On the Probabilistic Verification of Time Constrained SysML State Machines

Abdelhakim Baouya[1](✉), Djamal Bennouar[2], Otmane Ait Mohamed[3], and Samir Ouchani[4]

[1] CS Department, Saad Dahlab University, Blida, Algeria
baouya.abdelhakim@gmail.com
[2] CS Department, University of Bouira, Bouria, Algeria
dbennouar@gmail.com
[3] ECE Department, Concordia University, Montreal, Canada
otmane.aitmohamed@concordia.ca
[4] SnT Center, University of Luxembourg, Walferdange, Luxembourg
samir_ouchani@yahoo.com

Abstract. Software and hardware design of complex systems is becoming difficult to maintain and more time and effort are spent on verification than on construction. One of the reason is the number of constraints that must be hold by the system. Recently, Formal methods such as probabilistic approaches gain a great importance in real-time systems verification including avionic systems and industrial process controllers. In this paper, we propose a probabilistic verification framework of SysML state machine diagrams extended with time and probability features. The approach consists of mapping a SysML state machine diagrams to PRISM input language. To ensure the correctness of proposed approach, we capture the semantics of both SysML state machine diagrams and their generated PRISM code. We demonstrate the approach efficiency by analyzing PCTL temporal logic on ATM case study.

Keywords: Sysml state machine diagram · MARTE · Probability · Time

1 Introduction

Constraints on system design in terms of functionality, performance, availability, reliability and time to market are becoming more stringent. Therefore, the design and implementation of successful systems, represents the prime concerns of systems engineering (SE) but reveals several challenges [9]. Indeed, from one side the systems are becoming increasingly complex, in the other side the market pressure for rapid development of these systems makes the task of their designs a challenge. Thus, the evaluation and the correctness of systems at early stage of design reduces the design cost such as maintenance time and effort. Recently, the need of automated verification techniques to cope with errors is imminent, especially when *time* and *probability* are incorporated.

The *probabilistic verification* is used to verify systems whose behavior is unpredictable, unreliable, especially stochastic in nature. The verification of such systems can be focused on either qualitative or quantitative properties [4]. Quantitative properties puts the constraints on a certain event, e.g. the probability of processor failure in the next 3 hours is at a least 0.88, while qualitative properties assert that certain event will happen surely (i.e. Probability=1).

In this paper, we are interested in the formal verification of probabilistic systems under time constraints modeled as SysML state machine diagram extended with probability and time features of MARTE profile [13]. The overview of our framework is depicted in Fig. 1. It takes State machine diagrams and PCTL properties as input. Our approach is based on representing state machine diagram to an equivalent PRISM model (*Probabilistic Timed Automata*). The PRISM model checker verifies PCTL properties on the resulting model. We extract the adequate semantics model related to state machine diagram then, we present the underling semantics related to the produced PRISM model. Furthermore, we show tht the relation between both semantic preserves the satisfiability of PCTL properties.

The remainder of this paper is structured as follows: Sect. 2 discusses the related work. Section 3 describes SysML state machine diagram. Sections 4 and 5 provide syntax and semantic meaning of probabilistic and timed state machine diagrams. The syntax and semantics of PRISM Model Checker is presented in Sect. 6. Section 7 provides a mapping mechanism from state machine diagram into the input language of the probabilistic model checker PRISM. The approach soundness is proved in Sect. 8. Section 9 illustrates the application of our mapping rules on Automatic Teller Machine (ATM) case study. Section 10 draws conclusions and lays out the future works.

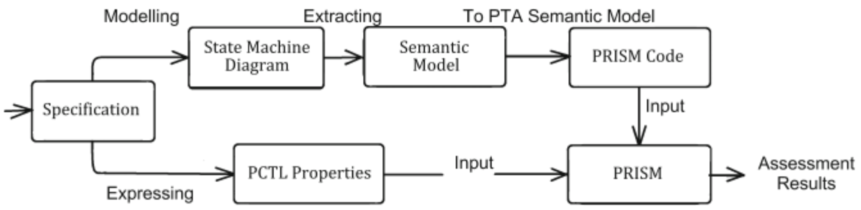


Fig. 1. A SysML State machine diagram verification approach

2 Related Work

In this section, we present the recent works related to the verification of behavioral models then we compare them with our proposed approach.

Doligalski and Adamski [8] propose a verification and simulation of UML State Machine. For this purpose, two mapping mechanisms are defined. The first consists on mapping the original model to Petri network for verification according the requirements. However, probability and time verification are not considered. When the requirements are satisfied, the second mapping occurs to generate VHDL or Verilog description for simulation. Huang et al. [11] propose a verification of SysML State Machine Diagram by extending the model with MARTE [13] features to express the execution time. The tool has as input the State Machine Diagram and as output timed automata expressed in UPPAAL syntax [5]. UPPAAL uses Computational Tree Logic (CTL) properties to check if the model is satisfied with liveness and safety properties. Ouchani et al. [17] propose a verification framework of SysML activity diagram. The authors address a subset of SysML activity diagram artifacts with control flow. The different artifacts have been formalized and mapping algorithm has been proposed to translate these artifacts to PRISM input language. The transformation result is a probabilistic automata to be checked by PRISM. Timing verification is not considered. Kaliappan et al. [12] propose a verification approach for system workflow especially in communication protocol. The approach takes as input three UML diagrams: state machine diagram, activity diagram and sequence diagram. State machine diagram or activity diagram is converted into PROMELA code as a protocol model and its properties are derived from the sequence diagram as Linear Temporal Logic (LTL). Pajic et al. [18] develop a framework for verification and generation of real time applications either in C/C++code for software or in Hardware description language (HDL) like VHDL or Verilog. The focus of the work is a development of model translation tool from UPPAAL [5] to Stateflow (UPP2SF). The checked UPPAAL model is translated to Stateflow using Simulink which provides full support for C/C++ and HDLs. Ando et al. [3] propose a verification approach of SysML state machine diagram. The diagrams are translated to communication sequential process description (CSP) and they apply the PAT [20] model checker to check the CSP models against the LTL properties. The paper proposes a mapping rules of different state machine artifacts. However, time and probability are not addressed.

Compared to the existing works Table 1, our contribution improves the verification of SysML State Machine diagram by extending state machine with elements of UML MARTE profile to support time and probability. From the comparison, we observe that few of them formalize the behavioral model and prove the soundness of their proposed verification approaches. Moreover, our verification framework is efficient as it preserves all properties.

Table 1. Comparison with the related work.

Approach	Formalization	Probability	Time	Soundness	Automation
[8], [3], [12]					√
[11], [18]			√		√
[17]	√	√		√	√
Our	√	√	√	√	√

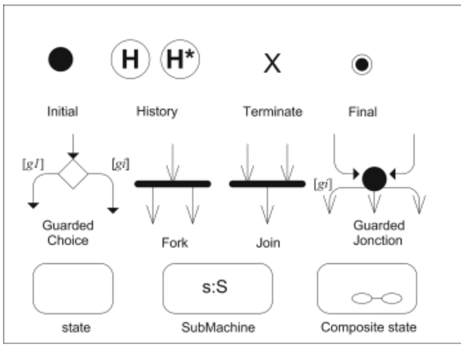


Fig. 2. A subset of State machine diagram artifacts

3 SysML State Machine Diagram

SysML State Machine diagram (SMD) is a graph-based diagram where states nodes are connected by states edges (i.e.transition)[1]. Figure 2 shows the set of interesting artifacts used for verification in this paper. The behavior of a state machine is specified by a set of regions, each of which defines its own set of states. The states in any one region are exclusive; that is, when the region is active, exactly one of its substates is active. A region starts (resp. stops) executing when it initial (resp. final) pseudo-state becomes active. When a state is entered, an (optional) *entry* behavior is executed. Similarly on exit, an optional *exit* behavior is executed. While in a state, a state machine can execute a *do* behavior. Transitions are defined by triggers, guards, and effects. The trigger cause a transition from the source state when the guard is valid, and the effect is a behavior executed once the transition is triggered (opac behavior). In addition, the control nodes supports a junction, choice, join, fork, terminate and history pseudo-state node. A junction splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice pseudo-state which realizes a dynamic conditional branch. To illustrate how a probability value is specified, the transition leaving choice nodes are annotated with the $\ll GaStep \gg$ stereotype using the element *prob* of MARTE profile [13]. The time is specified by applying the stereotype $\ll resourceUsage \gg$ with element $execTime$ to specify the maximum and the minimum value of the time duration written as (value, unit, min/max), where *min*, *max* are integer values. We present in Definition 1, the formal definition of Probabilistic and timed SMD. Then, we propose a property that explains the state transition in SMD.

Definition 1. Probabilistic and timed SysML state machine diagrams is a tuple $S = (i, fin, \mathscr{N}$,X, E, Inv, Enab, Prob), where:

- i is the initial node,
- $fin = \{\odot, \times\}$ is the set final nodes,
- $\mathscr{N}$ is a finite set of state machine nodes,

- X is a set of clocks,
- E is a set of events,
- $Inv : \mathscr{N} \rightarrow \mathbb{N}$ is the invariant constraint that represents the maximum clock value supported by state clock,
- Enab: $\mathscr{N} \rightarrow \mathbb{N}$ is an enabling condition that represent the minimum clock value for state transition,
- Prob : $(\{i\} \cup \mathscr{N}) \times E \rightarrow Dist(\mathscr{N} \times 2^X)$ is a probabilistic transition function that assigns for each state $s \in \mathscr{N}$ and $\alpha \in E$ a discrete probability distribution $\mu \in Dist(2^x \times \ \mathscr{N})$.

Property 1. There are two possible ways in which a SMD can proceed by taking a transition (State transition) or by letting time progress while remaining in a state (Delay transition):

- State transition : for $s, s', \in \mathscr{N}\ , \alpha \in E\ s \overset{\alpha,t}{\rightarrow}_p s'$ when $Enab(s) \leq t \leq Inv(s)$.
- Delay transition : for $s \in \mathscr{N}\ , \alpha \in E\ s \overset{\alpha,t}{\rightarrow} s$ when $t \leq Inv(s)$.

$$\boxed{\begin{aligned}
&\mathcal{S} ::= \epsilon \mid l : \bar{i}^{n} \rightarrowtail \mathcal{N}\\
&\mathcal{N} ::= \overline{\mathcal{N}} \mid l : F(\mathcal{N},\mathcal{N}) \mid l : D(p,g,\mathcal{N},\mathcal{N}) \mid \overline{\mathcal{O}}^{n} \rightarrowtail \mathcal{N} \mid \mathcal{H}(\mathcal{S}) \mid l : \odot \mid l : \times\\
&\mathcal{O} ::= s\mathcal{B} \mid s(S_{entry}, S_{sub}, S_{exit}) \mid s(S_{entry}, S_{do}, S_{exit}) \mid \mathcal{J}(x_1, x_2) \mid M(x_1, g_1, \mathcal{N})\\
&\mathcal{B} ::= \uparrow \mathcal{S} \mid \epsilon
\end{aligned}}$$

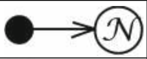
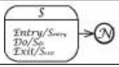
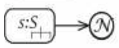
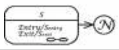
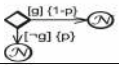
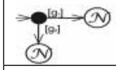
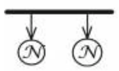
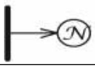
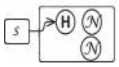
Fig. 3. Syntax of State Machine Calculus (SMC).

4 Syntax

Based on the SysML textual specification standard [1], we formalize SysML state machine diagrams by developing a calculus called *State Machine Calculus* (SMC) which is proposed in Fig. 3 that offers more flexibility than the graphical notation defined in the standard. In Table 2, each state machine diagram artifact is represented formally by its related SMC term. In SMC syntax, two main syntactic concepts are defined: marked and unmarked terms. A marked term is typically used to denote a reachable configuration. A configuration is characterized by the set of tokens locations in a given term. An unmarked SMC term corresponds to the static structure of the diagram.

To support tokens we augment the "Over bar" operator with integer value n such that the $\overline{\mathcal{N}}^{n}$ denotes the term $\mathcal{N}$ marked with n tokens. Furthermore, we use a prefix label l : for each node to uniquely reference it in the case of a backward flow connection. Particularly, labels are useful for connecting multiple incoming flows towards junction and join nodes. Let $\mathcal{L}$ be a collection of labels ranged over by $l; l_0; l_1,..$ and $\mathcal{N}$ be any node (except initial) in the SMD. We write l :$\mathcal{N}$ to denote a l-labeled state $\mathcal{N}$. It is important to note that nodes with multiple incoming edges (e.g. join and junction) are visited as many times as

Table 2. Formal Notation of SysML state Machine Artifacts

Artifact	Formal notation	Description
	$l : i \rightarrowtail \mathcal{N}$	Initial node is activated when a diagram is invoked.
◉	$l : \odot$	Final node stops the diagram execution
×	$l : \times$	Terminate node kills only the related token
	$l : s(S_{entry}, S_{do}, S_{exit}) \rightarrowtail \mathcal{N}$	State with entry/do/exit behavior defines a state represented by a sequence of events
	$l : s \uparrow S \rightarrowtail \mathcal{N}$	Call behavior state invokes a new behavior related to the state.
	$l : s(S_{entry}, S_{sub}, S_{exit}) \rightarrowtail \mathcal{N}$	Composite State calls the set of substates that can be parallel
	$l : \mathcal{D}(p, g, \mathcal{N}, \mathcal{N})$	Choice node with convex distribution p, 1-p and guarded edges g, ¬ g
	$l : \mathcal{M}(x, g_1, \mathcal{N}_1, y, g_2, \mathcal{N}_2)$	Junction node with static conditional branch g1, g2
	$l : \mathcal{F}(\mathcal{N}_1, \mathcal{N}_2)$	When an incoming transition is taken to the fork pseudo-state, all the outgoing transitions are taken
	$l : \mathcal{J}(x) \rightarrowtail \mathcal{N}$	Join node presents the synchronization, And x is the set of input pins x= {xl, x2}.
	$l : \mathcal{H}(\mathcal{N}^*, \mathcal{N})$	History pseudo-states allow a state to be interrupted and then subsequently resume its previously active state or states.

they have incoming edges. Thus, as a syntactic convention we use only a label (i.e. l) to express a SMC term if its related node is encountered already. We denote by $D(g, \mathcal{N}, \mathcal{N})$ and $D(p, g, \mathcal{N}, \mathcal{N})$ to express a non-probabilistic and a probabilistic choices, respectively.

5 Semantics

For the workflow observation on SMD, we use structural operational semantics [14] and [15] to formally describe how the computation steps of SMC atomic terms take place. An element α is the label of the event triggering state transition, x(y) inputs an object name on x and stores it in y to represent the effects of transition and τ represents a silent event. An element **t** is the time for state transition and p is a probability value such that $p \in]0, 1[$. The general form of a transition is $S \xrightarrow{t, \alpha/b(y)}_p S'$. The probability value specifies the likelihood of a given transition to occur and it is denoted by $P(S, t, \alpha, S')$ where $min \leq t \leq max$ $(max, min \in \mathbb{N})$. The case of p = 1 presents a non-probabilistic transition and it is denoted simply by $S \xrightarrow{t, \alpha} S'$. For simplicity, we denote by $S[\mathcal{N}]$ to specify $\mathcal{N}$ as a sub-term of S and by $|S|$ to denote a term S without tokens. For the call behavior case of $s \uparrow \mathcal{N}$, we denote S$[s \uparrow \mathcal{N}]$ by $S \uparrow_s \mathcal{N}$ and "*" is used to refers to the recent active substate in the state in case of shallow history. In the sequel, we describe the operational semantic rules of the SMC calculus.

Ax-1 $l : i \rightarrowtail \mathcal{N} \xrightarrow{l}_1 l : \overline{i} \rightarrowtail \mathcal{N}$. This axiom introduces the execution of $\mathcal{S}$ by putting a token on i.

Ax-2 $l : \overline{i} \rightarrowtail \mathcal{N} \xrightarrow{l}_1 l : i \rightarrowtail \overline{\mathcal{N}}$. This axiom propagates the token in the marked term i into its outgoing $\mathcal{N}$.

Ax-3 $\forall n > 0, m \geq 0\ l : \overline{\overline{s^m} \rightarrowtail \mathcal{N}}^n \xrightarrow{l}_1 l : \overline{\overline{s^{m+1}} \rightarrowtail \mathcal{N}}^{n-1}$. This axiom propagates the token from the global marked term to s.

Ax-4 $l : \overline{\overline{s^{m+1}} \rightarrowtail \mathcal{N}}^n \xrightarrow{t,\alpha/b(y)}_1 l : \overline{\overline{s^m} \rightarrowtail \overline{\mathcal{N}}}^n$. When event occurs; this axiom propagates the token from the marked term s to $\mathcal{N}$ after t time units and the effect b(y) inputs a name on b and stores it in y.

Ax-5 $\forall n > 0\ l : \overline{s \uparrow \mathcal{S}}^n \rightarrowtail \mathcal{N} \xrightarrow{l}_1 l : \overline{\overline{s} \uparrow \mathcal{S}}^{n-1} \rightarrowtail \mathcal{N}$. This axiom propagates the token from the global marked term to s.

Ax-6 $\dfrac{\mathcal{S}[\overline{l' : \odot}] \xrightarrow{l'}_1 |\mathcal{S}|}{l : \overline{s \uparrow \mathcal{S}^n} \rightarrowtail \mathcal{N} \xrightarrow{l'}_1 l : \overline{s \uparrow |\mathcal{S}|^n} \rightarrowtail \overline{\mathcal{N}}}$. The derivation rule Ax-6 finishes the execution of a call behavior and moves the token to the succeeding term $\mathcal{N}$.

Ax-7 $\dfrac{\mathcal{S} \xrightarrow{t,\alpha}_p \mathcal{S}'}{l : \overline{s \uparrow \mathcal{S}^n} \rightarrowtail \mathcal{N} \xrightarrow{t,\alpha}_p l : \overline{\overline{s} \uparrow \mathcal{S}'^n} \rightarrowtail \mathcal{N}}$. The derivation rules Ax-7 and Ax-8 present the effect on $\overline{s \uparrow \mathcal{S}^n}$ when $\mathcal{S}$ or $\mathcal{N}$ executes an action a with a probability p.

Ax-8 $\dfrac{\mathcal{N} \xrightarrow{t,\alpha}_1 \mathcal{N}'}{l : \overline{s \uparrow \mathcal{S}^n} \rightarrowtail \mathcal{N} \xrightarrow{t,\alpha}_p l : \overline{s \uparrow \mathcal{S}^n} \rightarrowtail \mathcal{N}'}$.

Ax-SUB $\dfrac{\mathcal{S}sub[\overline{l' : \odot}] \xrightarrow{l'}_1 |\mathcal{S}sub|}{l : \overline{s(\mathcal{S}_{entry}, \overline{\mathcal{S}_{sub}}, \mathcal{S}_{exit})} \rightarrowtail \mathcal{N} \xrightarrow{l'}_1 l : \overline{s(\mathcal{S}_{entry}, |\mathcal{S}_{sub}|, \overline{\mathcal{S}_{exit}})} \rightarrowtail \mathcal{N}}$. The derivation rule Ax-SUB finishes the execution of "Sub" behavior and moves the token to the EXIT behavior.

Ax-HIST $l : \overline{\overline{\mathcal{N}} \rightarrowtail l' : \mathcal{H}(S^*, S)}^n \xrightarrow{l} l : \overline{\mathcal{N} \rightarrowtail l' : \mathcal{H}(\overline{S}, S)}^n$. Ax-HIST is a shallow history; backs to the most recent active substate of its containing state.

FRK-1 $\forall n > 0\ l : \overline{F(\mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{l}_1 l : \overline{F(\overline{\mathcal{N}_1}, \overline{\mathcal{N}_2})}^{n-1}$. The FRK-1 axiom shows the multiplicity of the arriving tokens according to the outgoing sub-terms.

FRK-2 $\dfrac{\mathcal{N}_1 \xrightarrow{t,\alpha} \mathcal{N}_1'}{l : \overline{F(\mathcal{N}_1, \mathcal{N}_2)} \xrightarrow{t,\alpha} l : \overline{F(\mathcal{N}_1', \mathcal{N}_2)}}$. The FRK-2 derivation rule illustrates the changes on a fork term when its outgoing trigger a state.

CHOICE-1 $\forall n > 0\ l : \overline{D(g, \mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{g,\alpha} l : \overline{D(g, \overline{\mathcal{N}_1}, \mathcal{N}_2)}^{n-1}$. The axiom CHOICE-1 describes a non-probabilistic choice where a token flows through the edge satisfying its guard.

CHOICE-2 $\forall n > 0\ l : \overline{D(p, g, \mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{g,\alpha}_p l : \overline{D(p, g, \overline{\mathcal{N}_1}, \mathcal{N}_2)}^{n-1}$. The axiom CHOICE-2 describes a probabilistic decision where a token flows through the edge satisfying its guard with probability p.

MRG-1 $l : \overline{\overline{\mathcal{N}} \rightarrowtail l' : M(x, g_1, \mathcal{N}_1, y, g_2, \mathcal{N}_2)}^n \xrightarrow{l} l : \overline{\mathcal{N} \rightarrowtail l' : M(\overline{x}, g_1, \mathcal{N}_1, y, g_2, \mathcal{N}_2)}^n$. MRG-1 is a transition with a probability of value 1 to put a token coming from the sub-term $\mathcal{N}$ on a junction labeled by l.

MRG-2 $l : \overline{l' : M(\overline{x}, g_1, \mathcal{N}_1, \overline{y}, g_2, \mathcal{N}_2)}^n \xrightarrow{l,g_1}_1 l : \overline{l' : M(x, g_1, \overline{\mathcal{N}_1}, \overline{y}, g_2, \mathcal{N}_2)}^n$. MRG-2 is a transition with a probability of value 1 to present a token flowing from a junction labeled by l to the sub-term $\mathcal{N}_1$ an the guard g_1 is true.

MRG-3 $l : \mathcal{S}[l' : M(x, g_1, \mathcal{N}_1, y, g_2, \mathcal{N}_2), \overline{l_x}] \xrightarrow{l} l : \mathcal{S}[l' : M(\overline{x}, g_1, \mathcal{N}_1, y, g_2, \mathcal{N}_2), l_x]$. MRG-3 shows the junction enabled when token arrived at one of its pins.

JOIN-1 $l : \overline{\overline{\mathcal{N}} \rightarrowtail l' : J(x,y)}^n \xrightarrow{l}_1 l : \overline{\mathcal{N} \rightarrowtail l' : J(\overline{x},y)}^n$. JOIN-1 represents a transition with a probability of value 1 to activate the pin x in a join labeled by *l'*.
JOIN-2 $l : \overline{l' : J(\overline{x},\overline{y}) \rightarrowtail \mathcal{N}}^n \xrightarrow{\tau} l : \overline{l' : J(x,y) \rightarrowtail \overline{\mathcal{N}}}^n$. JOIN-2 represents a transition with a probability of value 1 to move a token in join to the sub-term $\mathcal{N}$.
JOIN-3 $l : \mathcal{S}[l' : J(x,y) \rightarrowtail \mathcal{N}, \overline{l_x}] \xrightarrow{\tau} l : \mathcal{S}[l' : J(\overline{x},y) \rightarrowtail \mathcal{N}, l_x]$. JOIN-3 shows the join input enabled when token arrived at one of its pins.
FFIN $\mathcal{S}[l : \overline{\times}] \xrightarrow{l} \mathcal{S}[l : \times]$. This axiom states that if the sub-term $l : \times$ is reached in $\mathcal{S}$ then a transition of probability one is enabled to produce a term describing the termination of a flow.
AFIN $\mathcal{S}[l : \overline{\odot}] \xrightarrow{l} |\mathcal{A}|$. This axiom states that if the sub-term $l : \odot$ is reached then no action is taken later by destroying all tokens.

6 PRISM Formalization

In this section, our formalization focus on probabilistic timed automata (PTA) that extends the standard probabilistic automata (PA). The PRISM model checker supports the PTA with the ability to model real-time behavior by adding real-valued clocks (i.e. clocks variable) which increases with time and can be reset (i.e. updated).

A Timed Probabilistic System (TPS) that represents a PRISM program (P) is composed of a set of "m" modules ($m > 0$). The state of each module is defined by the evaluation of its local variables V_L. The global state of the system is defined as the evaluation of local and global variables: V=$V_L \cup V_G$. The behavior of each module is described as a set of statements in the form of: $[act]guard \rightarrow p_1 : u_1 .. + p_n : u_n$, which means, for the action *act* if the guard g is true, then, an update u_i is enabled with a probability p_i. The update u_i is a set of evaluated variables expressed as conjunction of assignments $(V'_j = val_j)\&..\&(V'_k = val_k)$ where $V_j \in V_L$ U V_G and val_j are values evaluated via expressions denoted by *eval*, eval: V $\rightarrow \mathbb{R}$ U $\{True, False\}$. The formal definition of a command is given in Definition 2.

Definition 2. A PRISM command is a tuple c = < a, g, u >.

- act is an action label.
- guard is a predicate over V.
- u = $\{(p_i, u_i)\}$ $\exists m > 1, i < m,\ 0 < p_i < 1, \sum_i^m p_i = 1$ and u = $\{(v, eval(v)) : v \in V_l\}$.

The set of commands are associated with modules that are parts of a system and it definition is given in Definition 3.

Definition 3. A PRISM module is tuple M = <V$_l$, I$_l$, *Inv*, C>, where:

- V$_l$ is a set of local variable associated with a module,
- *Inv* is a time constraint of the form $v_l \bowtie d \backslash \bowtie \in \{\leq, \geq\}$ and d $\in \mathbb{N}$,
- I$_l$ is the initial value of V_l.
- C= $\{c_i, 0 < i \leq k\}$ is a set of commands that define the module behavior.

To describe the composition between different modules, PRISM uses CSP communication sequential process operators [10] such as Synchronization, Interleaving, Parallel Interface, Hiding and Renaming. Definition 4 provides a formal definition of PRISM system.

Definition 4. A PRISM system is tuple P = <V, I$_g$, exp, M, CSPexp>, where:

- V = $V_g \coprod_{(i=1)}^{m} V_{li}$ is the union of a set local and global variables.
- I$_g$ is initial values of global variables.
- *exp* is a set of global logic operators: - , $*, /, +, -, <, <=, >=, >, =, ! =,$!, &, | , <=> , => , ? (condition evaluation: condition ? a : b means "if condition is true then a else b").
- M is a set of modules composing a System.
- *CSPexp* is CSP algebraic expression:
  - M1 || M2 : alphabetised parallel composition of modules M1 and M2 (synchronising on only actions appearing in both M1 and M2)
  - M1 ||| M2 : asynchronous parallel composition of M1 and M2 (fully interleaved, no synchronisation)
  - M1 |[a,b,...]| M2 : restricted parallel composition of modules M1 and M2 (synchronising only on actions from the set a, b,...)
  - M $/\{a, b, ...\}$: hiding of actions a, b, ... in module M
  - M $\{a \leftarrow b, c \leftarrow d, ...\}$: renaming of actions a to b, c to d, etc.

6.1 PRISM Semantics

The probabilistic timed automata of a PRISM program $\mathscr{P}$ is based on the atomic semantics of a command C denoted by [[c]]. The latter is a set of transitions defined as follows: $[[c]] = \{(s, a, \mu) | s \models g\}$ where μ is a distribution over S such that $\mu(s, v_t) = \{|0 \leq p_i \leq 1, v \in V, s'(v) = eval(V)|\}$. The stepwise behavior of PRISM is described by the operational semantic as follows:
INIT $\langle V_i, init(V_i)\rangle \rightarrow \langle V_i([[init(V_i)]]), -\rangle$INIT initializes variables. For a module M_i, init returns the initial value of the local variable $v_i \in V_i$.
LOOP $\langle V_i, -\rangle \rightarrow \langle V_i\rangle$ This axiom presents a loop in a state without changing variables evaluations. It can be applied to avoid a deadlock.
UPDATE $\langle V_i, v_i' = eval(V)\rangle \rightarrow \langle V_i([[v_i]])\rangle$ UPDATE axiom describes the execution of a simple assignment for a given variable v_i. Its evaluation is updated in V_i of M_i.
CNJ-UPD $\left\langle V, v_i' = eval(V) \wedge v_j' = eval(V)\right\rangle \rightarrow \langle V([[v_i]], [[v_j]])\rangle$ CNJ-UPD implements the conjunction of a set of assignments.
PRB-UPD1 $\langle V_i, p : v_i' = eval(V)\rangle \rightarrow_p \langle V_i([[v_i]])\rangle$ $0 < p < 1$.
PRB-UPD2 $\left\langle V, p : v_i' = eval(V) \wedge v_j' = eval(V)\right\rangle \rightarrow_p \langle V([[v_i]], [[v_j]])\rangle$ $0 < p < 1$ PRB-UPD1 and PRB-UPD2 describe probabilistic updates.
ENB-CMD1 $\frac{V \models g, Inv(V)}{\langle V, M([a]g \rightarrow p_i : u_i)\rangle \rightarrow \mu}$ ENB-CMD1 enables the execution of a probabilistic command.
ENB-CMD2 $\frac{V \models g, Inv(V) \quad V \not\models g', Inv'(V)}{\langle V, [a]g \rightarrow u; [a']g' \rightarrow u'\rangle \xrightarrow{a} \langle V([[u]]), [a']g' \rightarrow u'\rangle}$ ENB-CMD2 enables the execution of a command in a module.

ENB-CMD3 $\frac{V \models g, Inv(V) \;\; V \models g', Inv'(V)}{\langle V,[a]g \rightarrow u;[a']g' \rightarrow u' \rangle \xrightarrow{a} \langle V([[u]]),[a']g' \rightarrow u' \rangle}$ ENB-CMD3 solves the non-determinism in a module by following a policy.

SYNC $\frac{\langle V_i, c_i \rangle \xrightarrow{a} \mu_i \;\; \langle V_j, c_j \rangle \xrightarrow{a} \mu_j}{\langle V_i \cup V_j, M_i || M_j \rangle \xrightarrow{a} \mu_i . \mu_j}$ SYNC derivation rule permits the synchronization between modules on a given action a.

INTERL $\frac{\langle V_i, M_i(c_i) \rangle \xrightarrow{a_j} \mu}{\langle V, M_i || \; | M_j \rangle \xrightarrow{a_j} \mu}$ INTERL derivation rule describes the interleaving between modules.

6.2 Property Specification in PRISM

In order to perform model-checking, a property should be specified. We selected PCTL to express such property. Formally, its syntax is given by the following BNF grammar:
$\varphi ::= true \mid ap \mid \varphi \wedge \varphi \mid \neg\varphi \mid P_{\bowtie p}[\psi]$,
$\psi ::= \varphi \cup^{\leq k} \varphi \mid \varphi \cup \varphi$,

Where "ap" is an atomic proposition, P is a probabilistic operator. p $\in$ [0, 1] and "$\bowtie$"$\in <, \leq, >, \geq$. Bound until means that a state satisfying $\varphi 2$ is eventually reached and that, at every time-instant prior to that, $\varphi 1$ is satisfied. The time-bounded variant has the same meaning, where the occurrence of $\varphi 2$ occur within time k. To specify the satisfaction relation of a PCTL formula a class of adversaries (Adv) has been defined [16] to solve the nondeterminism decision.

7 The Verification Approach

This section describes the transformation of SysML state machine diagrams $\mathcal{S}$ into a PTA written in PRISM input language. Listing. 1 propose a mapping function Γ that takes as input the SMC terms defined in Table 2 to produce a PRISM commands. The action label of a command is the label of its related term n. The guard of this command depends on how the term is activated and minimal clock valuation. The flag related to the term is its label l that is initialized to false except for the initial node it is true which conforms to the premise of the SMC rule **Ax-1**. The updates of the command deactivate the propositions of the term, activate that ones related to its successors, reset the clock variable of its successors. The functions $L(n)$, $Start(S_i)$ and $E(S_i)$, return the label of the initial term n , the initial and final term of S_i, respectively. Each PRISM code generated for each state machine diagram starts from module S_i and terminates with *endmodule*. The call of substates transitions (line 30 and 31) *synchronize* with the initial (line 42) and the final (line 44) transition, respectively to enable the internal transitions of substate. The final transition (line- 38) reset the local variables to false. However, the PTA model in PRISM model checker does not support the shared variables. To overcome, we use the implication operator to set the proposition to true as shown in line 31 and line 42. The clock variable x is used as guarded condition for state successors activation. In line 34, the next node is activated when the clock $x >= min$ and $x <=$ max defined in the invariant clause within the module as follows:

$$\textbf{invariant}\ \ (l = true) \Rightarrow (x \leq max)\ \ \textbf{endinvariant}$$

```
$\mathit{\Gamma} : S \rightarrow \mathscr{P}$
$\mathit{\Gamma}(S) = \forall n \in S,\ L(n == i) = true, L(n \neq i) = false$ , Case n of
$[l : i \rightarrow \mathcal{N}] \Rightarrow$ // the clock x is reset to 0
in {[l] l →(l'=false)&(L($\mathcal{N}$)'=true)&($x' = 0$)}$\cup\mathit{\Gamma}(\mathcal{N})$; end
$[l : M(x, y, g_1, N_1, g_2, N_2)] \Rightarrow$
in $\{[l_x]l_x \rightarrow (l'_x = false)\&(l_{g1})' = true)\}\cup\ \mathit{\Gamma}(\mathcal{N}_1) \cup \mathit{\Gamma}(\mathcal{N}_2)\cup$
$\{[l_y]l_y \rightarrow (l'_y = false)\&(l'_{g2} = true)\}\cup$
$\{[l_{g1}]l_{g1}\ \&\ g1 \rightarrow (l'_{g1} = false)\&L(\mathcal{N}_1)' = true)\ \&\ (x' = 0)\}\cup$
$\{[l_{g2}]l_{g2}\ \&\ g2 \rightarrow (l'_{g2} = false)\ \&\ L(\mathcal{N}_2)' = true)\ \&\ (x' = 0)\}$
end
$[l : J(x, y) \rightarrow \mathcal{N}] \Rightarrow$
in $\{[l]l_x \wedge l_y \rightarrow (l'_x = false)\&(l'_y = false)\&(L(\mathcal{N})' = true)\&(x' = 0)\}$ end
$[l : F(\mathcal{N}_1, \mathcal{N}_2)] \Rightarrow$
in $\{[l]l \rightarrow (l' = false)\&(L(\mathcal{N}_1)' = true)\&(L(\mathcal{N}_2)' = true)\&(x'_1 = 0)\&(x'_2 = 0)\}\cup\mathit{\Gamma}(\mathcal{N}_1) \cup \mathit{\Gamma}(\mathcal{N}_2)$
end
$[l : D(p, g, \mathcal{N}_1, \mathcal{N}_2)] \Rightarrow$
Case(p) of
]0, 1[ $\Rightarrow$
in $\{[l]l \rightarrow p : (l' = false)\&(l'_g = true) + (1 - p) : (l' = false)\&(l'_{\neg g} = false)\}\cup\mathit{\Gamma}(\mathcal{N}_1) \cup \mathit{\Gamma}(\mathcal{N}_2)\cup$
$\{[l_g]l_g \rightarrow (l'_g = false)\ \&\ (L(\mathcal{N}_1)' = true)\ \&\ (x'_1 = 0)\}$
$\{[l_{\neg g}]l_{\neg g} \rightarrow (l'_{\neg g} = false)\&\ (L(\mathcal{N}_2)' = true)\ \&\ (x'_2 = 0)\}$ end
Otherwise $\Rightarrow$
in $\{[l]l \rightarrow (l' = false)\&(l'_g = true)\}\cup\{\ [l]l \rightarrow (l' = false)\ \&\ (l'_{\neg g} = true)\}\cup\mathit{\Gamma}(\mathcal{N}_1) \cup \mathit{\Gamma}(\mathcal{N}_2)\cup$
$\{[l_g]l_g \rightarrow (l'_g = false)\ \&\ L(\mathcal{N}_1)' = true)\&(x'_1 = 0)\}\cup$
$\{[l_{\neg g}]l_{\neg g} \rightarrow (l'_{\neg g} = false)\&\ L(\mathcal{N}_2)' = true)\&(x'_2 = 0)\}$ end
$[l : s\mathcal{B} \xrightarrow{t} \mathcal{N}] \Rightarrow$
Case($\mathcal{B}$) of
$\uparrow \mathcal{S}_i\ \Rightarrow$
in
$\{[l]l \rightarrow (l' = false)\}\cup\mathit{\Gamma}(\mathcal{N})\cup$
$\{[L(E(\mathcal{S}_i))]((L(\mathcal{N}) = false) \Rightarrow true) \rightarrow (L(\mathcal{N})' = true)\&(x' = 0)\}\cup\mathit{\Gamma}'(\mathcal{S}_i)$
end
$\epsilon\ \Rightarrow$ // minimal time for state transition
in $\{[l]l\ \&\ (t >= min) \rightarrow (l' = false)\ \&\ (L(\mathcal{N})' = true)\ \&\ (x' = 0)\}$end
$[l : \times] \Rightarrow$
in $[l]l \rightarrow (l' = false)$ ; end
$[l : \odot] \Rightarrow$
in $[l]l \rightarrow\ \&\ _{l\in\mathcal{L}}(L(\mathcal{N})' = false)$; end
$\mathit{\Gamma}' : S \rightarrow P$
$\mathit{\Gamma}(S_i) = \forall m \in S_i,\ L(m) = false$ ,
$[l : i \rightarrow \mathcal{N}] \Rightarrow$
in $\{[l]\ ((L(Start(\mathcal{S}_i)) = false) => true) \rightarrow (L(Start(\mathcal{S}_i))' = true)\}\ \cup$
$\{[L(Start(\mathcal{S}_i))]\ L(Start(\mathcal{S}_i)) \rightarrow (L(Start(\mathcal{S}_i))' = false)\ \&\ (L(\mathcal{N})' = true)\ \}\cup\mathit{\Gamma}(\mathcal{N})$; end
$[l : \odot] \Rightarrow$ in $[L(E(S_i))]L(E(S_i)) \rightarrow (L(E(S_i))' = false)$; end
```

Listing 1. PRSIM Commands Generation

8 The Transformation Soundness

Our aim is to prove the soundness of the transformation algorithm $\mathit{\Gamma}$ by showing that the proposed algorithm preserves the satisfiability of PCTL properties. Let $\mathcal{S}$ be a SMC term and $M_{\mathcal{S}}$ is its corresponding PTA constructed by the SMD operational semantics denoted by $\mathcal{S}$ such that $\mathcal{X}(\mathcal{S}) = M_{\mathcal{S}}$. For the program $\mathscr{P}$ resulting after transformation rules, Let M_p its corresponding PTA constructed

$$\begin{tikzcd} S \arrow[r, "\mathscr{S}"] \arrow[d, "\Gamma"'] & M_S \arrow[d, "\mathscr{R}"] \\ \mathscr{P} \arrow[r, "\mathscr{S}'"] & M_{\mathscr{P}} \end{tikzcd}$$

Fig. 4. The transformation soundness.

by PRISM operational semantics denoted $\mathcal{X}'$ such that $\mathcal{X}'(\mathscr{P}) = M_{\mathscr{P}}$. As illustrated in Fig. 4, proving the soundness of Γ algorithm is to find the adequate relation $\mathcal{R}$ between $M_{\mathcal{S}}$ and $M_{\mathscr{P}}$.

To define the relation $M_{\mathcal{S}}\mathcal{R}M_{\mathscr{P}}$, we have to establish a step by step correspondence between $M_{\mathcal{S}}$ and $M_{\mathscr{P}}$. First, we introduce the notion of the timed probabilistic bisimulation relation [6,19] in Definitions 6 and 7. This relation is based on the probabilistic equivalence relation $\mathcal{R}$ defined in Definition 5 where $\delta/\mathcal{R}$ denotes the quotient space of δ with respect to $\mathcal{R}$ and $\equiv_{\mathcal{R}}$ is the lifting of $\mathcal{R}$ to a probabilistic space.

Definition 5 (The equivalence $\equiv_{\mathcal{R}}$). If $\mathcal{R}$ is an equivalence on δ, then the induced equivalence $\equiv_{\mathcal{R}}$ on $\mathrm{Dist}(\delta \times 2^x)$ is given by: $\mu_{\equiv_{\mathcal{R}}}\mu'$ iff $\mu(\delta, d) \equiv_{\mathcal{R}} \mu(\delta, d')$.

Definition 6 (Timed Probabilistic Bisimulation Relation). A binary relation $\mathcal{R}$ over the set of states of PTAs is timed bisimulation iff whenever $s_1\mathcal{R}s_2$, α is an event and d is a delay:

- if $s_1 \overset{d,\alpha}{\to} \mu(s_1, d)$ there is a transition $s_2 \overset{d',\alpha}{\to} \mu(s_2, d')$, such that $s_1\mathcal{R}s_2$. The delay d can be different from d';
- two states s, s' are time probabilistic bisimilar, written $s \sim s'$, iff there is a timed probabilistic bisimulation related to them.

Definition 7 (Timed Probabilistic Bisimulation of PTAs). Probabilistic Timed automata A_1 and A_2 are timed probabilistic bisimilar denoted $(A \sim A')$ iff their initial states in the union of the probabilistic timed transition systems T$(A_1)$and T(A_2) generated by A_1 and A_2 are timed probabilistic bisimilar.

For our proof, we stipulate herein the mapping relation $\mathcal{R}$ denoted by $M_{\mathcal{S}}\mathcal{R}M_{\mathcal{P}}$ between a SMC term $\mathcal{S}$ and its corresponding PRISM term $\mathscr{P}$.

Definition 11 (Mapping Relation). The relation $M_{\mathcal{S}}\mathcal{R}M_{\mathscr{P}}$ between a SMC term $\mathcal{S}$ and a PRISM term $\mathscr{P}$ such that $\Gamma(\mathcal{S}) = \mathscr{P}$ is a timed probabilistic bisimulation relation.

Finally, proving that Γ is sound means showing the existence of a timed probabilistic bisimulation between $M_{\mathcal{S}}$ and $M_{\mathscr{P}}$.

Lemma 1 (Soundness). The mapping algorithm Γ is sound, *i.e.* $M_{\mathcal{S}} \sim M_{\mathscr{P}}$.

Proof 1: We prove $M_{\mathcal{S}} \sim M_{\mathscr{P}}$ by following a structural induction on SMC terms and their related PRISM terms. For that, let $e_1, e_1' \in \mathcal{X}_{\mathcal{S}}$ and $e_2, e_2' \in \mathcal{X}_{\mathscr{P}}$. We distinguish the following cases where L(s) takes different values:

1. L(e_1) = $l : \overline{x} \rightarrowtail \mathcal{N}$ such as $x = \{i, s\} \Longrightarrow \exists e_1 \xrightarrow{d,\alpha}_1 e_1'$, L$(e_1$')= $l : x \rightarrowtail \overline{\mathcal{N}}$. For L$(e_2) = \Gamma(L(e_1))$, we have L$(e_2)$=$\langle L(x), \neg L(\mathcal{N})\rangle$ then $\exists e_2 \xrightarrow{d',\alpha}_1 e_2'$ where $L(e_2') = \langle \neg L(x), L(\mathcal{N})\rangle$.
2. L(e_1) = $l : \overline{D(g_1, \mathcal{N}_1, \mathcal{N}_2)}^n$ then $\exists e_1 \xrightarrow{g_1,\alpha}_1 e_1'$, L$(e_1')$= $l : \overline{D(g_1, \overline{\mathcal{N}_1}, \mathcal{N}_2)}^{n-1}$. For L$(e_2) = \Gamma(L(e_1))$, we have L$(e_2)$=$\langle l, \neg l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2}\rangle$ then $\exists e_2 \xrightarrow{g_1,\alpha}_1 s_2'$ where $L(e_2') = \langle \neg l, l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2}\rangle$.
3. L(e_1) = $l : \overline{\odot}$ then $\exists e_1 \xrightarrow{\alpha}_1 e_1'$, L$(e_1$')= $l : \odot$. For L$(e_2) = \Gamma($L$(e_1))$, we have L(e_2)=$\langle l\rangle$ then $\exists e_2 \xrightarrow{\alpha}_1 e_2'$ where $\forall l_i \in \mathcal{L} : L(e_2') = \langle \neg l_i\rangle$.

From the obtained results, we found that $\mu(e_1, d) = \mu(e_2, d') = 1$ then $e_1 \sim e_2$. In addition, the unique initial state of $M_{\mathcal{S}}$ is always corresponding to the unique initial state in $M_{\mathscr{P}}$. By studying all SMC terms, we find that $M_{\mathcal{S}} \sim M_{\mathscr{P}}$, which confirms that Lemma 1 holds.

In the following, we show that the mapping relation preserves the satisfiability of PCTL properties. This means, if a PCTL property is satisfied in the resulting model by a mapped function Γ then it is satisfied by the original one.

Proposition 1 (PCTL Preservation). For two PTAs $M_{\mathcal{S}}$ and $M_{\mathscr{P}}$ such that $\Gamma(\mathcal{S}) = \mathscr{P}$ where $M_{\mathcal{S}} \sim M_{\mathscr{P}}$. For a PCTL property ϕ, then: $(M_{\mathcal{S}} \vDash \phi) \Longleftrightarrow (M_{\mathscr{P}} \vDash \phi)$.

Proof 2: The preservation of PCTL properties is proved by induction on the PCTL structure and its semantics. Since $M_{\mathcal{S}} \sim M_{\mathscr{P}}$ and by relying to the semantics of each PCTL operator $\zeta \in \{$U, U$^{\leq k}$, F, P$_{\bowtie p}\}$, we find that $(M_{\mathcal{S}} \vDash \zeta) \Longleftrightarrow (M_{\mathscr{P}} \vDash \zeta)$ which means: $(M_{\mathcal{S}} \vDash \phi) \Longleftrightarrow (M_{\mathscr{P}} \vDash \phi)$.

9 Case Study

In the following, we present a case study [7] describing an automated teller machine (ATM). We perform the verification of this design with respect to predefined properties including time constraints and [2] is the corresponding generated code.

The ATM interacts with a potential customer (user) via a specific interface and communicates with the bank over an appropriate communication link. A user that requests a service from the ATM has to insert an ATM card and enter a personal identification number (PIN). The card number and the PIN need to be sent to the bank for validation. If the credentials of the customer are not valid, the card will be ejected. Otherwise, the customer will be able to carry out one or more transactions (e.g., cash advance or bill payment). The card will be retained in the ATM machine

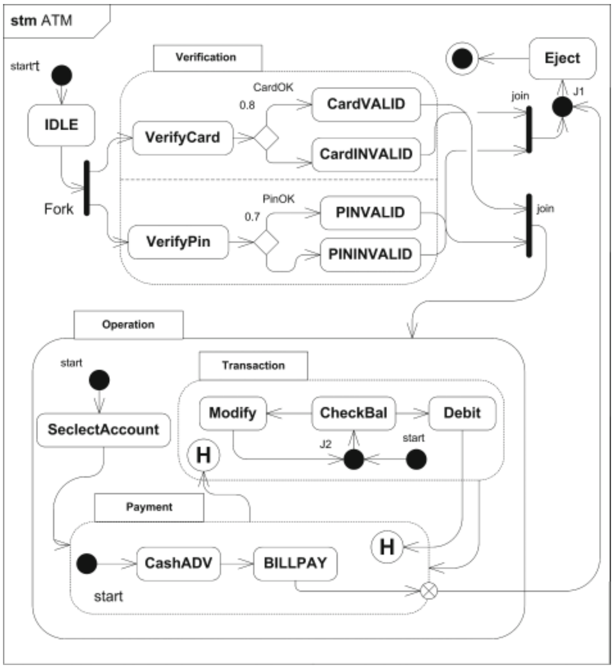


Fig. 5. ATM state machine diagram

during the customers interaction until the customer wishes for no further service. Figure 5 shows the SysML state machine diagram of the ATM system.

ATM state machine encloses four substates: IDLE, Verification, Eject, and Operation. The IDLE is the default initial substate of the top state. The Verification state represents the verification of the cards validness and authorization. VerifiyCard and VerifyPin substates have interval time]3s, 5s[,]4s, 5s[respectively (s for seconds). The SMD could let time progress while remaining in VerifiyCard and VerifyPin states until 5 seconds and the transition is triggered or the transition is triggered just after both states minimal time is attained (after time progress). The probability to get pin and card validated is 0.7 and 0.8, respectively. The Eject state depicts the phase of termination of the users transaction. The Operation state is a composite state that capture several functions related to banking operations. These are the *Selectaccount*, *Payment*, and *Transaction*. When Selectaccount is active, and the user selects an account, the next transition is enabled and the *Payment* is entered. The Payment state has two substates; for cash advancing and bill payment. It represents a two-item menu. Finally, the Transaction state captures the transaction phase and includes three substates: *CheckBal* for checking the balance, *Modify* for modifying the amount, if necessary, and *Debit* for debiting the account. Each one of the Payment and Transaction states contains a shallow history pseudostate. If a transition targeting a shallow history pseudostate is fired, the most recently active substate in the composite state containing the history connector is activated.

In order to check the correctness of the ATM system, we propose to verify two functional requirements at specific time or at different time stamps k. They are expressed in PCTL as follows:

1. The maximum probability value that the modification occurs during the Bill Payment after k=5 time units: $Pmax =?[F^{\leq k}(BillPAY\ \&\ Modify))]$.
2. The maximum probability value to get the card and pin validated after k time units: $Pmax =?[F^{\leq k}(CardVALID\ \&\ PinVALID\ )]$.

The maximum probability value for the modification that occurs during the Bill payment is equal to 0.3 when k equal to 5 (time units). The verification results of the second property are shown in Fig. 6. After 4 time units (seconds), the verification results converge to 0.3. However, the verification time for the first property took 246.3 s due to the state explosion during the model checking.

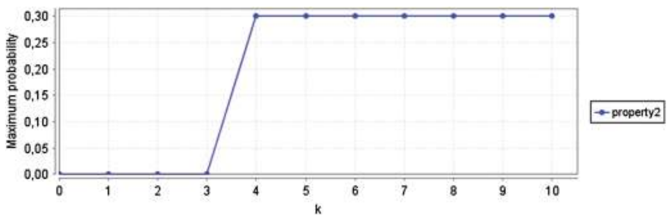


Fig. 6. Property2

10 Conclusion

In this paper, we presented a formal verification approach of probabilistic systems modeled by using SysML state machine diagram. For this purpose, the approach maps state machine into the input language of the probabilistic model checker PRISM. We proposed a calculus dedicated to this diagram that captures precisely their underlying semantics. In addition, we formalized PRISM language by showing its semantics. Thus, we proved the soundness of our proposed approach by defining adequately the relation between the semantics of the mapped diagrams and the resulting PRISM models. In addition, we proved the preservation of the satisfiability of PCTL properties. Finally, we have shown the effectiveness of our approach by applying it on a case study representing an ATM state machine diagram where time and probability are evaluated using PCTL properties. The presented work can be extended in the following two directions. First, we want to transform our behavioral diagram to its equivalent HDL (hardware description language) code for RTL verification. Second, we want to validate our approach on different real case studies.

References

1. OMG Systems Modeling Language (Object Management Group SysML). O. M. Group (Ed.) (2012)
2. Abdelhakim, B.: State machine diagram verification (2015). https://github.com/gitmodelcheking/ATM/blob/master/ATM.nm
3. Ando, T., Yatsu, H., Kong, W., Hisazumi, K., Fukuda, A.: Formalization and model checking of SysML state machine diagrams by CSP#. In: Murgante, B., Misra, S., Carlini, M., Torre, C.M., Nguyen, H.-Q., Taniar, D., Apduhan, B.O., Gervasi, O. (eds.) ICCSA 2013, Part III. LNCS, vol. 7973, pp. 114–127. Springer, Heidelberg (2013)
4. Baier, C., Katoen, J.P.: Principles of Model Checking (Representation and Mind Series). The MIT Press, Cambridge (2008)
5. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
6. Ben-Menachem, M.: Reactive systems: modelling, specification and verification. SIGSOFT Softw. Eng. Notes **35**(4), 34–35 (2010)
7. Debbabi, M., Hassane, F., Jarraya, Y., Soeanu, A., Alawneh, L.: Probabilistic model checking of SysML activity diagrams. In: Debbabi, M., Hassane, F., Jarraya, Y., Soeanu, A., Alawneh, L. (eds.) Verification and Validation in Systems Engineering, pp. 153–166. Springer, Berlin (2010)
8. Doligalski, M., Adamski, M.: UML state machine implementation in FPGA devices by means of dual model and verilog. In: 11th IEEE International Conference on Industrial Informatics, INDIN 2013, 29–31 July 2013, Bochum, Germany, pp. 177–184 (2013)
9. Gajski, D.D., Abdi, S., Gerstlauer, A., Schirner, G.: Embedded System Design: Modeling, Synthesis and Verification, 1st edn. Springer Publishing Company Incorporated, New York (2009)
10. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall Inc., Upper Saddle River (1985)
11. Huang, X., Sun, Q., Li, J., Pan, M., Zhang, T.: An MDE-based approach to the verification of SysML state machine diagram. In: Proceedings of the Fourth Asia-Pacific Symposium on Internetware, Internetware 2012, pp. 9:1–9:7. ACM, New York (2012)
12. Kaliappan, P.S., König, H., Kaliappan, V.K.: Designing and verifying communication protocols using model driven architecture and spin model checker. In: International Conference on Computer Science and Software Engineering, CSSE 2008, Volume 2: Software Engineering, 12–14 December 2008, Wuhan, China, pp. 227–230 (2008)
13. Mallet, F., de Simone, R.: MARTE: a profile for RT/E systems modeling, analysisand simulation? In: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, SimuTools 2008, 3–7 March 2008, Marseille, France, p. 43 (2008)
14. Milner, R.: Communicating and Mobile Systems: The π-calculus. Cambridge University Press, New York (1999)
15. Norman, G., Palamidessi, C., Parker, D., Wu, P.: Model checking the probabilistic π-calculus. In: Proceedings of the 4th International Conference on Quantitative Evaluation of Systems (QEST 2007), pp. 169–178. IEEE Computer Society (2007)

16. Norman, G., Parker, D., Sproston, J.: Model checking for probabilistic timed automata. Form. Methods Syst. Des. **43**(2), 164–190 (2013)
17. Ouchani, S., Mohamed, O., Debbabi, M.: A probabilistic verification framework of sysml activity diagrams. In: IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT), vol. 246, pp. 165–170, September 2013
18. Pajic, M., Jiang, Z., Lee, I., Sokolsky, O., Mangharam, R.: From verification to implementation: a model translation tool and a pacemaker case study, pp. 173–184 (2012)
19. Segala, R.: A compositional trace-based semantics for probabilistic automata. In: Lee, I., Smolka, S. (eds.) CONCUR '95: Concurrency Theory. Lecture Notes in Computer Science, vol. 962, pp. 234–248. Springer, Heidelberg (1995)
20. Sun, J., Liu, Y., Dong, J.: Model checking CSP revisited: introducing a process analysis toolkit. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. Communications in Computer and Information Science, vol. 17, pp. 307–322. Springer, Heidelberg (2008)