Towards a Call Behavior-Based Compositional Verification Framework for SysML Activity Diagrams

Samir Ouchani

Towards a Call Behavior–Based Compositional Verification Framework for SysML Activity Diagrams

Chapter · October 2019

DOI: 10.1007/978-3-030-32505-3_13

1 author:

Samir Ouchani
Groupe Cesi
100 PUBLICATIONS **667** CITATIONS

SEE PROFILE

Towards a Call Behavior-Based Compositional Verification Framework for SysML Activity Diagrams

Samir Ouchani

LINEACT, Laboratoire d'Innovation Numérique
École d'Ingénieur CESI, Aix-en-Provence, France

Abstract. SysML activity diagram is a standard modeling language for complex systems. It supports systems' composition by providing the operator '*call behavior*'. In general, the verification of systems modeled with those diagram inherit the limitations of the developed built-in tools, especially the case of model checking. To address this shortcoming, we propose a compositional verification framework based on the *call behavior* operator to alleviate the state space explosion problem of model-checking. The framework decomposes a property into local sub-properties and verify them separately on the composed behavioral diagrams. Further, we propose to ignore the diagrams artifacts that are useless with respect to the property under verification. We prove the soundness of the proposed approach by showing that the result deduced from the verification of the local properties is always preserved. The verification results are obtained by encoding SysML activity diagrams in the probabilistic model checker 'PRISM'. Finally, we demonstrate the effectiveness of our framework by verifying a set of properties on two use cases that require a large amount of memory and a considerable time processing.

Keywords: SysML; Activity Diagrams; Model-Checking; Compositional Verification; Abstraction, PCTL; PRISM.

1 Introduction

A major challenge in systems and software development process is to reduce as possible bugs by advancing the error detection at early stages of their life-cycles development. Experimentally, it has been shown that the cost of repairing a software flaw during maintenance is approximately 500 times higher than fixing it at early design phases [4]. Further, only 15% of flaws are detected in the initial design phase, whereas the cost of fixing them at this phase is extremely beneficial as compared to fixing them at the development and testing phases. Yet, a more ambitious challenge is to accelerate the verification process of a product based on its design artifacts. Here, we are interested on systems modeled by using modern and standard language like SysML [20]. The latter is a prominent object-oriented graphical language which today become defacto standard for software and systems modeling. Especially, SysML reuses a subset of UML packages [14] and extends others with specific systems' engineering features such as probability, time, and the rate. SysML covers mainly four perspectives of systems modeling: structure, behavior, requirement, and parametric diagrams. Particularly, SysML

activity diagrams are behavioral diagrams used to model system behaviors at various levels of abstraction [15].

For the verification of SysML activity diagrams, model checking is the most popular used technique [23]. Model checking [5] is a formal and automatic verification technique that checks systems specifications expressed as temporal logic formula or automata-based formalism on finite state concurrent systems. Compared to qualitative model checking, quantitative verification techniques based on probabilistic model checkers [4, 12] have recently gained popularity. Probabilistic verification offers the capability of measuring the satisfiability probability of a given property on systems that inherently exhibit probabilistic behavior. Despite its wide use, model checking in general is a resource-intensive process that requires a large amount of memory and time processing. This is due to the fact that the systems' state space may grow exponentially with the number of variables combined with the presence of concurrent behaviors. Consequently, it is of a major importance to reduce the verification process complexity.

To overcome this issue, various techniques have been explored [4, 5] for qualitative model checking and then leveraged to the probabilistic case. Among these techniques, several solutions aim at optimizing the employed model checking algorithms by introducing symbolic data structures based on binary decision diagrams, while others target the analysis of the model itself. Besides, two classes of solutions are found in the literature: abstraction and compositional verification. The former provides a minimized representation of the global system under verification. Whereas, the latter avoids the construction of the considered global system. Abstraction techniques can be classified into four categories [5]: abstraction by state merging, on variables, by restriction, or by observer automata. Besides, the well-known compositional verification techniques [6] are: partitioned transition relation, lazy parallel composition, interface processes, and assume-guarantee.

In this paper, we are interested by the interface processes and the abstraction by restriction techniques that are consistent within the composition by call behaviors in SysML activity diagrams. The provided framework considers as input a system modeled with SysML activity diagrams and its requirements expressed in PCTL [21]. Then it decomposes a property into local sub-properties in order to verify them separately for each system's sub-component in parallel. Further, in order to accelerate more the verification process, it ignores the diagrams artifacts that are useless with respect to the property and the local properties under verification. For verification, each system's component is transformed automatically into PRISM. Finally, the framework infers safely the verification result of the target property from the obtained results of the local properties. In a nutshell, the main contributions of this paper can be summarized as follows.

1. Proposing a complete formalization of the existing calculus dedicated to SysML activity diagrams.
2. Developing an efficient verification approach that reduces the verification costs overhead of probabilistic model checking.
3. Proving the soundness of the proposed approach.
4. Showing the effectiveness of the developed framework on two real use cases.

The next section compares our approach with the existing initiatives related to the verification of SysML activity diagrams. Then, the preliminaries needed for our work

are presented in Section 3. Section 4 describes and formalizes SysML activity diagrams. Then, our compositional verification framework is detailed in Section 5, and Section 6 presents the experimental results. Finally, Section 7 concludes the paper and provides future directions.

2 Related Work

In this section, we survey the research initiatives dedicated mainly to the formalization and the verification of SysML diagrams and to the compositional verification of probabilistic systems.

Yuan et al. [16] construct a set of rules to transform UML state machines to Timed Automata (TA). They apply the query view transformation approach in order to produce TA encoded in UPPAAL input language. The properties to be verified against TA are expressed in LTL. Apvrille and Saqui-Sannes [3] apply structural analysis to SysML by using the TTool open-source toolkit. They translate a subset of SysML diagrams into a Petri net and solves an equation system built upon the incidence matrix of the net. Then, a push-button approach is applied to display verification results.

Ando et al. [1] express SysML state machine diagrams in CSP# processes that could be verified by the PAT model checker. This work includes only a sub-set of rules and experimenting the transformation on a toy case study. In addition, they did not detail the temporal logic that expresses the system requirements. Carrillo et al. [8] define SysML blocks in a refinement process. The structural architecture of a SysML block is given by the internal block diagram and the behavior of each sub-block is described by an interface automaton. Their main intention in a refinement process is to ensure the consistency and the compatibility between different blocks.

Ermeson et al. [7] verify the embedded realtime systems with energy constraints that are modeled using SysML State Machine diagram, and the MARTE UML Profile (Modeling and Analysis of Real-Time and Embedded systems) is used to specify ERTSs (Embedded Real-time Systems) constraints such as execution time and energy. They map only states and transitions into ETPN (Time Petri Net with Energy constraints). In their transformation, they don't give the transformation of actions in a given state even the semantics of the mutual exclusive and orthogonal states by taking just the internal states into consideration. Furthermore, they propose a similar methodology [2] that maps one SysML activity diagram to time Petri Net for requirement validation of embedded real-time systems with energy constraints. The computation model formalized as an ETPN is not well presented and it misses the representation of the energy consumption values. The authors do not provide a formal transformation for SysML elements even the values represented from MARTE profile. Also, they do not clarify why they represent each constraint in an action by a separate transition.

Ouchani et al. [24] introduce the abstraction by merging states to reduce the verification cost of a SysML activity diagram. In [22], the authors transform a diagram into an equivalent hierarchical form in order to help the abstraction developed in [24].

David et al. [18] introduced an extension of UML statecharts with randomly varying duration that allows probabilistic decision in state. The Input/Output (I/O) automata is used to provide a compositional semantics for statecharts. Also, probability distribution

after a continuous or discrete time is introduced as an arbitrary operator. And in [17], they introduce means to specify system randomness within statecharts, and to verify probabilistic temporal properties. The model is represented as MDP, and the properties are expressed in PCTL.

Concerning the compositional verification for probabilistic systems, Feng et al. [11] discusses assume-guarantee technique for probabilistic system by focusing more on the learning algorithm to generate the minimal deterministic automata that represents a probabilistic safety property. And in [10], they propose the assume-guarantee approach where both the assumption and the guarantee properties are probabilistic safety properties such that assumptions are generated manually. Also in [9], they apply the assume-guarantee technique on synchronous systems modeled as DTMC, where assumptions are safety properties defined as probabilistic finite automata. To our knowledge, few probabilistic model checkers support abstraction and compositional verification techniques. As example, PRISM builds the symmetry reduction and LiQuor[1] implements the bi-simulation equivalence.

3 Preliminaries

In this section, we present the probabilistic automata as a modeling formalism and PCTL temporal logic as a specification language.

Probabilistic automata (PAs) [12] are a modeling formalism for systems that exhibit probabilistic and nondeterministic features. Definition 1 illustrates a PA where $Dist(S)$ denotes the set of convex distributions over S and $\mu = [\ldots, s_i \mapsto p_i, \ldots]$ is a distribution in $Dist(S)$ that assigns a probability $\mu(s_i) = p_i$ to the state s_i.

Definition 1 (Probabilistic Automaton). *A probabilistic automaton is a tuple* $M = (\overline{s},\ \mathrm{S},\ \mathrm{L},\ \Sigma,\ \delta)$*, where:*

- $\overline{s}$ *is an initial state, such that* $\overline{s} \in \mathrm{S}$*,*
- S *is a finite set of states,*
- $\mathrm{L} : S \to 2^{AP}$ *is a labeling function that assigns to each state a set of atomic propositions taken from the set of atomic propositions (AP),*
- Σ *is a finite set of actions,*
- $\delta : \mathrm{S} \times \Sigma \to\ Dist(S)$ *is a probabilistic transition function assigning for each* $s \in S$ *and* $\alpha \in \Sigma$ *a probabilistic distribution* $\mu \in Dist(S)$*.*

For PA's composition, this concept is modeled by the parallel composition as stipulated in Definition 2. During synchronization, each PA resolves its probabilistic choice independently. For transitions $s_1 \xrightarrow{\alpha} \mu_1$ and $s_2 \xrightarrow{\alpha} \mu_2$ that synchronize in α then the composed state (s_1', s_2') is reached from the state (s_1, s_2) with probability $\mu_1(s_1') \times \mu_2(s_2')$. In the no synchronization case, a PA takes a transition where the other remains in its current state with probability one.

Definition 2 (Parallel Composition of PAs). *The parallel composition of two PAs:* $M_1 = (\overline{s}_1,\ \mathrm{S}_1,\ \mathrm{L}_1,\ \Sigma_1,\ \delta_1)$ *and* $M_2 = (\overline{s}_2,\ \mathrm{S}_2,\ \mathrm{L}_2,\ \Sigma_2,\ \delta_2)$ *is a PA* $M = ((\overline{s}_1, \overline{s}_2),\ \mathrm{S}_1 \times$

[1] http://www.i1.informatik.uni-bonn.de/baier/projectpages/LIQUOR/LiQuor

$S_2,\ L(s_1) \cup L(s_2),\ \Sigma_1 \cup \Sigma_2,\ \delta)$, *where:* $\delta(S_1 \times S_2, \Sigma_1 \cup \Sigma_2)$ *is the set of transitions* $(s_1, s_2) \xrightarrow{\alpha} \mu_1 \times \mu_2$ *such that one of the following requirements is met.*

1. $s_1 \xrightarrow{\alpha} \mu_1, s_2 \xrightarrow{\alpha} \mu_2$, *and* $\alpha \in \Sigma_1 \cap \Sigma_2$,
2. $s_1 \xrightarrow{\alpha} \mu_1, \mu_2 = [s_2 \mapsto 1]$, *and* $\alpha \in \Sigma_1 \backslash \Sigma_2$,
3. $\mu_1 = [s_1 \mapsto 1],\ s_2 \xrightarrow{\alpha} \mu_2$, *and* $\alpha \in \Sigma_2 \backslash \Sigma_1$.

To verify a PA, we use PCTL to express its related specifications. The following grammar represents the PCTL syntax.

$$\phi\ ::=\ \top \mid ap \mid \phi \wedge \phi \mid \neg\phi \mid P_{\bowtie p}[\psi]$$
$$\psi\ ::= X\phi \mid \phi U^{\leq k} \phi \mid \phi U \phi$$

Where the term "$\top$" means *true*, "ap" is an atomic proposition, $k \in \mathbb{N}$, $p \in [0,1]$, and $\bowtie \in \{<, \leq, >, \geq\}$. The operator "$\wedge$" represents the *conjunction* and "$\neg$" is the *negation* operator, and P is the probabilistic operator. Also, "X", "$U^{\leq k}$", and "U" are the *next*, the *bounded until*, and the *until* temporal logic operators, respectively.

To specify a satisfaction relation of a PCTL formula in a state "s", a class of adversaries has been defined to solve the nondeterminism in PAs. Hence, a PCTL formula should be satisfied under all adversaries. The satisfaction relation ($\models$) of a PCTL formula is defined as follows, where "s" is a state and "π" is a path obtained by a memoryless adversary [12].

- $s \models \top$ is always satisfied.
- $s \models ap \Leftrightarrow\ ap \in L(s)$ and L is a labeling function.
- $s \models \phi_1 \wedge \phi_2 \Leftrightarrow\ s \models\ \phi_1 \wedge\ s \models\ \phi_2$.
- $s \models \neg\phi \Leftrightarrow\ s \not\models\ \phi$.
- $s \models P_{\bowtie p}[\psi] \Leftrightarrow\ P(\{\pi$ *is a path starts from the state* $s | \pi\ \models \psi\}) \bowtie p$.
- $\pi \models X\phi \Leftrightarrow \pi(1) \models \phi$ where $\pi(1)$ is the second state of π.
- $\pi \models \phi_1 U^{\leq k} \phi_2 \Leftrightarrow \exists i \leq k : \forall j < i,\ \pi(j) \models \phi_1 \wedge \pi(i) \models \phi_2$.
- $\pi \models \phi_1 U \phi_2 \Leftrightarrow \exists\, k \geq 0 :\ \pi \models \phi_1 U^{\leq k} \phi_2$.

4 SysML Activity Diagrams Formalization

In this section, we describe and formalize SysML activity diagrams by providing an adequate syntax and semantics.

As illustrated in Fig. 6, SysML activity diagrams are a graph-based representation where their main constructs (Fig. 1) can be decomposed into two categories: activity nodes and activity edges. The former contains three types: activity invocation, object and control nodes. Activity invocation includes receive and send signals, action, and call behavior. Activity control nodes are initial, flow final, activity final, decision, merge, fork, and join nodes. Activity edges are of two types: control flow and object flow. Control flow edges are used to show the execution path through the activity diagram and to connect activity nodes. Object flow edges are used to show the flow of data between activity nodes. Concurrency and synchronization are modeled using forks and joins, whereas, branching is modeled using decision and merge nodes. While a decision

node specifies a choice between different possible paths based on the evaluation of a guard condition (and/or a probability distribution), a fork node indicates the beginning of multiple parallel control threads. Moreover, a merge node specifies a point from where different incoming control paths follow the same path, whereas a join node allows multiple parallel control threads to synchronize and rejoin. In addition, the call behavior action consumes its input tokens and invoke its specified behavior. The execution of the calling artifact is blocked until it receives a reply from the invoked behavior.

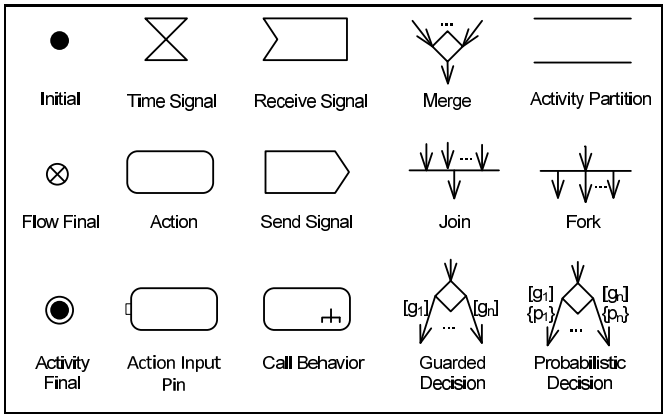


Fig. 1: SysML Activity Diagram Constructs.

4.1 Syntax of SysML Activity Diagrams

The UML superstructure [14] specifies basic rules for the execution of the various nodes by explaining textually how tokens are passed from one node to another. For formalization, we present in Table 1 SysML activity diagrams constructs and their representation as NuAC terms. At the beginning, a first token starts flowing from the initial node and moves downstream from one node to another with respect to the foregoing set of control routing rules defined by the control nodes until reaching either an activity final or a flow final node.

However, activity diagram semantics as specified in the standard stay informal since it is explained textually. We present in Fig. 2 the Backus-Naur-Form of the new version of Activity Calculus (NuAC) that helps to formalize SysML activity diagrams. This version of NuAC calculus optimizes the syntax presented in [24] and allows for multiplicity in join, merge, fork, and decision constructs by exploiting their commutativity and associativity properties. We denote by $\mathscr{A}[\mathscr{N}]$ to specify $\mathscr{N}$ as a sub term of $\mathscr{A}$ and by $|\mathscr{A}|$ to denote a term $\mathscr{A}$ without tokens. For the call behavior case of $a \uparrow \mathscr{A}'$, we denote $\mathscr{A}[a \uparrow \mathscr{A}']$ by $\mathscr{A} \uparrow_a \mathscr{A}'$.

During the execution, the structure of the activity diagram is kept unmodified and the only changes is the tokens locus. The NuAC syntax was inspired by this idea so that

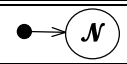
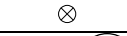
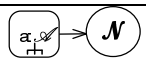
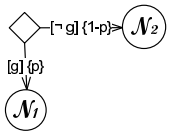
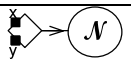
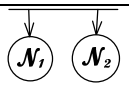
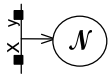
Activity Constructs	NuAC Terms	Description
	$l : \iota \rightarrowtail \mathscr{N}$	Initial node is activated when a diagram is invoked.
	$l : \odot$	Activity final node stops the execution of the diagram.
	$l : \otimes$	Flow final node terminates the execution in its path.
	$l : a \uparrow \mathscr{A} \rightarrowtail \mathscr{N}$	Action node defines an atomic action and it can invoke its related behavioral diagram.
	$l : D((p,g,\mathscr{N}),$ $(1-p,\neg g,\mathscr{N}))$	Decision node selects an execution path with a convex distribution $\{p, 1-p\}$ and/or a set of guards $\{g, \neg g\}$.
	$l : M(x,y) \rightarrowtail \mathscr{N},$ l_x or l_y	Merge node specifies the continuation, and x is the set of input flows $x = \{x_1, x_2\}$.
	$l : F(\mathscr{N}_1, \mathscr{N}_2)$	Fork node models the concurrency between $\mathscr{N}_1$ and $\mathscr{N}_2$. It begins multiple parallel control threads. UML 2.0 activity forks model unrestricted parallelism.
	$l : J(x,y) \rightarrowtail \mathscr{N},$	Join node presents the synchronization and x is the set of input pins $x = \{x_1, x_2\}$.

Table 1: Rewriting Activity Diagram Constructs in NuAC.

$$
\begin{array}{lll}
\mathscr{A} & ::= \varepsilon \;\mid\; \overline{l : \iota}^{\,n} \rightarrowtail \mathscr{N} & \\
\mathscr{N} & ::= \overline{\mathscr{N}}^{\,n} \;\mid\; l : M(x,y) \rightarrowtail \mathscr{N} \;\mid\; l : J(x,y) \rightarrowtail \mathscr{N} \;\mid\; l : F(\mathscr{N},\mathscr{N}) \;\mid\; \overline{l : a \uparrow \mathscr{A}}^{\,n} \rightarrowtail \mathscr{N} & \\
 & \quad \mid\; l : D((p,g,\mathscr{N}),(1-p,\neg g,\mathscr{N})) \;\mid\; l : \otimes \;\mid\; l : \odot \;\mid\; l &
\end{array}
$$

Fig. 2: Syntax of New Activity Calculus (NuAC).

a NuAC term presents a static structure while tokens are the only dynamic elements. We can distinguish two main syntactic terms: marked and unmarked. A marked NuAC term corresponds to an activity diagram with tokens. An unmarked NuAC term corresponds to the static structure of the diagram. A marked term is typically used to denote a reachable state that is characterized by the set of tokens locations in a given term.

To support multiple tokens, we augment the "overbar" operator with an integer n such that $\overline{\mathscr{N}}^{\,n}$ denotes a term marked with n tokens with the convention that $\overline{\mathscr{N}}^{\,1} = \overline{\mathscr{N}}$ and $\overline{\mathscr{N}}^{\,0} = \mathscr{N}$. Multiple tokens are needed when there are loops that encompass in their body a fork node. Furthermore, we use a prefix label for each node to reference it and uniquely use it in the case of a backward flow connection (case of merge or join). Particularly, labels are useful for connecting multiple incoming flows towards merge and join nodes. Let $\mathscr{L}$ be a collection of labels ranged over by $l_0, l_1, \cdots$ and $\mathscr{N}$ be any node (except initial) in the activity diagram. We write $l : \mathscr{N}$ to denote an l-labeled activity node $\mathscr{N}$. It is important to note that nodes with multi-inputs (e.g. join and merge) are

visited as many times as they have incoming edges. Thus, as a syntactic convention, we use either the NuAC term (i.e. $l : M(x,y) \rightarrowtail \mathscr{N}$ for merge and $l : J(x,y) \rightarrowtail \mathscr{N}$ for join) if the current node is visited for the first time or its corresponding label (i.e. l_x or l_y) if the same node is encountered later during the traversal process. Also, we denote by $D((g,\mathscr{N}_1),(\neg g,\mathscr{N}_2))$ or $D((p,\mathscr{N}_1),(1-p,\mathscr{N}_2))$ to express a decision without probabilities or guards, respectively.

4.2 Semantics of SysML Activity Diagrams

The execution of SysML activity diagrams is based on token's flow. To give a meaning to this execution, we use structural operational semantics to formally describe how the computation steps of NuAC atomic terms take place. The operational semantics of NuAC is based on the informally specified tokens-passing rules defined in [14].

INIT-1	$\overline{l : \iota} \rightarrowtail \mathscr{N} \xrightarrow{l} l : \iota \rightarrowtail \overline{\mathscr{N}}$
ACT-1	$\overline{l : a}^{m} \rightarrowtail \mathscr{N} \xrightarrow{l} \overline{l : a}^{m-1} \rightarrowtail \overline{\mathscr{N}} \quad \forall m > 0$
ACT-2	$\overline{\overline{l : a}^{m} \rightarrowtail \mathscr{N}}^{n} \xrightarrow{l} \overline{\overline{l : a}^{m+1} \rightarrowtail \mathscr{N}}^{n-1} \quad \forall m \geq 0, n > 0$
BH-1	$\dfrac{\mathscr{A} = l' : \iota \rightarrowtail \mathscr{N}' \quad \forall n > 0}{\overline{l : a \uparrow \mathscr{A}}^{n} \rightarrowtail \mathscr{N} \xrightarrow{l} \overline{l : a \uparrow \overline{l' : \iota} \rightarrowtail \mathscr{N}'}^{n-1} \rightarrowtail \mathscr{N}}$
BH-2	$\dfrac{\mathscr{A}[\overline{l' : \odot}] \xrightarrow{l'} \lvert\mathscr{A}\rvert \quad \forall n > 0}{\overline{l : a \uparrow \mathscr{A}}^{n} \rightarrowtail \mathscr{N} \xrightarrow{l'} \overline{l : a \uparrow \mathscr{A}}^{n} \rightarrowtail \overline{\mathscr{N}}}$
FORK-1	$\overline{l : F(\mathscr{N}_1,\mathscr{N}_2)}^{m} \xrightarrow{l} \overline{l : F(\overline{\mathscr{N}_1},\overline{\mathscr{N}_2})}^{m-1} \quad \forall m > 0$
PDEC-1	$\overline{l : D((p,g,\mathscr{N}_1),(1-p,\neg g,\mathscr{N}_2))}^{m} \xrightarrow{l}_{p} \overline{l : D((p,g,\overline{\mathscr{N}_1}),(1-p,\neg g,\mathscr{N}_2))}^{m-1} \quad \forall m > 0$
MERG-1	$\mathscr{A}[\overline{l : M(x,y) \rightarrowtail \mathscr{N}}^{n}, \overline{l_x}^{m}, \overline{l_y}^{k}] \xrightarrow{l_x} \mathscr{A}[\overline{l : M(x,y) \rightarrowtail \overline{\mathscr{N}}^{n}}, \overline{l_x}^{m-1}, \overline{l_y}^{k}] \quad \forall m > 0, k, n \geq 0$
MERG-2	$\mathscr{A}[\overline{l : M(x,y) \rightarrowtail \mathscr{N}}^{n}, \overline{l_x}^{m}, l_y] \xrightarrow{l_x} \mathscr{A}[\overline{l : M(x,y) \rightarrowtail \overline{\mathscr{N}}^{n}}, \overline{l_x}^{m-1}, l_y] \quad \forall m > 0, n \geq 0$
JOIN-1	$\mathscr{A}[\overline{l : J(x,y) \rightarrowtail \mathscr{N}}^{n}, \overline{l_x}^{m}, \overline{l_y}^{k}] \xrightarrow{l_x} \mathscr{A}[\overline{l : J(x,y) \rightarrowtail \overline{\mathscr{N}}^{n}}, \overline{l_x}^{m-1}, \overline{l_y}^{k-1}] \quad \forall m, k > 0, n \geq 0$
FLOWFINAL	$\mathscr{A}[\overline{l : \otimes}] \xrightarrow{l} \mathscr{A}[l : \otimes]$
FINAL	$\mathscr{A}[\overline{l : \odot}] \xrightarrow{l} \lvert\mathscr{A}\rvert$
ACTIVITY	$\dfrac{\mathscr{N} \xrightarrow{\alpha}_{p} \mathscr{N}'}{\mathscr{A}[\mathscr{N}] \xrightarrow{\alpha}_{p} \mathscr{A}[\mathscr{N}']}$

Fig. 3: NuAC Operational Semantic Rules.

We define Σ as the set of non-empty actions labeling the transitions (i.e. the alphabet of NuAC, to be distinguished from action nodes in activity diagrams). An element $\alpha \in \Sigma$ is the label of the executing active node. Let Σ^o be $\Sigma \cup \{o\}$ where o denotes the empty action. Let p be a probability value such that $p \in]0,1[$. The general form of a transition is $\mathscr{A} \xrightarrow{\alpha}_{p} \mathscr{A}'$ and $\mathscr{A} \xrightarrow{\alpha} \mathscr{A}'$ in the case of a Dirac (non probabilistic) transition. The probability value specifies the likelihood of a given transition to occur and it is denoted by $P(\mathscr{A},\alpha,\mathscr{A}')$. Fig. 3 shows the operational semantic rules of NuAC. The semantics of SysML activity diagrams expressed using $\mathscr{A}$ as a result of the defined semantic rules can be described in terms of the PA stipulated in Definition 3. In addition, we propose in Table 2 the NuAC axioms that are proved by using NuAC semantic rules.

Definition 3 (NuAC-PA). *A probabilistic automata of a NuAC term $\mathscr{A}$ is the tuple $M_{\mathscr{A}} = (\overline{s},\ \mathrm{L},\ \mathrm{S},\ \Sigma^o,\ \delta)$, where:*

- $\overline{s}$ *is an initial state, such that* $\mathrm{L}(\overline{s}) = \{\overline{l : \iota \rightarrowtail \mathscr{N}}\}$,
- $\mathrm{L} : S \to 2^{[\![\mathscr{L}]\!]}$ *is a labeling function where:* $[\![\mathscr{L}]\!] : \mathscr{L} \to \{\top, \bot\}$,
- S *is a finite set of states reachable from* $\overline{s}$*, such that,* $\mathrm{S} = \{s_{i:0\leq i\leq n} : \mathrm{L}(s_i) \in \{\overline{\mathscr{N}}\}\}$,
- Σ^o *is a finite set of actions corresponding to labels in* $\mathscr{A}$,
- $\delta : \mathrm{S} \times \Sigma^o \to \ Dist(S)$ *is a partial probabilistic transition function such that, for each* $s \in S$ *and* $\alpha \in \Sigma^o$ *assigns a probabilistic distribution* μ*, where:*
  - *For* $S' \subseteq \ S$ *such that* $S' = \{s_{i:0\leq i\leq n} : s \xrightarrow{\alpha}_{p_i} s_i\}$*, each transition* $s \xrightarrow{\alpha}_{p_i} s_i$ *satisfies one NuAC semantic rule and* $\mu(S') = \sum_{i=0}^{n} p_i = \sum_{i=0}^{n} \mu(s_i) = 1$.
  - *For each transition* $s \xrightarrow{\alpha}_1 s''$ *satisfying a NuAC semantic rule,* μ *is defined such that* $\mu(s'') = 1$.

DA-1	$l : D((p,g,\mathscr{N}_1),(1-p,\neg g,\mathscr{N}_2)) = l : D((1-p,\neg g,\mathscr{N}_2),(p,g,\mathscr{N}_1))$
DA-2	$l : D((p,\mathscr{N}_1),(1-p,l' : D((p',\mathscr{N}_2),(1-p',\mathscr{N}_3)))) = l : D((p+p'-p\times p',$ $l' : D((\frac{p}{p+p'-p\times p'},\mathscr{N}_1),(\frac{p'-p\times p'}{p+p'-p\times p'},\mathscr{N}_2))),(1-p-p'+p\times p',\mathscr{N}_3))$
DA-3	$l : D((p,g,\mathscr{N}_1),(1-p,\neg g,l' : D((p',g',\mathscr{N}_2),(1-p',\neg g',\mathscr{N}_3))))$ $= l : D((p,g,\mathscr{N}_1),(p'-p.p',\neg g\wedge g',\mathscr{N}_2),((1-p)(1-p'),\neg g\wedge\neg g',\mathscr{N}_3))$
FA-1	$l : F(\mathscr{N}_1,\mathscr{N})_1 = \mathscr{N}_1$
FA-2	$l : F(\mathscr{N}_1,\mathscr{N}_2) = l : F(\mathscr{N}_2,\mathscr{N}_1)$
FA-3	$l : F(\mathscr{N}_1,l' : F(\mathscr{N}_2,\mathscr{N}_3)) = l : F(l' : F(\mathscr{N}_1,\mathscr{N}_2),\mathscr{N}_3) = l : F(\mathscr{N}_1,\mathscr{N}_2,\mathscr{N}_3)$
JA-1	$\mathscr{A}[l : J(x,y) \rightarrowtail \mathscr{N}', \mathscr{N} \rightarrowtail l_x, \mathscr{N} \rightarrowtail l_y] = \mathscr{A}[\mathscr{N} \rightarrowtail \mathscr{N}']$
JA-2	$l : J(x,y) \rightarrowtail \mathscr{N} = l : J(y,x) \rightarrowtail \mathscr{N}$
JA-3	$\mathscr{A}[l : J(x,x') \rightarrowtail \mathscr{N}, l' : J(y,z) \rightarrowtail l_{x'}] = \mathscr{A}[l : J(x,y,z) \rightarrowtail \mathscr{N}]$
MA-1	$\mathscr{A}[l : M(x,y) \rightarrowtail \mathscr{N}', \mathscr{N} \rightarrowtail l_x, \mathscr{N} \rightarrowtail l_y] = \mathscr{A}[\mathscr{N} \rightarrowtail \mathscr{N}']$
MA-2	$l : M(x,y) \rightarrowtail \mathscr{N} = l : M(y,x) \rightarrowtail \mathscr{N}$
MA-3	$\mathscr{A}[l : M(x,x') \rightarrowtail \mathscr{N}, l' : M(y,z) \rightarrowtail l_{x'}] = \mathscr{A}[l : M(x,y,z) \rightarrowtail \mathscr{N}]$
CA-1	$l : a \uparrow \varepsilon = a$
CA-2	$\mathscr{A}_1 \uparrow_{a_1} (\mathscr{A}_2 \uparrow_{a_2} \mathscr{A}_3) = (\mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2) \uparrow_{a_2} \mathscr{A}_3 = \mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2 \uparrow_{a_2} \mathscr{A}_3$

Table 2: Axioms for NuAC.

5 The Approach

Fig. 4 depicts an overview of our compositional verification framework. It takes a set of SysML activity diagrams composed by the call behavior interface and a Probabilistic Computation Tree Logic (PCTL) [12] property as input. First, we develop an abstraction approach that restricts the verification of a PCTL property only on the influenced

diagrams instead of the whole composition. Then, we propose a compositional verification approach by interface processes that distributes a PCTL property into local ones which helps to verify them separately for each diagram. For verification, we encode the diagrams into the PRISM input language [19]. Finally, we deduce the result of the main property from the results of the local properties that are verified separately for each called diagram.

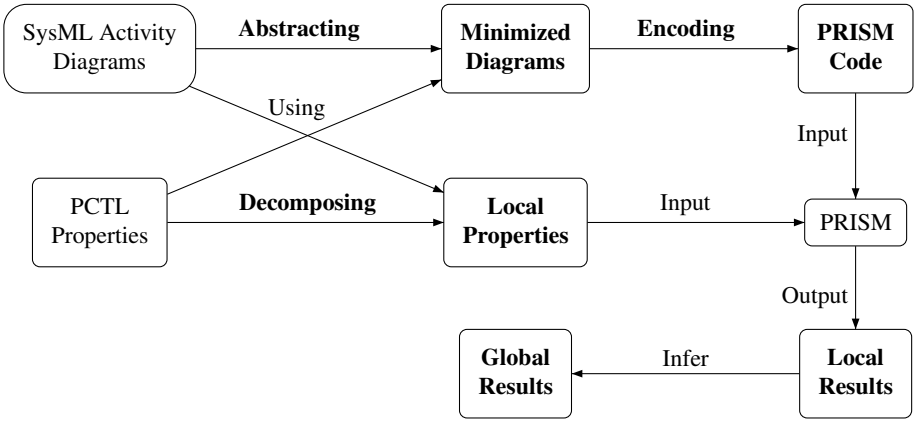


Fig. 4: A Compositional Verification Framework.

5.1 The Compositional Verification

Let $\mathscr{A}$ be a SysML activity diagram with n call behaviors denoted by $\mathscr{A} = \mathscr{A}_0 \uparrow_{a_0} \mathscr{A}_1 \cdots \mathscr{A}_{i-1} \uparrow_{a_{i-1}} \mathscr{A}_i \cdots \mathscr{A}_{n-1} \uparrow_{a_{n-1}} \mathscr{A}_n$. In order to reduce the diagram $\mathscr{A}$, we apply NuAC axioms and introduce the reduction rule defined in Definition 4 to remove diagrams $\mathscr{A}_i$ that are not influenced by the property ϕ to be verified. The obtained diagram after applying the reduction rule is denoted by $\widehat{\mathscr{A}}$.

Definition 4. *Let $\mathscr{A}$ be a diagram that contains n call behaviors, AP_ϕ is the atomic propositions of the PCTL property ϕ, and $AP_{\mathscr{A}_i}$ is the atomic propositions of the behavioral diagram $\mathscr{A}_i$. Reducing $\mathscr{A}$ to the diagram $\widehat{\mathscr{A}}$ with respect to ϕ is obtained by applying the following rule.*

$$\frac{\forall 0 \leq i \leq n, AP_\phi \cap AP_{\mathscr{A}_i} = \emptyset}{\mathscr{A}_i = \varepsilon}$$

Below, Proposition 1 shows the satisfiability probability after reduction.

Proposition 1. *For a reduced diagram $\widehat{\mathscr{A}}$ of $\mathscr{A}$ with respect to ϕ, we have:*

$$[\widehat{\mathscr{A}} \models \phi] \Rightarrow [\mathscr{A} \models \phi].$$

Proof. The proof of this proposition follows an induction reasoning on the PCTL structure. First, we take the case of $\psi = \phi_1 \mathrm{U} \phi_2$.

By definition, for $0 \leq i \leq n$ where $AP_{\psi} \cap AP_{\mathscr{A}_i} = \emptyset$, then: $\mathscr{A}_i = \varepsilon$. The result is $\widehat{\mathscr{A}} = \mathscr{A}_0 \uparrow_{a_0} \mathscr{A}_1 \cdots \mathscr{A}_{k-1} \uparrow_{a_{k-1}} \mathscr{A}_k$ and $k \leq n$.
From the PCTL semantics, we have $[(\mathscr{A}_0 \uparrow_{a_0} \mathscr{A}_1 \cdots \mathscr{A}_{k-1} \uparrow_{a_{k-1}} \mathscr{A}_k) \models \psi] \Leftrightarrow \exists m,\ \forall j < m : \pi(j) \models \phi_1 \wedge \pi(m) \models \phi_2$ where $\pi(j)$ and $\pi(m)$ are the states i and j respectively in a path π of $\mathscr{A}$. And, by calling $\mathscr{A}_i$ in a_i using BH-1, the only changes in the path π are the propositions of $\mathscr{A}_i$ till executing BH-2, then: $\exists m' \geq m,\ j' \geq j,\ \forall j' < m' : \pi(j') \models \phi_1 \wedge \pi(m') \models \phi_2 \Leftrightarrow\ \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k \ldots \uparrow_{a_i} \mathscr{A}_i \models \psi$.
By calling a new $\mathscr{A}_{i+1}$ in a_{i+1} up to n, we will have: $\exists m'' \geq m',\ j'' \geq j',\ \forall j'' < m'' : \pi(j'') \models \phi_1 \wedge \pi(m'') \models \phi_2 \Leftrightarrow \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_n} \mathscr{A}_n \models \psi \Leftrightarrow \mathscr{A} \models \phi_1 \mathrm{U} \phi_2$.
For $\phi_1 \mathrm{U}^{\leq k} \phi_2$ and $\mathrm{X}\phi$ cases, we deduce the following.

- $\forall 0 \leq i \leq n, AP_{\phi} \cap AP_{\mathscr{A}_i} = \emptyset : [\mathscr{A}_i = \varepsilon \wedge (\mathscr{A}_0 \uparrow_{a_0} \mathscr{A}_1 \cdots \mathscr{A}_{n-1} \uparrow_{a_{n-1}} \mathscr{A}_n) \models \phi_1 \mathrm{U}^{\leq k} \phi_2] \Rightarrow [\exists k' \geq k : \mathscr{A} \models \phi_1 \mathrm{U}^{\leq k'} \phi_2]$.
- $\forall 0 \leq i \leq n, AP_{\phi} \cap AP_{\mathscr{A}_i} = \emptyset : [\mathscr{A}_i = \varepsilon \wedge (\mathscr{A}_0 \uparrow_{a_0} \mathscr{A}_1 \cdots \mathscr{A}_{n-1} \uparrow_{a_{n-1}} \mathscr{A}_n) \models \mathrm{X}\phi] \Rightarrow [\mathscr{A} \models \mathrm{X}\phi]$.

□

For a parallel verification, we decompose the PCTL property ϕ into local ones $\phi_{i:0 \leq i \leq n}$ over $\mathscr{A}_i$ with respect to the call behavior actions $a_{i:0 \leq i \leq n}$ (interfaces), we introduce the decomposition operator "$\natural$" proposed in Definition 5. The operator "$\natural$" is based on substituting the propositions of $\mathscr{A}_i$ to the propositions related to its interface a_{i-1} which allows the compositional verification. We denote by $\phi[y/z]$ substituting the atomic proposition "z" in the PCTL property ϕ by the atomic proposition "y".

Definition 5 (PCTL Property Decomposition). *Let ϕ be a PCTL property to be verified on $\mathscr{A}_1 \uparrow_a \mathscr{A}_2$. The decomposition of ϕ into ϕ_1 and ϕ_2 is denoted by $\phi \equiv \phi_1 \natural_a \phi_2$ where $AP_{\mathscr{A}_i}$ are the atomic propositions of $\mathscr{A}_i$, then:*

1. *$\phi_1 = \phi([l_a/AP_{\mathscr{A}_2}])$, where l_a is the atomic proposition related to the action a in $\mathscr{A}_1$.*
2. *$\phi_2 = \phi([\top/AP_{\mathscr{A}_1}])$.*

The first rule is based on the fact that the only transition to reach a state in $\mathscr{A}_2$ from $\mathscr{A}_1$ is the transition of the action l_a (BH-1). The second rule ignores the existence of $\mathscr{A}_1$ while it kept unchanged till the execution of BH-2. To handle multiplicity for the operator "$\natural$", we have Property 1.

Property 1. The decomposition operator $\natural$ is associative for $\mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2 \uparrow_{a_2} \mathscr{A}_3$, i.e. :

$$\phi_1 \natural_{a_1} (\phi_2 \natural_{a_2} \phi_3) \equiv (\phi_1 \natural_{a_1} \phi_2) \natural_{a_2} \phi_3 \equiv \phi_1 \natural_{a_1} \phi_2 \natural_{a_2} \phi_3.$$

For the verification of ϕ on $\mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2$, Theorem 1 deduces the satisfiability of ϕ from the satisfiability of local properties ϕ_1 and ϕ_2 obtained by the operator $\natural$.

Theorem 1 (Compositional Verification). *The decomposition of the PCTL property ϕ by the decomposition operator $\natural$ for $\mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2$ is sound, i.e. :*

$$\frac{\mathscr{A}_1 \models \phi_1 \quad \mathscr{A}_2 \models \phi_2 \quad \phi = \phi_1 \natural_{a_1} \phi_2}{\mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2 \models \phi}$$

Proof. The proof of Theorem 1 follows a structural induction on the PCTL structure by using Definition 5. As an example, we take the until operator "U". Let $\phi = ap_1 \text{ U } ap_2$ where $ap_1 \in AP_{\mathscr{A}_1}$ and $ap_2 \in AP_{\mathscr{A}_2}$. By applying Definition 5, we have: $\phi_1 = ap_1 \text{ U } a_1$ and $\phi_2 = \top \text{ U } ap_2$. Let $\mathscr{A}_1 \models \phi_1 \Leftrightarrow \exists m_1,\ \forall j_1 < m_1 : \pi_1(j_1) \models ap_1 \wedge \pi_1(m_1) \models ap_1 \wedge a_1$ where π is a path in the NuAC PA of $\mathscr{A}$. For $\mathscr{A}_2 \models \phi_2 \Leftrightarrow \exists m_2,\ \forall j_2 < m_2 : \pi_2(j_2) \models \top \wedge \pi_2(m_2) \models ap_2$. To construct $\mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2$, BH-1 is the only transition to connect π_1 and π_2 which form: $\pi = \pi_1.\pi_2'$ such that $\pi_2'(i) = \pi_2(i) \cup \pi_1(m_1)$. Then: $\exists j \leq m,\ m = m_1 + m_2 :\ \pi(j) \models ap_1 \wedge \pi(m) \models ap_2 \Leftrightarrow \mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2 \models \phi$. □

Finally, Proposition 2 generalizes Theorem 1 to support the satisfiability of ϕ on an activity diagram $\mathscr{A}$ with n call behaviors.

Proposition 2 (CV-Generalization). *Let ϕ be a PCTL property to be verified on $\mathscr{A}$, such that: $\mathscr{A} = \mathscr{A}_0 \uparrow_{a_0} \cdots \uparrow_{a_{n-1}} \mathscr{A}_n$ and $\phi = \phi_0 \natural_{a_0} \cdots \natural_{a_{n-1}} \phi_n$, then:*

$$\frac{\mathscr{A}_0 \models \phi_0 \cdots \mathscr{A}_n \models \phi_n \qquad \phi = \phi_0 \natural_{a_0} \cdots \natural_{a_{n-1}} \phi_n}{\mathscr{A}_0 \uparrow_{a_0} \cdots \uparrow_{a_{n-1}} \mathscr{A}_n \models \phi}$$

Proof. We prove Proposition 2 by induction on n.

- The base step where "$n = 1$" is proved by Theorem 1.
- For the inductive step, first, we assume:

$$\frac{\mathscr{A}_0 \models \phi_0 \cdots \mathscr{A}_n \models \phi_n \qquad \phi = \phi_0 \natural_{a_0} \cdots \natural_{a_{n-1}} \phi_n}{\mathscr{A}_0 \uparrow_{a_0} \cdots \uparrow_{a_{n-1}} \mathscr{A}_n \models \phi}$$

  Let $\mathscr{A}' = \mathscr{A}_0 \uparrow_{a_0} \cdots \uparrow_{a_{n-1}} \mathscr{A}_n$ and $\phi' = \phi_0 \natural_{a_0} \cdots \natural_{a_{n-1}} \phi_n$. While $\natural$ and $\uparrow$ are associative operators, then: $\mathscr{A} = \mathscr{A}' \uparrow_{a_n} \mathscr{A}_{n+1}$ and $\phi = \phi' \natural_{a_n} \phi_{n+1}$. By assuming $\mathscr{A}_n \models \phi_n$ and applying Theorem 1, then:

$$\frac{\mathscr{A}' \models \phi' \qquad \mathscr{A}_{n+1} \models \phi_{n+1} \qquad \mathscr{A} = \mathscr{A}' \uparrow_{a_n} \mathscr{A}_{n+1} \qquad \phi = \phi' \natural_{a_n} \phi_{n+1}}{\mathscr{A} \models \phi}$$

5.2 The Encoding to PRISM

To encode a SysML activity diagram $\mathscr{A}$ into its equivalent PRISM code $\mathscr{P}$, we rely to the PRISM MDP formalism that refers to the PA[2] which coincides with the NuAC semantics. In PRISM, we define the NuAC transition $s \xrightarrow{l} \mu$ as a probabilistic command. Mainly, the probabilistic command takes the following form: $[l]\ g \rightarrow p_1 : u_1 + ... + p_m : u_m$, which means, for the action "l" if the guard "g" is true, then, an update "u_i" is enabled with a probability "p_i". The guard "g" is a predicate of a conjunction form consisting to the evaluation of the atomic propositions related to the state s. The update u_i

[2] http://www.prismmodelchecker.org/doc/manual.pdf, (The introduction section, line 10).

describes the evaluation of the atomic propositions related to the next state s_i of s such that $s \xrightarrow{l}_{p_i} s_i$ $(1 \leq i \leq m)$. For the Dirac case, the command is written simply by: $[l]$ $g \rightarrow u$.

The function Γ presented in Listing 1.1 produces the appropriate PRISM command for each NuAC term. The action label of a command is the label of its related term "l". The guard of this command depends on how the term is activated, therefore, a boolean proposition as a flag is assigned to define this activation. For simplicity, the flag related to a term labeled by l is denoted by a boolean proposition l that is initialized to false except for the initial node it is true which conforms to the premise of the NuAC rule "INIT-1". Concerning the command updates, they deactivate the propositions of a term $n \in \mathscr{A}$ and activate its successors. We define three useful functions: $L(n)$, $S(\mathscr{A}_i)$, and $E(\mathscr{A}_i)$ that return the label of a term n, the initial and the final terms of the diagram $\mathscr{A}_i$, respectively. For example, the call behavior action "$l : a \uparrow \mathscr{A}_i$" (line 32) produces two commands (line 34), and it calls the function Γ' (line 34). The first command in line 34 synchronizes with the first command in line 52 produced by the function Γ' in the action l from the diagram $\mathscr{A}$. Similarly, the second command in line 34 synchronizes with the command of line 56 in the action $L(E(\mathscr{A}_i))$ from the diagram $\mathscr{A}_i$. The first synchronization represents the NuAC rule BH-1 where the second represents the rule BH-2. The function Γ' is similar to the function Γ except for the initial and the final nodes as shown in lines 52 and 56, respectively. The generated PRISM fragment of each diagram $\mathscr{A}_i$ is bounded by two PRISM primitives: the module head "*Module* $\mathscr{A}_i$", and the module termination "*endmodule*".

```
1  Γ: 𝒜 → 𝒫
2  Γ(𝒜) = ∀n∈𝒜, L(n=ι)=⊤, L(n≠ι)=⊥, Case (n) of
3    l: ι↦𝒩 ⇒ in {[l]l ⟶ (l'=⊥)&(L(𝒩)'=⊤);}∪Γ(𝒩) end
4    l: M(x,y)↦𝒩 ⇒ in {[l_x]l_x ⟶ (l'_x=⊥)&(L(𝒩)'=⊤);}
5                 ∪{[l_y]l_y ⟶ (l'_y=⊥)&(L(𝒩)'=⊤);}∪Γ(𝒩) end
6    l: J(x,y)↦𝒩 ⇒ in {[l]l_x∧l_y ⟶ (l'_x=⊥)&(l'_y=⊥)&(L(𝒩)'=⊤);}∪Γ(𝒩) end
7    l: F(𝒩1,𝒩2) ⇒ in {[l]l ⟶ (l'=⊥)&(L(𝒩1)'=⊤)&(L(𝒩2)'=⊤);}∪Γ(𝒩1)∪Γ(𝒩2) end
8    l: D(𝒜,p,g,𝒩1,𝒩2) ⇒
9    Case (p) of ]0,1[ ⇒
10           in {[l]l ⟶ p:(l'=⊥)&(l'_g=⊤)+(1−p):(l'=⊥)&(l'_¬g=⊤);}
11            ∪{[l_¬g]l_g∧¬g ⟶ (l'_¬g=⊥)&(L(𝒩2)'=⊤);}
12            ∪{[l_g]l_g∧g ⟶ (l'_g=⊥)&(L(𝒩1)'=⊤);}∪Γ(𝒩1)∪Γ(𝒩2)end
13     Otherwise in {[l]l ⟶ (l'=⊥)&(l'_g=⊤);}∪{[l]l ⟶ (l'=⊥)&(l'_¬g=⊤);}
14           ∪{[l_g]l_g∧g ⟶ (l'_g=⊥)&(L(𝒩1)'=⊤);}
15           ∪{[l_¬g]l_g∧¬g ⟶ (l'_¬g=⊥)&(L(𝒩2)'=⊤);}
16           ∪Γ(𝒩1)∪Γ(𝒩2)end
17   l: aℬ↦𝒩, Case (ℬ) of
18        ↑𝒜_i ⇒
19          in {[l]l → (l'=⊥);}
20           ∪{[L(E(𝒜_i))]L(E(𝒜_i)) → (l'=⊥)&(L(𝒩)'=⊤);}∪Γ'(𝒜_i); end
21          ε ⇒ in {[l]l ⟶ (l'=⊥)&(𝒩'=⊤);}∪Γ(𝒩') end
22   l: ⊗⇒ in [l]l ⟶ (l'=⊥); end
23   l: ⊙⇒ in [l]l ⟶ &_{l∈𝒮}(l'=⊥);end
24 //Defining the function Γ'(a↑𝒜_i)
25 Γ': 𝒜 → 𝒫
26 Γ'(𝒜_i) = ∀m∈𝒜_i: L(m)=⊥, Case (m) of
27   l: ι↦𝒩 ⇒//The action l and the guard l are from the line 40.
28          in {[l]l → (L(S(𝒜_i))'=⊤);
29          [L(S(𝒜_i))]L(S(𝒜_i)) → (L(S(𝒜_i))'=⊥)&(L(𝒩)'=⊤);}∪Γ(𝒩) end
30   l: ⊙⇒ in [L(E(𝒜_i))]L(E(𝒜_i)) → (L(E(𝒜_i))'=⊥); end
31   Otherwise Γ(𝒜);
```

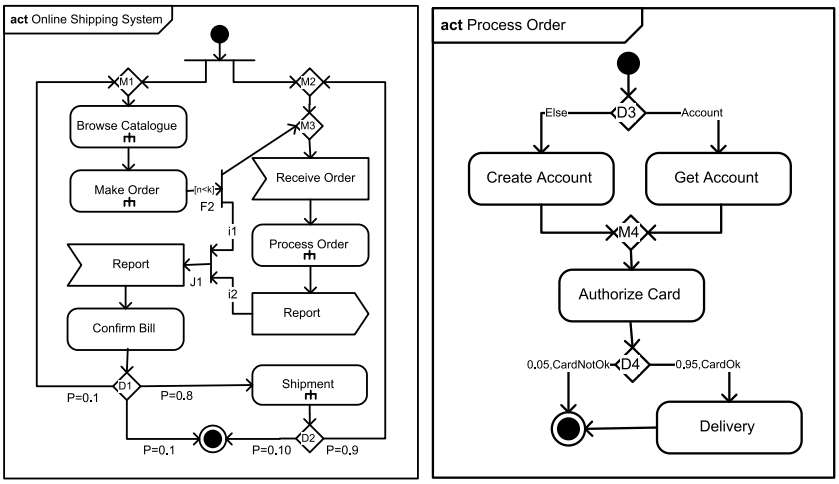
Listing 1.1: Generating PRISM Commands Function.

6 Implementation and Experimental Results

For the purpose of providing experimental results demonstrating the efficiency and the validity of our framework, we verify a set of PCTL properties on the online shopping system [13] and the automated teller machine [13]. To this end, we compare the verification results "β", the verification cost in terms of the model size[3] "γ", and the verification time "δ" (sec) with and without applying our approach.

6.1 Online Shopping System

The online shopping system aims at providing services for purchasing online items. Fig. 5a illustrates the corresponding SysML activity diagram. It contains four call-behavior actions[4], which are: "Browse Catalogue", "Make Order", "Process Order", and "Shipment" denoted by a, b, c and d, respectively. For simplicity, we take this order to denote their called diagrams by $\mathscr{A}_1$ to $\mathscr{A}_4$, respectively, where $\mathscr{A}_0$ denotes the main diagram. As an example, Fig. 5b expands the diagram related to the call behavior action "Process Order" and it is denoted by $\mathscr{A}_3$. The whole diagram is written by: $\mathscr{A} = \mathscr{A}_0 \uparrow_a \mathscr{A}_1 \uparrow_b \mathscr{A}_2 \uparrow_c \mathscr{A}_3 \uparrow_d \mathscr{A}_4$. Here, we propose to verify the properties Φ_1 and Φ_2 that are expressed in PCTL.



(a) Online Shopping System. (b) Process Order.

Fig. 5: SysML Activity Diagrams.

Property Φ_1. "For each order, what is the minimum probability value to make a delivery after browsing the catalogue?"
PCTL: $Pmin = ?[(Browse\ Catalogue) \Rightarrow (F(Delivery))]$.

[3] The model size is the number of transitions (edges).
[4] Each call-behavior action is represented by its proper diagram.

In this expression, the "Browse Catalogue" proposition is part of $\mathscr{A}_0$ and "*Delivery*" is a proposition of $\mathscr{A}_3$. For comparison, we verify first Φ_1 on $\mathscr{A}$. Then, by using Proposition 1, we reduce the verification of Φ_1 from $\mathscr{A}$ to $\mathscr{A}_0 \uparrow_c \mathscr{A}_3$. And, by using the decomposition rules of Definition 5, Φ_1 is decomposed into two properties: Φ_{11} and Φ_{12} such that: $\Phi_{11} \triangleq Pmin =?[(Browse\ Catalogue) \Rightarrow (F(Process\ Order))]$, and $\Phi_{12} \triangleq Pmin =?[(True) \Rightarrow (F(Delivery))]$. After the verification of Φ_1 on $\mathscr{A}$, Φ_{11} on $\mathscr{A}_0$ and Φ_{12} on $\mathscr{A}_3$, Table 3 summarizes the verification results and costs for different values of the number of orders "n". From the obtained results, we observe that the probability values are preserved where $\beta_1=\beta_{11} \times \beta_{12}$. In addition, the size of the diagrams is minimized $\gamma_{11}+\gamma_{12} < \gamma_1$. Consequently, the verification time is reduced significantly $\delta_{11}+\delta_{12} \ll \delta_1$.

n	3	4	5	6	7	8	9	10
β_1	0.76	0.76	0.76	0.76	0.76	0.76	0.76	0.76
γ_1	2,213,880	4,823,290	8,434,700	13,048,110	51,145,160	202,489,260	454,033,360	805,777,460
δ_1	10.764	24.364	44.098	72.173	358.558	1818.247	6297.234	17761.636
β_{11}	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
γ_{11}	5,486	7,266	9,046	10,826	12,606	14,386	16,166	17,946
δ_{11}	1.09	3.12	7.511	12.86	27.03	54.38	111.74	163.89
β_{12}	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
γ_{12}	12	12	12	12	12	12	12	12
δ_{12}	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005

Table 3: The Verification Cost for Properties Φ_1 Φ_{11}, and Φ_{12}.

Property Φ_2. "For each order, what is the maximum probability value to confirm a shipment?"
PCTL: $Pmax =?[G((CreateDelivery) \Rightarrow F(ConfirmShipment)]$.
The propositions of this property "CreateDelivery" and "ConfirmShipment" belong to $\mathscr{A}_2$, and $\mathscr{A}_4$, respectively. Similarly to the verification of Φ_1, we verify Φ_2 on $\mathscr{A}$. Then, we decompose Φ_2 to Φ_{21} and Φ_{22} with respect to $\mathscr{A}_0 \uparrow_b \mathscr{A}_2 \uparrow_d \mathscr{A}_4$. The PCTL expressions of the decomposition are: $\Phi_{21} \triangleq Pmax =?[G((CreateDelivery) \Rightarrow F(Shipment)]$, and $\Phi_{22} \triangleq Pmax =?[G((True) \Rightarrow F(ConfirmShipment)]$. Table 4 shows the verification results and costs of Φ_2 on $\mathscr{A}$, Φ_{21} on $\mathscr{A}_0 \uparrow_b \mathscr{A}_2$, and Φ_{22} on $\mathscr{A}_4$ for different values of the number of orders "n". We found: $\beta_2=\beta_{21} \times \beta_{22}$, $\gamma_{21}+\gamma_{22} < \gamma_2$ and $\delta_{21}+\delta_{22} \ll \delta_2$.

6.2 Automated Teller Machine

The Automated Teller Machine (ATM) is a system that interacts with a potential customer via a specific interface and communicates with the bank over an appropriate communication protocol. Fig. 6 represents the ATM SysML activity diagram ($\mathscr{A}'$) composed of the main diagram ($\mathscr{A}'_0$) "Figure 6-(a)" and three called diagrams: (a') Check Card ($\mathscr{A}'_1$)[5], (b') Authorize ($\mathscr{A}'_2$), and (c') Transaction ($\mathscr{A}'_3$) that is showed in Fig. 6-(b).

[5] The call behavior action "Check Card" is denoted by a' and calls the diagram $\mathscr{A}'_1$.

n	3	4	5	6	7	8	9	10
β_2	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377
γ_2	2,213,880	4,823,290	8,434,700	13,048,110	51,145,160	202,489,260	454033360	805,777,460
δ_2	33.394	78.746	168.649	354.211	2280.252	17588.755	34290.635	63097.014
β_{21}	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377
γ_{21}	9614	12017	14420	16823	19226	21629	24032	26435
δ_{21}	4.775	12.301	32.852	83.337	274.9	450.81	586.43	652.76
β_{22}	1	1	1	1	1	1	1	1
γ_{22}	9	9	9	9	9	9	9	9
δ_{22}	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003

Table 4: The Verification Cost for Properties Φ_2 Φ_{21}, and Φ_{22}.

Our goal is to measure the satisfiability probability of the PCTL properties Φ_3 and Φ_4 on $\mathscr{A}' = \mathscr{A}_0' \uparrow_{a'} \mathscr{A}_1' \uparrow_{b'} \mathscr{A}_2' \uparrow_{c'} \mathscr{A}_3'$.

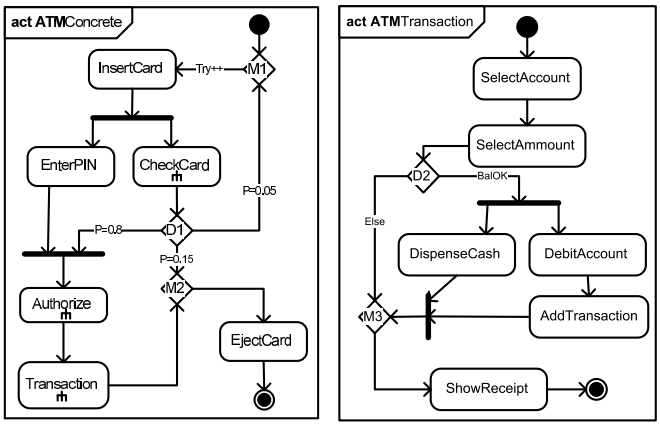
Property Φ_3. "What is the minimum probability of authorizing a transaction after inserting a card". PCTL: $Pmin=?[G(InstertCard \Rightarrow F(DebitAccount))]$.
After verifying Φ_3 on $\mathscr{A}'$, we verify Φ_{31} on $\mathscr{A}_0'$ and Φ_{32} on $\mathscr{A}_3'$ such that : $\Phi_{31} \triangleq Pmin=?[G(InstertCard) \Rightarrow (F(Transaction))]$ and : $\Phi_{32} \triangleq Pmin=?[G((True) \Rightarrow F(DebitAccount))]$. As a result we found the following: $\beta_3 = 0.8421$, $\gamma_3 = 606470$, $\delta_3 = 3.12$, $\beta_{31} = 0.8421$, $\gamma_{31} = 3706$, and $\delta_{31} = 0.64$, $\beta_{32} = 1$, $\gamma_{32} = 15$, and $\delta_{32} = 0.007$. From the obtained results, we found that the satisfiability probability is maintained β_3=$\beta_{31} \times \beta_{32}$, with a considerable verification costs $\gamma_{31} + \gamma_{32} < \gamma_3$ and δ_{31}+$\delta_{32} \ll \delta_3$.

Property Φ_4. "What is the maximum probability of inserting a card when it is not valid." PCTL: $Pmax=?[(CardNotValid) \Rightarrow (F(InsertCard))]$.
Similarly to the verification of Φ_3, instead of verifying Φ_4 on $\mathscr{A}'$ we verify Φ_{41} on $\mathscr{A}_1'$ and Φ_{42} on $\mathscr{A}_0'$ such that :
$\Phi_{41} \triangleq Pmax=?[(CardNotValid) \Rightarrow (F(EndCheckCard))]$, and
$\Phi_{42} \triangleq Pmax=?[(CheckCard) \Rightarrow (F(InsertCard))]$.

After verification, we found the following: $\beta_4 = 0.05$, $\gamma_4 = 606470$, $\delta_4 = 11.458$, $\beta_{41} = 1$, $\gamma_{41} = 11$, and $\delta_{41} = 0.004$, $\beta_{42} = 0.05$, $\gamma_{42} = 7211$, and $\delta_{42} = 1.584$. From these results, we have: β_4=$\beta_{41} \times \beta_{42}$, γ_{41}+$\gamma_{42} < \gamma_4$ and δ_{41}+$\delta_{42} \ll \delta_4$.

7 Conclusion

In this paper, we presented a compositional verification framework to improve the efficiency of probabilistic model-checking. More specifically, our target was verifying systems modeled using SysML activity diagrams composed by the call behavior interfaces. We improved their verification cost by introducing a probabilistic compositional verification approach based on decomposing a global PCTL property into local ones with respect to interfaces between diagrams. Moreover, the presented framework can ignore the called diagrams that are irrelevant to a given PCTL property. For verification, we proposed an algorithm to encode the composed diagrams into PRISM input language. Furthermore, we proposed a semantic for SysML activity diagrams that helps



(a) ATM. (b) Transaction.

Fig. 6: ATM SysML Activity Diagram.

on proofs and to encode easily the diagrams in PRISM. We proved the soundness of the proposed framework by showing the satisfiability preservation of PCTL properties. In addition, we demonstrated the effectiveness of our framework by verifying real systems that are not symmetric, which mean, we can not benefit from the symmetry reduction built within the PRISM model checker. In future, we would like to extend our work by investigating several directions. First, we plan to extend our framework to handle more compositional verification techniques like assume-guaranty and integrate them within the PRISM implementation. Then, we explore more system features such as time and object. Finally, we intend to apply our framework on a large systems' applications.

References

1. Takahiro Ando, Hirokazu Yatsu, Weiqiang Kong, Kenji Hisazumi, and Akira Fukuda. Formalization and model checking of sysml state machine diagrams by csp#. In *Computational Science and Its Applications ICCSA 2013*, volume 7973 of *LNCS*, pages 114–127. Springer Berlin Heidelberg, 2013.
2. Ermeson Andrade, Paulo Maciel, Gustavo Callou, and Bruno Nogueira. A Methodology for Mapping SysML Activity Diagram to Time Petri Net for Requirement Validation of Embedded Real-Time Systems with Energy Constraints. In *ICDS '09: Proc.of the 2009 Third Int. Conf. on Dig. Soc.*, pages 266–271, Washington, DC, USA, 2009. IEEE Computer Society.
3. Ludovic Apvrille and Pierre Saqui-Sannes. Static analysis techniques to verify mutual exclusion situations within sysml models. In Ferhat Khendek, Maria Toeroe, Abdelouahed Gherbi, and Rick Reed, editors, *SDL 2013: Model-Driven Dependability Engineering*, volume 7916 of *LNCS*, pages 91–106. Springer Berlin Heidelberg, 2013.
4. Christel Baier and Joost Pieter Katoen. *Principles of Model Checking*. The MIT Press, may 2008.
5. B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification.* Springer, 2001.

6. Sergey Berezin, Sérgio Vale Aguiar Campos, and Edmund M. Clarke. Compositional reasoning in model checking. In *Int. Symp. on Compositionality: The Significant Difference*, COMPOS'97, pages 81–102, 1998.
7. Ermeson Carneiro, Paulo Maciel, Gustavo Callou, Eduardo Tavares, and Bruno Nogueira. Mapping SysML State Machine Diagram to Time Petri Net for Analysis and Verification of Embedded Real-Time Systems with Energy Constraints. In *ENICS '08: Proc.of the 2008 Int.Conf.on Adv.in Elec.and Micro-elec.*, pages 1–6, Washington, DC, USA, 2008. IEEE Computer Society.
8. Oscar Carrillo, Samir Chouali, and Hassan Mountassir. Formalizing and verifying compatibility and consistency of sysml blocks. *SIGSOFT Softw. Eng. Notes*, 37(4):1–8, 2012.
9. Lu Feng, Tingting Han, Marta Kwiatkowska, and David Parker. Learning-based compositional verification for synchronous probabilistic systems. In *Proc. of the 9th int. conf. on Aut. tech. for verif. and analy.*, ATVA'11, pages 511–521. Springer-Verlag, 2011.
10. Lu Feng, Marta Kwiatkowska, and David Parker. Compositional verification of probabilistic systems using learning. In *Proceedings of the 2010 Seventh Int. Conf. on the Quant. Eval. of Sys.*, QEST '10, pages 133–142. IEEE Computer Society, 2010.
11. Lu Feng, Marta Kwiatkowska, and David Parker. Automated learning of probabilistic assumptions for compositional reasoning. In *Proc. of the 14th int. conf. on Fund. approaches to software engineering*, FASE'11/ETAPS'11, pages 2–17, Berlin, Heidelberg, 2011. Springer-Verlag.
12. V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated Verification Techniques for Probabilistic Systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, LNCS, pages 53–113. Springer, 2011.
13. H. Gomaa. *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*. Cambridge University Press, 2011.
14. Object Management Group. *OMG Unified Modeling Language: Superstructure 2.1.2*, Nov. 2007.
15. J. Holt and S. Perry. *SysML for Systems Engineering*. Institution of Engineering and Technology Press, January 2007.
16. Xiaopu Huang, Qingqing Sun, Jiangwei Li, and Tian Zhang. MDE-Based Verification of SysML State Machine Diagram by UPPAAL. In Yuyu Yuan, Xu Wu, and Yueming Lu, editors, *Trustworthy Computing and Services*, volume 320 of *Communications in Computer and Information Science*, pages 490–497. Springer Berlin Heidelberg, 2013.
17. David N. Jansen, Holger Hermanns, and Joost Pieter Katoen. A Probabilistic Extension of UML Statecharts - Specification and Verification. In *In Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT), LNCS 2469: 355374*, pages 76–91. Springer, 2002.
18. David N. Jansen, Holger Hermanns, and Joost Pieter Katoen. A QoS-Oriented Extension of UML Statecharts. *LNCS*, 2863:76–91, 2003.
19. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV*, LNCS, pages 585–591. Springer, 2011.
20. Object Management Group. *OMG Systems Modeling Language Specification*, Sep. 2007.
21. S. Ouchani, O. A. Mohamed, and M. Debbabi. A security risk assessment framework for sysml activity diagrams. In *2013 IEEE 7th International Conference on Software Security and Reliability*, pages 227–236, June 2013.
22. Samir Ouchani. Towards a fractionation-based verification: Application on sysml activity diagrams. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, pages 2032–2039. ACM, 2019.
23. Samir Ouchani and Mourad Debbabi. Specification, verification, and quantification of security in model-based systems. *Computing*, 97(7):691–711, 2015.
24. Samir Ouchani, Otmane Aït Mohamed, and Mourad Debbabi. Efficient probabilistic abstraction for sysml activity diagrams. In *SEFM*, pages 263–277, 2012.