A Security Verification Framework for SysML Activity Diagrams

Samir Ouchani

▶ **To cite this version:**

Samir Ouchani. A Security Verification Framework for SysML Activity Diagrams. Computer Science [cs]. Concordia University, 2013. English. NNT : . tel-04201144

HAL Id: tel-04201144
https://hal.science/tel-04201144

Submitted on 9 Sep 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Security Verification Framework for SysML Activity Diagrams

Samir Ouchani

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

September 2013

© Samir Ouchani, 2013

CONCORDIA UNIVERSITY

Division of Graduate Studies

This is to certify that the thesis prepared

By: **Samir Ouchani**

Entitled: **A Security Verification Framework for SysML Activity Diagrams**

and submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Dr. Akif A.Bulgak

_____ Dr. Yamine Aït-Ameur

_____ Dr. Ferhat Khendek

_____ Dr. Abdelwahab Hamou-Lhadj

_____ Dr. Olga Ormandjieva

_____ Dr. Mourad Debbabi

_____ Dr. Otmane Aït Mohamed

Approved by _____

Chair of the ECE Department

_____ 2012 _____

Dean of Engineering

ABSTRACT

A Security Verification Framework for SysML Activity Diagrams

Samir Ouchani

Concordia University, 2013

UML and SysML play a central role in modern software and systems engineering. They are considered as the de facto standard for modeling software and systems. Today's systems are created from a myriad of interacting parts that are combined to produce visible behavior. The main difficulty arises from the different ways in modeling each component and the way they interact with each other. Moreover, nowadays secure software has become an essential part in industrial development. One challenge in academia as well as in industry is to produce a secure product. Another challenge is to prove its correctness especially when the software environment is imprecise and uncertain.

The aim of this thesis is to provide a practical and formal framework that enables security risk assessment and security requirements verification on a system modeled as a composition of UML/SysML behavioral diagrams. Our main contribution is a novel approach to automatically verify security of systems on their design models based on security requirements, probabilistic adversarial interactions between potential attackers and the system's models. These structures are shaped to provide an elegant way to define the combination between different kinds of diagrams. We rely on stochastic security templates to specify security properties and a standard catalogue of attack patterns to build a library of attacks design patterns. The result of the interaction between selected attack scenarios and the composed diagrams with the instantiated security properties are used to quantify security risk by applying probabilistic model-checker. To handle the verification process

scalability, our approach allows the verification of large system efficiently by optimizing and avoiding the global model construction. To demonstrate the effectiveness of our approach, we apply our methodology on academia as well as industrial benchmarks.

keywords: Probabilistic Verification, Temporal Logic, Probabilistic Automata, Security Properties, Attack Pattern, Vulnerability Detection, Risk Assessment, UML, SysML, Activity Diagrams.

ACKNOWLEDGEMENTS

I would like to express my deeply-felt gratitude to my supervisors Professor Mourad Debbabi, and Dr. Otmane Ait Mohamed, for having accepted me as a Ph.D student. I could not have asked for better role models, each inspirational, supportive, and patient. I could not be prouder of my academic roots and hope that I can in turn pass on the research values and the dreams that these two men have given to me.

This thesis was co-funded by Concordia University and ERICSSON for MOBS II project, and I would like to thank both organizations for their generous support. As a member of MOBS II project, I have been surrounded by wonderful colleagues; both communities have provided a rich and fertile environment to study and explore new ideas. At ERICSSON, I would like to thank Dr. Makan Pourzandi, who has been extremely supportive.

l am beholden to all my friends in the computer security laboratory and the hardware verification group at Concordia university for their assistance. Thank you for welcoming me as a friend and helping to develop the ideas in this thesis.

More especially, I owe a huge debt of gratitude to my parents and to all my family members back home for their support, encouragement, and most of all patience throughout the course of my studies.

And last, but not least, to my wife, who shares my passions, thank you for your help. To my wonderful daughter who always brings me inspiration, meaning and purpose to my life. I dedicate to all of you this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

BNF	Backus-Naur Form
CAPEC	Common Attack Pattern Enumeration and ClassiïňĄcation
CTL	Computation Tree Logic
CTMC	Continuous Time Markov Chains
CTS	Configuration Transition System
CMRM	Continuous time Markov Reward Models
CTMDP	Continuous Time Markov Decision Processes
DTMC	Discrete Time Markov Chains
DMRM	Discrete time Markov Reward Models
ERTS	Embedded Real-time System
ETPN	Time Petri Net with Energy constraints
GSMP	Generalized Semi Markovian Process
GTS	Graph Transformation Systems
I/O	Input/Output
INCOSE	International Council on Systems Engineering
LTL	Linear Temporal Logic
L^2TS	Doubly-Labeled Transition System
MDP	Markov Decision Processes
MARTE	Modeling and Analysis of Real-Time and Embedded systems
ND	Non-Determinism
NuAC	New Activity Calculus
OMG	Object Management Group

PCTL	Probabilistic Computation Tree Logic
PRISM	PRobabilistIc Symbolic Model checker
PA	Probabilistic Automata
PTA	Probabilistic Timed Automata
PC	Probabilistic Choice
PP	Protocol Predictor
QVT	Query View Transformation
RTSP	Real Time Streaming Protocol
SPN	Stochastic Petri Nets
SysML	Systems Modeling Language
SVF	Static VeriïňĄcation Framework
SPSL	Symmetric probabilistic speciïňĄcation language
TOPCASED	Toolkit in OPen source for Critical Applications and SystEms Development
TABU	tool for the Active Behavior of UML
TPTL	Timed Propositional Temporal Logic
UML	Unified Modeling Language
WASC	Web Application Security Consortium
xUML	Executable UML

Chapter 1

Introduction

Unified and Systems Modeling Languages (UML and SysML) [60, 61] are software and systems engineering dedicated modeling languages developed by the Object Management Group (OMG) and the International Council on Systems Engineering (INCOSE). UML and SysML are prominent object-oriented modeling languages that have become today's de-facto standards for modern software and systems engineering. They support both structural and behavioral modeling, and they bear the composition and the interaction between different types of behavioral diagrams. SysML reuses a subset of UML packages and extends others with specific systems engineering features such as probability. It covers four main perspectives of systems modeling: structure, behavior, requirements, and parametric diagrams [60]. Particularly, SysML activity diagrams are behavioral diagrams used to model system behaviors at various levels of abstraction [35]. In addition, they support systems' composition by the call behavior and send/receive artifacts.

1.1 Motivation

A major challenge in software and systems development process is to advance error detection at early stages of their life-cycles. 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 [5]. Figure 1.1 depicts bugs introduction, detection, and repair costs during software life cycle. It shows that 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 testing phase. Therefore, an ambitious challenge is to accelerate the verification process of a product based on its design artifacts.

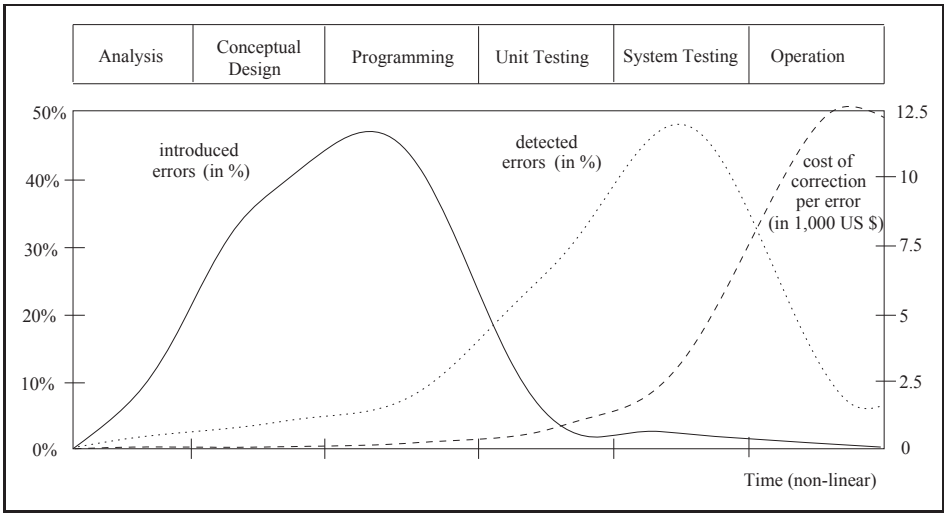


Figure 1.1: Software Development Error Cost [5].

Yet, another more ambitious challenge is to accurately specify, express and measure the security level of a product based on its design artifacts. Various techniques have been proposed for the verification of software and hardware such as model checking [15], type checking, equivalence checking [38], theorem proving [55], as well as static and dynamic analysis. Particularly in software and systems engineering [21], the most popular technique used to verify UML/SysML behavioral diagrams is model checking.

Model checking [9, 15] is an automatic formal verification technique that checks systems specifications on finite-state concurrent systems. Generally, temporal logic is used to express system requirements and symbolic-based algorithms are implemented to check if those requirements hold on the system model or not. If the property is violated, a counterexample is provided. In addition to qualitative model checking, quantitative verification techniques based on probabilistic model checkers [5, 27] 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. In this thesis, we use the PRISM probabilistic symbolic model checker [50] for our experiments.

Despite its wide use, model checking is generally a resource-intensive process that requires a large amount of memory and processing time. 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, which may hinder the verification process. Consequently, it is of a major importance to reduce the verification process complexity. To overcome this issue, various techniques have been explored [5, 9, 15, 90] 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 [15], while others target the analysis of the model itself. Henceforth, 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 during the verification process. This thesis concentrates on both classes.

Abstraction is one of the most relevant techniques for addressing the state explosion problem [5, 9, 15, 90]. It can be defined as a mapping from a concrete model into a more

abstract one that encapsulates the systems' behavior while being of a reduced size. The intuition behind this transformation is to be able to check a property against an abstract model and then to infer safely the same results on the concrete model. Abstraction techniques can be classified into four categories: 1) Abstraction by state merging, 2) Abstraction on variables, 3) Abstraction by restriction, and 4) Abstraction by observer automata. As well as, the well-known compositional verification techniques [10]: partitioned transition relation, lazy parallel composition, interface processes, and assume-guarantee. Our proposed framework takes advantage of the three first categories of abstraction, and the interface processes technique that complies with the composition of SysML activity diagrams.

Owing to the fact that it is never enough to just ensure the functional correctness of a given system [36], ensuring the security of a system is a real challenge. Especially, Symantec statistics [56] show an increasing by 42% of attacks for 5291 vulnerabilities in 2012 versus 4989 vulnerabilities in 2011. In addition, McAfee [53] announced in September 2012 that a mass fraud campaign planned against 30 US banks is supposed to occur by spring 2013. From a security perspective, a strong system is one in which the cost of an attack is greater than the potential gain to the attacker. Conversely, a weak system is one where the cost of an attack is less than the potential gain. The cost of an attack should take into account not only money, but also time to recovery and potential punishment for criminal activities [59]. For this purpose different attack models have been deployed such as: attack tree [52], attack graph [78], and network attack graph [78]. By using the existing attack models, there is a gap between the system and the attack semantics. Hence, we propose to use SysML activity diagrams to model attacks by relying on the attack standard CAPEC[1] that is developed by the community knowledge resource for building secure software.

It is extremely important to provide a mechanism employing quantitative techniques

[1] http://capec.mitre.org, Common Attack Pattern Enumeration and Classification sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security.

for security evaluation of software and systems based on their design models. In this thesis, we address the issue of security risk assessment of software/systems modeled by using SysML activity diagrams. The goal is to gauge how well a product is meeting its security requirements. Since we use model-checking technique, temporal logic is used to express security properties. The downside of temporal logic resides in its expressiveness. It is difficult to express a temporal logic formula starting from a natural language statement. To circumvent this downside, we propose to automatically generate security properties from SysML activity diagrams.

1.2 Problem Statement

Current research initiatives focus mainly on qualitative model checking of SysML to ensure the correctness of systems security and functionality. Furthermore, most of the proposed approaches are intended to verify or propose a formal model of only a single behavioral diagram [3, 8, 13, 20, 22, 40, 41, 43, 44, 46, 81]. To verify these diagrams, the existing approaches [3, 6, 7, 13, 18, 20, 22, 40, 41, 75] ignore systems composition. More especially, they do not show the preservation of the system requirements while they use generally model-checking. In addition, most of these approaches inherit the model checker limitations. Furthermore, security is rarely verified in UML and SysML behavioral diagrams. Since SysML is a young modeling language that extends UML with system features, only few related works exist. Herein, this thesis aims at investigating and answering fundamental questions:

1. How SysML activity diagrams are composed?
2. How to specify security aspect in SysML activity diagrams?
3. How to verify and evaluate security in SysML activity diagrams?

4. How to avoid the model checking limitations?

1.3 Objectives

The main goal of this thesis is to check and quantify the security of systems at design level by introducing security properties and potential attack models. Thus, we propose a set of stochastic security templates values and a library of attack models with varied potential gains that can exploit the main system vulnerabilities. The security properties to be verified are instantiated easily from the proposed templates. The objectives of this thesis are summarized as follows:

1. Providing an efficient verification framework to evaluate security of systems modeled by SysML activity diagrams,
2. Studying the formal verification background and surveying the existing related work and model checking tools,
3. Facilitating the specification of the system security and functional requirements,
4. Giving an adequate formal semantics for SysML activity diagram,
5. Demonstrating the efficiency of the proposed framework on real systems.

1.4 Proposed Methodology

This section describes our framework to specify and verify the security of systems modeled by using SysML activity diagrams. As an improvement over the existing solutions, our proposed framework targets systems composed of different parts. Furthermore, it automatically evaluates the security of systems at the design level by instantiating a set of

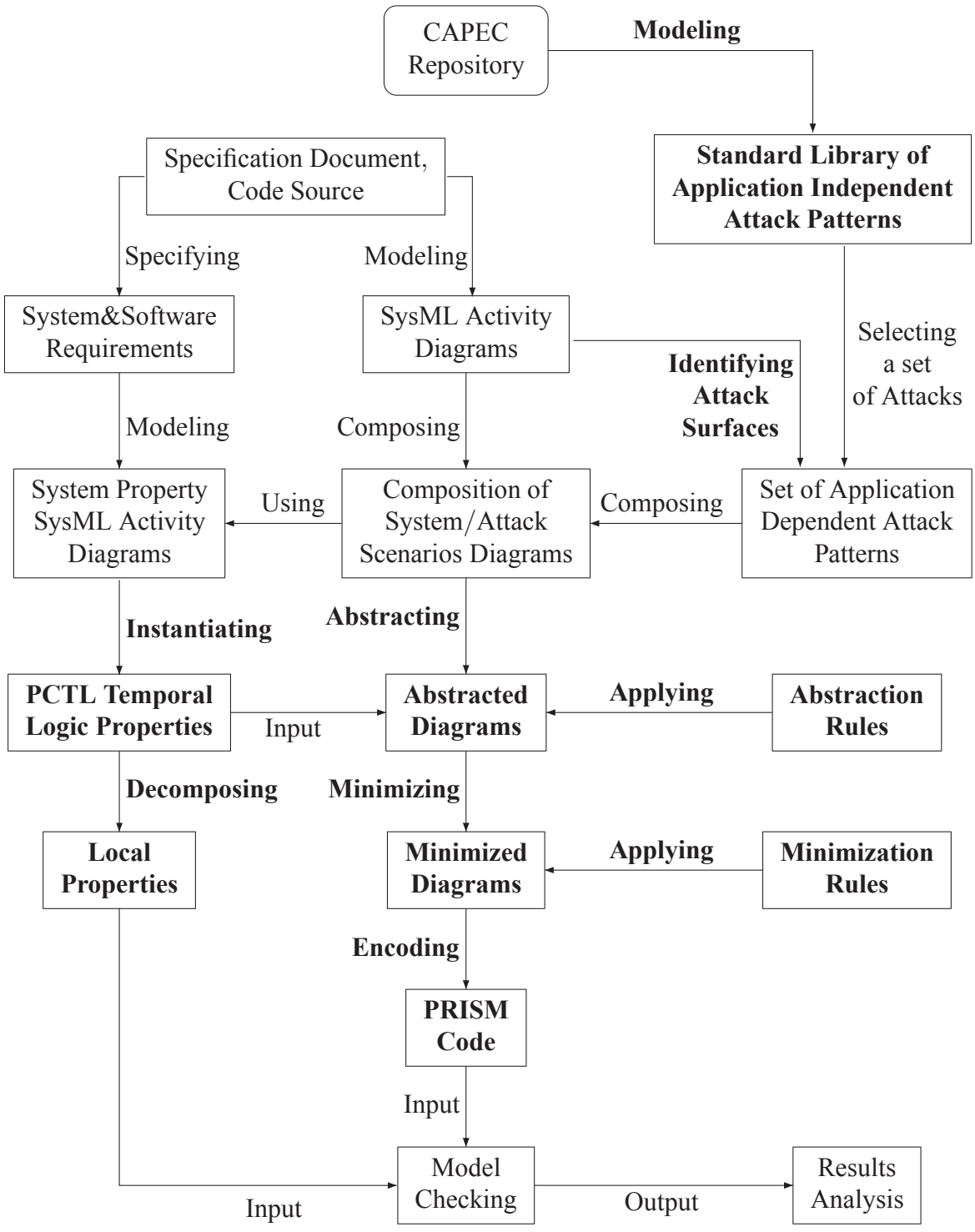


Figure 1.2: Security Verification Framework.

proposed security and attack templates. In addition, it overcomes the model checking limitations by proposing two efficient approaches: abstraction and compositional verification. The proposed framework depicted in Figure 1.2 is based on model checking, and it develops five main concepts: security templates, a standard library of attack patterns, extracting the semantics of the studied diagrams, coding the semantics in Prism input language, abstracting the diagrams, and dividing the property and conquer its satisfiability. The bold font in Figure 1.2 indicates the automatic procedures.

A given SysML activity diagrams can be obtained from a system specification document, or extracted from a source code. First, we use SysML activity diagrams to develop a library of CAPEC attack application-independent templates with varied potential gains that can exploit the system vulnerabilities. By using this library, application-dependent attack scenarios are instantiated and the interaction between the attack and the system diagrams is defined. In order to subject this interaction to the model checker, the security requirements are represented by SysML activity diagrams. Then, a specification algorithm is proposed to generate their equivalent PCTL expressions. Before verification, we propose to reduce the model size of the diagram under verification. First, we introduce a set of abstraction rules that abstracts the diagram with respect to a system requirement. Then, we minimize the abstracted diagram by providing a set of minimization rules. As we deal with a composed diagram, we propose to verify the system locally by distributing a property over each diagram component. For verification, we extract the formal semantics of SysML activity diagrams. This helps to encode easily diagrams into the input language of the PRISM model checker. To ensure the correctness of our proposed framework, we prove the soundness of each step in the framework.

1.5 Thesis Contributions

In this section, we summarize the main contributions in this thesis.

1. **Security verification framework**. We propose a practical and formal framework to specify and verify the security of systems modeled by SysML activity diagrams.

2. **Studying the related work**. We survey the most existing initiatives and tools dedicated to the formal description of UML and SysML behavioral diagrams, the system security specification, and the formal verification.

3. **Formal semantics of SysML activity diagrams**. We formalize SysML activity diagrams by giving an adequate meaning that can be supported by the existing model checking tools. As we study behavioral diagrams, we developed a calculus based on the structural operational semantics.

4. **Formal semantics of PRISM language**. We formalize PRISM model by providing a syntax and operational semantics that support the main operators of PRISM.

5. **Verification**. Our verification mechanism uses the PRISM kernel to check systems requirements. It encodes the semantics of SysML activity diagrams in the input language of PRISM. We prove the soundness of the verification by showing that our encoding preserves the satisfiability of the existing properties.

6. **Attack Modeling**. We propose to use SysML activity diagrams to model the attack standard CAPEC.

7. **Security Specification**. We express security properties by considering SysML activity diagrams as a specification language. We present an algorithm to generate PCTL expressions from SysML activity diagrams.

8. **Abstraction**. We propose to abstract SysML activity diagrams by ignoring the irrelevant behaviors and minimizing their structure. We prove the soundness of the proposed abstraction by showing the preservation of properties satisfiability before and after abstraction.

9. **Compositional Verification**. We propose a compositional verification approach dedicated to SysML activity diagrams. It distributes a property over diagrams which helps to verify the distributed properties locally. We prove the soundness of the property decomposition by showing that the satisfiability of the global property can be deduced from the results of the local properties.

10. **Application**. Our framework is implemented as a java plugin with the eclipse-based TOPCASED toolkit, and it is successfully applied on real case studies.

1.6 Thesis Organization

The rest of the thesis is organized as follows:

- Chapter 2 explores the background needed for our thesis. First, we explore some formal models dedicated to probabilistic system modeling. We present the appropriate temporal logics to specify system requirements. Then, the concepts of non-probabilistic and probabilistic verification procedures are detailed. Also, we show the main techniques needed in verification. Finally, a classification of the existing probabilistic model checking tools is given.

- Chapter 3 is a collection of four papers: [63], [64], [70], [72], and one submitted journal [65]. It presents the probabilistic verification framework of SysML activity diagrams. It is implemented as a frontend with PRISM. We first describe diagrams

as reported by OMG/INCOSE standard. Then, we formalize them by giving the expected semantics. And, we present the semantics of PRISM models that conform to the studied diagrams. Then, our verification framework is detailed and its soundness is proved. Finally, we describe the experimental results and we compare our framework to existing related work.

- Chapter 4 is the result of the following papers: [67], [68], and [66]. The proposed abstraction approach is detailed, and, its soundness is proved. The impact of the abstraction framework is shown by presenting the experimental results and comparing it to the related work.

- Chapter 5 details the compositional verification framework and proves its soundness. First, it shows the efficiency of the composition presenting a promising experimental results. Finally, it discusses existing related work. Chapter 5 is submitted as a conference paper [69].

- Chapter 6 proposes our security specification framework. First, it presents and models the standard attack patterns. Then, it details the security specification algorithm that generates a system requirements by defining the system/attack composition model. Finally, it presents the experimental results and surveys the existing related work. Chapter 6 contains two conference papers: [62] and [71].

- Chapter 7 concludes our work by summarizing the main contributions in the thesis, and discussing the possible future work that are potential research directions.

Chapter 2

Background

2.1 Introduction

Verifying a system using model checking is a three steps procedure: system modeling, system specification, and verification. In system modeling, the system design is converted into a formalism accepted by the used model checker tool. The specification step asserts how the system can behave. Commonly, temporal logic is used to express systems' specifications. The verification process explores exhaustively the state space of the system model to check automatically if the specifications hold or not. When a specification is not satisfied, a counterexample (sequence of states) is produced showing the system failures.

This chapter introduces the main concepts needed in this thesis. In Section 2.2, some formal models dedicated to system design are described. Also, temporal logics for system requirement specification are presented in Section 2.3. Then, the concepts of non-probabilistic and probabilistic verification procedures are detailed in Section 2.4. Also, two main verification techniques are described in Section 2.5. Finally, a classification of the existing probabilistic model checker tools is given in Section 2.6 and we conclude the present chapter in Section 2.7.

2.2 System Models

In the literature, some probabilistic formal models are used for the evaluation of performance and dependability of information-processing systems. Most of them are automaton, Markovian, or PetriNets based models. Mainly, we cite: Markov Decision Processes (MDP), Probabilistic Timed Automata (PTA), Discrete Time Markov Chains (DTMC), Continuous Time Markov Chains (CTMC), Discrete time Markov Reward Models (DMRM), Continuous time Markov Reward Models (CMRM), Continuous Time Markov Decision Processes (CTMDP), Generalized Semi Markovian Process (GSMP), and Stochastic Petri Nets (SPN). Next, we define a transition system and its probabilistic version. Then, we introduce SysML behavioral diagrams as a modeling formalism.

2.2.1 Transition Systems

Transition systems [5, 15, 54] are often used as models to describe systems behaviors. Basically, they are a directed graphs where nodes represent states, and edges model transitions, i.e., state changes. A state describes some information about a system at a certain moment of its execution. Definition 2.1 defines formally a transition system.

Definition 2.1 (Transition System)**.** A transition system is a tuple M=$(\overline{s}, \mathrm{S}, \mathrm{L}, \Sigma, \mathrm{R})$, where:

- $\overline{s}$ is an initial state, such that $\overline{s} \in \mathrm{S}$,
- S is a finite set of states,
- $\mathrm{L} : S \rightarrow 2^{AP}$ is a labeling function that assigns each state to a set of atomic propositions taken from the set of atomic propositions (AP),
- Σ is a finite set of actions,
- $\mathrm{R} \subseteq \mathrm{S} \times \Sigma \times \mathrm{S}$ is a transition relation.

2.2.2 Probabilistic Transition Systems

Probabilistic transition systems extend the transition systems to support the probabilistic decision. More specifically, Probabilistic Automata (PAs) [27] are a modeling formalism for systems that exhibit both probabilistic and nondeterministic features. Definition 2.2 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 p_i to a state s_i.

Definition 2.2 (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 each state to 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 $\mu \in Dist(S)$.

Generally, a system is composed of interacting parts. For PAs, this concept is modeled by the parallel composition as stipulated in Definition 2.3.

Definition 2.3 (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 $M = ((\overline{s}_1, \overline{s}_2),\ \mathrm{S}_1 \times \mathrm{S}_2,\ \mathrm{L}(s_1) \cup \mathrm{L}(s_2),\ \Sigma_1 \cup \Sigma_2,\ \delta)$, where: $\delta(\mathrm{S}_1 \times \mathrm{S}_2, \Sigma_1 \cup \Sigma_2)$ 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 $\alpha \in \Sigma_2 \backslash \Sigma_1$.

As example, we present in Figure 2.1 the probabilistic automata of two selected nodes (Node_0 and Node_1) from the Probabilistic Broadcast Protocol (PBP)[1]. Each node has two atomic propositions: Snd_i and Act_i to describe a local state. The actions in each node are described by m_i.

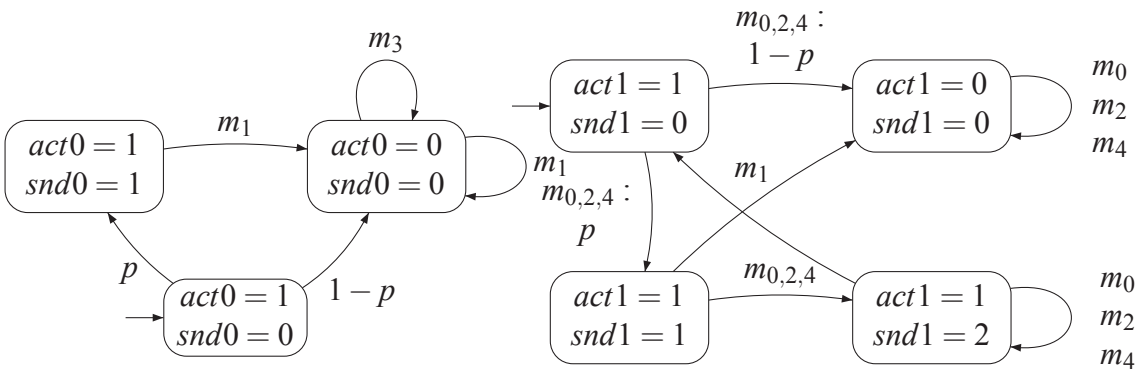


Figure 2.1: The PBP protocol : Node_0 (left) and Node_1 (right).

2.2.3 SysML Behavioral Diagrams

Here, we explore SysML behavioral diagrams and their composition. The diagrams considered are: *Interaction*, *State Machine* and *Activity* behavioral diagrams. The interaction diagram answers the question: " When does who call whom and how?", the state machine diagram answers the question: " How does an object respond to events in a specific state?", and the activity diagram answers the question:" What happens in which sequence?".

Interaction diagram. Interaction is a mechanism to describe an abstract communication between objects of a system. It provides different capabilities that makes it more appropriate for certain situations (i.e, Sequence Diagrams, Interaction Overview Diagrams,

[1] http://www.prismmodelchecker.org/benchmarks

Communication Diagrams, Timing Diagrams and Interaction Tables). SysML uses only the sequence diagram. The main elements of the syntax of an interaction are: lifelines and messages. A lifeline represents a communication role and a message is a communication between two lifelines. For interaction, UML defines branches and loops (alt, opt, break, and loop), concurrency and order (seq, strict, and par), filters and constraints (critical, neg, assert, consider, and ignore) operators are set in a combined fragment.

State Machine. A system in a given time, can be seen as a configuration containing a set of values describing its behavior. The values can be changed when a system reacts to an event. In UML, we have two kinds of state machines: *behavioral state machines* and *protocol state machines*. The first expresses the behavior of the system, and the second describes the protocol of a part of that system. Mainly, a state machine contains a set of states related by transitions. A state is to model a situation where some invariant condition holds. A transition is a directed relationship specifying the system changes between states. The sequence of events can be controlled by the following elements: shallow history, deep history, join, fork, junction, choice, exit point, entry point, and terminate.

Activity Diagram. Activity diagram can be used to model system's behavior at various level of abstractions. It allows low-level modeling compared with other behavioral diagrams [35]. An activity diagram notation can be decomposed into two basic categories: activity nodes and activity edges. There are three types of activity nodes, which are 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 edges. Control flow edges are used to show the execution path through the activity diagram and connects activity nodes that are not object nodes. Object flow edges are used to show the flow of information between activity nodes. More precisely, SysML extends

system features to UML activity diagram by using stereotype. The stereotype is an extensibility mechanism to define or refine the meaning of a model element. Four system features are supported in SysML:

- The «probability» stereotype is applied in two ways: (1) Extension of edges leaving decision nodes, or (2) Object flow edges extension of output parameter sets. It denotes the likelihood that a value will traverse an edge.

- The «rate» stereotype is applied to an activity edge to specify the expected value of the number of objects/values that traverse the edge per time interval.

- The «nobuffer» stereotype is applied to object nodes, tokens arriving at the node are discarded if they are refused by outgoing edges, or refused by actions for object nodes that are input pins.

- The «overwrite» stereotype is applied to object nodes, a token arriving at a full object node replaces the ones already there.

Other features supported in behavioral diagrams include: clocks, delay and action nondeterministic. The *Time* can be modeled as a time reference that other elements may be dependent on. A time reference can be established by a local or global clock that produces continuous or discrete time values. Also, the simple time model in UML can be used to represent the duration of actions in an activity model. The duration can be notated as constraint notes in an activity diagram. The nondeterministic choice can be found in "ConditionalNode" which is a generalization of "StructuredActivityNode" [61]. Herein, two attributes can be defined by the developer "isAssured" and "isDeterminate". The former asserts that at least one test will succeed, and the latter asserts that at most one test will succeed. Both of them are considered false by default.

Composition in UML and SysML behavioral diagrams. In UML as in SysML, a behavioral diagram can call others of the same or a different category. For interaction diagrams, a model can have an association with only one behavioral diagram at a time in a *BehaviorExecutionSpecification* element. It is a kind of *ExecutionSpecification* representing the execution of a behavior. In state machines, a behavioral diagram can be invoked in a state or in a transition. During the transition firing, the associated behavioral diagram with *Effect* will be executed. In a state, a behavior can be triggered in:

1. *Entry*, a behavioral diagram is executed whenever entering the state.
2. *doActivity*, a behavioral diagram is executed while the state is active.
3. *Exit*, a behavioral diagram is executed whenever this state is exited regardless of which transition was taken out of the state. If defined, exit actions are always executed to completion only after all internal activities and transition actions have executed.

For an activity diagram, only one association with another behavior at a time can be applied in the following activity elements:

1. CallBehavior Action: it is a call action that invokes a behavior directly rather than invoking a behavioral feature that, in turn, results in the invocation of that behavior.
2. CallOperation Action: it is an action that transmits an operation call request to the target object, where it may cause the invocation of the associated behavior.
3. DecisionNode: It is a control node that chooses between outgoing flows. It has an association in *decisionInput* to provide input to guard specifications on edges outgoing from the decision node.
4. ObjectFlow: It is an activity edge to model the flow of values to/or from object nodes that can have objects or data passing along them. It can have two associations with

a behavioral diagram in *Selection* object to select tokens from a source object node, and in *Transformation* object to change or replace data tokens flowing along the edge.

5. ObjectNode: It's an abstract activity node that contains only values at runtime that conform to the type of the object node. It has an association with a behavioral diagram in *Selection* to select tokens for outgoing edges.

In Table 2.1, we compare SysML activity diagrams to some formal models with respect to the main system features: Probabilistic Choice (PC), Non-Determinism (ND), Clock, Rate, and Buffer.

Features	MDP	PTA	DTMC	CTMC	DMRM	CMRM	CTMDP	GSMP	SPN	SysML
PC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ND	✓	✓					✓			✓
Clock		✓	✓	✓	✓	✓	✓			✓
Rate				✓			✓			✓
Buffer										✓

Table 2.1: SysML Activity diagrams Vs. Formal Models

2.3 System Requirements Specification

Here, we describe a logic for specifying requirements of transition-based systems. The logic uses atomic propositions and connective operators describing properties in states. For sequences of transitions, temporal logic is used to describe requirements. Two commonly used temporal logics are Computation Tree Logic (CTL) and Linear Temporal Logic (LTL). They differ in how they handle branching in the underlying tree structure. In LTL, operators are intended to describe properties of all possible computation paths, whereas in CTL temporal operators it is possible to quantify over the paths departing from a given state.

2.3.1 Temporal Logic

In temporal logic, time is not mentioned explicitly. For CTL, time is modelled as an infinite tree-like structure with many future paths. The transition system structure can be unwound into a computational tree. Thus, the future is nondeterministic as any of these paths can be considered as the future path. Temporal operators are used to reason over the paths and states of the structure. The operators reason over paths, where A means "along all paths" and E means "along at least one path". The second temporal operators that reason over states, where F means "some future state", G means "all future states", U means "until" and X means "next state". E is dual to $\exists$ as A is to $\forall$ in predicate logic. The syntax of CTL formulas is defined inductively via a Backus-Naur form [5, 9, 15]:

$$\phi \ ::= \ \top \mid ap \mid \phi \wedge \phi \mid \neg\phi \mid \mathrm{EX}\phi \mid \mathrm{E}[\phi \mathrm{U} \phi] \mid \mathrm{A}[\phi \mathrm{U} \phi].$$

where ap ranges over a countable set of atomic formulas. The symbols $\bot$ and $\top$ denote false and true, respectively.

Contrarily to CTL, Time is modelled in LTL as a single infinite future path. Therefore, LTL has no path quantifiers such as A and E of CTL. However, the temporal operators that deal with states of the model are the same as those of CTL. It may seem that LTL is less expressive than CTL. However, this is not true as LTL allows the nesting of boolean connectives and modalities in a way that is not allowed in CTL. The syntax for LTL is expressed in BNF as follows [5, 9, 15]:

$$\phi \ ::= \ \top \mid ap \mid \phi \wedge \phi \mid \neg\phi \mid \mathrm{X}\phi \mid \phi \mathrm{U} \phi.$$

where ap is any atomic proposition. An LTL formula is evaluated on a single path, or on a set of paths. A formula ϕ holds on a set of paths if it holds on every path in the set.

2.3.2 Probabilistic Temporal Logic

To verify probabilistic systems, PCTL [5, 27] is used to express its related specifications. PCTL is a probabilistic version of CTL. The following BNF represents the PCTL syntax.

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

Where "$\top$" means *true*, "ap" is an atomic proposition, $k \in \mathbb{N}$, $p \in [0,1]$, and $\bowtie \in \{<, \leq, >, \geq\}$. "$\wedge$" represents the *conjunction* operator and "$\neg$" is the *negation* operator. "X", "$\mathrm{U}^{\leq k}$", and "U" are the *next*, the *bounded until*, and the *until* temporal logic operators, respectively. Other useful operators can be derived such as:

- $\top \equiv \neg\bot$.
- $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$.
- $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$.
- $\phi_1 \leftrightarrow \phi_2 \equiv \phi_1 \rightarrow \phi_2 \wedge \phi_2 \rightarrow \phi_1$.
- Future: $F\phi \equiv \ \top \mathrm{U} \ \phi$ or $F^{\leq k}\phi \equiv \ \top \mathrm{U}^{\leq k} \ \phi$ and $k \geq 0$.
- Generally: $G\phi \equiv \ \neg(F\neg\phi)$ or $G^{\leq k}\phi \equiv \ \neg(F^{\leq k}\neg\phi)$ and $k \geq 0$.
- $\mathrm{P}_{\geq p}[G\phi] = \mathrm{P}_{\leq 1-p}[F\neg\phi]$.

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 (sequence of states) obtained by a memoryless adversary [27].

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

2.4 Verification Procedures

In this section, we present the verification procedures needed for both non-probabilistic and probabilistic systems.

2.4.1 Non-Probabilistic Verification

For a transition system M, the verification procedure determines the subset of states from S that satisfy a CTL formula ϕ. The algorithm labels the states that satisfy the subformulas of ϕ, then labels sequences satisfying the combined subformulas. As example, Algorithm 1 returns the set of states that satisfies the CTL formula $\mathrm{E}[\phi_1 \mathrm{U} \phi_2]$. First, it looks for the states that satisfy the formula ϕ_2. Then, it searches backward for states satisfying the formula ϕ_1.

Algorithm 1 Procedure of labeling the states satisfying $\mathrm{E}[\phi_1 \mathrm{U} \phi_2]$.

Input: A transition system M, and a CTL property $\phi = \mathrm{E}[\phi_1 \mathrm{U} \phi_2]$.
Output: A set of states $T \in S$ satisfying ϕ.

```
1:  procedure CHECKEU(M, φ1, φ2)
2:      T = {s | f2 ∈ label(s)};
3:      for all s ∈ T do
4:          label(s) = label(s) ∪ {E[φ1 U φ2]};
5:      end for
6:      while T ≠ ∅ do
7:          choose s ∈ T;
8:          T = T \ {s};
9:          for all t such that R(t, s) do
10:             if E[φ1 U φ2] ∉ label(t) and f1 ∈ label(t) then
11:                 label(t) = label(t) ∪ {E[φ1 U φ2]};
12:                 T = T ∪ {t};
13:             end if
14:         end for
15:     end while
16: end procedure
```

2.4.2 Probabilistic Verification

The actual probabilistic model checkers such as PRISM are mainly based on the stochastic version of the classical shortest path problem. This problem was first formulated by Eaton and Zadeh [11] who called it the problem of *pursuit*.

In this section, we describe probability computation in a symbolic model checker. It proceeds by induction on the parse tree of the formula, as in the case of CTL model checking [15]. To show that, we select MDP defined in Definition 2.4 as a special formalism [27] of probabilistic automata that exhibits both probabilistic and nondeterministic behaviors.

Definition 2.4 (Markov Decision Process)**.** A Markov decision process (MDP) is a tuple $M = (\mathrm{S}, \overline{s}, \alpha_{\mathrm{M}}, \delta_{\mathrm{M}}, \mathrm{L})$ where:

- S is a finite set of states,
- $\overline{s}$ is an initial state,

- α_{M} is a finite alphabet,
- $\delta_{\mathrm{M}} : \mathrm{S} \times \alpha_{\mathrm{M}} \rightarrow Dist(S)$ is a (partial) probabilistic transition function,
- $\mathrm{L} : \mathrm{S} \rightarrow 2^{AP}$ is a labeling function mapping each state to a set of atomic propositions taken from a set AP.

To reason formally about MDPs, we need a probabilistic space over it. And, as it is a nondeterministic behavior, the *adversary* notion is introduced to decide which action should be chosen in any state of the MDP. In general, the choice is made depending on the history execution of the MDP. The Definition 2.5 describes the adversary function.

Definition 2.5 (Adversary)**.** An adversary of an MDP M=$(\mathrm{S}, \overline{s}, \alpha_{\mathrm{M}}, \delta_{\mathrm{M}}, \mathrm{L})$ is a function $\sigma : FPath_M \rightarrow Dist(\alpha_M)$ that maps every finite path of the system into a distribution, where:

- $\sigma(\rho)(a) > 0$ only if $a \in A(last(\rho))$,
- $FPath_M$ is a finite set of states,
- $Dist(\alpha_M)$ is a labeled function assigning to each state of the automaton the set of atomic propositions that are true in that state.

Reachability analysis is the kernel of a model checker, and probabilistic reachability refers to the minimum/maximum probability with which a given set of states of a probabilistic system ($T \subseteq S$) can be reached from a particular state (s). To this end, $reach_s(T)$ is the set of paths that start from s and contain a state from T, and $IPath_{M,s}$ defines the infinite paths starting from a state s in M.
$reach_s(T) = \{\pi \in IPath_{M,s} | \pi(i) \in T \text{ and } i \in \mathbb{N}\} = \bigcup_{\rho \in I} \{\pi \in IPath_{M,s} | \pi \text{ has prefix } \rho\}$, where I is the (countable) set of all finite paths from s ending in T, and each element of this union is measurable. This is equivalent to computing the probabilistic bounds of the

reached paths:

$$P^{min}_{M,s}(reach_s(T)) = inf_{\sigma \in Adv_M} Prob^{\sigma}_{M,s}(s,\psi) \quad (2.1)$$

$$P^{max}_{M,s}(reach_s(T)) = sup_{\sigma \in Adv_M} Prob^{\sigma}_{M,s}(s,\psi) \quad (2.2)$$

In fact, finding the probability $x_s = P^{min}_{M,s}(reach_s(T)), s \in S$ is the unique solution of the following linear programming problem:

$$\begin{aligned} \underset{X_s}{\text{maximize}} \quad & \Sigma_{s \in S} x_s \\ \text{subject to} \quad & x_s = 1 && \forall \, \mathrm{s} \in \mathrm{S}^{\mathrm{s}=1}_{\mathrm{min}}, \\ & x_s = 0 && \forall \, \mathrm{s} \in \mathrm{S}^{\mathrm{s}=0}_{\mathrm{min}}, \\ & x_s \leq \sum \delta_M(s,a)(\mathrm{s}') \cdot \mathrm{x}_{\mathrm{s}'} && \forall \, s \notin (S^{s=1}_{min} \cup S^{s=0}_{min}). \end{aligned}$$

In the case of $x_s = P^{max}_{M,s}(reach_s(T)), s \in S$. It solution is similar to the previous and it has the following linear programming problem:

$$\begin{aligned} \underset{X_s}{\text{minimize}} \quad & \Sigma_{s \in S} x_s \\ \text{subject to} \quad & x_s = 1 && \forall \, \mathrm{s} \in \mathrm{S}^{\mathrm{s}=1}_{\mathrm{max}}, \\ & x_s = 0 && \forall \, \mathrm{s} \in \mathrm{S}^{\mathrm{s}=0}_{\mathrm{max}}, \\ & x_s \geq \sum \delta_M(s,a)(\mathrm{s}') \cdot \mathrm{x}_{\mathrm{s}'} && \forall \, s \notin (S^{s=1}_{max} \cup S^{s=0}_{max}). \end{aligned}$$

Bertsekas and Tsitsiklis [11], prove that this minimum is the unique solution for Bellman's equation and the successive approximation methods converge to the optimal vector. This leads to the fact that the linear programming problem can be solved as an equation system problem. This means finding the probability $x_s = P^{min}_{M,s}(reach_s(T)), s \in S$ is the unique

solution of Bellman's equation:

$$X_s = \begin{cases} 1 & \text{if } s \in S_{min}^{s=1} \\ 0 & \text{if } s \in S_{min}^{s=0} \\ \min_{a \in A(s)} \sum \delta_M(s,a)(s') \cdot X_{s'} & \text{Otherwise} \end{cases} \tag{2.3}$$

The Computing reachability probabilities can be computed through three ways: value iteration, the linear programming problem or policy iteration. The first one is the most used methos in practice due to its approximate algorithm that is based on an iterative solution method which corresponds to a fixed point computation. From practical experience, the second approach is more scalable than the first one.

2.5 Verification Techniques

The main challenge in model checking is the state explosion problem. To deal with this problem two approaches exist in literature: Abstraction and Compositional Verification.

2.5.1 Abstraction

Abstraction or compositional minimization refers generally to ignoring some behaviors of the involved system. Two situations may lead to the use of abstraction:

1. Size of system. The size of a system can grow by the presence of many variables, concurrency and clocks.

2. Type of system. The disposed model checkers don't support all systems' features such as buffers, channels, and real variables.

In verification, the abstraction aims to reduce a complex problem $M \models \phi$ to a simpler one $M' \models \phi'$ [9]. Many techniques have been introduced and they can be classified in four categories [9]: abstraction by state merging, abstraction on variables, abstraction by restriction, and observer automata. The first technique aims at merging states of systems that have similar features. Abstraction on variables targets the data in the model and aims at representing a set of values as one symbolic variable. The third category operates by forbidding some behaviors of the system. The method of observer automata restricts the system behaviors to those acceptable by an automaton that observes the system from outside.

2.5.2 Compositional Verification

The main objective of this method is to avoid the construction of the global model of the system. Each component of the system is verified separately, then the global property can be inferred directly. Several approaches have been proposed to compositional reasoning which are: partitioned transition relations, lazy parallel composition, interface processes and assume-guarantee reasoning. The first and the second compute the set of successors (or predecessors) of a state set without constructing the transition relation of the global system. The third one minimizes the global state transition graph by focusing on the communication among the component processes. The fourth one is described by the formulae (1) for a system containing two models M_1 and M_2. Their composition satisfies the property ϕ only if by assuming M_1 satisfies an assumption φ then it satisfies ϕ and if M_2 satisfies φ then the whole system satisfies ϕ [15].

$$\frac{< \varphi > \ M_1 \ < \phi > \qquad < true > \ M_2 \ < \varphi >}{< true > \ M_1 \parallel M_2 \ < \phi >} \ (1)$$

Model Checker	MDP	PTA	DTMC	CTMC	DMRM	CMRM	CTMDP	GSMP	SPN
PRISM	✓		✓	✓					
Modest	✓			✓			✓		
MRMC			✓	✓	✓	✓			
UPPAAL PRO			✓						
LiQuor			✓	✓			✓		
VESTA				✓			✓		✓
Ymer				✓				✓	
SMART			✓	✓					✓

Table 2.2: Model Checkers vs. Supported Formal Models

2.6 Probabilistic Verification Tools

Here, we compare most existing probabilistic model checkers. Our selection was based on two main features: the kind of models that they can support, and the temporal logic that they use to specify a given property.

In the Table 2.2, we compare the model checkers based on their supported model. To complete the comparison, we show in Table 2.3, the supported temporal logic for each tool. The most used temporal logics are mainly of probabilistic and stochastic nature such as Probabilistic Computation Tree Logic (PCTL), Continuous Stochastic Logic (CSL), Probabilistic Reward Computation Tree Logic (PRCTL), and Continuous Stochastic Reward Logic (CSRL). From Table 2.3, we found that only SMART doesn't support a probabilistic temporal logic. It supports only LTL and CTL.

From our perspective, PRISM is the most appropriate tool for our propose. It is open source, and, it supports nondeterminism and probabilistic choice behaviors, also, the properties have extensions for quantitative and costs/rewards specifications. The specifications can be expressed either in the probabilistic computation tree logic (PCTL) [5, 27] or in a continuous stochastic logic. The models can be described using the PRISM language as discrete-time Markov chains, continuous-time Markov chains, Markov decision

Model Checker	CSL	PCTL	PCTL*	PRCTL	CSRL
PRISM	✓	✓	✓		
Modest		✓	✓		✓
MRMC	✓	✓		✓	✓
UPPAALPRO		✓	✓	✓	
LiQuor			✓	✓	✓
VESTA	✓	✓			
Ymer	✓				
SMART					

Table 2.3: Model Checkers vs. Supported Temporal Logic

processes (MDPs) or probabilistic timed automata. PRISM also supports probabilistic automata (PAs), but for simplicity, PRISM refers to PAs by MDPs [27]. For the verification efficiency, the constructed models can be stored as binary decision diagrams (BDDs) and multi-terminal BDDs. In addition, PRISM has built-in symmetry reduction and implements iterative numerical computations to overcome the state explosion problem [5, 27].

2.7 Conclusion

In this chapter, we introduced the main background and concepts needed in the rest of the thesis. Also, we presented a comparison between the existing semantic models, temporal logics, the verification techniques, and existing tools. In the next chapter, we propose a model-checking based methodology for the verification of SysML activity diagrams.

Chapter 3

Verification of SysML Activity Diagrams

3.1 Introduction

In this chapter, we are interested in the formal verification of systems modeled by using SysML activity diagrams. These diagrams can call and communicate with other diagrams, and allow for probabilistic behavior specification. Our proposed verification framework is depicted in Figure 3.1. It takes a composition of SysML activity diagrams and a set of PCTL properties as input. Our framework is based on extracting the formal semantics of SysML activity diagrams. This helps to easily express the diagrams in the input language of the used model checker. From our comparative studies of Chapter 2, we selected PRISM as a verification engine. Herein, we express SysML activity diagrams in PRISM input language. Then, we use PRISM model checker to verify PCTL properties on the obtained PRISM model. To prove the soundness of our verification approach, we compare the underlying semantics of both the SysML activity diagrams and their generated PRISM code. We found that the probabilistic equivalence relation between both semantics preserve the satisfaction of all PCTL operators.

As a motivating example, we present in Figure 3.2 an ATM system to be verified

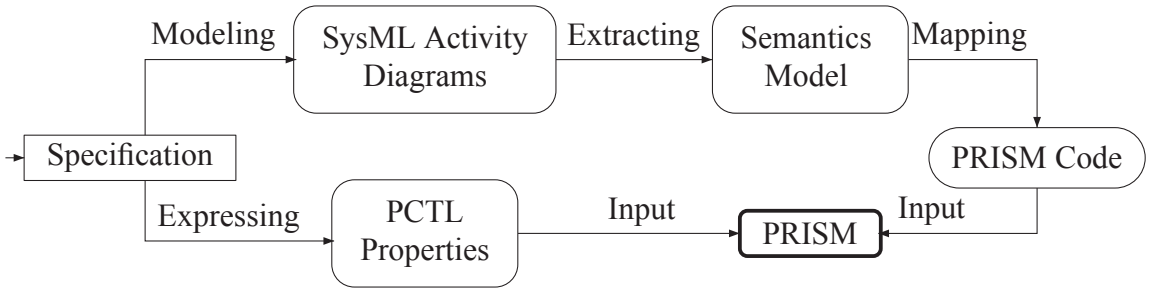
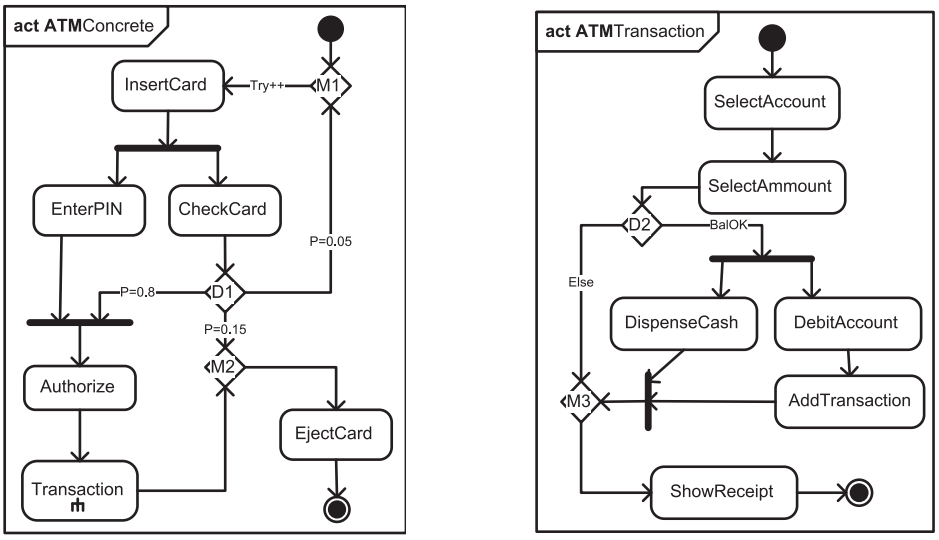


Figure 3.1: A Probabilistic Verification Framework.

using our framework. It represents the ATM SysML activity diagram that is composed of the main diagram "Figure 3.2-(a)" and its call behavior diagram "Figure 3.2-(b)". Our goal is to measure the minimum probability of authorizing a transaction after inserting a card. By using our verification approach, we found the probability value that satisfies this requirement is 0.84.



(a) Main ATM Diagram. (b) Transaction Call Behavior.

Figure 3.2: ATM SysML Activity Diagram.

The remainder of this chapter is organized as follows. Section 3.2 and Section 3.3 describes and formalizes SysML activity diagrams, respectively. The semantics of PRISM models is presented in Section 3.4. Our verification framework is detailed in Section 3.5 and

its soundness is proved in Section 3.6. Section 3.7 describes the experimental results and Section 3.8 surveys the existing related work. Finally, Section 3.9 concludes this chapter.

3.2 SysML Activity Diagrams

SysML activity diagrams are graph-based diagrams where activity nodes are connected by activity edges. They consist of three types of artifacts: activity nodes, activity control nodes, and activity edges. In our work, we take into consideration the artifacts that are illustrated in Figure 3.3.

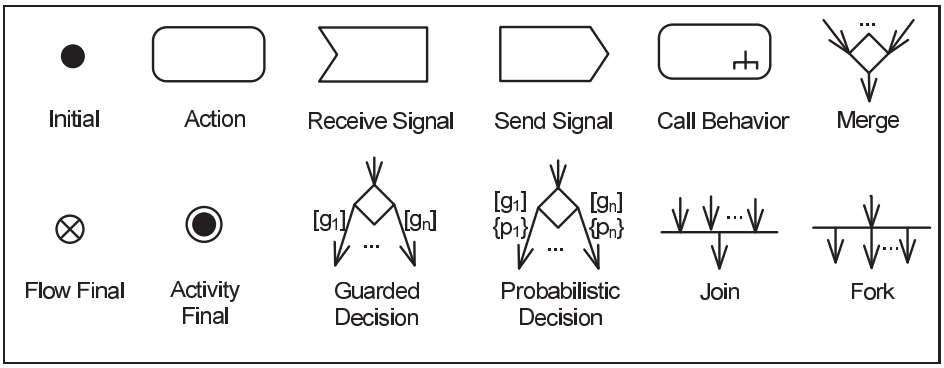


Figure 3.3: SysML Activity Diagram Artifacts.

Activity nodes have three types: activity invocation, object and control nodes. Activity invocation includes receive and send signals (or objects), actions, and call behaviors. Activity control nodes can be 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 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. More specifically, UML 2.0 [61] activity forks model unrestricted parallelism. Thus, a token evolves asynchronously according to an interleaving semantics. 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.

Initially, when a SysML activity diagram is invoked, its initial node is activated. Then, the activation of any other node depends only on the deactivation of its predecessor node and the guard satisfaction of its input edge. In addition, the call behavior action or the decision node can consume its input tokens to invoke its specified behavior. In this case, the invocation supports both synchronous and asynchronous calls. In the asynchronous case, the execution of the invoked behavior proceeds without any further dependency on the execution of the activity containing the invoking artifact. But in the synchronous case, the execution of the calling artifact is blocked until it receives a reply token from the invoked behavior. In the case of the decision node, when the invoked behavior enables more than one guard; the nondeterminism mechanism is adopted.

3.3 SysML Activity Diagram Formalization

In this section, we formalize SysML activity diagrams by providing an adequate calculus that helps to formalize and prove the soundness of our approach.

3.3.1 Syntax of SysML Activity Diagrams

Based on the textual specification in the UML superstructure standard [61] and the SysML specification standard [60], we formalize SysML activity diagrams by developing a calculus called “New Activity Calculus (NuAC)”. In Table 3.1, each SysML activity diagram artifact

is described and represented by its NuAC term.

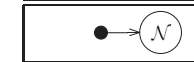
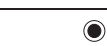
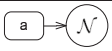
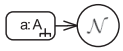
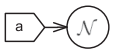
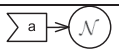
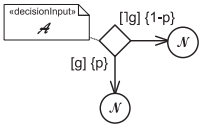
Artifacts	NuAC Terms	Description
	$l : \iota \rightarrowtail \mathscr{N}$	Initial node is activated when a diagram is invoked.
	$l : \odot$	Activity final node stops the diagram execution.
	$l : \otimes$	Flow final node kills its related path execution.
	$l : a \rightarrowtail \mathscr{N}$	Action node defines an atomic action execution.
	$l : a \uparrow \mathscr{A} \rightarrowtail \mathscr{N}$	Call behavior node invokes a new behavior.
	$l : a!v \rightarrowtail \mathscr{N}$	Send node is used to send a signal/object.
	$l : a?v \rightarrowtail \mathscr{N}$	Receive node is used to receive a signal/object.
	$l : D(\mathscr{A}, p, g, \mathscr{N}, \mathscr{N})$	Decision node with a call behavior $\mathscr{A}$, a convex distribution $\{p, 1-p\}$ and guarded edges $\{g, \neg g\}$.
	$l : M(x) \rightarrowtail \mathscr{N}$ or l	Merge node specifies the continuation and $x = \{x_1, x_2\}$ is a set of input flows.
	$l : F(\mathscr{N}_1, \mathscr{N}_2)$	Fork node models the concurrency that begins multiple parallel control threads. UML 2.0 activity forks model unrestricted parallelism.
	$l : J(x) \rightarrowtail \mathscr{N}$ or l	Join node presents the synchronization where $x = \{x_1, x_2\}$ is a set of input pins.

Table 3.1: NuAC Terms of SysML Activity Diagrams Artifacts.

The NuAC syntax presented in Figure 3.4 optimizes the syntax in [18] by eliminating the redundant terms. Also, NuAC exploits the commutativity and the associativity properties for multi-input/output nodes that are described by Property 3.1 and Property 3.2, respectively. These properties allow handling multiplicity by considering only two inputs/outputs. Furthermore, NuAC covers more important behaviors such as: behavior calls, and communication by sending and receiving messages (signals or objects).

Property 3.1 (Commutativity)**.** *In a SysML activity diagram $\mathscr{A}$; fork, join, decision, and merge nodes are commutative.*

- $l : F(\mathscr{N}_1, \mathscr{N}_2) = l : F(\mathscr{N}_2, \mathscr{N}_1)$.

$$
\begin{array}{lcl}
\mathscr{A} & ::= & \varepsilon \quad | \quad l:\overline{\iota}^{n} \rightarrowtail \mathscr{N} \\
\mathscr{N} & ::= & \overline{\mathscr{N}} \quad | \quad l:F(\mathscr{N},\mathscr{N}) \quad | \quad l:D(\mathscr{A},p,g,\mathscr{N},\mathscr{N}) \quad | \\
 & & l \quad | \quad l:\overline{\mathscr{X}}^{n} \rightarrowtail \mathscr{N} \quad | \quad l:\otimes \quad | \quad l:\odot \\
\mathscr{X} & ::= & a\mathscr{B} \quad | \quad J(x_1,x_2) \quad | \quad M(x_1,x_2) \\
\mathscr{B} & ::= & \uparrow \mathscr{A} \quad | \quad !v \quad | \quad ?v \quad | \quad \varepsilon
\end{array}
$$

Figure 3.4: Syntax of New Activity Calculus (NuAC).

- $l:J(x_1,x_2) \rightarrowtail \mathscr{N} = l:J(x_2,x_1) \rightarrowtail \mathscr{N}$.
- $\forall p \in]0,1[,\ l:D(\mathscr{A},p,g,\mathscr{N}_1,\mathscr{N}_2) = l:D(\mathscr{A},1-p,\neg g,\mathscr{N}_2,\mathscr{N}_1)$.
- $l:M(x_1,x_2) \rightarrowtail \mathscr{N} = l:M(x_2,x_1) \rightarrowtail \mathscr{N}$.

Property 3.2 (Associativity)**.** *In a SysML activity diagram $\mathscr{A}$; fork, join, decision, and merge nodes are associative.*

- $l:F(l':F(\mathscr{N}_1,\mathscr{N}_2),\mathscr{N}_3) = l:F(\mathscr{N}_1,l'':F(\mathscr{N}_2,\mathscr{N}_3))$.
- $l:J(x_1,l':J(x_2,x_3)) \rightarrowtail \mathscr{N} = l:J(l'':J(x_1,x_2),x_3) \rightarrowtail \mathscr{N}$.
- $\forall p,p' \in]0,1[,\ l:D(\mathscr{A},p,\mathscr{N}_1,l':D(\mathscr{A}',p',\mathscr{N}_2,\mathscr{N}_3))$
  $= l:D(F(\mathscr{A},\mathscr{A}'),p+p'-p.p',l'':D(\frac{p}{p+p'-p.p'},\mathscr{N}_1,\mathscr{N}_2),\mathscr{N}_3)$.
- $\forall p,p' \in]0,1[,\ l:D(\mathscr{A},p,g,\mathscr{N}_1,l':D(\mathscr{A}',p',g',\mathscr{N}_2,\mathscr{N}_3)) = l:D(F(\mathscr{A},\mathscr{A}'),$
  $(p,g,\mathscr{N}_1),(p'(1-p),\neg g \wedge g',\mathscr{N}_2),((1-p')(1-p),\neg g \wedge \neg g',\mathscr{N}_3)$.
- $l:M(x_1,l':M(x_2,x_3)) \rightarrowtail \mathscr{N} = l:M(l'':M(x_1,x_2),x_3) \rightarrowtail \mathscr{N}$.

In NuAC syntax, we can distinguish two main syntactic concepts: *marked* and *unmarked* terms. A marked NuAC term corresponds to an activity diagram with tokens. But, the 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 "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 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 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 multiple incoming edges (e.g. join and merge) are visited as many times as they have incoming edges. Thus, as a syntactic convention we use only a label (i.e. l) to express a NuAC term if its related node is encountered already. We denote by $D(g, \mathscr{N}, \mathscr{N})$ and $D(p, \mathscr{N}, \mathscr{N})$ to express the guarded and the probabilistic decisions without any behavior invocation.

3.3.2 Semantics of SysML Activity Diagrams

To give a meaning to the execution of SysML activity diagrams, we use structural operational semantics to formally describe how the computation steps of NuAC atomic terms take place. NuAC derivation rules are based on the informally specified locus of control rules defined in the standard [61].

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 include the empty action denoted by ε and p be probability values such that $p \in]0, 1]$. The general form of a transition is $\mathscr{A} \xrightarrow{\alpha}_p \mathscr{A}'$. The probability value specifies the likelihood of a given transition to occur and it is denoted by $P(\mathscr{A}, \alpha, \mathscr{A}')$. The case of $p = 1$ presents a non-probabilistic transition and it is denoted simply by $\mathscr{A} \xrightarrow{\alpha} \mathscr{A}'$. For simplicity, we denote by $\mathscr{A}[\mathscr{N}]$ to specify $\mathscr{N}$ as a sub-term of

$\mathscr{A}$ and by $|\mathscr{A}|$ to specify a term $\mathscr{A}$ without tokens. For the call behavior case of $a \uparrow \mathscr{N}$, we denote $\mathscr{A}[a \uparrow \mathscr{A}']$ by $\mathscr{A} \uparrow_a \mathscr{A}'$. In the sequel, we describe the operational semantic rules of the NuAC calculus.

INT-1 $l : \iota \rightarrowtail \mathscr{N} \xrightarrow{\varepsilon} l : \overline{\iota} \rightarrowtail \mathscr{N}$

This axiom introduces the execution of $\mathscr{A}$ by putting a token on ι.

INT-2 $l : \overline{\iota} \rightarrowtail \mathscr{N} \xrightarrow{l} l : \iota \rightarrowtail \overline{\mathscr{N}}$

This axiom propagates the token in the marked term ι into its outgoing $\mathscr{N}$.

ACT-1 $l : \overline{\overline{a}^m \rightarrowtail \mathscr{N}}^n \xrightarrow{l} l : \overline{\overline{a}^{m+1} \rightarrowtail \mathscr{N}}^{n-1} \quad \forall n > 0,\ m \geq 0$

ACT-2 $l : \overline{\overline{a}^{m+1} \rightarrowtail \mathscr{N}}^n \xrightarrow{l} l : \overline{\overline{\overline{a}^m \rightarrowtail \overline{\mathscr{N}}}}^n \qquad \forall n,\ m \geq 0$

ACT-3 $$\frac{\mathscr{N} \xrightarrow{\alpha}_p \mathscr{N}'}{l : \overline{\overline{a}^m \rightarrowtail \mathscr{N}}^n \xrightarrow{\alpha}_p l : \overline{\overline{a}^m \rightarrowtail \mathscr{N}'}^n} \quad \forall n, m \geq 0$$

The ACT-1 axiom introduces the execution of an action a and ACT-2 shows its execution propagation to the succeeding behavior $\mathscr{N}$. The derivation rule ACT-3 illustrates the evolution of the term $l : \overline{\overline{a}^m \rightarrowtail \mathscr{N}}^n$ when the action α in the term $\mathscr{N}$ is executed with a probability p.

BEH-0 $l : \overline{a \uparrow \mathscr{A}}^n \rightarrowtail \mathscr{N} \xrightarrow{l} l : \overline{\overline{a} \uparrow \mathscr{A}}^{n-1} \rightarrowtail \mathscr{N} \qquad \forall\, n > 0$

BEH-1 $$\frac{\mathscr{A} = l' : \iota \rightarrowtail \mathscr{N}' \qquad \mathscr{A}' = l' : \overline{\iota} \rightarrowtail \mathscr{N}' \qquad \forall\, n > 0}{l : \overline{\overline{a} \uparrow \mathscr{A}}^n \rightarrowtail \mathscr{N} \xrightarrow{l} l : \overline{a \uparrow \mathscr{A}'}^{n-1} \rightarrowtail \mathscr{N}}$$

BEH-2 $$\frac{\mathscr{A}[\overline{l' : \odot}] \xrightarrow{l'} |\mathscr{A}| \qquad \forall n \geq 0}{l : \overline{a \uparrow \mathscr{A}}^n \rightarrowtail \mathscr{N} \xrightarrow{l'} l : \overline{a \uparrow |\mathscr{A}|}^n \rightarrowtail \overline{\mathscr{N}}}$$

BEH-3 $$\frac{\mathscr{A} \xrightarrow{\alpha}_p \mathscr{A}' \qquad \forall n > 0}{l : \overline{a \uparrow \mathscr{A}}^n \rightarrowtail \mathscr{N} \xrightarrow{\alpha}_p l : \overline{a \uparrow \mathscr{A}'}^n \rightarrowtail \mathscr{N}}$$

BEH-4 $$\frac{\mathscr{N} \xrightarrow{\alpha}_p \mathscr{N}' \qquad \forall n \geq 0}{l : \overline{a \uparrow \mathscr{A}}^n \rightarrowtail \mathscr{N} \xrightarrow{\alpha}_p l : \overline{a \uparrow \mathscr{A}}^n \rightarrowtail \mathscr{N}'}$$

The BEH-1 axiom introduces the execution of the behavior $\mathscr{A}$ called by a. The derivation rule BEH-2 finishes the execution of a call behavior and moves the token to the succeeding term $\mathscr{N}$. The derivation rules BEH-3 and BEH-4 present the effect on $\overline{a \uparrow \mathscr{A}}^{n} \rightarrowtail \mathscr{N}$ when $\mathscr{A}$ or $\mathscr{N}$ executes an action α with a probability p, respectively.

FRK-1 $l : \overline{F(\mathscr{N},\mathscr{N})}^{n} \xrightarrow{l} l : \overline{F(\overline{\mathscr{N}},\overline{\mathscr{N}})}^{n-1} \quad \forall n > 0$

FRK-2 $$\frac{\mathscr{N} \xrightarrow{\alpha}_{p} \mathscr{N}' \quad \forall n \geq 0}{l : \overline{F(\mathscr{N},\mathscr{N})}^{n} \xrightarrow{\alpha}_{p} l : \overline{F(\mathscr{N}',\mathscr{N})}^{n}}$$

The FRK-1 axiom shows the multiplicity of the arriving tokens according to the outgoing sub-terms. The FRK-2 derivation rule illustrates the changes on a fork term when its outgoing execute an action α with a probability p.

DEC-1 $l : \overline{D(g,\mathscr{N},\mathscr{N})}^{n} \xrightarrow{l,g} l : \overline{D(g,\overline{\mathscr{N}},\mathscr{N})}^{n-1} \quad \forall n > 0$

DEC-2 $l : \overline{D(p,g,\mathscr{N},\mathscr{N})}^{n} \xrightarrow{l,g}_{p} l : \overline{D(p,g,\overline{\mathscr{N}},\mathscr{N})}^{n-1} \quad \forall\, n > 0$

DEC-3 $$\frac{\mathscr{A} = l' : \iota \rightarrowtail \mathscr{N}' \quad \mathscr{A}' = l' : \overline{\iota} \rightarrowtail \mathscr{N}' \quad \forall \quad n > 0}{\overline{l : D(\mathscr{A},p,g,\mathscr{N},\mathscr{N})}^{n} \xrightarrow{l} \overline{l : D(\mathscr{A}',p,g,\mathscr{N},\mathscr{N})}^{n-1}}$$

DEC-4 $$\frac{\mathscr{A}[\overline{l' : \odot}] \xrightarrow{l'} |\mathscr{A}| \quad \forall \quad n > 0}{\overline{l : D(\mathscr{A},p,g,\mathscr{N},\mathscr{N})}^{n} \xrightarrow{l',g}_{p} \overline{l : D(|\mathscr{A}|,p,g,\overline{\mathscr{N}},\mathscr{N})}^{n}}$$

DEC-5 $$\frac{\mathscr{N} \xrightarrow{\alpha}_{p} \mathscr{N}' \quad \forall n > 0}{\overline{l : D(\mathscr{A},p,g,\mathscr{N},\mathscr{N}'')}^{n} \xrightarrow{\alpha}_{p} \overline{l : D(\mathscr{A},p,g,\mathscr{N}',\mathscr{N}'')}^{n}}$$

The axiom DEC-1 describes a guarded decision where a token flows through the edge satisfying its guard g. Contrary, DEC-2 describes a probabilistic decision where a token flows with a probability p through the edge satisfying its related guard g. DEC-3 axiom shows a transition of probability one to initiate an invoked behavior. DEC-4 derivation rule shows the termination of a behavior with a transition of probability one and shows how a token can flow from a behavior call execution to a guarded path

(or unguarded) with a probability value (or without probability value). DEC-6 shows the evolution of a decision term when one of its behaviors is changed.

MRG-1 $l : \overline{\overline{\mathscr{N}} \rightarrowtail l' : M(x,y)}^{n} \xrightarrow{l} l : \overline{\mathscr{N} \rightarrowtail l' : M(\overline{x},y)}^{n} \quad \forall\, n \geq 0$

MRG-2 $l : M(\overline{x},\overline{y}) \rightarrowtail \mathscr{N} \xrightarrow{l} l : M(x,\overline{y}) \rightarrowtail \overline{\mathscr{N}}$

MRG-3 $\mathscr{A}[l : M(x,y) \rightarrowtail \mathscr{N}, \overline{l_x}] \xrightarrow{o} \mathscr{A}[l : M(\overline{x},y) \rightarrowtail \mathscr{N}, l_x]$

MRG-4 $\dfrac{\mathscr{N} \xrightarrow{\alpha}_p \mathscr{N}'}{\overline{l : M(x,y) \rightarrowtail \mathscr{N}} \xrightarrow{\alpha}_p \overline{l : M(x,y) \rightarrowtail \mathscr{N}'}}$

MRG-1 is a transition with a probability of value 1 to put a token coming from the sub-term $\mathscr{N}$ on a merge labeled by l. MRG-2 is a transition with a probability of value 1 to present a token flowing from a merge node labeled by l to the sub-term $\mathscr{N}$. MRG-2 fuses labels referring to the same merge node. The derivation rule MRG-3 presents the subsequence of $l : M(x,y) \rightarrowtail \mathscr{N}$ when $\mathscr{N}$ executes an action α with a probability of value p.

JON-1 $l : \overline{\overline{\mathscr{N}} \rightarrowtail l' : J(x,y)}^{n} \xrightarrow{l} l : \overline{\mathscr{N} \rightarrowtail l' : J(\overline{x},y)}^{n} \quad \forall\, n \geq 0$

JON2 $l : J(\overline{x},\overline{y}) \rightarrowtail \mathscr{N} \xrightarrow{l} l : J(x,y) \rightarrowtail \overline{\mathscr{N}}$

JON-3 $\mathscr{A}[l : J(x,y) \rightarrowtail \mathscr{N}, \overline{l_x}] \xrightarrow{l} \mathscr{A}[l : J(\overline{x},y) \rightarrowtail \mathscr{N}, l_x]$

JON-4 $\dfrac{\mathscr{N} \xrightarrow{\alpha}_p \mathscr{N}'}{\overline{l : J(x,y) \rightarrowtail \mathscr{N}} \xrightarrow{\alpha}_p \overline{l : J(x,y) \rightarrowtail \mathscr{N}'}}$

JON-1 and JON-2 are axioms representing a transition with a probability of value 1 to activate the pin x in a join labeled by l and to move a token in join to the sub-term $\mathscr{N}$, respectively. JON-3 fuses labels referring to the same join. The derivation rule JON-4 presents the subsequence of $\overline{l : J(x,y) \rightarrowtail \mathscr{N}}$ when $\mathscr{N}$ executes an action α with a probability p.

SND $\overline{l:a!v}^{n} \rightarrowtail \mathscr{N} \xrightarrow{l} \overline{l:a!v}^{n-1} \rightarrowtail \overline{\mathscr{N}} \quad \forall n > 0$

SND describes the evolution of the token after sending an object v.

REC $$\frac{\overline{l':a!v}^{m} \rightarrowtail \mathscr{N}' \xrightarrow{l'} \overline{l':a!v}^{m-1} \rightarrowtail \overline{\mathscr{N}'} \qquad \forall n,\ m \geq 1}{\overline{l:a?v}^{n} \rightarrowtail \mathscr{N} \xrightarrow{l} \overline{l:a?v}^{n-1} \rightarrowtail \overline{\mathscr{N}}}$$

REC describes the reception of an object v just after sending it which is a synchronization communication.

COM $$\frac{\overline{l:a!v}^{n} \rightarrowtail \mathscr{N} \xrightarrow{l} \overline{l:a!v}^{n-1} \rightarrowtail \overline{\mathscr{N}} \qquad \overline{l':a?v}^{m} \rightarrowtail \mathscr{N}_2 \xrightarrow{l'} \overline{l':a?v}^{m-1} \rightarrowtail \overline{\mathscr{N}_2}}{\mathscr{A}[\overline{l:a!v}^{n} \rightarrowtail \mathscr{N}, \overline{l':a?v}^{m-1} \rightarrowtail \overline{\mathscr{N}_2}] \longrightarrow \mathscr{A}[\overline{l:a!v}^{n} \rightarrowtail \mathscr{N}_1, \overline{l':a?v}^{m-1} \rightarrowtail \overline{\mathscr{N}_2}]}$$

COM describes sending and receiving an object v.

FFin $\mathscr{A}[l:\overline{\bigotimes}] \xrightarrow{l} \mathscr{A}[l:\bigotimes]$

This axiom states that if the sub-term $l:\bigotimes$ is reached in $\mathscr{A}$ then a transition of probability one is enabled to produce a term describing the termination of a flow.

AFin $\mathscr{A}[l:\overline{\odot}] \xrightarrow{l} |\mathscr{A}|$

This axiom states that if the sub-term $l:\odot$ is reached then no action is taken later by destroying all tokens.

PRG1 $$\frac{\mathscr{N} \xrightarrow{\alpha}_{p} \mathscr{N}'}{\mathscr{A}[\mathscr{N}] \xrightarrow{\alpha}_{p} \mathscr{A}[\mathscr{N}']}$$

PRG2 $$\frac{\mathscr{N}_1 \xrightarrow{\alpha_1}_{p_1} \mathscr{N}_1' \qquad \mathscr{N}_2 \xrightarrow{\alpha_2}_{p_2} \mathscr{N}_2'}{\mathscr{A}[\mathscr{N}_1, \mathscr{N}_2] \xrightarrow{\alpha}_{p_1 \times p_2} \mathscr{A}[\mathscr{N}_1', \mathscr{N}_2']}$$

The ACT3, BEH-3, BEH-4, FRK2, DEC5, MRG4, JON4, PRG1 derivation rules preserve the evolution when a sub-term $\mathscr{N}$ evolves to $\mathscr{N}'$ by executing the action α with a probability p. The PRG2 derivation rule describes the interleaving between two terms $\mathscr{N}_1$ and $\mathscr{N}_2$.

The semantics of SysML activity diagrams expressed using $\mathscr{A}$ is the result of the defined inference rules. The NUAC semantics can be described in terms of a PA as stipulated by Definition 3.1.

Definition 3.1 (NuAC-PA)**.** A probabilistic automaton of an activity calculus term $\mathscr{A}$ is a tuple $M_{\mathscr{A}} = (\overline{s},\ \mathrm{L},\ \mathrm{S},\ \Sigma, Steps)$, where:

- $\overline{s}$ is an initial state, such that $\mathrm{L}(\overline{s})$=$l : \overline{\tau} \rightarrowtail \mathscr{N}$,
- $\mathrm{L} : S \rightarrow 2^{\{\mathscr{A}\}}$ is a labeling function that assigns for each state an $\mathscr{A}$ marked term,
- S is a finite set of states reachable from $\overline{s}$, such that, $\mathrm{S} = \{s_{i:0 \leq\ i\ \leq\ n} | \mathrm{L}(s_i) \in \{\mathscr{A}\}\}$,
- Σ is a finite set of actions corresponding to the labels in $\mathscr{A}$,
- $Steps : \mathrm{S} \times \Sigma \rightarrow 2^{Dist(S)}$ is a (partial) probabilistic transition function such that:
  - $\forall s \in S$, and $\alpha \in \Sigma$: $Steps(s) =$ $\{\langle \alpha, \mu \rangle | s \xrightarrow{\alpha} \mu\}$, $Steps_\alpha(s) = \{\mu \in Dist(S) | \langle \alpha, \mu \rangle \in Steps(s)\}$ where $Dist(S)$ is a set of convex probability distributions over S.
  - If $Steps(s) = \emptyset$ then s is the terminal state.

3.4 PRISM Formalization

In this section, we formalize PRISM by presenting its syntax and semantics. In our formalization, we focus more on the probabilistic automata model in PRISM.

Generally, a probabilistic system "*S*" that is described as a PRISM program "*P*" that comprises a set of n modules ($n > 0$), the state of each one is defined by the evaluation of a countable set of finite-ranging local variables. The global state of the system is the evaluation of the local variables (V_l) and the global ones (V_g) denoted by $V = V_g \cup V_l$. The behavior of each module is a set of guarded commands.

A guarded command describes the main changes of P behaviors. It takes the following form: [a] g $\rightarrow p_1 : u_1$+...+$p_m : u_m$, or, [a] g $\rightarrow u$, which means, for the action

"a" if the guard "g" is true, then, an update "u_i" is enabled with a probability "p_i". For the second case, for the action "a" if the guard "g" is true, then, the update "u" is enabled. A guard is a logical proposition consisting of variables evaluation and propositional logic operators. The update "u_i" is an evaluation of variables expressed as a conjunction of assignments: $(v'_j = val_j)\&\cdots\&(v'_k = val_k)$ where v_i are local variables and val_i are values evaluated via expressions denoted by "*eval*" that requires type consistency ($eval : V_l \rightarrow \mathbb{N} \cup \{true, false\}$). A command is formally defined in Definition 3.2.

Definition 3.2 (PRISM Command)**.** A PRISM command is a tuple $c = (a, g, u)$, where:

- a: is an action label,
- g: is a predicate over V,
- $u = \{(p_i, u_i) | m > 1, 0 < p_i < 1, \sum_{i=1}^{m} p_i = 1 \text{ and } u_i = \{(v, eval(v)) : v \in V_l\}\}$
  $\cup \{(v, eval(v)) : v \in V_l\}$.

A module that describes the behavior of a sub-part of a system can be considered as a set of commands. The variables of each module are declared and initialized locally. A module is formally defined in Definition 3.3.

Definition 3.3 (PRISM Module)**.** A PRISM module "M" is a tuple $M = (V_l, I_l, C)$, where:

- V_l is a finite set of local variables associated to the module M,
- I_l is the initial values of V_l,
- $C = \{c_i : 0 \leq i \leq k\}$ is a finite set of commands that defines the behavior of the PRISM module M.

To describe the composition between modules, PRISM uses the following Communicating Sequential Processes (CSP) [34] operators.

1. Synchronization: It is a parallel composition of modules. For two modules M_1 and M_2, their synchronization is denoted by $M_1||M_2$ and they can synchronize only on actions appearing in both M_1 and M_2.

2. Interleaving: It is an asynchronous parallel composition of modules that are fully interleaved without synchronization. M_1 interleaves with M_2 is denoted by $M_1|||M_2$.

3. Parallel Interface: It is a restricted parallel composition of modules. The modules synchronize only on shared actions. For example, let $\{a,b,\cdots\}$ be the set of shared actions between M_1 and M_2, the interface parallel composition of M_1 and M_2 in $\{a,b,\cdots\}$ is denoted by: $M_1|[a,b,\cdots]|M_2$.

Other useful CSP operators supported by PRISM are hiding and renaming:

1. Hiding: This operation permits to hide actions in a module. We denote by $M/\{a,b,\cdots\}$ to hide actions a, b, $\cdots$ in the module M.

2. Renaming: This operator facilitates rewriting the behavior of a module by renaming its actions. We denote by $M\{a \leftarrow b, c \leftarrow d, \cdots\}$ to rename actions a by b, c by d, $\cdots$ in the module M.

As a result, Definition 3.4 stipulates formally a system containing n modules and combined by a CSP algebraic expression.

Definition 3.4 (PRISM System)**.** A PRISM system is a tuple $P = (V, I, exp, M_1, \ldots, M_n, CSPexp)$, where:

- $V = V_G \bigcup_{i=1}^{n} V_{li}$ is a finite set of the union of global and local variables,
- $I = I_G \bigcup I_l$ is a finite set of the initial values of global (I_G) and local (I_l) variables,
- exp is a set of global logic expressions,

- $M_1, \ldots, M_n$ is a countable set of modules,
- *CSPexp* is a CSP algebraic expression.

3.4.1 PRISM Syntax

The PRISM syntax of a probabilistic automata is defined by the BNF grammar presented in Figure 3.5. To clarify PRISM syntax, we define the following:

- [min..max] is a range of values such that min, max $\in \mathbb{N}$ and min $<$ max.
- $p \in]0..1[$ is a probability value.
- *eval* is an evaluation expression that can be composed of the following operators: $-(\textit{unary minus}), +, -, *, /, <, <=, >=, >, =, ==, !, \&, |, <=>, =>, ?(g\ ?\ a : b)$.
- $val \in \mathbb{N} \bigcup \{true, false\}$ is a value given by the function *eval*.
- v is a string describing a variable ($v \in V$) and $init(v)$ is its initial value.
- *name* is a string label describing the module name. For the module i, its name label is denoted by M_i.
- *CSPexp* is a CSP expression composed of the following operators $||, |||, |[a, b, \cdots]|$, $/\{a, b, \cdots\}$, and $\{a \leftarrow b, c \leftarrow d, \cdots\}$.

3.4.2 PRISM Semantics

The probabilistic automata of a PRISM program P is based on the atomic semantics of a command "c" denoted by $[\![c]\!]$. The latter is a set of transitions defined as follows: $[\![c]\!] = \{(s, a, \mu) \mid s \models g\}$ where μ is a distribution over S such that $\mu(s') = \{|0 < p_i \leq 1 \ : \forall v \in V,\ s'(v) = eval_i(V)|\}$.

$\mathscr{P}$	::= **MDP** < *Variables* > < *eval* > < *Modules* > < *System* >
< *Variables* >	::=ε
	| < *kindVariables* > *v*: < *VariablesType* > init *init*(*v*);< *Variables* >
< *kindVariables* >	::=ε |global
< *VariablesType* >	::= bool | int | [min..max]
< *Modules* >	::= **module** *name* < *Variables* > < *ModuleBehavior* > **endmodule**
< *ModuleBehavior* >	::=ε |[a] g→< *update* > ;< *ModuleBehavior* >
< *update* >	::=< *p* >< *eval* >; | < *p* >< *eval* > + < *update* >
< *eval* >	::=($v' = eval(V)$) | (*v*'=*eval*(*V*))&< *eval* >
< *p* >	::=ε | *p* :
< *System* >	::=**system** < *AlgebraExpressions* > **endsystem**
< *AlgebraExpressions* >	::= ε | *name CSPexp name*;< *AlgebraExpressions* >

Figure 3.5: The Syntax of PRISM Probabilistic Automata.

Definition 3.5 stipulates the formal definition of PRISM probabilistic automata denoted by M_P. The states of M_P take the form $\langle V_1, \cdots, V_n, eval \rangle$. The stepwise behavior of M_P is described by the operational semantic rules provided as follows.

INIT $\langle V_i, init(V_i) \rangle \longrightarrow \langle V_i([\![init(V_i)]\!]), - \rangle$

INIT initializes variables. For a module M_i, init returns the initial value of the local variable $v_j \in V_i$.

LOOP $\langle V_i, - \rangle \longrightarrow \langle V_i \rangle$

This axiom presents a loop in a state without changing variables' evaluations. It can be applied to avoid a deadlock.

UPDATE $\langle V_i, v_i' = eval(V) \rangle \longrightarrow \langle V_i([\![v_i]\!]) \rangle$

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 $\langle V_i, v_i' = eval(V) \wedge v_j' = eval(V) \rangle \longrightarrow \langle V_i([\![v_i]\!], [\![v_j]\!]) \rangle$

CNJ-UPD implements the conjunction of a set of assignments.

PRB-UPD1 $\langle V_i, p_i : v_i' = eval(V) \rangle \longrightarrow_{p_i} \langle V_i([\![v_i]\!]) \rangle \quad 0 < p_i \leq 1$

PRB-UPD2 $\langle V_i, p_i : v_i' = eval(V) \wedge v_j' = eval(V)\rangle \longrightarrow_{p_i} \langle V_i([\![v_i, v_j]\!])\rangle 0 < p_i \leq 1$

PRB-UPD1 and PRB-UPD2 describe probabilistic updates.

ENB-CMD1 $$\frac{V \models g}{\langle V, M([a]g \rightarrow \sum_i p_i : u_i)\rangle \xrightarrow{a} \mu}$$

ENB-CMD1 enables the execution of a probabilistic command.

ENB-CMD2 $$\frac{V \models g \quad V \not\models g'}{\langle V, [a]g \rightarrow u; [a']g' \rightarrow u'\rangle \xrightarrow{a} \langle V([\![u]\!]), [a']g \rightarrow u'\rangle}$$

ENB-CMD2 enables the execution of a command in a module.

ENB-CMD3 $$\frac{V \models g \wedge g'}{\langle V, [a]g \rightarrow u; [a']g \rightarrow u'\rangle \longrightarrow \langle V([\![u]\!]), [a']g' \rightarrow u'\rangle}$$

ENB-CMD3 solves the nondeterminism in a module by following a policy.

SYNC $$\frac{\langle V_i, c_i\rangle \xrightarrow{a} \mu_i \quad \langle V_j, c_j\rangle \xrightarrow{a} \mu_j}{\langle V_i \cup V_j, M_i || M_j\rangle \xrightarrow{a} \mu_i.\mu_j}$$

SYNC derivation rule permits the synchronization between modules on the action labeled by "a".

INTERL $$\frac{\langle V_i, M_i(c_j)\rangle \xrightarrow{a_j} \mu}{\langle V, M_i ||| M_j\rangle \xrightarrow{a_j} \mu}$$

INTERL derivation rule describes the interleaving between modules.

Definition 3.5 (PRISM-PA)**.** A probabilistic automaton of a PRISM program P is a tuple $M_P = (s_i,\ \mathrm{S},\ \mathrm{L},\ \Sigma,\ \delta)$ where:

- s_i is an initial state, such that $\mathrm{L}(s_i)$=$[\![init(V)]\!]$,
- S is a finite set of states reachable from $\overline{s}$, such that, $\mathrm{S} = \{s_{i:0 \leq\ i\ \leq\ n} | \mathrm{L}(s_i) \in \{\mathrm{AP}\}\}$,
- $\mathrm{L} : S \rightarrow 2^{AP}$ is a labeling function that assigns for each state a set of valuated propositions,
- Σ is a finite set of actions,

- $\delta : \mathrm{S} \times \Sigma \rightarrow \ Dist(S)$ is a (partial) probabilistic transition function assigning for each $s \in S$ and $\alpha \in \Sigma$ a probabilistic distribution $\mu \in Dist(S)$. μ is a convex combination of distributions. $Distr(S)$ denotes the set of (sub) distributions over S. δ defines non deterministic and probabilistic steps as follows: Steps(s)=$\{\langle a, \mu \rangle : s \xrightarrow{a} \mu\}$ and $Steps_a(s) = \{\mu \in Dist(S) : \langle a, \mu \langle \in Steps(s)\}$.

3.5 The Verification of SysML Activity Diagrams

This section describes the transformation of SysML activity diagrams $\mathscr{A}$ into a PA written in PRISM input language. Algorithm 2 illustrates the transformation algorithm T that takes $\mathscr{A}$ as input and returns its PRISM code image denoted by *PrismCode*. The diagram $\mathscr{A}$ is visited using a depth-first search procedure and the algorithm's output produces PRISM synchronized modules. The algorithm is described as follows. First, the initial node is pushed into the stack of nodes denoted by *nodes* (line 5). While the stack is not empty (line 6-17), the algorithm pops a node from the stack into the current node denoted by *cNode* (line 7). The current node is added into the list *vNode* of visited nodes (line 9) if it is not already visited (line 8). *PrismCode* is constructed by calling the function Γ that has two arguments which are the current node and its successors (line 11). The explored successors are pushed into the stack *nodes* (line 13-15). The algorithm terminates the execution when all nodes are visited.

The mapping function Γ presented in Listing 3.1produces the appropriate PRISM command for the current node. The action label of the command is the label of the current node. The guard of this command depends on how the current node can be activated, therefore, a boolean expression as a flag is assigned to define this activation. The variables of each module are locals of type boolean initialized to false except for the initial node that is initialized to true. It marks the first token produced by the rule "INT-1". Generally, the

Algorithm 2 Transformation Algorithm T of SysML Activity Diagrams into PRISM Code

Input: SysML activity diagrams $\mathscr{A}$.
Output: PRISM code *PrismCode*.

```
1:  nodes as Stack;                          ▷ A stack of nodes which is initially empty.
2:  cNode as Node;                           ▷ The current node which is initially empty.
3:  nNode, vNode as list_of_ Node;           ▷ List of nodes that are initially empty.
4:  procedure T(𝒜)
5:      nodes.push(in);                      ▷ Read the initial node.
6:      while not nodes.empty() do
7:          cNode := nodes.pop();            ▷ Pop the current node.
8:          if cNode not in vNode then
9:              vNode.add(cNode);            ▷ Consider the current node as a visited node.
10:             nNode := cNode.successors(); ▷ Get the successors of the current node.
11:             PrismCode.add(Γ(cNode,nNode)); ▷ Call the mapping rules function.
12:         end if
13:         for all n in nNode do            ▷ Stores all newly discovered nodes in the stack.
14:             nodes.push(n);
15:         end for
16:         nNode.clear();                   ▷ Empty the list nNode.
17:     end while
18: end procedure
```

updates deactivate the propositions of the current node and activate that the ones related to its successors. For a node n, we define three useful functions: $L(n)$, $S(n)$ and $E(n)$ that return the label, the start and the end of its related call behaviors, respectively.

Now, we calculate the time complexity of the algorithm T for a SysML activity diagram $\mathscr{A}$ of n nodes (we consider n as the maximum number of nodes supported by $\mathscr{A}$). In Algorithm 2, the while loop can run at most n times (line 6). Recursively, Γ can recall the algorithm T when a call behavior node exists. This mechanism of recalling produces a hierarchical form of diagrams where each node can call a new SysML activity diagram. We introduce the notion of hierarchy depth denoted by k, which means the level in the hierarchy from where a node has a call behavior. The worst case is that for any level of the hierarchy, each node can call a new behavior. The computing complexity of T is of class P with a worst case running time of $\mathscr{O}(n^k)$.

$\Gamma\colon\ \mathscr{A} \mapsto P$

$\Gamma(\mathscr{A}) = \forall n \in \mathscr{A}\text{, Case } (n) \textbf{ of}$

$\quad l : \iota \rightarrowtail \mathscr{N} \Rightarrow$

$\quad\quad \textbf{in}$

$\quad\quad\quad \{[l]l \longrightarrow (l' = \bot)\&(L(\mathscr{N})' = \top);\} \cup \Gamma(\mathscr{N})$

$\quad\quad \textbf{end}$

$\quad l : M(x,y) \rightarrowtail \mathscr{N} \Rightarrow$

$\quad\quad \textbf{in}$

$\quad\quad\quad \{[l_x]l_x \longrightarrow (l'_x = \bot)\&(L(\mathscr{N})' = \top);\}$

$\quad\quad\quad \cup\{[l_y]l_y \longrightarrow (l'_y = \bot)\&(L(\mathscr{N})' = \top);\} \cup \Gamma(\mathscr{N})$

$\quad\quad \textbf{end}$

$\quad l : J(x,y) \rightarrowtail \mathscr{N} \Rightarrow$

$\quad\quad \textbf{in}$

$\quad\quad\quad \{[l]l_x \wedge l_y \longrightarrow (l'_x = \bot)\&(l'_y = \bot)\&(L(\mathscr{N})' = \top);\} \cup \Gamma(\mathscr{N})$

$\quad\quad \textbf{end}$

$\quad l : F(\mathscr{N}_1, \mathscr{N}_2) \Rightarrow$

$\quad\quad \textbf{in}$

$\quad\quad\quad \{[l]l \longrightarrow (l' = \bot)\&_{1 \le i \le 2}(L(\mathscr{N}_i)' = \top);\}$

$\quad\quad\quad \bigcup_{1 \le i \le 2} \Gamma(\mathscr{N}_i)$

$\quad\quad \textbf{end}$

$l : a\mathscr{B} \rightarrowtail \mathscr{N}\text{, Case } (\mathscr{B}) \textbf{ of}$

$\quad \uparrow \mathscr{A} \Rightarrow$

$\quad\quad \textbf{in}$

$\quad\quad\quad \{[l_s]l \rightarrow (l' = \top);\}$

$\quad\quad\quad \cup\{[l_e]l\&L(E(\mathscr{A})) \rightarrow (l' = \bot)\&(L(\mathscr{N})' = \top);\}$

$\quad\quad\quad \cup\Gamma'(\mathscr{A});$

$\quad\quad \textbf{end}$

$\quad !v \Rightarrow$

$\quad\quad \textbf{in}$

$\quad\quad\quad \{[l]l \longrightarrow (l' = \bot)\&(L(\mathscr{N})' = \top);\} \cup \Gamma(\mathscr{N}')$

$\quad\quad \textbf{end}$

$\quad ?v \Rightarrow$

$\quad\quad \textbf{in}$

$\quad\quad\quad \{[L(a?v)]l\&\neg L(a!v) \longrightarrow (l' = \top);\}$

$\quad\quad\quad \cup\{[L(a!v)]l\&L(a!v) \longrightarrow (l' = \bot)\&(L(\mathscr{N})' = \top);\} \cup \Gamma(\mathscr{N}')$

$\quad\quad \textbf{end}$

$\quad \varepsilon \Rightarrow$

$\quad\quad \textbf{in}$

$\quad\quad\quad \{[l]l \longrightarrow (l' = \bot)\&(L(\mathscr{N})' = \top);\} \cup \Gamma(\mathscr{N}')$

```
                    end

        l : D(𝒜, p, g, 𝒩1, 𝒩2)  ⇒
            Case (𝒜) of
             ε      ⇒
            Case (p) of
             ]0,1[  ⇒
                      in
                       {[l]l ⟶ p : (l' = ⊥)&(l'_g = ⊤) + (1 − p) : (l' = ⊥)&(l'_¬g = ⊤);}
                       ⋃{[l_g]l_g ∧ g ⟶ (l'_g = ⊥)&(L(𝒩1)' = ⊤);}
                       ⋃{[l_¬g]l_g ∧ ¬g ⟶ (l'_¬g = ⊥)&(L(𝒩2)' = ⊤);}
                       ⋃_{i=1..2} Γ(𝒩i)
                      end
              −      ⇒
                      in
                       {[l]l ⟶ (l' = ⊥)&(L(g)' = ⊤);}
                       ⋃{[l]l ⟶ (l' = ⊥)&(L(¬g)' = ⊤);}
                       ⋃{[l_g]l_g ∧ g ⟶ (l'_g = ⊥)&(L(𝒩1)' = ⊤);}
                       ⋃{[l_¬g]l_g ∧ ¬g ⟶ (l'_¬g = ⊥)&(L(𝒩2)' = ⊤);}
                       ⋃_{i=1..2} Γ(𝒩i)
                      end
          Otherwise
                      in
                       {[l_s]l → (l' = ⊤);} ∪ Γ'(𝒜);
                       ∪{[l_e]l&L(E(𝒜)) → (l' = ⊥)&(L(D(p, g, 𝒩1, 𝒩2))' = ⊤);}
                       ∪Γ(l : D(p, g, 𝒩1, 𝒩2))
                      end
        l : ⊗       ⇒
                in
                    [l]l ⟶ (l' = ⊥);
                end
        l : ⊙       ⇒
                in
                    [l]l ⟶ &_{l∈ℒ}(l' = ⊥);
                end
// Defining the function Γ'(𝒜)
Γ' : 𝒜 → P
Γ'(𝒜) = ∀m ∈ 𝒜 : L(m) = ⊥ , Case (m) of
                l_ι : ι ↣ 𝒩  ⇒
```

```
79                    in
80                      {[l_s]l → (L(S(𝒜))′ = ⊤);
81                      [L(S(𝒜))]L(S(𝒜)) ⟶ (L(S(𝒜))′ = ⊥)&(L(𝒩)′ = ⊤);} ∪ Γ(𝒩)
82                    end
83              l : ⊙ ⇒
84                  in
85                    [l_e]l&L(E(𝒜)) → &_{l∈ℒ′}(l′ = ⊥);
86                  end
87            Otherwise        Γ(𝒜);
```

Listing 3.1: Generating PRISM Commands Function.

3.6 The Soundness of the Verification Approach

Our aim is to prove the soundness of our transformation algorithm T and to show that it preserves PCTL properties. Let $\mathscr{A}$ be a NuAC term and $M_{\mathscr{A}}$ be its corresponding PA constructed by the NuAC operational semantics denoted by $\mathscr{S}$ such that $\mathscr{S}(\mathscr{A}) = M_{\mathscr{A}}$. The function Γ is defined previously to implement the transformation rules that produce a PRISM program $\mathscr{P}$ such that $\Gamma(\mathscr{A}) = \mathscr{P}$. For the program $\mathscr{P}$, let $M_{\mathscr{P}}$ be its corresponding PA constructed by the PRISM operational semantics denoted by $\mathscr{S}'$ such that $\mathscr{S}'(\mathscr{P}) = M_{\mathscr{P}}$. As illustrated in Figure 3.6, proving the soundness of Γ algorithm is to find the adequate relation $\mathscr{R}$ between $M_{\mathscr{A}}$ and $M_{\mathscr{P}}$.

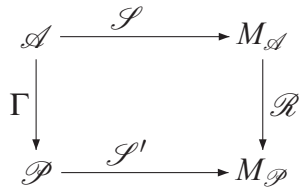


Figure 3.6: Mapping Soundness.

To define the relation $M_{\mathscr{A}}\ \mathscr{R}\ M_{\mathscr{P}}$, we have to establish a step by step correspondence

between $M_{\mathscr{A}}$ and $M_{\mathscr{P}}$. First, we introduce the notion of the probabilistic strong bisimulation relation. This relation is based on the probabilistic equivalence relation $\mathscr{R}$ defined in Definition 3.6 where $X/\mathscr{R}$ denotes the quotient space of X with respect to $\mathscr{R}$ and $\equiv_{\mathscr{R}}$ is the lifting of $\mathscr{R}$ to a probabilistic space.

Definition 3.6 (The equivalence $\equiv_{\mathscr{R}}$)**.** If $\mathscr{R}$ is an equivalence on X, then the induced equivalence $\equiv_{\mathscr{R}}$ on $Dist(X)$ is given by: $\mu \equiv_{\mathscr{R}} \mu'$ iff $\mu[C] \equiv_{\mathscr{R}} \mu'[C]$ for all $C \in X/\mathscr{R}$.

Hence, Definition 3.7 stipulates the probabilistic strong bisimulation relation.

Definition 3.7 (Strong Probabilistic Bisimulation)**.** A strong probabilistic bisimulation between two probabilistic automata M_1 and M_2 is an equivalence relation $\mathscr{R} \sqsubseteq S_1 \times S_2$ where:

1. Each initial state of M_1 is related to at least one initial state of M_2;
2. For each pair of states $s_1 \mathscr{R} s_2$ and each transition $s_1 \xrightarrow{a} \mu_1$ of M_1 of either M_1 or M_2, there exists a transition $s_2 \xrightarrow{a} \mu_2$ of either M_1 or M_2, such that $\mu_1 \equiv_{\mathscr{R}} \mu_2$.

Here, $\sqsubseteq_{\mathscr{R}}$ is the lifting of $\mathscr{R}$ to a probability space. It is achieved by finding a weight function [79] that associates each state of M_1 with others in M_2 by a certain probability value. The weight function is defined below in Definition 3.8.

Definition 3.8 (Weight Function)**.** Given two sets of states S_1 and S_2. Let $\mathscr{R} \sqsubseteq S_1 \times S_2$, $\mu_1 \in Dist(S_1)$ and $\mu_2 \in Dist(S_2)$ are two distributions over S_1 and S_2. A weight function for (μ_1, μ_2) w.r.t. $\mathscr{R}$ is a function $\triangle : S_1 \times S_2 \to [0,1]$ such that:

1. $\forall\, s_1 \in S_1 : \Sigma_{s_2 \in S_2} \triangle(s_1, s_2) = \mu_1(s_1)$,
2. $\forall\, s_2 \in S_2 : \Sigma_{s_1 \in S_1} \triangle(s_1, s_2) = \mu_2(s_2)$,
3. $\triangle(s_1, s_2) > 0 \Rightarrow s_1 \mathscr{R} s_2$.

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

Definition 3.9 (Mapping Relation)**.** The relation $M_{\mathscr{A}}\mathscr{R}M_{\mathscr{P}}$ between a NuAC term $\mathscr{A}$ and a PRISM term $\mathscr{P}$ such that $\Gamma(\mathscr{A}) = \mathscr{P}$ is a strong probabilistic bisimulation relation.

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

Lemma 3.1 (Soundness)**.** *The mapping algorithm* Γ *is sound, i.e.* $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}} M_{\mathscr{P}}$.

Proof. Our abstraction is sound and the proof is provided in Appendix B.4.

□

In the following, we show that the mapping relation preserves the satisfaction 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 3.1 (PCTL Preservation)**.** *For two PAs* $M_{\mathscr{A}}$ *and* $M_{\mathscr{P}}$ *such that* $\Gamma(\mathscr{A}) = \mathscr{P}$ *where* $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}} M_{\mathscr{P}}$. *For a PCTL property* ϕ, *then:* $(M_{\mathscr{A}} \models \phi) \Leftrightarrow (M_{\mathscr{P}} \models \phi)$.

Proof. Our abstraction is sound and the proof is provided in Appendix B.4.

□

3.7 Experimental Results

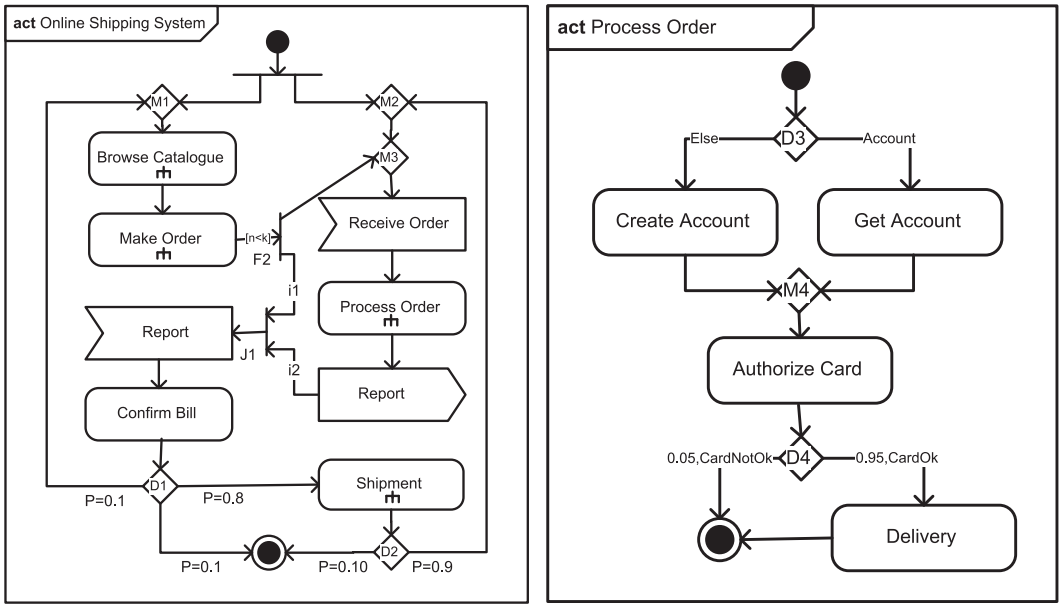
In this section, we apply our verification framework on the online shopping system [32] and the Real Time Streaming Protocol (RTSP) [1] hypothetical application. The related SysML

[1] http://tools.ietf.org/html/rfc2326

activity diagrams are modeled on Topcased[2] then mapped into Prism code via our Java implementation. In the purpose of providing experimental results demonstrating the efficiency and the validity of our approach, we verify a set of functional requirements of the system under study.

3.7.1 Online Shopping System

The online shopping system aims at providing services for purchasing online items. Figure 3.7a illustrates the corresponding SysML activity diagram. It contains four call-behavior actions[3], which are: "Browse Catalogue", "Make Order", "Process Order" and "Shipment". As example, Figure 3.7b expands the call behavior action "Process Order".



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

Figure 3.7: The Online Shopping System SysML Activity Diagram.

In order to check the correctness of the online shopping system, we propose to verify four functional requirements. They are expressed in PCTL as follows where n ($n \in [0..K]$)

[2]http://www.topcased.org Toolkit in OPen source for Critical Applications and SystEms Development.
[3]Each call-behavior action is represented by its proper diagram.

and m represent the order and the shipment numbers, respectively.

1. For each order, what is the minimum probability value to make a delivery?

   PCTL: $Pmin = ?[(n \leq K)\ \mathrm{U}\ (Delivery)]$.

2. After browsing the catalogue, what is the minimum probability value to ship a selected item?

   PCTL: $Pmin = ?[((SelectItem \wedge m = n \wedge m \leq K) \Rightarrow F(Delivery)) \Rightarrow F(Shipment)]$.

3. For a given customer, what is the maximum probability value to make a new order after confirming the bill?

   PCTL: $Pmax = ?[G((ConfirmBill) \Rightarrow F(MakeOrder))]$.

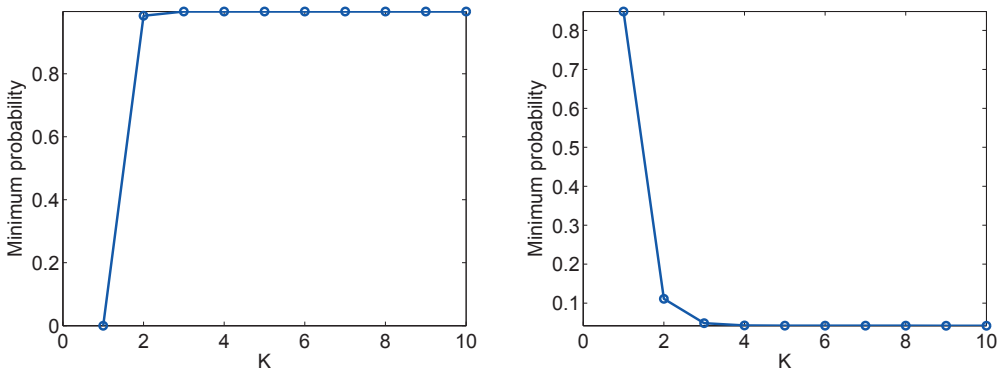
4. For each order, what is the maximum probability value to enter a wrong credit card code?

   PCTL: $Pmax = ?[(n = m) \Rightarrow (!CardOk))]$.

The verification results of the above four properties are shown in Figure 3.8. For different values of the maximum number of orders "K", Figure 3.8a shows that the verification result for Property 1 converges to 0.9977 after three steps. Figure 3.8b presents the verification result of Property 2 decrementing quickly from an initial value of 0.8487 to a steady-state value of 0.0420. The same applies to Property 3 which converges to 0.9622 and to Property 4 which converges to 0.0977.

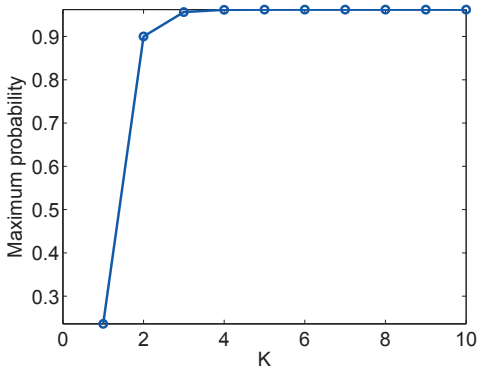
3.7.2 Real Time Streaming Protocol

RTSP is a client-server application for the delivery of data with real-time features. To deliver the continuous RTSP streams, it relies on the control of multiple data delivery sessions and provides means for choosing delivery mechanisms based upon RTP or Secure RTP

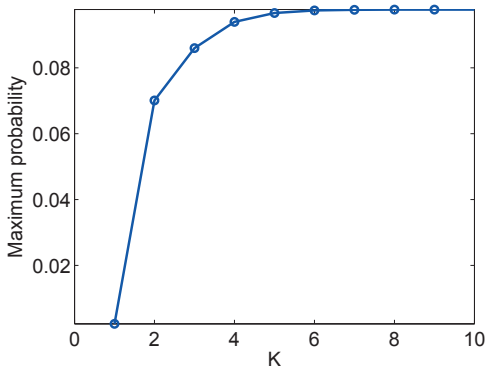


(a) Property 1.

(b) Property 2.



(c) Property 3.



(d) Property 4.

Figure 3.8: The Verification of PCTL Properties on the Shopping Online System.

(SRTP)[4]. SRTP provides confidentiality of RTP data, message integrity and supports source origin authentication. From RTSP and SRTP RFC's, we have extracted and designed the behavior of RTSP upon Secure RTP client-server application as SysML activity diagrams. Figure 3.9 shows the SysML activity diagram of RTSP client and Figure 3.10 presents the SysML activity diagram of the server specific to RTSP application. The main methods used to define RTSP vocabulary are:

- DESCRIBE: a request includes an RTSP URL, and the type of reply data that can be handled.

[4]http://tools.ietf.org/html/rfc3711

- SETUP: causes the server to allocate resources for a stream and start an RTSP session.
- PLAY and RECORD: starts data transmission on a stream allocated via SETUP.
- PAUSE: temporarily halts a stream without freeing server resources.
- TEARDOWN: frees resources associated with the stream. The RTSP session ceases to exist on the server.
- SUCCESS and ERROR: server's response to client requests.

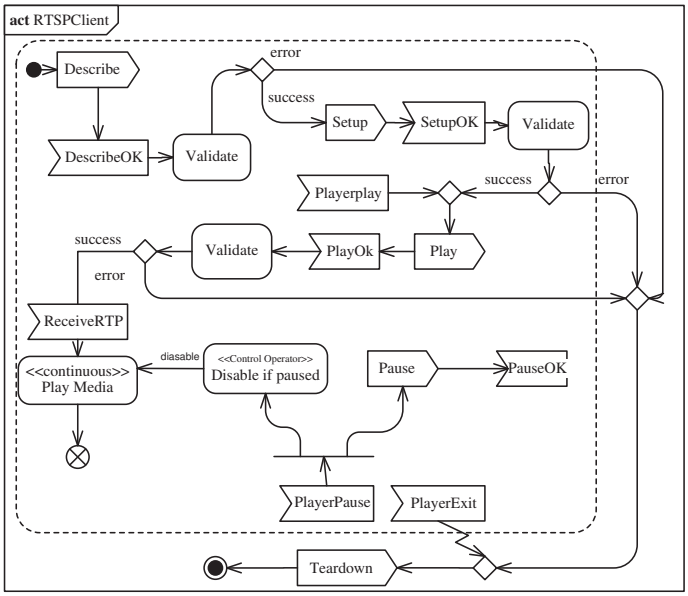


Figure 3.9: The Client SysML Activity Diagram for RTSP Protocol.

Here, we propose a set of PCTL properties to be verified against the composed model.

1. Compute the maximum probability to disconnect a client immediately by a TEARDOWN attack.

   Pmax=?[(¬Client!SendTeardown U≤step (Client.End))].

2. Measure the maximum probability of an attacker intercepting DESCRIBE message

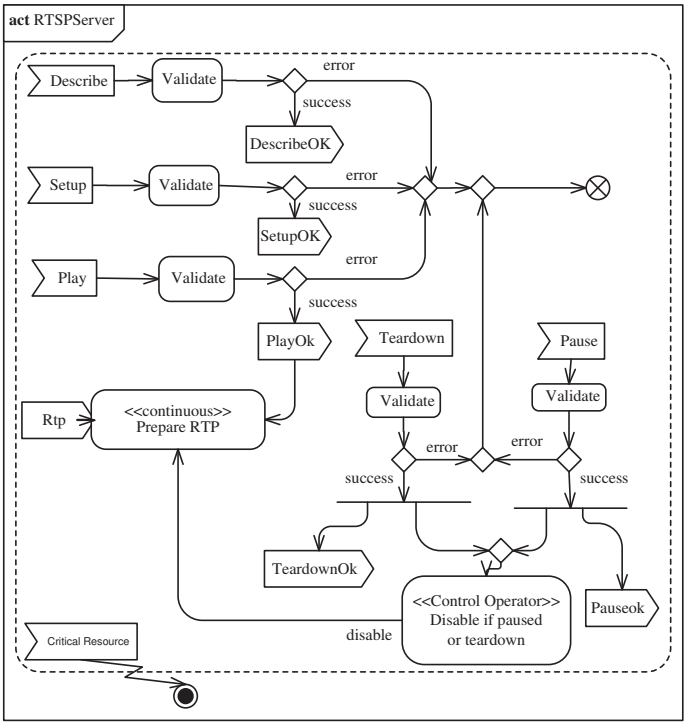


Figure 3.10: The Server SysML Activity Diagram for RTSP Protocol.

of a client.

Pmax=? [true U≤step (Attack?Describe)].

3. Evaluate the maximum probability to successfully hijack a session.

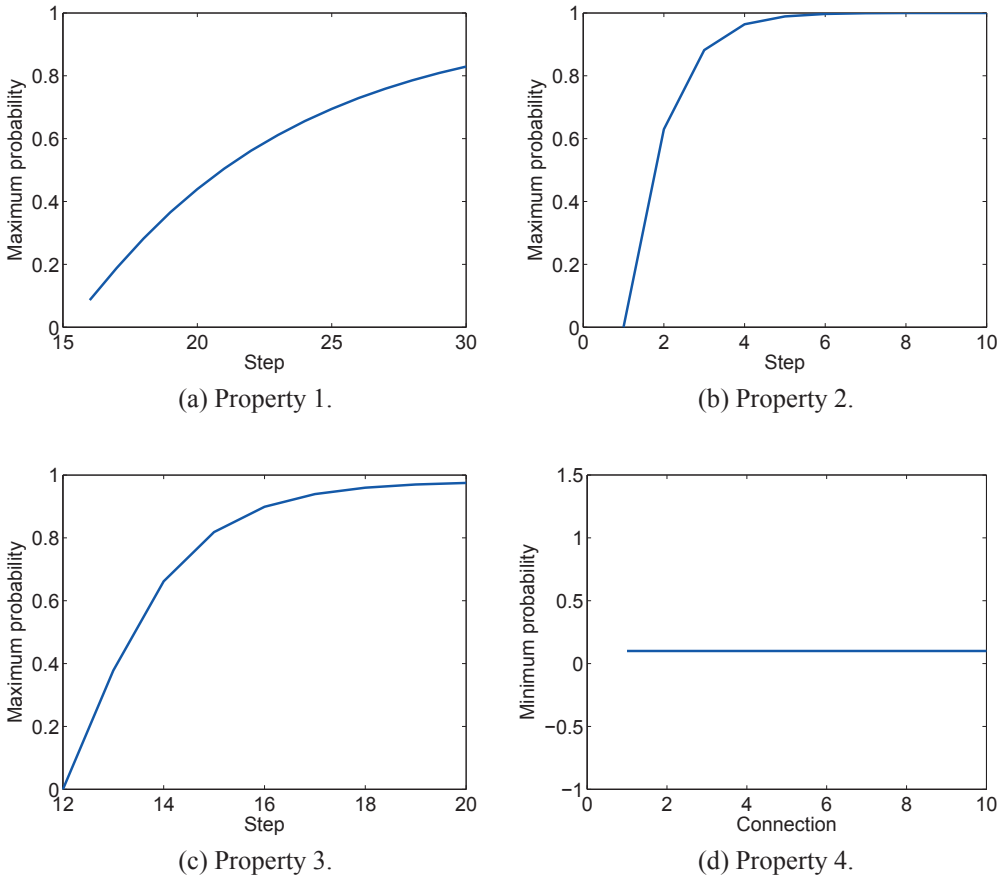
   Pmax=? [Client?RTP & Attack?RTP].

4. Find the minimum probability that a client fails to connect.

   Pmin=?[Client.start ⇒ (X(Client.start))].

By interpreting the results showed in Figure 3.11, we conclude that the RTSP application is free of deadlocks and the service of the client can be interrupted with a maximum probability of 0.82 after a TEARDOWN attack. Furthermore, the attacker's probability to intercepting a message is at least 0.63. Finally, an attacker can hijack the client session by a probability greater than 0.378 and a client can fail to be connected by a probability of

0.099.



(a) Property 1.

(b) Property 2.

(c) Property 3.

(d) Property 4.

Figure 3.11: The Verification of PCTL Properties on the RTSP Diagrams.

3.8 Related Work

In this section, we cite the state-of-the-art related to the formalization and the verification of UML/SysML interaction, state machine and activity diagrams, respectively.

3.8.1 Verification of UML Interaction Diagrams

Amstel et al. [85] propose an approach to analyze the quality of UML sequence diagram. It contains four steps: 1) analyzing the interaction traces and identify ambiguities by using

SPIN model checker, 2) identifying patterns of common interactive behavior, 3) identifying syntactic defects, 4) metric-based technique to describe properties such as coverage is used. The translation in the first step is based on state machine. The pattern identification is to locate message exchange and recurring message sequences. The static analysis is used in the third step to detect a sequence diagram syntactic defects such as unnamed message, and the last step is the stochastic version of basic and coverage analysis.

Lima et al. [51] verify UML 2.0 sequence diagrams by mapping each fragment into a PROMELA process where the send/receive event specified the communication between processes. The security properties are specified using LTL temporal logic so that they can be verified using SPIN model checker. The counterexample is mapped to a trace in the model to be analyzed later by the user.

3.8.2 Verification of UML State Machine Diagrams

The Static Verification Framework (SVF) developed by Siveroni et al. [81] translates the UML state machine into PROMELA models based on automata concept. The communication assigns two channels to an object one for receiving and the other for sending. To specify a property, they proposed a high level grammar to express properties in a user-friendly form and translate them later to LTL.

Beato et al. [8] present a tool for the Active Behavior of UML (TABU) to verify UML active diagrams by using SMV model checker. Three kinds of diagram are taken into account: class, state and activity diagrams. The first provides information concerning the elements that make up the system and their relationships, while the second and third provide information about the timing behavior of each of those elements. The properties are specified by the assistance of two pattern schemes: occurrence and order. Occurrence patterns describe

properties with respect to the occurrence of a state or signal during the evolution of a system. Order patterns establish properties with respect to the order in which they occur.

Kaliappan et al. [46] design and verify communication protocols by using model driven architecture and SPIN Model checker. The state machine is converted into PROMELA code as a protocol model and its properties are derived from the sequence diagram as Linear Temporal Logic (LTL) by using the Protocol Predictor (PP). The mapping rules are written in the standard OMG Query View Transformation (QVT) model.

3.8.3 Verification of UML/SysML Activity Diagrams

R. Eshuis [22] translates an activity diagram to NuSMV code. The translation is based on state machine and follows four steps: 1) Inserting a WAIT node for each edge entering a join, 2) Inserting a WAIT node between a join and a fork, 3) Replacing object nodes and flows by wait nodes and control flows, 4) Eliminating pseudonodes and define hyperedges. PLTL temporal logic is used for property specification.

Das et .al [17] present a timing verification of activity diagrams. The timing queries are described in a sub-set of Timed Propositional Temporal Logic (TPTL). The constructed semantic model is a reduced timed reachability graph which is Kripke structure-based. It represents a set of locations and time events for each token in the system at any time. To verify a TPTL property, two steps are: 1) Creating the tableaux of the complement of the underlying LTL formula of the given TPTL formula, 2) Deriving a B*ü*chi automata from the presented semantic model. Finally, applying a double DFS algorithm for checking the product of both automaton for a reachable accepting cycle. Rafe et .al [75] determine the correctness behavior and formal semantics of UML2.0 activity diagrams by Graph Transformation Systems (GTS).

Paolo et .al [6] specify and verify an airport case study described by a class diagram and

its behavioral activity diagram. The class diagram is interpreted as a graph, and the activity as graph transformations. $\mu\mathscr{L}2$ temporal logic is introduced which allows for formulating relevant properties of a GTS. The satisfaction of a property is defined inductively.

Federico et .al [7] map a UML4SOA activity diagram to COW. UML4SOA is an UML profile that has been designed for modeling SOA. COW is a process calculus for specifying service-oriented systems. An encoding function on the form of a predicate is defined to transform each UML4SOA construct to COW term. The authors don't show how they deal with merge element to define recursivity.

The approach proposed by Raida et .al [20] transforms a UML activity diagram to CSP expressions by using a graph transformation tool called ATOM. A meta-model for UML activity diagrams is proposed for UML activity diagram and a graph grammar that performs the transformation.

Ermeson et .al [13] propose a solution for verification of embedded realtime systems with energy constraints. Real-time systems are modeled using SysML State Machine diagram, and MARTE UML Profile (Modeling and Analysis of Real-Time and Embedded systems) to specify ERTS's (Embedded Real-time Systems) constraints such as execution time and energy. They map only the states and the 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 semantic transformation of the mutual exclusive, and orthogonal states by taking just the states inside into consideration.

The same authors of [13] propose in [3] a similar methodology for mapping SysML activity diagram to time Petri Net for requirement validation of embedded real-time systems with energy constraints. The computation model formalized as an extended Time Petri Net (TPN) is not well written. It misses the representation of the energy consumption values. The authors don't provide a formal transformation for the UML elements even the values

represented from MARTE profile. Also, they don't clarify why they present each constraint in an action by a separate transition.

Yosr et .al [44] use PRISM Model checker to verify a SysML Activity Diagram where the execution time of actions are formalized as constraints and the outputs of a decision node are attributed by a specific probabilities. Their approach is to map the source model to a Discrete-Time Markov Chains (DTMC) in order to use PRISM model checker for the assessment and evaluation of performance characteristics.

Yosr et .al [43] propose an algebraic calculus called Activity Calculus (AC) for SysML activity diagram. The semantic of the given calculus is constructed from five sets of axioms and derivation rules (construction rules) that are used to describe the behavior evolution of the studied diagram.

David et .al [41] introduced an extension of UML statecharts with randomly varying duration. It allows state transition to select probability. The Input/Output (I/O) automata concept is used to provide a compositional semantics. Also, probability distribution after a continuous or discrete time is introduced as an arbitrary operator. David et .al [40] introduce means to specify system randomness within statecharts, and to verify probabilistic temporal properties. The model is represented as MDP, and the properties ares expressed in PCTL. In [2] a framework has been proposed for verifying UML behavioral diagrams (State machine, Activity and Sequence diagrams). Each diagram is mapped into its proper semantic model that takes the form of of labeled transition system called Configuration Transition System (CTS). The resulting CTS is translated into NuSMV input language.

3.9 Conclusion

In this chapter, we presented a formal verification framework to improve the requirement checking of SysML activity diagrams. To verify these diagrams, we have devised an approach that maps a set of SysML activity diagrams composed by call behavior and communication actions into the input language of the probabilistic model checker PRISM. We proposed a calculus dedicated to these diagrams to capture precisely their underlying semantics. In addition, we formalized PRISM language. To this end, we proved the soundness of our proposed approach by defining adequately the relationship between the semantics of the mapped diagrams and the resulting models. In addition, we proved the preservation of the satisfaction of PCTL properties by this relation. Finally, we demonstrated the effectiveness of our approach by applying it on a real studies representing an online shopping system and the real time streaming protocol. The proposed framework could form the basis of any future work targeting any other operation rather than the mapping for SysML diagrams. In the next chapter, we propose a property-based abstraction methodology to reduce the verification complexity of SysML activity diagrams.

Chapter 4

Abstraction of SysML Activity Diagrams

4.1 Introduction

It is of a major importance to reduce the size complexity of SysML activity diagrams while any approach translating the concrete diagrams into the input language of the model checker would be limited by the tool's abstraction mechanism, if exists. Moreover, abstracting the semantic model instead of the concrete diagram can be costly, while the size of the semantic model is greater than that of the diagram itself (Appendix B.1). In this chapter, we are interested in the efficient verification of SysML activity diagrams by proposing an abstraction framework.

Our proposed framework is based on abstracting the irrelevant action nodes and guards with respect to a specific requirement, then, collapsing nodes that share similar behaviors. Figure 4.1 presents an overview of the proposed abstraction framework. It takes SysML activity diagrams and PCTL [5, 27] expressions as input. To perform verification, we devise using the verification framework developed in Chapter 3. We prove the soundness of our abstraction approach by comparing the satisfaction relation of PCTL properties on both the abstract and the concrete diagrams. To do so, we define the adequate relation

between the semantics models related of both diagrams (concrete and abstract). Then, we show the PCTL operators that can be preserved by this relation.

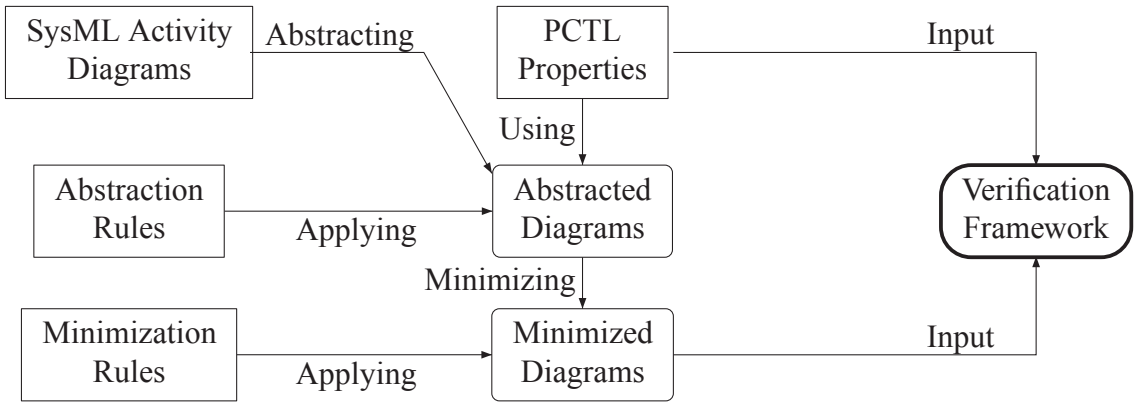


Figure 4.1: A Probabilistic Abstraction Framework.

By applying our approach on the example of ATM system introduced in Chapter 3, we get a new model "Figure 4.2". From the obtained results, we find that our approach preserves the requirement probability value (0.84). As expected, our approach consumes less time and size memory.

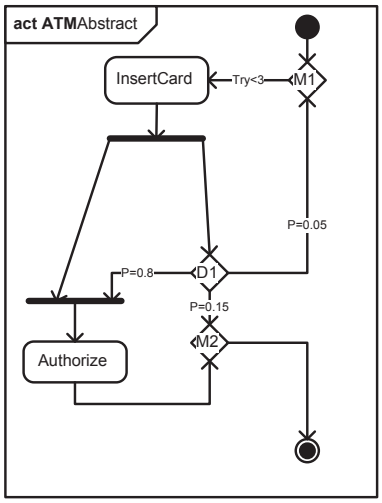


Figure 4.2: Abstract ATM SysML Activity Diagrams.

The remainder of this chapter is organized as follows. The proposed abstraction approach is detailed in Section 4.2 then its soundness is proved in Section 4.3. Section 4.5

surveys the related work. Section 4.4 describes the experimental results. Finally, Section 4.6 concludes this chapter.

4.2 Abstraction Approach

This section describes our abstraction approach for SysML activity diagrams. First, we present our abstraction algorithm. Then, we cover the special case of calling behaviors. Finally, we calculate the complexity of our abstraction algorithm.

4.2.1 The Abstraction Algorithm

The abstraction of a SysML activity diagram $\mathscr{A}$ is based on both structures of $\mathscr{A}$ and the PCTL property ϕ to be verified. The abstraction algorithm δ illustrated in Algorithm3 takes $\mathscr{A}$ and ϕ as input and returns a reduced SysML activity diagram denoted by $\widehat{\mathscr{A}}$. The diagram $\mathscr{A}$ is visited using a depth-first search procedure that can be described as follows. First, the initial node is pushed into the stack of nodes denoted by *nodes* (line 5). While the stack is not empty (line 6-18), the algorithm pops a node from the stack into the current node denoted by *cNode* (line 7). The current node is added into the list *vNode* of visited nodes (line 9) if it is not already visited (line 8). $\widehat{\mathscr{A}}$ is constructed by calling both functions Υ and Ψ. The function Υ has the current node along with ϕ as arguments (line 10). It forbids $\mathscr{A}$ behaviors and represents symbolically decision guards. The function Ψ has two arguments that are the current node and its successors (line 13). It merges the current node and its successors that share specific properties. The successor nodes are pushed into the stack *nodes* (line 14) to be explored later. The algorithm ends when all nodes are visited.

The function Υ uses the atomic propositions of the PCTL property ϕ. These are principally formed from actions and guards labels while the other nodes are used to control

Algorithm 3 The abstraction algorithm δ of SysML Activity Diagrams

Input: SysML activity diagram $\mathscr{A}$, a PCTL property ϕ.
Output: SysML activity diagram $\widehat{\mathscr{A}}$.

```
1:  nodes as Stack;                                  ▷ A stack of nodes which is initially empty.
2:  cNode as Node;                                   ▷ The current node which is initially empty.
3:  nNode, vNode as list_of_ Node;                   ▷ List of nodes that are initially empty.
4:  procedure δ(A,φ)
5:      nodes.push(in);                              ▷ Read the initial node.
6:      while not nodes.empty() do
7:          cNode := nodes.pop();                    ▷ Pop the current node.
8:          if cNode not in vNode then
9:              vNode.add(cNode);                    ▷ Consider the current node as a visited node.
10:             ϒ(cNode,φ);                          ▷ Call the function ϒ.
11:             nNode := cNode.successors();         ▷ Get the successors of the current node.
12:             for all n in nNode do                ▷ Stores all newly discovered nodes in the stack.
13:                 Ψ(cNode,n);                      ▷ Call the function Ψ.
14:                 nodes.push(n);                   ▷ Stores all newly discovered nodes in the stack.
15:             end for
16:             nNode.clear();                       ▷ Empty the list nNode.
17:         end if
18:     end while
19: end procedure
```

the execution of these actions. Let Σ_ϕ be a set of independent atomic propositions such that $\Sigma_\phi \subseteq \{a_i : i \leq n\} \cup \{g_i : i \leq m\}$ where a_i is a label corresponding to an action, g_i is a guard label, n and m are the number of actions and guards in $\mathscr{A}$, respectively. The function Υ hides the action nodes and guards of the SysML activity diagram that are not part of the atomic propositions of the PCTL property to be verified. It takes as input a NuAC term $\mathscr{N}$ along with Σ_ϕ and generates an abstract term $\widehat{\mathscr{N}}$ such that $\Upsilon(\mathscr{N}, \Sigma_\phi) = \widehat{\mathscr{N}}$. The function Υ implements the six inference rules stipulated in Figure 4.3 that are explained as follows.

ABS-1 rule preserves an action label when it appears in ϕ propositions set.

ABS-2 rule hides an action label when it does not appear in ϕ propositions set. The produced result is that its predecessor will be connected directly with its successor after applying the rule ABS-5.

ABS-1	$\dfrac{a \rightarrowtail \mathcal{N} \quad a \in \Sigma_\phi}{a \rightarrowtail \mathcal{N}}$	ABS-2	$\dfrac{a \rightarrowtail \mathcal{N} \quad a \notin \Sigma_\phi}{\varepsilon \rightarrowtail \mathcal{N}}$
ABS-3	$\dfrac{\mathcal{N}_1 \overset{g}{\mapsto} \mathcal{N}_2 \quad g \in \Sigma_\phi}{\mathcal{N}_1 \overset{g}{\mapsto} \mathcal{N}_2}$	ABS-4	$\dfrac{\mathcal{N}_1 \overset{g}{\mapsto} \mathcal{N}_2 \quad g \notin \Sigma_\phi}{\mathcal{N}_1 \mapsto \mathcal{N}_2}$
ABS-5	$\dfrac{\mathcal{N}_1 \rightarrowtail \varepsilon \rightarrowtail \mathcal{N}_2}{\mathcal{N}_1 \rightarrowtail \mathcal{N}_2}$	ABS-6	$\dfrac{D(g_1, \mathcal{N}_1, g_2, \mathcal{N}_2, g_3, \mathcal{N}_3) \quad g_{i:i>1} \notin \Sigma_\phi}{D(g_1, \mathcal{N}_1, \neg g_1, \mathcal{N}_2, \neg g_1, \mathcal{N}_3)}$

Figure 4.3: Inference Rules for Υ Function.

ABS-3 rule preserves the guard label in an edge when it belongs to the set of ϕ propositions.

ABS-4 rule empties a guarded edge when its guard g is not a part of ϕ propositions.

ABS-6 rule modifies decision guarded edges by replacing guards of the unconcerned edges by the negation of the other guards. ABS-6 can be applied on the extended versions of decision nodes riches in probabilities and/or behavior calls.

The second function Ψ minimizes the diagram by collapsing specific control nodes while preserving the number of tokens and their control paths too. This is achieved by preventing the modification of guarded and probabilistic choices and the modification in multi-input/output nodes. The function Ψ implements twelve derivation rules (MIN-1,12). The rules MIN-1,...,MIN-6 are based on the commutativity described in Property 3.2. These rules aim at merging consecutive control nodes of the same type. The rules MIN-7,...,MIN-10 merge parallel paths that have similar destination. The rules MIN-11 and MIN-12 eliminate useless nodes resulting from the application of the described ABS and MIN rules. Hence, the minimization rules are explained as follows.

MIN-1 rule consolidates two consecutive fork nodes to produce an equivalent one.

$$\frac{l : F(\mathcal{N}_1, \mathcal{N}_2) \quad \mathcal{N}_1 = l_1 : F(\mathcal{N}_{11}, \mathcal{N}_{12})}{l : F(\mathcal{N}_1, \mathcal{N}_{12}, \mathcal{N}_{22})}$$

MIN-2 rule merges two consecutive join nodes in only one join node.

$$\frac{l : J(x_1,x_2) \rightarrowtail \mathcal{N} \quad \mathcal{N} = l' : J(x_3,x_4) \rightarrowtail \mathcal{N}'}{l : J(x_1,x_2,x_3,x_4) \rightarrowtail \mathcal{N}'}$$

MIN-3 rule incorporates two successive merge nodes in one.

$$\frac{l : M(x_1,x_2) \rightarrowtail \mathcal{N} \quad \mathcal{N} = l' : M(x_3,x_4) \rightarrowtail \mathcal{N}'}{l : M(x_1,x_2,x_3,x_4) \rightarrowtail \mathcal{N}'}$$

MIN-4 rule collapses two successive probabilistic decision nodes to form one equivalent probabilistic decision node.

$$\frac{l : D(\mathcal{A},p,g,\mathcal{N}_1,\mathcal{N}_2) \quad \mathcal{N}_1 = l_1 : D(\mathcal{A}_1,p_1,g_1,\mathcal{N}_{11},\mathcal{N}_{12})}{l : D(F(\mathcal{A},\mathcal{A}_1),p \times p_1, g \wedge g_1, \mathcal{N}_{11}, p \times (1-p_1), g \wedge \neg g_1, \mathcal{N}_{12}, 1-p, \neg g, \mathcal{N}_2)}$$

MIN-5 rule constructs a deterministic decision node starting from two consecutive deterministic decision nodes.

$$\frac{l : D(\mathcal{A},g,\mathcal{N}_1,\mathcal{N}_2) \quad \mathcal{N}_1 = l_1 : D(\mathcal{A}_1,g_1,\mathcal{N}_{11},\mathcal{N}_{12})}{l : D(F(\mathcal{A},\mathcal{A}_1),g \wedge g_1, \mathcal{N}_{11}, g \wedge \neg g_1, \mathcal{N}_{12}, \neg g, \mathcal{N}_1)}$$

MIN-6 rule merges a guarded decision node followed by another probabilistic decision node. The result is a probabilistic decision node.

$$\frac{l : D(\mathcal{A},g,\mathcal{N}_1,\mathcal{N}_2) \quad \mathcal{N}_1 = l_1 : D(\mathcal{A}_1,p_1,g_1,\mathcal{N}_{11},\mathcal{N}_{12})}{l : D(F(\mathcal{A},\mathcal{A}_1),p_1, g \wedge g_1, \mathcal{N}_{11}, 1-p_1, g \wedge \neg g_1, \mathcal{N}_{12}, \neg g, \mathcal{N}_2)}$$

MIN-7 rule minimizes outputs of a decision node that have the same join successor to only one output. This also results in the minimization of input pins of the join node.

$$\frac{l : D(\mathcal{A},p_1,g_1,\mathcal{N}_1,p_1,\neg g_1 \mathcal{N}_2,p_3,\neg g_1,\mathcal{N}_3) \quad \mathcal{N}_i = l' : J(x_i) \rightarrowtail \mathcal{N}' \;\; i = 2,3}{l : D(\mathcal{A},p_1,g_1,\mathcal{N}_1,\mathcal{N}_2) \quad x = x \backslash \{x_3\}.}$$

MIN-8 rule replaces a decision node outputs that have the same merge successor with only one output.

$$\frac{l : D(\mathcal{A},p,g,\mathcal{N},\mathcal{N}_1,\mathcal{N}_2) \quad \mathcal{N}_i = l' : M(x_i) \rightarrowtail \mathcal{N}' \;\; i = 1,2.}{l : D(\mathcal{A},p,g,\mathcal{N},\mathcal{N}_1) \quad x = x \backslash \{x_2\}.}$$

MIN-9 rule reduces outputs of a fork node followed by a join.

$$\frac{l : F(\mathcal{N},\mathcal{N}_1,\mathcal{N}_2) \;\; \mathcal{N}_i = l' : J(x_i) \rightarrowtail \mathcal{N}' \;\; i = 1,2.}{l : F(\mathcal{N},\mathcal{N}_1) \quad x = x \backslash \{x_2\}.}$$

MIN-10 rule reduces outputs of a fork node followed by a merge node.

$$\frac{l : F(\mathcal{N},\mathcal{N}_1,\mathcal{N}_2) \;\; \mathcal{N}_i = l' : M(x_i) \rightarrowtail \mathcal{N}' \;\; i = 1,2.}{l : F(\mathcal{N},\mathcal{N}_1) \quad x = x \backslash \{x_2\}.}$$

MIN-11 rule eliminates a merge node when it has only one input.

$$\frac{l : M(x) \rightarrowtail \mathcal{N} \quad |x| = 1.}{\varepsilon \longrightarrow \mathcal{N}}$$

MIN-12 rule eliminates a join node when it has only one input.

$$\frac{l : J(x) \rightarrowtail \mathcal{N} \quad |x| = 1.}{\varepsilon \longrightarrow \mathcal{N}}$$

4.2.2 The Call Behavior Case

In order to ensure the scalability of the verification process of a SysML activity diagram $\mathscr{A}$ composed of k call behaviors $\mathscr{A}_{i:1 \leq i \leq k}$, we have proved Proposition 4.1, and Proposition 4.2. First, Property 4.1 stipulates the associativity property of the call behavior operator "$\uparrow$" that is needed in our proofs.

Property 4.1. *The call behavior operator $\uparrow$ is associative:*

$$(\mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2) \uparrow_{a_2} \mathscr{A}_3 \equiv \mathscr{A}_1 \uparrow_{a_1} (\mathscr{A}_2 \uparrow_{a_2} \mathscr{A}_3)$$

Proof. The proof is based on the call behavior definition by following five steps are: 1) constructing $M_1 = \mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2$, 2) constructing $M = M_1 \uparrow_{a_2} \mathscr{A}_3$, 3) constructing $N_1 = \mathscr{A}_2 \uparrow_{a_2} \mathscr{A}_3$, 4) constructing $N = \mathscr{A}_1 \uparrow_{a_1} N_1$, 5) comparing M and N. □

Based on Property 4.1, we have proved Proposition 4.1 to handle the verification of the diagram $\mathscr{A}$ of k associated behaviors with respect to a PCTL property ϕ. It takes into consideration a set of call behaviors of size less than k. The call behavior composition is built upon the fact that the property ϕ holds on the constructed diagram.

Proposition 4.1. *Let $\mathscr{A} = \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k$ be a SysML activity diagram with k call behaviors and ϕ be a PCTL property, we have:*

$$\forall i \leq k : \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_i} \mathscr{A}_i \models \phi \;\Rightarrow\; \mathscr{A} \models \phi \,.$$

Proof. The proof of Proposition 4.1 is in Appendix B.2. □

To further improve Proposition 4.1, we define the identity element associated to the operator $\uparrow$ and denote it by $\mathscr{A}_{id}$ such that:

$$\mathscr{A} \uparrow_a \mathscr{A}_{id} \equiv \mathscr{A} \text{ and } \mathscr{A}_{id} = \varepsilon.$$

By using the identity element, we focus more on behaviors influenced by the property ϕ. This is achieved by replacing the unconcerned behaviors in Proposition 4.1 by $\mathscr{A}_{id}$. In Proposition 4.2, we use $\Sigma_{\mathscr{A}}$ and Σ_{ϕ} to denote the set of action labels of the diagram $\mathscr{A}$ and the set of atomic propositions of the property ϕ, respectively.

Proposition 4.2. *Let $\mathscr{A} = \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k$ be a SysML activity diagram with k call behaviors, $\mathscr{A}_{id}$ is the identity element for "$\uparrow$" operator and ϕ be a PCTL property. For a proposition α, we have the following:*

$$\forall 1 \leq i \leq k,\ \alpha \notin (\Sigma_{\phi} \cap \Sigma_{\mathscr{A}_i}) : [\mathscr{A}_i = \mathscr{A}_{id} \wedge (\mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k) \models \phi] \Rightarrow [\mathscr{A} \models \phi].$$

Proof. The proof follows the structural induction on PCTL syntax and it is provided in Appendix B.2. □

4.2.3 The Complexity Measure

Basically, the function Υ produces a new model that includes mainly the specified actions and guards in the property ϕ where other actions are considered as silent action. Thus, the resulting diagram has a reduced number of actions, which increases the occurrence of consecutive control nodes. Consequently, applying Υ before Ψ is more efficient in terms of time complexity as shown by Proposition 4.3.

Proposition 4.3 (Application Order)**.** *Let "$\mathscr{A}$" be a NuAC term and "ϕ" be a PCTL property, we have:* $\Psi(\Upsilon(\Psi(\mathscr{A}), \phi)) \equiv \Psi(\Upsilon(\mathscr{A}, \phi))$.

Proof. The proof of Proposition 4.3 is in Appendix B.3. □

Now, we calculate the time complexity of the algorithm δ for a SysML activity diagram $\mathscr{A}$ of n nodes (we consider n as the maximum number of nodes supported by $\mathscr{A}$). In Algorithm 3, the while loop can run at most n times (line 6). Recursively, functions Υ and Ψ recall the algorithm δ when a call behavior node exists. This mechanism of recalling produces a hierarchical form of diagrams where each node can call a new SysML activity diagram. We introduce the notion of hierarchy depth denoted by k, which means the level in the hierarchy from where a node has a call behavior. The worst case is that for any level of the hierarchy, each node can call a new behavior. The computing time complexity of δ is of class P with a worst case running time of $\mathscr{O}(n^k)$.

4.3 The Soundness of the Abstraction Approach

In this section, we prove the soundness of our proposed abstraction algorithm. More precisely, we prove that our algorithm preserves the satisfaction of PCTL properties.

Let $\mathscr{A}$ be a NuAC term and $M_{\mathscr{A}}$ be its corresponding PA constructed by NuAC operational semantics $\mathscr{S}$ presented in Section 3.3 such that $\mathscr{S}(\mathscr{A}) \equiv M_{\mathscr{A}}$. The abstraction algorithm δ calls both procedures Υ and Ψ such that $\delta(\mathscr{A}, \phi) = \widehat{\mathscr{A}}$, where $\widehat{\mathscr{A}}$ denotes the abstracted term of $\mathscr{A}$ with respect to the PCTL property ϕ. Let $M_{\widehat{\mathscr{A}}}$be its corresponding PA defined using NuAC operational semantics $\mathscr{S}$ such that $\mathscr{S}(\widehat{\mathscr{A}}) \equiv M_{\widehat{\mathscr{A}}}$. Proving the soundness of the algorithm δ is to find a pre-order relation $\mathscr{R}$ between $M_{\mathscr{A}}$ and $M_{\widehat{\mathscr{A}}}$. This relation represents the degree of precision of $M_{\mathscr{A}}$ in $M_{\widehat{\mathscr{A}}}$. As illustrated in Figure 4.4, the relation $\mathscr{R}$ is composed of two relations which are $\mathscr{R}_\Upsilon$ and $\mathscr{R}_\Psi$ to specify both functions Υ and Ψ.

To define the relation $M_{\mathscr{A}}\ \mathscr{R}_\Upsilon\ M_{\widehat{\mathscr{A}}}$, we introduce the notion of weakness [79] while

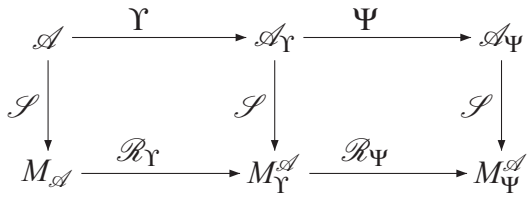


Figure 4.4: Abstraction Correctness.

Υ stutters action nodes and guarded edges. The probabilistic version of a weak transition is denoted by ($s \stackrel{a}{\Longrightarrow} \mu$) where μ is the distribution over states reached from s through a *sequence of mimicked steps*. The probabilistic weak simulation relation is formally described in Definition 4.1.

Definition 4.1 (Probabilistic Weak Simulation)**.** A probabilistic weak simulation between two probabilistic automata M_1 and M_2 is a relation $\mathscr{R} \sqsubseteq S_1 \times S_2$, such that:

1. Each initial state of M_1 is related to at least one initial state of M_2,
2. For each pair of states $s_1 \mathscr{R} s_2$ and each transition $s_1 \stackrel{a}{\rightarrow} \mu_1$ of M_1, there exist a weak combined transition $s_2 \stackrel{a}{\Longrightarrow} \mu_2$ of M_2 such that $\mu_1 \sqsubseteq_{\mathscr{R}} \mu_2$.

Here, $\sqsubseteq_{\mathscr{R}}$ is the lifting of $\mathscr{R}$ to a probability space. It is achieved by finding a weight function [79] that associates each state of M_1 with others in M_2 by a certain probability value. Definition 4.2 defines the weight function.

Definition 4.2 (Weight Function)**.** A function $\triangle : S \times S' \rightarrow [0,1]$ is a weight function for the two distributions $\mu_1, \mu_2 \in Dist(S)$ w.r.t. $\mathscr{R} \sqsubseteq S \times S'$, iff:

1. $\triangle(s_1, s_2) > 0 \Rightarrow (s_1, s_2) \in \mathscr{R}$,
2. $\forall\, s_1 \in S : \Sigma_{s_2 \in S} \triangle(s_1, s_2) = \mu_1(s_1)$,
3. $\forall\, s_2 \in S : \Sigma_{s_1 \in S} \triangle(s_1, s_2) = \mu_2(s_2)$ then $s_1 \mathscr{R} s_2$.

For our proof, we stipulate herein Definition 4.3 Υ-abstraction relation between $M_{\mathscr{A}}$ and $M_{\widehat{\mathscr{A}}}$ that is denoted by $M_{\mathscr{A}}\ \mathscr{R}_{\Upsilon}\ M_{\widehat{\mathscr{A}}}$ where $M_{\widehat{\mathscr{A}}} \equiv M_{\Upsilon}^{\mathscr{A}}$.

Definition 4.3 (Υ-Abstraction Relation)**.** An Υ-abstraction relation is a weak probabilistic simulation relation between a term $\mathscr{A}$ and its abstracted term $\mathscr{A}_{\Upsilon}$ after applying Υ algorithm.

In the following, we present the soundness of Υ algorithm. Let $M_{\mathscr{A}}$ be a PA representing the semantics of the NuAC term $\mathscr{A}$, and $M_{\widehat{\mathscr{A}}}$is the PA representing the semantics of $\widehat{\mathscr{A}}$ such that $\widehat{\mathscr{A}} = \Upsilon(\mathscr{A}, \phi)$. Proving that Υ is sound means proving there exists a weak probabilistic simulation between $M_{\mathscr{A}}$ and $M_{\widehat{\mathscr{A}}}$.

Lemma 4.1 (Υ-Soundness)**.** *The abstraction algorithm Υ is sound, i.e. $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_{\Upsilon}} M_{\widehat{\mathscr{A}}}$.*

Proof. Our abstraction is sound and the proof is provided in Appendix B.4. □

Now, we show that Υ-abstraction relation preserves PCTL properties. To achieve that, we prove by induction on the structure of the PCTL syntax to check PCTL preservation in both abstract and concrete models. We found that, except for the neXt operator ($PCTL_{\setminus X}$), such a formula (ϕ) holds in the concrete model if it holds in the abstracted model as stated in Proposition 4.4.

Proposition 4.4 (Υ-Preservation)**.** *For two PAs $M_{\mathscr{A}}$ and $M_{\widehat{\mathscr{A}}}$such that $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_{\Upsilon}} M_{\widehat{\mathscr{A}}}$. If ϕ is a $PCTL_{\setminus X}$ property, then we have: $(M_{\widehat{\mathscr{A}}} \models \phi) \Rightarrow (M_{\mathscr{A}} \models \phi)$.*

Proof. The proof of Proposition 4.4 is provided in Appendix B.4. □

Similarly to Υ-abstraction relation, we define in Definition 4.4 Ψ-Abstraction Relation.

Definition 4.4 (Ψ-Abstraction Relation)**.** An Ψ-abstraction relation is a weak probabilistic simulation relation between a term $\mathscr{A}$ and its abstracted term $\widehat{\mathscr{A}}$ by applying Ψ algorithm.

In Proposition 4.2, we present the soundness of Ψ-abstraction relation.

Lemma 4.2 (Ψ-Soundness)**.** *The abstraction algorithm Ψ is sound, i.e. $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_\Psi} M_{\widehat{\mathscr{A}}}$.*

Proof. Our abstraction is sound and the proof is provided in Appendix B.4. □

In Proposition 4.5, we prove the set of PCTL operators that are preserved by Ψ-abstraction relation.

Proposition 4.5 (Ψ-Preservation)**.** *For two PAs $M_{\mathscr{A}}$ and $M_{\widehat{\mathscr{A}}}$such that $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_\Psi} M_{\widehat{\mathscr{A}}}$. If ϕ is a PCTL property, then we have: $(M_{\widehat{\mathscr{A}}} \models \phi) \Rightarrow (M_{\mathscr{A}} \models \phi)$.*

Proof. The proof of Proposition 4.5 is provided in Appendix B.4. □

4.4 Experimental Results

In this section, we apply our abstraction approach on an online shopping system and the real time streaming protocol detailed in Chapter 3. In the purpose of providing experimental results demonstrating the efficiency and the validity of our abstraction, we verify PCTL properties on both: the concrete and the abstract diagrams. These diagrams are modeled on Topcased, then abstracted via our Java implementation in order to gain from the implementation of Chapter 3.

To this end, we compare the results perspective of the verification cost (β) and the abstraction efficiency (η). To evaluate the verification cost, we measure the time required for verifying a given property, denoted by T_v. To evaluate the abstraction efficiency, we measure the time required to construct the model, denoted by T_c. The verification cost is given by $\beta = 1 - \frac{|Tv(\widehat{M})|}{|Tv(M)|}$ and the abstraction efficiency by $\eta = 1 - \frac{|Tc(\widehat{M})|}{|Tc(M)|}$. Concerning the abstraction efficiency, we measure the number of states ($\#s$) and transitions ($\#t$) for both concrete and abstract diagrams. The result of the verification is denoted by (Res).

4.4.1 Online Shopping System.

The online shopping system [32] aims at providing services for purchasing online items. It contains four call-behavior actions[1], which are: "Browse Catalogue", "Make Order", "Process Order" and "Shipment".

In order to prove the correctness of the online shopping system, we propose to verify four functional requirements. They are expressed in PCTL as follows where n ($n \in [0..K]$) and m represent the order and the shipment numbers, respectively.

1. For each order, what is the minimum probability value to make a delivery? We express this property in PCTL as follows, where K is the maximum allowed number to make an order.

$$Pmin = ?[(n \leq K)\ U\ (Delivery)].$$

   From this expression, it is clear that only the main diagram and "Process Order" behavior are affected.

2. After browsing the catalogue, what is the minimum probability value to ship a selected item? Its corresponding PCTL expression is:

$$Pmin = ?[((SelectItem \wedge m = n \wedge m \leq K) \Rightarrow F(Delivery)) \Rightarrow F(Shipment)].$$

   The propositions of this property belong to the main diagram and the behaviors of: "Browse Catalogue", "Process Order" and "Shipment".

3. For a given customer, what is the maximum probability value to make a new order after confirming his bill? Hence, the PCTL property related to this statement is formulated.

[1] Each call-behavior action is represented by its proper SysML activity diagram.

$$Pmax = ?[G((ConfirmBill) \Rightarrow F(MakeOrder))].$$

The atomic propositions of this property belongs to the main diagram.

4. For each order, what is the maximum probability value to enter a wrong code? We write its PCTL expression as:

$$Pmax = ?[(n = m) \Rightarrow F(!CardOk))].$$

We observe that the atomic propositions of this property belong to the "Process Order" SysML activity diagram.

After applying our abstraction approach, we obtain for each property a new abstracted SysML activity diagrams and its verification results.

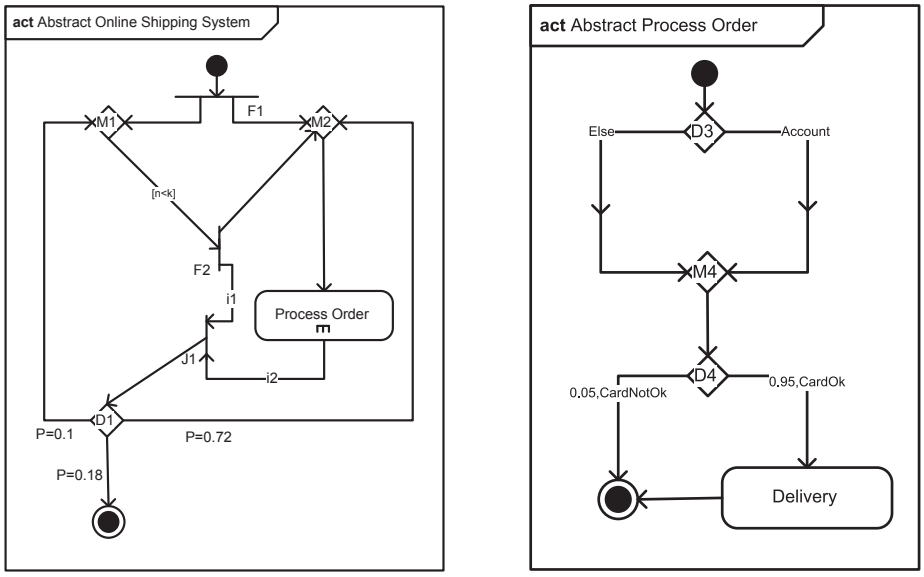
For Property 1, Figure 4.5 shows its abstracted diagram and Table 4.1 presents its different verification results in function of the number of orders "*n*". Furthermore, the verification results for Property 2, Property 3 and Property 4 are 0.88889, 0.145 and 0.14726, respectively. The obtained values show that our abstraction algorithm actually preserves the verification results. In addition, Figure 4.6 illustrates both abstraction rates in terms of the model size and the computation time for the verification of the above PCTL properties on the online shopping system. The evolution of both abstraction rates are important compared to the growing of the model size which means the proposed abstraction improves the verification cost.

4.4.2 Real Time Streaming Protocol

Here, we propose a set of PCTL properties to be verified against the composed model.

1. Compute the maximum probability to disconnect a client immediately by a TEARDOWN attack.

   Pmax=?[(¬Client!Teardown $\Rightarrow F$ (Client.End))].



(a) Online Shopping System (b) Process Order

Figure 4.5: The Abstract SysML Activity Diagram for Property 1.

2. Measure the maximum probability of an attacker intercepting DESCRIBE message of a client.

   Pmax=? [true $\Rightarrow F$ (Attack?Describe)].

3. Evaluate the maximum probability to successfully hijack a session.

   Pmax=? [Client?RTP & Attack?RTP].

4. Find the minimum probability that a client fails to connect.

   Pmin=?[Client.start $\Rightarrow$ (F(Client.start))].

The results obtained from the verification of the above properties with and without abstraction show that Property 1 achieves a maximum probability of 0.82 and at least a probability value of 0.63 for Property 2. Furthermore, the maximum probability of the third property is 0.974 and the minimum probability for Property 4 is 0.099. To show the abstraction efficiency of the verification of these properties, Figure 4.7 illustrates their abstraction rates in terms of the model size and the computation time in function of the number of the sent

	Concrete Model				Abstract Model				Res
n	*#s*	*#t*	Tc	Tv	*#s*	*#t*	Tc	Tv	
5	4848	9682	0.404	33.31	1339	2727	0.058	5.198	0.9977
10	9308	18607	0.548	63.501	0.122	2619	5352	10.49	0.9977
15	13768	27532	0.72	94.264	3899	7977	0.147	15.914	0.9977
20	18228	36457	1.137	124.904	5179	10602	0.23	21.444	0.9977
25	22688	45382	1.156	155.256	6459	13227	0.266	25.976	0.9977
30	27148	54307	1.182	187.068	7739	15852	0.325	32.037	0.9977
35	31608	63232	1.632	217.651	9019	18477	0.457	36.725	0.9977
40	36068	72157	1.819	248.451	10299	21102	0.479	41.758	0.9977
45	40528	81082	2.11	277.156	11579	23727	0.566	47.764	0.9977
50	44988	90007	2.354	306.741	12859	26352	0.399	49.84	0.9977

Table 4.1: Verification Results for Property 1.

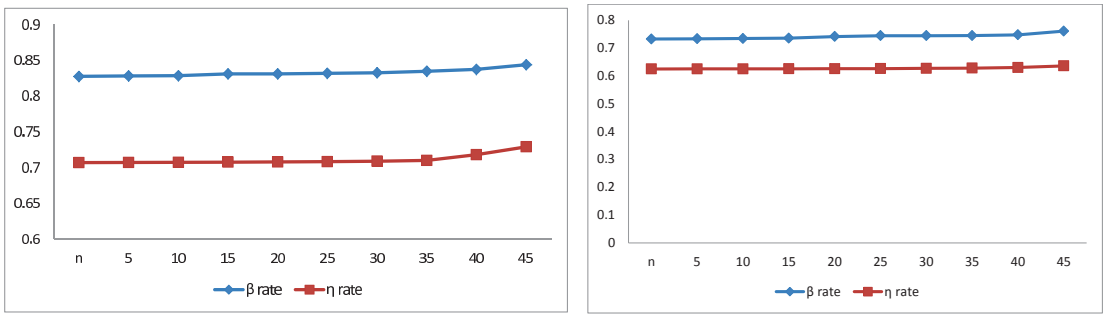
messages. After 50 messages streaming, the model checker could not establish the verification without abstraction. Then, we conclude that those properties are difficult to be verified without abstraction.

4.5 Related Work

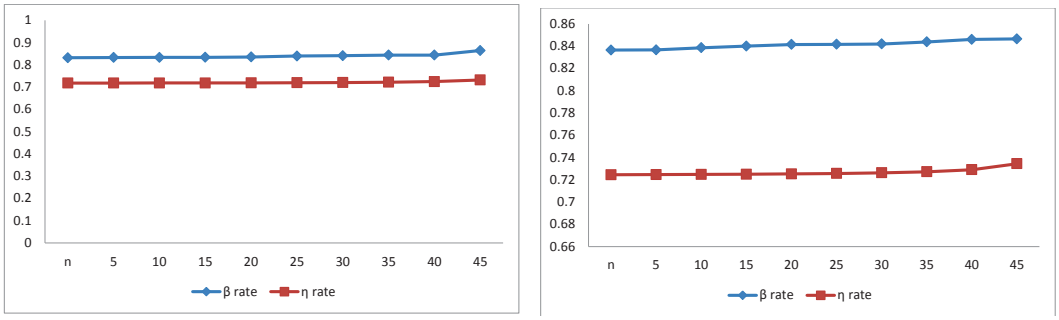
In the literature, few works examine the abstraction of SysML activity diagrams before verification and the majority rely on the implemented abstraction algorithms within the model checker. To our knowledge, some probabilistic model checkers support abstraction, for example PRISM builds the symmetry reduction and LiQuor[2] includes bi-simulation equivalences. In this section, we survey the existing initiatives that propose abstraction techniques to deal with the verification of probabilistic systems and that ones expressed as activity diagrams.

The probabilistic abstraction [4] is a probabilistic version of the partial order reduction technique on CTL. It extends the four reduction rules of the action-sets ample by two new rules to handle non-determinism and probabilistic decision in MDPs. The correctness

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



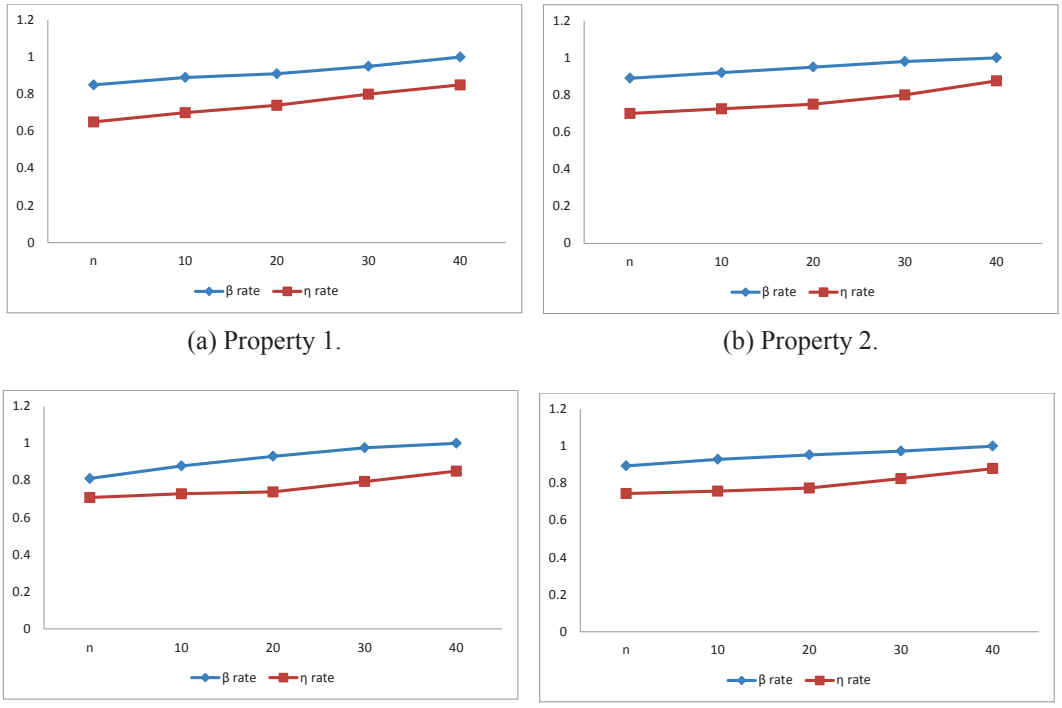
(a) Property 1. (b) Property 2.



(c) Property 3. (d) Property 4.

Figure 4.6: The Abstraction Rates for SOS.

of the extended ample-sets has been proved to preserve $PCTL^*_{\setminus X}$. [19] presents a symmetric probabilistic specification language (SPSL) close to PRISM input language. SPSL specifies MDP symmetric models and a symmetric PCTL formula is satisfied on isomorphic MDPs. For verification, SPSL specification is mapped into PRISM. [47] applies the bisimulation minimization to preserve PCTL until operator "U" on DTMCs. They use the partition refinement algorithm to obtain Markov chain bisimulation quotient. [76] applies the magnifying lens abstraction algorithm on MDPs. This algorithm is of two steps, the magnified computation is performed on regions and the sliding of the magnification in a region is performed symbolically. [48, 49] apply the predicate abstraction on MDPs. The proposed abstraction is based on stochastic games that provide distinct lower and upper bounds of probabilistic requirements. The key idea is to separate non-determinism in the abstract MDPs from the original ones.



(a) Property 1.

(b) Property 2.

(c) Property 3.

(d) Property 4.

Figure 4.7: The Abstraction Rates for RTSP.

Ober et al [58, 57] propose a framework to map UML models into communicating extended timed automata (CETA) expressed in an intermediate formal representation (IF format). To do verification, the IF validation environment incorporating a model checker and a simulation tool is used. The former implements a set of model reduction techniques including static analysis, partial order reduction and model minimization. In their work, the abstraction is performed on CETA instead of UML diagrams.

Westphal [87] proposes to exploit the symmetry of UML models on the type of object references. It uses the Rhapsody UML verification environment where the requirements are expressed in Live Sequence Charts. Query reduction and data-type reduction are proposed. Query reduction reduces quantified verification tasks if the model is symmetric in the quantified variable's type whereas data-type reduction interprets the concrete operators in an abstract domain. In contrast with our approach, the proposed abstraction therein focuses on

symmetric UML models, which represent special cases.

Prashanth and Shet [74] propose an abstraction technique for statechart models. Their approach consists of determining the set of relevant events with respect to the safety property (considering only the events that may lead to a non-safe state) and then constructing the state space accordingly.

Daoxi et al [16] propose an abstraction framework of Promela models for UML behavioral diagrams. The abstraction is driven by LTL properties on the Kripke model obtained from the Promela code. Then, the abstract Kripke model is converted to Promela code. The latter can be verified using SPIN[3] model checker. Therein, the proposed abstraction does not show the resulting abstract UML diagram. Moreover abstracting the semantics of Promela instead of its code needs more processing steps.

Xie and Browne [88, 89] propose a verification framework for executable UML (xUML) models. The properties are also expressed using xUML. Their framework includes a user-driven state space reduction procedure supporting the decomposition and the symmetry reduction. The resulting model is translated into S/R language to be verified on COSPAN model checker.

Beek et al present in [83] a framework called (UMC) for the formal analysis of concurrent systems specified by a collection of UML state machines. The formal model of a system is given by a doubly-labeled transition system (L^2TS), and the logic used to specify its properties is the state-based and event-based logic UCTL. It is an on-the-fly based analysis with a user-guided abstraction of the transition system. The main limitation of the approaches in [88, 89, 83] is that they are not fully automatic.

Del Mar Gallardo et al [91] propose a verification framework for UML behavioral diagrams. The approach is devised to abstract data and events in state chart diagrams. The first one replaces the original access definitions of variables, so the interval of their possible

[3] http://spinroot.com

values is minimized. Next, abstracting events consists of using a single event name to represent a set of real ones. In their approach, they took already abstracted models where a mother state encompasses a set of states that will be abstracted to just one state by ignoring what's inside.

R. Eshuis [22, 23] propose a translation of UML activity diagrams to NuSMV code. The proposed data abstraction is applied on guards and events. But from our experience, some activity diagrams can be poor in these two features because the activity diagram is an action-based diagram.

In Table 4.2, we compare our approach to the existing ones. We observe that few of them formalize SysML activity diagrams and prove the soundness of their proposed abstraction approaches. Moreover, our abstraction approach is efficient as it reduces the size of the model by a considerable rate. Furthermore, our mechanism allows to gain advantage from algorithms built within the tool in use.

Approach	Design	Probabilistic	Property	Formalization	Soundness
[58, 57]					
[87]	✓				
[74]	✓		✓		
[16]			✓		
[88, 89]	✓				
[83]	✓				
[91]	✓				
[22, 23]	✓				
Our	✓	✓	✓	✓	✓

Table 4.2: Comparison with the Related Work

4.6 Conclusion

In this chapter, we presented a formal abstraction framework to improve the scalability of probabilistic model-checking in general, and more especially for verifying UML and

SysML activity diagrams. The presented framework implements two algorithms, the first abstracts the irrelevant action nodes and guards in a diagram with respect to a PCTL. And, the second algorithm collapses nodes that share similar behaviors. We proved the soundness of our abstraction algorithms by defining a probabilistic weak simulation relation between the semantics of the abstract and the concrete diagrams. In addition, the preservation of the satisfaction of PCTL properties by this relation is proved. Finally, we demonstrated the effectiveness of our approach by applying it the online shopping system and the real time streaming protocol. In the next chapter, we propose a compositional verification methodology to avoid the verification of the whole diagrams.

Chapter 5

Compositional Verification of SysML Activity Diagrams

5.1 Introduction

The exhaustive research algorithms in model checking are generally 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 such as invoking behaviors in SysML activity diagrams. In this chapter, we overcome this limitation by proposing a compositional verification solution.

Figure 5.1 presents an overview of our proposed compositional verification framework. It takes a set of SysML activity diagrams composed by the call behavior interface and a PCTL [5, 27] property as input. First, we rely to the abstraction framework developed in the previous chapter that collapses similar behaviors and ignores the useless ones in diagrams with respect to a PCTL property. In addition, we propose a compositional verification approach by interface processes to distribute a PCTL property into local ones

and to verify them separately on the obtained reduced diagrams. Each single property is verified locally by using the verification framework proposed in Chapter 3. Finally, we deduce the result of the main property from the local properties results. In a nutshell, the main contributions of the present chapter can be summarized as follows: 1) Proposing an efficient verification approach that reduces the verification overhead of probabilistic model checkers, and 2) Proving the soundness of the proposed approach.

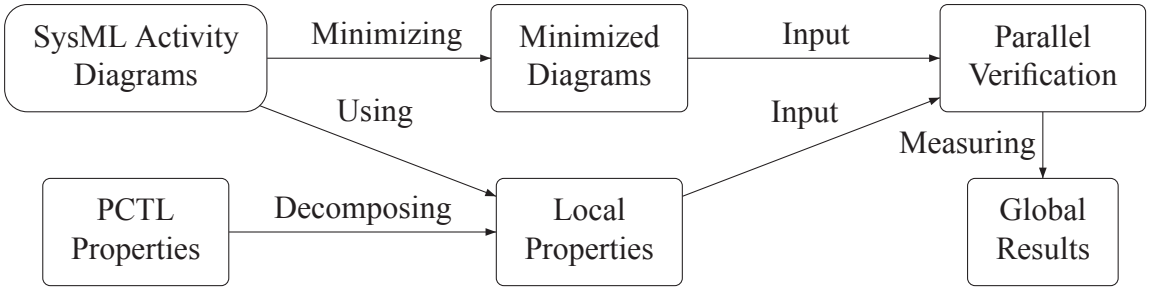


Figure 5.1: A Compositional Verification Framework.

The remainder of this chapter is organized as follows. Our compositional verification framework is detailed in the next section and Section 5.3 presents the experimental results. Finally, Section 5.5 concludes this chapter.

5.2 The Compositional Verification Approach

This section describes first our compositional approach to verify a PCTL property on a SysML activity diagram. Then, it shows how an activity diagram is mapped into PRISM. Let $\mathscr{A}$ be a SysML activity diagram with n call behaviors denoted by:

$$\mathscr{A} = \mathscr{A}_0 \uparrow_{a_0} \mathscr{A}_1 \cdots \mathscr{A}_{i-1} \uparrow_{a_{i-1}} \mathscr{A}_i \cdots \mathscr{A}_{n-1} \uparrow_{a_{n-1}} \mathscr{A}_n.$$

First, we reduce the diagram $\mathscr{A}$ by applying NuAC axioms and introducing Proposition 5.1 that ignores behaviors $\mathscr{A}_i$ that are not influenced by the property ϕ to be verified.

Proposition 5.1. *Let $\mathscr{A}$ be a diagram that contains n behaviors, AP_ϕ is the atomic propositions of the PCTL property ϕ and $AP_{\mathscr{A}_i}$ is the atomic propositions of the behavior diagram $\mathscr{A}_i$. Then, we have:*

$$\forall \mathscr{A}_{i:0\leq i\leq n}, AP_\phi \cap AP_{\mathscr{A}_{i:0\leq i\leq n}} = \emptyset : \mathscr{A}_i = \varepsilon \wedge [(\mathscr{A}_0 \uparrow_{a_0} \mathscr{A}_1 \cdots \mathscr{A}_{n-1} \uparrow_{a_{n-1}} \mathscr{A}_n) \models \phi] \Rightarrow [\mathscr{A} \models \phi].$$

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

For $\forall \mathscr{A}_{i:0\leq i\leq n}, AP_\phi \cap AP_{\mathscr{A}_{i:0\leq i\leq n}} = \emptyset : \mathscr{A}_i = \varepsilon$, we will have: $\mathscr{A}_0 \uparrow_{a_0} \mathscr{A}_1 \cdots \mathscr{A}_{k-1} \uparrow_{a_{k-1}} \mathscr{A}_k)$.

Let $[(\mathscr{A}_0 \uparrow_{a_0} \mathscr{A}_1 \cdots \mathscr{A}_{k-1} \uparrow_{a_{k-1}} \mathscr{A}_k) \models \phi]$

$\Leftrightarrow \exists m,\ \forall j < m : \pi(j) \models \phi_1 \wedge \pi(m) \models \phi_2$.

By calling $\mathscr{A}_i$ in a_i using BH-1, the only changes in π are the propositions of $\mathscr{A}_i$ till executing BH-2, then:

$\exists m' \geq m,\ j' \geq j,\ \forall j' < m' : \pi(j') \models \phi_1 \wedge \pi(m') \models \phi_2$

$\Leftrightarrow\ \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k \ldots \uparrow_{a_i} \mathscr{A}_i \models \phi$.

By calling new $\mathscr{A}_{i+1}$ in a_{i+1} up to n, we will have:

$\exists m'' \geq m',\ j'' \geq j',\ \forall j'' < m'' : \pi(j'') \models \phi_1 \wedge \pi(m'') \models \phi_2$

$\Leftrightarrow \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_n} \mathscr{A}_n \models \phi \Leftrightarrow \mathscr{A} \models \phi_1 \mathrm{U} \phi_2$.

For $\phi_1 \mathrm{U}^{\leq k} \phi_2$ and $\mathrm{X}\phi$ cases, we deduce the following.

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

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

□

To 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 "♮"

proposed in Definition 5.1. 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. It uses the π-calculus substitution notation $Q[z/y]$ which means in the structure Q, the term z is substituted for the term y.

Definition 5.1 (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 $\phi = \phi_1 \natural_a \phi_2$ 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 prove Property 5.1.

Property 5.1. *The decomposition operator "$\natural$" is commutative and associative, i.e.*

1. *For $\mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2$, the decomposition operator $\natural$ is commutative: $\phi_1 \natural_{a_1} \phi_2 \equiv \phi_2 \natural_{a_1} \phi_1$.*
2. *For $\mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2 \uparrow_{a_2} \mathscr{A}_3$, the decomposition operator $\natural$ is associative: $\phi_1 \natural_{a_1} (\phi_2 \natural_{a_2} \phi_3) \equiv (\phi_1 \natural_{a_1} \phi_2) \natural_{a_2} \phi_3$.*

Proof. 1. The commutativity of $\natural$ is proved by constructing $\Phi_1 = \phi_1 \natural \phi_2$ and $\Phi_2 = \phi_2 \natural \phi_1$, then comparing Φ_1 and Φ_2.

2. Similarly to the commutativity, the associativity of $\natural$ is proved.

□

For the verification of ϕ on $\mathscr{A}_1 \uparrow_{a_1} \mathscr{A}_2$, Theorem 5.1 deduces the satisfiability of ϕ from the satisfiability of local properties ϕ_1 and ϕ_2 obtained by $\natural$.

Theorem 5.1 (Compositional Verification-CV)**.** *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 Proposition 5.1 follows a structural induction on the PCTL structure by using Definition 5.1.

As an example, we take the until operator "U". Let $\phi = ap_1$ U ap_2 where $ap_1 \in AP_{\mathscr{A}_1}$ and $ap_2 \in AP_{\mathscr{A}_2}$.

By applying Definition 5.1, we have: $\phi_1 = ap_1$ U a_1 and $\phi_2 = \top$ U ap_2. Let $\mathscr{A}_1 \models \phi_1 \Leftrightarrow \exists m_1,\ \forall j_1 < m_1 : \pi_1(j_1) \models ap_1 \wedge \pi_1(m_1) \models ap_1 \wedge a_1$.

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 5.2 generalizes Theorem 5.1 to support the satisfiability of ϕ on an activity diagram with n behaviors.

Proposition 5.2 (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:*

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

Proof. We prove Proposition 5.2 by induction on n.

- The base step where "$n = 1$" is proved by Theorem 5.1.
- For the inductive step, first, we assume:

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

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 5.1, then:

$$\frac{\begin{array}{cc}\mathscr{A}' \models \phi' & \mathscr{A}_{n+1} \models \phi_{n+1} \\ \mathscr{A} = \mathscr{A}' \uparrow_{a_n} \mathscr{A}_{n+1} & \phi = \phi' \natural_{a_n} \phi_{n+1}\end{array}}{\mathscr{A} \models \phi}$$

□

5.3 Experimental Results

In the purpose of providing experimental results demonstrating the efficiency and validity of our framework, we verify a set of PCTL properties on the online shopping system [32] studied in the previous chapters. To this end, we compare the results "β" and the verification cost in terms of the model size [1] "γ" and the verification time "δ" (sec) with and without applying our approach.

For the diagram depicted in Figure 3.7a, the call behaviors action nodes: "Browse Catalogue", "Make Order", "Process Order" and "Shipment" are denoted by a, b, c and d, respectively. For simplicity, we take this order to denote their associated diagrams by $\mathscr{A}_1$ to $\mathscr{A}_4$, respectively, where $\mathscr{A}_0$ denotes the main diagram. As example, Figure 3.7b expands the call behavior action "Process Order" 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 the following properties that are expressed in PCTL.

[1] The model size is the number of transitions (edges).

Property Φ_1. "For each order, what is the minimum probability value to make a delivery after browsing the catalogue?"

$$Pmin = ?[(Browse\,Catalogue) \Rightarrow (F(Delivery))].$$

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 on $\mathscr{A}$. Then, by using Proposition 5.1, we reduce the verification of Φ on $\mathscr{A}$ to Φ on $\mathscr{A}_0 \uparrow_c \mathscr{A}_3$. By using the decomposition rules of Definition 5.1, 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 5.1 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 $\beta(\Phi_1)$=$\beta(\Phi_{11})\times\beta(\Phi_{12})$. In addition, the size of the diagrams is minimized $(\gamma(\Phi_{11}) + \gamma(\Phi_{12}) < \gamma(\Phi_1))$. Consequently, the verification time is reduced significantly: $(\delta(\Phi_{11}) + \delta(\Phi_{12}) \ll \delta(\Phi_1))$.

Property Φ_2. "For each order, what is the maximum probability value to confirm a shipment?"

$$Pmax = ?[G((CreateDelivery) \Rightarrow F(ConfirmShipment)].$$

The propositions of this property "CreateDelivery" and "ConfirmShipment" belong to $\mathscr{A}_2$, and $\mathscr{A}_4$, respectively. Similarly to the verification Φ_1, we verify Φ_2 on $\mathscr{A}$. Then, we decompose Φ_2 to Φ_{21}, 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 as follows.

$$\Phi_{21}:\ Pmax = ?[G((CreateDelivery) \Rightarrow F(Shipment)].$$

$$\Phi_{22}:\ Pmax = ?[G((True) \Rightarrow F(ConfirmShipment)].$$

n	1	2	3	4	5	6	7	8	9	10
$\beta(\Phi_1)$	0.76	0.76	0.76	0.76	0.76	0.76	0.76	0.76	0.76	0.76
$\gamma(\Phi_1)$	41982	606470	2213880	4823290	8434700	13048110	51145160	202489260	454033360	805777460
$\delta(\Phi_1)$	0.872	3.12	10.764	24.364	44.098	72.173	358.558	1818.247	6297.234	17761.636
$\beta(\Phi_{11})$	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
$\gamma(\Phi_{11})$	1926	3706	5486	7266	9046	10826	12606	14386	16166	17946
$\delta(\Phi_{11})$	0.46	0.64	1.09	3.12	7.511	12.86	27.03	54.38	111.74	163.89
$\beta(\Phi_{12})$	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
$\gamma(\Phi_{12})$	12	12	12	12	12	12	12	12	12	12
$\delta(\Phi_{12})$	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005

Table 5.1: The Verification Cost for Properties Φ_1, Φ_{11}, and Φ_{12}.

j	1	2	3	4	5	6	7	8	9	10
$\beta(\Phi_2)$	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377
$\gamma(\Phi_2)$	41982	606470	2213880	4823290	8434700	13048110	51145160	202489260	454033360	805777460
$\delta(\Phi_2)$	4.217	11.458	33.394	78.746	168.649	354.211	2280.252	17588.755	34290.635	63097.014
$\beta(\Phi_{21})$	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377	0.9377
$\gamma(\Phi_{21})$	4808	7211	9614	12017	14420	16823	19226	21629	24032	26435
$\delta(\Phi_{21})$	0.926	1.584	4.775	12.301	32.852	83.337	274.9	450.81	586.43	652.76
$\beta(\Phi_{22})$	1	1	1	1	1	1	1	1	1	1
$\gamma(\Phi_{22})$	9	9	9	9	9	9	9	9	9	9
$\delta(\Phi_{22})$	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003

Table 5.2: The Verification Cost for Properties Φ_2 Φ_{21}, and Φ_{22}.

Table 5.2 shows the verification results and costs of Φ_2 on $\mathscr{A}$, Φ_{21} $\mathscr{A}_0 \uparrow_b \mathscr{A}_2$, and Φ_{22} on $\mathscr{A}_4$ for different values of the number of orders "n". We found that $\beta(\Phi_2) = \beta(\Phi_{21}) \times \beta(\Phi_{22})$, $\gamma(\Phi_{21}) + \gamma(\Phi_{22}) < \gamma(\Phi_2)$ and $\delta(\Phi_{21}) + \delta(\Phi_{22}) \ll \delta(\Phi_2)$.

5.4 Related Work

To compare our work to the existing ones, we survey the verification initiatives dedicated mainly to SysML activity diagrams and the probabilistic compositional verification. [18, 42] map the basic artifacts of a single SysML activity diagram into PRISM code where both of them inherit PRISM verification limitations. [67] introduces the abstraction by state merging to reduce the verification cost of SysML activity diagrams whereas [68] improves the abstraction technique in [67] by proposing the abstraction by restriction on more behaviors. Concerning the compositional verification for probabilistic systems, [26] discusses assume-guarantee technique for probabilistic system by focusing more on the leaning algorithm to generate the minimal deterministic automata that represents a probabilistic safety property. [25] proposes assume-guarantee where both assumption and guarantee properties are probabilistic safety properties such that assumptions are generated manually. [24] applies 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[2] implements the bi-simulation equivalence.

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

5.5 Conclusion

In this chapter, we presented a compositional verification framework to improve the efficiency and the scalability of probabilistic model-checking. More specifically, our target was verifying systems modeled using SysML activity diagrams. The presented framework is based on abstraction by ignoring and merging behaviors that are irrelevant to a given PCTL property. Moreover, we introduced a probabilistic compositional verification approach based on decomposing a global PCTL property into local ones with respect to the interfaces between diagrams. We proved the soundness of the proposed framework by showing that the satisfaction relation of the PCTL properties are preserved. Furthermore, we proposed a semantic for SysML activity diagrams that helps on proofs and to encode easily the diagrams in PRISM. In addition, we demonstrated the effectiveness of our approach by applying it on the online shopping system. In the next chapter, we propose a security specification by using SysML activity diagrams that helps to express the system requirements automatically.

Chapter 6

Security Specification of SysML Activity Diagrams

6.1 Introduction

In this chapter, we address the issue of security specification and security risk assessment of software/systems modeled by using SysML activity diagrams. The goal is to gauge how well a product is meeting its security requirements. Our security specification framework uses the verification framework introduced in the previous chapter to assess security risk. It is composed of two stages: specification and evaluation as depicted in Figure 6.1. The former is to express easily the system requirements by introducing potential attack templates, whereas the latter is a verification approach to evaluate their security level. First, we use SysML activity diagrams to develop a library of CAPEC attack application-independent templates with varied potential gains that can exploit the system vulnerabilities. By using this library, application-dependent attack scenarios are instantiated and the interaction between the attack and the system diagrams is defined. In order to subject this interaction

to the verification framework (Chapter 3), the interaction is defined and the security requirements are represented by SysML activity diagrams. Then, a specification algorithm is proposed to generate their equivalent PCTL expressions. Finally, the verification framework produces the probability value that represents the satisfaction degree of the generated security property on the system under test.

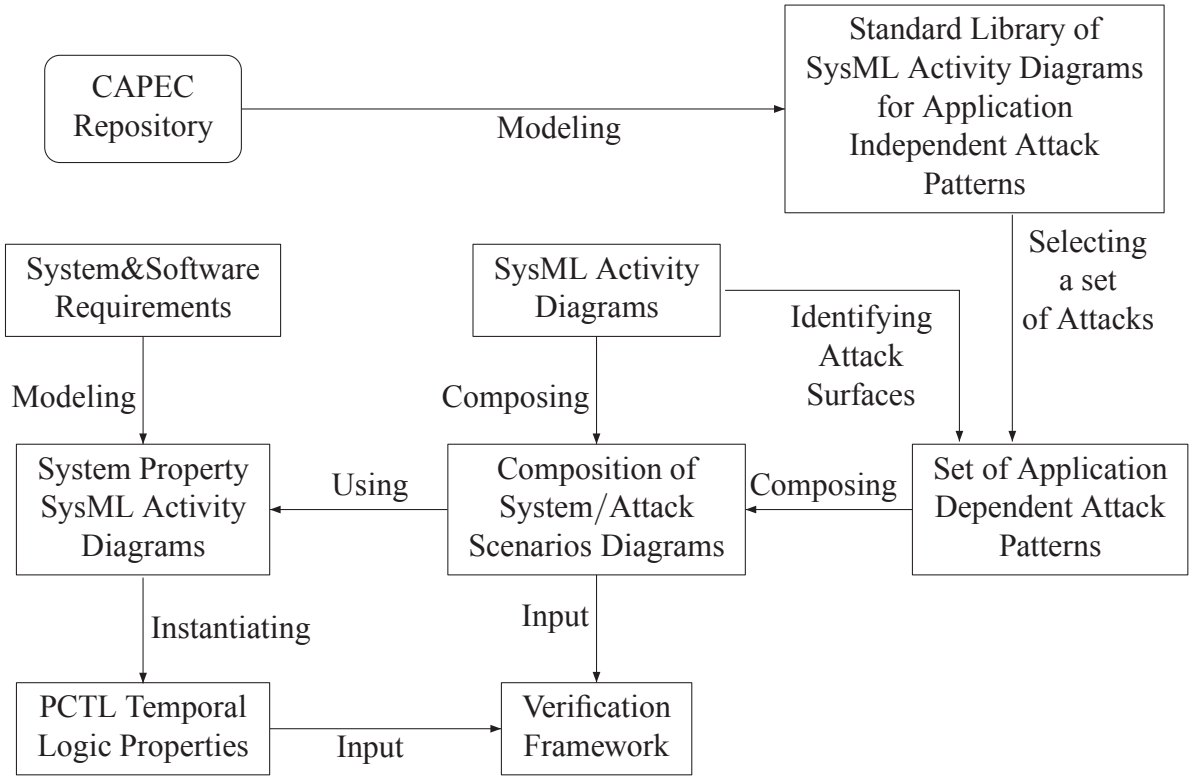


Figure 6.1: Security Risk Assessment Framework.

The remainder of this chapter is organized as follows. Section 6.2.3 presents the attack patterns. Our framework that contains the security specification and evaluation is detailed in Section 6.3. Section 6.4 describes the experimental results and Section 6.5 surveys the existing related work. Finally, Section 6.6 concludes this chapter.

6.2 Software Security

In this chapter, we introduce the concept of security requirements and security attack models. The, we model a standard catalogue of attacks.

6.2.1 Security Properties

A software should satisfy its related security requirements to avoid the exploitation of vulnerabilities by attacks and threats. These requirements are also called security properties and some of them are used by measure community to evaluate security. Here, we cite the main known security properties:

- **Authentication**. It identifies an entity before granting access to a system's resource.
- **Confidentiality**. It is the concealment of information or resources. The data is disclosed only as intended by the enterprize.
- **Integrity**. The integrity of a variable is preserved if it doesn't take any value different from the ones it should have at any time during the execution of a system.
- **Availability**. It is the property that enterprize assets, including business processes, will be accessible when needed for authorized use.
- **Secrecy**. A system preserves the secrecy of an information, if, it does not appear for an adversary.
- **Authenticity**. The message authenticity is to secure the information on the message origin.
- **Freshness**. A message is fresh, only if, it is unpredictable and it has never appeared during the system execution.

6.2.2 Attack Scenario

Here, we present the attack behavior and show its impact on a system. As defined in [1], an attack is an attempt to gain unauthorized access to an Information System's (IS) services, resources, or information or the attempt to compromise an IS's integrity, availability, or confidentiality, as applicable. Different attack models have been deployed such as: attack tree, attack graph, and network attack graph.

- An attack tree [52] is a tree where the nodes represent attacks. The root node of the tree is the global goal of the attack. Children of a node are refinements of this goal, and leafs therefore represent attacks that can no longer be refined. A refinement can be done by either an aggregation or a choice.

- An attack graph [78] is a graph where each vertex represents the entire network state and the arcs represent state transitions caused by an attacker's actions. Also, a vertex can not represent the entire state of a system but rather a system condition in a predicate form and the arcs represent the relations between the system conditions. In this case, the attack graph is called a dependency attack graph.

- A network attack [80] is an attack model [80] composed of the computer network, the attacker, and the defender. A state transition in a network attack model corresponds to a single action by the intruder, a defensive action by the system administrator, or a routine network action.

6.2.3 Standard Attack Patterns

This section presents the concept of attack patterns, introduces the notion of probabilistic likelihood of an attack and formalizes attack patterns as SysML activity diagrams.

Software security attack patterns are currently being researched and designed to expose exploited code development flaws and describe the common methods, techniques, and logic that attackers use to exploit software [31]. Many researches and organizations such as CAPEC and WASC[1] show special interest in attack patterns. They are expressed in a textual format and described using a set of predefined elements including the attack goal, the preconditions, the attack steps and the post-conditions of a successful attack.

Particularly, CAPEC is a software assurance strategic initiative that provides a publicly available catalog of attack patterns. It counts 311 attack patterns (each identified using a unique identifier) organized into 67 categories and sub-categories. The CAPEC attack patterns are documented according to a standard schema devised by CAPEC that includes both primary and supporting schema elements. Primary schema elements include the pattern id, the description of the attack, related weaknesses, typical severity, likelihood of exploitation, attack surface, and abstraction level. The supporting schema elements are categorized into describing, diagnosing, and enhancing information. Our first objective is to model attack patterns using SysML activity diagrams. To this end, we propose the template of attack patterns described as a SysML activity diagram shown in Figure 6.2.

Each concrete attack pattern is built by instantiating this template and specifying the call behavior action denoted by "Pattern Behavior". The main control flow in this template is a probabilistic decision used to specify the likelihood of the attack occurrence, which corresponds to the probability "P". The value of "P" is determined based on the "typical likelihood of exploitation" schema element provided within CAPEC catalogue. However, this schema element is a qualitative description of the likelihood that ranges from "low" to "high". In order to quantify this attribute, we propose to assign ranges of probabilities to each qualitative description based on the standard of security risk management [39] in

[1] http://www.webappsec.org, The Web Application Security Consortium.

combination with the Kent's Words of Estimative Probability[2]. The probability ranges corresponding to the estimative terms are specified in Table 6.1. The probability related to the instantiated attack pattern is obtained by a centroid defuzzification function[3].

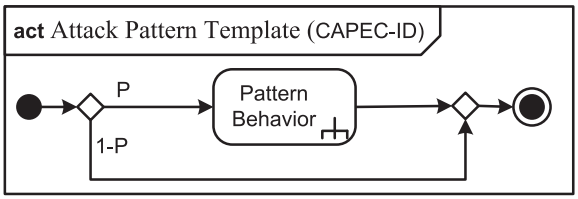


Figure 6.2: SysML Activity Diagram of the Attack Pattern Template.

Table 6.1: Probability Values Scale.

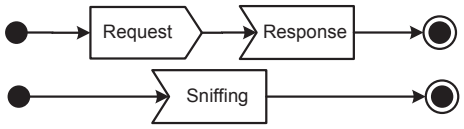
CAPEC terms	Kent's Estimative terms	Probability values
High	Certain	100
High	Almost Certain	93% (±6%)
Medium to High	Probable	75% (±12%)
Medium	Chances About Even	50%(±10%)
Low to Medium	Probably Not	30%(±10%)
Low	Almost Certainly Not	7%(±5%)
Low	Impossible	0

Since not all attacks in CAPEC can be applied at the design level, we need to select attack patterns that are technology and platform independent. Thus, we rely on the "pattern abstraction level" schema element to select the applicable attack patterns. The latter schema element defines three levels of abstraction: meta, standard, and detailed. Particularly, we focus on the standard level, which corresponds to a typically functional context-dependent but technology context-independent attack pattern. In the following, we provide the set of the considered CAPEC attack patterns with their corresponding activity flows representing the pattern behavior and their associated likelihoods.

[2]https://www.cia.gov/library; Words of Estimative Probability.
[3]http://www.osti.gov; Office of Scientific and Technical Information.

- Spoofing (CAPEC-156): An attacker builds a message such that it is able of masquerading an authorized message from a trusted principal. As a result, consumers of these messages can be manipulated into responding or processing the deceptive message. It may refer to spoofing the content (CAPEC-148) or the id (CAPEC-151). Their pattern is depicted by the following figure such that P(CAPEC-148)=P(CAPEC-151)=0.8.

- Data Leakage (CAPEC-118): In this class, the attacker uses well-formed requests to get sensitive information by exploiting weaknesses in the design. The attacker may collect this information through a variety of methods including active querying as well as passive observation. Three techniques are used in this class: Data excavation (CAPEC-116), Data interception (CAPEC-117), and Sniffing (CAPEC-148). CAPEC-116 and CAPEC-117 are presented by the first control flow with P(CAPEC-116)=0.5 and =P(CAPEC-117)=0.5. Also, CAPEC-148 is illustrated in the second control flow with a probability value P(CAPEC-148)=0.2.



- Resource Depletion (CAPEC-119): The attacker depletes a resource to the point that the target's functionality is affected. The result is usually the degradation or denial of one or more services offered by the target. In order to deplete the target's resources, the attacker can achieve its objective through flooding (CAPEC-125), through leak (CAPEC-131) by uploading a malicious file, or through allocation (CAPEC-131) by sending a formatted request. The pattern of these attacks is depicted by the following figure and launched by these probability values: P(CAPEC-125$|n \geq m$)= 0.8,

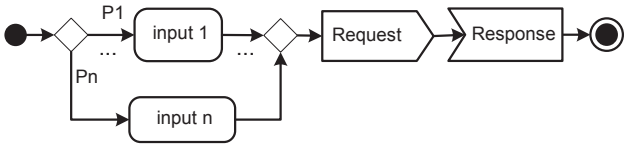
P(CAPEC-131$|n < m$)=0.8 where n is the number of requests and m is a number fixed by the designer.

- Injection (CAPEC-152): The attacker is able to control or disrupt the behavior of a target through crafted input data submitted using an interface functioning to process data input. Different resource-dependent patterns are detailed in CAPEC and abstracted to design level such as SQL (CAPEC-66), email (CAPEC-134), format string (CAPEC-135), LDAP (CAPEC-136), resource injection (CAPEC-240), script injection (CAPEC-242), and command injection (CAPEC-248). All of them take the form of the following control flow with a probability value equal to 0.8.

- Exploitation of Authentication (CAPEC-225) takes advantage of weaknesses related to authentication mechanisms including authentication bypass by spoofing (CWE-ID-290), authentication bypass by assumed immutable data (CWE-ID-302), and origin validation error (CWE-ID-346). Particularly, its descendent sub-category CAPEC-21 aims at exploiting session variables, resource IDs and other trusted credentials to take advantage of the fact that some software accepts user input without verifying its authenticity. They have the following work flow with P = 0.8.

- Fuzzing (CAPEC-28): It is part of the probabilistic techniques (CAPEC-223) and it is inspired by a software testing method. The attacker provides randomly generated input to the system and looks for an indication to identify weaknesses in the

system. The pattern of this attack is depicted by the following control flow such that P(CAPEC-28) = 0.8 and P_1, P_2,$\cdots$, P_n are probability values (e.g. uniformly distributed).



6.3 Security Properties Specification

In this section, we detail the specification and the evaluation procedures of the proposed framework presented in Section 6.1.

6.3.1 The Security Requirements Specification

This section describes the specification of SysML activity diagrams $\mathscr{A}$ as PCTL temporal logic expressions. Algorithm 4 illustrates the specification procedure Ξ that takes $\mathscr{A}$ as input and produces its equivalent PCTL expression denoted by *PCTLexp*. The procedure Ξ parses the diagram $\mathscr{A}$ with a complexity of $\mathscr{O}(|n|^2)$ by using a depth-first search where $|n|$ is the number of activity nodes in $\mathscr{A}$.

The procedure Ξ is described as follows. First, the initial node is pushed into the stack of nodes denoted by *nodes* (line 9). While the stack is not empty (line 10-27), the algorithm pops a node from the stack *nodes* into the current node denoted by *cNode* (line 12). Then, the current node is added into the list *vNode* of visited nodes (line 15) if it is not already visited (line 13). The PCTL expressions denoted by *PCTLexp* is constructed by calling the function Λ (line 19) that has two arguments which are the current node *cNode* and its successors saved in the list *nNode* (line 17). The explored successors are pushed into

the stack *nodes* (line 22-24), then, they are cleared from the list *nNode* (line 26). Finally, the algorithm terminates when all nodes are visited.

Algorithm 4 Specification Algorithm Ξ of SysML Activity Diagrams into PCTL Expression.

Input: SysML activity diagrams $\mathscr{A}$.
Output: PCTL expression *PCTLexp*.

```
 1: nodes as Stack;                          ▷ A stack of nodes which is initially empty.
 2: cNode as Node;                           ▷ The current node which is initially empty.
 3: nNode, vNode as list_of_Node;            ▷ List of nodes that are initially empty.
 4: procedure Ξ(𝒜)
 5:     nodes.push(in);                      ▷ Push the initial node in the stack nodes.
 6:     while not nodes.empty() do           ▷ Pop the current node.
 7:         cNode := nodes.pop();
 8:         if cNode not in vNode then       ▷ Consider the current node as a visited node.
 9:             vNode.add(cNode);            ▷ Get the successors of the current node.
10:             nNode := cNode.successors();
                                             ▷ Call the mapping rules function.
11:             PCTLexp.add(Λ(cNode,nNode));
12:         end if                           ▷ Stores all newly discovered nodes.
13:         for all n in nNode do
14:             nodes.push(n);
15:         end for                          ▷ Empty the list nNode.
16:         nNode.clear();
17:     end while
18: end procedure
```

The function Λ presented in Listing 6.1 produces the appropriate PCTL expression for the current node. It takes into consideration the current node and its successors, both of them constitute the atomic propositions of the resulted PCTL property. The selected PCTL operators (X, $U^{\leq k}$ or U) to express the produced property depends on the structure of the current node and the semantic operational rules developed by NuAC. The rule 1 (line 3) expresses the existence of an element "$\mathscr{N}$" during the execution of a SysML activity diagram, whereas the rule 2 (line 7) shows the execution of successive elements "$\mathscr{N}_1$" and "$\mathscr{N}_2$" that both of them have one output and one input, respectively. The rule 4 (line 18) specifies the properties of a guarded decision node and the rule 3 (line 12) is a probabilistic

version of the previous rule (rule 4). The rule 5 in line 25 denotes properties proper to a merge node and the rule 6 in line 31 specifies properties related to a fork node. The rule 7 in line 38 describes the properties related to a join node. Finally, the rule 8 in line 46 measures the minimum/maximum probability of the satisfaction of a generated PCTL property. The min/max probability values are considered with respect to the non-determinism in the system/attack interaction. Each rule is referenced by its order, for example "$l \Rightarrow \mathrm{F}(L(\mathscr{N}_2))$" (line 9) is denoted by Rule 2-b and $L(\mathscr{N}_2)$ returns a label of the term $\mathscr{N}_2$.

Listing 6.1: PCTL Specification Function.

```
1  Λ : 𝒜 ↦ ϕ
2  Λ(𝒜) = ∀n ∈ 𝒜, Case (n) of
3  l : 𝒩  ⇒
4        in
5           {X(l)} ∪ {F(l)} ∪ {⊤ U^≤k l}
6        end
7  l : 𝒩1 ↣ 𝒩2  ⇒
8        in
9           {l ⇒X(L(𝒩2))} ∪ {l ⇒F(L(𝒩2))} ∪ {l U L(𝒩2)}
10          ∪ {l U^≤k L(𝒩2)}∪ Λ(𝒩1) ∪ Λ(𝒩2)
11       end
12 l : D(p,g,𝒩1,𝒩2) ⇒
13       Case (p) of
14          ]0,1[  ⇒
15                in
16                   {P(Λ(l : D(g,𝒩1,𝒩2))) ⋈ pi}
17                end
18       Otherwise ⇒
19                in
20                   {l ⇒X(L(𝒩1) ∨ L(𝒩2))} ∪ {l ⇒F(L(𝒩1))}
21                   ∪ {l ⇒F(L(𝒩2))} ∪ {g ⇒X(L(𝒩1))}
```

$\cup \{g \Rightarrow \mathrm{F}(L(\mathscr{N}_1))\} \cup \{\neg g \Rightarrow \mathrm{X}(L(\mathscr{N}_2))\}$

$\cup \{\neg g \Rightarrow \mathrm{F}(L(\mathscr{N}_2))\} \cup \Lambda(\mathscr{N}_1) \cup \Lambda(\mathscr{N}_2)$

end

$l : M(x,y) \rightarrowtail \mathscr{N} \Rightarrow$

in

$\{l_x \Rightarrow \mathrm{X}(L(\mathscr{N}))\} \cup \{l_x \Rightarrow \mathrm{F}(L(\mathscr{N}))\} \cup \{l_x \,\mathrm{U}\, L(\mathscr{N})\}$

$\cup \{l_x \mathrm{U}^{\leq k} \; L(\mathscr{N})\} \cup \{l_y \Rightarrow \mathrm{X}(L(\mathscr{N}))\} \cup \{l_y \mathrm{U}^{\leq k} L(\mathscr{N})\}$

$\cup \{l_y \Rightarrow \mathrm{F}(L(\mathscr{N}))\} \cup \{l_y \; \mathrm{U} \; L(\mathscr{N})\} \cup \Lambda(\mathscr{N})$

end

$l : F(\mathscr{N}_1, \mathscr{N}_2) \Rightarrow$

in

$\{l \Rightarrow \mathrm{X}(L(\mathscr{N}_1) \wedge L(\mathscr{N}_2))\} \cup \{l \Rightarrow \mathrm{X}(L(\mathscr{N}_1)\}$

$\cup \{l \Rightarrow \mathrm{X}(L(\mathscr{N}_2)\} \cup \{l \Rightarrow \mathrm{F}(L(\mathscr{N}_1) \wedge L(\mathscr{N}_2))\}$

$\cup \{l \Rightarrow \mathrm{F}(L(\mathscr{N}_1))\} \cup \{l \Rightarrow \mathrm{F}(L(\mathscr{N}_2))\}$

$\cup \Lambda(\mathscr{N}_1) \cup \Lambda(\mathscr{N}_2)$

end

$l : J(x,y) \rightarrowtail \mathscr{N} \Rightarrow$

in

$\{l_x \Rightarrow \mathrm{X}(L(\mathscr{N}))\} \cup \{l_x \Rightarrow \mathrm{F}(L(\mathscr{N}))\}$

$\cup \{l_x \,\mathrm{U}\, L(\mathscr{N})\} \cup \{l_x \,\mathrm{U}^{\leq k} \, L(\mathscr{N})\} \cup \{l_y \Rightarrow \mathrm{X}(L(\mathscr{N}))\}$

$\cup \{l_y \Rightarrow \mathrm{F}(L(\mathscr{N}))\} \cup \{l_y \,\mathrm{U}\, L(\mathscr{N})\} \cup \{l_y \,\mathrm{U}^{\leq k} \, L(\mathscr{N})\}$

$\cup \{l_x \wedge l_y \Rightarrow \mathrm{X}(L(\mathscr{N}))\} \cup \{l_x \wedge l_y \Rightarrow \mathrm{F}(L(\mathscr{N}))\}$

$\cup \{l_y \Rightarrow \; \mathrm{U}\, L(\mathscr{N})\} \cup \Lambda(\mathscr{N})$

end

Otherwise

in

$\{Pmin = ?[\mathrm{G}(\Lambda(\mathscr{N}))]\} \cup \{Pmin = ?[\mathrm{F}(\Lambda(\mathscr{N}))]\}$

$\cup \{Pmin = ?[\mathrm{GF}(\Lambda(\mathscr{N}))]\} \cup \{Pmax = ?[\mathrm{G}(\Lambda(\mathscr{N}))]\}$

$\cup \{Pmax = ?[\mathrm{F}(\Lambda(\mathscr{N}))]\} \cup \{Pmax = ?[\mathrm{GF}(\Lambda(\mathscr{N}))]\}$

$\cup \{Pmin = ?[(\Lambda(\mathscr{N}))]\} \cup \{Pmax = ?[(\Lambda(\mathscr{N}))]\}$

52 **end**

6.3.2 The Security Requirements Evaluation

In this section, we define the interaction operator between the system and the attack diagrams. For evaluation, we express this interaction in PRISM.

Basically, a system's attack surface is the way in which an adversary can interact with the system and potentially cause damage to it. The larger the attack surface, the greater will be the number of potential attacks. The definition of an attack surface for an activity diagram is given by Definition 6.1.

Definition 6.1 (NuAC Attack Surface)**.** An attack surface of a NUAC term $\mathscr{A}$ is the tuple $\text{AS}_{\mathscr{A}} = (N,\, X,\, O,\, Ch)$, where:

- N is the set of entry points that can receive signal and object tokens,
- X is the set of exit points that produces tokens,
- O is the set of un-trusted data/tokens,
- $Ch : N \times X \rightarrow O$ is a function assigning to each entry and exit point a token/object.

To define the global model, we introduce the concept of interaction interface denoted by $\mathscr{I}$ between two SysML activity diagrams $\mathscr{A}_1$ and $\mathscr{A}_2$. It is based on the attack surfaces of the interacting diagrams. Specifically, it matches exit points of one activity diagram to the entry points of the other. The interaction interface is given in Definition 6.2.

Definition 6.2 (NuAC Interaction Interface)**.** An interaction interface between two SysML activity diagrams $\mathscr{A}_1$ with an attack surface $AS_{\mathscr{A}_1}$ and $\mathscr{A}_2$ with an attack surface $AS_{\mathscr{A}_2}$ is a set of triplet $\mathscr{I} = \{(n,\, x,\, o) \in N_{\mathscr{A}_1} \times X_{\mathscr{A}_2} \times O_2 \cup N_{\mathscr{A}_2} \times X_{\mathscr{A}_1} \times O_1 | Ch_1(n) = Ch_2(x) = o\}$.

In our perspective, we consider the interaction between the system and the attack scenario. Thus, given the system diagram $\mathscr{A}_1$, the attack diagram $\mathscr{A}_2$, and their interaction interface $\mathscr{I}$, the global diagram can be written as follows $\mathscr{A} = \mathscr{A}_1 \parallel_{\mathscr{I}} \mathscr{A}_2$. We define the composition in PRISM as a system combined by the synchronization CSP algebraic operator as presented in Listing 6.2.

Listing 6.2: PRISM Fragment code for the Composition

```
1
2  // PRISM Fragment code for the  diagram A_i
3
4        Module A_i
5        //Variable initialization.
6        l_0 bool init true;
7        ...
8        l_i bool init false;
9        ...
10       l_n bool init false;
11       //Module commands.
12       //Initial node command.
13       [l_0] l_0 -> (l_0' = false)&(L(N)' = true);
14       ...
15       //A probabilistic node command.
16       [l_i] g_i -> ...+p_k : u_k+...;
17       ...
18       //A non probabilistic node command.
19       [l_j] g_j -> u_m;
20       ...
21       //Final node command.
22       [l_n] l_n -> (l_0' = false)&...&(l_n' = false);
23       endModule
```

```
24
25  // PRISM Fragment code for the composition A1 ||_I A2
26      system
27      A1|[L(n1)_L(x1),L(n2)_L(x2)]|A2
28      endsystem
```

6.4 Experimental Results

We apply the developed approach in this chapter on the Real Time Streaming Protocol. First, we describe the builded attack scenario. Then, We evaluate the proposed security properties.

6.4.1 Attack Scenario

In Figure 6.3, we present the SysML activity diagram that describes the behavior of the attack scenario specific to RTSP application. It is composed of the attack patterns CAPEC-(156 ,118, 119, 225). The main abilities of these attacks are: flooding the server by sending DESCRIBE messages (CAPEC-119), hijacking users sessions by modifying SETUP messages (CAPEC-156), disturbing the service by sending PAUSE message after a successful hijacking session (CAPEC-156), illegitimate access to media by sending PLAY message after a successful session hijacking (CAPEC-118), sniffing the media in transit (CAPEC-156), and interrupting the service by sending TEARDOWN message (CAPEC-119). The generated attack model interacts with the RTSP client and server models as a third party. The composition of the three SysML activity diagrams are automatically translated into PRISM code by using the framework developed in Chapter 3.

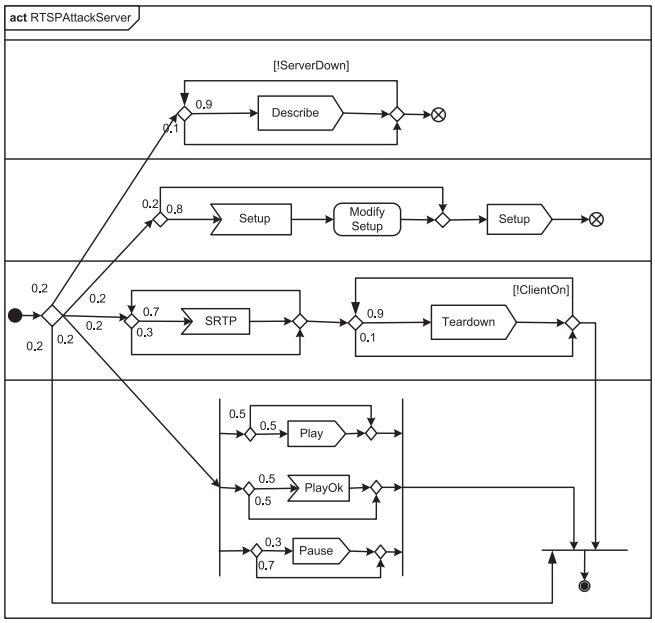
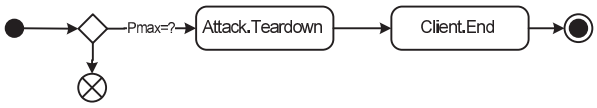


Figure 6.3: SysML activity diagram for RTSP Attack Scenario.

6.4.2 Properties Specification and Evaluation

Here, we propose four security requirements to be verified against the composition of Server/Client/Attack diagrams. Each security requirement is described by a SysML activity diagram then mapped into PCTL expression.

Property 1. "Compute the maximum probability to disconnect a client immediately by a TEARDOWN attack". This requirement is described by the following diagram.



We refer to the labels "Attack.Teardown" send message node from the attack diagram and "Client.End" action node from the client SysML activity diagram to consider them as two atomic propositions. In addition, this requirement asks for an immediate consequence which means Rule 2-a will be applied. Then, the obtained property will be:

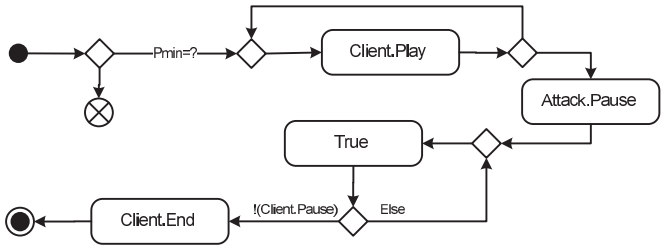
(Attack.Teardown⇒(X (Client.End))).

Furthermore, to measure the maximum probability we rely on Rule 8-d that generates the following PCTL property.

Pmax=?[G(Attack.Teardown⇒(X (Client.End)))].

After the verification of this property, we observe that the client can be interrupted with a probability of 0.82 after a Teardown attack.

Property 2. "Compute the minimum probability of an attacker pausing the media viewed by a client". This requirement is expressed in the following diagrams where "Client.Play", "Attack.Pause", and "Client.Stop" are the atomic propositions.



First, we apply Rule 5-b related to the second merge node we have:

(Client.Play)⇒(F(Attack.Pause)).

Then, we apply Rule 3-b for the second decision node, we get:

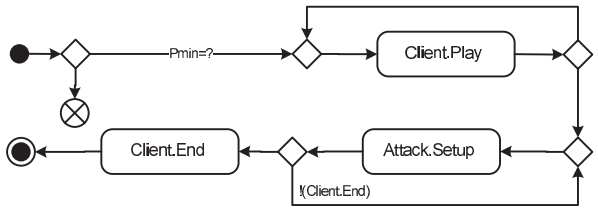
((Client.Play)⇒(F(Attack.Pause)))⇒(F(Client.Stop)).

Finally, by applying Rule 8-g we get the last PCTL expression.

Pmin=? [((Client.Play)⇒(F(Attack.Pause)))⇒(F(Client.Stop))].

After verification, we found that the attacker's probability to disturb a client from viewing media is 0.6.

Property 3. "Measure the minimum probability to successfully hijack a session". This requirement is described by the following SysML activity diagram such that "Client.Play", "Attack.Setup" and "Client.End" are the atomic propositions.



First, by applying Rule 5-b for the third merge node we have:

$$\text{Client.Play} \Rightarrow (\text{F}(\text{Attack.Setup})).$$

Then, we apply Rule 2-c to have:

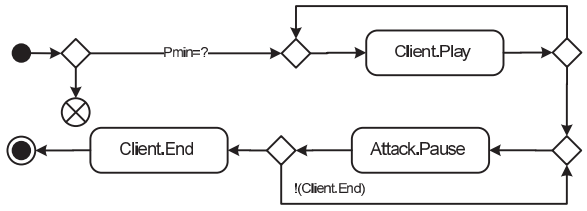
$$(\text{Client.Play} \Rightarrow (\text{F}(\text{Attack.Setup})) \text{ U Client.End}$$

Finally, Rule 8-g is applied to generate the final property:

$$\text{Pmin=? [G((Client.Play} \Rightarrow (\text{F}(\text{Attack.Setup})) \text{ U Client.End)]}.$$

After the evaluation of this property, we conclude that the client can lose its session by a probability value 1.

Property 4. "Find the minimum probability that an attacker enforces a client to pause viewing". The SysML activity diagram of this requirement is shown here.



First, we rely to Rule 5-b for the third merge node which result:

Client.Play⇒(F(Attack.Pause)).

Then, we apply Rule 2-c to have:

Client.Play⇒(F(Attack.Pause)U Client.End).

Finally, Rule 8-a is applied to obtain the last PCTL expression:

Pmin=?[G(Client.Play⇒(F(Attack.Pause)U Client.End))].

The result obtained regarding the verification of the fourth property deduces that a client can be stopped to view the media by a probability of 0.8.

6.5 Related Work

In this section, we cite the existing works related to security specification and attack-based security analysis.

[29] identifies security vulnerabilities in code level by tailoring attack patterns based on the software components. These patterns take the form of regular expressions that are generic representations of vulnerabilities. [33] proposes a risk-based approach that creates modular attack trees for each component in the system. These trees are specified as parametric constraints, which allow quantifying the probability of security breaches that occur due to internal and external component vulnerabilities. [30] advocates the use of the Aspect-Oriented Risk-Driven Development (AORDD) methodology based on formal security evaluation and a trade-off analysis. [12] verifies security requirements by a qualitative analysis and then computes performance measure as a quantitative entity provided by a performance analysis of UML sequence diagram. [77] verifies the security of communication systems designed with UML by using AVISPA[4] and TTool[5] tools. [73] analyzes the

[4]http://www.avispa-project.org
[5]http://labsoc.comelec.enst.fr/turtle/

security of system configurations in terms of the weakest adversary that can compromise the network. [28] models probability metrics based on attack graphs as a special Bayesian Network. Each node of the network represents vulnerabilities as well as the pre and post conditions.

[45, 37] extract specific cryptography-related information from UMLsec diagrams. Moreover, the Dolev-Yao model of an attacker is included with UMLsec to model the interaction with the environment. [14] proposes a collection of security patterns that include security-specific and requirements-oriented information. The behavioral aspects of a pattern are depicted as state machine or sequence diagram, and constraints that must hold are expressed in LTL. [82] extends UMLsec to model peer-to-peer applications along with their security aspects. It relies on the concept of abuse cases defined as UML use cases and state machine diagrams to represent attack scenarios. [86] elaborates security requirements by constructing intentional anti-models. It addresses malicious obstacles set up by attackers to threaten security goals. [84] presents an approach to verify security and time-related requirements. Both, UMLsec and MARTE profiles are used to address security and time.

6.6 Conclusion

In this chapter, a security assessment framework has been proposed to automatically express and evaluate security requirements in systems/software modeled by using SysML activity diagrams. This framework models the standard catalogue of common attack patterns as application-independent SysML activity diagrams templates. These templates are used to easily instantiate an application-dependent attack diagrams. To this end, we have defined the attack/system diagram interaction and map it into PRISM code. In addition, we have developed a specification algorithm that transforms SysML activity diagrams into the probabilistic temporal logic ”PCTL”. The effectiveness of our contribution has been

demonstrated by illustrating how expressing security properties and evaluating them easily on the secure real time streaming protocol. This framework helps to reduce the development cost by allowing flaws detection and measuring security level at an earlier stage of software/system life cycle. The next chapter concludes the thesis by summing up the major contributions and points out open research questions for future work.

Chapter 7

Conclusion

7.1 Conclusion

To tackle the difficulty in the security verification process of model-based systems, we developed a novel formal framework that includes two parts: one for security specification and a second for the formal verification of security. The main purpose of this framework is to improve the verification process of model-based system, so they can meet the essential security requirements.

We described the state of the art in the area of the verification and the security specification of model-based systems using model checking that, to the best of our knowledge, does not have the required infrastructure to deal with security formal verification of composed and interacted SysML behavioral diagrams.

To deal with the formal verification, we defined the adequate formal semantics of SysML activity diagrams that helps to verify them easily by using model checker. Our verification approach maps a set of SysML activity diagrams composed by call behavior and communication actions into the input language of the probabilistic model checker PRISM. To this end, we proved the soundness of our proposed verification approach by defining

adequately the relationship between the semantics of the mapped diagrams and the resulting PRISM models. This is done by formalizing PRISM models. In addition, we proved the preservation of the satisfaction of PCTL properties by this relation.

Moreover, we defined the behavior of attacks as SysML activity diagrams based on a standard catalogue of common attack patterns. These patterns are used to build attack scenarios encompassing probabilistic and non-deterministic behaviors. The result is a library of security templates that facilitate the formal expressiveness of temporal logic. We developed a specification algorithm that generates temporal logic expressions starting from security templates. Then, by using the proposed verification approach, we showed how to evaluate systems' security level based on its design model and a set of attack scenarios.

To minimize the verification cost, we presented a formal abstraction framework to improve the scalability of probabilistic model-checking in general, and more especially for verifying UML and SysML activity diagrams. We proved the soundness of the abstraction algorithm by defining a probabilistic weak simulation relation between the semantics of the abstract and the concrete diagrams. In addition, the preservation of the satisfaction of PCTL properties by this relation is proved.

To avoid the verification complexity of the composed diagrams, we proposed a compositional verification approach to improve the efficiency and the scalability of probabilistic model-checking. The presented solution is based on decomposing a global PCTL property into local ones with respect to interfaces between diagrams. We proved the soundness of the proposed framework by showing that the satisfaction relation of the PCTL properties are preserved. In addition, we demonstrated the effectiveness of our framework by verifying real systems that require a large amount of memory and time processing.

To the best of our knowledge, this thesis proposes an efficient security verification framework of model-based systems, which behavior can be expressed as a probabilistic

system that exhibit non-determinism.

7.2 Future Work

In future, we would like to extend our work by investigating several directions. First, we extend our framework to support more system features, and more diagrams such as state machines and sequence diagrams. This targets to extend the proposed formal semantics to handle the new features. Also, this helps to ensure the equivalence between the different diagrams by finding a common semantics. Another important aspect is extracting the formal syntax of diagrams automatically from the existing standard meta-models. Concerning the specification part of the framework, we intend to handle the reliability requirements of systems, and, prove the completeness of the proposed extension. In addition, expressing the system requirements by using different diagrams. For the abstraction part, we intend to transform a SysML activity diagram to its fractal form in order to benefit from our abstraction framework. This helps in the abstraction/refinement process of UML/SysML diagrams. By adding new system features, we have to explore other abstraction approaches especially data abstraction targeting specific system features like time and objects. Also, we intend to investigate reducing the property within the model. In the compositional part, we target to extend the compositional verification technique for other composition operators such as send and receive messages. And, we plan to extend our framework to handle more compositional verification techniques like assume-guaranty. Finally, we want to validate our framework on more complex system especially the cyber-physical systems.

Bibliography

[1] Marshall D. Abrams. Nims information security threat methodology. Mitre Technical Report MTR 98 W000009, MITRE, Center for Advanced Aviation System Development,McLean, Virgini, August 1998.

[2] L. Alawneh, M. Debbabi, Y. Jarraya, A. Soeanu, and F. Hassayne. A Unified Approach for Verification and Validation of Systems and Software Engineering Models. In *ECBS '06.*, pages 409–418, Washington, DC, USA, 2006. IEEE Computer Society.

[3] 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.

[4] Christel Baier, Pedro R. D'Argenio, and Marcus Größer. Partial order reduction for probabilistic branching time. *Electr. Notes Theor. Comput. Sci.*, 153(2):97–116, 2006.

[5] Christel Baier and Joost Pieter Katoen. *Principles of Model Checking*. The MIT Press, may 2008.

[6] Paolo Baldan, Andrea Corradini, and Fabio Gadducci. Specifying and Verifying UML Activity Diagrams Via Graph Transformation. In Corrado Priami and Paola Quaglia,

editors, *Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 18–33. Springer Berlin / Heidelberg.

[7] Federico Banti, Rosario Pugliese, and Francesco Tiezzi. An accessible verification environment for uml models of services. *Journal of Symbolic Computation*, 46(2):119 – 149, 2011. Automated Specification and Verification of Web Systems.

[8] M. Encarnación Beato, Manuel Barrio-Solórzano, Carlos E. Cuesta, and Pablo de la Fuente. UML Automatic Verification Tool with Formal Methods. *Electron. Notes Theor. Comput. Sci.*, 127:3–16, April 2005.

[9] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification.* Springer, 2001.

[10] 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.

[11] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16:580–595, August 1991.

[12] Mikael Buchholtz, Stephen Gilmore, Valentin Haenel, and Carlo Montangero. Endto-end integrated security and performance analysis on the DEGAS Choreographer platform. In *Proceedings of the International Symposium of Formal Methods Europe (FM 2005), number 3582 in LNCS*, pages 286–301. Springer-Verlag, 2005.

[13] 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.

[14] B H. C. Cheng, S. Konrad, L. A. Campbell, and R. Wassermann. Using Security Patterns to Model and Analyze Security. In *In IEEE W. on Requirements for High Assurance Systems*, pages 13–22, 2003.

[15] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

[16] Chen Daoxi, Zhang Guangquan, and Fan Jianxi. Abstraction framework and complexity of model checking based on the Promela models. In *Computer Science Education, 2009. ICCSE '09. 4th International Conference on*, pages 857 –861, july 2009.

[17] D. Das, R. Kumar, and P.P. Chakrabarti. Timing Verification of UML Activity Diagram Based Code Block Level Models for Real Time Multiprocessor System-on-Chip Applications. In *Software Engineering Conference, 2006. APSEC 2006. 13th Asia Pacific*, pages 199 –208.

[18] Mourad Debbabi, Fawzi Hassaïne, Yosr Jarraya, Andrei Soeanu, and Luay Alawneh. *Verification and Validation in Systems Engineering - Assessing UML / SysML Design Models*. Springer, 2010.

[19] Alastair F. Donaldson, Alice Miller, and David Parker. Language-level symmetry reduction for probabilistic model checking. QEST '09, pages 289–298. IEEE Computer Society, 2009.

[20] Raida Elmansouri, Houda Hamrouche, and Allaoua Chaoui. From UML Activity Diagrams to CSP Expressions: A Graph Transformation Approach using Atom3 Tool. *IJCSI International Journal of Computer Science Issues*, 8, March 2011.

[21] A. Enders and H.D. Rombach. *A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories*. Fraunhofer IESE science series on software engineering. Addison-Wesley Longman, Incorporated, 2003.

[22] Rik Eshuis. Symbolic Model Checking of UML Activity Diagrams. *ACM Transactions on Software Engineering and Methodology*, 15:2006, 2006.

[23] Rik Eshuis and Roel Wieringa. Tool support for verifying uml activity diagrams. *IEEE Transactions on Software Engineering*, 30:2004, 2004.

[24] 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.

[25] 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.

[26] 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, 2011.

[27] 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.

[28] M. Frigault and Lingyu Wang. Measuring Network Security Using Bayesian Network-Based Attack Graphs. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pages 698–703, August 2008.

[29] M Gegick and L Williams. On The Design of More Secure Software-Intensive Systems by Use of Attack Patterns. *Inf. Softw. Technol.*, 49:381–397, April 2007.

[30] Geri Georg, Kyriakos Anastasakis, Behzad Bordbar, Siv Hilde Houmb, Indrakshi Ray, and Manachai Toahchoodee. Verification and Trade-Off Analysis of Security Properties in UML System Models. *IEEE Transactions on Software Engineering*, 36:338–356, 2010.

[31] Karen Mercedes Goertzel, Theodore Winograd, Holly Lynne McKinley, Lyndon Oh Michael Colon, Thomas McGibbon, Elaine Fedchak, and Robert Vienneau. Software Security Assurance, State-of-the-Art Report *SOAR*. State-of-the-Art Report SPO700-98-D-4002, IATAC and DACS, July 2007.

[32] H. Gomaa. *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*. Cambridge University Press, 2011.

[33] L Grunske and D Joyce. Quantitative Risk-Based Security Prediction for Component-Based Systems with Explicitly Modeled Attack Profiles. *J. Syst. Softw.*, 81:1327–1345, August 2008.

[34] C. A. R. Hoare. Communicating sequential processes, 2004.

[35] J. Holt and S. Perry. *SysML for Systems Engineering*. Institution of Engineering and Technology Press, January 2007.

[36] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[37] Siv Hilde Houmb, Shareeful Islam, Eric Knauss, Jan Jürjens, and Kurt Schneider. Eliciting Security Requirements and Tracing them to Design: An Integration of Common Criteria, Heuristics, and UMLsec. *Requir. Eng.*, 15:63–93, March 2010.

[38] H.S.Y. Huang and K.T.G. Cheng. *Formal Equivalence Checking and Design Debugging*. Frontiers in electronic testing; FRET 12. Kluwer Academic, 1998.

[39] ISO. *Information technology – Security techniques – Information security risk management*, 2008.

[40] 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: 355Ű374*, pages 76–91. Springer, 2002.

[41] David N. Jansen, Holger Hermanns, and Joost Pieter Katoen. A QoS-Oriented Extension of UML Statecharts. *Lecture notes in computer science*, 2863:76–91, 2003.

[42] Yosr Jarraya and Mourad Debbabi. Formal specification and probabilistic verification of sysml activity diagrams. In *TASE*, pages 17–24, 2012.

[43] Yosr Jarraya, Mourad Debbabi, and Jamal Bentahar. On the Meaning of SysML Activity Diagrams. In *Proc. of the 2009 16th Ann. IEEE Int. Conf. and Work. on the Eng. of Comp. Based Sys.*, ECBS '09, pages 95–105, Washington, DC, USA, 2009. IEEE Computer Society.

[44] Yosr Jarraya, Andrei Soeanu, Mourad Debbabi, and Fawzi Hassaine. Automatic Verification and Performance Analysis of Time-Constrained SysML Activity Diagrams. In *ECBS '07: Proc. of the 14th An. IEEE Int. Conf. and Work. on the Eng. of Comp.-Bas. Sys.*, pages 515–522, Washington, DC, USA, 2007. IEEE Computer Society.

[45] Jan Jürjens and Pasha Shabalin. Automated Verification of UMLsec Models for Security Requirements. In *UML 2004 Ű The Unified Modeling Language, volume 2460 of LNCS*, pages 412–425. Springer, 2004.

[46] P.S. Kaliappan, H. Koenig, and V.K. Kaliappan. Designing and Verifying Communication Protocols Using Model Driven Architecture and Spin Model Checker. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 2, dec. 2008.

[47] Joost-Pieter Katoen, Tim Kemna, Ivan Zapreev, and David N. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. TACAS'07, pages 87–101, Berlin, Heidelberg, 2007. Springer-Verlag.

[48] Mark Kattenbelt, Marta Kwiatkowska, Gethin Norman, and David Parker. Game-based probabilistic predicate abstraction in prism. *Electron. Notes Theor. Comput. Sci.*, 220(3):5–21, December 2008.

[49] Mark Kattenbelt, Marta Kwiatkowska, Gethin Norman, and David Parker. A game-based abstraction-refinement framework for markov decision processes. *Formal Methods in System Design*, 36:246–280, 2010.

[50] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV*, LNCS, pages 585–591. Springer, 2011.

[51] V. Lima, C. Talhi, D. Mouheb, M. Debbabi, L. Wang, and Makan Pourzandi. Formal verification and validation of uml 2.0 sequence diagrams using source and destination of messages. *Electron. Notes Theor. Comput. Sci.*, 254:143–160, October 2009.

[52] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In *International Conference on Information Security and Cryptology Ű ICISC 2005. LNCS 3935*, pages 186–198. Springer, 2005.

[53] McAfee. *2013 Threats Predictions*. McAfee an Intel Company, April 2013.

[54] Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[55] Monty Newborn. *Automated theorem proving - theory and practice*. Springer, 2001.

[56] Norton. *Internet Security Threat Report*. Symantec Corporation, April 2013.

[57] Iulian Ober, Susanne Graf, and Ileana Ober. Validating Timed UML models by simulation and verification. In *Workshop SVERTS, San Francisco*, October 2003.

[58] Iulian Ober, Susanne Graf, and Ileana Ober. Model Checking of UML Models via a Mapping to Communicating Extended Timed Automata. In *11th International SPIN Workshop on Model Checking of Software, 2004*, volume 2989 of *LNCS*, 2004.

[59] L. OGorman. Comparing Passwords, Tokens, and Biometrics for User Authentication. *Proceedings of the IEEE*, 91(12):2021–2040, 2003.

[60] OMG. *OMG Systems Modeling Language (OMG SysML) Specification*. Object Management Group, September 2007.

[61] OMG. *OMG Unified Modeling Language: Superstructure 2.1.2*. Object Management Group, November 2007.

[62] Samir Ouchani, Yosr Jarraya, and Otmane Aït-Mohamed. Model-based systems security quantification. In *PST*, pages 142–149, 2011.

[63] Samir Ouchani, Yosr Jarraya, Otmane Ait Mohamed, and Mourad Debbabi. Security estimation in Streaming Protocols. In *International Conference on Innovations in Information Technology*, 2011.

[64] Samir Ouchani, Yosr Jarraya, Otmane Aït Mohamed, and Mourad Debbabi. Probabilistic attack scenarios to evaluate policies over communication protocols. *JSW*, 7(7):1488–1495, 2012.

[65] Samir Ouchani, Otmane Aït Mohamed, and Mourad Debbabi. A Formal Verification Framework for SysML Activity Diagrams. *Software and System Modeling (SoSyM)*.

[66] Samir Ouchani, Otmane Aït Mohamed, and Mourad Debbabi. A Property-Based Abstraction Framework for SysML Activity Diagrams. *Journal of Systems and Software (JSS)*.

[67] Samir Ouchani, Otmane Aït Mohamed, and Mourad Debbabi. Efficient probabilistic abstraction for sysml activity diagrams. In *SEFM*, pages 263–277, 2012.

[68] Samir Ouchani, Otmane Aït Mohamed, and Mourad Debbabi. A probabilistic verification framework for sysml activity diagrams. In *SoMeT*, pages 108–123, 2012.

[69] Samir Ouchani, Otmane Aït Mohamed, and Mourad Debbabi. A compositional verification framework for sysml activity diagrams. In *ICFEM*, page Submitted, 2013.

[70] Samir Ouchani, Otmane Aït Mohamed, and Mourad Debbabi. A probabilistic model checking framework for sysml activity diagrams. In *SoMeT*, page To appear, 2013.

[71] Samir Ouchani, Otmane Aït Mohamed, and Mourad Debbabi. A security risk assessment framework for sysml activity diagrams. In *SERE*, page To appear, 2013.

[72] Samir Ouchani, Otmane Ait Mohamed, Mourad Debbabi, and Makan Pourzandi. Verification of the Correctness in Composed UML Behavioural Diagrams. In *SERA (selected papers)*, pages 163–177, 2010.

[73] Joseph Pamula, Sushil Jajodia, Paul Ammann, and Vipin Swarup. A Weakest-Adversary Security Metric For Network Configuration Security Analysis. In *the Proceedings of the 2nd ACM workshop on Quality of protection*, QoP'06, pages 31–38, New York, NY, USA, 2006. ACM.

[74] Chikmagalur Manjappa Prashanth and K. Chandrashekhar Shet. Efficient Algorithms for Verification of UML Statechart Models. *Journal of Software*, 4:175–182, 2009.

[75] V. Rafe, R. Rafeh, S. Azizi, and M.R.Z. Miralvand. Verification and Validation of Activity Diagrams Using Graph Transformation. In *Computer Technology and Development, 2009. ICCTD '09. International Conference on*, volume 1, pages 201 –205, nov. 2009.

[76] Pritam Roy, David Parker, Gethin Norman, and Luca de Alfaro. Symbolic magnifying lens abstraction in markov decision processes. QEST '08, pages 103–112, Washington, DC, USA, 2008. IEEE Computer Society.

[77] P. Saqui-Sannes, T. Villemur, B. Fontan, S. Mota, M. S. Bouassida, N. Chridi, I. Chrisment, and L. Vigneron. Formal Verification of Secure Group Communication Protocols Modelled in UML. *Innovations in Systems and Software Engineering*, 6:125–133, 2010.

[78] R. Sawilla and Defence R&D Canada Ottawa. *Googling attack graphs*. Technical memorandum. Defence R&D Canada - Ottawa, 2007.

[79] Roberto Segala. A compositional trace-based semantics for probabilistic automata. In Insup Lee and Scott Smolka, editors, *CONCUR '95: Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 234–248. Springer Berlin / Heidelberg, 1995.

[80] Oleg Mikhail Sheyner. *Scenario graphs and attack graphs*. PhD thesis, Pittsburgh, PA, USA, 2004. AAI3126929.

[81] Igor Siveroni, Andrea Zisman, and George Spanoudakis. Property Specification and Static Verification of UML Models. In *Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*, pages 96–103, 2008.

[82] Igor Siveroni, Andrea Zisman, and George Spanoudakis. A UML-Based Static Verification Framework for Security. *Requirements Engineering*, 15:95–118, 2010.

[83] Maurice H. ter Beek, Alessandro Fantechi, Stefania Gnesi, and Franco Mazzanti. A State/Event-Based Model-Checking Approach for the Analysis of Abstract System Properties. *Sci. Comput. Program.*, 76:119–135, February 2011.

[84] V. Thapa, Eunjee Song, and Hanil Kim. An Approach to Verifying Security and Timing Properties in UML Models. In *15th IEEE Int. Conf. on ICECCS*, 2010.

[85] M.F. van Amstel, C.F.J. Lange, and M.R.V. Chaudron. Four Automated Approaches to Analyze the Quality of UML Sequence Diagrams. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, volume 2, pages 415 –424, july 2007.

[86] A. van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-Models. In *26th ICSE 2004.*, pages 148 – 157.

[87] Bernd Westphal. LSC Verification for UML Models with Unbounded Creation and Destruction. *Electronic Notes in Theoretical Computer Science*, 144:133–145, 2006.

[88] Fei Xie and James C. Browne. Integrated State Space Reduction for Model Checking Executable Object-oriented Software System Designs. In *Proc. of FASE 2002*. Springer-Verlag, 2002.

[89] Fei Xie, Vladimir Levin, and James C. Browne. ObjectCheck: A Model Checking Tool for Executable Object-Oriented Software System Designs. In *Fundamental Approaches to Software Engineering*, pages 331–335, 2002.

[90] Zhu Xin-feng, Wang Jian-dong, Zhu Xin-feng, Li Bin, Zhu Jun-wu, and Wu Jun. Methods to tackle state explosion problem in model checking. In *Proceedings of the 3rd int. conf. on IITA*, pages 329–331, NJ, USA, 2009. IEEE Press.

[91] Hongji Yang. *Software Evolution With Uml And Xml*, chapter Abstracting UML Behavior Diagrams for Verification, pages 296–320. IGI Publishing, Hershey, PA, USA, 2005.

Appendix A

Chapter 3

In this Appendix, we present proofs of the previous theorems and propositions.

Lemma A.1 (Soundness)**.** *The mapping algorithm* Γ *is sound, i.e.* $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}} M_{\mathscr{P}}$.

Proof. To prove $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}} M_{\mathscr{P}}$, we follow a structural induction reasoning on NuAC terms and their related PRISM terms. For that, let $s, s' \in S(M_{\mathscr{A}})$ and $t, t' \in S(M_{\mathscr{P}})$. We distinguish the following cases where $L(s)$ takes different values:

1. $L(s) = l : \overline{x} \rightarrowtail \mathscr{N}$ such as $x \in \{\iota, a, a!v\} \Rightarrow \exists s \xrightarrow{l}_1 s'$, $L(s') = l : x \rightarrowtail \overline{\mathscr{N}}$.
   For $L(t) = \Gamma(L(s))$, we have $L(t) = \langle L(x), \neg L(\mathscr{N}) \rangle$ then $\exists t \xrightarrow{l}_1 t'$ where $L(t') = \langle \neg L(x), L(\mathscr{N}) \rangle$.

2. $L(s) = l : \overline{\odot}$ then $\exists s \xrightarrow{l}_1 s'$ such as $L(s') = |\mathscr{A}|$. For $L(t) = \Gamma(L(s))$, we have $L(t) = \langle l \rangle$ then $\exists t \xrightarrow{l}_1 t'$ where $\forall l_i \in \mathscr{L} : L(t') = \langle \cdots, \neg l_i, \cdots \rangle$.

3. $L(s) = l : \overline{\otimes}$ then $\exists s \xrightarrow{l}_1 s'$ such as $L(s') = l : \otimes$. For $L(t) = \Gamma(L(s))$, we have $L(t) = \langle l \rangle$ then $\exists t \xrightarrow{l}_1 t'$ where $L(t') = \langle \neg l \rangle$.

4. $L(s) = l : \overline{F(\mathscr{N}_1, \mathscr{N}_2)}^m$ then $\exists s \xrightarrow{l}_1 s'$ such as $L(s') = l : \overline{F(\overline{\mathscr{N}_1}, \overline{\mathscr{N}_2})}^{m-1}$. For $L(t) = \Gamma(L(s))$, we have $L(t) = \langle l, \neg l_{\mathscr{N}_1}, \neg l_{\mathscr{N}_2} \rangle$ then $\exists t \xrightarrow{l}_1 t'$ where $\forall l : L(t') = \langle \neg l, l_{\mathscr{N}_1}, l_{\mathscr{N}_2} \rangle$.

From the obtained results, we found that $\mu(s) = \mu(t) = 1$ then $s \sqsubseteq_{\mathscr{R}} t$. In addition, the unique initial state of $\mathscr{M}_{\mathscr{A}}$ is always corresponding to the unique initial state in $\mathscr{M}_{\mathscr{P}}$. By following the same style of proof, we find that $\mathscr{M}_{\mathscr{A}} \sqsubseteq_{\mathscr{R}} \mathscr{M}_{\mathscr{P}}$, which confirms that Lemma 3.1 holds.

□

Proposition A.1 (PCTL Preservation)**.** *For two PAs $M_{\mathscr{A}}$ and $M_{\mathscr{P}}$ such that $\Gamma(\mathscr{A}) = \mathscr{P}$ where $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}} M_{\mathscr{P}}$. For a PCTL property ϕ, then:* $(M_{\mathscr{A}} \models \phi) \Leftrightarrow (M_{\mathscr{P}} \models \phi)$.

Proof. To prove the preservation of PCTL properties, we follow an inductive reasoning on the PCTL structure. While $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}} M_{\mathscr{P}}$, for each PCTL operator $\zeta \in \{\neg, \wedge, \mathrm{X}, \mathrm{U}^{\leq k}, \mathrm{U}, \mathrm{P}_{\bowtie p}\}$ we have: $M_{\mathscr{A}} \models \zeta \Longleftrightarrow M_{\mathscr{P}} \models \zeta$ which means:

$(M_{\mathscr{A}} \models \phi_{PCTL}) \Longleftrightarrow (M_{\mathscr{P}} \models \phi_{PCTL})$.

□

Appendix B

Chapter 4

B.1 Motivation Proof

Let $\mathscr{A}$ be a SysML activity diagram and $\widehat{\mathscr{A}}$ is its abstracted model obtained by the algorithm δ with respect to a PCTL property ϕ. The complexity of checking ϕ takes into consideration the complexity of the abstraction algorithm δ, the complexity of the mapping algorithm of the obtained diagram into PRISM code and the complexity of the probabilistic model checking procedure. The both first algorithms are a DFS-based procedures with a time complexity of $\mathscr{O}(|\mathscr{A}|)$, the third one is of $\mathscr{O}(Poly(|M_{\mathscr{A}}|) \times \upsilon_{max} \times |\phi|)$ such that $\upsilon_{max} = max\{k : \phi_1 \mathrm{U}^{\leq k} \phi_2 \text{ occurs in } \phi\}$ and $M_{\mathscr{A}} \equiv \mathscr{S}(\mathscr{A})$. Hence, Equation B.1 and Equation B.2 present the complexity with and without abstraction, respectively.

$$A = \mathscr{O}(\mathscr{A} \models p) = \mathscr{O}(|\mathscr{A}|) + O(Poly(|M_{\mathscr{A}}|) \times \upsilon_{max} \times |\phi|) \tag{B.1}$$

$$B = \mathscr{O}(\mathscr{A} \models p) = \mathscr{O}(|\mathscr{A}|) + \mathscr{O}(|\widehat{\mathscr{A}}|) + \mathscr{O}(Poly(|M_{\widehat{\mathscr{A}}}|) \times \upsilon_{max} \times |\phi|) \tag{B.2}$$

By comparing the equations B.1 and B.2, we have:

$$
\begin{aligned}
B < A \quad &\Leftrightarrow \quad |\mathscr{A}| + |\widehat{\mathscr{A}}| + Poly(|M_{\widehat{\mathscr{A}}}|) \times \upsilon_{max} \times |\phi| < |\mathscr{A}| + Poly(|M_{\mathscr{A}}|) \times \upsilon_{max} \times |\phi| \\
&\Leftrightarrow \quad |\widehat{\mathscr{A}}| + Poly(|M_{\widehat{\mathscr{A}}}|) \times \upsilon_{max} \times |\phi| < Poly(|M_{\mathscr{A}}|) \times \upsilon_{max} \times |\phi| \\
&\Leftrightarrow \quad \frac{|\widehat{\mathscr{A}}|}{Poly(|M_{\mathscr{A}}|) \times \upsilon_{max} \times |\phi|} + \frac{Poly(|M_{\widehat{\mathscr{A}}}|)}{Poly(|M_{\mathscr{A}}|)} < 1 \\
&\Leftrightarrow \quad \frac{Poly(|M_{\widehat{\mathscr{A}}}|)}{Poly(|M_{\mathscr{A}}|)} \lesssim 1 (while \frac{|\widehat{\mathscr{A}}|}{Poly(|M_{\mathscr{A}}|) \times \upsilon_{max} \times |\phi|} \simeq 0)
\end{aligned}
$$

From this result of comparison, we conclude that applying the abstraction is useful for any model. □

B.2 The Abstraction Approach Proof

Proposition B.1. *Let $\mathscr{A} = \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k$ be a SysML activity diagram with k call behaviors and ϕ be a PCTL property. Then:*

$$\forall i \leq k : \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_i} \mathscr{A}_i \models \phi \;\Rightarrow\; \mathscr{A} \models \phi.$$

Proof. The proof of this proposition follows an induction reasoning on PCTL structure by taking into consideration the size of call behaviors composition "k".

1. Case of $\phi = \phi_1 \mathrm{U} \phi_2$

   Let $i < k$: $\mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_i} \mathscr{A}_i \models \phi$ (Assumption)

   $\Leftrightarrow \exists m, \forall j < m : \pi(j) \models_{Adv} \phi_1 \wedge \pi(m) \models_{Adv} \phi_2.$ (Definition)

   By calling a new behavior $\mathscr{A}_{i+1}$, the satisfaction path π will not be changed while a_{i+1} stays unchanged during the execution of $\mathscr{A}_{i+1}$ then we have:

   $\exists m', \forall j' < m' : \pi(j') \models_{Adv} \phi_1 \wedge \pi(m') \models_{Adv} \phi_2 \Leftrightarrow \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_{i+1}} \mathscr{A}_{i+1} \models \phi.$

   By following the same kind of construction up to k behavior, we will have:

$\mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k \models \phi$ which means:

$$\exists m'',\ \forall j'' < m'' : \pi(j'') \models_{Adv} \phi_1 \wedge \pi(m'') \models_{Adv} \phi_2.$$ Then: $\mathscr{A} \models \phi_1 \mathrm{U} \phi_2$.

By following the same kind of proof, we deduce the satisfaction relation for $\phi_1 \mathrm{U}^{\leq k} \phi_2$ and $\mathrm{X}\phi$ cases.

2. Case of $\phi = \phi_1 \mathrm{U}^{\leq l} \phi_2$

   When $j'' = j$ and $m'' = m$ after calling a new behavior then $\mathscr{A} \models \phi_1 \mathrm{U}^{\leq l} \phi_2$. Else, $\mathscr{A} \models \phi_1 \mathrm{U}^{\leq l'} \phi_2$ such that $l' \geq l$

3. Case of $\mathrm{X}\phi$

   While a_{i+1} stays true during the execution of $\mathscr{A}_{i+1}$ then the next operator is preserved.

□

Proposition B.2. *Let $\mathscr{A} = \mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k$ be a SysML activity diagram with k call behaviors, $\mathscr{A}_{id}$ is the identity element for "$\uparrow$" operator and ϕ be a PCTL property. For a proposition α, we have the following:*

$$\forall 1 \leq i \leq k,\ \alpha \notin (\Sigma_\phi \cap \Sigma_{\mathscr{A}_i}) : [\mathscr{A}_i = \mathscr{A}_{id} \wedge (\mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k) \models \phi] \Rightarrow [\mathscr{A} \models \phi].$$

Proof. The proof is similar to that one of Proposition 4.1, for both cases of until and next operators. For the case of bounded until, the satisfaction of ϕ is not deduced while some steps are mimicked by the identity element. But, we can infer the satisfaction of two important properties are:

1. $\forall 1 \leq i \leq k,\ \alpha \notin (\Sigma_\phi \cap \Sigma_{\mathscr{A}_i})$:

   $$[\mathscr{A}_i = \mathscr{A}_{id} \wedge (\mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k) \models \phi_1 \mathrm{U}^{\leq l} \phi_2] \Rightarrow \mathscr{A} \models \phi_1 \mathrm{U} \phi_2.$$

2. $\forall 1 \leq i \leq k,\ \alpha \notin (\Sigma_\phi \cap \Sigma_{\mathscr{A}_i})$:

   $$[\mathscr{A}_i = \mathscr{A}_{id} \wedge (\mathscr{A}_0 \uparrow_{a_1} \ldots \uparrow_{a_k} \mathscr{A}_k) \models \phi_1 \mathrm{U}^{\leq l} \phi_2] \Rightarrow \exists l' \geq l : \mathscr{A} \models \phi_1 \mathrm{U}^{\leq l'} \phi_2.$$

□

B.3 Complexity

Proposition B.3 (Application Order)**.** *Let "$\mathscr{A}$" be a NuAC term and "ϕ" be a PCTL property, we have:* $\Psi(\Upsilon(\Psi(\mathscr{A}),\phi)) \equiv \Psi(\Upsilon(\mathscr{A},\phi))$.

Proof. Let $M_1 = \Psi(\mathscr{A}), M_2 = \Upsilon(M_1,\phi)$ and $M_3 = \Psi(M_2)$, we have:

1. $M_1 = \Psi(\mathscr{A}) \Leftrightarrow$ if $\exists l : \mathscr{N}_k \rightarrowtail \mathscr{N}_m \in \mathscr{A}$, then $l : \mathscr{N}_k \rightarrowtail \mathscr{N}_m$ is replaced by $l : \mathscr{N}_{km}$ if one of the control merging rules is satisfied.

2. $M_2 = \Upsilon(M_1,\phi) \Leftrightarrow \forall a \notin \Sigma_\phi : \Upsilon(\overline{l : a}^n \rightarrowtail \mathscr{N},\phi) = \overline{l : \varepsilon}^n \rightarrowtail \mathscr{N}$. In fact, Υ produces new consecutive control nodes and preserves the diagram structure.

3. $M_3 = \Psi(M_2) \Leftrightarrow$ if $\exists l : \mathscr{N}_i \rightarrowtail \mathscr{N}_j \in \mathscr{A}$, then $l : \mathscr{N}_i \rightarrowtail \mathscr{N}_j$ is replaced by $l : \mathscr{N}_{ij}$. The minimization rules are applied on the initial and the produced one by Ψ function.

It is clear that the first step has no effect on the second one. In addition, applying Ψ two times successively is equivalent to applying it once. Thus, the proposition holds. □

B.4 Abstraction Soundness Proof

Lemma B.1 (Υ-Soundness)**.** *The abstraction algorithm Υ is sound, i.e.* $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_\Upsilon} M_{\widehat{\mathscr{A}}}$.

Proof. To prove $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_\Upsilon} M_{\widehat{\mathscr{A}}}$, we follow a structural induction reasoning on NuAC terms such that $\widehat{\mathscr{A}} = \Upsilon(\mathscr{A})$. For that, let $s,s' \in S(M_{\mathscr{A}})$[1] and $t,t' \in S(M_{\widehat{\mathscr{A}}})$. We distinguish the following cases where $L(s)$ takes different values:

- Case $a \rightarrowtail \mathscr{N}$:

  Let $L(s) = \overline{a} \rightarrowtail \mathscr{N} \Rightarrow \exists\, s' : s \rightarrow\ s'$ by applying ACT-2 such that:

  $L(s') = a \rightarrowtail \overline{\mathscr{N}} \Rightarrow \mu_s(s') = 1$.

[1] $S(M)$ is the set of states of M.

By considering t as the abstracted state of s where $L(t) = \Upsilon(L(s))$, we can distinguish two cases for ABS-1 and ABS-2 that are presented respectively as follows:

1. $a \in \Sigma_\phi : L(t) = \Upsilon(\overline{a} \rightarrowtail \mathscr{N}) = \overline{a} \rightarrowtail \Upsilon(\mathscr{N})$. By applying ACT-2, $\exists\, t' : t \rightarrow t'$ such that: $L(t') = a \rightarrowtail \overline{\Upsilon(\mathscr{N})} \Rightarrow \mu_t(t') = 1$. Then, it exists a weight function $\triangle$ for $\mathscr{R}_\Upsilon = \{(s',t')\}$ such that:
   $\triangle(s',t') = 1 \Rightarrow \mu_s(s') = \triangle(s',t')$ and $\triangle(t',s') = 1 \Rightarrow \mu_t(t') = \triangle(t',s')$.
   While $\triangle(s,t) > 0$ then $s \sqsubseteq_{\mathscr{R}_\Upsilon} t$.
2. $a \notin \Sigma_\phi : L(t) = \Upsilon(s) = \Upsilon(\overline{a} \rightarrowtail \mathscr{N}) = \overline{\varepsilon} \rightarrowtail \Upsilon(\mathscr{N}) \Rightarrow \exists\, t' :\ t \rightarrow\ t'$.
   By applying ACT-2 rule such that $L(t') = \varepsilon \rightarrowtail \overline{\mathscr{N}} \Rightarrow \mu_2(t') = 1$. Then it exists a weight function $\triangle$ for $\mathscr{R}_\Upsilon = \{(s',t')\}$ such that:
   $\triangle(s',t') = 1 \Rightarrow \triangle(s',t') = \mu_s(s')$ and $\triangle(t',s') = 1 \Rightarrow \mu_t(t') = \triangle(t',s')$, with $\triangle(s,t) > 0$ then $s \sqsubseteq_{\mathscr{R}_\Upsilon} t$.

- Case $\mathscr{N}_1 \overset{g}{\mapsto} \mathscr{N}_2$ is similar to the case of $\overline{a} \rightarrowtail \mathscr{N}$ after applying ABS-3 and ABS-4.

It is clear that, the marked NuAC term $\mathscr{A}$ is the unique initial state of $M_{\mathscr{A}}$ corresponding to the unique initial state in $M_{\widehat{\mathscr{A}}}$. By following the same style of proof, we find:

$M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_\Upsilon} M_{\widehat{\mathscr{A}}}$, which confirms that Proposition 4.1 holds. □

Proposition B.4 (Υ-Preservation)**.** *For two PAs $M_{\mathscr{A}}$ and $M_{\widehat{\mathscr{A}}}$ such that $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_\Upsilon} M_{\widehat{\mathscr{A}}}$. If ϕ is a PCTL$_{\setminus X}$ property, then we have:* $(M_{\widehat{\mathscr{A}}} \models \phi) \Rightarrow (M_{\mathscr{A}} \models \phi)$.

Proof. To prove the preservation of PCTL properties by $\mathscr{R}_\Upsilon$, we follow an inductive reasoning on PCTL structure after applying each Υ-abstraction rule.

Let $\pi \in M$ with $\pi = (s_0 \cdots s_i \cdots s_j \cdots s_n)$ and $\pi' \in \widehat{M}$ with $\pi' = (t_0 \cdots t_k \cdots t_l \cdots t_m)$ are two finite paths such that; $\forall s \in \pi : \exists s' \in \pi'$, $s \sqsubseteq_{\mathscr{R}_\Upsilon} s'$. For PCTL expression satisfaction, we distinguish these cases:

- Case $\phi_1 \mathrm{U} \phi_2$:

  For π and π' such that: $\pi' \models \phi_1 \mathrm{U} \phi_2 \Leftrightarrow \exists r' \leq m : \forall u \leq r'-1,\ L(s'_u) = \phi_1$ and $L(s'_r) = \phi_2$. Similarly, $\pi \models \phi_1 \mathrm{U} \phi_2 \Leftrightarrow \exists r \leq n : \forall w \leq r-1,\ L(s_w) = \phi_1$ and $L(s_r) = \phi_2$. By applying the ABS rules, the states that do not satisfy either ϕ_1 and ϕ_2 in π are mimicked.

  Consequently, $\pi' \models \phi_1 \mathrm{U} \phi_2 \Rightarrow \pi \models (\phi_1 \Rightarrow F \phi_2)$.

- Case $\mathrm{P}_{\bowtie p}[\phi_1 \mathrm{U} \phi_2]$:

  We have $\pi' \models \phi_1 \mathrm{U} \phi_2 \Rightarrow \pi \models (\phi_1 \Rightarrow F \phi_2)$ and the mimicked states with the ABS rules have a Dirac distribution, then: $\pi' \models P_{\bowtie p}[\phi_1 \mathrm{U} \phi_2] \Rightarrow \pi \models P_{\bowtie p}[\phi_1 \Rightarrow F \phi_2]$.

By following the same proof style on PCTL structure, we conclude that:

$$(M_{\widehat{\mathscr{A}}} \models P_{\bowtie p}[\phi]) \Rightarrow\ (M_{\mathscr{A}} \models P_{\bowtie p}[\phi]).$$

□

Lemma B.2 (Ψ-Soundness)**.** *The abstraction algorithm* Ψ *is sound, i.e.* $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_\Psi} M_{\widehat{\mathscr{A}}}$.

Proof. To prove $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_\Psi} M_{\widehat{\mathscr{A}}}$, we follow a structural induction reasoning on NuAC terms by comparing each term before and after applying Ψ rules. Let $s_0, s_1, s_2, s_3, s_4 \in S(M_{\mathscr{A}})$ and $t_0, t_1, t_2, t_3 \in S(M_{\widehat{\mathscr{A}}})$. We distinguish the following cases where $L(s_0)$ takes different values:

- **Case** $D(p, g, \mathscr{N}, \mathscr{N}')$: Let $L(s_0) = \overline{D(p_1, g_1, \mathscr{N}_1, \mathscr{N}_1')}$, by applying PDEC-1 rule, we will have: $s_0 \rightarrow\ \mu_0(s_1, s_2)$ such that:

  - $L(s_1) = D(p_1, g_1, \overline{\mathscr{N}_1}, \mathscr{N}_1')$ such as $\mu_0(s_1) = p_1$,
  - $L(s_2) = D(p_1, g_1, \mathscr{N}_1, \overline{\mathscr{N}_1'})$ such as $\mu_0(s_2) = 1 - p_1$.

  By considering $\mathscr{N}_1 = D(p_2, g_2, \mathscr{N}_2, \mathscr{N}_2')$, let t_0 be the merged state of s_0 with s_1 by applying the MIN-2 rule, we will have:

$L(t_0) = D(p_1 \times p_2, g_1 \wedge g_2, \mathscr{N}_2, \mathscr{N}_2^{'}, 1-p_1, \mathscr{N}'_1) \Rightarrow \exists t_1,t_2,t_3 : t_0 \rightarrow \mu'(t_1,t_2,t_3)$. By applying PDEC-1 rule, we have:

- $L(t_1) = D(p_1 \times p_2, g_1 \wedge g_2, \overline{\mathscr{N}}_2, \mathscr{N}_2^{'}, 1-p_1, \neg g_1, \mathscr{N}'_1) \Rightarrow \mu'(t_1) = p_1 \times p_2$.
- $L(t_2) = D(p_1 \times p_2, g_1 \wedge g_2, \mathscr{N}_2, \overline{\mathscr{N}_2^{'}}, 1-p_1, \neg g_1, \mathscr{N}'_1) \Rightarrow \mu'(t_2) = p_1 \times (1-p_2)$.
- $L(t_3) = D(p_1 \times p_2, g_1 \wedge g_2, \mathscr{N}_2, \mathscr{N}_2^{'}, 1-p_1, \neg g_1, \overline{\mathscr{N}_1^{'}}) \Rightarrow \mu'(t_3) = 1-p_1$.

It exists a weight function $\triangle$ for $\mathscr{R}_\Psi = \{(s_0,t_0),(s_3,t_1),(s_4,t_2),(s_2,t_3)\}$ such that:

1. $\triangle(s_3,t_1) = p_1 \Rightarrow \triangle(s_3,t_1) = \mu'(t_1)$
2. $\triangle(s_4,t_2) = p_1 \times (1-p_2) \Rightarrow \triangle(s_4,t_2) = \mu'(t_2)$
3. $\triangle(s_2,t_3) = 1-p_1 \Rightarrow \triangle(s_2,t_3) = \mu'(t_3)$

We have: $(\triangle(s_3,t_1) > 0 \wedge \triangle(s_2,t_3) > 0) \wedge \triangle(s_4,t_2) > 0) \Rightarrow \ \mu \sqsubseteq_{\mathscr{R}_\Psi} \ \mu'$.

It is clear that, the marked NuAC term $\mathscr{A}$ is the unique initial state of $M_{\mathscr{A}}$ corresponding to the unique initial state in $M_{\widehat{\mathscr{A}}}$. By following the same style of proof, we find:

$M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_\Psi} M_{\widehat{\mathscr{A}}}$, which confirms that Proposition 4.1 holds. □

Proposition B.5 (Ψ-Preservation)**.** *For two PAs $M_{\mathscr{A}}$ and $M_{\widehat{\mathscr{A}}}$ such that $M_{\mathscr{A}} \sqsubseteq_{\mathscr{R}_\Psi} M_{\widehat{\mathscr{A}}}$. If ϕ is a PCTL property, then we have: $(M_{\widehat{\mathscr{A}}} \models \phi) \Rightarrow (M_{\mathscr{A}} \models \phi)$.*

Proof. To prove the preservation of PCTL properties by $\mathscr{R}_\Psi$, we follow an inductive reasoning on PCTL structure for each Υ-abstraction rule.

Let $\pi \in M$ with $\pi = (s_0 \cdots s_i \cdots s_j \cdots s_n)$ and $\pi' \in \widehat{M}$ with $\pi' = (t_0 \cdots t_k \cdots t_l \cdots t_m)$ are two finite paths such that; $\forall s \in \pi : \exists s' \in \pi'$, $s \sqsubseteq_{\mathscr{R}_\Psi} s'$. For PCTL expression satisfaction, we distinguish these cases:

- Case $\phi_1 \mathrm{U} \phi_2$:

  For π and π' such that: $\pi' \models \phi_1 \mathrm{U} \phi_2 \Leftrightarrow \exists r' \leq m : \forall u \leq r'-1$, $L(s'_u) = \phi_1$ and $L(s'_r) =$

ϕ_2. Similarly, $\pi \models \phi_1 \mathrm{U} \phi_2 \Leftrightarrow \exists r \leq n : \forall w \leq r-1,\ L(s_w) = \phi_1$ and $L(s_r) = \phi_2$.

By applying MIN rules, some states in π are merged.

Consequently, $\pi' \models \phi_1 \mathrm{U} \phi_2 \Rightarrow \pi \models \phi_1 \Rightarrow F\phi_2$.

- Case $\mathrm{P}_{\bowtie p}[\phi_1 \mathrm{U} \phi_2]$:

  We have $\pi' \models \phi_1 \mathrm{U} \phi_2 \Rightarrow \pi \models \phi_1 \Rightarrow F\phi_2$ and the merged states with MIN rules accumulate the probabilistic distribution, then: $\pi' \models P_{\bowtie p}[\phi_1 \mathrm{U} \phi_2] \Rightarrow \pi \models P_{\bowtie p}[\phi_1 \Rightarrow F\phi_2]$.

By following the same proof style on PCTL structure, we conclude that:

$$(M_{\widehat{\mathscr{A}}} \models \phi) \Rightarrow (M_{\mathscr{A}} \models \phi).$$

□