' (next) from X (current) state variables.

For a given valuation of variables, a transition can be executed if the guard evaluates to true and some *interaction* involving the port is enabled. The execution is an atomic sequence of two micro-steps: 1) execution of the interaction involving the port, which is a synchronization between several components, with possible exchange of data, followed by 2) execution of internal computation associated with the transition. Formally:

Definition 4 (Semantics of atomic component). The semantics of $B = (L, P, T, X, \{g_\tau\}_{\tau\in T}, \{f_\tau\}_{\tau\in T})$ is a transition system (Q, P, T_0) such that

– $Q = L \times \mathbf{X}$ where $\mathbf{X}$ denotes the set of valuations v_X of variables in X.
– T_0 is the set including transitions of the form $((l, v_X), p, (l', v'_X))$ such that $g_\tau(v_X) \wedge v'_X = f_\tau(v_X)$ for some $\tau = (l, p, l') \in T$. As usual, if $((l, v_X), p, (l', v'_X)) \in T_0$, we write $(l, v_X) \xrightarrow{p} (l', v'_X)$.

Composite components are defined by assembling sub-components (atomic or composite) using *connectors*. Connectors relate ports from different sub-components. They represent sets of interactions, that are, non-empty sets of ports that have to be jointly executed. For every such interaction, the connector provides the guard and the data transfer, that are, respectively, an enabling condition and an exchange of data across the ports involved in the interaction. Formally:

For a model built from a set of component $B_1, B_2, \ldots, B_n$, where $B_i = (L_i, P_i, T_i, X_i, \{g_\tau\}_{\tau\in T_i}, \{f_\tau\}_{\tau\in T_i})$ we assume that their respective sets of ports and variables are pairwise disjoint, i.e. for any two $i \neq j$ in $\{1 \ldots n\}$, we require that $P_i \cap P_j = \emptyset$ and $X_i \cap X_j = \emptyset$. Thus, we define the set $P = \bigcup_{i=1}^n P_i$ of all ports in the model as well as the set $X = \bigcup_{i=1}^n X_i$ of all variables.

Definition 5 (Interaction). An interaction a is a triple (P_a, G_a, F_a) where $P_a \subseteq P$ is a set of ports, G_a is a guard, and F_a is a data transfer function. We restrict P_a so that it contains at most one port of each component, therefore we denote $P_a = \{p_i\}_{i\in I}$ with $p_i \in P_i$ and $I \subseteq \{1 \ldots n\}$. G_a and F_a are defined on the variables available on the interacting ports $\bigcup_{p\in a} X_p$.

Given a set of interactions γ, the composition of the components following γ is the component $B = \gamma(B_1, \ldots, B_n) = (L, \gamma, \mathcal{T}, X, \{g_\tau\}_{\tau\in\mathcal{T}}, \{f_\tau\}_{\tau\in\mathcal{T}})$, where $(L, \gamma, \mathcal{T})$ is the transition system such that $L = L_1 \times \ldots \times L_n$ and $\mathcal{T} \subseteq L \times \gamma \times L$ contains transitions of the form $\tau = ((l_1, \ldots, l_n), a, (l'_1, \ldots, l'_n))$ obtained by synchronization of sets of transitions $\{\tau_i = (l_i, p_i, l'_i) \in T_i\}_{i\in I}$ such that $\{p_i\}_{i\in I} = a \in \gamma$ and $l'_j = l_j$ if $j \notin I$. The resulting set of variables is $X = \bigcup_{i=1}^n X_i$, and for a transition τ resulting from the synchronization of a set of transitions $\{\tau_i\}_{i\in I}$, the associated guard (resp. update relation) is the conjunction of the individual guards (resp. update relations) involved in the transition.

Finally, *priorities* provide a means to coordinate the execution of interactions within a BIP system. They are used to specify scheduling or similar arbitration policies between simultaneously enabled interactions. More concretely, priorities are rules, each consisting of an ordered pair of interactions associated with a condition. When the condition holds and both interactions of the corresponding pair are enabled, only the one with the highest priority can be executed. Non-determinism appears when several interactions are enabled. In the following, when we introduce probabilistic variables, we will thus have to make sure that non-determinism is resolved in order to produce a purely stochastic semantics.

Example 1. Figure 1 shows a graphical representation of an example model in BIP. It consists of atomic components *Sender*, *Buffer* and *Receiver*. The behavior of the *Sender* is described as a transition system with control locations l_1 and l_2. It communicates through ports *tick* and *out*. Port *out* exports the variable *x*. Components *Sender*, *Buffer* and *Receiver* are composed by two binary connectors *io1, io2* and a ternary connector *tick*. *tick* represents a rendezvous synchronization between the *tick* ports of the respective components. *io1* represents an interaction with data transfer from the port *out* of

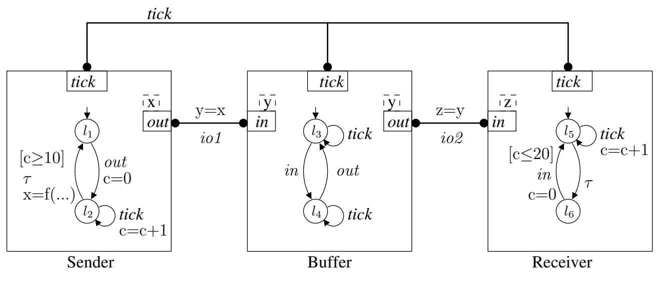


Fig. 1: BIP example: Sender-Buffer-Receiver system.

Sender to the port *in* of *Buffer*. As a result of the data transfer associated with *io1*, the value of variable x of *Sender* is assigned to the variables y of the *Buffer*.

BIP can model various types of synchronization. Using less expressive frameworks e.g. based on a single composition operator, often leads to intractable models. For instance, BIP directly encompasses multiparty interaction between components. Modeling multiparty interaction in frameworks supporting only point-to-point interaction e.g. function call or binary synchronization, requires the use of protocols. This can lead to overly complex models with complicated coordination structure. Similarly, priorities in BIP allow to express scheduling policies or general arbitration mechanisms between interactions in a declarative way. The use of scheduler components and explicit coordination between components may also obscure the overall design. The use of multiparty interactions and priorities confers a highly expressive power. This has been not only formally proven e.g., in [12] but also practically illustrated on the modeling of several complex case studies [3,1,2].

Finally, it is worth noticing that the clear separation between architecture (interactions and priorities) and behavior (automata) in BIP allows compositional and incremental analysis. This is advantageously exploited by tools like D-Finder [6] which separately analyzes behavior of atomic components and extracts interaction invariants characterizing architectural constraints.

4 SBIP: A Stochastic Extension for BIP

The stochastic extension of BIP allows (1) to specify stochastic aspects of individual components and (2) to provide a purely stochastic semantics for the parallel composition of components through interactions and priorities.

Stochastic Variables. Syntactically, we add stochastic behavior to atomic components in BIP by allowing the definition of probabilistic variables. Probabilistic variables x^P are attached to given distributions μ_{x^P} implemented as C functions. These variables can then be updated on transition using the attached distribution. The semantics on transitions is thus fully stochastic. We first define atomic components and interaction between them in SBIP, and then define the corresponding stochastic semantics.

Definition 6 (Atomic Component in SBIP). An atomic component in SBIP is a transition system extended with data $B = (L, P, T, X, \{g_\tau\}_{\tau\in T}, \{f_\tau\}_{\tau\in T})$, where $L, P, T, \{g_\tau\}_{\tau\in T}$ are defined as in Definition 3, and

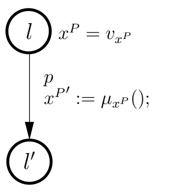
- $X = X_D \cup X_P$, with $X_D = \{x_1, \ldots, x_n\}$ the set of deterministic variables and $X_P = \{x_1^P, \ldots, x_m^P\}$ the set of probabilistic variables.
- For each $\tau \in T$, the update function $X' = f_\tau(X)$ is a pair $(X'_D = f_\tau^D(X), R_\tau)$ where $X'_D = f_\tau^D(X)$ is an update relation for deterministic variables and $R_\tau \subseteq X_P$ is the set of probabilistic variables that will be updated using their attached distributions. Remark that the current value of the probabilistic variables can be used in the update of deterministic variables.

In the following, given a valuation v_X of all the variables in X, we will denote by v_Y the projection of v_X on a subset of variables $Y \subseteq X$. When clear from the context, we will denote by v_y the valuation of variable $y \in X$ in v_X.

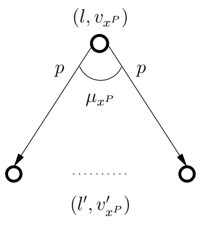
Some transitions in the associated semantics are thus probabilistic. As an example, consider an atomic component B with a transition τ that goes from a location l to a location l' using port p and updates a probabilistic variable x^P with the distribution μ_{x^P} over the domain $\mathbf{x^P}$. In the associated semantics, assuming the initial value of x^P is v_{x^P}, there will be several transitions from state (l, v_{x^P}) to states (l', v'_{x^P}) for all $v'_{x^P} \in \mathbf{x^P}$. According to the definition of probabilistic variables, the probability of taking transition $(l, v_{x^P}) \xrightarrow{p} (l', v'_{x^P})$ will then be $\mu_{x^P}(v'_{x^P})$. This example is illustrated in Figure 2. When several probabilistic variables are updated, the resulting distribution on transitions will be the product of the distributions associated to each variable. These distributions are fixed from the declaration of the variables, and are considered to be independent. The syntactic definitions of interactions and composition are adapted from BIP in the same manner. For the sake of simplicity, we restrict data transfer functions on interactions to be deterministic.

Remark 2. We write a transition in SBIP as $l_i \xrightarrow[f]{p,g} l_j$, where $l_i, l_j \in L, p \in P, g \in \{g_t\}_{t\in T}$ and $f \in \{f_t\}_{t\in T}$.

Stochastic Semantics for Atomic Components. Adapting the semantics of an atomic component in BIP as presented in Definition 4 to atomic components with probabilistic variables leads to transition systems that combine both stochastic and non-deterministic aspects. Indeed, even if atomic transitions are either purely deterministic or purely stochastic, several transitions can be



(a) Component B in SBIP



(b) Semantics of B according to SBIP

Fig. 2: Example of an abstract component B and its semantics in SBIP.

enabled in a given system state. In this case, the choice between these potential transitions is non-deterministic. In order to produce a purely stochastic semantics for components defined in SBIP, we resolve any non deterministic choice left after applying the priorities by applying uniform distributions. Remark that other distributions could be used to resolve this non-determinism and that using uniform distributions is the default choice we made. In the future, we will allow users to specify a different way of resolving non-determinism.

Consider a component $B = (L, P, T, X, \{g_\tau\}_{\tau\in T}, \{f_\tau\}_{\tau\in T})$ in SBIP. Given a state (l, v_X) in $L \times \mathbf{X}$, we denote by $\mathsf{Enabled}(l, v_X)$ the set of transitions in T that are enabled in state (l, v_X), i.e. transitions $\tau = (l, p, l') \in T$ such that $g_\tau(v_X)$ is satisfied. Since priorities only intervene at the level of interactions, the semantics of a single component does not take them into account. Remark that the set $\mathsf{Enabled}(l, v_X)$ may have a cardinal greater than 1. This is the only source of non-determinism in the component. In the semantics of B, instead of non-deterministically choosing between transitions in $\mathsf{Enabled}(l, v_X)$, we will choose probabilistically using a uniform distribution. Formally:

Definition 7 (Semantics of a single component in SBIP). The semantics of $B = (L, P, T, X, \{g_\tau\}_{\tau\in T}, \{f_\tau\}_{\tau\in T})$ in SBIP is a probabilistic transition system (Q, P, T_0) such that $Q = L \times \mathbf{X}$ and T_0 is the set of probabilistic transitions of the form $((l, v_X), p, (l', v'_X))$ for some $\tau = (l, p, l') \in \mathsf{Enabled}(l, v_X)$ such that $v'_{X_D} = f^D_\tau(v_X)$, and for all $y \in X_P \setminus R_\tau$, $v'_y = v_y$.

In a state (l, v_X), the probability of taking a transition $(l, v_X) \xrightarrow{p} (l', v'_X)$ is the following:

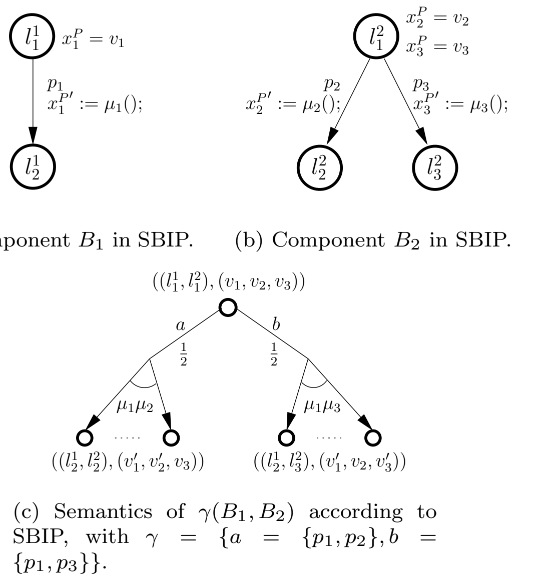
$$\frac{1}{|\mathsf{Enabled}(l, v_X)|}\left[\sum_{\substack{\{\tau\in\mathsf{Enabled}(l,v_X)\\ \text{s.t. } \tau=(l,p,l')\}}}\left(\prod_{y\in R_\tau}\mu_y(v'_y))\right)\right].$$

The probability of taking transition $(l, v_X) \xrightarrow{p} (l', v'_X)$ is computed as follows. For each transition $\tau = (l, p, l') \in \mathsf{Enabled}(l, v_X)$ such that $v'_{X_D} = f^D_\tau(v_X)$ and for each $y \in X_P \setminus R_\tau$, $v'_y = v_y$, the probability of reaching state (l', v'_X) is $\prod_{y\in R_\tau}\mu_y(v'_y)$. Since there may be several such transitions, we take the sum of their probabilities and normalize by multiplying with $\frac{1}{|\mathsf{Enabled}(l,v_X)|}$.

Stochastic Semantics for Composing Components. When considering a system with n components in SBIP $B_i = (L_i, P_i, T_i, X_i, \{g_\tau\}_{\tau\in T_i}, \{f_\tau\}_{\tau\in T_i})$ and a set of interactions γ, the construction of the product component $B = \gamma(B_1, \ldots, B_n)$ is defined as in BIP. The resulting semantics is given by Definition 7 above, where $\mathsf{Enabled}(l, v_X)$ now represents the set of *interactions* enabled in global state (l, v_X) that are maximal with respect to priorities. By construction, it follows that the semantics of any (composite) component in SBIP is purely stochastic.

Example 2. Consider SBIP components B_1 and B_2 given in Figures 3a and 3b. B_1 has a single probabilistic variable x^P_1, to which is attached distribution μ_1 and a single transition from location l^1_1 to location l^1_2 using port p_1, where x_1 is updated. In location l^1_1, the variable x^P_1 is assumed to have value v_1. B_2 has two probabilistic variables x^P_2 and x^P_3, to which are attached distributions μ_2 and μ_3 respectively. B_2 admits two transitions: a transition from location l^2_1 to location l^2_2 using port p_2, where x_2 is updated, and a transition from location l^2_1 to location l^2_3 using port p_3, where x_3 is updated. In location l^2_1, the variables x^P_2 and x^P_3 are assumed to have values v_2 and v_3 respectively. Let $\gamma = \{a = \{p_1, p_2\}, b = \{p_1, p_3\}\}$ be a set of interactions such that interactions a and b have the same priority. The semantics of the composition $\gamma(B_1, B_2)$ is given in Figure 3c. In state $((l^1_1, l^2_1), (v_1, v_2, v_3))$ of the composition, the non-determinism is resolved between interactions a and b, choosing one of them with probability 1/2. After choosing the interaction, the corresponding transition is taken, updating the corresponding probabilistic variables with the associated distributions. Remark that this gives rise to a single purely stochastic transition. As an example, the probability of going to state $((l^1_2, l^2_2), (v'_1, v'_2, v_3))$ with interaction a is $1/2 \cdot \mu_1(v'_1) \cdot \mu_2(v'_2)$, while the probability of going to state $((l^1_2, l^2_3), (v'_1, v_2, v'_3))$ with interaction b is $1/2 \cdot \mu_1(v'_1) \cdot \mu_3(v'_3)$.

An execution π of a BIP model is a sequence of states that can be generated from an initial state by following a



(a) Component B_1 in SBIP. (b) Component B_2 in SBIP.

(c) Semantics of $\gamma(B_1, B_2)$ according to SBIP, with $\gamma = \{a = \{p_1, p_2\}, b = \{p_1, p_3\}\}$.

Fig. 3: Illustration of the purely stochastic semantics of composition in SBIP.

sequence of (probabilistic) transitions. From the above, one easily sees that the semantics of any SBIP (composite) system has the structure of a discrete Markov chain. Consequently, one can define a probability measure μ on its set of executions in the usual way [34].

4.1 DTMC Modeling in SBIP

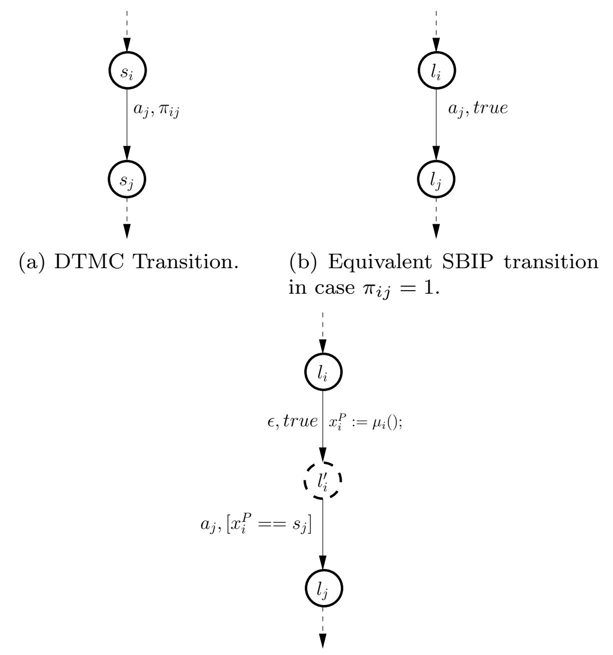
In the previous section, we saw that the semantics of an SBIP model is purely stochastic and is equivalent to a Discrete Markov Chain. In this section we provide an operational semantics that deals with Markov chains to SBIP model transformation.

Definition 8. Let ϵ be the empty action. Given a DTMC $M = \langle S, Act, \iota, \pi, L^M \rangle$, we define the transformation from M to a stochastic BIP model $B = (L, P, T, X, \{g_\tau\}_{\tau \in T}, \{f_\tau\}_{\tau \in T})$ as follow:

- $L = \{l_i$ for each $s_i \in S\} \cup \{l_i'$ for each $s_i \in S \mid \exists$ unique $a \in Act$ s.t. $\pi(s_i, a, s_j) = 1\}$,
- $P = Act \cup \{\epsilon\}$,
- $T \subseteq L^M \times P \times L^M$,
- $X = \{x_i^P$ for each $s_i \in S \mid \mu(x_i^P = s_j) = \pi_{ij}\}$, and

$$\frac{s_i \xrightarrow{a_j, \pi_{ij} > 0} s_j}{l_i \xrightarrow[x_i^P := \mu_i()]{\epsilon, true} l_i',\ l_i' \xrightarrow{a_j, [x_i^P == s_j]} l_j},\ if\ \pi_{ij} < 1 \quad (2)$$

$$\frac{s_i \xrightarrow{a_j, \pi_{ij} > 0} s_j}{l_i \xrightarrow{a_j, true} l_j},\ if\ \pi_{ij} = 1 \quad (3)$$



(a) DTMC Transition. (b) Equivalent SBIP transition in case $\pi_{ij} = 1$.

(c) Equivalent SBIP transition in case $\pi_{ij} < 1$.

Fig. 4: Illustration of the transformation from DTMC to SBIP model.

Intuitively, the transformation states that for a given Markov Chain M, each transition $s_i \xrightarrow{a, \pi_{ij}} s_j$ that has a probability $\pi_{ij} < 1$, is associated, in the corresponding SBIP model, with two transitions. The first is $l_i \xrightarrow[x_i^P := \mu_i()]{\epsilon, true} l_i'$ that is a probabilistic step based on the related distribution which is directly obtained from the DTMC (the one that characterize the next state weights from the state s_i), while the second is $l_i' \xrightarrow{a_j, [x_i^P == s_j]} l_j$ which stand for a next location choice as shown in Figure 4 and rule (2) of Definition 8. Another case is also presented in this definition where the transition probability $\pi_{ij} = 1$, the Markov Chain transition is then associated with a unique SBIP transition $l_i \xrightarrow{a_j, true} l_j$ as specified by rule (3) of the same definition. Note that, in the first case, the first transition correspond to a sampling operation over possible next locations ($x_i^P := \mu_i()$) (since there are more than one possible transition with different probabilities in the DTMC) and that the second transition uses BIP guards to select the next location with respect to the chosen value.

Example 3. Figure 5 shows the DTMC Model of a simple sending protocol. Initially, the protocol *try* to send which leads to the state s_1. From that state, the process could *try* again with probability 1/6, *fail* with probability 1/6, or *success* with probability 2/3. In case of fail, the protocol is restarted through the *init* action. The probabilities 1 on the transitions *try*, *init* and *success* are omitted.

Next State ($\mathbf{x_1^P}$ domain)	Probability ($\mu_1(x_1^P := s_i)$)
s_1	1/6
s_2	1/6
s_3	2/3

Table 1: Probability distribution in state s_1.

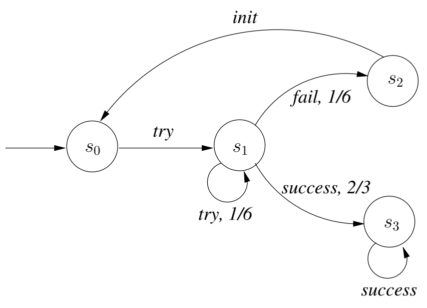


Fig. 5: A DLMC for a sending protocol example.

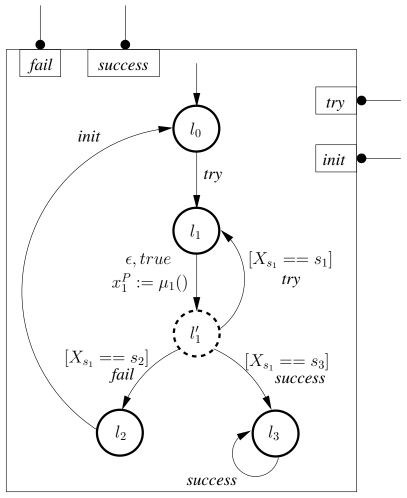


Fig. 6: Corresponding SBIP model for the sending protocol example.

The corresponding SBIP model is shown in figure 6. It consists in one SBIP component where the probabilistic variable x_1^P that models the next state distribution from s_1 is described in table 1.

Remark that the *try* transition from state s_0 in the DLMC in Figure 5 is preserved as it is in the SBIP component in Figure 6 as well as *init* and *success* transitions from state s_3. In fact, since their probabilities are equal to 1, the rule (3) of Definition 8 is applied. For the transitions *fail*, *success*, and *try* from state s_1 in the DLMC, they are transformed using the rule (2) since their probabilities are smaller that 1 which gives an additional sampling step from l_1 to l_1' in the SBIP component that uses the X_{s_1} distribution.

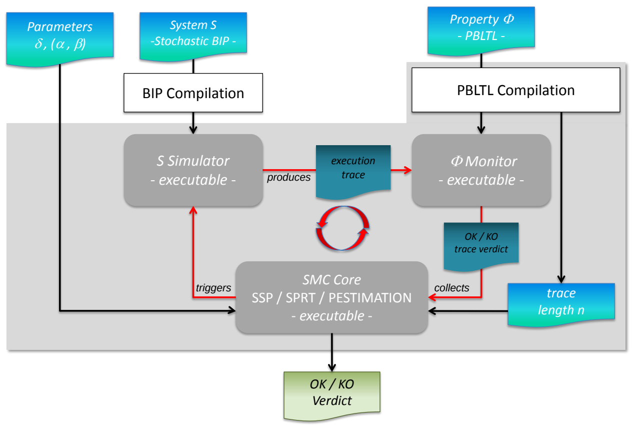


Fig. 7: SBIP tool architecture and work flow.

5 SMC for SBIP

Any statistical model checking of Markov Chains and BLTL properties requires to implement two routines: 1. a runtime verification procedure to decide whether a finite execution satisfies a BLTL formula, and 2. one or many SMC algorithms as described earlier. In this section, we first present the SMC capabilities and the architecture of SBIP. Then, we describe the implemented runtime verification procedure.

5.1 Tool Architecture

The SBIP tool [33] implements the statistical algorithms described in section 2, namely, *SSP*, *SPRT*, and *PESTIMATION* for stochastic BIP systems. Figure 7 shows the tool architecture and execution flow. SBIP takes as inputs a stochastic system written in the BIP language, a PBLTL property, and a series of confidence parameters needed by the statistical test. First, the tool generates an executable model and builds a monitor for the property under verification. Afterward, it iteratively triggers the stochastic BIP engine to generate random execution traces (sampling) which are checked with respect to the input property using the monitor. This procedure is repeated until a decision can be taken by the SMC core. As our approach relies on SMC and consider bounded LTL formulas, we are guaranteed that the procedure will eventually terminate.

5.2 Monitoring and Runtime Verification

Monitoring. For applying statistical model checking on stochastic systems it is mandatory to be able to evaluate the BLTL property under consideration on system execution traces. Indeed, this monitoring operation shall generate binary observations $x_i = \{0, 1\}$ (single trace verdict) which are requested by the statistical algorithms to provide a global verdict that concerns the whole system (all traces verdict). In theory, monitoring consists to check if some word (labeling the current execution trace) belongs to the language generated by some automaton encoding the property. Actually, there exist an important research literature about the efficient transformation from LTL to Buchi [17,43] or alternating [40] automata. Some of these works cover bounded LTL [16,18]. Nonetheless, despite these important theoretical results, it seems that no efficient method to transform BLTL to finite automata is yet established nor implemented.

To avoid this technical difficulty, in the current SBIP implementation, we restricted syntactically BLTL to a fragment where the temporal operators cannot be nested. This simplification restricts the definition to a finite number of automata patterns that covers all property classes. Moreover, this fragment has been expressive enough to cover all properties of interest in practical applications. Furthermore, it is always possible to enrich this set with additional patterns, as needed.

Runtime Verification (RV) [20,15,37] refers to a series of techniques whose main objective is to instrument the specification of a system (code, etc.) in order to observe and potentially refute complex properties at execution. The main issue of the runtime verification approach is, however, that it does not permit to assess the overall correctness of the entire system but only to identify potential errors.

In order to support runtime verification, the BIP framework allows for addition of observer components that enable to observe specific events of the system and/or to (partially) encode the evaluation of requirements (if they are otherwise difficult to express using BLTL). It is important to mention that such observers can be added to a BIP system in a totally non-intrusive way, that is, they run in parallel to the system components and only interact loosely with them, through specific connectors. A detailed presentation of the approach for construction and insertion of observers in BIP systems can be found in [44].

6 How to Use SBIP

In this section we show how to practically use the SBIP tool [33] to model a stochastic system and to verify it using statistical model checking technique.

6.1 Modeling in SBIP Language

The first step to use SBIP is to formally model the system to verify using the stochastic BIP formalism. Syntactically, using stochastic BIP is same as using BIP language [11] since the extension concerns essentially the semantics level and also because BIP is able to use external C++ code that is a strong way to extend it. Nevertheless, SBIP provides an additional library that should be used jointly with BIP and which provides probabilistic and tracing functionality to build an SBIP compatible model.

In the following we give an example of an SBIP component that uses the aforementioned functionality. We illustrate on the sending protocol component in Figure 5.

```
/* Declares an atomic BIP component */
atomic type sending_protocol

  /* Declares a probabilistic variable */
  data int Xs1
  /* Declares a probabilistic distribution */
  data distribution_t dist_1
  /* Declares an integer variable */
  data int success
  ...

  /* Declares and exports ports:
  init, try, fail, success */
  export port Port init
  export port Port try
  export port Port fail
  export port Port success
  /* Declares an internal BIP port */
  port Port epsilon
  ...

  /* Declares BIP locations */
  place l0, l1, l1', l2, l3

  /* Initialization */
  initial to l0 do {
    /* Init dist_1 from empirical dist. */
    dist_1 = init_distribution(``dist_1.txt'');
    /* update success flag and trace it */
    success = 0;
    trace_i(``sending_protocol.success'', success);
  }

  /* Transition from l0 to l1 */
  on try from l0 to l1
  on epsilon from l1 to l1' do {
      /* Updates Xs1 wrt. dist_1 */
      Xs1 = select(dist_1);
  }
  /* Transition from l1' to l1 */
  on try from l1' to l1 provided (Xs1 == s1)
  /* Transition from l1' to l3 */
  on success from l1' to l3 provided (Xs1 == s3) do {
    /* update success flag and trace it*/
    success = 1;
    trace_i(``sending_protocol.success'', success);
  }
  /* Transition from l1' to l2 */
  on fail from l1' to l2 provided (Xs1 == s2) do {
    /* update success flag and trace it*/
    success_flag = 0;
    trace_i(``sending_protocol.success'', success);
  }
  /* self loop on l3 */
  on success from l3 to l3
  /* Transition from l2 to l0 */
  on init from l2 to l0 do {
    /* update success flag and trace it*/
    success_flag = 0;
    trace_i(‘‘sending_protocol.success’’, success);
  }

end
```

The code above, describes the SBIP sending protocol model that uses some of the provided functionality in SBIP. For instance, the *distribution_t* predefined type is used to define a probabilistic distribution which is initialized, in this case, using *init_distribution()* function. This one optionally takes as input a text file that contains an empirical distribution. The declared distribution could be then used to update probabilistic variables (declared as classical BIP variables) using the *select()* function that returns a value with respect to its weight in the input distribution parameter. Similar functions could be also used to sample from standard probabilistic distributions such as *Uniform*, *Normal*, *Exponential*, etc. For instance, *Uniform* sampling could be done by just specifying the bounds of the interval to consider and without any initialization. For example, the call $select(125, 500)$ returns uniformly selected values in the interval $[125, 500]$.

Remark 3. The choice of using text files to describe empirical distributions, is made for practical reasons. Such files are usually automatically generated through system simulation.

Another functionality shown in this code is variables tracing which is mandatory to do trace monitoring. SBIP provides several tracing procedures with respect to variables type: *trace_i()* for Integer, *trace_b()* for Boolean, *trace_d()* for Double, and *trace_f()* for Float. Those functions take as parameters a string that specifies the component name and the variable name, in addition to the variable value. In the code sample above, the variable of interest that is, subject to verification, is *success* (note that this step of code annotation with tracing functions should be done when a property to check is fixed that is, to identify the variables to trace). This variable is of type Integer, hence the function call

```
trace_i(‘‘sending_protocol.success’’, success)
```

is used.

6.2 Properties Specification in SBIP

Whenever, the stochastic BIP model is built, the next step is to specify the property to be checked. As mentioned before, in the case of SBIP, this should be done in PBLTL syntax which is defined with respect to the following grammar:

$$\begin{aligned}
&\Phi ::= P \geqslant \theta[\Psi] \mid P =?[\Psi] \\
&\Psi ::= \varphi\ U\{i\}\ \varphi \mid (\ G\{i\} \mid F\{i\}\ )\ \varphi \mid N\ \varphi \\
&\varphi ::= true \mid false \mid \omega \mid \varphi\ (\ \wedge \mid \vee\ )\ \varphi \\
&\omega ::= \upsilon \mid !\,\upsilon \mid \varepsilon\ (\ > \ \mid\ < \ \mid\ \geq \ \mid\ \leq \ \mid\ = \ \mid\ \neq\ )\ \varepsilon \\
&\varepsilon ::= \upsilon \mid K \mid \varepsilon\ (\ + \mid - \mid \times \mid / \mid \%\ )\ \varepsilon \mid F(\upsilon, \cdots, \upsilon)
\end{aligned}$$

In this grammar, θ is a probability threshold, U, G, F, N are respectively *Until*, *Always*, *Eventually*, and *Next* temporal operators, i is an integer bound on the mentioned operators, υ is a state variable, K is an integer constant, and F denotes predefined functions.

Note that it is possible through this syntax to either ask for a probability estimation using $P =?$ operator or to check if the property probability respects some bound θ using $P \geqslant \theta$ operator. For example, given the SBIP model of the sending protocol above, a requirement to check could be that the probability to send always succeed is greater than a fixed threshold $\theta = 0.9$, which is formulated in PBLTL as follow:

$$P \geq 0.9[G\{1000\}(sending\_protocol.success)]$$

It is also possible to ask what is the probability that the send action eventually fails which is specified in PBLTL as follow:

$$P =?[F\{1000\}(!sending\_protocol.success)]$$

6.3 Statistical Model Checking with SBIP

Once the stochastic BIP model and the corresponding PBLTL properties are ready, SBIP could be used as follow to probabilistically check if the specified property hold on the system under consideration.

To use SBIP tool, the first step to do is to download it from the Web page on `http://www-verimag.imag.fr/Statistical-Model-Checking.html`. In addition, you should download and correctly set up the BIP tool (SBIP works with the old and the new BIP version) to be able to build stochastic BIP models as shown above.

When downloaded and extracted, the obtained tool directory is structured as follow:

– **lib\** directory which hold tool libraries/dependencies,
– **bin\** directory that contains tool binaries,
– **examples\** directory that contains some stochastic BIP examples,
– **setup.sh** file that should be used to install the tool, and finally,
– **README** file that explains the tool usage.

To set up the tool environment, go under the tool root directory and type the command below:

```
$ source setup.sh
```

Henceforth, it is possible to statistical model check stochastic systems built as BIP models using the following command prototype:

```
$ sbip [-htest|-pestim] [Formula] [Delta] [Alpha]
       [Beta] [-bip1|-bip2] [Executable]
```

where [-htest | -pestim] are options to specify hypothesis testing or probability estimation as statistical test, [Formula] is a PBLTL formula to check, [Delta], [Alpha] and [Beta] defines the level of confidence of the statistical tests, [-bip1 | -bip2] are options to specify which BIP language version will be used, and finally, [Executable] is the BIP binary of the system to verify. For example, to verify the quantitative property $P >= 0.8[G\{10000\}(sending\_protocol.success)]$ with confidence 10^{-5} on the sending sending protocol model, it is possible to use the following command:

```
$ sbip -htest \
"P >= 0.8[G{1000}(sending_protocol.success)] " \
0.05 0.00001 0.00001 -bip2 sending_protocol
```

The command line specifies that the hypothesis testing technique is used as statistical test (with 0.8 as a threshold), and that the probability to make errors (typeI and typeII) is bounded to 10^{-5} with an indifference region of 5.10^{-2}.

7 Case Studies

While still at prototype level, SBIP has been already applied to several case studies coming from serious industrial applications.

7.1 Accuracy of Clock Synchronization Protocol IEEE.1588

Model Description. The case study concerns a clock synchronization protocol running within a distributed heterogeneous communication system (HCS) [1]. This protocol allows to synchronize the clocks of various devices with the one of a designated server. It is important that this synchronization occurs properly, i.e., that the difference between the clock of the server and the one of any device is bounded by a small constant.

To verify such property, we build the stochastic model depicted in Figure 8. This model is composed by two deterministic components namely *Master*, and *Slave* and two communication channels. In the PTP model, the time of the master process is represented by the clock variable θ_m. This is considered the reference time and is used to synchronize the time of the slave clock, represented by the clock variable θ_s. The synchronization works by messages exchange between the server and a slave device. Each one of them saves the time of message reception $(t_i)_{i=1,4}$ with respect to its local clock. Finally, the slave device computes the offset between its time and the master time and updates its clock accordingly. Communication channels have been modeled using stochastic components. These components model communication delays over the network using empirical distributions obtained by simulating a detailed HCS model.

The accuracy of the synchronization is defined by the absolute value of the difference between the master and slave

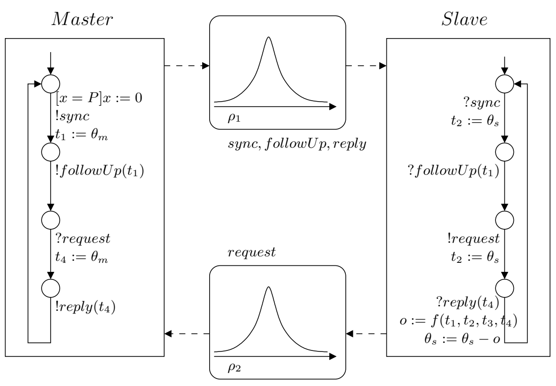


Fig. 8: PTP stochastic model.

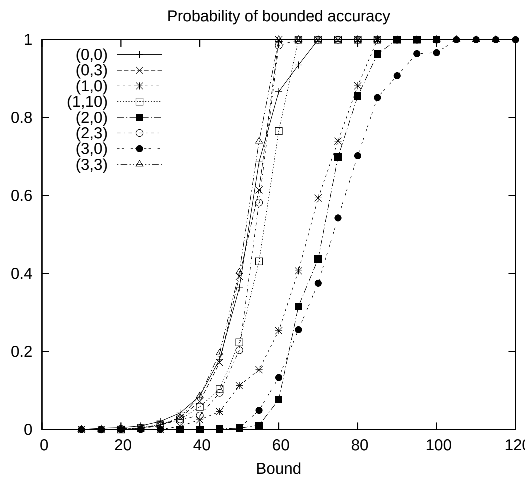


Fig. 9: Probability of satisfying bounded accuracy property as functions of the bound Δ.

clocks $|\theta_m - \theta_s|$, during the lifetime of the system we consider (in this case, 1000 steps). Our aim is to verify the satisfaction of the bounded LTL formula $P =?[G\{1000\}(abs(Master.\theta_m - Slave.\theta_s) >= \Delta)]$ for arbitrary fixed non-negative Δ.

Experiments and results. Two types of experiments are conducted. The first one is concerned with the bounded accuracy property ϕ. In the second one, we study average failure per execution for a given bound.

Property 1: Synchronization. To estimate the best accuracy bound, we have computed, for each device, the probability for synchronization to occur properly for values of Δ between $10\mu s$ and $120\mu s$. Figure 9 gives the results of the probability of satisfying the bounded accuracy property ϕ as a function of the bound Δ. The figure shows that the smallest bound which ensures synchronization for any device is $105\mu s$ (for Device $(3, 0)$). However, devices $(0, 3)$ and $(3, 3)$ already satisfy the property ϕ with probability 1 for $\Delta = 60\mu s$. For this experiments, we have used SPRT and SSP jointly with PESTIMATION for a higher degree of confidence. The results, which are presented in Table 2 for Device $(0, 0)$, show that SPRT is faster than SSP and PESTIMATION.

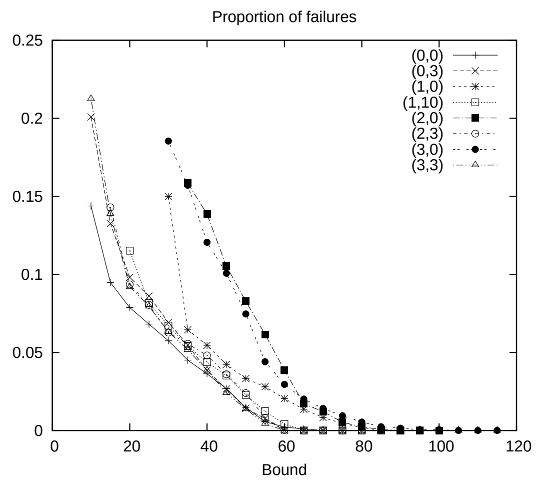


Fig. 10: Average proportion of failures as functions of the bound Δ.

Precision	10^{-1}		10^{-2}		10^{-3}	
Confidence	10^{-5}	10^{-10}	10^{-5}	10^{-10}	10^{-5}	10^{-10}
PESTIM	4883 $17s$	9488 $34s$	488243 $29m$	948760 $56m$	48824291 $> 3h$	94875993 $> 3h$
SSP	1604 $10s$	3579 $22s$	161986 $13m$	368633 $36m$	16949867 $> 3h$	32792577 $> 3h$
SPRT	316 $2s$	1176 $7s$	12211 $53s$	22870 $1m38s$	148264 $11m$	311368 $31m$

Table 2: Number of simulations / Amount of time required for PESTIMATION, SSP and SPRT.

Property 2: Average failure. In the second experiment, we try to quantify the average and worst number of failures in synchronization that occur *per simulation* when working with smaller bounds. Our goal is to study the possibility of using such bounds. For a given simulation, the *proportion of failures* is obtained by dividing the number of failures by the number of rounds of PTP. We will now estimate, for a simulation of 1000 steps (66 rounds of the PTP), the average value for this proportion. To this purpose, we have measured for each device this proportion on 1199 simulations with a different synchronization bounds Δ between $10\mu s$ and $120\mu s$. Figures 10 gives the average proportion of failure as a function of the bound.

7.2 Playout Buffer Underflow in MPEG2 Player

In multimedia literature [42], it has been shown that some quality degradation is tolerable when playing MPEG2-coded video. In fact, a loss under two consecutive frames within a second can be accepted. In this study, we want to check an MPEG2 player implementation with respect to the aforementioned QoS property, in addition to buffer size reduction [36].

Model Description. We illustrate the multimedia player setup that has been modeled using the stochastic BIP framework. The designed model captures the stochastic system

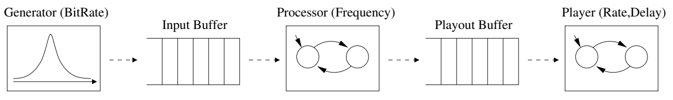


Fig. 11: MPEG2 player stochastic model.

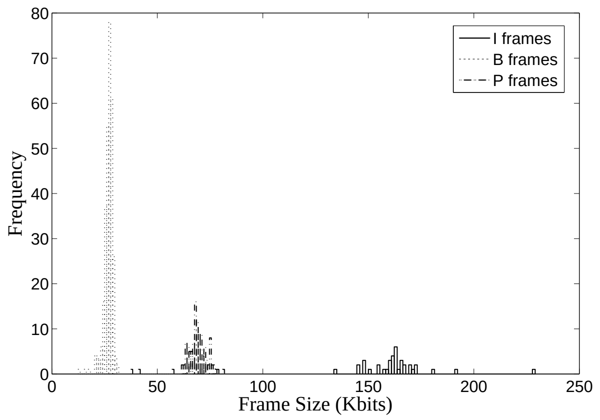


Fig. 12: Frequency distribution of I, P, and B frames in an MPEG2 video.

aspects that are, the macro-blocks arrival time to the input buffer and the their processing time.

The stochastic system model is shown in Figure 11. It consists of three functional components namely *Generator*, *Processor*, and *Player*. In addition to these, the buffers between the above functional components are modeled by explicit *buffer* components, namely *Input buffer* and *Playout buffer*. The transfer of the macro-blocks between the functional blocks and the buffers are described using interactions. All the functional components are timed, and the simulated time is modeled by the *tick* connector, which provides global synchronization between them.

The *Generator* is a stochastic component which models macro-blocks production based on three probabilistic distribution in a frame-type fashion as shown in Figure 12. It generates an MPEG2-coded stream with respect to a fixed Group-of-Pictures (GOP) pattern [28,29] and simulates the arrival time of macro-blocks to the *input buffer*.

The *Processor* reads them sequentially, decodes them and write them to the *Playout buffer*. The *Player* starts to read macro-blocks from the *Playout buffer* after a defined initial delay namely *Playout Delay*. Once this delay ends, the consumption is performed periodically with respect to a fixed consumption rate. Each period, the *Player* sends a request of N macro-blocks to the *Playout buffer*, where $N = 1$ the first time. Then it gets a response of M macro-blocks, where $0 \leq M \leq N$. An underflow happens when $M < N$. In this case, the next request N will be $(N - M) + 1$. That is, the player will try to read all the missed macro-blocks.

Experiments and results. To check the described model with respect to the desired QoS property, we used the SBIP tool. The PBLTL specification of the QoS property to check is $P = ?[G\{1500000\}(!Observer.fail)]$, where *fail* denotes a failure state condition corresponding to the underflow of two

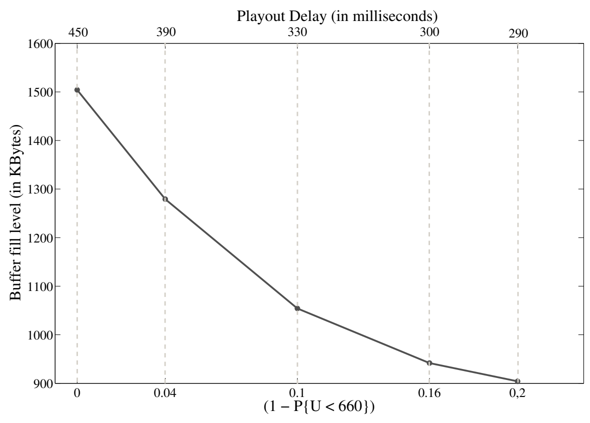


Fig. 13: Playout buffer fill level as function of playout delay and probability of property failure for `mobile.m2v` video.

consecutive frames within a second. The *fail* state is represented in an *Observer* BIP component which captures the failure condition by monitoring the *Player* frame consumption.

Figure 13 shows a bench of results for the `mobile.m2v` open source video. In this figure, the x-axis represents the probability of failure (a loss of two consecutive frames within a second) and the y-axis illustrates the playout buffer fill level. In addition, it shows, in the top, the playout delay evolution. We can see first, that for a high playout delay, the playout buffer is highly filled and hence that the probability of underflow is null. If we start reducing the playout delay, the playout buffer fill level decreases, which induces some probability of failure since the player starts to consume the frames sooner. The goal of the analysis is to enable designer to choose a trade-off amount of quality degradation that reduces the buffer size and does not imply a big playout delay.

8 Conclusion and Related Work

Stochastic systems can also be analyzed with a pure probabilistic model checking approach. While there is no clear winner, SMC is often more efficient in terms of memory and time consumption [23]. The above experiments are out of scope of probabilistic model checking. Also, there are properties such as clock drift in Clock Synchronization Protocols (see [1]) that could not have been analyzed with a pure formal approach. The PRISM toolset [30] also incorporates a statistical model checking engine. However, it can only be applied to those systems whose individual components are purely stochastic. Moreover, probability distributions are described in a very simple and restrictive language, while we can use the full fledged C to describe complex distributions. Nevertheless, we have observed that PRISM can be faster than our tool on various case studies such as those where the same process is repeated a certain number of times. A comparison between PRISM and SBIP is beyond the scope of this paper. Solutions to considerably enhance the efficiency of SMC in particular cases have recently been developed [24], but have not yet been implemented in SBIP. In a recent work [13], it has been proposed to use partial order to solve non-determinism when applying SMC (which rarely works). Another approach [8] consists to automatically synthesize distributed scheduling that accounts for concrete implementation information to solve non-determinism. In SBIP, the order is directly given in the design through priorities specified by the user.

We shall continue the development by implementing new heuristics to speed up simulation and to reduce their number as well as techniques to support unbounded properties. We shall also implement an extension of the stochastic abstraction principle from [1] that allows to compute automatically a small stochastic abstraction from a huge concrete system.

References

1. A. Basu, S. Bensalem, M. Bozga, B. Caillaud, B. Delahaye, and A. Legay. Statistical abstraction and model-checking of large heterogeneous systems. In *FORTE*, volume 6117 of *LNCS*, pages 32–46. Springer, 2010.
2. A. Basu, S. Bensalem, M. Bozga, B. Delahaye, A. Legay, and E. Sifakis. Verification of an afdx infrastructure using simulations and probabilities. In *Proceedings of the First international conference on Runtime verification*, RV'10, pages 330–344, Berlin, Heidelberg, 2010. Springer-Verlag.
3. A. Basu, S. Bensalem, M. Gallien, F. Ingrand, C. Lesire, T.-H. Nguyen, and J. Sifakis. Incremental component-based construction and verification of a robotic system. In *ECAI*, 2008.
4. A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Systems in BIP. In *SEFM06*, pages 3–12, September 2006.
5. S. Bensalem, M. Bozga, B. Delahaye, C. Jégourel, A. Legay, and A. Nouri. Statistical model checking qos properties of systems with sbip. In *ISoLA (1)*, pages 327–341, 2012.
6. S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis. D-finder: A tool for compositional deadlock detection and verification. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, pages 614–619, Berlin, Heidelberg, 2009. Springer-Verlag.
7. S. Bensalem, B. Delahaye, and A. Legay. Statistical model checking: Present and future. In *RV*, volume 6418 of *LNCS*. Springer, 2010.
8. S. Bensalem, A. Legay, A. Nouri, and D. Peled. Synthesizing distributed scheduling implementation for probabilistic component-based systems. In *MEMOCODE*, pages 87–96, 2013.
9. S. Bensalem, L. Silva, A. Griesmayer, F. Ingrand, A. Legay, and R. Yan. A formal approach for incremental construction with an application to autonomous robotic systems. In *SC'11*, LNCS. Springer, 2011.
10. Bip tools. http://www-verimag.imag.fr/BIP-Tools,93.html.
11. Bip2 language. http://www-verimag.imag.fr/TOOLS/DCS/bip/doc/latest/html/.
12. S. Bliudze and J. Sifakis. The algebra of connectors-structuring interaction in bip. *IEEE Trans. Comput.*, 57(10):1315–1330, Oct. 2008.
13. J. Bogdoll, L.-M. Fiorti, A. Hartmanns, and H. Hermanns. Partial order methods for statistical model checking and simulation. In *FORTE*, LNCS. Springer, 2011.

14. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
15. Y. Falcone, M. Jaber, T.-H. Nguyen, M. Bozga, and S. Bensalem. Runtime verification of component-based systems. In *SEFM*, pages 204–220, 2011.
16. B. Finkbeiner and H. Sipma. Checking finite traces using alternating automata. *Form. Methods Syst. Des.*, 24(2):101–127, Mar. 2004.
17. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65, Paris, France, July 2001. Springer.
18. D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In *Proceedings of the 16th IEEE international conference on Automated software engineering*, ASE '01, pages 412–, Washington, DC, USA, 2001. IEEE Computer Society.
19. R. Grosu and S. A. Smolka. Monte carlo model checking. In *TACAS*, volume 3440 of *LNCS*, pages 271–286. Springer, 2005.
20. K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *TACAS*, LNCS, pages 342–356. Springer, 2002.
21. T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *VMCAI*, pages 73–84, January 2004.
22. W. Hoeffding. Probability inequalities. *Journal of the American Statistical Association*, 58:13–30, 1963.
23. D. N. Jansen, J.-P. Katoen, M.Oldenkamp, M. Stoelinga, and I. S. Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In *HVC*, volume 4899 of *LNCS*. Springer, 2007.
24. C. Jégourel, A. Legay, and S. Sedwards. Cross entropy optimisation of importance sampling parameters for statistical model checking. In *CAV*, 2012.
25. C. Jégourel, A. Legay, and S. Sedwards. A platform for high performance statistical model checking - plasma. In *TACAS*, LNCS, pages 498–503. Springer, 2012.
26. J.-P. Katoen and I. S. Zapreev. Simulation-based ctmc model checking: An empirical evaluation. In *QEST*, pages 31–40. IEEE Computer Society, 2009.
27. J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker mrmc. In *QEST*, pages 167–176. IEEE Computer Society, 2009.
28. M. Krunz, R. Sass, and H. Hughes. Statistical characteristics and multiplexing of MPEG streams. In *INFOCOM*, pages 455–462, April 1995.
29. M. Krunz and S. K. Tripathi. On the characterization of VBR MPEG streams. In *SIGMETRICS*, pages 192–202, June 1997.
30. M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism 2.0: A tool for probabilistic model checking. In *QEST*, pages 322–323. IEEE, 2004.
31. S. Laplante, R. Lassaigne, F. Magniez, S. Peyronnet, and M. de Rougemont. Probabilistic abstraction for model checking: An approach based on property testing. *ACM TCS*, 8(4), 2007.
32. A. Legay and B. Delahaye. Statistical model checking : An overview. *CoRR*, abs/1005.1327, 2010.
33. A. Nouri, A. Legay, S. Bensalem, and M. Bozga. Sbip: A statistical model checking extension for the bip framework. In *First Workshop on Statistical Model Checking*, 2013.
34. E. Parzen. *Stochastic Processes*. Holden Day, 1962.
35. D. E. Rabih and N. Pekergin. Statistical model checking using perfect simulation. In *ATVA*, volume 5799 of *LNCS*, pages 120–134. Springer, 2009.
36. B. Raman, A. Nouri, D. Gangadharan, M. Bozga, A. Basu, M. Maheshwari, A. Legay, S. Bensalem, and S. Chakraborty. Stochastic modeling and performance analysis of multimedia socs. In *ICSAMOS*, pages 145–154, 2013.
37. G. Rosu and S. Bensalem. Allen linear (interval) temporal logic - translation to ltl and monitor synthesis. In *CAV*, volume 4144 of *LNCS*, pages 263–277. Springer, 2006.
38. K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. In *CAV*, LNCS 3114, pages 202–215. Springer, 2004.
39. K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *CAV*, pages 266–280, 2005.
40. M. Y. Vardi. Alternating automata and program verification. In *In Computer Science Today. LNCS 1000*, pages 471–485. Springer-Verlag, 1995.
41. A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.
42. D. Wijesekera and J. Srivastava. Quality of Service (QoS) Metrics for Continuous Media. *Multimedia Tools and Applications*, 3(2):127–166, July 1996.
43. P. Wolper. Lectures on formal methods and performance analysis. chapter Constructing automata from temporal logic formulas: a tutorial, pages 261–277. Springer-Verlag New York, Inc., New York, NY, USA, 2002.
44. F. Ylies, J. Mohamad, N. Thanh-Hung, B. Marius, and B. Saddek. Runtime verification of component-based systems in the bip framework with formally-proved sound and complete instrumentation. *SOSYM*, pages 1–27, 2013.
45. H. L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon, 2005.
46. H. L. S. Younes, E. M. Clarke, and P. Zuliani. Statistical verification of probabilistic properties with unbounded until. In *SBMF*, pages 144–160, 2010.
47. P. Zuliani, C. Baier, and E. M. Clarke. Rare-event verification for stochastic hybrid systems. In *HSCC*, pages 217–226. ACM, 2012.
48. P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *HSCC*, pages 243–252. ACM, 2010.