Modeling and Analysis of Stochastic Real-Time Systems

Braham Lotfi Mediouni

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTE UNIVERSITE GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Braham Lotfi Mediouni

Thèse dirigée par **Saddek Bensalem**

préparée au sein du **Laboratoire Verimag**
dans **l'École Doctorale Mathématique, Sciences et Technologies de l'information (MSTII)**

Modeling and Analysis of Stochastic Real-Time Systems

Thèse soutenue publiquement le **28/06/2019**,
devant le jury composé de :

Erika Abraham
Professeur, Université RWTH Aachen, Rapporteur
Enrico Vicario
Professeur, Université de Florence, Rapporteur
Eugene Asarin
Professeur, Université Paris-Diderot, Président
Kim G. Larsen
Professeur, Université d'Aalborg, Membre
Marius-Dorel Bozga
Ingénieur de recherche, CNRS, Membre
Saddek Bensalem
Professeur, Université Grenoble Alpes, Directeur de thèse

Modeling and Analysis of Stochastic Real-Time Systems

Braham Lotfi Mediouni

Laboratoire Verimag

Université Grenoble Alpes

This dissertation is submitted for the degree of

Doctor of Philosophy

Ecole Doctorale Mathématique,
Sciences et Technologies de
l'Information, Informatique (MSTII)

June 2019

Abstract

In this thesis, we address the problem of modeling and verification of complex systems exhibiting both probabilistic and timed behaviors. Designing such systems has become increasingly complex due to the heterogeneity of the involved components, the uncertainty resulting from open environment and the real-time constraints inherent to their application domains. Handling both software and (abstraction of) hardware in a unified view while also including performance information (e.g. computation and communication times, energy consumption, etc.) becomes a must. Building and analyzing performance models is of paramount importance in order to give guarantees on the functional and extra-functional system requirements and to make well-founded design decisions based on quantitative measures at early design stages.

This thesis brings several new contributions. First, we introduce a new modeling formalism called Stochastic Real-Time BIP (SRT-BIP) for the modeling, the simulation and the code generation of component-based systems. This formalism inherits from the BIP framework its component-based and real-time modeling capabilities and, extends it by providing comprehensive primitives to express complex stochastic behaviors.

Second, we investigate machine learning techniques to ease the construction of performance models. We propose to enhance and adapt a state-of-the-art learning procedure to infer stochastic real-time models from concrete system execution and to represent them in the SRT-BIP formalism.

Third, given performance models in SRT-BIP, we explore the use of statistical Model Checking (SMC) for the anaysis of system's functional and performance requirements. To do so, we provide a full framework, called $\mathcal{S}$BIP, as a support tool for the modeling, simulation and analysis of SRT-BIP systems. $\mathcal{S}$BIP is an Integrated Development Environment (IDE) that implements SMC algorithms for quantitative, qualitative and rare events analyses together with an automated exploring procedure for parameterized requirements. We validate our proposals on real-life case studies ranging from communication protocols and concurrent systems to embedded systems.

Finally, we further investigate the interest of SMC when included in elaborated system analysis workflows. We illustrate this by proposing two risk assessment approaches. In the first approach, we introduce a spiral methodology to build resilient systems with FDIR components that we validate on the safety assessment of a planetary rover locomotion system. The second approach is concerned with the security assessment of organization's defenses following an

offensive security approach. The goal is to synthesize impactful defense configurations against optimized attack strategies (that minimize attack cost and maximize success probability). These attack strategies are obtained by combining model learning with meta-heuristics, and where SMC is used to score and prioritize potential candidate strategies.

Résumé

Dans cette thèse, nous abordons le problème de la modélisation et de la vérification de systèmes complexes présentant des comportements à la fois probabilistes et temporisés. La conception de tels systèmes est devenue de plus en plus complexe en raison de l'hétérogénéité des composants impliqués, l'incertitude découlant d'un environnement ouvert et les contraintes temps réel inhérentes à leurs domaines d'application. La gestion à la fois du logiciel et du matériel dans une vue unifiée tout en incluant des informations sur les performances (par exemple, temps de calcul et de communication, consommation d'énergie, etc.) devient indispensable. Construire et analyser des modèles de performance est d'une importance primordiale pour donner des garanties sur les exigences fonctionnelles et extra-fonctionnelles des systèmes, et permettre une prise de décision fondée sur des mesures quantitatives dès les premières étapes de la conception.

Cette thèse apporte plusieurs nouvelles contributions. Tout d'abord, nous introduisons un nouveau formalisme de modélisation appelé BIP stochastique et temps réel (SRT-BIP) pour la modélisation, la simulation et la génération de code de systèmes à base de composants. Ce formalisme hérite du framework BIP ses capacités de modélisation basées sur les composants et le temps réel et, en outre, il fournit des primitives pour exprimer des comportements stochastiques complexes.

Deuxièmement, nous étudions des techniques d'apprentissage automatique pour faciliter la construction de modèles de performance. Nous proposons d'améliorer et d'adapter une procédure d'apprentissage présentée dans la littérature pour déduire des modèles stochastiques et temporisés à partir d'exécutions concrètes du système, et de les exprimer dans le formalisme SRT-BIP.

Troisièmement, étant donné les modèles de performance dans SRT-BIP, nous explorons l'utilisation du model checking statistique (SMC) pour l'analyse d'exigences concernant la fonctionnalité et les performances du système. Pour ce faire, nous fournissons un framework complet, appelé $\mathcal{S}$BIP, en tant qu'outil de support pour la modélisation, la simulation et l'analyse des systèmes SRT-BIP. $\mathcal{S}$BIP est un environnement de développement intégré (IDE) qui implémente des algorithmes SMC pour des analyses quantitatives, qualitatives et d'événements rares, en plus d'une procédure d'automatisation pour l'exploration des paramètres d'une propriété. Nous validons nos propositions sur des études de cas réelles touchant à des domaines variés tels que les protocoles de communication, les systèmes concurrents et les systèmes embarqués.

Enfin, nous étudions plus en détail l'intérêt du SMC lorsqu'il est inclus dans des méthodes d'analyse de système élaborées. Nous illustrons cela en proposant deux approches d'évaluation des risques. Dans la première approche, nous introduisons une méthodologie en spirale pour modéliser des systèmes résilients avec des composants FDIR que nous validons à travers l'évaluation de la sécurité du système de locomotion d'un rover d'exploration planétaire. La deuxième approche concerne l'évaluation des politiques de sécurité des organisations selon une approche de sécurité offensive. L'objectif est de synthétiser des configurations de défense efficaces contre des stratégies d'attaque optimisées (qui minimisent le coût d'attaque et maximisent la probabilité de succès). Ces stratégies d'attaque sont obtenues en combinant l'apprentissage de modèles et les méthodes méta-heuristiques, dans lesquels le SMC a le rôle principal d'évaluer et de prioriser les potentielles stratégies candidates.

Contents

Chapter 1

Introduction

Computing systems are ubiquitous in modern society thanks to the advances in the hardware, information and communication technologies. These evolutions have contributed to the emergence of new applications and concepts such as Internet of Things (IoT), big data and Artificial Intelligence (AI), and autonomous driving. These new trends tend to drastically transform the society by breaking down the barriers between the physical and digital worlds, and bring changes to every sector, spanning from transportation to social interactions and even working habits.

As an answer to the newly appearing elaborate needs, systems are in a constant development to cover advanced services, resulting in products with increasing complexity. For example, recent cars provide a wide panel of advanced driver assistance systems (ADAS). Some systems, such as advanced cruise control and automatic parking, are meant to enhance the driver's experience, while other systems such as automatic braking and blind spot detection, directly address safety issues and aim at reducing the risk of car accidents.

Pushing to the limits the notions of speed, communication and utility, these systems have evolved, seeking for efficiency, interconnectivity and user customization. This quest of interconnectivity has even affected everyday objects. Nowadays, smart devices are everywhere. Put together, these devices provide capabilities such as information collection and environment control. In smart homes for example, domotic systems give a total and remote control over smart objects constituting the lighting and temperature systems and home appliance. The interaction with such connected structures is often operated through dedicated interfaces or simply smartphones.

With the boom of mobile applications, smartphones no longer only respond to the basic communication needs but serve to entertainment and navigation, and more recently, domotics and electronic payment. In 2017, studies estimated the number of smartphone users in the

world to 5 billions with an average number of installed applications reaching 80 [1]. Smartphones utilization observes a constant increase, reaching an average of 3.1 hours per day (in 2017) [2].

The increasing interest towards computing systems has made this market even more attractive to providers of information and communications technologies and smart devices. In this competitive and innovative context, reducing a product time-to-market plays a major positive role in the increase of market shares, implying drastic changes to the conventional design processes. Short design time is not the only requirement: design processes have to grasp the growing complexity of the systems and their uncertain environment, and to guarantee their functional correctness. Beyond functionality, the race for efficiency has become the central selling point. Systems are meant to be interactive, with a quick response time, albeit, limited in terms of autonomy and processing capacity, and communication is expected to be quasi instantaneous, even when dealing with a dynamic environment.

Additional design challenges arise from the involvement of these systems in daily life. When ensuring critical tasks involving human lives, users safety is a major concern. Security aspects are also to consider since these systems have access to (sensitive) user information and can be aimed for malicious purposes, such as, identity theft or ransomware.

This thesis intends to help in the design of complex systems. The focus is on the proposition of a design framework to shorten product time-to-market, while improving the quality of the designed systems in terms of performance. To overcome the complexity of systems, the heterogeneity of their components and their environment, we adopt a rigorous model-based and component-based approach relying on formal methods. In this approach, a centralized and unified view of the system is provided as a basis to reason about functional and extra-functional aspects at different abstraction levels.

1.1 Designing Complex Systems

The design of complex systems is at the intersection of several disciplines where control, software and hardware designers closely interact to produce the final product. The success of the design is dependent on their ability to account for all the components of the system when elaborating the solution. Indeed, it is crucial to clearly define and understand the interactions between system components, especially when the development of these components involves various expertise domains.

Model-based design has emerged as a promising solution to close the gap between cross-disciplinary design teams by providing them a unified view of the final result. The system model is the central element and serves as a high-level representation of the system under design. It encapsulates the expected behavior of the system together with an abstraction of its

[1] https://expandedramblings.com/index.php/smartphone-statistics, last access: 14-03-2019
[2] https://www.servicesmobiles.fr/20-chiffres-sur-le-marche-mobile-a-connaitre-en-2018-38749, last access: 14-03-2019

environment. Accounting for performance can be easily done by augmenting functional models with performance information.

Models turn out to be handy artifacts for the validation of design choices and the verification of the designed system. The reliance on high-level models allows for early fault detection and for high flexibility to requirement changes during the design process. Quantitative measurements can be easily extracted from performance models and used to guide further design steps. If correctly mastered, model-based design shortens the product time-to-market by minimizing communication delays between design teams, and the errors induced by the lack of information.

1.1.1 Model-Based Design Concepts and Methodology

Model-based design encompasses the activities of modeling the system at different abstraction levels, simulation, automatic code generation, and verification. Model-based design is a collection of eight concepts [5]:

1. Executable specification: the model is a representation of the system specification in an unambiguous manner. It can include a high-level representation of the targeted system behavior, and alternative use cases that the system has to manage and that are represented textually in the specification.

2. System-level simulation: simulating the system model is useful to gain a better understanding of the system and to increase its faithfulness. It provides a way to investigate system performance and identify design problems and potential errors.

3. What-if analysis: a model can serve to run simulation tests on individual components seeking to acquire knowledge about each component and its interactions with the system. It also enables for rapid exploration and assessment of different design alternatives.

4. Model elaboration: it consists in the refinement process that iterates over the system model by incrementally introducing more details. Each refinement is followed by the verification of system requirements. Corrections may be necessary when some requirements are not fulfilled.

5. Virtual prototyping: this concept consists in simulating the system under real-world operating conditions on a virtual prototype of the target architecture, before committing to a physical implementation of this latter. For example, abstracting at register-transfer level (RTL) provides a cycle-accurate simulation of a hardware component and enables designers to extract precise estimation of power consumption.

6. Continuous testing: it concerns the use of testing procedures at each stage of the system development. It consists in simulating the system with increasing levels of details, namely, (1) on predefined input and output (unit testing), (2) with an abstraction of

the environment and the target platform (closed-loop testing), (3) software-in-the-loop simulation, and (4) hardware-in-the-loop simulation. It increases the quality of the product by identifying errors at early stages of the design.

7. Automation: it consists in the development of dedicated tools or scripts to replace manual, repetitive and error-prone tasks. This includes model transformations, code generation, test case generation and formal verification.

8. Knowledge capture and management: this concept identifies the model as the central source of information. It represents all the knowledge about the system, including functional and extra-functional information.

These concepts are orthogonal to the development methodology, and can be applied to improve any development process, such as, the V-model, spiral, Scrum or extreme programming (XP). The most common development methodology in the design of embedded systems is the V-model. This latter relies on the association of a testing phase to each development step (see Figure 1.1).

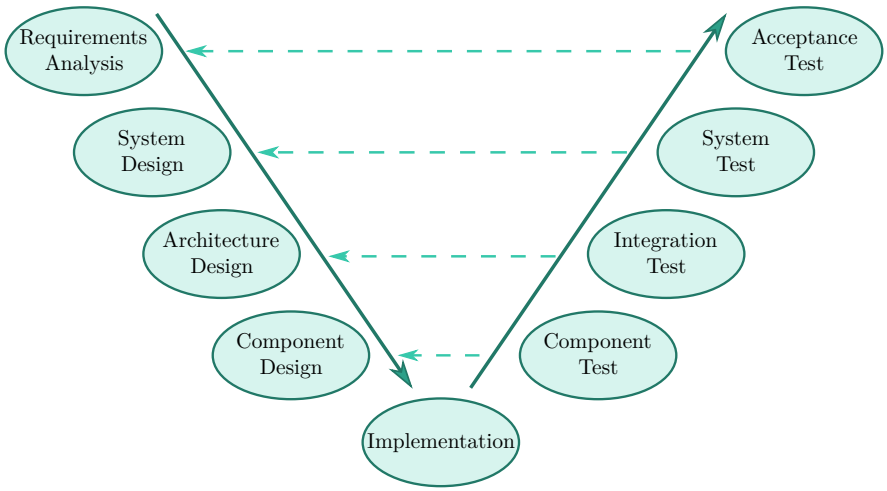


Figure 1.1: The V-model methodology

The main disadvantage of the V-model is the necessity to develop the entire system at the beginning which is not a simple task to do when some information about components or their interactions is not known initially. Model-based design contributes to the V-model methodology by allowing an early incomplete design of the system at a high-level of abstraction, and then specifying the missing details as part of a later refinement. It enables, hence, flexibility regarding changes.

1.1.2 System Requirements

In a system design project, system specification is the main source of information about the expectations on the system under design. This system is described by a collection of requirements that represent desired properties of the final product. Requirements are extracted from system specification and can be classified in two categories:

1. *Functional requirements* concern the functionality of the system and indicate what the system should do. These requirements include the expected calculations, data processing and technical details about the system components.

2. *Extra-functional requirements* concern how the system behaves when fulfilling its function. They are described with respect to a (quantitative) metric that permits to judge the operation of the system. These requirements include performance aspects such as computation time, communication latency, energy consumption, temperature and memory usage. They also embody sensitive aspects such as security and safety.

Requirements are often expressed in natural language as part of the system specification document. This latter often uses terminologies that are specific to the client's business domain, which sometimes introduce some incomprehension and ambiguity to non-experts in that domain. In model-based design, requirements are also modeled to facilitate their comprehension and avoid ambiguity. When formally described using property specification languages, they enable advanced capabilities such as monitor synthesis and automated verification.

1.1.3 System Model Characteristics

With the growing complexity of systems, building faithful system models becomes a challenge. Several factors impact the system complexity, such as the desired functionality and performance, environmental aspects and the nature of the involved components. As a matter of fact, recent systems tend to make hardware and software components of different nature coexist together. The designer has to cautiously study the different interactions, communications and data flow to ensure some compatibility in such heterogeneous systems.

The appeal for recent industrial and research activities, such as exploration of outer space, has created a new era of systems that carry out extremely advanced and elaborate tasks where time plays a major role. These complex systems have to independently operate in very hostile and unpredictable environment. To enable an accurate description of such systems, several key features are required from the modeling languages to manage complexity, describe time evolution and account for uncertainty.

Decomposition to handle complexity. A simple yet efficient way to comprehend system complexity is to model systems as the composition of less complex elements. Following this divide-and-conquer vision, the design of a system boils down to the design of its components and

their interactions. Component-based decomposition brings several advantages. The design of loosely coupled components enables re-usability which significantly reduces the design time. It also opens the doors for the design teams to work in parallel towards common agreed objectives.

Real-time is key. In complex applications, time has a serious impact on the system state and influences its behavior. Timed systems are often defined as systems where not only the outcome of the operated computation has to be correct, but the time needed to produce it also matters. In addition to functional constraints, timed systems have to meet temporal constraints.

Considering timing aspects at design time is an obligation as they may impact the correctness of the system or degrade its performance. The discrete and continuous time frameworks propose two different dynamics in modeling time progression. The former sees time as a succession of steps whereas the latter defines it as a continuum. Discrete time has long been considered as a viable abstraction of the system behavior. However, its modeling capabilities are very limited when designing time-sensitive systems such as in avionics or in cyber-physical security. In [92], the authors emphasize the necessity for modeling time over dense domains by investigating in-between-ticks attacks on cyber-physical security protocols. This work shows that some attacks can only be mitigated by models using dense time, as opposed to discrete time.

Probabilities for uncertainty and more. Probabilistic design puts forward the study of variability in the system. This variability often comes from the interaction of the system with external elements in its environment. This environment is, by its nature and size, very hard to represent and sometimes unpredictable. Abstractions are often required as a means to carry out very complex processes or interactions. Probabilities allow designers to intuitively represent uncertainty that may arise from the environment.

Probabilities also enable designers to account for potential imperfections in the software or hardware that constitute the system. For example, failure rates are the most common representation of the life cycle of hardware components.

Furthermore, probabilistic behaviors are sometimes artificially introduced in order to solve problems due to system symmetry or determinism. For example, the FireWire network protocol embeds a probabilistic waiting time sampling during the topology leader election in order to avoid node contention. Another example, in concurrent systems, is the adoption of a randomized access to shared resources as a mechanism to prevent starvation.

An important aspect of a modeling language is its ability to express complex (real-time and stochastic) models with a clearly defined semantics. Several modeling formalisms are used in formal methods and have the advantage to be extensively studied. These formalisms are endowed with efficient simulation methods and rigorous verification techniques. Relying on such formalisms provides a valuable automation in the design process. This design process can be even more simplified and enhanced with automatic code generation, which positively

impacts the quality of the final product since it eliminates potential errors due to traditional manual coding and repetitive tasks.

1.1.4 Challenges

Model-based design requires a specific training for the design team in order to acquire modeling skills and to master the different modeling tools. In addition, it may imply training and tools license costs, and time delays in the design process due to an initial learning curve. Regarding modeling tools, they often provide a partial coverage of the different design steps, as they lack accurate automatic code generation and/or formal verification algorithms.

The main difficulty of this approach is the construction of the system model, as it requires design skills and expertise. This step can be time consuming, especially for complex systems. Moreover, it consists in chasing contradictory objectives as the construction of a faithful yet high-level model requires to find a good balance between details and abstraction.

The construction of performance models is even more challenging since performance information are not always available at early design stages. A conventional way to obtain such information would be to extract it from system specification, or to compute it using static analysis. For example, the performance of a software program can be estimated by its worst case execution time (WCET). However, considering the impact of the hardware platform and environment uncertainty adds another dimension to the complexity of characterizing performance. As a solution, approaches [9, 123] based on the characterization of performance from concrete system executions have been proposed.

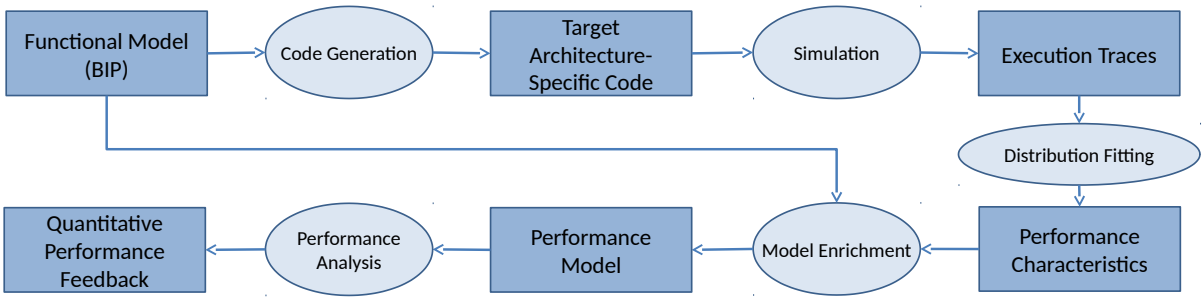


Figure 1.2: Overview of the ASTROLABE approach

In ASTROLABE [123], illustrated in Figure 1.2, the BIP framework [24] is used for the construction of system functional models and for the generation of platform-specific executables. The collected system executions are analyzed using statistical inference techniques to infer probability distribution functions describing system performance. Finally, the performance model is obtained by augmenting the functional model with the inferred performance information. This workflow is meant for general purpose systems and was tested on image processing algorithms and communication protocols. However, this method embodies several manual tasks that are tedious and error-prone. More recently, the authors of [9] proposed to use

property-based testing as a means to generate execution traces, and linear regression for the extraction of execution time distributions. This method is tailored for the study of web services response time.

1.2 Analysis of System Models

The quality of a system is dependent of its ability to offer a good performance and its absence of malfunctions and errors. To mitigate errors, industrials often adopt intensive testing policies, starting from unit testing, followed by integration testing and finally system testing. However, these tests, as intensive and costly as they can be, are unable to cover all the possible usage scenarios. Besides, the cost of a bug fix grows exponentially with the design stage [21]. The detection of a bug after the product release is a nightmare for companies since it may imply prohibitive costs, without mentioning the negative impact on their corporate identity.

Model-based design is supported by a plethora of analysis techniques that allow designers to identify errors at early design stages, and to extract performance measures that help making design decisions and avoiding architecture re-engineering costs. Analyzing system models consists in verifying whether the system satisfies formally expressed properties, representing functional or extra-functional requirements. A system is correct if it satisfies all described requirements. So, the correctness of a system is not absolute, but it is specific to the considered specification. It is worth mentioning that the accuracy of the analysis results strongly depends on the ability of the model to faithfully represent the target system.

Formal models are supported by systematic techniques to construct the set of all system states. This state space serves as a basis for exhaustive exploration through model checking, or for partial exploration using scenario-based testing or simulation.

1.2.1 Model Checking

Model checking is a verification technique that relies on a systematic exploration of the whole system state space. This technique, illustrated in Figure 1.3, was proposed as the result of two independent works [44] and [132]. Given a system model, a model checker enumerates all the states of the system and verifies whether the property under verification is satisfied at each system state. The exploration can prematurely abort in case the property is violated. The path leading to the violating state is presented as a diagnostic information, and showcases an error in the model.

This algorithmic approach for verification is faster than deductive methods such as theorem proving [76]. The challenge with such an approach is to handle large-scale models. With an explicit state space enumeration, a model checker can handle up to 10^9 states. More sophisticated explorations have been proposed in the literature. Symbolic model checking [114] relies on a symbolic representation for state transition systems using Binary Decision Diagrams

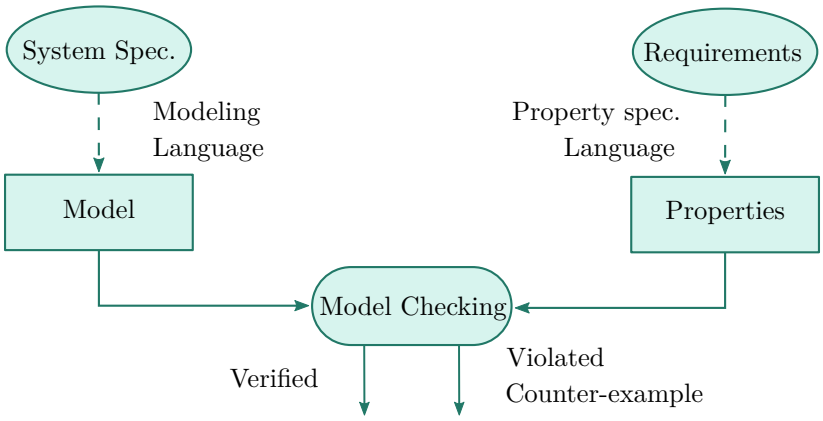


Figure 1.3: Illustration of the model checking technique

(BDD) [4] and is able to handle up to 10^{20} states. In [74], the authors propose to accelerate the model checking procedure by applying a partial state reduction. The idea here is to avoid checking the interleaving of independent actions. Bounded model checking was proposed in [29] as an exhaustive exploration up to a bounded horizon. This falsification method is useful to identify failures but do not guarantee system correctness. Other tracks have been explored for the sake of scalability of model checking such as compositional reasoning [47], abstraction [50], symmetry reduction [45], and parametrized verification [64].

Several model checking tools exist in the literature. For example, SLAM [19] is a proprietary tool developed by Microsoft Research for the checking of C programs against safety properties. BLAST [78] addresses the same problem of checking C programs but using the concept of *lazy abstraction* that relies on the principles of on-the-fly abstraction and on-demand refinement of the state space. SPIN [80] is a model checker for Linear-time Temporal Logic. It implements a finite automata-based model checking that computes the intersection of the system model, and an automaton that represents the complement of the desired property. KRONOS [35] is a model checker for Timed Computational Tree Logic(TCTL) over real-time systems. It considers components expressed as timed automata. Except for the integer clocks to express time, KRONOS does not support data variables.

Model checking suffers from scalability issues due to state-space explosion especially when applied on models with a large number of variables and/or control locations. Moreover, in case of property violation, this verification method does not tell how close the system is to satisfying the property and is limited to qualitative properties about the system functioning. In other words, it does not support quantitative analysis such as performance evaluation.

1.2.2 Probabilistic Model Checking

Probabilistic model checking [16] is an automated verification technique targeting systems exhibiting stochastic behaviors. It enables to compute the probability of a given stochastic

system to satisfy some property of interest in a numerical manner. This technique is based on the probability space induced on system runs to compute quantitative statements about the behavior and performance of the system expressed as probabilities.

Checking a probabilistic model against a property consists first to identify the states of the model where the property is satisfied (respectively violated). Secondly, a collection of linear equations are extracted from the model, encoding the different constraints represented by the model transitions, and accounting for the previously computed state labeling. Finally, the probability of the model satisfying the property is obtained by the numerical resolution of the constituted system of linear equations.

Several tools implement probabilistic model checking techniques. For example, PRISM [101] implements probabilistic model checking techniques for Markov Decision Processes (MDPs) and Probabilistic Timed Automata (PTAs), and includes support for cost and reward structures. This tool optimizes the memory required to run these techniques by using a symbolic representation of data. The Markov Reward Model Checker (MCMR) [93] supports Discrete and Continuous-Time Markov Chains (DTMCs and CTMCs) and was recently extended to Continuous-Time Markov Decision Processes (CTMDPs).

Despite their wide acceptance in academia and research, probabilistic model checking techniques are known to be memory-intensive, which makes them not scalable to large-size models.

1.2.3 Statistical Model Checking

Statistical Model Checking (SMC) is a special case of probabilistic model checking techniques following a simulation-based approach to estimate satisfaction probabilities. SMC was initially introduced by Younes in his thesis [151]. It relies on statistical tests (parametrized by precision and confidence attributes) to guard errors when generalizing a finite set of observations on the (probabilistic) system to global claims over the whole system.

SMC techniques have been developed for qualitative analysis (similarly to model checking) based on hypothesis testing, and for quantitative analysis (as for probabilistic model checking). The interest of SMC is also its support for the analysis of rare events, that is, events having a very low probability of occurrence, and hence, that are very unlikely to be observed with pure simulation. Such events can represent system errors that need to be fixed.

Several dedicated SMC tools have been recently developed to address different purposes, such as Ymer [153], Plasma-Lab [88] and $\mathcal{S}$BIP [122]. SMC facilities are also present in some of the probabilistic model checking tools such as PRISM and UPPAAL-SMC [53]. SMC tools mainly differ in the supported modeling formalisms, specification languages and the implemented SMC techniques.

Eventhough numerical methods such as probabilistic model checking often provide a higher accuracy than statistical methods [152], SMC has emerged as the best solution to handle

system complexity. This approach provides a parametrized trade-off between analysis speed and precision, controlled by the value of the statistical parameters. In addition, SMC is applicable to both system models and black-boxes, provided these latter behave entirely probabilistically.

1.3 Contributions

In this thesis, we address the problem of modeling and verification of complex systems exhibiting both probabilistic and timed behaviors. We focus more particularly on the analysis of extra-functional requirements including performance, security and safety, using statistical model checking. To do so, we define a rigorous model-based and component-based approach relying on formal methods and for which we propose a framework that supports the modeling, simulation, code generation and analysis of these stochastic real-time systems. This thesis brings several new contributions, as listed below.

1.3.1 Stochastic Real-Time Modeling Formalism

We introduce a new modeling formalism called Stochastic Real-Time BIP (SRT-BIP) for the modeling, the simulation and the code generation of component-based systems. This formalism inherits from the BIP framework its component-based and real-time modeling capabilities and, in addition, it provides comprehensive primitives to express complex stochastic behaviors. The resulting models have an underlying semantics of a Generalized Semi-Markov Process (GSMP), that is known to be the most expressive stochastic mathematical framework for the study of discrete-event systems.

1.3.2 Learning Performance Models

We investigate machine learning techniques to facilitate the construction of performance models from functional ones. We propose to enhance and adapt a state-of-the-art learning procedure to infer performance models expressed in the SRT-BIP formalism.

1.3.3 Modeling and Analysis Framework

Given performance models in SRT-BIP, we analyze requirements concerning system functionality and performance using Statistical Model Checking (SMC) techniques. To do so, we provide a full framework, called $\mathcal{S}$BIP, as a support tool for the modeling, code generation, simulation and analysis of SRT-BIP systems. $\mathcal{S}$BIP is an Integrated Development Environment (IDE) that implements SMC algorithms for quantitative, qualitative and rare events analyses together with an automated procedure for parameter exploration. We validate our proposals on real-life case studies ranging from communication protocols and concurrent systems to embedded systems.

1.3.4 Safety and Security Risk Assessment Approaches

We further investigate the interest of SMC when included in elaborate system analysis workflows where SMC is the heart of the overall process. We illustrate this by proposing two risk assessment approaches. In the first approach, we introduce a spiral methodology to model resilient systems with FDIR components that we validate on the safety assessment of a planetary rover locomotion system. The second approach addresses the security assessment of organization defenses following an offensive security approach. The goal is to synthesize impactful defense configurations against optimized attack strategies (minimizing attack cost and maximizing success probability). These attack strategies are obtained by combining model learning and meta heuristics, in which SMC plays the major role of scoring potential candidates.

1.4 Outline

This thesis is organized as follows:

- In Chapter 2, we present the state-of-the-art on modeling and analysis of stochastic systems. We recall general stochastic modeling formalisms and specification languages. Then we describe the statistical model checking algorithms and we give a short survey of SMC tools, before discussing the recent trends in the utilization of SMC.
- Chapter 3 presents the theoretical foundation of the SRT-BIP modeling formalism. We also discuss its implementation details.
- In Chapter 4, we propose the automatic construction of performance models expressed in SRT-BIP using machine learning techniques.
- Chapter 5 details the design of the $\mathcal{S}$BIP tool including its features and architecture.
- In Chapter 6, we address the performance analysis of real-life case studies in a variety of application domains using $\mathcal{S}$BIP.
- We introduce an iterative and incremental methodology, developed around the $\mathcal{S}$BIP framework, for the design of resilient systems equipped with FDIR capabilities, in Chapter 7.
- Chapter 8 addresses the risk assessment of organization security policy by confronting their defenses to sophisticated attacks. The proposed approach relies on the $\mathcal{S}$BIP framework for the quantitative evaluation of defense configurations when seeking for the most impactful defense mechanisms.
- Finally, conclusions and future work are presented in Chapter 9.

1.5 Publications

- [116] Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, and Saddek Bensalem. Improved learning for stochastic timed models by state-merging algorithms. In *NASA Formal Methods Symposium*, pages 178–193. Springer, 2017.
- [117] Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Mahieddine Dellabani, Axel Legay, and Saddek Bensalem. $\mathcal{S}$BIP 2.0: Statistical Model Checking Stochastic Real-Time Systems. In *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, pages 536–542, 2018.
- [124] Ayoub Nouri, Braham Lotfi Mediouni, Marius Bozga, Jacques Combaz, Saddek Bensalem, and Axel Legay. Performance evaluation of stochastic real-time systems with the SBIP framework. *International Journal of Critical Computer-Based Systems*, 8(3-4):340–370, 2018.
- [118] Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Axel Legay, and Saddek Bensalem. Mitigating security risks through attack strategies exploration. *Inter- national Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, 2018

1.6 Tools

During this thesis, we developed several tools including:

- Improved learning algorithm for timed and stochastic models.
  Available on-line: http://www-verimag.imag.fr/~nouri/drta-learning
- SRT-BIP: a framework for the modeling, code generation and simulation of stochastic real-time systems.
  Available on-line: https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/bip/compiler/tree/stochastic-real-time
- $\mathcal{S}$BIP: an integrated tool for modeling and analyzing SRT-BIP models.
  Available on-line: http://www-verimag.imag.fr/BIP-SMC-A-Statistical-Model-Checking.html
- A tool for the synthesis of impactful defense configurations and attack strategy exploration, in the context of security risk assessment.
  Not available on-line.

Chapter 2

State of the Art on Stochastic Modeling and Analysis

Due to the difficulty to match all the systems requirements, early analysis has become very popular in the design of complex systems. System-level design methodology has been introduced to provide a high-level reasoning over a representation of the system. This cost-effective manner to design systems aims at optimizing the design process while minimizing architecture re-engineering costs. High-level representations of the system are usually based on formal models, which have a solid theoretical background.

Besides correct functional behavior, systems are required to be competitive in terms of their response time and resource utilization. These performance aspects are strongly related to the computational power of the system, but are also influenced by stochastic fluctuations in the system behavior or in its environment. Considering performance at a model-level involves expressive formalisms that simultaneously handle complex timed and stochastic behaviors in order to faithfully represent performance information.

The correctness of systems is often conditioned by their ability to satisfy a set of requirements, describing the desired functionality or performance. Several specification languages exist to formally represent these requirements. Verifying such requirements on a system model is an important step towards guaranteeing them at the system deploy-time. Statistical Model Checking (SMC) has emerged as a promising approach for the quantitative analysis of systems. This lightweight verification technique provides scalable algorithms, compared to exhaustive methods such as Model Checking, and is supported by a wide panel of tools.

In this chapter, we present the state of the art on stochastic modeling and analysis in the context of SMC. We start by recalling general stochastic modeling formalisms in Section 2.1. Section 2.2 introduces two temporal logics, namely, Linear-time Temporal Logics (LTL) and Metric Temporal Logics (MTL). In Section 2.3, we present the statistical model checking algorithms and we give a short survey of SMC tools in Section 2.4. We finally discuss the recent trends in the utilization of SMC in Section 2.5, before concluding in Section 2.6.

2.1 Modeling Formalisms

Modeling formalisms are useful to represent phenomena at higher levels of abstraction as a solution to cope with the difficulty and the cost to understand and analyze such complex systems. This difficulty grows even faster when addressing both functional and extra-functional aspects. Several modeling formalisms exist in the literature and provide different expressiveness capabilities that are often classified based on five concepts: action and delay non-determinism, probabilistic branching, clocks and random variables. *Action non-determinism* refers to the ability of a system to behave differently even when facing similar conditions whereas *delay non-determinism* is a means to under-specify the timing of events by only attaching them with an interval of viable time valuations.

Probabilistic branching and *random variables* are key aspects is the description of probabilistic behavior. The former enables to attach probabilities to alternative events, whereas the latter allows one to describe the timing of an event through a probability distribution function. These distributions arise the question of memorylessness, that is, the probability of an event to occur only depends on the current state of the model, and not on the event history. This memorylessness implies the irrelevance of past information to determine what happens next, but also that the time spent in the current state does not affect the selection of the next event. Models that fulfill these state and age memories constraints are called *Markov models*, and usually rely on exponential distributions to express time stochasticity. These exponential distributions are known to be the only continuous ones that are memoryless. However, in practice, exponential distribution is not always sufficient to model real-life phenomena and more complex distributions are often needed such as Normal and Weibull distributions. This is the case, for example, when describing a computer process lifetime that is known to have a large number of short jobs and only few long jobs but with a large variance [77]. It is sometimes possible to simulate complex distributions by decomposing them to exponential distributions depending on their properties. For example, summing n exponential random variables with parameter λ would result in a random variable following a gamma distribution with parameters $(n,\ \lambda)$. But it is not always that straightforward to achieve. In addition, manipulating such composite models would introduce additional complexity in terms of readability and comprehension. Hence, the need for an inherent support for general distributions has arisen as a key feature of stochastic formalisms.

The time dynamics indicates how time elapses in the model, and can be either continuous or discrete. *Clocks* are well-known structures in Timed Automata (TA) theory [11] that are used to represent real-time. They are a key concept in building *continuous* time models where time takes values in a dense time domain. In these models, the time elapsing between two events can be measured and can be subject to constraints. In contrast, *discrete* time models evolve step-wise, that is, the inter-event time is abstracted to the notion of steps. This time dynamics does not use clocks, and is applied when the amount of time that elapses between events is not

	DTMC	MDP	CTMC	GSMP	PTA	STA
Action non-determinism	-	+	-	-	+	+
Delay non-determinism	-	-	-	-	+	+
Probabilistic branching	+	+	+	+	+	+
Random variables	-	-	EXP	+	-	+
Clocks	-	-	+	+	+	+

Table 2.1: Modeling formalisms with probabilistic branching [32]

relevant. For example, when studying the number of heads or tails in n coin tosses, the time spent between two tosses does not actually affect the outcome of the experiment.

Table 2.1 non-exhaustively lists some probabilistic modeling formalisms and their expressiveness capabilities. The focus is put on formalisms that allow for probabilistic branching, and hence, formalisms such as Timed Automata (TA) [11] and Labeled Transitions Systems (LTS) [94] are out of this scope. Discrete Time Markov Chains (DTMC) [99] and Markov Decision Processes (MDP) [65] adopt a discrete time dynamics and differ in terms of non-determinism. DTMCs are fully stochastic, that is, non-determinism in not allowed whereas MDPs support action non-determinism. Except for DTMCs and MDPs, the remaining formalisms in Table 2.1 rely on clocks to express continuous time and also differ in terms of non-determinism in addition to the probability distributions allowed for the modeling of time stochasticity. Continuous Time Markov Chains (CTMC) [99] and Generalized Semi-Markov Processes (GSMP) [73] do not permit neither delay nor action non-determinism. CTMC respects the Markov property since time progression is exponentially distributed. However, GSMP relaxes the age memory restriction and supports generally distributed inter-event times. Regarding non-determinism, Probabilistic Timed Automata (PTA) [102] and Stochastic Timed Automata (STA) [28] include capabilities to model both action and delay non-determinisms. In these probabilistic extensions of TA, a system can either perform an action by making discrete transitions, or remain in the same control state and let time pass. The choice of the action to perform is non-deterministic and and the target state is determined probabilistically. The main difference between these two formalisms is in their time evolution. PTA determines event delays non-deterministically, whereas these delays follow a general distribution function in an STA.

In this section, we give an overview of stochastic models that we will be referring to throughout this dissertation. More specifically, we present the discrete and continuous time Markov chains, and the generalized semi-Markov processes. Since we are in a context of a verification using SMC, it is important to focus on models that can be simulated. Hence, we will not consider modeling formalisms that contain action or delay non-determinism, such as PTA and STA. However, we briefly recall MDPs, that we will be using in the application of SMC to Security Risk Assessment, in Chapter 8.

2.1.1 Discrete Time Markov Chain

Discrete time Markov chains are probabilistic models built from a finite set of control states connected with edges. These transition systems determine the successors of each state based on probability distributions. Formally, a DTMC is described as follows:

Definition 2.1.1 (Discrete Time Markov Chain)**.** A discrete time Markov chain is a tuple $\langle S, \iota, \pi, \Sigma, L \rangle$ where:

- S is a finite and non-empty set of control states,
- $\iota : S \longrightarrow [0, 1]$ is the initial states distribution such that $\Sigma_{s \in S} \iota(s) = 1$,
- $\pi : S \times S \longrightarrow [0, 1]$ is the transition probability function such that $\forall s \in S, \Sigma_{s' \in S} \pi(s, s') = 1$,
- Σ is the set of atomic propositions,
- $L : S \longrightarrow 2^{\Sigma}$ is the state labeling function. It assigns to each state a set of atomic propositions that are true at that state.

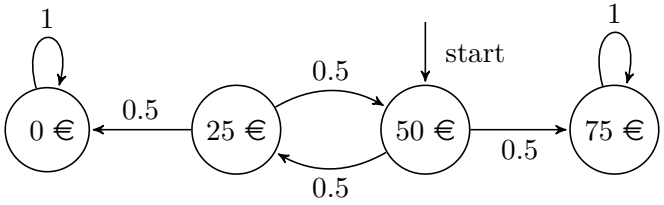


Figure 2.1: A DTMC of a gambler at a roulette game

Example 2.1.1. Figure 2.1 is an example of DTMC that models a roulette gambling game. In this game, the gambler sits at the table with 50 €. A dealer spins a roulette that outputs "red" or "black" colored numbers. At each spin, the gambler bets 25 € on "red". If "red" occurs, he wins 25 € in addition to recouping the amount gambled. However, if "black" occurs, he looses his 25 €. The player leaves the table under two conditions : either he has no more money to gamble, or he reaches 75 €. In this example, the control states are labeled by the possible amounts of money that the gambler can have at any time. The transitions between states represent the occurrence of "red" or "black" and have equal chances to occur. We can see that the states 0 € and 75 € are absorbing states, i.e., no other state is reachable from them.

Let $\mathcal{D} = \langle S, \iota, \pi, \Sigma, L \rangle$ be an DTMC. $\mathcal{D}$ is said to be deterministic if and only if two conditions are verified: The first one is the uniqueness for the initial state, i.e., $\exists s_0 \in S$ such that $\iota(s_0) = 1$. The second condition ensures that for every state s, two different transitions cannot lead to states having the same label, i.e., $\forall s \in S, \forall \sigma \in \Sigma$, it exists at most one s' such that $\pi(s, s') > 0 \wedge L(s') = \sigma$.

In a deterministic DTMC, no ambiguity is met when choosing a destination state, knowing the current state and the observed label. A path p of $\mathcal{D}$ is a sequence of states $s_0 s_1 ...$ such that

$\iota(s_0) > 0$ and $\pi(s_i, s_{i+1}) > 0$, for all $i \geq 0$. A trace ω associated with a path is the word $\sigma_0\sigma_1...$ such that $L(s_i) = \sigma_i$, for all $i \geq 0$.

2.1.2 Continuous Time Markov Chain

Continuous time Markov chains extend DTMCs with the notion of continuous time-steps. This formalism is suited to represent stochastic models where the inter-event time is relevant. In such models, time progresses according to exponential distributions which parameters λ are specified for each transition. Recall that an exponential distribution is a continuous distribution with support $[0, +\infty)$ and mean μ, parametrized by $\lambda = {}^1/_\mu$. Its probability density function is given as follows:

$$f(t) = \begin{cases} \lambda.e^{-\lambda t}, & \text{if } t > 0 \\ 0, & \text{otherwise} \end{cases}$$

Formally, a CTMC is defined as follows:

Definition 2.1.2 (Continuous Time Markov Chain)**.** A continuous time Markov chain is a tuple $\langle S, s_0, R, \Sigma, L\rangle$ where:

- S is a finite and non-empty set of control states,
- s_0 is the initial state,
- $R : S \times S \longrightarrow \mathbb{R}_{\geq 0}$ is the transition rate matrix,
- Σ is the set of atomic propositions,
- $L : S \longrightarrow 2^\Sigma$ is the state labeling function. It assigns to each state a set of atomic propositions that are true at that state.

The transition rate matrix R encodes the presence of a transition between a pair of states $\langle s_i, s_j\rangle$ with strictly positive rate values, i.e., $R_{s_i,s_j} > 0$. This latter represents the parameter of the exponential distribution quantifying the delay of the considered transition.

The system starts at the initial state s_0. At a given state s_i, the set of enabled transitions T_i is composed of all the transitions available at s_i and decorated with a strictly positive rate, i.e., $T_i = \{t_{ij} \mid R_{s_i,s_j} > 0\}$. For each enabled transition t_{ij} between the states $\langle s_i, s_j\rangle$, a waiting delay ω_{ij} is sampled according to an **exponential**(R_{s_i,s_j}). The selection of the transition to execute among the enabled ones follows a race policy, namely, the one that was assigned the shortest waiting delay.

We recall that the minimum over a set of n independent exponentially distributed random variables with different rate parameters λ_k is exponentially distributed with a parameter representing the sum of these λ_k. Consequently, the race policy boils down to the sampling from an underlying exponential distribution with a rate parameter $r(s_i) = \Sigma_{t_{ij} \in T_i} R_{s_i,s_j}$.

Given a CTMC $\mathcal{C} = \langle S, s_0, R, \Sigma, L \rangle$, the probability to select a transition can be determined independently of its waiting time, in a straightforward manner:

$$\pi^{\mathcal{C}}(s_i, s_j) = \begin{cases} {}^{R(s_i,s_j)}/_{r(s_i)}, & \text{if } r(s_i) > 0 \\ 1, & \text{if } r(s_i) = 0 \wedge s_i = s_j \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

The resulting stochastic model is called *embedded DTMC* and is formally defined as follows:

Definition 2.1.3 (Embedded DTMC in a CTMC)**.** Given a CTMC $\mathcal{C} = \langle S, s_0, R, \Sigma, L \rangle$, the embedded DTMC $\mathcal{D}$ in $\mathcal{C}$ is a tuple $\langle S', \iota', \pi', \Sigma', L' \rangle$ where:

- $S' = S$, $\Sigma' = \Sigma$, and $L' = L$,
- $\iota'(s_0) = 1$, and $\forall s \neq s_0 \ \iota'(s) = 0$,
- $\pi' = \pi^{\mathcal{C}}$ as defined in Equation 2.1.

At a given state s, the transition probability function of the embedded DTMC assigns to each transition a probability that is proportional to its rate parameter if the sum of the rates $r(s)$ at this state s is positive. Note that a rate sum equal to zero characterizes absorbing states, that are obviously represented in the embedded DTMC as a self-loop with probability 1.

As a result, a second way to interpret a CTMC would be to sample waiting delays from exponential distributions with parameters $r(s)$, then to randomly select the target state according to the transition probability function $\pi^{\mathcal{C}}$.

A path p of $\mathcal{C}$ is an alternation of states and waiting delays $s_0 t_0 s_1 t_1 ...$ such that the rates $R(s_i, s_{i+1})$ and the waiting delays t_i are strictly positive.

2.1.3 Generalized Semi-Markov Process

Generalized semi-Markov processes have been proposed as a comprehensive framework for the description and the analysis of discrete-event systems. In such systems, time does not always progress exponentially. In GSMPs, each event is attached with a cumulative distribution function (cdf) ruling its timing behavior.

GSMPs are semi-Markov processes, in the sense that the determination of the next state is not only dependent on the current state, but also on the amount of time spent in that state. More formally, a GSMP is defined as follows:

Definition 2.1.4 (Generalized Semi-Markov Process)**.** A generalized semi-Markov process is a tuple $\langle S, \iota, \Sigma, \mathit{active}, T, G \rangle$ where:

- S is a finite and non-empty set of control states,
- $\iota : S \longrightarrow [0, 1]$ is the initial states distribution such that $\Sigma_{s \in S} \iota(s) = 1$,
- Σ is the set of events (actions),

- *active*: $S \longrightarrow 2^{\Sigma}$ is the set of active events at each state,
- $T : S \times \Sigma \longrightarrow S$ is the (deterministic) transition function,
- G is the stochastic clock structure, that assigns a cdf to each event $\sigma \in \Sigma$.

The set of active events contains the events that can be executed at a given state. This set is updated when entering a new state (after the execution of an event), that is, the newly active events are added to the set while the events that became inactive at the new state are removed from that set.

The system starts at an initial state s_0 selected by sampling ι. When entering a state s, the set of active events is updated accordingly. For each newly active event σ_i, a waiting delay is sampled according to the cdf G_i associated with this event, and a clock is initiated to count down its remaining lifetime. The choice of the event to be executed follows a race policy, that is, by selecting the event having the shortest remaining lifetime. The execution itself occurs when the remaining lifetime reaches 0 and triggers a state change.

2.1.4 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical framework for modeling decision making. In addition to a probabilistic behavior, actions are chosen in a non-deterministic manner.

A formal definition of a Markov decision process is given below:

Definition 2.1.5 (Markov Decision Process)**.** A Markov decision process is a tuple $\langle S, Act, \iota, \pi, \Sigma, L\rangle$ where:

- S is a finite and non-empty set of states,
- Act is a finite set of action labels,
- $\iota : S \longrightarrow [0, 1]$ is the initial states distribution, such that $\Sigma_{s\in S}\iota(s) = 1$
- $\pi : S \times Act \times S \longrightarrow [0, 1]$ is the transition probability function, such that $\forall s \in S$ and $\forall a \in Act, \Sigma_{s'\in S}\pi(s, a, s') \in \{0, 1\}$,
- $\Sigma = 2^{AP}$ is the state-label alphabet representing a set of atomic propositions,
- $L : S \longrightarrow \Sigma$ is the state labeling function. It assigns to each state a set of atomic propositions that are true at that state.

At each state of the MDP, several actions can be available. Choosing which action to perform is done non-deterministically. Then, the target state is selected according to a probability distribution function corresponding to the source state and the chosen action.

A path p of an MDP is a sequence $p = s_0a_1s_1...a_is_i...$, where s_0 is an initial state, i.e., $\iota(s_0) > 0$, and $\pi(s_i, a_{i+1}, s_{i+1}) > 0$. For a given path p, the corresponding trace is an alternation

of action and state labels $\omega = \sigma_0 a_1 \sigma_1 ... a_i \sigma_i ...$ such that the transition between s_i and s_{i+1} is labeled a_{i+1}, i.e., and $L(s_i) = \sigma_i$. A trace corresponding to a possible path of an MDP is a word that belongs to the language recognized by this MDP.

2.2 Specification Languages

System requirements are a set of desired properties that characterize a system under study/design. In system design, these requirements are often expressed in natural language and are subject to misleading interpretation. To cope with this ambiguity, specification languages have been proposed as a rigorous framework for clear requirement formulation and consistent interpretation. Temporal logic is a formalism for the specification of correctness properties in an intuitive syntactic manner and with a well-defined semantics. It enriches propositional or predicate logic with temporal operators to allow for reasoning about infinite behavior of systems.

Time in temporal logics can be of two different natures, namely, linear or branching. At each moment, the linear view identifies a single successor moment, whereas in the branching view time may split into alternative courses, in a tree-like structure. The progression of time is done step-wise when defined over a discrete time domain, or continuous when defined over a dense time domain.

In the literature, several specification languages exist and are suited for different purposes. Table 2.2 presents a subset of temporal logics frequently used in practice, together with their features. For a more exhaustive description and comparison of temporal logics, we refer the interested reader to [17].

	Time logic	Time domain
LTL [131]	linear	discrete
MTL [96]	linear	continuous
CTL [46]	tree	discrete
TCTL [10]	tree	continuous

Table 2.2: Example of temporal logics and their distinguishing features

In this thesis, we focus on linear-time logics for their qualitative notion of time that is path-based. This makes them suitable candidates in the context of SMC algorithms that reason about one trace at a time. More specifically, in this section, we will give an overview of the Linear-time Temporal Logic (LTL) and its real-time extension, denoted Metric Temporal Logic (MTL).

2.2.1 Linear-time Temporal Logic

Linear-time temporal logic is a formalism for expressing properties with a linear notion of time. It provides temporal operators as a means to reason over infinite behaviors. This formalism is useful to specify notions such as order in the occurrence of state labels, but also safety or liveness properties. In this logic, time progresses discretely, that is, it does not support real-time.

LTL syntax. For a given set of atomic propositions, denoted AP, the syntax of an LTL formula ϕ is inductively defined by the following grammar:

$$\phi ::= \boldsymbol{t} \mid \boldsymbol{f} \mid ap \mid \neg ap \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 \,\mathcal{U}\, \phi_2 \mid \phi_1 \,\mathcal{R}\, \phi_2, \text{ where } ap \in AP$$

The operator $\bigcirc \phi$ is the *next* operator, while $\phi_1 \,\mathcal{U}\, \phi_2$ is the *Until* operator, which stands for ϕ_1 holds until ϕ_2. The *Release* operator $\mathcal{R}$ is the dual of $\mathcal{U}$, i.e., $\phi_1 \,\mathcal{R}\, \phi_2 \equiv \neg(\neg\phi_1 \,\mathcal{U}\, \neg\phi_2)$. The *Eventually* and the *Globally* operators are expressed respectively as $\Diamond\phi \equiv \boldsymbol{t} \,\mathcal{U}\, \phi$, and $\Box\phi \equiv \boldsymbol{f} \,\mathcal{R}\, \phi$. Their meaning is respectively ϕ eventually holds, and ϕ always holds. Note that we restrict to LTL formulas written in Negative Normal Form (NNF), that is, the negation is pushed down to the atomic propositions.

In terms of precedence order on the operators, the unitary operators ($\neg$ and $\bigcirc$) are stronger than binary ones, and temporal binary operators ($\mathcal{U}$ and $\mathcal{R}$) stronger than boolean binary ones ($\wedge$ and $\vee$).

LTL Semantics. An LTL property ϕ induces a language $\mathcal{L}_\phi$ over the alphabet 2^{AP}. This language includes all the words σ over 2^{AP} that satisfy ϕ, i.e., $\mathcal{L}_\phi = \{\sigma \in (2^{AP})^\omega \mid \sigma \models \phi\}$. Let a word $\sigma = \sigma_0\sigma_1...$ be a sequence of symbols $\sigma_i \in 2^{AP}$ and $\sigma[i..] = \sigma_i\sigma_{i+1}...$ a suffix of σ starting from the i^{th} position. Similarly, $\sigma[..i] = \sigma_0...\sigma_i$ is the prefix of σ that ends at the i^{th} position. The satisfaction relation ($\models$) is given as follows:

- $\sigma \models \boldsymbol{t}$
- $\sigma \models ap$ iff $ap \in \sigma_0$
- $\sigma \models \neg ap$ iff $ap \notin \sigma_0$
- $\sigma \models \phi_1 \wedge \phi_2$ iff $\sigma \models \phi_1$ and $\sigma \models \phi_2$
- $\sigma \models \bigcirc\phi$ iff $\sigma[1..] \models \phi$
- $\sigma \models \phi_1 \,\mathcal{U}\, \phi_2$ iff $\exists j \geq 0.\ \sigma[j..] \models \phi_2$ and $\sigma[i..] \models \phi_1$, for all $0 \leq i < j$

The satisfaction relation for the boolean *Or* ($\vee$) is deductible from the rules for the *And* ($\wedge$) and negation ($\neg$) operators. Similarly, the interpretation of the temporal operators *Release* ($\mathcal{R}$), *Eventually* ($\Diamond$) and *Globally* ($\Box$) can be obtained by first rewriting them using the *Until* ($\mathcal{U}$) and negation operators.

Bounded LTL. The LTL semantics is defined over words of *infinite* length. Since we are interested in checking such properties in a context of SMC, we only consider a bounded fragment of LTL, denoted Bounded LTL or BLTL [66]. In [155], E.M. Clarke et al. proved that any BLTL property can be checked only using a finite prefix of a word σ. This theorem guarantees that only finite words (simulations) are required to perform SMC on BLTL properties, and hence, that the SMC procedure always terminates.

BLTL restricts the scope of temporal operators to a bounded number of steps. In other words, these modalities are decorated with an integer value k indicating the maximum number of tolerated steps before concluding, and we write: bounded *Next* ($\bigcirc^k$), bounded *Until* ($\mathcal{U}^k$), bounded *Release* ($\mathcal{R}^k$), bounded *Eventually* ($\Diamond^k$) and bounded *Globally* ($\Box^k$). Semantically, the interpretation of $\bigcirc^k$ and $\mathcal{U}^k$ is given as follows:

- $\sigma \models \bigcirc^k \phi$ iff $\sigma[1..] \models \phi \ \wedge \ \sigma[..k] \models \phi$
- $\sigma \models \phi_1 \ \mathcal{U}^k \ \phi_2$ iff $\exists j \in [0,k].\ \sigma[j..] \models \phi_2$ and $\sigma[i..] \models \phi_1$, for all $0 \leq i < j$

Likewise, the rest of the bounded temporal operators are interpreted accordingly.

2.2.2 Metric Temporal Logic

Metric Temporal Logic (MTL) [96] is an expressive temporal logic that extends LTL by introducing an explicit representation of time. It provides a linear view of time with a real-time progression. MTL temporal operators are similar to LTL ones with the difference of having a time interval $I \subseteq \mathbb{N}^+$ constraining them.

MTL syntax. For a given a set of atomic propositions (AP), the syntax of an MTL formula ϕ is inductively defined by the following grammar:

$$\phi ::= \boldsymbol{t} \mid \boldsymbol{f} \mid ap \mid \neg ap \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 \ \mathcal{U}_I \ \phi_2 \mid \phi_1 \ \mathcal{R}_I \ \phi_2, \text{ where } ap \in AP$$

The *Until* operator $\phi_1 \ \mathcal{U}_I \ \phi_2$ stands for ϕ_1 holds until ϕ_2 does at some time in I. The *Release* operator is written as $\phi_1 \ \mathcal{R}_I \ \phi_2 \equiv \neg(\neg\phi_1 \ \mathcal{U}_I \ \neg\phi_2)$. The *Eventually* and the *Globally* operators are expressed respectively as $\Diamond_I \phi \equiv \boldsymbol{t} \, \mathcal{U}_I \ \phi$, and $\Box_I \phi \equiv \boldsymbol{f} \, \mathcal{R}_I \ \phi$. Their meaning is respectively ϕ eventually holds at some time in I, and ϕ always holds at any time in I.

MTL semantics. An MTL formula is interpreted over timed words. Formally, a timed word σ is a sequence of pairs $\langle \sigma_i, v_i \rangle$, such that timestamps v_i constitute an increasing time sequence, i.e., $v_i \leq v_{i+1}$. The notation of prefix and suffix of timed words applies similarly to the previously defined one for (untimed) words, that is, respectively $\sigma[..i]$ and $\sigma[j..]$. The inductive definition of the satisfaction relation is given as follows:

- $\sigma \models \boldsymbol{t}$

- $\sigma \models ap$ iff $ap \in \sigma_0$
- $\sigma \models \neg ap$ iff $ap \notin \sigma_0$
- $\sigma \models \phi_1 \wedge \phi_2$ iff $\sigma \models \phi_1$ and $\sigma \models \phi_2$
- $\sigma \models \phi_1 \vee \phi_2$ iff $\sigma \models \phi_1$ or $\sigma \models \phi_2$
- $\sigma \models \bigcirc \phi$ iff $\sigma[1..] \models \phi$
- $\sigma \models \phi_1 \, \mathcal{U}_{[a,b]} \, \phi_2$ iff $\exists j.\ a \leq v_j - v_0 \leq b,\ \sigma[j..] \models \phi_2$ and for all $0 \leq j < i.\ \sigma[j..] \models \phi_1$

MTL variants. The problems of satisfiability and model checking for MTL is known to be highly undecidable [34]. To cope with the undecidability, several variants have been proposed. Table 2.3 summarizes decidable fragments of MTL together with their restriction on the constraining intervals I of temporal modalities. For a complete survey on the decidability of fragments of MTL and their expressiveness, see [129].

Sublogic	Restriction
$\text{MTL}_{0,\infty}$ [12]	The bounds on the time constraints are limited to: 0 for the lower bound or ∞ for the upper bound
MITL [12]	No punctual formulas : time intervals cannot be a singleton value
BMTL [34]	All time constraints have finite length : No right-open interval is accepted. However, it does not handle invariance (unrestricted $\Box$)
Safety MTL [128]	Time-bounded $\mathcal{U}_I$ and $\Diamond_I$ but allows unbounded $\Box$ (invariance)
Co-Flat MTL [34]	Restrict *Until* operators $\phi_1 \, \mathcal{U}_I \, \phi_2$ to the cases where, if I is unbounded then ϕ_2 is an LTL formula

Table 2.3: Decidable fragments of MTL

Since we are interested in monitoring finite simulations in the context of SMC procedures, we restrict ourselves to the BMTL sublogic. In this logic, all the temporal operators must be attached with a right-closed time interval.

2.3 Statistical Model Checking Techniques

Probabilistic Model Checking (PMC) [17] is a means to evaluate quantitative properties of stochastic systems. This numerical approach relies on the computation of probability measures of paths that exhibit a desired behavior, usually expressed in some kind of temporal logic. This approach is context-specific, that is, algorithms are designed for specific modeling and specification formalisms. Furthermore, PMC leads to intractable state spaces in most real-life applications, in addition to being a time and memory consuming process.

Statistical Model Checking (SMC) has been proposed as a simulation-based alternative to cope with these limitations. Compared to other formal verification techniques, including probabilistic model checking, SMC is *scalable*. This feature is inherent to the method: only a subset of the system's executions is explored, while the underlying statistical algorithms can be easily parallelized. Its reliance on system traces makes it applicable to any executable system, independently on its size or structure. Even though the obtained results are only estimations, their accuracy is controlled with confidence parameters that bound the estimation error, and which distinguish it from pure simulation techniques. Also, corner cases and rare events which might be missed by simulations can be handled with SMC. Specific techniques such as *importance sampling* and *importance splitting* [87, 89] have been recently adapted to SMC in order to efficiently deal with this class of events. Another important feature of SMC is its usability on both models and implementations, including black-box systems, provided that implementations are obtained from formally defined models with a purely stochastic semantics and that code generation preserves this semantics.

SMC answers two types of questions; **Qualitative:** is the probability for S to satisfy ϕ greater or equal to a certain threshold θ? and **Quantitative:** what is the probability for S to satisfy ϕ?

The essence of SMC is to randomly draw a finite number of independent traces that are monitored against the properties under study. Then, the approach uses the collected verdicts over system traces to conclude a global verdict over the whole system, as depicted in Figure 2.2. Each trace is obtained by a discrete event simulation and the output of its monitoring can be seen as an observation of a Bernoulli distribution Y, which is $y = 1$ if the trace satisfies the property and $y = 0$ otherwise. This Bernoulli distribution is parametrized by p ($Y \sim Bernoulli(p)$), where p is the unknown probability of the system S to satisfy the property ϕ, i.e., $Pr[S \models \phi] = p$.

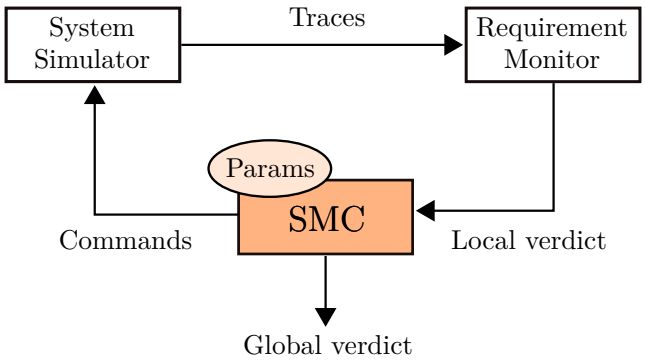


Figure 2.2: The statistical model checking process

Among statistical testing algorithms for stochastic systems verification we can cite Single Sampling Plan (*SSP*), Simple Probability Ratio Test (*SPRT*) [143, 151], and Probability

Estimation (*PE*) [79]. In the following, we briefly recall these three procedures, in addition to rare events analysis algorithms, namely, importance sampling and importance splitting.

2.3.1 Qualitative Analysis

The main approach to answer the qualitative question is based on *hypothesis testing* [151]. To determine whether $p \geq \theta$, one can test the null hypothesis $H : p \geq \theta$ against its alternative $K : p < \theta$. A test-based solution does not guarantee a correct result but it is possible to bound the probability of making an error. The *strength* (α, β) of a test is determined by two parameters, α and β, such that the probability of accepting K (resp., H) when H (resp., K) holds is less or equal to α (resp., β). Since it is impossible to ensure a low probability for both types of errors simultaneously [8], a solution is to relax the test by using an *indifference region* $[p_1, p_0]$ centered around θ and to test $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$. This indifference region is defined as a function of θ and a user-specified parameter δ representing the half-width of that indifference region, that is, $p_1 = \theta - \delta$ and $p_0 = \theta + \delta$. It is worth mentioning that both H_0 and H_1 are considered false when $p \in [p1, p0]$.

Depending on the sampling policy, hypothesis testing can be performed either on a precomputed set of observations, or by generating observations on-demand. In the former, the test is run on a sample of predetermined size that is computed from the statistical parameters [144], whereas, the latter implements a sequential test that does not require to know the sample size beforehand.

Single Sampling Plan (SSP). It is a strategy that permits to test hypotheses on fixed-size samples. Given y_i a Bernoulli observation, SSP consists in identifying a pair $\langle c, n \rangle$ such that $\Sigma_i^n y_i > c$. A procedure for finding the optimal single sampling for different values of p_0 and p_1 is presented in [151]. The application of SSP to SMC aims at identifying the number n of required traces that guarantees a desired precision.

Sequential Ratio Testing Procedure (SPRT). It is used for sequential testing of stochastic systems. The idea is to collect m observations on-the-fly and to compute a measure f_m defined as:

$$f_m = \prod_{i=1}^{m} \frac{Pr[Y_i = y_i \mid p = p_1]}{Pr[Y_i = y_i \mid p = p_0]} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}}$$

where $d_m = \Sigma_{i=1}^{m} y_i$. The procedure iterates on the triggering of new simulation traces and the update of f_m until SPRT can conclude, in one of these two cases:

1. if $f_m \leq \beta/(1-\alpha)$, then accept H_0,
2. if $f_m \geq (1-\beta)/\alpha$, then accept H_1.

In [151], Younes proposes a logarithmic-based algorithm for *SPRT*, given p_0, p_1, α and β parameters (see [143] for details). When one has to test $\theta \leq 1$ or $\theta \geq 0$, it is however better to use *SSP* (see [27, 151] for details). In general, the precomputed number of traces for *SSP* is higher than the one needed by *SPRT*, but is known to be optimal for the above-mentioned values. We refer the interested reader to [27] for more details about hypothesis testing algorithms and a comparison between *SSP* and *SPRT*.

2.3.2 Probability Estimation

The probability estimation procedure (PE) was proposed in [79] to answer the quantitative question. It enables to estimate the probability p for S to satisfy ϕ. Given a *precision* δ, this procedure computes an estimate value $p' = \Sigma_i^m \, {}^{y_i}/_m$, such that $|p' - p| \leq \delta$ with *confidence* $1 - \alpha$.

To find the required number m of traces, the procedure is based on the *Okamoto lower bound* [1] [126] that is defined as follows:

$$Pr[\ |p' - p| > \delta] \leq 2e^{-2m\delta^2}$$

where m is the number of simulation traces guaranteeing the precision δ with confidence $1 - \alpha$. Consequently, by taking $m \geq \frac{log({}^2/_\alpha)}{2\delta^2}$ one is guaranteed to have $Pr[\ |p' - p| \leq \delta] > 1 - \alpha$.

2.3.3 Rare Events Analysis

In stochastic models, rare events are the ones that have a very low probability to occur during a discrete event simulation. These events are often of crucial importance since they represent potential system failures, errors or any special behavior that are important to consider during system verification. These events are intrinsically covered by the exhaustive scope of techniques such as model checking, but remain often out of the reach of testing and simulation approaches. The classical SMC algorithms also fail to scale since the number of simulations can increase drastically when observing properties with a very low probability p [105]. Indeed, accurately estimating p using Monte Carlo requires to see it at least N times, which would result in collecting simulations $(1/p) * N$. As an illustration, observing 3 times an event that has a 10^{-6} probability to occur would require around 3 million simulations.

Rare events simulation methods such as *importance sampling (IS)* and *importance splitting (IP)* are applied in SMC to reduce the number of simulations and to increase the accuracy of the probability estimates, i.e., with tighter confidence intervals and closer estimated values. In the following, we briefly present IS and IP techniques to analyze rare properties.

[1] Sometimes called the Chernoff bound in the literature.

2.3.3.1 Importance Sampling

Importance sampling is a technique based on altering the model in a way that it makes a rare property more likely to be observed. The model alteration consists in changing the probabilities of the transitions leading to satisfy this property. This can be done by specifying a new transition matrix for the model.

In SMC, the Monte Carlo techniques that are used provide a bound for the absolute error. However, this absolute error becomes less useful for rare events since their probability is very low (at a close order of this absolute error sometimes). Instead the interest is put on the relative error that is computed as the ratio of the standard deviation and the expectation of the estimate. Important sampling techniques provide a bounded relative error whereas this measure may be unbounded in the case of Monte Carlo techniques.

In the general case, the probability of satisfying a property is estimated as the number of the generated traces that satisfy that property divided by the size of the traces sample. Since the property is rare, the proportion of traces satisfying it is quasi null. By biasing the model, this proportion increases. However, the introduced bias has to be taken into account when computing the estimate. Hence, the obtained estimation on the biased model is corrected by multiplying it by $L(\omega) = f(\omega)/f'(\omega)$, where f and f' are the probabilities to generate the trace ω in the original and the biased models, respectively. Hence, given m simulation traces, the estimate is computed as :

$$\bar{\gamma} = \frac{1}{m} \sum_{i=1}^{m} L(\omega_i) \times y_i$$

The effectiveness of IS is conditioned by the quality of the importance sampling distribution [85] for biasing the original model. This quality is evaluated in terms of:

(i) the ability of the biased model to frequently simulate the rare property, and

(ii) the closeness of the distribution of the traces satisfying the property in the biased model to their distribution in the original model, up to a normalizing factor.

In fact, finding a good biasing distribution meeting both requirements (i) and (ii) can be a real challenge. As a solution, the authors in [87] propose an iterative procedure to find an optimal biasing scheme optimizing requirement (ii) based on the Kullback-Leibler divergence measure. However, this method does not always apply in practice, especially for systems modeled as the hierarchical composition of components, or for closed black-box systems. The biasing distribution becomes even more problematic to identify when dealing with properties that require long simulation traces, This comes from the fact that the probability measure over long traces tend to be very small, and hence it makes it more difficult to precisely assess the quality of the proposed bias.

2.3.3.2 Importance Splitting

Importance splitting [89] overcomes the problem of estimating the probability $Pr[S \models \phi]$ of a system S to satisfy a property ϕ representing a rare event. This is done by considering a set of intermediate levels l_i corresponding to less rare properties ϕ_i, s.t., $\phi_n \Rightarrow \phi_{n-1} \Rightarrow \ldots \Rightarrow \phi_0$, where $\phi_n \equiv \phi$ and $\phi_0 \equiv \mathbf{t}$. $Pr[S \models \phi]$ is therefore computed as the product of the conditional probabilities to reach l_i from l_{i-1}, i.e.,

$$\bar{\gamma} = \prod_{i=1}^{n} Pr[S \models \phi_i \mid S \models \phi_{i-1}]$$

In practice, the intermediate levels l_i and associated ϕ_i are defined via a score function given as input. Its role is to evaluate the level reached by a given trace, i.e., the highest index of intermediate property satisfied by that trace.

The algorithm iterates over levels, and for each one, it simulates m trace prefixes among which m_s reach the next level and m_f do not. The conditional probability to reach the next level is thus estimated as the ratio m_s/m. In the next iteration, the simulation of successful prefixes is resumed, while the rest (m_f) are replaced by successful ones selected uniformly.

Importance splitting estimates the probability of a rare property on a user-defined number of traces m. In addition to the computed estimate $\bar{\gamma}$, the (fixed-level [2]) IP algorithm provides a $(1-\alpha)$ confidence interval (CI) defined as follows:

$$CI \in \left[\bar{\gamma}\left(\frac{1}{1+\frac{z_\alpha \times \sigma}{\sqrt{m}}}\right), \bar{\gamma}\left(\frac{1}{1-\frac{z_\alpha \times \sigma}{\sqrt{m}}}\right)\right], \quad \text{with } \sigma^2 \geq \sum_{i=1}^{n} \frac{1-\gamma_i}{\gamma_i} \tag{2.2}$$

where z_α is the $(1-\frac{\alpha}{2})$ quantile of a normal distribution and γ_i is the conditional probability $Pr[S \models \phi_i \mid S \models \phi_{i-1}]$. Note that γ_i cannot be obtained in practice. Hence, σ is estimated by substituting γ_i by its estimate $\bar{\gamma_i}$, in Equation 2.2.

Importance splitting does not require to alter the system model and, hence is applicable to complex models and to black-box systems. It relies on the decomposition of the rare property, that is highly dependent on this latter. The decomposition is easily achievable when the property is a conjunction of boolean formulas[3]. Usually, a natural decomposition can be found for most physical systems, and the levels are identified by nested propositions, that is, $\phi_i \equiv \phi_{i-1} \wedge \varphi_i$. Decomposition rules for temporal operators are provided in [89] and a temporal decomposition is also presented. It states that, for example, a bounded globally is equivalent to verify the property at every step provided the bound is not passed. The main difficulty is to provide a decomposition with the sufficient number of levels and that evenly distributes the conditional probabilities. If not, heuristic decompositions are to favor [42]. Also, the decomposition can sometimes be unachievable, for example, when the property concerns a

[2] The number of levels is defined a priori. See [85] for more information.
[3] Applying DeMorgan laws can help to obtain such a property shape.

rare state (e.g. $\phi \equiv \Diamond failure\_state$), that is, a state that is very unlikely to be reached in the model.

2.4 Statistical Model Checking Tools

Several frameworks exist for modeling and analyzing stochastic systems. In this section, we restrict ourselves to discuss frameworks following a model checking-like procedure. Other methods related to the Queuing Theory and Network Calculus are beyond the scope of this discussion. The considered frameworks generally differ in three points, namely, the expressiveness of their modeling formalism, the proposed analysis techniques, and the properties specification language they offer.

For instance, Uppaal-smc [53, 86] supports *Stochastic Timed Automata* (STAs), which are general models including *Discrete* and *Continuous Time Markov Chains* (DTMCs and CTMCs) for system modeling and *Weighted Metric Temporal Logic* (WMTL) for properties specification. In addition to DTMCs and CTMCs, Prism [101] allows for modeling *Markov Decision Processes* (MDPs) and *Probabilistic Timed Automata* (PTAs). For properties specification, it uses *Probabilistic Computation Tree Logic* (PCTL/PCTL*), *Continuous Stochastic Logic* (CSL), *Linear-time Temporal Logic* (LTL).

Other tools like Vesta [138] support, in addition to DTMCs and CTMCs, algebraic specification languages, i.e., PMaude [100]. Plasma-Lab [88] is a modular statistical model checker that may be extended with external simulators and checkers. Its default configuration accepts discrete-time models specified in the Prism modeling language and properties expressed in Probabilistic Bounded LTL (PBLTL). Ymer [153] is one of the first frameworks to implement hypothesis testing algorithms. It considers GSMPs and CTMCs specified using a dialect of the Prism modeling language, and accepts both PCTL and CSL for requirements specification. Cosmos [20] considers Hybrid Automata Stochastic Logic (HASL) as properties specification language and Generalized Stochastic Petri Net (GSPN) as a modeling language. Nouri et al. [122] proposed the first version of the $\mathcal{S}$BIP framework, that supports the modeling of DTMCs and the specification of bounded LTL (BLTL) properties.

In this thesis, we introduce the $\mathcal{S}$BIP 2.0 framework (see Chapter 5 for more details) to support GSMPs and D/CTMCs using our newly introduced Stochastic Real-Time BIP (SRT-BIP) formalism (see Chapter 3 for more details). As opposed to the aforementioned tools, it allows for using general probability density functions. Except Ymer [153], that is no longer maintained, we are not aware of other tools offering such a general model. From a modeling perspective, it differs from Prism as the latter considers PTAs as the underlying probabilistic and timed model. These PTAs incorporate non-determinism and are generally analyzable using numerical probabilistic model checking. In Prism, only DTMCs and CTMCs are supported for SMC. Our framework is closer to Uppaal-smc and Ymer. The latter relies on GSMPs, whereas we consider GSMPs with fixed-delays events as in [36]. Uppaal-smc provides

a general stochastic timed semantics, however it is only limited to exponential and uniform density functions. These functions can also be combined to express phase-type distribution functions. Furthermore, the stochastic real-time BIP formalism allows for specifying urgency types on systems events. For properties specification, we rely on bounded variants of LTL and MTL.

In terms of statistical analysis capabilities, all the tools mentioned above implement algorithms for hypothesis testing (HT) and for probability estimation (PE), except for Ymer and Vesta that only support hypothesis testing. Regarding rare events analysis, only few tools [20, 88, 117] provide support for importance sampling (IS) and/or importance splitting (IP). For example, Plasma-Lab implements both IS and IP whereas Cosmos only considers IS. In $\mathcal{S}$BIP 2.0, we provide an implementation of IP that we applied on a model of concurrent systems to study fairness in the access to a shared resource.

Table 2.4 summarizes the state-of-the-art SMC tools, along with their distinguishing features, namely, the supported modeling formalism, specification language and analysis capabilities. Please note that tools, such as Prism and Uppaal-smc, are not dedicated SMC tools and, consequently, provide reacher features but they are not enabled when performing an SMC analysis. One can see that $\mathcal{S}$BIP 2.0 contributes to the state-of-the-art by handling the most general (stochastic and timed) modeling formalism, i.e., GSMPs, and by providing the majority of SMC algorithms including support for rare events analysis.

	Modeling formalisms	Specification languages	Statistical analyses			
			HT	PE	IS	IP
Uppaal-smc	STA	WMTL	✓	✓	✗	✗
Prism	DTMC/CTMC	PCTL/CSL/pLTL/PCTL*	✓	✓	✗	✗
Vesta	DTMC/CTMC/PMaude	PCTL/CSL/QuaTEx	✓	✗	✗	✗
Ymer	DTMC/CTMC/GSMP	PCTL/CSL	✓	✗	✗	✗
Cosmos	GSPN	HASL	✓	✓	✓	✗
Plasma-Lab	DTMC	BLTL	✓	✓	✓	✓
$\mathcal{S}$BIP 1.0	DTMC	BLTL	✓	✓	✗	✗
$\mathcal{S}$BIP 2.0	DTMC/CTMC/GSMP	BLTL/BMTL	✓	✓	✗	✓

Table 2.4: Distinguishing features of the state-of-the-art SMC tools

In Chapter 5.4, we present a more detailed comparison of $\mathcal{S}$BIP 2.0, Uppaal-smc and Prism where we consider additional features related to the tools usability. For a more exhaustive survey of the SMC tools, we refer the interested reader to [8].

2.5 Statistical Model Checking in Practice

Recently, statistical model checking has drawn lots of interest in the research community. It is often used as an alternative to time and memory intensive methods such as model checking. In

this section, we give an overview of the recent work on SMC regrouped in different research axes, namely, the usage of SMC for the analysis of various case studies, the extension of SMC to new models and properties, the improvement of SMC algorithms and the complex workflows developed around SMC.

Application domains. SMC has recently been used for the analysis of various case studies in different application domains. In particular, numerous cyber-physical and embedded systems have been analyzed with SMC, such as, a flood monitoring system [41], autonomous driving controllers and Advanced Driver Assistance Systems (ADAS) [60, 72, 133], a moving block railway signaling scenario [22], a train compressor system [136], and an MPEG2 image processing system [122]. This lightweight verification method has also shown its benefits and applicability in the context of software analysis [106], performance evaluation of communication and clock synchronization protocols, such as, FireWire [117], Bluetooth [117] and PTP [122], but also in the study of biological systems [54, 85].

Extension to new models. In addition to the existing implementations for stochastic models such as DTMC, CTMC and GSMP, several works proposed the consideration of complex models that are not trivial to simulate: hybrid systems, models with non-determinism and incomplete stochastic systems.

Applying SMC to hybrid systems is often done by combining simulation tools with support for continuous dynamic behavior, and classical SMC tools that provide formalisms to simulate discrete events. For example, [60] combines Simulink and Cosmos for the study of autonomous driving controllers. Similarly, the authors in [54] use Matlab SimBiology for the modeling of dynamic systems specialized on pharmacodynamics and systems biology that they paired with Uppaal-smc in order to study the differentiation of PC-12 cells.

Stochastic models that support non-determinism are powerful formalisms that allow for modeling uncertainty together with unpredictability. To simulate such models, it is required to define a policy for the non-determinism resolution, called scheduler. In [61], the authors describe the advances in applying SMC to non-deterministic continuous-time formalisms. The described methods usually consist in the computation of upper and lower bounds for the probability of a system to satisfy a property, instead of a single estimate. For example, [51] presents a methodology that serves this purpose in the context of MDPs. This approach relies on exploring the scheduler space through a lightweight sampling of schedulers encoded as uniform pseudo-random number generators (PRNG).

Incomplete stochastic systems are the ones exhibiting an incomplete design or containing components with unspecified behavior. In [14], such systems are modeled as discrete time Markov chains with unknown values (qDTMC) and analyzed using model checking techniques for three-valued temporal logics.

SMC has also been applied in the analysis of fault maintenance trees (FMT). In [136], the authors present a methodology that first consists of a model transformation of FMT models to a network of timed automata from which quantitative metrics are computed using UPPAAL-SMC.

Extension to new logics. SMC tools cover a wide range of temporal logics with both branching-time and linear-time views. However, with the emergence of new systems and needs, these logics become insufficient to handle aspects such as system dynamism and the reasoning over multiple paths. The authors of [134] introduce a new temporal logic, called DynBLTL, to express structural and behavioral properties in dynamic software architectures, that is, systems in which the set of deployed components at a given time cannot be foreseen. HyperPCTL* [148] is a hyper temporal logics to reason about multiple executions at a time. It is an extension of PCTL* with quantifiers over paths.

Improvement of SMC algorithms. To improve the performance of SMC algorithms, several tools provide parallel implementations that rely on distributing the simulation (trace generation) in a client-server architecture. For example, Plasma-Lab provides distributed algorithms that can be run on a dedicated cluster and follows the SMC distribution algorithm described in [151] to avoid the statistical bias due to load balancing.

Another important avenue of investigation involves the consideration of different mathematical bounds relating the statistical (confidence and precision) parameters to the required number of simulations. For example in [90], the authors propose sequential algorithms based on Massart bounds as a means to reduce the required sample size with guaranteed error bounds. They showed that their proposals outperform the standard algorithms implemented in statistical model checkers such as UPPAAL-SMC and PRISM.

SMC in complex workflows. SMC has been recently integrated into complex workflows as a means to obtain quantitative measurements, that are later on used for decision making. For example, [37] addresses the synthesis of correct-by-design concurrent code from a temporal specification. The proposed workflow relies on genetic programming in which SMC defines the fitness function, namely, it scores the candidate generated solutions (concurrent code).

In Chapter 7, we introduce a spiral methodology to model resilient systems with FDIR behavior. In this approach, the system is built through iterative and incremental model transformations that may introduce risks, such as faults in the system or its environment. These risks are assessed through SMC analyses and are mitigated by adding FDIR behavior. The latter behavior is then validated using also SMC. We illustrate this methodology on the safety assessment of a Bridget Rover demonstrator control system.

In Chapter 8, we propose a risk assessment approach for the analysis of the defenses deployed in a corporation's system, and to synthesize defense configurations making sophisticated attacks harder to achieve. The methodology relies on the IO-Def heuristic for the defense synthesis, and

on a combination of machine learning and genetic algorithms for the exploration of sophisticated attack strategies. SMC is used to evaluate these strategies in terms of their attack cost and success probability.

2.6 Conclusion

In this chapter, we presented the commonly used formalisms, in the literature, for the modeling of stochastic and timed systems, and we recalled two specification languages for the formal description of system requirements, namely, LTL and MTL. We also gave an overview of the statistical model checking technique, a state-of-the-art technique for the quantitative analysis of stochastic systems. We described the main SMC algorithms and discussed the existing SMC tools. Finally, we investigated the current trends regarding the utilization of SMC in the literature.

We identified the need for formalisms that allow one to express general probability distributions and to reason about time in a continuous manner, such as GSMPs. A more important observation is the lack of SMC tools that provide at the same time expressive modeling languages, and a wide range of SMC algorithms, specially for the analysis of rare properties.

In this thesis, we introduce a component-based modeling framework with an underlying GSMP formalism, called Stochastic Real-Time BIP (SRT-BIP) and detailed in the next chapter 3. We also implement an SMC tool, called $\mathcal{S}$BIP 2.0 (see Chapter 5), for the analysis of SRT-BIP models. This tool provides the classical SMC analyses (HT and PE) and an implementation of the importance splitting (IP) algorithm for rare events analysis.

Chapter 3

Modeling Component-Based Stochastic Real-Time Systems

In the previous chapter, we reviewed the state of the art in building and analyzing stochastic models. We established the lack of modeling formalisms that meet both the accuracy of a continuous time representation and the flexibility of general probability distributions.

In model-based design, the power of a modeling formalism resides in its expressiveness, namely, its ability to represent complex behaviors and interactions in terms of provided concepts but also regarding its conciseness. This expressiveness is highly desirable when designing real-time cyber-physical systems in which uncertainties in their behaviors and variability in their environments are not negligible. In such systems, functionality and performance are of equal importance. Hence, modeling formalisms that enable to capture both stochastic behavior and timing constraints are essential for building faithful system models at a high-level of abstraction, and to allow for their trustworthy assessment. In this context, we present a new formalism for modeling component-based stochastic real-time systems. This proposed formalism enables for associating system events with timing constraints. Furthermore, in order to express uncertainty regarding events occurrences, it is possible to associate them with general probability density functions.

This chapter is organized in two sections: the first formally presents the proposed modeling formalism, and the second details its implementation. In the first section, we introduce the proposed stochastic real-time BIP (SRT-BIP) formalism for which we present the construction of stochastic real-time components in Section 3.1.1. The composition of these components is introduced in Section 3.1.2 and the simulation of the resulting models is given in Section 3.1.3. We illustrate this formalism and its simulation semantics (Section 3.1.4) on a sender-receiver model before concluding this section with additional modeling features in Section 3.1.5. In the second part of the chapter, we first give an overview of the RT-BIP framework (Section 3.2.1), before providing details of steps taken to implement SRT-BIP in Section 3.2.2.

3.1 Stochastic Real-Time BIP

The *stochastic real-time BIP* framework reconciles the real-time and stochastic extensions of the BIP [24] framework. We recall that BIP has been introduced as a component-based framework where systems are obtained by composition of untimed atomic components with multi-party interactions, and coordinated using dynamic priorities. RT-BIP [7] extended BIP with real-time features and has (dense) real-time semantics based on timed automata concepts [11]. S-BIP [122] extended BIP with stochastic features and has (discrete) stochastic semantics based on Markov chains.

In the newly proposed stochastic real-time BIP formalism, atomic components are defined as timed automata extended with stochastic timing constraints. Composition is performed as in BIP using multi-party interactions, that is, n-ary synchronization among component actions. Priorities are not supported. The underlying semantics is defined as a Generalized Semi-Markov Process (GSMP) [98] where the interpretation of time is dense.

We start by defining the syntax of our model at the level of components and their composition. Then, we present the underlying stochastic simulation semantics.

3.1.1 Stochastic Real-Time Components

Stochastic real-time BIP components are essentially timed automata with urgencies, augmented with a new form of stochastic guards on clocks.

Let Δ be a set of density functions, that is, functions $\rho : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ such that $\int_0^\infty \rho(t)dt = 1$. We denote by $dom(\rho) = \{t \mid \rho(t) \neq 0\}$ the definition domain of ρ, that is, the set of values with a non-zero probability to occur. Let X be a set of clocks. We consider timed constraints (or guards) c^t and stochastic constraints (or guards) c^s on X, defined by:

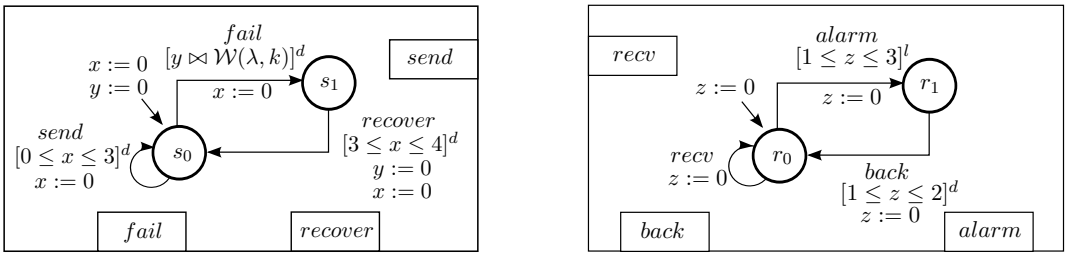
$$c^t ::= true \mid x \sim k \mid x - y \sim k \mid c^t \wedge c^t \qquad\qquad c^s ::= x \bowtie \rho$$

where $x, y \in X$, $k \in \mathbb{R}_{\geq 0}$, $\sim \in \{<, \leq, =, \geq, >\}$ and $\rho \in \Delta$. The meaning of timed constraints is as usual in timed automata [11]. A stochastic constraint $x \bowtie \rho$ holds iff $x \in dom(\rho)$. Nonetheless, in contrast to a timed constraint, it will enforce the specific stochastic distribution ρ on the values of x used to effectively satisfy the constraint, when used as a transition guard.

Definition 3.1.1. A stochastic real-time BIP component B is an extended timed automaton (L, X, P, T, ℓ^0), where L is a finite set of locations, $\ell^0 \in L$ is the initial location, X is a finite set of real-valued clocks, P is a finite set of ports, and T is a finite set of transitions. Every transition is of the form (ℓ, p, g^u, r, ℓ'), denoted for more convenience as $\ell \xrightarrow{p\ g^u\ r} \ell'$, where $\ell, \ell' \in L$ are the source and target locations, $p \in P$ is the triggering port, g^u is a constraint g on X with an urgency $u \in \{lazy, delayable\}$, and $r \subseteq X$ is the set of clocks to be reset on that transition.

The only noticeable difference between our definition of components and the timed automata concerns the meaning to control the progress of time. Usually, timed automata rely on location invariants and/or specific types of locations (e.g., committed) to explicitly constrain the time progress. In our case, we rely on urgency types of transitions with the following intuitive meaning. A *delayable* transition (abbreviated to d) prevents time progress at the upper time bound in g, i.e., time is enabled to progress in the source location at most to that bound. In contrast, a *lazy* transition (abbreviated to l) does not have any impact on the time progress. Such a transition might not be fired at all in spite of the upper bound in g, i.e., time is enabled to progress indefinitely in the source location.

We call a port timed (resp. stochastic) if it appears on transitions with timed (resp. stochastic) constraints. We tacitly restrict to components where every port is either timed or stochastic, but not both. Moreover, we restrict to components that are time-port deterministic, i.e., for a given source location ℓ, time t and port p, only one target location ℓ' can be reached.



(a) Sender (b) Receiver

Figure 3.1: Examples of stochastic real-time BIP components

Example 3.1.1. The **Sender** component shown in Figure 3.1a has control locations s_0, s_1, ports *send, fail, recover* and clocks x, y. This component starts in s_0, where it may send data periodically in a specific time slot, defined through the timed constraint $[0 \le x \le 3]^d$. The **Sender** component may fail, which is modeled by the stochastic port *fail*. The latter is associated with the guard $[y \bowtie \mathcal{W}(\lambda, k)]^d$, where $\mathcal{W}(\lambda, k)$ is the *Weibull* probability density function with parameters λ, k, and $dom(\mathcal{W}) \subseteq \mathbb{R}_{\ge 0}$. This guard indicates that the component fails after some time scheduled according to $\mathcal{W}(\lambda, k)$. After a failure, the component recovers (on transition *recover* after some delay in $[3 \le x \le 4]^d$, where it goes back to s_0 where it can send or fail, as described earlier.

Example 3.1.2. Figure 3.1b shows a second component, namely the **Receiver**, which has control locations r_0, r_1, ports *recv, alarm, back* and a clock z. The **Receiver** starts in location r_0, where z is set to 0. From this initial location, the component either receives some data through the self loop on r_0 labeled by the timed port *recv* (*recv* is a timed port with a guard implicitly set to true, i.e. it might be taken whenever the component is in r_0), in which case z is reset to 0. Or, it may fire the timed transition labeled by the port *alarm* and moves to

location r_1, where the *recv* port is not enabled. One may think of this behavior as a degraded mode, e.g., energy saving with an alarm. Note that the *alarm* port is associated with the timed constraint $[1 \leq z \leq 3]^l$. Since the latter is lazy, the upper bound 3 could be ignored and the *alarm* transition might not be fired. From location r_1, the component takes transition *back* after a delay specified through $[1 \leq z \leq 2]^d$, to r_0 and starts receiving again. This transition is delayable so it must be taken at most at $z = 2$.

3.1.2 Composition of Stochastic Real-Time Components

Stochastic real-time components are composed using multi-party interactions. An interaction represents a strong synchronization (i.e, *rendez-vous*) between transitions located in different components.

Definition 3.1.2. Given n stochastic real-time components $(B_i)_{i=1,\ldots,n}$, with disjoint sets of ports P_i, we define interactions a as subsets of ports from $\cup_{i=1}^{n} P_i$, where:

- $|a \cap P_i| \leq 1$, for every $i = 1, \ldots, n$, i.e., each component B_i participates in a by at most one port P_i,
- a contains either one stochastic port and any number of timed ports with *true* guards, or any number of timed ports with arbitrary timed guards.

Consequently, an interaction is associated with a guard obtained by the conjunction of the guards of the participating ports. An interaction is called stochastic if it contains a stochastic port, and timed otherwise. The timed ports participating in stochastic interactions are restricted to have precisely *true* guards. Intuitively, this ensures that the execution time for such interactions is solely determined by the stochastic port. While this restriction could be avoided at the price of slightly increasing the complexity of the forthcoming stochastic semantics, it has limited impact on the modeling capabilities – we did not encounter the need for other types of interaction beyond the two categories above, in any of the real-life examples we considered.

Definition 3.1.3. A stochastic real-time BIP system is defined as the composition $\gamma(B_1, ..., B_n)$ of n components $B_1, ..., B_n$ with a set of interactions γ.

Example 3.1.3. Consider the composition of the **Sender** and **Receive** components described in Examples 3.1.1 and 3.1.2. The composition is operated through the interaction $\{send, recv\}$, which relates the *send* port of the **Sender** with the port *recv* of the **Receiver**. Figure 3.2 shows the two components and how they interact through the interaction $\{send,\ recv\}$. The nominal behavior is when the **Sender** sends data to the **Receiver** through interaction $\{send,recv\}$. However, the former may fail when interaction $\{fail\}$ takes place, which is potentially detected by the **Receiver**. The latter emits an alarm and switches into a non-receiving mode by

executing interaction $\{alarm\}$. The two components resume their normal activity after some delay, through interactions $\{recover\}$ and $\{back\}$ respectively.

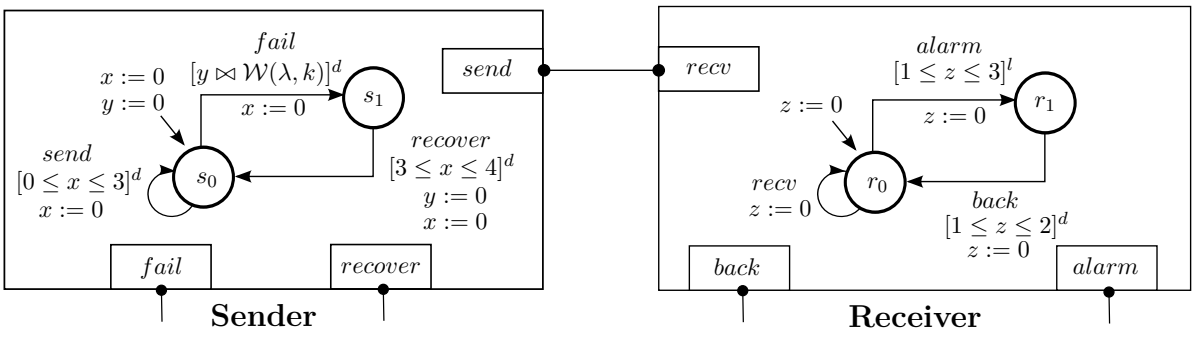


Figure 3.2: Composition of two stochastic real-time BIP components

We further introduce some additional notations for defining the underlying operational semantics of a stochastic real-time BIP system. Let $\gamma(B_1, ..., B_n)$ be a stochastic real-time BIP system, where $B_{i=1,...,n} = (L_i, X_i, P_i, T_i, \ell_i^0)$.

Definition 3.1.4. We define system states as couples $s = (\vec{\ell}, \vec{v})$, where $\vec{\ell} = (\ell_1, ..., \ell_n) \in L_1 \times ... \times L_n$ is a global location, and $\vec{v} : \cup_{i=1}^n X_i \to \mathbb{R}_{\ge 0}$ is a vector of clocks valuations.

We further define *d-succ*$((\vec{\ell}, \vec{v}), a)$ as the **discrete successor** (partial) function that computes the successor of a state $(\vec{\ell}, \vec{v})$ when taking an interaction a. Let $a = \{p_i\}_{i \in \mathcal{I}}$ such that $\mathcal{I}$ denotes the set of indices of the components participating in a, and p_i their participating port. We define *d-succ*$((\vec{\ell}, \vec{v}), a) = (\vec{\ell'}, \vec{v'})$ whenever:

- for every $i \in \mathcal{I}$, there exists an enabled transition $\ell_i \xrightarrow{p_i \ g_i^{u_i} \ r_i} \ell_i'$ of B_i, that is:
  - either g_i is a *timed constraint* which is satisfied by the valuations of the concerned clocks, i.e. $\vec{v}_{|X_i} \models g_i$, where $\vec{v}_{|X_i}$ is the projection of the set of clocks on the subset of clocks that are used in g_i,
  - or g_i is a *stochastic constraint* $x \bowtie \rho$ and $\vec{v}(x) \in dom(\rho)$.
- all these transitions are simultaneously executed, that is, clocks are reset $\vec{v'}(x) = 0$ for all $x \in \cup_{i \in \mathcal{I}} r_i$ and stay unchanged, $\vec{v'}(x) = \vec{v}(x)$ otherwise.
- all the components that do not participate in a remain unchanged, that is, for every $j \notin \mathcal{I}$ it holds $\ell_j = \ell_j'$.

If no successor by interaction a exists at state $(\vec{\ell}, \vec{v})$, we define *d-succ*$((\vec{\ell}, \vec{v}), a) = \bot$.

We define *t-succ*$((\vec{\ell}, \vec{v}), t)$ as the **time successor** function that computes the successor of a state $(\vec{\ell}, \vec{v})$ for a time progress of t. It is a total function and is defined as *t-succ*$((\vec{\ell}, \vec{v}), t) = (\vec{\ell}, \vec{v} + t)$. That is, it increases all the clocks in $\vec{v}$ by the amount of time t.

Finally, we define the function $succ((\vec{\ell},\vec{v}),t,a)$ that computes the **successor** of a state $(\vec{\ell},\vec{v})$ when taking an interaction a after the time progress of t, which is a partial function defined as the composition $succ((\vec{\ell},\vec{v}),t,a) = d\text{-}succ(t\text{-}succ((\vec{\ell},\vec{v}),t),a)$.

Definition 3.1.5. The operational semantics of a stochastic real-time BIP system is defined as the timed transition system $\mathcal{T} = (S, s_0, \rightarrow_S)$ where

- S is the set of states, and s_0 is the initial state,
- $\rightarrow_S \subseteq S \times (\gamma \cup \mathbb{R}_{\geq 0}) \times S$ are transitions defined by the two rules

$$\textsc{Discrete}\ \frac{d\text{-}succ((\vec{\ell},\vec{v}),a) = (\vec{\ell'},\vec{v'})}{(\vec{\ell},\vec{v}) \xrightarrow{a}_S (\vec{\ell'},\vec{v'})}$$

$$\textsc{Time}\ \frac{t > 0, \qquad \forall a\ \textit{delayable}.\ (\exists t'.\ succ((\vec{\ell},\vec{v}),t',a) \neq \bot) \Rightarrow (\exists t'' \geq t.\ succ((\vec{\ell},\vec{v}),t'',a) \neq \bot)}{(\vec{\ell},\vec{v}) \xrightarrow{t}_S (\vec{\ell},\vec{v}+t)}$$

That is, according to the first rule, an enabled transition can be fired at the current instant and the state updated. According to the second rule, time can progress as long as all enabled *delayable* transitions remain enabled. Note that a run of $\mathcal{T}$ is an infinite sequence $\sigma = s_0 s_1 s_2 \cdots$, such that $s_i \xrightarrow{t_i,a_i}_S s_{i+1}$, for some $t_i \in \mathbb{R}_{\geq 0}$ and $a_i \in \gamma$, for all $i \geq 0$.

3.1.3 Stochastic Simulation Semantics

So far, we introduced the concepts of stochastic real-time BIP components and presented their composition from an operational viewpoint. In this section, we show how this model embraces a stochastic semantics in terms of a Generalized Semi-Markov Process.

GSMPs are stochastic process descriptions for a large class of discrete-event systems. A configuration of the GSMP is usually determined by a state and a set of active events, every one associated with a *remaining lifetime*, i.e. the amount of time during which it remains active. The choice of the event to be executed follows a *race policy*, which consists of selecting the event having the smallest remaining lifetime. The execution itself occurs when the remaining lifetime reaches 0 and triggers a state change and moreover, an update of the set of active events. That is, several events could become inactive and therefore removed from the set, or could become active, and therefore added to the set. In the latter case, the remaining lifetime is randomly chosen according to a (usually dense support) probability density function associated to the event.

The stochastic real-time BIP semantics follow the same intuition by considering interactions defined at composition as the GSMP events. Moreover, the associated probability density functions are obtained from the explicit density functions used in stochastic guards of stochastic interactions or by some default densities (uniform or exponential) in the case of timed interactions.

In the remainder of this section we introduce the stochastic simulation algorithm and define precisely the different densities and sampling procedures.

3.1.3.1 Stochastic Simulation Algorithm

As for a GSMP, our simulation keeps track of the remaining lifetime of each interaction in order to implement the race policy. To this end, we define configurations as follows.

Definition 3.1.6. We define a configuration z as a couple $\langle(\vec{\ell}, \vec{v}), \vec{w}\rangle$, where $(\vec{\ell}, \vec{v})$ is a state (as in Definition 3.1.4) and $\vec{w} : \gamma \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ is a vector of remaining lifetime of interactions.

For an interaction a, the value $\vec{w}(a)$ represents the remaining lifetime at the current global location $\vec{\ell}$ and a is said to be active if $\vec{w}(a) < \infty$. Moreover, we need to identify dependencies between interactions. As explained for the GSMPs, the execution of an interaction might activate and/or deactivate other interactions. In the case of stochastic real-time BIP we consider that an interaction a has an impact on another interaction b, denoted by $a \triangleright b$, iff the guard of b changes due to the execution of a, that is, either because b has different timing constraints at the location(s) reached after executing a, or because a resets some clocks explicitly involved in one of the constraints of b (before or after executing a). It is worth mentioning that, according to this definition, any interaction b activated or deactivated due to the execution of a is considered to be impacted by a.

Algorithm 1 below presents the stochastic execution dynamics of a stochastic real-time BIP system. The algorithm shows how to move from one configuration $z_k = \langle(\vec{\ell}_k, \vec{v}_k), \vec{w}_k\rangle$ to another $z_{k+1} = \langle(\vec{\ell}_{k+1}, \vec{v}_{k+1}), \vec{w}_{k+1}\rangle$, starting from an initial configuration $z_0 = \langle(\vec{\ell}_0, \vec{v}_0), \vec{w}_0\rangle$. The first part of the algorithm computes this initial configuration as a vector $\vec{\ell}_0$ of the initial locations of components B_i of the system, a vector of initial valuations of the clocks $\vec{v}_0$, and a vector of initial remaining lifetimes of interactions $\vec{w}_0$. In the latter, each interaction b which is not enabled at the initial state, i.e., $\forall t.succ((\vec{\ell}_0, \vec{0}), t, b) = \bot$, is assigned an infinite remaining lifetime, each enabled interaction b, is assigned a remaining lifetime through the sampling function $\mathcal{R}_b((\vec{\ell}_0, \vec{v}_0))$, which will be formally defined in the next sub-section.

The main loop of the algorithm is executed while there are still active interactions in $\vec{w}_k$. Each iteration determines the next configuration z_{k+1} from the current one z_k. Given the current configuration, active interactions race to determine which one will be executed, i.e., the one with the minimum remaining lifetime in $\vec{w}_k$. Given the winning interaction a_k and its remaining lifetime t_k, we compute the successor state $(\vec{\ell}_{k+1}, \vec{v}_{k+1})$ by using the *succ* function defined earlier. Finally, the remaining lifetimes of interactions are updated in this new state. Three cases can be distinguished for updating the vector of remaining lifetimes $\vec{w}_{k+1}$.

1. if interaction a_k has no impact on b, then the remaining lifetime of b at $(\vec{\ell}_{k+1}, \vec{v}_{k+1})$ is its remaining lifetime at $(\vec{\ell}_k, \vec{v}_k)$ decreased by t_k, i.e., the amount of time progress,

input : $\gamma(B_1, \ldots, B_n)$, where $B_{i=1,\ldots,n} = (L_i, X_i, P_i, T_i, \ell_i^0)$
output : An execution trace

`/* Compute the initial state` $(\vec{\ell}_0, \vec{v}_0)$ `*/`
$\vec{\ell}_0 := (\ell_1^0, \ldots, \ell_n^0)$ `/*` ℓ_i^0 `is the initial location of` B_i `*/`
$\vec{v}_0 := \vec{0}$ `/*` $\vec{0}$ `is the vector of initial clocks valuations */`

`/* Compute the initial remaining lifetime */`

foreach *interaction* $b \in \gamma$ **do** $\vec{w}_0(b) := \begin{cases} \infty & \text{if } \forall t.\ succ((\vec{\ell}_0, \vec{0}), t, b) = \bot \\ \mathcal{R}_b((\vec{\ell}_0, \vec{v}_0)) & \text{if } \exists t.\ succ((\vec{\ell}_0, \vec{0}), t, b) \neq \bot \end{cases}$

`/* Compute the initial configuration` z_0 `*/`
$z_0 := \langle(\vec{\ell}_0, \vec{v}_0), \vec{w}_0\rangle$
$k := 0$

`/* Main loop: computes` z_{k+1} `from` z_k `*/`
while $\exists b \in \gamma.\ \vec{w}_k(b) \neq \infty$ **do**
  `/* Race: determines the interaction` a_k `to execute */`
  Let $t_k = min_{a \in \gamma} \vec{w}_k(a)$, and let a_k be the associated min event
  `/* Update successor state */`
  $(\vec{\ell}_{k+1}, \vec{v}_{k+1}) := succ((\vec{\ell}_k, \vec{v}_k), t_k, a_k)$
  `/* Update remaining lifetime for interactions */`
  foreach *interaction* $b \in \gamma$ **do**
  $\vec{w}_{k+1}(b) := \begin{cases} \vec{w}_k(b) - t_k & \text{if } \neg(a_k \triangleright b) \\ \infty & \text{if } a_k \triangleright b \text{ and } \forall t.\ succ((\vec{\ell}_{k+1}, \vec{v}_{k+1}), t, b) = \bot \\ \mathcal{R}_b((\vec{\ell}_{k+1}, \vec{v}_{k+1})) & \text{if } a_k \triangleright b \text{ and } \exists t.\ succ((\vec{\ell}_{k+1}, \vec{v}_{k+1}), t, b) \neq \bot \end{cases}$
  `/* Compute the next configuration` z_{k+1} `*/`
  $z_{k+1} := \langle(\vec{\ell}_{k+1}, \vec{v}_{k+1}), \vec{w}_{k+1}\rangle$
  $k := k + 1$
end

Algorithm 1: Stochastic Simulation Algorithm

2. if interaction a_k has an impact on b, and b is not active at $(\vec{\ell}_{k+1}, \vec{v}_{k+1})$, then $\vec{w}_{k+1}(b)$ is set to ∞, that is, will not race in this new configuration,

3. if interaction a_k has an impact on b, and b is active at $(\vec{\ell}_{k+1}, \vec{v}_{k+1})$, then its remaining lifetime is sampled according to the function $\mathcal{R}_b((\vec{\ell}_{k+1}, \vec{v}_{k+1}))$.

Note that the enumerated settings include the case where new interactions are becoming active at $(\vec{\ell}_{k+1}, \vec{v}_{k+1})$. The reason is that any such interaction b is seen to be impacted by a_k as explained earlier.

The rational of distinguishing the interactions impacted by the executed interaction a_k and the non impacted ones, regarding the sampling operation can be explained as follows. The former interactions involve ports of a shared component (i.e. a component involved in two or more interactions with other components), thus by executing a_k the system state changes (potentially, the location of the shared component changes, some clocks are reset, etc.). These are really seen as new interactions, hence, they need to be re-sampled. From the non-impacted interactions point of view, nothing has changed but time has evolved, so we do not need to re-schedule them (by re-sampling) but just to update their remaining lifetime accordingly. This distinction can be seen as an optimization that avoids systematic re-sampling.

3.1.3.2 The Time Sampling Procedure

The sampling function $\mathcal{R}_b((\vec{\ell}_{k+1}, \vec{v}_{k+1}))$ used in Algorithm 1 computes the remaining lifetime for each interaction b when entering the state $(\vec{\ell}_{k+1}, \vec{v}_{k+1})$ by taking interaction a_k from the state $(\vec{\ell}_k, \vec{v}_k)$. It depends on the type of interaction b, that is timed or stochastic, and *delayable* or *lazy*. For the sake of simplicity, we define the sampling procedure in two phases: (1) in this subsection, we define the sampling procedure without detailing the underlying probability density function, (2) in the next subsection, we will define how the density function is actually computed.

First let us consider the partitioning of interactions in a configuration $\langle(\vec{\ell}, \vec{v}), \vec{w}\rangle$ as either *fixed-delay* interactions, denoted $\mathcal{F}$ or *variable-delay* interactions, denoted $\mathcal{V}$. *Fixed-delay* interactions are induced by timed interactions having equality on their associated timed constraints (also potentially by stochastic interactions following them), whereas *variable-delay* interactions are timed or stochastic interactions with an interval of possible remaining lifetime values.

$$\begin{aligned} \mathcal{F} &= \{a \in \gamma \mid \vec{w}(a) \neq \infty,\ \exists! t.\ succ((\vec{\ell}, \vec{v}), t, a) \neq \bot\} \\ \mathcal{V} &= \{a \in \gamma \mid \vec{w}(a) \neq \infty\} \setminus \mathcal{F} \end{aligned}$$

Based on this partitioning, the time sampling function for an interaction b when entering a new state $(\vec{\ell}, \vec{v})$ is defined as follows.

$$\mathcal{R}_b((\vec{\ell}, \vec{v})) = \begin{cases} t & \text{if } b \in \mathcal{F} \text{ is delayable \& enabled at } t \\ \text{if } X \text{ then } t \text{ else } \infty & \text{if } b \in \mathcal{F} \text{ is lazy \& enabled at } t \\ \mathrm{F}^{-1}_{\tilde{\rho}_b}(Y) & \text{if } b \in \mathcal{V} \text{ is delayable} \\ \text{if } X \text{ then } \mathrm{F}^{-1}_{\tilde{\rho}_b}(Y) \text{ else } \infty & \text{if } b \in \mathcal{V} \text{ is lazy} \end{cases} \tag{3.1}$$

where $X \sim \mathcal{B}(\frac{1}{2})$ is a random Bernoulli variable over $\{\textit{true, false}\}$, i.e., *true* and *false* have a probability $\frac{1}{2}$, $Y \sim \mathcal{U}(0,1)$ is a random variable with standard uniform distribution, and $\mathrm{F}^{-1}_{\tilde{\rho}_b}$ is the inverse *cumulative distribution function* (CDF) of the probability density function $\tilde{\rho}_b$ associated to b at $(\vec{\ell}, \vec{v})$.

For *fixed-delay* interactions ($b \in \mathcal{F}$), if b is delayable, the sampling function $\mathcal{R}_b((\vec{\ell}, \vec{v}))$ returns the single time value t that satisfies the guard g_b. Whereas, if b is lazy, a discrete choice according to X is first performed to determine whether b will be considered and scheduled to t, or not considered and scheduled to ∞.

The sampling function in the case of *variable-delay* interactions ($b \in \mathcal{V}$) is slightly more involved since it requires choosing from an interval of time values. The same treatment with respect to the urgency types of interactions is performed i.e., a discrete choice on X is used to consider a lazy interaction or not. The time value is obtained by sampling according to the probability distribution $\tilde{\rho}_b$. Technically, this corresponds to computing the inverse CDF ($\mathrm{F}^{-1}_{\tilde{\rho}_b}$) on a random value Y uniformly distributed in the interval $[0, 1]$. The detailed definition of the probability density function $\tilde{\rho}_b$, in the case of timed and stochastic interactions, is given below.

3.1.3.3 Density Functions for Variable-delay Interactions

In this subsection, we define the density function $\tilde{\rho}_b$ associated with a *variable-delay* interaction b at a state $(\vec{\ell}, \vec{v})$. We recall that such an interaction may be either timed or stochastic. For the former case, since no density function is explicitly specified on the associated guard, the function $\tilde{\rho}_b$ is implicitly obtained from a uniform or exponential density function. For the latter case, the function $\tilde{\rho}_b$ is obtained from the density function associated with the guard of b.

$$\tilde{\rho}_b(t) = \begin{cases} \frac{1}{u-l} \cdot \mathbf{1}[l \le t \le u] & \text{if } b \text{ is timed with guard } g_b \text{ true on } [l,u] \\ & \text{that is, } \vec{v}_b + t \models g_b \text{ iff } t \in [l,u] \\ \lambda e^{-\lambda(t-l)} \cdot \mathbf{1}[l \le t] & \text{if } b \text{ is timed with guard } g_b \text{ true on } [l,\infty) \\ & \text{that is, } \vec{v}_b + t \models g_b \text{ iff } t \in [l,\infty) \\ \frac{\rho(\vec{v}_b(x)+t)}{\int_{\vec{v}_b(x)}^{\infty} \rho(s)ds} & \text{if } b \text{ is stochastic with guard } [x \bowtie \rho] \end{cases}$$

where $\mathbf{1}[t \in D]$ is the identity function, which gives 1 if $t \in D$, and 0 otherwise.

The first two cases correspond to timed interactions. We distinguish two situations in this setting, **(i)** when interaction b is timed and has a right-bounded guard, i.e., u is finite, the sampling in the interval $[l, u]$ is done uniformly, **(ii)** when the timed constraint is of the form $[l, \infty)$, the sampling is done according to the exponential density function. In both scenarios, the time t to sample must be within the interval specified by the time constraint. Stated differently, the current valuations of clocks in $\vec{v}_b$ increased by the sampled time t must satisfy the guard g_b.

Remark that for **(i)** and **(ii)**, i.e., for timed interactions, the time constraint g_b may involve several clocks (potentially because of the composition, recall that an interaction involves several ports). Moreover, when entering a new state $(\vec{\ell}, \vec{v})$, the concerned clocks $\vec{v}_b$ may have valuations different from 0. Hence, the computation of the final time bounds u, l in which the time t will be sampled, for b, either uniformly or exponentially is more involved. Generally, given a guard g_b of the form $\bigwedge_i (l_i \leq x_i \leq u_i)$ and the valuations $\vec{v}_b(x_i)$, the bounds of the sampling interval of b are actually computed as $l = max(l_i - \vec{v}(x_i))$ and $u = min(u_i - \vec{v}_b(x_i))$ as illustrated in the example below (3.1.4).

Example 3.1.4. The situation depicted in Figure 3.3 shows a global state of the system $(\vec{\ell}, \vec{v})$, where the valuations of clocks x and y are respectively $\vec{v}(x) = 1$ and $\vec{v}(y) = 2$, and the time constraint is $[(2 \leq x \leq 6) \wedge (2 \leq y \leq 5)]^d$. For the clock x, the remaining lifetime interval t_x is computed as $(2-1) = 1 \leq t_x \leq (6-1) = 5$. Similarly, for y, $(2-2) = 0 \leq t_y \leq (5-2) = 3$. Hence, the obtained sampling interval $[l, u]$ is $max(1, 0) \leq t \leq min(5, 3)$. Note that guards of the form $x - y \sim k$ have the same interpretation since the difference $x - y$ is constant over time as both clocks evolve identically.

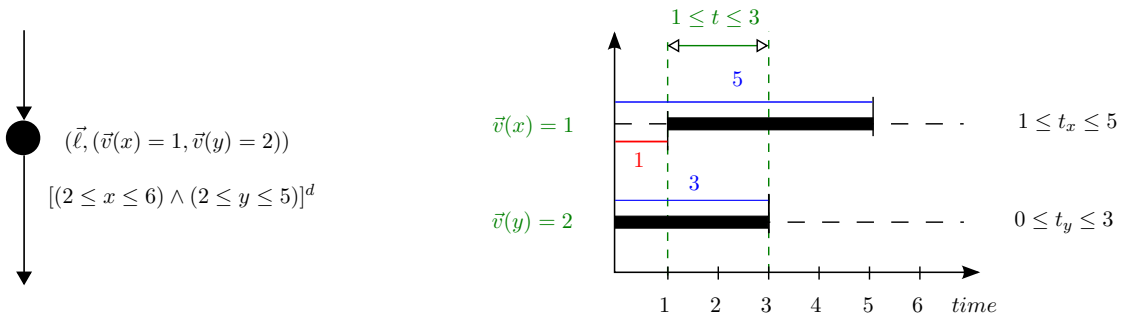


Figure 3.3: Computation of upper and lower bounds in the case of timed interactions; $l = max(1, 0) = 1$ and $u = min(5, 3) = 3$, hence the sampling will be uniform in $[1, 3]$.

The third case in the definition of $\tilde{\rho}_b(t)$ concerns variable-delay interactions obtained from a stochastic interaction b with a guard $[x \bowtie \rho]^d$. In this scenario, the sampling is done in $dom(\rho)$ according to a potentially shifted and normalized density function. This transformed function takes into account the case where the clock valuation of x, i.e., $\vec{v}(x)$ is not 0 when entering the state $(\vec{\ell}, \vec{v})$. Below is a concrete illustration of the transformation.

Example 3.1.5. The transformation is illustrated in Figure 3.4, where $\rho(t)$ is a Normal density function and $\vec{v}(x) = 1$. The function is first shifted to the current valuation of x, i.e., $\rho(1+t)$. Since this shifted function is no longer a proper probability density function, i.e., its area is lower than 1, it is normalized, i.e., divided by $\int_1^\infty \rho(s)\mathrm{d}s$.

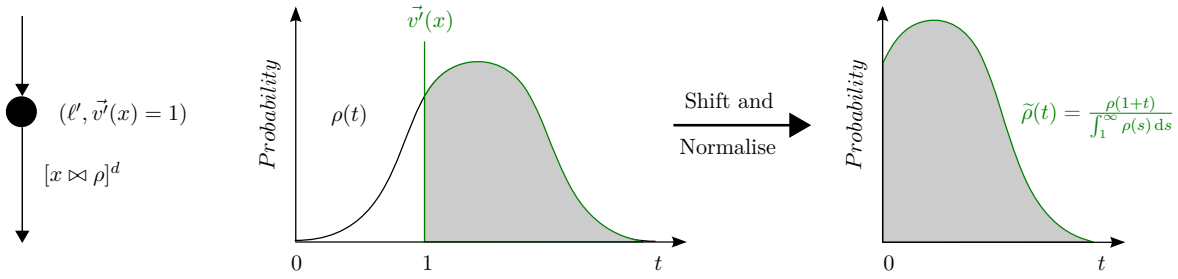


Figure 3.4: Shifting and normalizing a Normal density function in the case of stochastic interactions

3.1.4 An Example of Stochastic Simulation

In Figure 3.5, we illustrate the stochastic semantics on Example 3.1.3 of the Sender-Receiver. We actually show a specific execution trace by sampling particular time values in each configuration. In this figure, configurations are of the form $\langle (s_i, r_j), (\vec{v}(x), \vec{v}(y), \vec{v}(z)), (\vec{w}(\{send, recv\}), \vec{w}(\{fail\}), \vec{w}\{recover\}, \vec{w}(\{alarm\}), \vec{w}(\{back\}))\rangle$. In each configuration, newly sampled remaining lifetimes are denoted by a box $\boxed{\mathrm{t}}$, and updated remaining lifetimes are either ∞ or underlined $\underline{t}$ according to the definition of the sampling function $\mathcal{R}_b$. To make the example readable, we only show the discrete transition, i.e. induced by the uniform choice over lazy interactions.

In this example, there are two possible initial configurations corresponding to the choice of considering the lazy interaction *alarm* $\langle (s_0, r_0), (0,0,0), (\boxed{1.3}, \boxed{7.4}, \infty, \boxed{2.8}, \infty)\rangle$ or not $\langle (s_0, r_0), (0,0,0), (\boxed{1.5}, \boxed{6}, \infty, \infty, \infty\rangle$ at the beginning. Both configurations have the same global location and clocks valuation $(s_0, r_0), (0,0,0)$, but differ in their sampling of the remaining lifetime of the initially racing interactions, namely $\{send, recv\}, \{fail\}$ and $\{alarm\}$. In one case (left branch), we have $(1.5, 6, \infty, \infty, \infty)$, i.e., *alarm* is scheduled at ∞, while in the second case (right branch), $(1.3, 7.4, \infty, 2.8, \infty)$, i.e., *alarm* is scheduled at 2.8. Note that the probability to start in one of these configurations corresponds to the probability to get the sampled remaining lifetime values weighted by a half. For the sake of simplicity, we preferred to detail only one branch of the execution trace, i.e., the one on the left in Figure 3.5. The complete execution trace shown in the example consists of the sequence of transitions $\xrightarrow{1.5,\{send,recv\}} \xrightarrow{3,\{send,recv\}} \xrightarrow{1.5,\{fail\}} \xrightarrow{1.5,\{alarm\}} \xrightarrow{1.3,\{back\}} \xrightarrow{0.7,\{recover\}} \xrightarrow{0.5,\{send,recv\}}$, which corresponds to two send-receive operations, followed by a fail of the **Sender**, which is detected by the **Receiver** that emits an alarm and moves to a degraded mode then gets back to

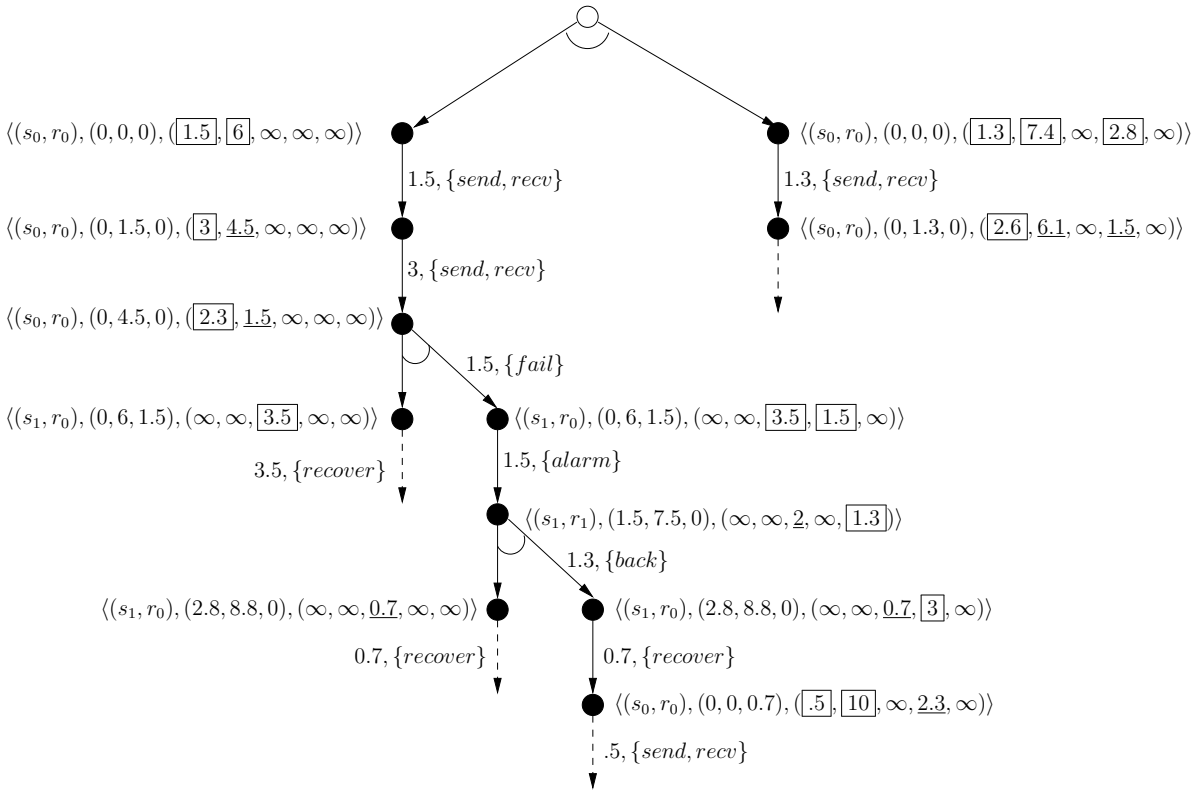


Figure 3.5: Illustration of the stochastic simulation semantics on Example 3.1.3

its normal working mode, followed by a recover of the **Sender**, and finally another send-receive operation.

3.1.5 Additional Modeling Features

The proposed formalism can be enriched with the usual cost/reward structures and data variables. These structures allow for assigning a score to execution traces, such as to model energy consumption or failure time. In addition, we show how we can model *eager* interactions. Indeed, *delayable* and *lazy* urgencies are not always enough to specify real-time systems with complex timing behavior, especially when an urgent transition must be taken as soon as possible.

3.1.5.1 Handling Cost/Reward Structures

A cost/reward structure in this model can be obtained in a straightforward manner by adding a data-structure in our simulation algorithm and associate it with states and interactions. Since in our model we know how long the system remains in each state (as we keep track of the remaining lifetimes of interactions), and which interactions are executed; we can, by specifying unit cost/reward for states and interactions, compute the global cost for each execution trace of the system. For instance, in the previous example, assume that we have this modeling feature

and that we specified a cost of a fail to be 2, and the cost of remaining in a failure mode as 1 per time unit. The total cost of the fragment of the execution trace shown in Figure 3.5 will be 5.5. That is, 2 (the fail interaction cost) plus $[(1.5 + 1.3 + 0.7) \times 1]$ (the total time spent by the **Sender** component in a failure mode, i.e., from executing interaction *fail* to executing interaction *recover*.

3.1.5.2 Considering Eager Interactions

An *eager* interaction is an urgent action that must be taken whenever it becomes enabled. The consideration of the *eager* urgency requires to define the outcome of the sampling procedure and to adapt the stochastic simulation algorithm.

Extending the time sampling function. Sampling a remaining lifetime for *eager* interactions (abbreviated ε) is straightforward. Being at the state $(\vec{\ell}_k, \vec{v}_k)$, the sampling function $\mathcal{R}_a((\vec{\ell}_k, \vec{v}_k))$ determines the remaining lifetime for the *eager* interaction a as the earliest time value, depending on its associated time constraint. Equation 3.2 below extends the time sampling procedure introduced in Equation 3.1 with sampling rules for *eager* interactions.

$$\mathcal{R}_a((\vec{\ell}, \vec{v})) = \begin{cases} c, & \text{if } a \in \mathcal{F} \text{ is eager \& enabled at } c \\ l, & \text{if } a \in \mathcal{V} \text{ is eager \& enabled for } t \geq l \\ \perp, & \text{if } a \in \mathcal{V} \text{ is eager \& enabled for } t > l \end{cases} \tag{3.2}$$

For fixed-delays interactions, the sampled value is obviously the fixed delay c, while for variable-delays interactions, the smallest possible value corresponds to the lower bound of the time constraint. Please note that this eager interaction must not be assigned a clock constraint of the form $t > l$. Indeed this class of constraints is ambiguous on dense time domains and the sampling algorithm fails to precisely determine the earliest date for the interaction. We consider this problem as a modeling error that can be corrected by introducing a step variable ϵ to identify the closest time value t' that satisfies $t > l$. Hence, this clock constraint is equivalent to $t \geq l + \epsilon$ and the selected remaining lifetime is $\mathcal{R}_a((\vec{\ell}, \vec{v})) = l + \epsilon$. Furthermore, we allow *eager* interactions to be only timed, not stochastic.

Updating the Stochastic Simulation Algorithm. From the simulation point of view, *eager* interactions compete with *delayable* and *lazy* interactions following the same race policy, namely, the one with the minimal remaining lifetime t_{min} is selected for execution. Nonetheless, considering *eager* interactions increases the likelihood of ending with interactions assigned the same value t_{min}, i.e., the set of candidate interactions for execution would be $\mathcal{A} = \{a \in \gamma \mid \vec{w}(a) = min_{a \in \gamma} \vec{w}(a) = t_{min}\}$, where $|\mathcal{A}| > 1$. For instance, two *eager* interactions enabled from the same state with the same lower bound would lead to this situation. A straightforward solution for such situations would be to uniformly select one interaction among them. A more

realistic choice though, is to allow for explicitly weighting these interactions. That is, at a given state, the probability to execute one of the candidate interactions is specified by a probability mass function over $\mathcal{A}$.

To model this function, each interaction is attached with a numerical value representing its weight. Let $W : \gamma \longrightarrow \mathbb{N}$ be the weight function that maps each interaction $a \in \gamma$ to its weight $W(a)$. Hence, the probability to select interaction a among the set of candidates $\mathcal{A}$ is given by the following equation:

$$\vartheta_{\mathcal{A}}(a) = \begin{cases} 0, & \text{if } a \notin \mathcal{A} \\ \frac{W(a)}{\sum\limits_{b \in \mathcal{A}} W(b)}, & \text{if } a \in \mathcal{A} \end{cases}$$

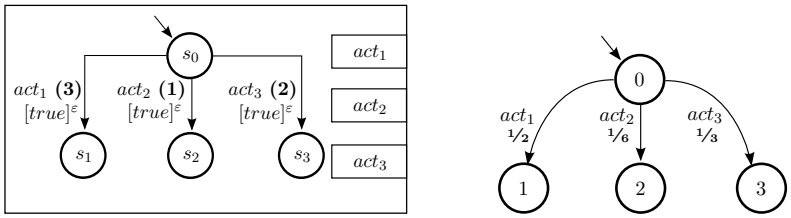
Please note that the candidate interactions are equally likely to be observed when assigned the same weight. In that case, $\vartheta_{\mathcal{A}}$ becomes a discrete uniform distribution with probability $1/|\mathcal{A}|$.

In Algorithm 1, we described the stochastic evolution of an SRT-BIP model. As a reminder, the first step is the initialization phase, in which the initial configuration is computed. Then, the main loop identifies the interaction to execute based on a race policy and computes the next configuration accordingly. More specifically, the race identifies the interaction a_k corresponding to the smallest remaining lifetime t_k. Algorithm 2 shows how to update Algorithm 1 in order to cope with the situation discussed above.

```
/* The initialization remains unchanged */
...
/* Main loop: computes z_{k+1} from z_k */
while ∃b ∈ γ. w⃗_k(b) ≠ ∞ do
    /* Race: determines the interaction a_k to execute */
    /* Replace this line :
       Let t_k = min_{a∈γ} w⃗_k(a), and let a_k be the associated min event */
    /* By these three lines */
    A_k = {a ∈ γ | w⃗_k(a) = min_{a∈γ} w⃗_k(a)}
    a_k = F^{-1}_{ϑ_{A_k}}(Y)
    t_k = w⃗(a_k)
    /* The rest of the computation remains unchanged */
    ...
end
```

Algorithm 2: Updated Stochastic Simulation Algorithm

In Algorithm 2, $Y \sim \mathcal{U}(0,1)$ is a random variable with standard uniform distribution, and $\mathrm{F}^{-1}_{\vartheta_{\mathcal{A}_k}}$ is the inverse CDF of the probability mass function $\vartheta_{\mathcal{A}_k}$ associated with the set of candidate interactions $\mathcal{A}_k$.



(a) SRT-BIP component (b) Underlying DTMC

Figure 3.6: Example of an SRT-BIP component with discrete probabilities

It is worth mentioning that the updated simulation algorithm allows for modeling both stochastic time behavior and transition with discrete probabilities. The latter can be represented as weighted *eager* transitions without time constraints, where the weights are proportional to the probability of the corresponding transition. Figure 3.6 shows a model with a discrete choice at the initial state s_0 between three transitions labeled by act_1, act_2 and act_3. These *eager* transitions are assigned with respective weights of 3, 1 and 2, that correspond to discrete probabilities of ½, ⅙ and ⅓, respectively. These probabilities are computed by the function $\vartheta_{\mathcal{A}}$, with a set of candidate interactions $\mathcal{A} = \{act_1, act_2, act_3\}$.

3.2 Implementation of SRT-BIP

In this section, we present an implementation of the proposed SRT-BIP formalism for modeling component-based systems with time and probabilities. This implementation plays a major role in, on the one hand, validating the semantics described in the previous section, and on the other hand, providing a mean to build and simulate such SRT-BIP models which is essential in the context of verification.

We consider the RT-BIP framework [7] as a starting point of our implementation. In the current implementation, we fully support the semantics introduced in the previous section, except for the weighting mechanism to explicitly model discrete probabilities. In the following, we start by giving an overview of the RT-BIP framework, then we detail the steps taken to implement it.

3.2.1 Overview of the RT-BIP Framework

RT-BIP is a framework for the modeling of real-time systems. It extends the BIP framework [24] by enriching the language with real-time modeling features and by providing a real-time execution engine. Figure 3.7 illustrates the tool-chain and workflow for the RT-BIP framework. This

process exhibits the main elements of the framework, namely, the modeling language, the compiler and the engine.

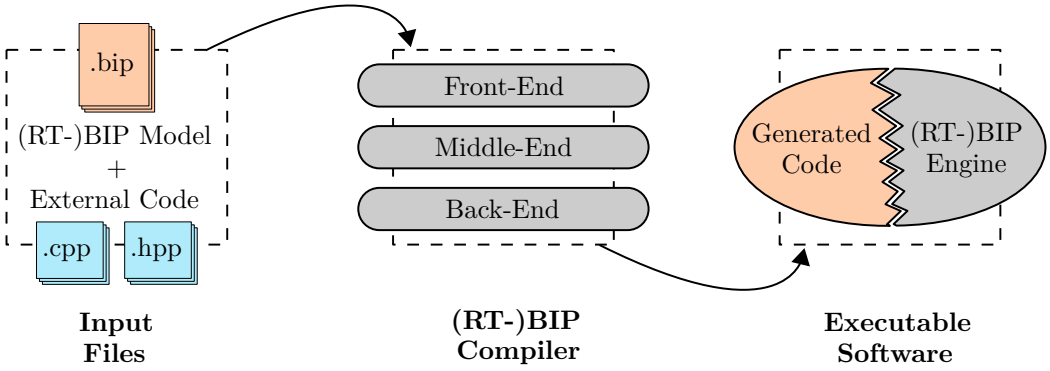


Figure 3.7: Code generation process for RT-BIP models

3.2.1.1 The Modeling Language

An RT-BIP model is the combination of several components interacting through connectors. These components can be either *atomic*, describing the behavior of a single element of the system, or *compound* component combining a subset of components with their interactions and priorities. The behavior of atomic components is modeled based on the timed automata framework [11], extended with data variables. The framework provides a textual language that allows one to describe such components using specific keywords and patterns. An atom is defined as a set of locations declared with the keyword **place**, a set of communication ports identified by the lexeme **port**, an a set of transitions. An atom starts at the initial location, declared with "**initial to** LOCATION_NAME **do** {ACTIONS;}", and transits from locations to others using transitions of the shape:

on PORT_NAME
from SOURCE_LOCATION **to** TARGET_LOCATION
provided (GUARD)
urgency_level
do {ACTIONS;}.

In this component, data variables are defined by their type and name preceded by the keyword **data**, and clocks follow a specific structure "**clock** CLOCK_NAME **clock** CLOCK_UNIT". The update on value of variables is defining through a set of ACTIONS.

The language enables one to include external code written in C++. This feature increases the modeling capabilities with complex computations and user-defined data structures. This external code can be either external functions used to update the values of data variables at the execution of an interaction, or external data types.

Syntactically, each element of a BIP model can be annotated. These annotations are specific instructions that give the possibility to extend the BIP language without modifying the BIP parser. For example, the annotation `@cpp{src="...", include="..."}` is used to include external C++ libraries.

3.2.1.2 The Compiler

The compiler takes as input a RT-BIP model and produces, if the model conforms with the syntax, a platform-dependent executable. This module is decomposed in 3 layers:

1. the *front-end* is responsible for parsing the '.bip' text file, checking its syntactic correctness, and building an internal representation of it in the form of a BIP-EMF[1] model.

2. the *middle-end* is useful to implement model to model transformation, which may be of interest in order to remove dead code, or to perform architectural modifications (eg. flattening).

3. the *back-end*'s role is to produce a source code compliant with C++11 standards, given the BIP-EMF model. Currently, this layer is developed using Eclipse Acceleo that provides tools for model-to-text transformation. This code generation is platform-dependent, that is, the back-end is designed for a specific target architecture and produces a source code that conforms to that platform.

The generated code is organized in terms of classes that represent the BIP entities declared in the textual input model, namely, a class for each component, port, connectors, etc. In addition, a `deploy` file that serves to assemble the generated classes is generated. It also links the assembled components to the simulation engine. This file represents the entry-point of the system simulation.

3.2.1.3 The Execution Engine

The engine implements the coordination semantics of BIP models, i.e., the scheduling of interactions, while the semantics of the individual components is part of the generated code. The engine drives the execution of the BIP model with respect to the defined semantics.

The simulation of an RT-BIP model is handled by the engine's scheduler that chooses an interaction amongst all the enabled one, at each iteration. The scheduling policy consists to first compute a race interval by intersecting the valid time values for all the enabled interactions. Then the scheduler plans a date for each interaction in the resulting race interval. Finally, the interaction with the smallest waiting time wins the race.

In BIP, the code for the engines is written in C++ and is decomposed into a *generic* and a $specific$ parts. The former is composed of 33 header and 30 source files that define the

[1] EMF stands for Eclipse Modeling Framework used to produce Java classes from a model specification.

interfaces between the generated code of the model and the execution engine. The *specific* part implements all the methods of the generic interface, plus additional classes, to express the semantics, optimizations and the deployment supported by the engine. For example, BIP provides engines for untimed or timed models, with or without (called reference-engine) optimized computation, in a monolithic, multi-threaded or distributed deployments.

The entry-point of the engine is the class `Launcher`, that is responsible of consuming the simulation parameters and setting the parameters of the system scheduler accordingly. The latter is defined in the `RandomScheduler` that implements the simulation algorithm.

3.2.2 The SRT-BIP Extension

SRT-BIP combines the real-time modeling capabilities of the RT-BIP framework with the ability to model probabilistic information, similarly to stochastic BIP [122]. We build SRT-BIP as an extension of the RT-BIP framework to implement the semantics defined in Section 3.1. In the RT-BIP framework, all the needed mechanisms relative to timed interactions already exist. Hence, efforts are concentrated on the newly introduced stochastic behavior. This is achieved by modifications applied at different levels: the modeling language, the code generation and the simulation engine.

Furthermore, to increase the usability of the new engine and its debugging capabilities, we extend the set of simulation parameters with two optional attributes, namely, `--log-stoch-choice` to display the intermediate scheduling decisions, i.e., sampled dates and content of planning memory, and `--log-variables` to systematically log all the data variables of the BIP model.

3.2.2.1 Expressing Probabilities

From the language point of view, we define stochastic interactions using specific annotations `@stochastic(dist="...", clk=..., param="...")` to tag components ports. Such annotations specify a probability density function and its parameters through, respectively, the `dist` and `param` attributes, and associate a clock through the `clk` attribute. For example, a stochastic port following a normal distribution $\mathcal{N}(10, 2)$ associated with the clock y is defined by `@stochastic(dist="normal", clk=y, param="10,2")`. Currently, the language supports a number of built-in density functions, namely, normal, gamma and χ^2. Additionally, empirical density functions can be used through the same mechanism: by setting `dist="custom"`, and `param` pointing to a file that characterizes the underlying cumulative distribution.

The stochastic annotation is parsed by the BIP parser in the front-end and propagated to the back-end of the compiler, where we treat it, as explained next.

3.2.2.2 Enriching the Generated Code

This modification concerns the back-end responsible for producing C++ source code from the SRT-BIP model. Generating code for stochastic behaviors requires to extract the information contained in stochastic annotations and use it for code generation. To do so, we extend the classes representing the `Port` declarations with additional attributes:

- **String** *distribution* : this attribute records the content of the annotation by concatenating the annotation keys `dist` and `param`, separated by a comma.
- **Clock** *clock* : this attribute represents the stochastic clock pointed out by the annotation key `clk`.

These attributes are added during the generation of the port classes, but their initialization is performed at deploy time, i.e., in the `deploy` file. In addition to the getters and setters of these attributes, we extend the component classes with a utility function, denoted `print_data_vars()`, which allows one to log their data variables.

The obtained code is partially validated at compile time: the syntactic correctness of the annotation and the declaration of the clock associated with the stochastic annotation are checked. For practical reasons, the validation relative to the stochastic semantics and the associated assumptions, such as the participation of at most one stochastic port in an interaction, is performed at runtime and delegated to the stochastic real-time engine.

The implement related to the update of the modeling language and the compiler(s back-end (code generation) required to modify 5 Acceleo files for a total of 161 added lines of code.

3.2.2.3 The Stochastic Real-Time Engine

The Stochastic Real-Time engine implements the operational semantics of stochastic real-time BIP systems. We first describe this engine from the functional view then we detail its implementation.

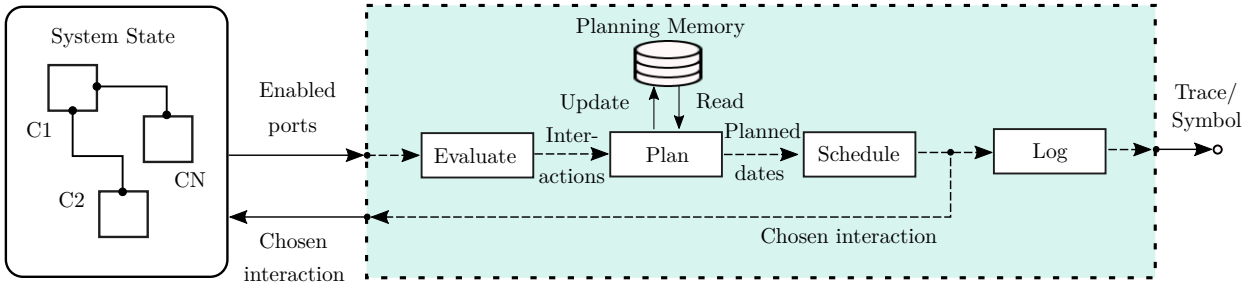


Figure 3.8: Functional view of the stochastic simulation engine

The functional description. The functioning of the stochastic simulation engine is depicted in Fig. 3.8. The engine iteratively selects one interaction to execute among the enabled ones.

At every iteration, the engine computes the firing time interval for every interaction, based on current clock valuations and interaction guards (**Evaluate**). Next, an execution date is chosen for every future enabled interaction (**Plan**). For *timed interactions*, the date is chosen by sampling a value in the associated firing interval, using either a uniform or exponential law, depending if the firing interval is bounded or not (see Section 3.1.3.3 for more details). For *stochastic interactions*, the date is chosen according to their associated probability density function (encoded by the *distribution* attribute of the `Port` class, see Section 3.2.2.2) and the clock value. Two cases are distinguished: when the current value of the clock is zero, the date is chosen by a direct sampling of the corresponding density. However, when the clock has a strict positive value, the execution date is planned using the truncated density function at that value.

Once all the future enabled interactions are planned, the scheduler applies a race policy to select for execution the one having the earliest planned date (**Schedule**). The simulation time is advanced to that date and the interaction is executed on the system and logged (**Log**).

For efficiency reasons, planned execution dates are stored in the **planning memory**, to avoid re-planning interactions that remain enabled when moving to the next system state. A new execution date is chosen only for newly enabled interactions and/or in conflict with the executed interaction. That is, when the associated clock (for stochastic interactions) has been reset, or the firing interval has changed due to execution of the previous interaction.

The implementation details. We extend the RT-BIP engine to support the new simulation options, the planning memory, the stochastic simulation algorithm, and the time sampling method. The simulation options are parsed at the entry-point of the engine, namely in the class `Launcher`. We enrich the set of simulation options with two optional parameters to automatically log data variables and to display the scheduler choices. This boils down to adding a flag for each option which indicates whether it has been enabled by the designer. The flag of the `--log-stoch-choice` is used by the class `RandomScheduler` to know whether to display the firing interval, the sampling technique and the planned value for each enabled interaction, together with the content of the planning memory. The option `--log-variables` enables the call of the function `print_data_vars()` implemented by the generated code of the root component.

The planning memory is deployed in the `RandomScheduler`. For each type of transitions, namely, interaction, internal and external ports, we implement a map structure of the form:

```
std::map <const  Transition_Type_Class* , std::pair<TimeValue, Context> >
          Transition_Type_Memory;
```

This structure is used to memorize the planned date (of type `TimeValue`) and the context in which this date has been sampled (`Context`). This context represents either the firing interval if the transition is `timed`, or a pair of global and stochastic clock valuations if the transitions is *stochastic*. In addition, we implement the necessary functions to update, search and to log the

map structures. The function `cleanMemory()` is an example of update functions which role is to apply the re-sampling rules by removing the conflicting transitions from the memories.

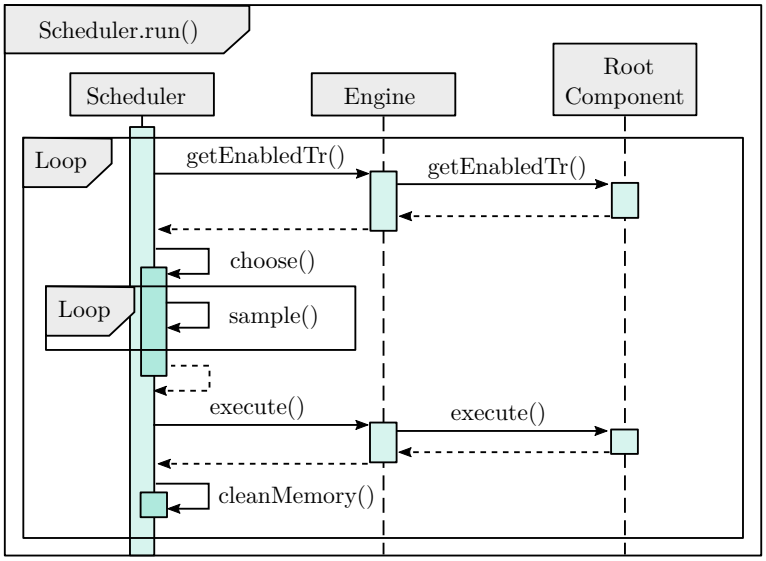


Figure 3.9: Sequence diagram of the simulation algorithm

For the simulation algorithm and the sampling method, the main changes are concentrated in the `RandomScheduler`. Figure 3.9 depicts the implementation of the simulation algorithm in the class `RandomScheduler`. In this class, the main simulation loop is achieved by the function `run()`. This latter starts by collecting the set of enabled transitions by communicating with the root component through an instance of the class `Engine`. Then, a date is planned for each one of them by calling the function `sample()`. The scheduler uses these planned dates to select one transition for execution (using function `choose()`), and it notifies the `Engine` to execute that transition. Finally, the planned memory is cleaned with respect to the selected transition before starting the next iteration.

The function `sample()` is overloaded for timed and stochastic transitions. Timed transitions are sampled using uniform or exponential distributions. For stochastic ones, they are either sampled from the original distributions or from truncated ones. For the former, we implement a function based on the GNU Scientific Library (GSL) that works as follows: we first compute an offset according to the corresponding distribution before computing the planned date by adding that offset to the current value of the global time. To sample from truncated distributions, we use rejection sampling [97]. The principle of this Monte Carlo technique is to generate random points in the 2D plan and to accept the ones that are under the graph of the target density function, as represented by the green area in Figure 3.10. In this case, the offset to the planned date is computed as the difference between the sampled date and the value of the clock.

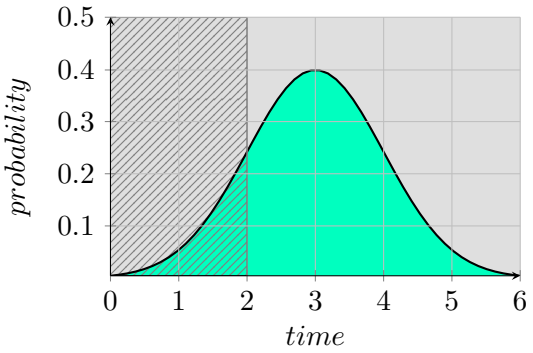


Figure 3.10: Rejection sampling from a normal distribution $\mathcal{N}(3,1)$ truncated at time $t = 2$. Random points are generated in the non-dashed area. The green area represents the accepted points and the light gray one identifies the rejected ones.

The stochastic engine ensures that the simulation conforms to the defined SRT-BIP semantics under the assumption that the syntactic restrictions are respected. This is done by systematically checking the following at runtime:

- at most one stochastic port is part of an interaction.
- an interaction does not combine stochastic and timed ports with time guards.
- the distributions specified in the stochastic annotations are supported by the sampling algorithm.
- for stochastic interactions, the sampled value is always greater or equal to the valuation of the associated clock.
- the content of the planning memory is always consistent with the global time.

The implementation related to the support of the new simulation options, the planning memory, the stochastic simulation algorithm, and the time sampling method required to modify 15 header and source files, for a total number of 1055 updated lines of code. It is worth mentioning that the major changes are concentrated in the class `RandomScheduler` that represents about 80% of the updates.

3.3 Conclusion

In this chapter, we presented a new component-based formalism that allows for the modeling of stochastic real-time systems. We detailed its semantics and showed that is corresponds to a GSMP. We also presented an implementation based on previous efforts of simulating real-time systems (RT-BIP). Our goal was to cope with the lack of modeling formalisms that handle arbitrary stochastic behaviors with timing aspects, and to close the gap between high-level models and low-level system implementations.

The stochastic real-time BIP formalism enables to build stochastic timed models and to compose them through multi-party interactions. It offers a mean to express stochastic timing constraints over systems interactions by attaching probability density functions to the guards of ports composing them, and to specify different urgency levels for them. In contrast to the previous stochastic semantics of BIP [122], which is discrete, this new formalism supports reasoning over dense time.

Thanks to its expressiveness, the SRT-BIP formalism allows one to model real-life systems with complex behavior. However, faithfully designing such systems is challenging. In the next chapter, we present a machine learning technique that aims to cope with this difficulty by automatically inferring timed models with probabilities from a set of observations on the system.

Chapter 4

Learning Timed Models with Probabilities

Analyzing system performance in a model-based approach requires first to build a faithful performance model. This can be done by augmenting a functional model with performance information that are obtained in different manners, namely, extracted from documentation, computed by static analysis or collected at system runtime. Recently, Nouri et al. have proposed a rigorous way of building such models [123]. The proposed methodology, called ASTROLABE, consists of building a performance model from functional representations using statistical inference and a manual model calibration, in a meet-in-the-middle fashion. ASTROLABE relies on BIP for the modeling of the system and the generation of an executable version of it (code generation), before using statistical inference in order to extract performance information from actual system runs. The performance model is finally obtained by manually augmenting the functional model with the inferred performance information. Although rigorous, this methodology suffers from different weaknesses, mostly due to its reliance on statistical inference. First, this latter requires data independence which is not always the case in practice. Moreover, the distribution characterization is inaccurate since it depends on the designer interpretation of plots and charts. The inference is tedious and manual. But also, the fact that the model calibration operation is manual introduces a serious overhead in the design time and can be error-prone. To cope with these limitations, we propose to automatically learn the entire performance model from executions of the system using Machine Learning (ML) techniques in general, and Grammatical Inference (GI) algorithms in particular.

Machine learning is an active field of research where new algorithms are constantly developed and improved in order to address new challenges and new classes of problems. Grammatical inference is a sub-category of ML that studies the automatic construction of a model out of system observations, i.e., given a learning sample S, a GI algorithm infers an automaton that, in the limit[1], would represent the language L of the actual system. Despite the wide

[1] By considering a sufficient number of observations [55].

development of ML techniques, only few works were interested in learning stochastic timed models [56, 110, 137, 142]. These system models are of paramount importance for performance evaluation, extending existing systems with new functionalities, or for documenting legacy code.

RTI+ algorithm [142] is one of the algorithms that learn a sub-class of timed automata augmented with probabilities, called Deterministic Real-Time Automata (DRTA). This method relies on a tree representation of S, that we identified to introduce uncontrolled generalization, which impacts the quality of the learned models. In this context, we propose a more accurate learning procedure by investigating different representations of the learning sample.

The remainder of this chapter is organized as follows. Section 4.1 introduces the principles of Grammatical Inference and existing algorithms. In Section 4.2, we present the RTI+ algorithm for learning timed models with probabilities and we introduce our formalization of it. We present our improvements on the representation of the learning sample in Section 4.3. The obtained experimental results are given in Section 4.4. Finally, Section 4.5 presents a systematic method to transform the learned DRTA models to SRT-BIP.

4.1 Grammatical Inference

Grammatical Inference [55] has been initially developed in the context of natural language recognition. The purpose was to identify production rules that govern natural languages. GI techniques are further applied to extract some properties given samples from the language or to find a more compact representation of the language in the form of grammars or automata. In this section, we introduce principles of grammatical inference. Additional existing techniques to learn models from traces are also discussed and classified.

4.1.1 Principles

Grammatical Inference consists on learning language representations given information about the language. This learning setting depends on the class of languages $\mathcal{L}$ we are interested to learn. Learning timed languages, which is the subject of this chapter, is different from learning untimed ones in terms of complexity, the input information that are given about the language and how to deal with the given information.

When designing a learning algorithm, the first question is to know what sort of *languages* one intends to learn, according to the Chomsky hierarchy for instance. These languages may be infinite and need to be formally *represented* in a finite manner. Usually, several representations can be used to encode the same language. For example, a regular language can be represented either as a regular expression, a deterministic finite state automation or a non-deterministic one. The choice of the representation impacts the size and the shape of the manipulated structure in the learning process. Finally, one has to determine the shape of the information available about

the language, such as, labeled words or simply words from the language, and how information are *presented* to the learner, such as, one information at a time or all at once.

The learning setting identifies the interaction between the main notions in learning, namely, the targeted languages, representations and presentations. Given the target language L, the learning algorithm is presented a set of information about L in order to learn a representation , denoted grammar G, of that language. These information are given by a presentation function Φ. To allow learning L, the presentation must be complete, that is, it eventually outputs all the information about L. Presentations determine the shape of the information given about the language. Standard presentations for L are *text* and *informant*. Learning from a text presentation consists of learning given only examples of the language, namely, words that belong to the target language, Informant presentation mode returns examples and counter-examples of the target language. Words are attached with labels that indicate whether they belong to L or not.

4.1.2 Learnability

Learning a language relies on several parameters such as the language complexity and the shape of the presentations. A class of languages is considered *learnable* with respect to a presentation if there exists an algorithm that can be used by the learner to build the correct representation. With time, the algorithm is given more information to refine its guess about the target language. This algorithm must have the property that, after some finite time, the learning process converges to the correct grammar (or an equivalent one). This learning framework is called **identification in the limit**.

Since learning is based on a given amount of information, called learning sample, the learning algorithms can be characterized according to the required quantity of information to reach the convergence point. A learner A learns from polynomial data if it requires a polynomial number of presentations of polynomial lengths to converge to a correct grammar. Conversely, if the required amount or length is exponential then the learner is said to learn from exponential data. This latter kind of learners is not very useful in a context where the available amount of data may be restricted by some environmental constraints, which may be the case in practice.

In addition to the amount of data, the update time is also a characterization criterion for learning algorithms. It represents the required execution time to build the i^{th} grammar from the i first presentations. A learner A learns in polynomial update time if constructing the i^{th} hypothesis (grammar) requires an execution time that is polynomial in the number of symbols of the i first presentations.

We talk about *efficiently* learning in the limit when an algorithm is guaranteed to learn the correct grammar G from polynomial data and in polynomial update time.

4.1.3 Learning Algorithms: an Overview

Inferring grammars is a complex problem. Indeed, Gold [75] showed that positive and negative samples are both required to correctly identify even the simplest class of languages, that is, the class of regular languages, and equivalently identifying Deterministic Finite-state Automata (DFA).

RPNI. This algorithm [127] has been proposed to learn DFAs from informant. This state-merging algorithm starts by building an initial representation of the learning sample called Prefix Tree Acceptor (PTA). States in this tree are explored in a lexicographic order, seeking for possible merges. The role of the merge operation is to generalize the learned language since it initially contains only the words that are in the positive sample. The negative sample is used to reject a merge by checking the consistency of the model after merge, that is, this model must not accept words belonging to the negative sample.

EDSM. This algorithm [104] improves RPNI's exploration policy: Instead of performing the first possible merge, all the possible merges are scored and the best merge is selected. The idea is to do the best choices at each step in order to minimize the snow ball effect caused by early bad decisions. As with RPNI, EDSM cannot be used in the absence of negative examples.

Learning k-testable languages. A constructive approach [71] has been proposed to learn DFAs from a text presentation. This method is applicable for the inference of a subset of regular languages denoted k-testable, that is, identifiable with a window of size k In this approach, a DFA is built by observing the positive examples with a sliding window of k symbols. Given the learning sample S, a 5-tuple $Z_k = \{\Sigma, I, F, T, C\}$ is constructed such that:

- Σ is the alphabet,
- I is the set of prefixes of length $k-1$,
- F is the set of suffixes of length $k-1$,
- T is the set of sub-strings of length k,
- C is the set of words $\omega \in S$, such that $|\omega| < k$.

The recognized language is the set of words which prefixes of length $k-1$ are in I, suffixes of length $k-1$ are in F , all sub-strings of length $k-1$ are in T, union the set of words in C. It can be written as $L(Z_k) = I\Sigma^* \cap \Sigma^* F - \Sigma^*(\Sigma^k - T)\Sigma^* \cup C$. The main drawback of this method is that it cannot be applied to arbitrary regular languages since they are not all k-testable.

Alergia. This algorithm was proposed in [40] for the inference of probabilistic DFAs from text presentations. This RPNI-like algorithm extends the Prefix Tree Acceptor with frequencies on the states and the transitions. These frequencies are used to evaluate the compatibility of

two candidate states for a merge. The compatibility criterion, based on the Hoeffding bound, quantifies the similarity of these two states: they are compared in terms of probability to transit with symbols of the alphabet and their ending probability.

MDI. In the MDI [140] algorithm, the authors proposed to substitute this local-range criterion by the Kullback-Leibler (KL) divergence. Given the Frequency Prefix Tree Acceptor (FPTA), the model before merge and the model after merge, the KL divergence value is computed between the FPTA and each model. A merge is considered compatible with the learning sample if the difference between the two divergences relative to the reduction of the number of states provided by the merge, is small enough.

AAlergia. A variant of Alergia called Angluin-based criterion Alergia (AAlergia) is proposed in [110] in the context of deterministic LMC inference. Checking compatibility of two states is based on local information about probability distribution over transition symbols. For these states, the highest difference in probability to transit with the same symbol is compared to a data-dependent threshold in order to evaluate whether to perform the merge. Since it is based on Alergia, that learns DFA, an additional operation is performed to transform the DFA into a deterministic LMC by removing the final states of the DFA, and updating the transition probabilities.

RTI+. Verwer et al. proposed a state-merging algorithms to learn timed models. RTI+ [142] is an EDSM-like variation to learn a sub-class of timed automata with probabilities, called Deterministic Real-Time Automata (DRTA). In this algorithm, transitions are labeled with time intervals that represent clock constraints. Beside the merge operation, a time-split operation is introduced to discriminate between distinct time behaviors. The Likelihood Ratio (LR) test is implemented for both scoring operations and identifying their feasibility.

BUTLA. Another variation of state-merging algorithms for timed automata learning called BUTLA is presented in [108]. Unlike RTI+, clock constraints on the transitions are expressed as probability density functions over time values. In this algorithm, merges are explored in a bottom-up order to minimize the determinization cost. After all the merging steps, an additional loop is added to detect multi-modality in the probability density function and candidate transitions are split into unimodal Gaussian distributions.

The L* algorithm. Active learning relies on the interaction between a learner that wants to learn the target language L and an oracle that knows L. The learner can send several kinds of queries, namely, membership, equivalence, subset and sampling queries. Membership queries allow the learner to ask the oracle whether a word belongs to L, while equivalence queries aim at comparing the learner's hypotheses to L. In the latter, the oracle answers positively if

the two languages are equivalent. Otherwise, a negative answer is returned together with a counter-example, but only for *strong* equivalent queries. Angluin showed that membership and strong equivalence queries are enough to correctly identify a regular language, and proposed the L* algorithm [13]. In this algorithm, the learner maintains a table that is filled using membership queries. At each iteration, the learner submits a closed and consistent table for equivalence checking and the oracle evaluates the corresponding DFA. If this DFA is not exactly correct, the oracle returns a counter-example. Tis counter-example is taken into account by the learner to refine his table, through consistency and closure operations. More recently, a variant of this algorithm called TTT[83] was proposed as a means to support long counter-examples.

4.1.4 Classification

Learning models has become a challenging task due to its multiple applications, such as in model-based design, verification, testing and diagnosis. Several algorithms have been proposed for automata learning [13, 40, 71, 104, 108, 110, 127, 140, 142]. These approaches differ in the class of models they learn and in their learning algorithm.

4.1.4.1 Target Model

GI approaches vary in terms of learned models. Indeed, each algorithm is fashioned to learn a specific class of models. This specificity impacts the shape of the input data and the required operations during the learning process in order to infer the chosen model class. Algorithms for inferring DFAs [13, 71, 104, 127] and Probabilistic DFAs (PDFA) [40, 140] have been proposed.

These methods often serve as a basis for methods that learn more complex modeling formalisms. For example, Alergia was extended for Discrete-Time Markov Chains (DTMC) [110], timed automata with probabilities [108, 142] CTMCs [137], GSMPs [56] but also MDPs [111]. The authors in [70] proposed to learn Directed Acyclic Graphs (DAG) based on the k-testable learning algorithm. The approach in [145] proposes to learn DFAs augmented with data variables and constraints by following an EDSM-like process. The L* algorithm was extended to learn NFAs [33], Mealy machines [139], MDPs [43] and more recently non-deterministic automata on infinite alphabets [120]. The interested reader can find more information about the extensions of L* in [81].

4.1.4.2 Active Vs. Passive Learning

Learning approaches can be split into two types: active learning and passive learning techniques. Active learning algorithms [13, 33, 43, 139] are interactive and iteratively build their hypothesis by refinement, given an example or a counter-example at each iteration. Active learning represents a powerful manner to correctly learn a DFA. However, it requires to have a perfect oracle that provides useful examples and counter-examples, and that can answer strong equivalence queries in an automated way. When the target system is a white-box, this

equivalence can be checked. But in practice, the system to learn is often a black-box and the only possible interaction is restricted to membership queries through the system interfaces. In literature, solutions are proposed as an alternative to the strong equivalence queries, such as conformance testing using the partial W-method [68] or Model-Checking (MC) [130], and Probably Approximately Correct (PAC) learning [113]. The partial W-method is a conformance testing technique that constructs a set of test queries to distinguish between the learned model and the target language. This method is guaranteed to find a counterexample if the number of states in this target model is known. The idea of using MC relies on the ability to formally specify the expected behavior of the target system using a set of properties that are then verified against the learned hypothesis. Using PAC-learning for approximating equivalence queries identifies the minimal number of membership queries to perform in order to claim that the learned model is equivalent to the target language.

Passive learning algorithms [40, 71, 104, 108, 110, 127, 140, 142] are off-line techniques that run on a fixed set of observations given as input. Several algorithms have been proposed to learn models from informant presentations [104, 127]. These algorithms are not always applicable in some contexts where negative examples are not available. Techniques that learn from text presentations cope with the aforementioned limitation [40, 71, 108, 110, 140, 142]. In both cases, the idea is to generalize a set of observations when trying to infer the target language. The difficulty is to find the right level of generalization. Informed learners use the negative sample to reject an operation since the inferred language must be consistent with the learning sample. This consistency cannot be verified in the case of text learners and it has to be approximated by statistical tests in order to find the correct degree of generalization.

More recently, a combination of active learning (the TTT algorithm) and passive learning (RPNI for learning DFA) was proposed in [150], to accelerate the testing phase of active learning algorithms. In this approach, the equivalence is answered by a sequence of three oracles of increasing complexity:

(1) a log oracle verifying that the observed system traces are accepted by the proposed hypothesis,

(2) a passive-learning oracle produced by generalizing these traces, and

(3) a conformance testing oracle implementing the partial W-method.

4.1.4.3 Iterative Vs. Constructive Approach

There are few GI algorithms that work in a constructive manner [70, 71]. These approaches aim to build the learned model state by state, given information about the language. Most of the algorithms that are proposed for inferring languages are iterative. We distinguish two types of iterative approaches: top-down and bottom-up techniques. In the top-down, the process starts with a sub-language described in the learning sample. The goal of the iterations

is to generalize that sub-language using model transformation on the sample representation in order to induce the target language. This is the case for state-merging (SM) algorithms [40, 104, 108, 110, 127, 140, 142]. In these iterative approaches, the initial sample is represented as a tree with annotated transitions then the size of the model is reduced by merging equivalent states of the model. The condition on a merge varies from an algorithm to another, and depends on the targeted model type and the presentation mode (informant or text). In contrast, bottom-up approaches [13, 33, 43, 139] start from a single-state model thats accepts the universal language and refine it based on information that are incrementally collected through iterations. In such approaches, a learner aims at deducing the target language by interacting with an entity that knows the target language, namely, the oracle.

4.1.4.4 Summary

Table 4.1 summarizes the above-mentioned algorithms. As one can see, there are only few algorithms that are interested in learning probabilistic timed models. Our work is based on the work of Verwer et al. [142] that we improve and apply in the context of real-time embedded systems. This choice is justified by the good quality of the results presented in [142], its reliance on EDSM[2], and the availability of the tool in open-source.

Algorithm	Learning type	Algorithm type	Presentation	Representation	Compatibility criterion
RPNI	Passive	SM	Informant	DFA	Consistency with negative sample
EDSM	Passive	SM	Informant	DFA	Consistency with negative sample
K-testable	Passive	Constructive	Text	DFA	-
Alergia	Passive	SM	Text	PDFA	Hoeffding bound
MDI	Passive	SM	Text	PDFA	KL Divergence
AAlergia	Passive	SM	Text	LMC	Angluin-based
RTI+	Passive	SM	Text	TA	Likelihood ratio test
BUTLA	Passive	SM	Text	TA	Hoeffding bound
L*	Active	Constructive	Interactive	DFA	-

Table 4.1: Classification of GI algorithms

4.2 The RTI+ Learning Procedure

RTI+ [142] is a state-merging algorithm for learning DRTA models from a sample of timed words. The algorithm first builds a PTA then reduces it by merging locations having similar

[2]EDSM won the Abbadingo DFA learning competition in 1997

behaviors, according to a given compatibility criterion. Compared to other state-merging algorithms, RTI+ relies on a time-split operation to identify the different time behaviors and to split them into disjoint transitions. The algorithm is able to learn a stochastic DRTA, i.e, a $DRTA^+$ where the strategy is obtained from the associated annotation function $\mathcal{N}$. In this section, we introduce a formal definition of the RTI+ learning algorithm, its operations and the targeted sub-class of models.

4.2.1 The Learned Model

Let Σ be a finite alphabet, Σ^* the set of words over Σ and ϵ the empty word. Let $\mathbb{T}$ be a time domain and $\mathbb{T}^*$ the set of time sequences over $\mathbb{T}$. We consider integer time values, i.e., $\mathbb{T} \subseteq \mathbb{N}$. For a set of clocks $\mathcal{C}$, let $CC(\mathcal{C})$ denote the set of clock constraints over $\mathcal{C}$. Let $\mathcal{I}$ be the intervals domain, where $I \in \mathcal{I}$ is an interval of the form $[a; b],\ a, b \in \mathbb{T}$ such that $a \leq b$, and represents the set of integer values between a and b. Let ω be an untimed word over Σ and τ a time sequence over $\mathbb{T}$. We write $\omega \leq \omega'$ (resp. $\tau \leq \tau'$) whenever ω (resp. τ) is a prefix of ω' (resp. τ'). We also write ωu (resp. τv) for the concatenation of ω (resp. τ) and $u \in \Sigma^*$ (resp. $v \in \mathbb{T}^*$). $|\omega|$ (resp. $|\tau|$) is the size of ω (resp. τ).

4.2.1.1 Deterministic Real-Time Automata (DRTA)

A Real-Time Automaton (RTA) is a timed automaton with a single clock that is systematically reset on every transition.

Definition 4.2.1 (Real-Time Automaton (RTA))**.** An RTA is a tuple $\mathcal{A} = \langle \Sigma, L, l_0, \mathcal{C}, T, inv \rangle$ where:

- Σ is the alphabet,
- L is a finite set of locations,
- $l_0 \in$ L is the initial location,
- $\mathcal{C} = \{c\}$ contains a single clock,
- T $\subseteq L \times \Sigma \times CC(\mathcal{C}) \times \mathcal{C} \times L$ is a set of edges with a systematic reset of the clock c,
- inv : L $\longrightarrow CC(\mathcal{C})$ associates invariants to locations.

For more convenience, transitions are denoted as $l \xrightarrow{\sigma, I} l'$, where $\sigma \in \Sigma$ and $I \in \mathcal{I}$ is a time interval including both transition guards and location invariants. For simplicity, we also omit the systematic reset.

An RTA is deterministic ($DRTA$) if, for each location l and a symbol σ, the timing constraints of any two transitions starting from l and labeled with σ are disjoint, i.e., $\forall l \in L, \forall \sigma \in \Sigma, \forall t_1, t_2 \in T$, $t_1 = \langle l, \sigma, I_1, l_1 \rangle$ and $t_2 = \langle l, \sigma, I_2, l_2 \rangle, (I_1 \cap I_2 \neq \emptyset) \Leftrightarrow (t_1 = t_2)$. A DRTA generates timed words over $\Sigma \times \mathbb{T}$. Each timed word is a sequence of timed symbols (ω, τ)=$(\sigma_1, \tau_1)(\sigma_2, \tau_2)...(\sigma_m, \tau_m)$, representing an untimed word ω together with a time sequence τ. A set of n timed words constitute a *learning sample* $S = \{(\omega, \tau)^i, i \in [1; n], \omega \in \Sigma^*, \tau \in \mathbb{T}^*\}$. We denote by $\mathbb{T}^S$ the set of time values appearing in S.

A Prefix Tree Acceptor (PTA) is a tree representation of the learning sample S where locations represent prefixes of untimed words in S. Timing information is captured in a PTA in the form of intervals $I \in \mathcal{I}$ over transitions. This structure is called Augmented PTA (APTA). In the latter, each transition $l \xrightarrow{\sigma, I} l'$ is annotated with a frequency that represents the number of words in S having l' as a prefix. An APTA can be seen as an acyclic DRTA annotated with frequencies. Let $\mathcal{N} : T \longrightarrow \mathbb{N}$ be this annotation function. Given a $DRTA$ $\mathcal{A}$, a pair $(\mathcal{A}, \mathcal{N})$ is an *annotated* DRTA, denoted $DRTA^+$.

4.2.1.2 Stochastic Interpretation of a DRTA

A $DRTA$ starts at the initial location l_0 with probability 1. It moves from a location to another using transitions in T. At each location l, a transition is chosen among the set of available transitions T_l. Selecting a transition consists of choosing a timed symbol (σ_i, τ_i). A probabilistic strategy φ that associates a probability function to each location l over the set of transitions T_l is used to make this choice: $\varphi : L \times T \longrightarrow [0, 1]$, such that $\Sigma_{t \in T_l}\ \varphi(l, t) = 1, \forall l \in L$. For the chosen transition $l \xrightarrow{\sigma, I} l'$, the choice of the time value is done uniformly over the time interval I. Figure 4.1 shows an example where two transitions labeled A and B are possible from location 1. The strategy φ associates probability 0.6 to A and 0.4 to B. Then, uniform choices on the associated time intervals are performed.

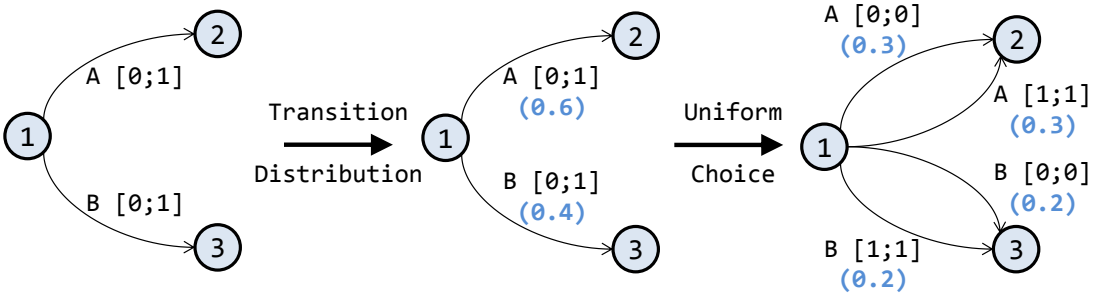


Figure 4.1: Probabilistic strategy with uniform choice

4.2.2 Building the APTA

The timed learning sample S is represented as an APTA where all the time intervals span over $\mathbb{T}$. Initially, the built APTA only contains a root node consisting of the empty word ϵ.

RTI+ proceeds by adding a location in the tree for each prefix of untimed words in S. Then, a transition labeled with σ is created from location l to location l' if the prefix of l' is obtained by concatenating the prefix of l and symbol σ. Finally, transitions are augmented with the largest time constraint $[0; max(\mathbb{T}^S)]$, where $max(\mathbb{T}^S) = Max\{a \in \mathbb{T}^S\}$. The annotation function $\mathcal{N}$ is built at the same time and represents transitions frequencies. In this work, we denote this construction as *generalized-bound APTA*.

Definition 4.2.2 (Generalized-bound APTA)**.** A generalized-bound $APTA$ is a $DRTA^+$ $(\langle \Sigma, L, l_0, \mathcal{C}, T, inv \rangle, \mathcal{N})$ where:

- $L = \{\omega \in \Sigma^* \mid \exists (\omega', \tau') \in S, \omega \leq \omega'\}$, $l_0 = \epsilon$,
- T contains transitions of the form $t = \langle \omega, \sigma, [0; max(\mathbb{T}^S)], \omega' \rangle$ s.t. $\omega\sigma = \omega'$, and $\mathcal{N}(t) = |\{(\omega' u, \tau) \in S, u \in \Sigma^*\}|$.

4.2.3 The Learning Process

The learning process aims to identify the $DRTA^+$ that represents the target language while reducing the size of the initial APTA. At each iteration, the algorithm first tries to identify the timing behavior of the system, by using time-split operations. The second step consists of merging compatible locations that show similar stochastic and timed behaviors. Locations that are not involved in merge or split operations are marked as belonging to the final model (promote operation). The algorithm proceeds by initially marking the root of the APTA as part of the final model and its successors as candidate locations. The latter will be considered for time-split, merge or promote operations.

4.2.3.1 Time-split Operation.

The time-split operation aims to identify different timing behaviors by splitting transitions. This produces models after time-split that are significantly different from models before time-split. This is done by splitting candidate transitions $t = \langle q, \sigma, [a; b], q' \rangle \in T$ into two transitions $t_1 = \langle q, \sigma, [a; c], q_1 \rangle, t_2 = \langle q, \sigma, [c+1; b], q_2 \rangle$. This operation requires to reassign the timed words attached to the candidate location q' by splitting them on the locations q_1 and q_2. The subtrees whose root are q_1 and q_2 must be rebuilt using the same APTA policy we used to represent the learning sample S.

Definition 4.2.3 and Algorithm 3 present the time-split operation relative to a generalized-bound APTA. We define primitives that are used in Definition 4.2.3. Let $\mathcal{A} = \langle \Sigma, L, l_0, \mathcal{C}, T \rangle$ be a $DRTA$ and $\mathcal{N}$ an annotation function. $\mathcal{A}^+ = (\mathcal{A}, \mathcal{N})$ is a $DRTA^+$. We define $Subtree_L$ of a location q as a set of all locations that are in the subtree whose root is q, i.e.

$$Subtree_L(q) : \{q' \mid \exists t_1 t_2 ... t_m \in T^*, t_i = \langle q_{i-1}, \sigma, [a; b], q_i \rangle, q_0 = q, q_m = q'\} \cup \{q\}$$

$Subtree_T$ denotes the set of transitions that are in the subtree whose root is q:

$$Subtree_T(q) : \{t \in T \mid t = \langle q', \sigma, [a;b], q'' \rangle, q', q'' \in Subtree_L(q)\}$$

$Reach(q, (\omega, \tau))$ is a boolean function that returns true if the $DRTA^+$ is at the location q having the timed word (ω, τ) as input:

$$Reach(q, (\omega, \tau)) = \begin{cases} True,\ if\ \exists t_1 t_2 ... t_i .. t_m \in T^*, t_i = \langle q_{i-1}, \sigma_i, [a;b], q_i \rangle, \tau_i \in [a;b], q_m = q. \\ False,\ otherwise. \end{cases}$$

$S[q]$ denotes the set of all the timed words (ω, τ) in the learning sample S that lead $\mathcal{A}^+$ to pass by location q. In other words, it is the set of timed words that have a prefix reaching location q:

$$S[q] = \{(\omega, \tau) \in S \mid \exists (\omega', \tau') \in \Sigma^* \times \mathbb{T}^*, Reach(q, (\omega', \tau')) = true, \omega' \leq \omega, \tau' \leq \tau\}$$

Definition 4.2.3 (Time-split Operation)**.** A time-split of a transition $t = \langle q, \sigma, [a;b], q' \rangle \in T$ at time $c \in [a;b]$ is an operation that updates L and T such that:

- $L := L \cup L_1 \cup L_2 \setminus Subtree_L(q')$, where:
  - $L_1 = \{\omega u_1 \in \Sigma^* \mid Reach(q', (\omega, \tau)) = true, a \leq \tau_{|\omega|} \leq c, \exists u_2, v, (\omega u_1 u_2, \tau v) \in S[q']\}$
  - $L_2 = \{\omega u_1 \in \Sigma^* \mid Reach(q', (\omega, \tau)) = true, c < \tau_{|\omega|} \leq b, \exists u_2, v, (\omega u_1 u_2, \tau v) \in S[q']\}$
- $T := T \cup T_1 \cup T_2 \cup \{ t_1 = \langle q, \sigma, [a;c], q_1 \rangle, t_2 = \langle q, \sigma, [c+1;b], q_2 \rangle \} \setminus Subtree_T(q')$, where:
  - T_1, T_2 are transitions in subtrees L_1, L_2, respectively.
  - q_1, q_2 are roots of subtrees L_1, L_2, respectively.
  - $\mathcal{N}(t_i) = |S[q_i]|$, and $\sum \mathcal{N}(t_i) = \mathcal{N}(t)$, $i \in \{1, 2\}$.

4.2.3.2 Merge Operation

Given a marked location l (belongs to the final model) and a candidate location l', this operation is performed by first redirecting the transitions targeting l', to l and then by folding the subtree of l' on l. The merge operation is also dependent on the APTA model we are using. This operation is detailed is Definition 4.2.4 and algorithm 4 in the case of generalized-bound APTA.

$Pref\_S(q)$ is a set of timed prefixes (ω, τ) of timed words (ω'', τ'') in $S[q]$ that have a prefix (ω', τ') reaching location q, that is, $Reach(q, (\omega', \tau')) = true$. This can be written as :

Data: A $DRTA^+$
Result: At location l^*, split σ-transition at time c.
Find the transition $t = \langle l^*, \sigma, [a; b], \bar{l}\rangle \in T$, where $c \in [a, b]$;
if $c \neq b$ **then**
  Delete the subtree whose root is $\bar{l}$ and the transition t;
  Create two new transitions $t_1 = \langle l^*, \sigma, [a; c], l_1\rangle$ and $t_2 = \langle l^*, \sigma, [c + 1; b], l_2\rangle$;
  $T = T \cup \{t_1, t_2\}$;
  $\mathcal{N}(t_1) = \mathcal{N}(t_2) = 0$;
  for *each timed word* (ω, τ) *in* S **do**
    Boolean split = false;
    Set the current location l to l_0;
    for *each timed symbol* (σ_i, τ_i) *in* (ω, τ) **do**
      Find the transition $t = \langle l, \sigma_i, [a; b], l'\rangle$, where $\tau_i \in [a, b]$;
      if $\exists t \in T$ **then**
        if $l' == l_1 \vee l' == l_2 \vee split == true$ **then**
          $split = true$;
          $\mathcal{N}(t) = \mathcal{N}(t) + 1$;
        end
      else
        Create a new location l';
        $L = L \cup \{l'\}$;
        Create a new transition $t = \langle l, \sigma_i, [0, max(\mathbb{T})], l'\rangle$;
        $T = T \cup \{t\}$;
        $\mathcal{N}(t) = 1$;
      end
    end
    $l = l'$;
  end
end

Algorithm 3: Time-split algorithm using the generalized-bound policy

$Pref\_S(q) = \{(\omega, \tau) \in \Sigma^* \times \mathbb{T}^* \mid \exists(\omega', \tau') \in \Sigma^* \times \mathbb{T}^*,\ Reach(q, (\omega', \tau')) = true,\ \exists(\omega'', \tau'') \in S,\ \omega' \leq \omega \leq \omega'', \tau' \leq \tau \leq \tau''\}$.

Definition 4.2.4 (Merge Operation)**.** Merging two locations $q, q' \in L$ is an operation that updates L and T such that:

- $L := L \cup L_1 \setminus Subtree_L(q')$, where:
  - $L_1 = \{\omega \in \Sigma^* \mid (\omega, \tau) \in Pref\_S(q') \wedge (\omega, \tau) \notin Pref\_S(q)\}$
- For each $t = \langle q'', \sigma, [a; b], q'\rangle \in T$:
  - If $\exists t' = \langle q'', \sigma, [a; b], q\rangle \in T$, $T = T \setminus \{t\}$ and $\mathcal{N}(t') = \mathcal{N}(t) + \mathcal{N}(t')$.
  - Else, $T = T \cup \{t = \langle q'', \sigma, [a; b], q\rangle\}$.

- $T := T \cup T_1 \setminus Subtree_T(q')$, where:

  - T_1 are transitions in subtree L_1.

The algorithm of the merge operation, in this case, is as follows:

Data: A $DRTA^+$
Result: Merge locations l_1 and l_2.
Find all the transitions of the form $t = \langle l, \sigma, [a; b], l_2 \rangle \in T$;
Update the found transitions to $\langle l, \sigma, [a; b], l_1 \rangle$;
Delete the subtree whose root is l_2;
for *each transition* $t = \langle l, \sigma_i, [a; b], l' \rangle \in T$ **do**
  $\mathcal{N}(t) = 0$;
end
for *each timed word* (ω, τ) *in* S **do**
  Set the current location l to l_0;
  for *each pair* (σ_i, τ_i) *in* (ω, τ) **do**
    Find the transition $t = \langle l, \sigma_i, [a; b], l' \rangle$, where $\tau_i \in [a; b]$;
    if $\nexists t \in T$ **then**
      Create a new location l'; $L = L \cup \{l'\}$;
      Create a new transition $t = \langle l, \sigma_i, [0; max(\mathbb{T})], l' \rangle$;
      $T = T \cup \{t\}$; $\mathcal{N}(t) = 0$;
    end
    $\mathcal{N}(t) = \mathcal{N}(t) + 1; l = l'$;
  end
end

Algorithm 4: Merge algorithm using the generalized-bound policy

4.2.4 Compatibility Evaluation

The compatibility criterion used in RTI+ is the Likelihood Ratio (LR) test. Intuitively, this criterion measures a distance between two hypotheses with respect to specific observations. In our case, the considered hypotheses are two $DRTA^+$ models: H with m transitions and H' with m' transitions ($m < m'$), where H is the model after a merge operation (resp. before a split) and H' is the model before a merge (resp. after a split). The observations are the traces of S.

We define the likelihood function that estimates how S is likely to be generated by each model (H or H'). It represents the product of the probability to generate each timed word in S [3]. Note that the i^{th} timed word $(\omega, \tau)^i$ in S corresponds to a unique path p^i in the $DRTA^+$. The probability to generate $(\omega, \tau)^i$ is the product of the probabilities of each transition in

[3] Since the timed words in S are generated independently.

$p^i = s_1 s_2 ... s_{|\omega|}$:

$$f((\omega,\tau)^i, H) = \prod_{j=1}^{|\omega|-1} \pi((\omega_j,\tau_j)^i, s_j)$$

Where $\pi((\omega_j,\tau_j)^i, s_j)$ corresponds to the probability to transit from the location s_j to s_{j+1} in H with the j^{th} timed symbol $(\omega_j,\tau_j)^i$. Given a learning sample S of size n, the likelihood function of H is $Likelihood(S,H) = \prod_{i=1}^{n} f((\omega,\tau)^i, H)$. The likelihood ratio is then computed as follows

$$LR = \frac{Likelihood(S,H)}{Likelihood(S,H')}$$

Let Y be a random variable following a χ^2 distribution with $m' - m$ degrees of freedom, i.e., $Y \rightsquigarrow \chi^2(m'-m)$. Then, $y = -2ln(LR)$ is asymptotically χ^2 distributed. In order to evaluate the probability to obtain y or more extreme values, we compute the p-value $pv = P(Y \geq y)$. If $pv < 0.05$, then we conclude that H and H' are significantly different, with 95% confidence.

The compatibility criterion concludes that a time-split operation is accepted whenever it identifies a new timing behavior, that is, the model after split is significantly different from the model before split ($pv < 0.05$). In contrast, a merge is rejected whenever the model after merge is significantly different from the model before merge since the merged locations are supposed to have similar stochastic and timed behaviors. Consequently, merge (respectively time-split) operations with a p-value closer to 1 (respectively 0) are favored.

4.2.5 Shortcomings

The RTI+ algorithm relies on the generalized-bound APTA as initial representation of the learning sample S. As pointed out before, this kind of APTA augments an untimed PTA with the largest possible time intervals, without considering the values that concretely appear in S. This introduces an *initial generalization* that leads the APTA to accept words that do not actually belong to S and might not belong to the target language L. In Example 4.2.1, we show that this initial generalization cannot be refined later in the learning process. More concretely, we observe that the time intervals that do not appear in S are not isolated and removed from the $DRTA^+$.

Example 4.2.1. Let us consider the following learning sample S = {(A,5)(B,5); (A,4)(A,3); (A,3)(B,5); (B,1)(A,5); (B,3)(B,5); (B,5)(A,1)}. The left-hand model in Figure 4.2 presents the initial $DRTA^+$ (H) of S on which we evaluate a time-split operation. The latter is expected to identify the empty interval $[0;2]$ on transition $t = \langle 1, A, [0;5], 2\rangle$, since no timed word in S takes this transition with time values in $[0;2]$. The right-hand figure represents the model assuming a split of transition t at time value 2 (H'). The LR test returns $pv = 1$ which leads to reject the time-split operation, and hence, the empty interval $[0;2]$ is not identified during the learning process.

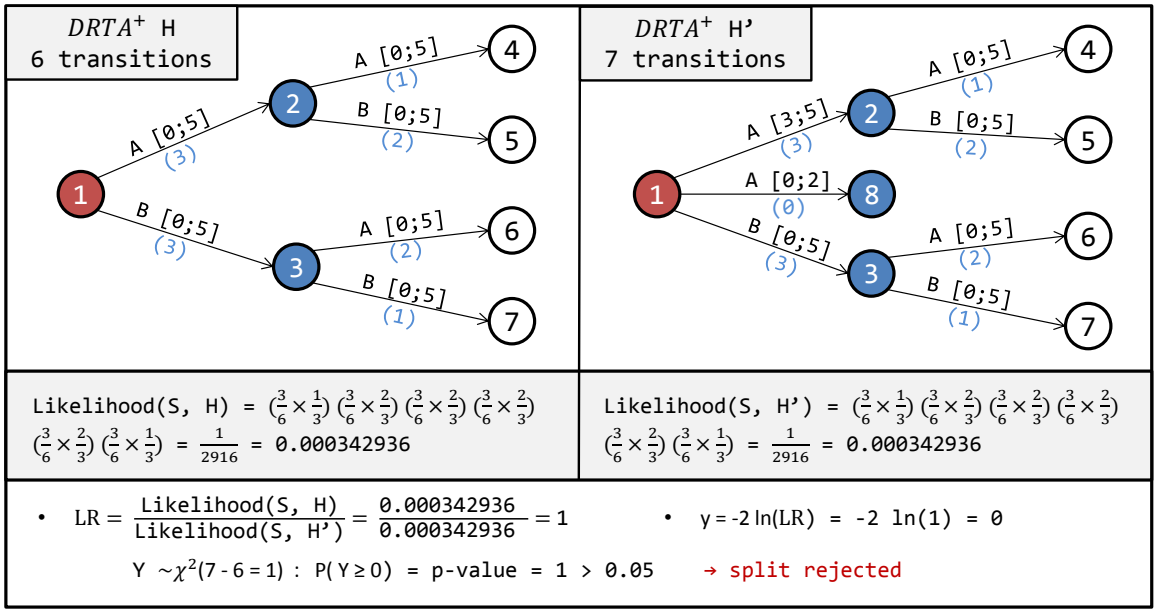


Figure 4.2: Identifying empty intervals with time-split operation

The generalized-bound APTA introduces empty time intervals that cannot be removed during the learning process. To overcome this issue, we propose, in the next section, new representations of the learning sample S.

4.3 Learning More Accurate Models

A faithful representation of the learning sample consists of building an APTA that accepts only words in S by taking into account the time values during this process. This can be done at different granularities, which results in different trade-offs between the introduced initial generalization and the APTA size. We propose three APTA models denoted unfolded, constructive-bound and tightened-bound APTAs.

4.3.1 Unfolded APTA

This APTA model fits perfectly the traces in S, that is, accepts exactly the timed words in S. Hence, it does not introduce any initial generalization. To build such a model, we need to consider both symbols and time values. The APTA initially contains the empty word. Locations are added for every timed prefix and corresponding transitions are created such that each transition only accepts a single time value, i.e., time intervals are equalities of the form $[a; a] \in \mathcal{I}$, as shown in Definition 4.3.1.

Definition 4.3.1 (Unfolded APTA)**.** An unfolded $APTA$ is a $DRTA^+$ where:

- $L = \{(\omega, \tau) \in \Sigma^* \times \mathbb{T}^* | \ \exists (\omega', \tau') \in S, \omega \leq \omega', \tau \leq \tau'\}$,

- T contains transitions of the form $t = \langle(\omega, \tau), \sigma, [a; a], (\omega', \tau')\rangle$ such that: (1) $\omega\sigma = \omega'$, (2) $\tau a = \tau'$, and $\mathcal{N}(t) = |\{(\omega' u, \tau' v) \in S, u \in \Sigma^*, v \in \mathbb{T}^*\}|$.

4.3.2 Constructive-bound APTA

A more compact representation of S compared to the unfolded APTA can be obtained by reducing the size of the initial APTA. At each location, a reduction of the number of transitions is performed by grouping all the contiguous time values for the same symbol into a single transition where the time interval $\boldsymbol{I}$ is the union of the different time intervals.

Definition 4.3.2 (Constructive-bound APTA)**.** A constructive-bound $APTA$ is a $DRTA^+$ where:

- $L = \{(\omega, \{I_i\}_{i=1}^{|\omega|}), \omega \in \Sigma^*, I_i \in \mathcal{I} \mid \forall i \in [1; |\omega|], \forall c \in I_i, \exists(\omega', \tau') \in S, \forall j < i, \tau'_j \in I_j, c = \tau'_i, \omega \leq \omega'\} \cup \{l_0 = (\epsilon, \emptyset)\}$,
- T contains transitions of the form $t = \langle(\omega, \{I_i\}_{i=1}^{|\omega|}), \sigma, \mathbf{I}, (\omega', \{I'_i\}_{i=1}^{|\omega'|})\rangle$ such that: (1) $\omega\sigma = \omega'$, (2) $\forall i \in [1; |\omega|], I_i = I'_i$ and $I'_{|\omega'|} = I'_{|\omega|+1} = \mathbf{I}$, and $\mathcal{N}(t) = |\{(\omega' u, \tau' v) \in S, \forall i \in [1; |\omega'|], \tau'_i \in I'_i, u \in \Sigma^*, v \in \mathbb{T}^*\}|$.

In Definition 4.3.2, each location corresponds to a subset of timed words that have a common untimed prefix where each symbol (of the prefix) appears with contiguous time values. A location is labeled by the given untimed prefix ω and the sequence of intervals $\{I_i\}_{i=1}^{|\omega|}$ corresponding to each symbol of ω. I_i is the interval grouping the contiguous time values for the symbol σ_i of ω. All time values of these intervals are present in at least one timed word in S. A transition is added between locations l and l' such that: (1) the concatenation of the untimed prefix relative to l and symbol σ produces the untimed prefix relative to l', and (2) adding $\boldsymbol{I}$ to the interval sequence of l gives the interval sequence of l'.

4.3.3 Tightened-bound APTA

The minimal size of APTA is obtained by allowing the minimal number of transitions from each location. This minimal number is obtained by assigning at most one transition for each symbol of Σ. The initial generalization is reduced (compared to the generalized-bound APTA) by identifying the minimum (resp. the maximum) time value t_{min} (resp. t_{max}) among all the time values for each location l and symbol σ. Then, a single transition is created from l with symbol σ and a time interval $[t_{min}; t_{max}]$. We call this APTA model a *tightened-bound* APTA and we formally define it in Definition 4.3.3. It has the same structure as the generalized-bound APTA but with tighter bounds. The time interval $[t_{min}; t_{max}]$ of each transition is computed locally depending on the corresponding timed words in S.

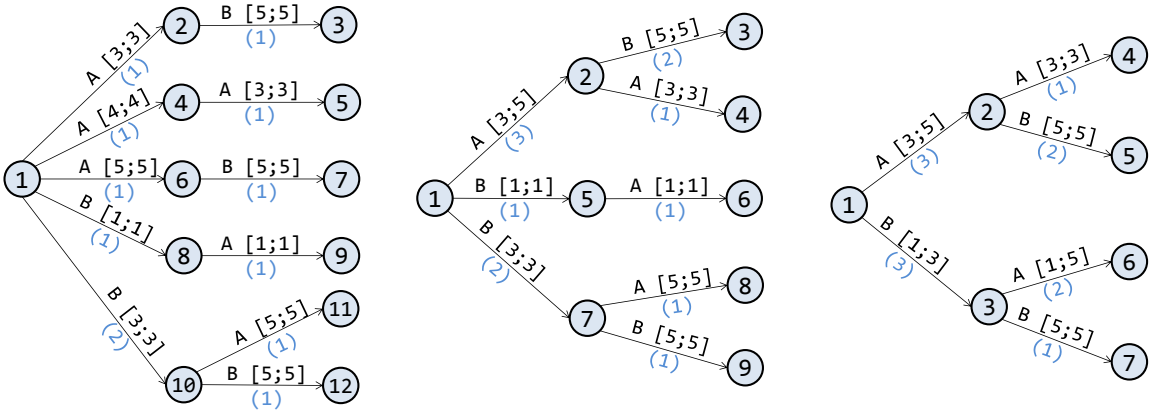
Definition 4.3.3 (Tightened-bound APTA)**.** A tightened-bound $APTA$ is a $DRTA^+$ where:

- $L = \{\omega \in \Sigma^* \mid \exists(\omega', \tau') \in S, \omega \leq \omega'\}$, $l_0 = \epsilon$,

- T contains transitions of the form $t = \langle \omega, \sigma, [t_{min}; t_{max}], \omega' \rangle$ such that: (1) $\omega\sigma = \omega'$, (2) $t_{min} = Min\{\tau_{|\omega'|} \mid (\omega' u, \tau) \in S\}$, (3) $t_{max} = Max\{\tau_{|\omega'|} \mid (\omega' u, \tau) \in S\}$, and $\mathcal{N}(t) = |\{(\omega' u, \tau) \in S\}|$, where $u \in \Sigma^*$.

4.3.4 Discussion

In this section, we discuss the proposed APTA models with respect to their ability to faithfully represent S and to their size. We consider the following learning sample S = {(A,3)(B,5); (A,4)(A,3); (A,5)(B,5); (B,1)(A,1); (B,3)(A,5); (B,3)(B,5)}. Figure 4.3 illustrates the construction of the three types of APTAs for S.



(a) Unfolded (b) Constructive-bound (c) Tightened-bound

Figure 4.3: The three APTA models for the sample S

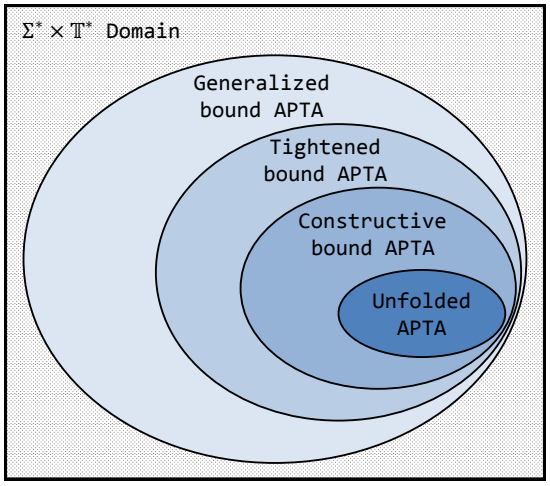


Figure 4.4: Generalization introduced by the different *APTAs*

4.3.4.1 Initial Generalization Perspective

The unfolded APTA does not introduce any initial generalization (Figure 4.3a). The constructive-bound APTA is a more compact representation of S compared to the unfolded APTA with less generalization than the generalized-bound APTA. Some generalization is introduced due to the possible combination of grouped time values. In other words, the time values of each time interval appear in S, but the language generated by the APTA over-approximates S since it accepts more time sequences. For instance, in Figure 4.3b, the timed word (A,3)(A,3) is accepted although not in S. This is due to the combination of time values coming from the timed words (A,3)(B,5) and (A,4)(A,3).

The tightened-bound APTA introduces two kinds of generalization. The first is due to the combination of grouped time values, as for the constructive-bound APTA. The second one is caused by the presence of empty intervals. An example is given in Figure 4.3c where transiting from the root is possible using the timed symbol (B,2) which is not in S. This latter generalization is similar to the one we pointed out for the generalized-bound APTA albeit with more restrictive time intervals, since the empty intervals $[0; t_{min} - 1]$ and $[t_{max} + 1; max(\mathbb{T}^S)]$ are initially removed. The relationship between these models and the generalization they introduce is summarized in Figure 4.4.

4.3.4.2 APTA Size Perspective

In terms of the size of initial representation, the unfolded APTA is the largest. Its size depends on the size of Σ and $\mathbb{T}^S$. The worst case is encountered when all the traces in S are of the same length N and when S contains all the combinations of symbols and time values. The resulting complete tree, in this case, represents the upper bound on the number of locations. This upper bound is computed as the sum of a geometric series with a common ratio $r = |\Sigma| \times |\mathbb{T}^S|$ and $U_0 = 1$ (the tree root). It can be expressed as :

$$MaxSize_{(unfolded)} = \frac{1 - (|\Sigma| \times |\mathbb{T}^S|)^{N+1}}{1 - (|\Sigma| \times |\mathbb{T}^S|)}$$

This maximum number of locations is reduced in the constructive-bound APTA by grouping contiguous time values. However, this improvement is minimal in the case where all the time values are disjoint, that is, when there are no contiguous time values to regroup. For a given interval $[0; max(\mathbb{T}^S)]$, the maximum number of disjoint intervals is equal to $(max(\mathbb{T}^S) + 1)\ div\ 2$. The upper bound on the number of locations of a complete tree of depth $N + 1$ is:

$$MaxSize_{(constructive)} = \frac{1 - (|\Sigma| \times ((max(\mathbb{T}^S) + 1)\ div\ 2))^{N+1}}{1 - (|\Sigma| \times ((max(\mathbb{T}^S) + 1)\ div\ 2))}$$

The number of locations, in the case of tightened-bound APTA, is highly dependent on the size of Σ and not on the size of $\mathbb{T}^S$, since it allows at most one interval per symbol at each

location. This is also the case of the generalized-bound APTA which returns the smallest upper bound on the number of locations, expressed as follows:

$$MaxSize_{(tighetened-generalized)} = \frac{1-(|\Sigma|)^{N+1}}{1-(|\Sigma|)}$$

4.4 Experiments

In this section, we evaluate the learned model according to its ability to accept the words belonging to L and to reject the others. This gives insight into how accurate the learned model is. A *C++* implementation of the proposed algorithms and the considered examples can be found in http://www-verimag.imag.fr/~nouri/drta-learning.

4.4.1 Evaluation Procedure

The accuracy of the learned model can be quantified using two metrics: the precision and the recall. The precision is calculated as the proportion of words that are correctly recognized (true positives) in the learned model H' over all the words recognized by H', while the recall represents the proportion of words that are correctly recognized in H' over all the words recognized by the initial model H. The precision and the recall can be combined in a single metric called F1 score. A high F1 score corresponds to a high precision and recall, and conversely.

$$Precision(H', H) = \frac{|\{(\omega,\tau) \in \Sigma^* \times \mathbb{T}^* \mid (\omega,\tau) \in \mathcal{L}(H) \wedge (\omega,\tau) \in \mathcal{L}(H')\}|}{|\mathcal{L}(H')|}$$

$$Recall(H', H) = \frac{|\{(\omega,\tau) \in \Sigma^* \times \mathbb{T}^* \mid (\omega,\tau) \in \mathcal{L}(H) \wedge (\omega,\tau) \in \mathcal{L}(H')\}|}{|\mathcal{L}(H)|}$$

$$F1\_score(H', H) = 2 \times \frac{prec(H', H) \times recall(H', H)}{prec(H', H) + recall(H', H)}$$

Based on these metrics, we distinguish four degrees of generalization for the learned models (see Figure 4.5):

1. The maximum F1 score is obtained when the exact target language $L(H)$ is learned.
2. A precision of 1 and a recall strictly lower than 1 characterize an over-approximation, i.e., the learned model H' recognizes a subset of words of $L(H)$.
3. A recall of 1 and a precision strictly lower than 1 characterize an under-approximation, i.e., the learned model H' accepts all the words of $L(H)$ in addition to extra words not in $L(H)$.
4. A precision and a recall strictly lower than 1 characterize a cross-approximation, i.e., $L(H')$ contains only a subset of words in $L(H)$ plus additional words not in $L(H)$.

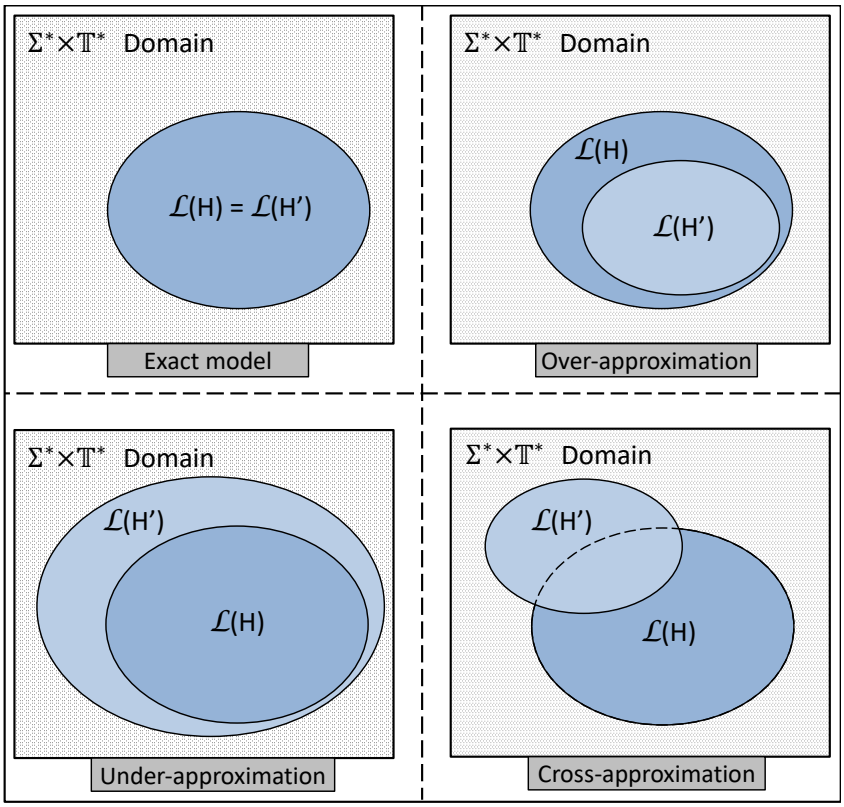


Figure 4.5: Degrees of generalization of the learned language $L(H')$ with respect to the target language $L(H)$

Our experimental setup shown in Figure 4.6, consists of three modules responsible for trace generation, model learning and model evaluation. Since we are trying to evaluate how accurate the learning algorithm is, the initial model H, designed as a $DRTA^{+}$, is only known by the trace generator and the model evaluator, while the model learner has to guess it. The trace generator produces a timed learning sample S and a test sample. The latter contains timed traces that do not appear in S. This sample is used to evaluate the learned model with respect to new traces that were not used during the learning phase.

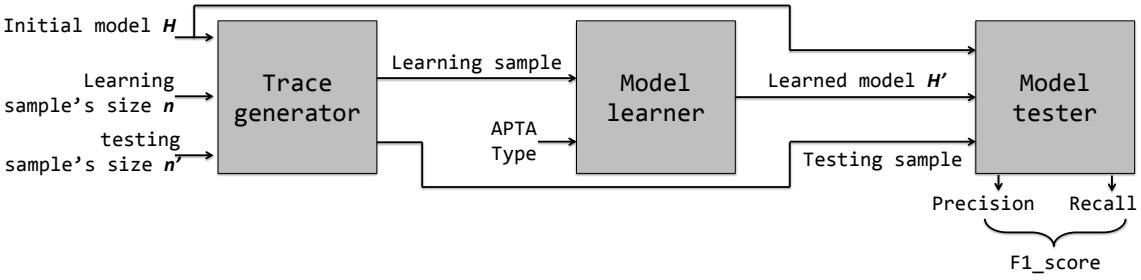


Figure 4.6: Experimental setup to validate the improved learning procedure

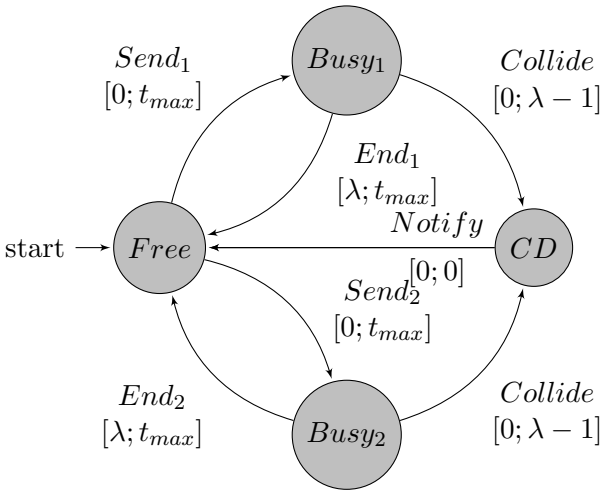


Figure 4.7: CSMA/CD communication medium model for a 2-station network

4.4.2 Benchmarks

We run our experimental setup on three examples, namely, *Periodic A*, *Periodic A-B* and *CSMA/CD communication medium model*.

Periodic A is a synthetic periodic task A that executes for 1 to 3 time units in a period of 5 time units. The goal of this benchmark is to check if the algorithm is able to learn the periodicity and the duration of a single task. Two less constrained variants of this model are also considered. In both of them, we remove the periodicity of the task by setting a predefined waiting time of 5 time units after the task A finishes. In the first variant, called aperiodic contiguous-time (**ap_cont A**), the execution time of the task A can take contiguous time values in $[1; 3]$. In the second one, called aperiodic disjoint (**ap_dis A**), the execution time takes the disjoint time values 0, 2 and 4. Our goal is to check if the algorithm is able to detect the unused time values 1 and 3.

Periodic A-B consists of two sequential tasks A and B, taking execution time values, respectively, in intervals [1;3] and [1;2] with a periodicity of 5 time units. In this example, the learning algorithm is faced with dependencies between clock constraints for the task A, the task B and their periodicities, which is a more complex setting.

CSMA/CD communication medium model is a media access control protocol for single-channel networks that solves data collision problems. We focus on the **CSMA/CD** communication medium model for a 2-station network presented in [103]. Figure 4.7 represents the underlying **CSMA/CD** communication medium model where λ represents the propagation time. We assume that t_{max} is the maximum time elapsing between two consecutive events.

4.4.3 Results

Experiments have been done on the described examples using a learning sample of size 200 and a test sample of size 1000.

4.4.3.1 The Synthetic Examples.

Table 4.2 summarizes the results for **periodic A** (and variants) and **periodic A-B**. Since all the learned models have a 100% recall, only the precision is discussed in the sequel. The obtained results show that the original RTI+ learns an under-approximating model with a poor precision for all the considered examples. In contrast, as shown by the F1 score, the exact model is learned using the unfolded APTA for **periodic A** and its variants. Both the constructive and the tightened-bound APTAs do not learn the exact **periodic A** model (although more accurate than RTI+). By analyzing the learned models, we see that they actually fail to identify the periodicity of task A. For **ap_cont A**, the constructive and the tightened-bound APTAs learn the exact model. However, for **ap_dis A**, the constructive-bound approach learns the exact model, while the tightened-bound one returns a model with a low precision since it does not detect the unused time values 1 and 3.

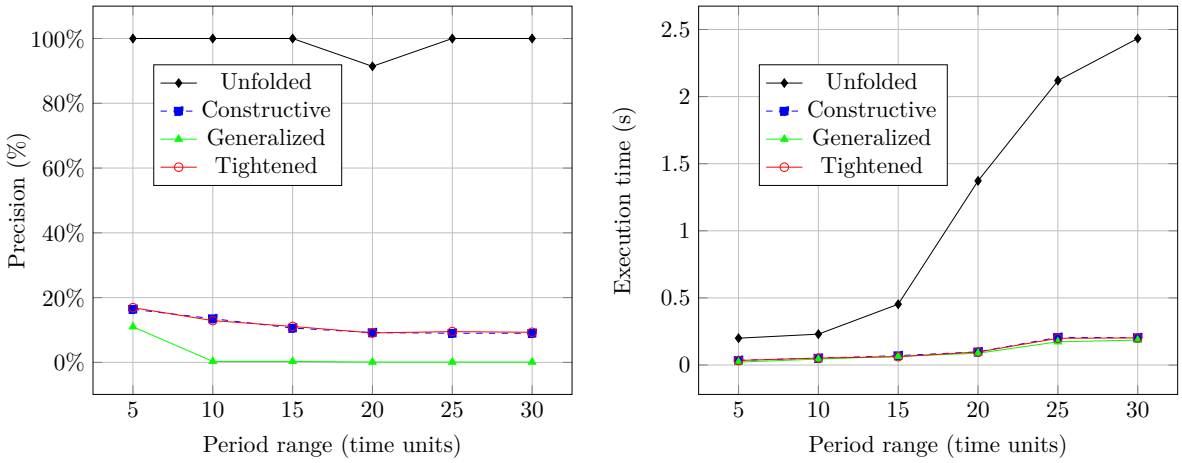
For the **periodic A-B** example, none of the variants was able to learn an accurate model: the obtained precision is at most 2.27% (using the constructive-bound APTA). They all fail to capture dependencies over clock constraints. Nevertheless, the precision is still better than the original RTI+ (0.18%).

Table 4.2: Accuracy results for the synthetic benchmarks with the four APTAs

Benchmark		Periodic A	Ap_dis A	Ap_cont A	Periodic AB
Generalized -bound (RTI+)	Precision	11%	0.8%	0.6%	0.18%
	Recall	100%	100%	100%	100%
	F1 score	0.1982	0.0159	0.0119	0.0036
Unfolded	Precision	100%	100%	100%	1.97%
	Recall	100%	100%	100%	100%
	F1 score	1.0000	1.0000	1.0000	0.0386
Constructive -bound	Precision	16.4%	100%	100%	2.27%
	Recall	100%	100%	100%	100%
	F1 score	0.2818	1.0000	1.0000	0.0444
Tightened -bound	Precision	16.9%	3.01%	100%	2.18%
	Recall	100%	100%	100%	100%
	F1 score	0.2891	0.0584	1.0000	0.0427

Figure 4.8 shows the impact of bigger time periods in the **periodic A** example on the quality of the learned model (Figure 4.8a) and the learning time (Figure 4.8b). We observe that increasing the period makes it more difficult to learn accurate models; Increasing the period

decreases the precision as shown in Figure 4.8a and increases the learning time as shown in Figure 4.8b. For instance, the original RTI+ with generalized-bound APTA is quite fast but its precision tends to zero. Using constructive and tightened-bound APTAs improves the precision with a similar learning time. Finally, relying on the unfolded APTA produces very precise models but induces an important learning time when the period exceeds 15 time units.



(a) Precision of the learned model

(b) Learning time

Figure 4.8: Impact of varying the task A period on the precision/the learning time

4.4.3.2 The CSMA/CD Example

Table 4.3 summarizes the experiments performed on **CSMA/CD**. On the one hand, one can notice that RTI+, like in the previous cases, learns an under-approximating model with a poor precision (6.20%) but in a short time ($\sim$ 6 s). Moreover, the generalized-bound APTA, initially having 2373 locations, is reduced to a final model with only 4 locations which represents a high reduction. On the other hand, the proposed APTAs produce significantly different models that cross-approximate the original **CSMA/CD**. For instance, the tightened-bound APTA learns a very precise model (93.70%). However, the model is obtained in more than 8 hours and has 370 locations. Using the constructive-bound APTA gives a model with less precision (85.80%) in a lower execution time ($\sim$ 3 h). Finally, the unfolded APTA gives a model with a 49.40% precision and a 96.70% recall, which corresponds to the best F1 score (0.6539). Furthermore, compared to constructive and tightened-bound APTAs, the learning time for the unfolded APTA is lower ($\sim$ 9 min). Hence, we conclude that, for this example, the unfolded APTA provides a good trade-off between accuracy and learning time.

Table 4.3: Experimental results for **CSMA/CD** using the four APTA models

APTA type	Precision	Recall	F1 score	Time	APTA size	DRTA size	Reduction
Generalized	6.20%	100.00%	0.1168	$\sim$ 6 sec	2373	4	99.83%
Unfolded	49.40%	96.70%	0.6539	$\sim$ 9 min	3586	19	99.47%
Constructive	85.80%	52.00%	0.6475	$\sim$ 3 hrs	2652	207	92.19%
Tightened	93.70%	49.90%	0.6512	$\sim$ 8 hrs	2373	370	84.41%

4.5 DRTA$^+$ to SRT-BIP Model Transformation

This section presents the relationship between the modeling formalism described in Chapter 3 and *DRTA*$^+$ models. More precisely, we focus on the procedure to follow in order to transform annotated *DRTA* models, learned by our improved learning algorithm proposed in Section 4.3, to SRT-BIP.

As defined earlier in this chapter, a *DRTA*$^+$ exhibits both probabilistic and timed behaviors. Let $\mathcal{A} = \langle \Sigma, L, l_0, \mathcal{C}, T, inv \rangle$ be a *DRTA* and $\mathcal{N}$ its annotation function. The simulation of such models consists first in identifying the transition to perform, and its associated action. Let the primitive $T(l)$ denote the subset of transitions in T available at location $l \in L$, i.e., $T(l) = \{t \in T \mid t = \langle l, \sigma, I, l' \rangle,\ \sigma \in \Sigma,\ I \in \mathcal{I},\ l' \in L\}$. The strategy φ defines the probability to select a transition t at location l, and is obtained by normalizing the annotation function over the set of available transitions, as follows:

$$\varphi(l,t) = \begin{cases} 0, & \text{if } t \notin T(l) \\ \dfrac{\mathcal{N}(t)}{\sum\limits_{t' \in T(l)} \mathcal{N}(t')}, & \text{if } t \in T(l) \end{cases}$$

Once a transition is identified, its firing date is computed by uniformly selecting a time value among the valid ones described by the time interval I of the selected transition. Besides the two-step interpretation of this model, time is modeled by a single discrete clock that is reset on every transition. Finally, *DRTAs* allow one to have several transitions labeled with the same symbol at the same location (i.e., non-determinism on input symbols) as long as their time constraints are disjoint.

Except for the non-determinism on input symbols, *DRTAs* and SRT-BIP models are quite similar at the syntax level. However, they drastically differ in their interpretation. Unlike *DRTA*, the simulation algorithm of SRT-BIP first samples remaining lifetimes to identify the next interaction to execute. A discrete choice is only made if this identification assigns the minimal remaining lifetime to more than one interaction (see Section 3.1.5).

The model transformation described in Definition 4.5.1 takes into account the characteristics of $(\mathcal{A}, \mathcal{N})$ when building the corresponding SRT-BIP model B. It is worth mentioning that

learned *DRTA* models are always one automaton. Hence, this transformation produces a monolithic SRT-BIP model with only one component. In the following, states (resp. transitions) of B are marked with the superscript $\tilde{l}$ (resp. $\tilde{t}$).

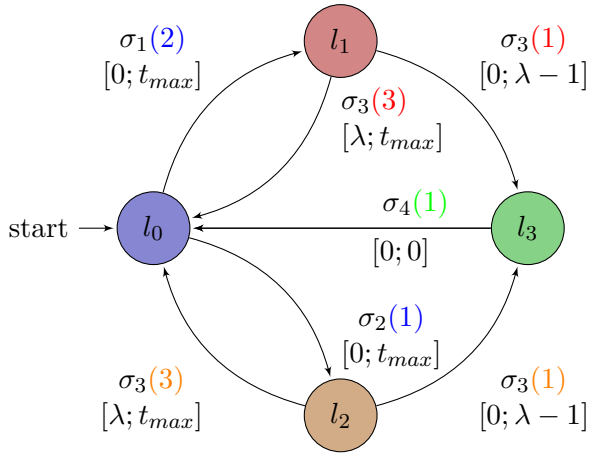
Definition 4.5.1 (*DRTA*$^+$ to SRT-BIP Transformation)**.** Given a *DRTA* and its annotation function ($\mathcal{A} = \langle \Sigma, L, l_0, \mathcal{C}, T, inv \rangle, \mathcal{N}$), a *DRTA*$^+$ to SRT-BIP transformation is a function that produces an SRT-BIP model $B = (\tilde{L}, X, P, \tilde{T}, \tilde{l}_0)$ and a weight function W such that:

- $\tilde{L} = \bigcup_{\langle l,\sigma,I,l' \rangle \in T} \{\tilde{l}_{(l,\sigma,l')}\} \cup L$, is the set of locations, and $\tilde{l}_0 = l_0$, is the initial location,
- $X = \{x\}$, a singleton set containing the clock x,
- $P = \bigcup_{\langle l,\sigma,I,l' \rangle \in T} \{s_{(l,\sigma,l')}\} \cup \Sigma$ is the set of ports,
- $\tilde{T} = \tilde{T}_1 \cup \tilde{T}_2$, such that for each $t = \langle l, \sigma, I = [a; b], l' \rangle \in T$:
  - $\tilde{T}_1 = \tilde{T}_1 \ \cup \ \{\ \tilde{t}_1 = (l, s_{(l,\sigma,l')},\ [true]^{\varepsilon},\ \{x\},\ \tilde{l}_{(l,\sigma,l')}\ \}$, and $W(\tilde{t}_1) = \mathcal{N}(t)$.
  - $\tilde{T}_2 = \tilde{T}_2 \cup \{\ \tilde{t}_2 = (\tilde{l}_{(l,\sigma,l')},\ \sigma,\ [a \leq x \leq b]^{d},\ \emptyset,\ l'\ \}$, and $W(\tilde{t}_2) = 1$.

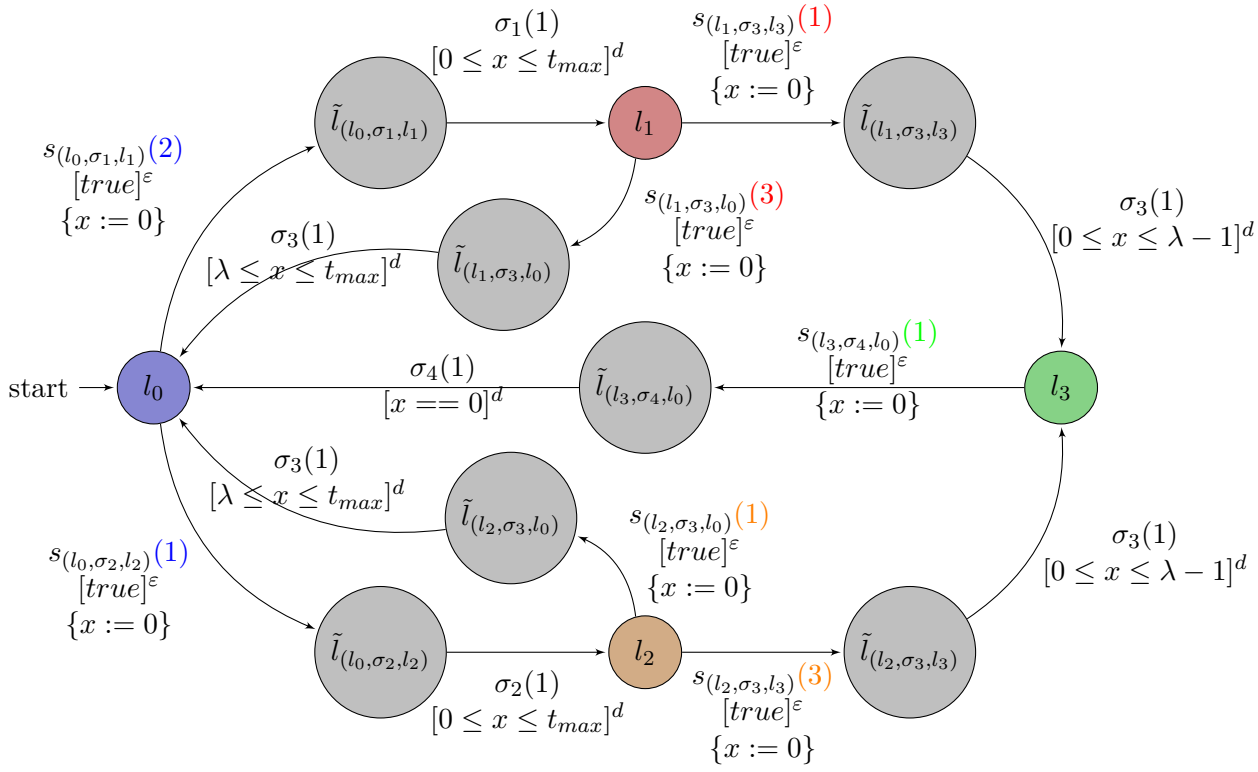
To enforce the two-step interpretation of $\mathcal{A}$, each of its transitions ($t = \langle l, \sigma, I = [a; b], l' \rangle \in T$) corresponds to two SRT-BIP transitions $\tilde{t}_1$ and $\tilde{t}_2$. Transition $\tilde{t}_1$ encodes the discrete choice of the transition. This is done in SRT-BIP by assigning it an *eager* urgency with no time constraints. Transition $\tilde{t}_2$ simulates the selection of the time value and represents the actual execution of t. It is worth mentioning that $\tilde{t}_2$ is *delayable*, that is, it can be delayed but not be ignored. Its time constraint is deduced from I as $[a \leq x \leq b]$, where x is the unique clock of B. Conforming to the *DRTA* definition, this clock is reset to zero before the selection of the time values, namely on transition $\tilde{t}_1$.

Model B includes all the locations L and ports (symbols) Σ from $\mathcal{A}$. In addition, for each transition t of $\mathcal{A}$, an intermediate location $\tilde{l}_{(l,\sigma,l')}$ (respectively a sampling port $s_{(l,\sigma,l')}$) is added to the set of locations $\tilde{L}$ (respectively the set of ports P). These additional locations and ports are labeled with the information of their corresponding transition t, namely, the source and target locations and the symbol. This labeling allows: (1) to keep track of the matching between *DRTA* transitions and the added entities, and (2) to conserve information about the non-determinism on input symbols.

Example 4.5.1. Figure 4.9 illustrates the model transformation on a *DRTA*$^+$ $\mathcal{A}$ depicted in Figure 4.9a. The resulting SRT-BIP model B is shown in Figure 4.9b. Each *DRTA* transition is transformed into a sampling and an execution SRT-BIP transitions, together with its corresponding intermediate state and sampling port. For example, transition $t = \langle l_0, \sigma_1, [0; t_{max}], l_1 \rangle$, annotated $\mathcal{N}(t) = 2$, corresponds to the sampling transition $\tilde{t}_1 = \langle l_0, s_{(l_0,\sigma_1,l_1)},\ [true]^{\varepsilon},\ \{x\},\ \tilde{l}_{(l_0,\sigma_1,l_1)} \rangle$, weighted $W(\tilde{t}_1) = 2$, and the execution transition $\tilde{t}_2 = \langle \tilde{l}_{(l_0,\sigma_1,l_1)},\ \sigma_1,\ \emptyset,\ l_1 \rangle$, weighted $W(\tilde{t}_2) = 1$. The processing of t introduces the intermediate location $\tilde{l}_{(l_0,\sigma_1,l_1)}$ and the sampling port $s_{(l_0,\sigma_1,l_1)}$, both labeled with information about t.



(a) *DRTA*$^+$ model $\mathcal{A}$



(b) Converted SRT-BIP model B

Figure 4.9: Example of a model transformation

Structurally, $\mathcal{A}$ contains four locations connected through seven transitions which are defined over an alphabet of four symbols. Model B contains a total of eleven locations with four locations coming from $\mathcal{A}$ and seven intermediate locations (colored in gray), that is, one per transition in $\mathcal{A}$. Also, B has twice as many transitions as $\mathcal{A}$.

It is worth mentioning that $\mathcal{A}$ exhibits non-determinism on the symbol σ_3 at location l_1. Transforming such transitions results in the generation of two transitions labeled by the sampling ports $s_{(l_1,\sigma_3,l_0)}$ and $s_{(l_1,\sigma_3,l_3)}$. Therefore, the non-determinism is removed, as required by the SRT-BIP language, even though all the information are still conserved thanks to the labeling of intermediate states and sampling ports.

Comparison of the accepted languages. In Definition 4.5.1, every location l of $\mathcal{A}$ corresponds to itself in B, noted $\tilde{l}$. Also, being at the location $\tilde{l}$ corresponding to l, if l' is reachable in $\mathcal{A}$ by timed symbol (σ, τ) from l then its equivalent location $\tilde{l'}$ is reachable in B by port σ and time τ. More precisely, l' being reachable by timed symbol (σ, τ) from l implies the existence of a transition $t = \langle l, \sigma, I, l' \rangle \in T$, with $\tau \in I$. By construction, there exist in B two transitions $\tilde{t}_1 = \langle \tilde{l},\ s_{(l,\sigma,l')},\ [true]^{\varepsilon},\ \{x\},\ \tilde{l}_{(l,\sigma,l')} \rangle$ and $\tilde{t}_2 = \langle \tilde{l}_{(l,\sigma,l')},\ \sigma,\ [x \in I]^{d},\ \emptyset,\ \tilde{l'} \rangle$, and therefore $\tilde{l'}$ is reachable by (σ, τ).

The main difference between the *DRTA*$^+$ model and its corresponding SRT-BIP model lies in the framework describing time progression. The former has a discretized time progression while the latter expresses time through clock variables defined over dense time domains. Hence, a given path $\pi = l_0 \xrightarrow{\sigma_1,I_1} l_1 \xrightarrow{\sigma_2,I_2} l_2 \ ... \xrightarrow{\sigma_n,I_n} l_n$ in $\mathcal{A}$ would generate $\prod_i |I_i|$ different time sequences for the same untimed word $\omega = \sigma_1\sigma_2 \ ...\sigma_n$. The cardinality $|I_i|$ represents the number of discrete time values included in the time interval $I_i = [a; b]$, namely, $|I_i| = b - a + 1$. On a similar path $\tilde{\pi}$, B would generate an infinite number of time sequences since $\tilde{I}_i$ is dense. Such an interval obviously includes the discrete time values generated by $\mathcal{A}$.

Simulating an SRT-BIP model obtained from a *DRTA* alternates sampling and execution ports (representing *DRTA* symbols). These sampling ports correspond to an implicit step in the *DRTA*$^+$. Therefore, we consider sampling ports to be non-observable in the produced SRT-BIP traces.

Consequently, the language accepted by $\mathcal{A}$ is included in the language generated by B, that is, every trace of $\mathcal{A}$ is a trace of B. Furthermore, interpreting SRT-BIP model over discrete time would result in both models $\mathcal{A}$ and B to accept exactly the same language.

4.6 Conclusion

In this chapter, we proposed different variants of the RTI+ algorithm for learning models with both stochastic and timed behaviors. We formally defined three APTA models with different levels of generalization and representation sizes. We validated our proposals with a variety of experiments that showed that using the new APTA variants produces more accurate

models regarding time. However, we observed that a higher learning time is generally required, depending on the desired accuracy. We also presented a systematic transformation that builds SRT-BIP models, introduced in Chapter 3, from the learned $DRTA^{+}$. We identified that work still has to be done on learning timed models in an efficient and convergent manner.

In the next chapter, we present our formal framework, denoted $\mathcal{S}$BIP, for the modeling and analysis of real-time systems exhibiting stochastic behaviors.

Chapter 5

The $\mathcal{S}$BIP Framework

In the previous chapters, we presented the SRT-BIP modeling formalism and how to learn this kind of models from execution traces. The usefulness of a modeling language is tightly related to its support tools and the variety of the analyses that can be done on the built models. The $\mathcal{S}$BIP framework [122] was initially proposed for the modeling of stochastic systems and their analysis with Statistical Model Checking (SMC) techniques. This framework was restricted to discrete time models and limited to specifications expressed in a subset of Bounded LTL where operator nesting was not allowed.

In this chapter, we present our $\mathcal{S}$BIP2 framework [117], that extends the previous version with a more expressive modeling language SRT-BIP, richer specification languages and a larger set of analyses. First, this new modeling language puts forward the modeling of uncertainty in a context where time progresses continuously. Secondly, $\mathcal{S}$BIP2 now supports the operator nesting in bounded LTL but also a timed version of it, namely the Metric Temporal Logic (MTL). In addition to hypothesis testing and probability estimation, the tool's capabilities are enriched with rare events analysis and an automated and compact way to explore a family of properties, namely, parametric exploration.

$\mathcal{S}$BIP2 is the subject of a completely new development and is implemented as an Integrated Development Environment for (SRT-)BIP models. Our framework is supported by a tool that centralizes the model edition, compilation, simulation, code generation and analysis, and enhances the user experience with a Graphical User Interface. For simplicity, in the rest of the chapter, we refer to this new implementation as $\mathcal{S}$BIP, while references to the previous version of the framework are explicitly pointed out.

The remainder of the chapter is organized as follows. In Section 5.1, we present the tool's modular design. Section 5.2 introduces the workflows and activities integrated in $\mathcal{S}$BIP. We give its implementation details together with its availability and documentation in Section 5.3. Finally, we discuss the tool's capabilities and usability compared to state-of-the-art tools in Section 5.4.

5.1 $\mathcal{S}$BIP Design

The new version of $\mathcal{S}$BIP was completely re-designed as an IDE including all the activities from the modeling, to the simulation and the SMC analysis. It offers a set of functionalities organized in a clear and fluid workflow as illustrated in Figure 5.1. All interactions with $\mathcal{S}$BIP go through a graphical user interface (GUI), which allows for setting the inputs, running analysis and getting the outputs.

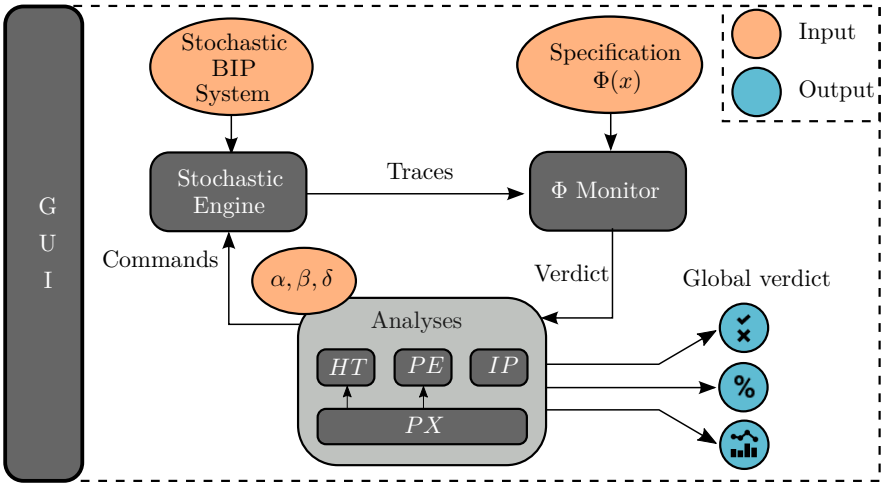


Figure 5.1: $\mathcal{S}$BIP architecture

The tool architecture was designed modularly for more flexibility and to enable extensibility. The tool relies on five main generic functional modules, namely, a *Stochastic Simulation Engine*, a *Monitoring* module, an *SMC Engine*, a *Rare-Event Engine*, a *Parametric Exploration* module plus additional data structures, i.e., to represent execution traces and logical formulas. The stochastic engine encapsulates an executable model simulator and is used to produce (random) execution traces on-demand. The monitor is used to evaluate properties on traces. The SMC engine implements the main statistical model checking loop depending on the statistical method used, namely, hypothesis testing or probability estimation. Finally, the parametric exploration module coordinates the evaluation of a parametric property. All these modules are fully independent and interact through well-defined Java interfaces.

The tool has been instantiated for the BIP formalism as input model, where different stochastic simulation engines can be used (discrete/real-time). Regarding monitoring, the tool currently supports parametric bounded LTL and MTL.

5.1.1 Stochastic Simulation Engine

Currently, $\mathcal{S}$BIP supports the use of two different stochastic simulators, namely, for classical stochastic BIP [122] that enables to model discrete-time systems (DTMCs) and for the Stochastic Real-Time BIP (presented in Section 3) for continuous-time systems with general distributions

(GSMPs and CTMCs)[1]. The former produces untimed traces needed to verify bounded LTL properties (and to guarantee backward compatibility), whereas the latter generates timed traces necessary to verify MTL properties. We implemented simulators to produce traces in different modes, i.e., symbol-wise, piece-wise and trace-wise. We use the first mode for on-line monitoring and to be able to interrupt simulations as soon as a verdict is obtained. The second is primordial for rare events analysis and allows one to generate traces as a concatenation of trace-fragments. Finally, we use the third mode for off-line monitoring, and to generate traces of a given length.

In terms of analyses, the simulation mode impacts both the analysis time and its result. The trace-wise simulator controls the SMC analysis time by bounding the trace length. However, this length must be determined carefully: it can be insufficient to conclude a verdict for some properties. For example, the verification of a BIP system with traces of length 5 against an LTL property $\Box_{\{10\}}\phi$ may result in non-informative verdicts. It is worth mentioning that we adopt a pessimistic policy when dealing with non-informative verdicts, by considering them to eventually evaluate to false. The symbol-wise simulator ensures informative verdicts (true or false) but does not provide any mechanism to control the analysis time. The simulation only ends when a local verdict is reached by the monitor. It is guaranteed to end by the fact that we only allow for bounded (resp. time-bounded) LTL (resp. MTL) properties.

This module being designed as a standalone (as described in Section 3.2), it can be used independently from the tool. We illustrated it by interfacing our piece-wise simulator with the Plasma-Lab [88] tool in order to perform rare events analyses. This required the development of a Java wrapper to that BIP simulator for the implementation of the methods required by the `Simulator` interface of Plasma-Lab, as shown in Figure 5.2.

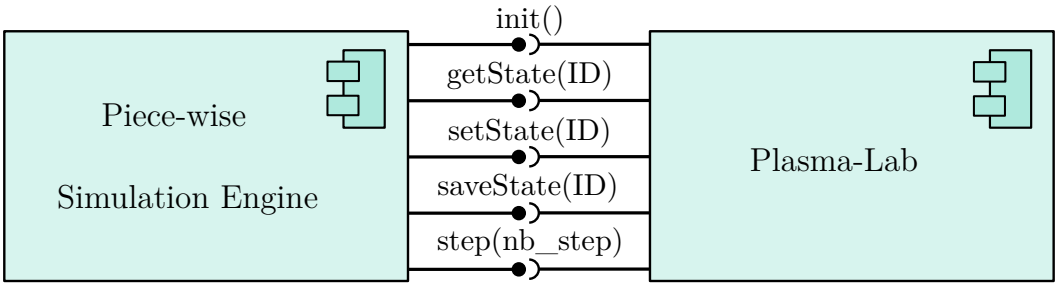


Figure 5.2: BIP engine wrapper

The wrapper allows Plasma-Lab to perform the elementary operations on a BIP model for an importance splitting analysis, namely, saving, restoring and getting a representation of a BIP global state, in addition to advancing the simulation by a given number of steps. Please note that $\mathcal{S}$BIP also possesses its own implementation of the *IP* analysis (see Section 5.1.3 for more details).

[1]SRT-BIP sources are available at https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/bip/compiler/tree/stochastic-real-time

5.1.2 Monitoring Properties

This module implements the generic infrastructure for on-line/off-line monitoring of properties. At abstract level, the module takes as inputs a formula and either an entire trace or an on-line stream of trace symbols, and computes a verdict stating whether the trace satisfies the formula. Traces, formulas and symbols are designed as Java interfaces that can be extended with specific implementations.

In the current version of $\mathcal{S}$BIP, we integrate the monitoring of Bounded LTL and MTL formulas on untimed and timed BIP traces. Bounded LTL was already included in the first version of the tool. However, it was restricted to formulas without nested temporal operators. At contrary, the monitoring of MTL formulas represents a completely new development. Both LTL and MTL monitors implement online monitoring algorithms based on rewriting rules, that we detail hereafter for the monitoring of MTL properties.

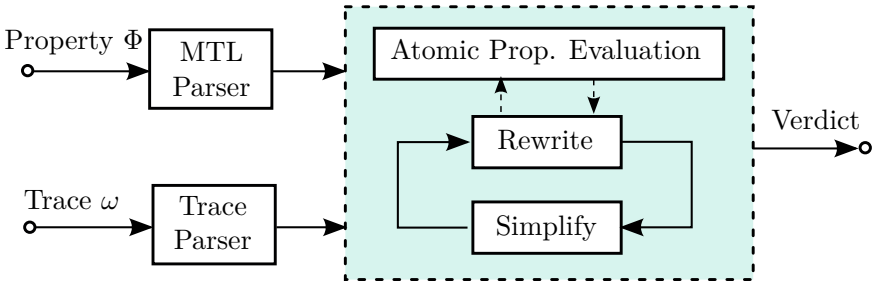


Figure 5.3: Functional view of the MTL Monitor

Given an MTL formula ϕ and a timed trace (ω, τ), the MTL monitor, illustrated in Figure 5.3, alternates rewriting and simplification phases. Rewriting consumes a timed symbol $\lambda_i = (\sigma_i, \tau_i)$ of the timed trace and partially evaluates the current formula ϕ into ϕ'. Partial evaluation includes the unfolding of temporal operators and evaluation of atomic state formulas to their truth value. In practice, a timed symbol represents an actual system state at global time τ_i, and is described by the valuation of variables and clocks when entering this state. We write $\sigma_i \models p$ to indicate that the i^{th} system state satisfies the proposition p. Definition 5.1.1 describes the rewriting rules from [38].

Definition 5.1.1. Given a formula ϕ defined over the set of propositions $\mathcal{P}$, a system state σ and the delay v before the arrival of the next state change, *rewrite* is a recursive function producing a formula ϕ', as follows:

- $rewrite(p,\ \sigma,\ v) = \begin{cases} \mathbf{t}, & \text{if } \sigma \models p \\ \mathbf{f}, & \text{if } \sigma \not\models p \end{cases}$

- $rewrite(\neg p,\ \sigma,\ v) = \begin{cases} \mathbf{f}, & \text{if } \sigma \models p \\ \mathbf{t}, & \text{if } \sigma \not\models p \end{cases}$

- $rewrite(\phi_1 \wedge \phi_2,\ \sigma,\ v) = rewrite(\phi_1, \sigma, v)\ \wedge\ rewrite(\phi_2, \sigma, v)$

- $rewrite(\phi_1 \vee \phi_2,\ \sigma,\ v) = rewrite(\phi_1, \sigma, v)\ \vee\ rewrite(\phi_2, \sigma, v)$

- $rewrite(\bigcirc\phi,\ \sigma,\ v) = \phi$

- $rewrite(\phi_1\ \mathcal{U}_{[a,b]}\ \phi_2,\ \sigma,\ v) =$

$$\begin{cases} rewrite(\phi_1,\ \sigma,\ v) \wedge \phi_1\ \mathcal{U}_{[max(a-v,\ 0),\ b-v]}\ \phi_2, & \text{if } a > 0 \wedge v \leq b \\ rewrite(\phi_2,\ \sigma,\ v) \vee (rewrite(\phi_1,\ \sigma,\ v) \wedge \phi_1\ \mathcal{U}_{[0,\ b-v]}\ \phi_2), & \text{if } a = 0 \wedge v \leq b \\ rewrite(\phi_2,\ \sigma,\ v), & \text{if } a = 0 \wedge v > b \\ \mathbf{f}, & \text{if } a > 0 \wedge v > b \end{cases}$$

- $rewrite(\phi_1\ \mathcal{R}_{[a,b]}\ \phi_2,\ \sigma,\ v) =$

$$\begin{cases} rewrite(\phi_2,\ \sigma,\ v) \wedge \phi_1\ \mathcal{R}_{[max(a-v,\ 0),\ b-v]}\ \phi_2, & \text{if } a > 0 \wedge v \leq b \\ rewrite(\phi_2,\ \sigma,\ v) \wedge (rewrite(\phi_1,\ \sigma,\ v) \vee \phi_1\ \mathcal{R}_{[0,\ b-v]}\ \phi_2), & \text{if } a = 0 \wedge v \leq b \\ rewrite(\phi_2,\ \sigma,\ v), & \text{if } v > b \end{cases}$$

In this definition, the i^{th} delay v_i is computed as the difference between the global time of system states σ_{i+1} and σ_i, i.e., $v_i = \tau_{i+1} - \tau_i$. The resulting property identifies what remains to be observed in the trace before being able to conclude a verdict. For example, for a property $\phi = (x > 0)\ \mathcal{U}_{[0,5]}\ (y < 0)$ and a trace $(\{x = 5; y = 2\},\ 0)(\{x = -1; y = -1\},\ 3)$, consuming the first timed symbol would result in :

$$\begin{aligned} rewrite(\phi, \{x = 5; y = 2\},\ v = 3) &= rewrite((y < 0), \{x = 5; y = 2\},\ 3)\ \vee \\ &\quad (rewrite((x > 0),\ \{x = 5; y = 2\},\ 3) \wedge (x > 0)\ \mathcal{U}_{[0,2]}\ (y < 0)) \\ &= \mathbf{f}\ \vee\ (\mathbf{t}\ \wedge (x > 0)\ \mathcal{U}_{[0,\ 2]}\ (y < 0)) = \phi' \end{aligned}$$

This can be read as: at the current observation, we either have that $y > 0$, or $x > 0$ still holds and we still have to observe the *Until* on a window of 2 time units.

It is worth mentioning that we corrected the rewriting rules from [38], by fixing the computation of the time bounds for the rewriting of *Until* and *Release* operators in the case of $a > 0$ and $\tau \leq b$ for which we substitute the *min* function by a *max*.

Simplification applies reduction rules on the formula ϕ' based on Boolean logic (e.g., $(\boldsymbol{t}\ \wedge\ \phi') \equiv \phi'$) so as to conclude or to simplify it as much as possible before the next cycle. The simplification rules [38] are given in Definition 5.1.2 below.

Definition 5.1.2. Given a formula ϕ defined over the set of propositions $\mathcal{P}$, $simplify$ is a recursive function producing a formula ϕ', as follows:

- $simplify(\phi_1 \wedge \phi_2) =$

$$\begin{cases} \mathbf{f}, & \text{if } simplify(\phi_1) = \mathbf{f} \vee simplify(\phi_2) = \mathbf{f} \\ simplify(\phi_1), & \text{if } simplify(\phi_2) = \mathbf{t} \\ simplify(\phi_2), & \text{if } simplify(\phi_1) = \mathbf{t} \\ simplify(\phi_1) \wedge simplify(\phi_2), & \text{otherwise} \end{cases}$$

- $simplify(\phi_1 \vee \phi_2) =$

$$\begin{cases} \mathbf{t}, & \text{if } simplify(\phi_1) = \mathbf{t} \vee simplify(\phi_2) = \mathbf{t} \\ simplify(\phi_1), & \text{if } simplify(\phi_2) = \mathbf{f} \\ simplify(\phi_2), & \text{if } simplify(\phi_1) = \mathbf{f} \\ simplify(\phi_1) \vee simplify(\phi_2), & \text{otherwise} \end{cases}$$

- $simplify(\phi) = \phi$, otherwise

Simplification works in a straightforward manner. For example, simplifying $\phi' = \mathbf{f} \vee (\mathbf{t} \wedge (x > 0)\ \mathcal{U}_{[0,\ 2]}\ (y < 0))$ would result in $\phi' = (x > 0)\ \mathcal{U}_{[0,\ 2]}\ (y < 0)$.

For the sake of usability, we allow expressing parametric LTL/MTL formula $\phi(x)$, where x is an integer parameter taking values in some bounded domain Π. The parameter can appear either in a state formula or as a bound (of time intervals I) and is statically assigned a value from its domain before starting an analysis (see Section 5.1.3.1 for more details). For instance, $\phi(t) = \Diamond_{[0,t]}[(node_3.status = leader)]$ states that $node_3$ eventually becomes the leader before t time units, where t is the parameter of the property.

5.1.3 Statistical Analyses Core

In addition to classical SMC algorithms, i.e., *HT* [151] and *PE* [79], our framework proposes two additional analyses for **(1)** the exploration of properties parameters, namely, *Parametric eXploration (PX)*, and for **(2)** rare events analysis, namely, *Importance Splitting (IP)* [89]. *HT* answers qualitative queries, i.e., given a stochastic system S and a property ϕ, it enables one to assess whether the probability for S to satisfy ϕ is greater or equal to a given threshold θ. *PE* addresses quantitative queries, that is, to compute a probability estimate p for S to satisfy ϕ.

Hereafter, we detail the parametric exploration *PX* and *IP* for rare events analysis. In addition, we propose an algorithm for analyzing rare properties based on a simulation guided by the property under study. Similarly to *Importance Sampling (IS)*, this algorithm alters the probability to select transitions of the model in order to increase the likelihood to observe rare events. But unlike *IS*, the tilting strategy is built on the fly throughout the guided simulation. The property under study is monitored at runtime by a rewrite-based monitor. This monitor builds a set of symbols that remain to be observed at the next step, and that have to be prioritized during simulation. Note that this algorithm is part of an ongoing work and has not yet been the subject of experiments.

5.1.3.1 Parametric Exploration

Parametric exploration is an automatic way to perform statistical model checking on a family of properties that differ by the value of a constant. The family of properties is specified in a compact way as a parametric property $\phi(x)$, where x is an integer parameter ranging over a finite instantiation domain Π. Algorithm 5 illustrates the different phases of a parametric exploration. The algorithm returns a set of SMC verdicts corresponding to the verification of the instances of $\phi(x)$ with respect to $x \in \Pi$. It is very useful for exploring unknown system parameters such as, buffers sizes guaranteeing no overflow, or the amount of consumed energy. It automates the exploration for large parameters domains as opposed to tedious and time consuming manual procedures.

```
Data: system S, parametric property φ(x),
      instantiation domain Π
Result: A set of SMC verdicts V
Monitor m; Engine e; V = ∅;
foreach v ∈ Π do
    smc.init();
    while !smc.conclude() do
        tr = e.generate(S);
        verdict = m.check(φ(v), tr);
        smc.add(tr, verdict);
    end
    V = V ∪ smc.getVerdict() ;
end
```

Algorithm 5: Parametric exploration

5.1.3.2 Importance Splitting

$\mathcal{S}$BIP provides an implementation of the *IP* algorithm that relies on the piece-wise simulator to analyze rare properties. These properties specify requirements where some rare event eventually happens but with very low probability, and are often of the shape $\Diamond_{[0,t]}\varphi$.

Importance splitting [89] overcomes the problem of estimating the probability $P(S \models \phi)$ of a system S to satisfy these properties. This is done by considering a set of intermediate levels l_i that correspond to less rare properties ϕ_i, s.t., $\phi_n \Rightarrow \phi_{n-1} \Rightarrow \ldots \Rightarrow \phi_1$, where $\phi_n = \phi$. $P(S \models \phi)$ is therefore computed as the product of the conditional probabilities to reach l_i from l_{i-1}, i.e., $\Pi_{i=1}^{n} P(S \models \phi_i \mid S \models \phi_{i-1})$. In our implementation, the intermediate levels l_i and associated ϕ_i are defined via a score function given as input. More precisely, each level l_i is identified by a state formula ϕ_i and f returns the highest index of state formula that a system state s satisfies, i.e., $f(s) = Max_{\ l_i}\{i \in \mathbf{N} \mid s \models \phi_i\}$.

Our algorithm is similar to the analysis procedure proposed in Plasma-Lab. It iterates over levels, and for each one, it simulates m trace prefixes among which m_s reach the next level and m_f do not. The conditional probability to reach the next level is thus estimated as the ratio m_s/m. In the next iteration, the simulation of successful prefixes is resumed, while the rest (m_f) are replaced by successful ones sampled uniformly. Note that our implementation of *IP* is currently limited to the analysis of DTMCs.

5.1.3.3 Objective-Guided Simulation for Rare Events Analysis

We present a guided simulation algorithm with a state-dependent tilting scheme using an implicit representation of the tilting transition matrix. This algorithm takes advantage of the knowledge of the monitored property to guide the simulation towards useful traces. The introduced bias is computed on the fly and returned as the algorithm output.

The algorithm. Let $\mathcal{M}$ be a DTMC and S the set of its states. $s_0 \in S$ is the (unique) initial state of $\mathcal{M}$. The transition matrix π is a function $\pi : S \times S \rightarrow [0,1]$ such that $\forall s \in S, \Sigma_{s' \in S} \pi(s,s') = 1$. $L : S \rightarrow \Sigma$ is the state labeling function, and Σ is the alphabet. Execution paths in this model are sequences of the form $p = s_0 s_1 s_2 ... s_n$. The trace ω can be deduced from the path p and is of the form $\omega = L(s_0)L(s_1)L(s_2)...L(s_n)$. In a more general way, a trace is a sequence of symbols $\sigma_i = L(s_i)$. In the following, we restrict $\mathcal{M}$ to the deterministic case such that $\forall s \in S, \forall \sigma \in \Sigma$, it exists at most one state s' with $\pi(s,s') > 0 \wedge L(s') = \sigma$. We can then write the probability to observe symbol σ being at state s as $P(s,\sigma) = \Sigma_{s' \in S} \pi(s,s') \times \mathbf{1}[L(s') = \sigma]$, where $\mathbf{1}[.]$ is an indicator function returning 1 if its attribute is true, 0 otherwise.

Let ϕ be a bounded LTL property. A rewrite-based monitor for the property ϕ is an iterative process that, for each symbol $\sigma_i \in \omega$ of a given trace ω, rewrites ϕ_i to $\phi_{i+1} = simplify(rewrite(\phi_i, \sigma_i))$, starting with $\phi_0 = \phi$. The iterative process finishes after n steps (guaranteed by the fact that the property is bounded) when $\phi_n = \mathbf{t}$ or $\phi_n = \mathbf{f}$. One has to notice that intermediate formula ϕ_i represents the part of the formula ϕ that still has to be observed before being able to conclude the verdict for the evaluated trace. The function $Next : \Phi \rightarrow 2^{\Sigma}$ takes as parameter a property ϕ and returns the set of expected symbols. It is defined as follows:

- $Next(\sigma) = \{\sigma\}$ and $Next(\mathbf{t}) = Next(\mathbf{f}) = \emptyset$
- $Next(\phi_1 \vee \phi_2) = Next(\phi_1) \cup Next(\phi_2)$
- $Next(\phi_1 \wedge \phi_2) = Next(\phi_1) \cap Next(\phi_2)$
- $Next(\neg\phi) = \Sigma \setminus Next(\phi_1)$
- $Next(\bigcirc\phi) = \emptyset$
- $Next(\phi_1 \mathcal{U} \phi_2) = Next(\phi_1) \cup Next(\phi_2)$
- $Next(\Box\phi) = Next(\phi)$
- $Next(\Diamond\phi) = Next(\phi)$

Note that $Next(\bigcirc\phi)$ returns an empty set instead of the whole alphabet Σ. This is because no specific symbol is expected in particular since any symbol is accepted. Hence, a $\bigcirc$ (next) operator will have no impact on the guided simulation.

We want to use the information provided by this $Next$ function to guide the simulation and increase the likelihood to observe the rare property. Let $Enabled(s) = \{\sigma \in \Sigma : P(s, \sigma) > 0\}$ be the set of symbols enabled at the state s of $\mathcal{M}$, $s \in S$. Algorithm 6 details the process.

Data: A model $\mathcal{M}$, a property ϕ, and a parameter α
Result: A real value B in [0,1]
Initialize the current state s to s_0;
Initialize $B = 1$;
while $\phi \neq \boldsymbol{t} \wedge \phi \neq \boldsymbol{f}$ **do**
  Set of enabled transitions $E = Enabled(s)$;
  Set of expected events $N = Next(\phi)$;
  if $E \cap N = \emptyset$ **then**
    Select a symbol $\sigma \in E$ based on the function P;
  else
    // Take the rarest event;
    Float $min\_proba = 1$;
    foreach $\sigma' \in E \cap N$ **do**
      if $P(s, \sigma') < min\_proba$ **then**
        $min\_proba = P(s, \sigma')$;
        $\sigma = \sigma'$;
      end
    end
    Simulate X with a $Bernoulli(\alpha)$;
    if $X = 1$ **then**
      Select the rarest event σ;
      $B = B \times \frac{min\_proba}{\alpha}$;
    else
      Select a random symbol $\sigma \in E$ based on the function P;
    end
  end
  $s = s'$, such that $\pi(s, s') > 0 \wedge L(s') = \sigma$;
  $\phi = simplify(rewrite(\phi, \sigma))$;
end
if $\phi = \boldsymbol{f}$ **then**
  $B = 0$;
end
return B;

Algorithm 6: Monitor-based guided simulation algorithm

The algorithm is parametrized by α that determines the probability to select the rarest event when entering the guidance phase of the simulation. Values of α closer to 1 lead to more determinism in the simulation but increase the likelihood to generate the rare events, whereas smaller values of α reduce the introduced bias by guiding the simulation in a probabilistic way. Selecting a symbol $\sigma \in E$ based on the function P is done in the conventional way: build the

cumulative of P, sample a random value in [0,1] and select the symbol corresponding to it in the cumulative of P.

The algorithm distinguishes two cases. The first case is encountered when none of the enabled symbols are expected by the monitor. The selection of the σ is done based on the probabilities of the original model. The fact that the monitor is not expecting any of the enabled symbols does not necessarily mean that the property will evaluate to false. For example, with a property $\Diamond(a)$, the set of expected symbols is $Next(\phi) = \{a\}$ but observing b does not falsify it. On the contrary, the second case is characterized by an intersection between the set of enabled symbols and expected ones that is not empty. The algorithm selects the symbol σ corresponding to the lowest probability in this intersection, then probabilistically decides whether to tilt its probability to α, or to ignore it and simulate based on function P. This choice is made according to a *Bernoulli* distribution with parameter α.

During a run, the algorithm cumulates the bias introduced by the performed tilting, in a variable B that is returned at the end of the exploration. This real value lies in the interval $[0, 1]$ where the value 0 means that the generated trace does not satisfy the rare property, whereas any other value means that this trace does satisfy it.

Discussion. This algorithm bases its distribution tilting on an implicit transition matrix. Let s_i be the i^{th} state of a path p, and $X_i \sim Bernoulli(\alpha)$ be a random variable following a *Bernoulli* distribution. This biased transition matrix π' is then defined as :

$$\pi'(s_i, s') = \begin{cases} \pi(s_i, s'), & \text{if } Enabled(s_i) \cap Next(\phi_i) = \emptyset \vee X_i = 0 \\ \alpha, & \text{if } Enabled(s_i) \cap Next(\phi_i) \neq \emptyset \wedge \pi(s_i, s') = min\_proba(s_i) \wedge X_i = 1 \\ \frac{\pi(s_i,s') \times (1-\alpha)}{1-min\_proba(s_i)}, & \text{if } Enabled(s_i) \cap Next(\phi_i) \neq \emptyset \wedge \pi(s_i, s') \neq min\_proba(s_i) \wedge X_i = 1 \\ 0, & \text{otherwise.} \end{cases}$$

where $min\_proba(s_i) = min_{s \in S}\{\pi(s_i, s)\}$. The transition matrix is kept the same if the enabled symbols are not expected by the monitor but also if the simulation of the *Bernoulli* distribution rejects the probability change. Otherwise, the probabilities of transition enabled at state s_i are updated as follows: the probability of the rarest symbol σ is set to α while the probability of the remaining symbols is updated proportionally. In other words, when excluding σ, the proportion of the probability attached to an enabled symbol in the original model is preserved in the biased model.

Given this transition matrix, one can see that the probability ψ to take a path p in the original model $\mathcal{M}$ can be easily written as a function of its probability to be taken in the biased model $\mathcal{M}'$, as follows:

$$\begin{aligned}\psi(p,\mathcal{M}) = \ & \Pi_{i=0}^{n-1}\pi(s_i,s_{i+1}) \\ & = \Pi_{j\in J}\ \pi(s_j,s_{j+1}) \times \Pi_{i\notin J}\ \pi(s_i,s_{i+1}) \times \frac{\alpha^{|J|}}{\alpha^{|J|}} \\ & = \frac{\Pi_{j\in J}\ min\_proba(s_j)}{\alpha^{|J|}} \times \Pi_{i\notin J}\ \pi(s_i,s_{i+1}) \times \alpha^{|J|} \\ & = B \times \Pi_{i\notin J}\pi(s_i,s_{i+1}) \times \alpha^{|J|} = B \times \psi(p,\mathcal{M}')\end{aligned}$$

where J is the set of indexes of the steps that were biased during the guided simulation. The likelihood ratio is then $LR(p,\mathcal{M}) = \frac{\psi(p,\mathcal{M})}{\psi(p,\mathcal{M}')} = B$.

Hence, an SMC algorithm could use the output of this simulation to estimate the probability for $\mathcal{M}$ to satisfy ϕ by simulating m traces with Algorithm 6 and compute the empirical estimator:

$$\hat{\gamma} = \frac{1}{m}\Sigma_{i=1}^{m} B_i$$

Instead of choosing the rarest event, several alternatives can be proposed as a condition to the guiding phase of the algorithm:

1. One option is to only alter the next symbol choice if $min\_proba < \epsilon$, that is, only if the event has a low probability to occur (smaller than a given threshold ϵ). Hence, only rare events are tilted.

2. Another option would be to bound the relative error introduced by choosing the rarest σ. For example we say that we only alter the simulation if $rel\_error = \frac{\alpha - min\_proba}{min\_proba} < \delta$, with δ a bound depending on the SMC parameters. However, you can expect δ to be big in the case of events that are unlikely to be simulated. For instance, if $min\_proba = 10^{-3}$ and $\alpha = 10^{-1}$ then δ has to be greater than 99.

5.1.4 Graphical User Interface

We implemented a user-friendly graphical interface (GUI) that centralizes all the interactions with the tool. The GUI is organized in three main regions: (1) a project explorer, (2) a toolbar and (3) a central panel, as illustrated in Figure 5.4.

The project explorer gives a centralized and organized view of the different items of a project during the modeling and the analysis. Such items usually include different files organized in a tree hierarchy. The *Models* folder contains (.bip) models, external (.cpp/hpp) source code, custom probability distributions, and executables. The *Properties* folder stores (.mtl) and (.ltl) properties, and also scoring functions (.sf) for *Importance Splitting*. Finally, the *Outputs* folder contains the execution traces generated during analyses.

The toolbar is organized in six functional groups: (*i*) project management, allowing the designers to create/remove projects, (*ii*) file management for model files creation, deletion,

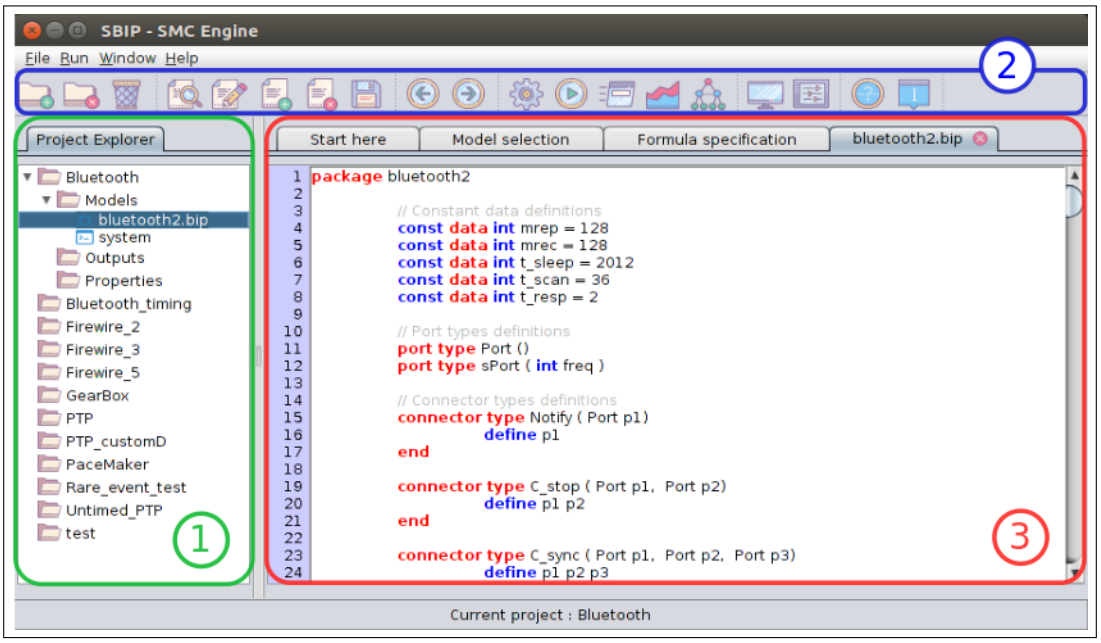


Figure 5.4: Screen-shot of $\mathcal{S}$BIP graphical user interface

edition and visualization, (*iii*) tab navigation, (*iv*) workflow management for model compilation, simulation and analysis, (*v*) configuration setup, and (*vi*) help buttons.

The central panel is the main region where the designer can load/visualize/edit inputs, configure parameters, run analyses, and visualize results. Each of these operations is provided through a specific view displayed in a separate tab:

- edition view: used to edit models and properties. This also allows for loading and saving various files. Specific capabilities, such as code auto-completion and keyword highlighting, are provided for BIP models.
- configuration view: used to select the simulation engine, the SMC algorithm and parameters, and the instantiation domain for parametric properties.
- analysis view: used to initiate and track the progress of analysis.
- results view: used to provide a summary of the performed analysis, the verdict and/or the set of verdicts on different traces, specific curves and/or plots, overall and partial running times, etc.

5.2 Integrated Workflows and Activities

The tool takes as inputs a stochastic system model to be analyzed/simulated, a property of interest, and a set of parameters mainly required by the SMC algorithms. Three analysis workflows are provided by $\mathcal{S}$BIP , namely, the classical SMC, parametric exploration and

the rare events analysis. These workflows are dependent on activities of model edition and debugging, and requirement expression.

In this section, we detail the activities and workflows that are integrated in $\mathcal{S}$BIP.

5.2.1 Design Activities

5.2.1.1 Model Construction

Building a system model is an iterative process that requires the alternation between several activities such as model edition, simulation and debugging. For $\mathcal{S}$BIP, this process is described in Figure 5.5.

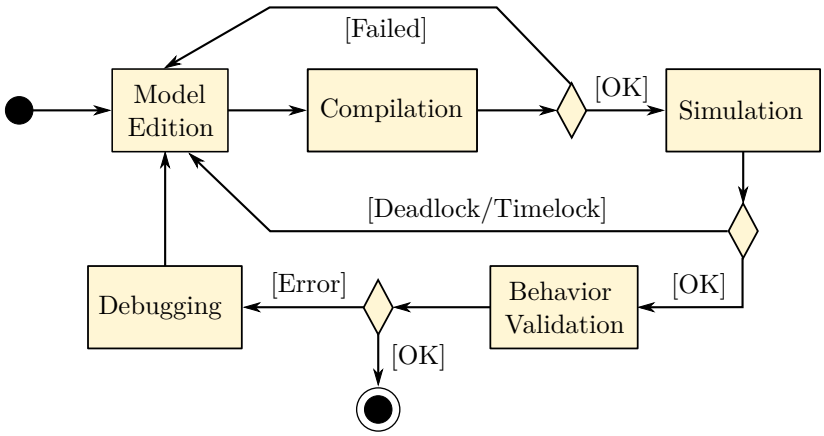


Figure 5.5: Model construction diagram

Model edition. The tool provides several functionalities regarding model edition. Indeed, the designer can create, edit, save and remove model files including BIP models and external C++ code but also distribution files describing customized probability distributions. In addition, the model edition is helped by the editing features provided by the tool, such as, code auto-completion, keyword coloring and highlighting, and find/replace and undo/redo capabilities.

Compilation. The compilation checks that the input model complies with the BIP syntax. The outcome of this activity is an executable if the model is syntacticly correct, or it triggers errors otherwise. The compilation process is described in a bash script that can be also edited through the GUI.

Simulation and functional validation. The simulation consists in running the system executable in order to validate that this system behaves as expected by observing concrete execution traces. During a simulation, the designer can read the sequence of

decisions taken by the (SRT-)BIP engine and assess the system behavior. This activity may identify two types of errors: either deadlocks/timelocks interrupt the system simulation, or the designer detects a non-expected sequence of actions. Both errors redirect the designer back to the model edition but a debugging is sometimes required to identify the source of the error.

Simulation traces can be displayed in different forms depending on the chosen simulation arguments. For the SRT-BIP engine, the option "–log-stoch-choice" shows in details the decisions taken by the stochastic simulation algorithm. Also, "–log-variable" allows one to display the value of all the data variables. For an exhaustive list of arguments, one can use the option "–help".

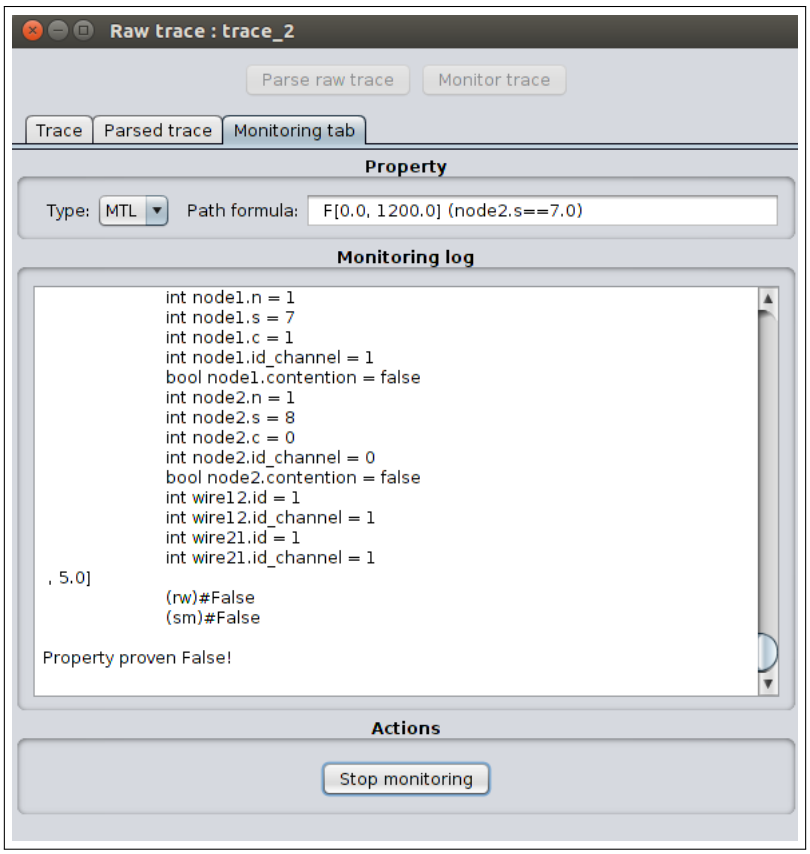


Figure 5.6: Screenshot of a property-based debugging

Debugging. This activity consists in identifying the source of the error occurring at simulation time. The designer looks at the faulty execution trace and tries to locate the needed structural/computational changes in the BIP model. In $\mathcal{S}$BIP, we provide a property-based debugging interface relying on a property monitor to give insight on the source of the malfunctioning. Given a property describing a nominal system behavior, the interface displays a step-by-step consumption of the faulty trace by the property

monitor which allows identifying the exact sequence of events that lead to that unwanted behavior. Figure 5.6 is a screenshot of a property-based debugging interface, where, one can visualize the BIP trace, its parsing and its detailed monitoring. This activity can be undertaken to debug both the system against unwanted behaviors, and properties with an unexpected monitoring verdict.

5.2.1.2 Requirements Formalization

Once a first model is obtained, the system requirements are formalized using specification logic. This formalization consists in describing the exigencies of the system, usually expressed in natural language, using a set of properties. Figure 5.7 illustrates the requirements formalization process.

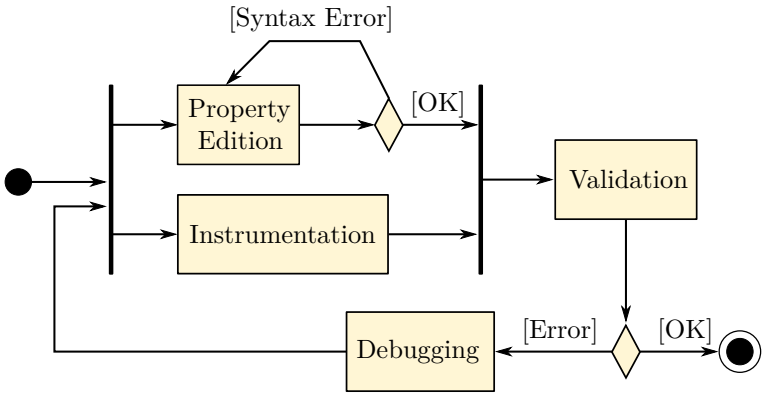


Figure 5.7: Requirements formalization diagram

Property edition. $\mathcal{S}$BIP supports LTL and MTL as specification formalisms in their bounded variants. Syntactically, the temporal operators are represented by a single glyph, namely, **[G]** for the always ($\Box$), **[F]** for the eventually ($\Diamond$), **[N]** for the next ($\bigcirc$), **[U]** for the until ($\mathcal{U}$) and **[R]** for the release ($\mathcal{R}$). Boolean operators are written in the conventional manner: **[&&]** for the AND ($\wedge$), **[||]** for the OR ($\vee$) and **[!]** for the NOT ($\neg$). Note that the logical true (**t**) and false (**f**) are represented by dedicated lexemes, namely, **[#True]** and **[#False]** respectively.

The property parser validates the syntactic correctness of the property, such as the operators syntax and nesting in addition to the parentheses. $\mathcal{S}$BIP allows the designers to save parsed properties using the Java serialization mechanism. The saved properties are organized in the *Properties* folder of the project, and listed in an alphabetical order. The reverse mechanism, namely deserialization, is used to load saved properties into the property edition tab.

Instrumentation. Model instrumentation consists in displaying raw and aggregated information about the system behavior and performance in the execution traces. In $\mathcal{S}$BIP, this information is carried by the valuation of data variables constituting a system state. Instrumenting BIP models requires the addition of aggregation and flag variables in order to compute indicators and identify the occurrence of tracked events, respectively. These variables are then exhibited in the system traces using the conventional "printf" function with a predefined parameter format "**var** data_type data_name data_value", where **var** is a keyword and the remaining attributes are information about the logged variable. For example, the integer variable x is logged with the instruction printf("var int x %d\n", x). The trace parser only considers a subset of native BIP data-types, namely, integers (int), floating-point numbers (float) and boolean values (bool), and does not support sequences of characters (string) and external C++ types. These latter can still be used in the BIP model but must not be visible in the execution traces.

Instrumentation is highly dependent on the property to monitor and the designer experience. It is important to optimize the set of instrumentation variables in order to minimize their impact on the system performance, that can be at two levels: (1) due to the added computation on the interactions/transitions, and (2) due to the introduction of more outputs in the traces. Note that the latter also impacts the performance of analyses, and especially the parsing of these traces that is achieved by the property monitor.

Validation and debugging. The validation and debugging of a property relies on the property-based trace debugging. A trace is generated and then monitored against the target property. The property-based debugging then gives evidence of possible errors in the property formalization, or with the instrumentation variables, such as missing flag resets or miscalculation of the aggregation.

5.2.2 Analysis Workflows

The tool enables three analysis workflows, namely, the classical SMC including *HT* and *PE*, the parametric exploration (*PX*) and the rare events analysis. Depending on the used workflow, one can visualize the analysis results as a single probability, a yes/no answer, or a chart/bar plot. The tool also allows for visualizing the generated execution traces. Storing execution traces can be enabled/disabled by the user as it may be memory consuming.

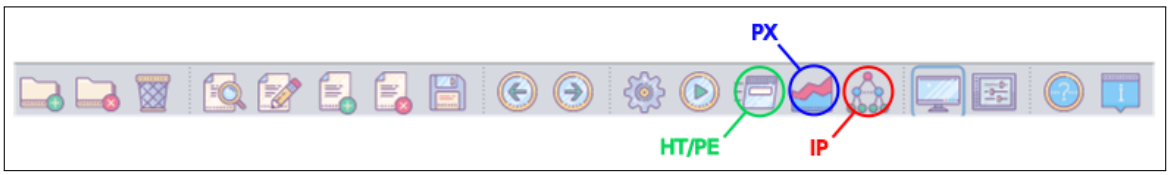


Figure 5.8: Screenshot of the $\mathcal{S}$BIP toolbar

These analysis workflows are initiated by pressing on their corresponding button in the $\mathcal{S}$BIP toolbar (as depicted in Figure 5.8). They take as inputs models and properties resulting from the design activities. Consequently, the model under study has been previously validated and instrumented in such a way that it can be monitored against a syntactically correct property. It is worth mentioning that these workflows can be used to tackle the functional validation of designed models and properties. Furthermore, they represent a lightweight means to quantitative analyses in the context of performance, security and safety assessments.

The analysis workflows share the same phases, that consist first in selecting the inputs, then setting the analysis parameters, running that analysis and finally interpreting its results. In the following, we explicit the steps for performing analyses using the $\mathcal{S}$BIP tool.

5.2.2.1 Classical SMC Workflow

The tool proposes two classical SMC procedures, namely, *Hypothesis Testing* (HT) and *Probability Estimation* (PE), that rely on the following four steps.

Input selection phase. This step identifies the model and property under study in the current project, from the project explorer. The classical SMC procedure starts by first selecting the executable of a model usually referred to a *system*. This executable results from the compilation phase and is located in the *Models* folder of the project. The selection of the property is done in a straightforward manner, by double-clicking on it in the *Properties* folder of the project.

SMC setting phase. This step determines the analysis settings that fall into two categories: simulation and algorithm settings. The first category represents the simulation settings where the designer can specify the simulation mode (symbol-wise or trace-wise) but also decide whether the evaluated traces must be saved. The second parameters category determines the SMC procedure and its statistical parameters.

Analysis phase. Once the parameters are set, the designer launches the analysis by pressing the button *Start simulation*. A secondary window appears and displays details about the analysis progression in real-time. This interface gives feedback on the number of traces generated so far, together with their local verdicts. A progress bar is also displayed in the case of *Probability Estimation* to visualize the progression of the analysis. Note that this progress bar is not available for *Hypothesis Testing* procedures, since the total number of required runs is not known before-hand.

Results interpretation phase. At the end of the analysis phase, a tab is added to the central panel of the GUI, summarizing the analysis and detailing the obtained results. The summary of the analysis recalls the chosen analysis and parameters, in addition to the

model and property under study. The results are represented by the global SMC verdict, the required number of traces and the global analysis time. Further details about the evaluated traces are displayed in a table. For each trace, this table informs the designer of the number of symbols consumed by the monitor before concluding, its process time and monitoring verdict. Note that the evaluated traces can be viewed and an individual monitoring of these traces can be rerun using the property-based debugging interface.

5.2.2.2 Parametric Exploration (PX) Workflow

The second workflow consists in analyzing different instances of a parametric property by performing several iterations of SMC analyses. It encompasses four phases of input selection, exploration setting, analysis and result interpretation, detailed as follows.

Input selection phase. The user first selects the model and the property to explore, similarly to the classical SMC workflow. It is important to make sure that the selected property is parametric. Using a non-parametrized property for *PX* would result in performing the SMC analysis on exactly the same property several times, namely, repeating it $|\Pi|$ times where Π is the instantiation domain (see Section 5.1.3.1 for more details).

PX setting phase. This step is a special case of the SMC parameter settings. In addition to the simulation and algorithm settings, the designer determines the exploration parameters. These parameters encompass the parameter name and type (integer or boolean), and the instantiation domain Π. For a parameter of type integer, this domain is represented by three values: a lower l and upper u bounds, identifying an interval $[l, u]$, plus a discretization *step*. For example, defining an integer parameter x with $l = 5$, $v = 13$ and $step = 5$ would result in an instantiation domain $\Pi = \{5, 10, 13\}$, and $x \in \Pi$. Note that boolean parameters are implicitly assigned an instantiation domain $\Pi = \{\mathbf{t}, \mathbf{f}\}$.

The size of the instantiation domain, together with the SMC parameters (determining the level of confidence), allow the designer to control the overall exploration time, while searching for an optimal parameter value x^*. An efficient way to perform this latter would be to start the exploration with a wide interval, a big *step* and a low level of confidence, then iteratively refine the interval, reduce the *step* and increase the confidence level, until the optimal parameter is identified. Finally, the optimal value can be validated by running a classical SMC workflow, to guarantee the desired level of confidence.

Analysis phase. The parametric simulation starts when the designer presses the button *Start simulation* in the exploration setting tab. As a result, a secondary window pops on top of the main window and shows the exploration progression details. The displayed information indicates the current state of the exploration, namely, the current valuation of the explored parameter, in addition to the generated traces and local verdicts. Similarly

to the classical SMC workflow, the secondary window is equipped with a progress bar that displays the progression of the exploration in terms of proportion of the instantiation domain that has been explored. It contrasts with its usage in the classical SMC workflow that represents the progression of the SMC analysis (in number of traces).

Results interpretation phase. Once the exploration is achieved, a tab is added to the central panel of the tool, displaying a summary of the parametric exploration. This tab is organized in two panels. The first recalls the parameters of the exploration and presents its results in a comprehensive manner. The second panel is dedicated to the visualization of the SMC verdicts and the processing time as functions of the parameter value. This visualization as charts allows the designer to quickly identify trends, and hence interpret his/her results. Figure 5.9 is a screenshot of a PX results tab.

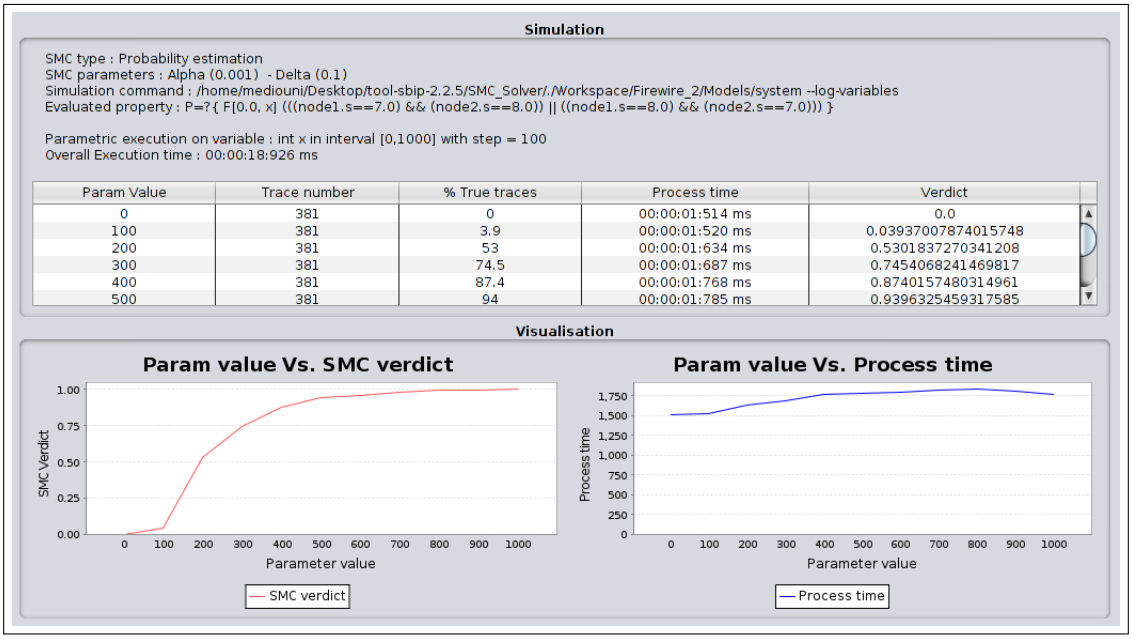


Figure 5.9: Screenshot of the parametric exploration results

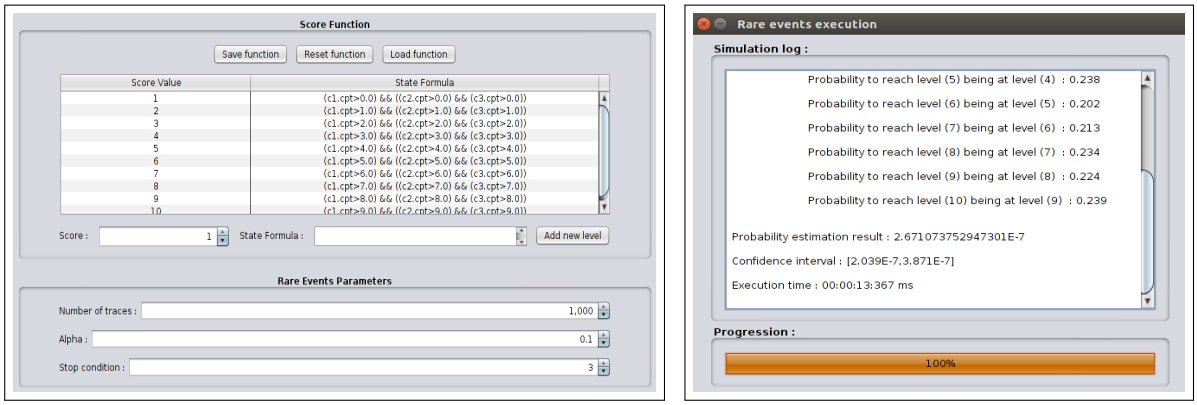
5.2.2.3 Rare Events Analysis Workflow

The third workflow aims at to estimate the probability of a rare property on stochastic systems. This property is subdivided into n intermediate properties of lower rarity that are represented as a scoring function.

Input selection phase. The inputs for a rare event analysis are a BIP model and a score function. The designer can select the executable of the model under study in the project explorer. Currently, the rare events workflow only supports untimed

systems, namely, system with a DTMC underlying semantics. It is important that the BIP model is compiled using the untimed BIP compiler (not SRT-BIP) with the option `--gencpp-enable-marshalling` that enables to manipulate the system states, namely, to save, load and remove these states. In contrast to other workflows, the score function is not selected similarly to MTL/LTL properties, but is specified as part of the PX setting phase detailed next.

IP setting phase. The initialization of the *IP* analysis instantiates a setting panel, depicted in Figure 5.10a, where one can specify: (1) the score function, and (2) the algorithm parameters. For the former, $\mathcal{S}$BIP provides the operations necessary to manipulate score functions, namely, initialize a new function, add a rarity level, save and load a score function. Note that saved functions are identified by the file extension (.sf) and are stored in the *Properties* folder of the project.



(a) Importance splitting setting (b) Importance splitting results

Figure 5.10: Screenshots of the rare events workflow

The algorithm parameters specify the number of execution traces, the statistical parameter α and the stop condition. The parameter α serves in the computation of the confidence interval for the resulting estimated probability. the stop condition indicates the upper limit on the number of steps for a trace to successfully reach a higher level of rarity.

The difficulty with *IP* is to express a rare property since it often requires to find a good representation of the rarity levels and to properly define the stop condition. for example, one can be interested to express the rare property that three components access a shared resource exactly the same number of time, over a total of 30 steps. This property can be expressed in LTL as $\phi =$ `G{30} (F{3} ((c1.cpt==c2.cpt) && (c3.cpt==c2.cpt)))`. The equivalent score function f is defined by the intermediate state formulas ϕ_i given as follows: $\phi_i =$ `c1.cpt>i-1 && c2.cpt>i-1 && c3.cpt>i-1`. Each level of rarity i

indicates whether all the components accessed the resource at least i times. To make the score function conform to ϕ, we set the stop condition to 3 (similarly to the bound of the F operator in ϕ).

Analysis and results interpretation phases. Once the analysis is initiated, the tool displays a secondary window, depicted in Figure 5.10b, composed of a text area and a progress bar. The former provides analysis logs allowing the designer to identify the current state of the analysis, namely, the conditional probability (of the level) that is being estimated. The progress bar represents the advance of the analysis in terms of the proportion of estimated levels among all the levels represented by the score function.

At the end of the exploration (when the progress bar reaches 100%), the text area contains a summary of all the information computed during the analysis, namely, the conditional probabilities to go from a level to the next, the overall probability to satisfy the rare property, the confidence interval and the analysis time.

5.3 Implementation Details

$\mathcal{S}$BIP is fully developed in the Java programming language, and requires at least the Java Runtime Environment (JRE) 7. It uses ANTLR 4.7 [1] for LTL/MTL properties parsing.

5.3.1 Overview of the Code Structure

The $\mathcal{S}$BIP architecture follows the Model-View-Controller pattern. The *model* encompasses the classes describing the data entities manipulated by the tool, such as formulas and traces. The *view* is a presentation of data and the functionalities that are proposed by the tool in the GUI module. The *controller* validates the commands coming from the GUI and manipulates the data entities accordingly. It includes computations such as the statistical analyses. Figure 5.11 shows a simplified package diagram of the $\mathcal{S}$BIP tool, where model packages are represented in orange, view packages are colored in beige, and controller packages are depicted in green.

5.3.1.1 The Model

The model is decomposed in four packages representing the simulation traces, the formulas, in addition to data required by the analyses and the graphical components.

The trace package. It provides a high-level representation of the BIP simulation traces. In this representation, a trace (of type `Trace`) is a sequence of timestamped states. These states are characterized by a real value (representing the global clock valuation) and a set of state variables, where each state variable has a name, type and value.

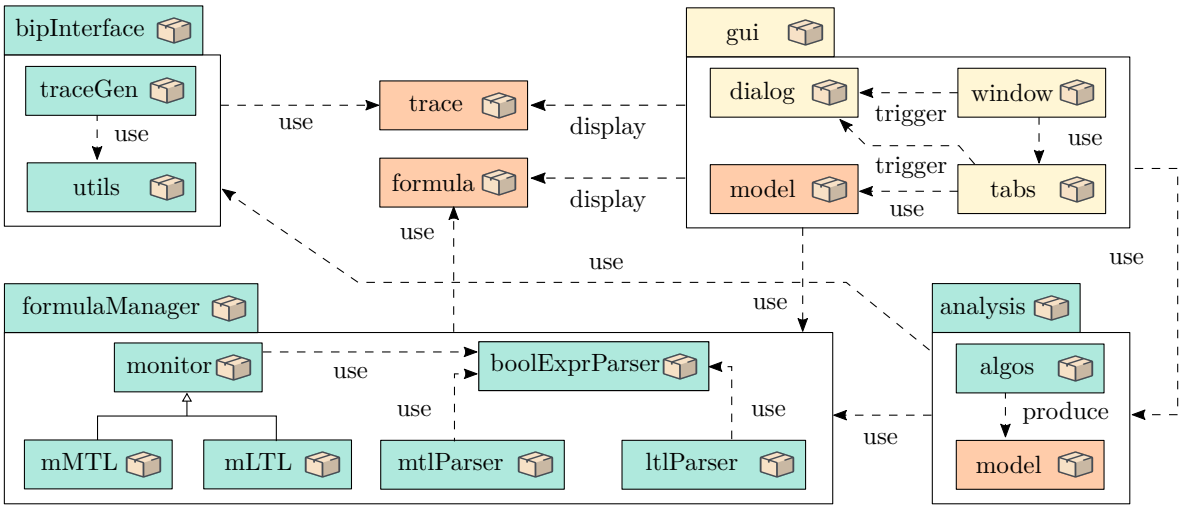


Figure 5.11: Simplified package diagram of the $\mathcal{S}$BIP tool

This high-level representation results from the parsing of the raw BIP traces produced by simulation engines. In practice, we use this same structure to represent timed but also untimed traces, by simply assigning a zero value to the timestamps, in the latter case.

The formula package. This package includes the necessary classes to define MTL/LTL properties and score functions. An MTL (resp. LTL) path formula is obtained by composing state formulas with time-bounded (step-bounded) temporal operators. To enable the nesting of operators, path and state formulas both extend an abstract class `Formula`. The latter also makes it very easy to extend the tool to support more specification formalisms.

The gui.model package. This package includes classes to present data in shapes that are recognized by graphical components such as tables and charts. For example, some of these classes implement the `AbstractTableModel` interface to facilitate the graphical display of tables.

The analysis.model package. In this package are regrouped all the data structures that are created during an analysis, either as intermediate information or as a final output. *HT* and *PE* analyses generate intermediate trace verdicts that are stored in a class as a pair of trace and verdict which can be either a real number or a boolean value. Similarly, *PX* produces intermediate SMC verdicts for which the value of the parameter, the execution time and the actual verdict are stored to be later on displayed in the GUI, after being structured accordingly.

5.3.1.2 The View

The GUI is developed using the Java Swing graphical library, a platform-independent framework for graphical interfaces development in Java. It provides sophisticated, powerful and flexible

components for defining a wide range of designs. Also, it supports pluggable *look and feel* to define colors, shapes and layout etc. For our implementation, we use the cross-platform *Nimbus* look and feel.

The GUI is composed of a main window and a group of secondary windows. The main window is implemented in the class `UserInterface` of type `JFrame`. It instantiates a menubar (not functional yet), a *toolbar* and a *split panel* that allows for the dynamic vertical separation between the two panels: *project explorer* and *central panel*. The former is developed around a `JTree` component that displays the content of the *Workspace* folder on the file system. This folder is created at the first execution of the tool and is managed through the different project creation/deletion functions in the toolbar. The central panel is of type `JTabbedPane`, used for the navigation between several containers through tabs (developed in the package `gui.tabs`). This panel is the most important component of the tool since it hosts the different activities of edition, analysis setting and results visualization. Each activity is accessible through a dedicated tab. Except for the starter, the model selection and the formula specification tabs that are open on program startup, all the others are added upon request of the user and can be closed as well.

Besides the main window, secondary windows are also developed (in the package `gui.dialog`) and meant for different purposes:

1. the progression viewer to follow the status of an analysis,
2. the trace viewer for the visualization, the parsing and the monitoring of raw BIP traces,
3. the compilation viewer to display the output of the compilation scripts including eventual warnings and errors from the BIP compiler,
4. the help box to provide useful information about the installation, configuration and utilization of the tool,
5. the *about us* box to inform of the purpose of the tool and its version, in addition to a contact e-mail address,
6. the dialog boxes to display messages, such as errors, and to request input or confirmation from the user.

These secondary windows result from an action of the user, either on the toolbar or in the tabs of the central panel.

5.3.1.3 The Controller

It is structured in three groups of packages: the `formulaManager` and the `bipInterface` packages, for properties and traces manipulation respectively, and the `analysis` package. The manipulation of properties is two-fold: the parsing and the monitoring. The property parsing is based on ANTLR and is provided for both LTL and MTL properties. The parsing of such

properties takes a textual representation and builds a high-level representation by instantiating a `Formula` object. Monitoring these properties is done using the rewrite-simplify approach that consumes the property given an observed system state. For extensibility, we define an abstract class `Monitor` that we extend for both MTL and LTL monitors by implementing their specific rewriting procedures. It is worth mentioning that the interpretation of the state formulas of MTL/LTL properties, at analysis time, is ensured by a specific parser, denoted `boolExprParser`, that first instantiates the variables in the formula with their valuation in the observed state, then evaluates the resulting boolean formula.

The simulation of BIP models is covered by the trace generation controllers. These controllers interface with the BIP engines to produce raw BIP traces in three different modes. The `TraceGenerator` is responsible for the trace-wise simulation mode, that is, produce a trace of a predefined length. At contrary, the `ThreadedTraceGenerator` generates a trace symbol by symbol, and saves the generated symbols in a buffering structure. The piece-wise generator implements the BIP binding defined for the interfacing with Plasma-Lab. In any of these three cases, the generated BIP traces are obtained by redirecting the standard output stream of the BIP simulation process to a `StringBuilder`. The resulting sequence of characters is then parsed using the `Utilities` class to build a high-level representation of type `Trace`.

The `analysis` package regroups the analysis cores, namely, HT, PE, PX and IP. These cores are implemented as an extension of the abstract class `Algorithm` to provide flexibility with respect to extending the tool with additional analyses. In addition, this package also contains the class `Simulator` that is responsible for the execution of these algorithms in an offline (with traces of fixed length) or in an online mode. In this latter mode, the `Simulator` implements a producer-consumer design pattern where the trace generation and its monitoring are done in parallel: the generation of the trace is done symbol-wise in a first thread, that feeds a buffering structure, then a threaded monitor consumes the generated symbols on-the-fly. The simulation ends whenever the monitor is able to conclude a global verdict. It is worth recalling that the rare event analysis using an objective-guided simulation, that we proposed in Section 5.1.3.3, is not implemented yet.

5.3.1.4 Programming Efforts

The $\mathcal{S}$BIP has been the subject of a completely new development that took us approximatively four months. Our source code is organized in 86 Java classes structured in packages for a total number of 15454 lines of code. The main effort is concentrated on the development of the GUI that represents more than 53% of the source code.

5.3.2 Availability and Documentation

At this stage, $\mathcal{S}$BIP only runs on the Linux operating systems as it relies on BIP simulation engines. The tool is freely available for download from the BIP-SMC web page http://www-verimag.imag.fr/Statistical-Model-Checking.html.

It is distributed with a `CeCILL-B` free software license in three different forms: standalone, source code and a preconfigured virtual machine. The standalone is compiled on an Ubuntu 16.04-64 bits operating system running Java Runtime Environment (JRE) 7 and GNU Scientific Library (GSL) 2.3. It is portable to different Linux OS but the BIP engines must be recompiled on the target architecture. To reduce the installation overhead, we also provide a fully configured virtual machine as a turn key solution. This latter also guarantees that the tool works in its development conditions, isolated from any interaction with wrong dependencies or library versions. For more advanced users, we make the source code available online[2] to allow for adaptation to custom needs (models and requirements).

The archive of the tool contains an installation script for the validation of the JRE version and the creation of a desktop launcher icon. In addition, a folder with all the necessary files is included. This folder provides the compiled untimed and SRT-BIP engines, the help files in HTML format, the compilation scripts, the tool icon images and finally the *Workspace* folder. We provide in this latter a bunch of case studies that may be used as a basis for further designs, or as a starting point to learn how to use the tool.

$\mathcal{S}$BIP is simple to master and easy to use thanks to the intuitive activities and workflows, described in Section 5.2. Its rich documentation makes it even simpler for a beginner user. This documentation span from YouTube video tutorials to textual reports. In the latter category, in addition to the *help* window accessible from the GUI, we distribute a user manual, a technical report describing the tool and some experiments performed on case studies, and links to the BIP framework explaining its syntax and semantics. All these documents and links are available on the BIP-SMC web page.

5.4 Related Tools

In this section, we focus on comparing our statistical model checking tool with the existing ones and more specifically Uppaal-smc [53], Prism [101]. Table 5.1 summarizes the features and characteristics of the state-of-the art SMC tools. We restrict this table to the comparison with the most used tools, namely, Uppaal-smc and Prism, but we also discuss some of the capabilities of Cosmos [20] and Plasma-Lab [88].

The SMC tools mainly differ in several aspects spanning from the modeling framework, specification languages and the statistical analyses, defining the tool's *capabilities*, to the user

[2]Link to the GitLab repository of $\mathcal{S}$BIP: https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/bip/sbip2

experience, the design and performance, describing the tool's *usability*. In terms of modeling and specification capabilities, we focus on the expressiveness of the provided modeling language to build complex and hierarchical systems. We also discuss the tool's usability from the end-user point of view but also for advanced usage in terms of tool customization and performance.

Most SMC tools [20, 53, 88, 101, 151] use dedicated abstract models as input for verification. In contrast, $\mathcal{S}$BIP uses BIP, a full-fledged expressive component-based framework developed to support system design from specification to analysis and implementation. It allows for incremental building of complex systems from elementary components and offers real-time capabilities, in addition to high-level coordination and synchronization primitives e.g. multi-party interactions and priorities. Furthermore, it enables including external C++ code, e.g. for modeling complex data structures and integrating legacy code.

We briefly discuss $\mathcal{S}$BIP analysis capabilities with respect to major SMC tools. Regarding the analyses, $\mathcal{S}$BIP implements the *HT* and *PE* algorithms similarly to Uppaal-smc , Prism and Plasma-Lab. Besides, only Prism offers a parametric functionality similar to *PX*. Furthermore, to the best of our knowledge only Plasma-Lab and Cosmos implement rare events analyses. The former is the only one implementing *IP* as in our tool, while the latter rather relies on importance sampling. Our underlying modeling formalism supports generally distributed time delays. It offers built-in standard distributions, e.g. Normal, and a simple mechanism for specifying custom distributions. In contrast, Prism is restricted to uniform and exponential distributions, whereas in Uppaal-smc one needs to define such distributions manually by using a subset of the C language.

The integration of graphical user interfaces in these SMC tools creates a pleasant experience for the user. Uppaal-smc facilitates the modeling by proposing a graphical modeling tool, unlike Prism and $\mathcal{S}$BIP. However, $\mathcal{S}$BIP is built around distinct workflows to allow for a simplified analysis setting, and an easy results interpretation. Also, the detailed analysis feedback, including the ability to visualize the generated concrete traces and their local verdicts, gives the user a better understanding of his system and a higher confidence in the obtained results. Besides, the trace inspection comes out very useful in the context of system debugging.

The expressiveness of BIP together with the reliance on concrete executions result in lower runtime performance compared to Uppaal-smc and Prism. Comparatively, the authors of Plasma-Lab chose to focus on modularity at the expense of performance.

5.5 Conclusion

$\mathcal{S}$BIP is a framework for modeling and analyzing BIP models. It supports both the standard BIP and the Stochastic Real-Time BIP modeling formalisms that have DTMC and CTMC/GSMP underlying semantics, respectively. Requirements in this framework are formally expressed using the bounded versions of LTL and MTL that can also be parametrized to specify families of properties. The analysis capabilities of the tool rely on Statistical Model Checking techniques

Tool		$\mathcal{S}$BIP	UPPAAL-SMC	PRISM (SMC)
Modeling and specification	Semantics	**GSMP**, CTMC, DTMC	STA	CTMC, DTMC
	Language	Built-in density functions	Uniform and exponential distribs	Uniform and exponential distribs
		Custom distribution through files	-	-
		External C++ code:	Functions in C dialect:	-
		→ **complex** data structs, functions	→ no external or native C libraries	
		Concrete execution traces	Simulation traces	Simulation traces
		Extensible language using annotations	-	-
	Design	Component-based with connectors	comp-based with sync on ports	module based
		Multi-party interactions	(Multi-party disabled for SMC)	-
		No global vars	Global vars	Global vars
	Specs	LTL, MTL	Weighted MTL	PCTL, CSL, LTL
		Parametric properties on time and vars	Manual exploration	Param properties on vars
		Manual encoding of cost/rewards	Operators on cost/reward	Operators on cost/reward
Analysis	HT/PE	Yes	Yes	Yes
	Exploration	**Yes**	No	Yes
	Rare events	**Yes**	No	No
User experience		Clear and distinct workflows	Monolithic analysis workflow	single parametrable workflow
		Visualization of sim traces	Not visible	Not visible
		No graphical modeling tool	Graphical modeling tool	No graphical modeling tool
		Compact representation of parametric properties	Tedious and manual	Compact representation
		Select different simulation engines	No	No
Tool design		**Open source**	Closed source	Open source
		Extensible and modular	No	No
Performance	Comparison	**Slower**	-	-
	Reasons	90% time in trace simulation	-	-
		Runtime analysis progression feedback	-	-
		Traces save for user inspection	-	-

Table 5.1: Comparison table of the state-of-the-art SMC tools

that have the advantage of scalability compared to exhaustive methods, while giving statistical guarantees on the errors, as opposed to pure simulation.

$\mathcal{S}$BIP is designed as an IDE and is released with a GUI that facilitates the different design activities. Also, the tool proposes classical SMC algorithms and rare events analysis, in addition to a means to explore a family of properties in an automated manner. These analyses are accessible in clear and distinct workflows. The modularity and extensibility of the tool makes it a good candidate for advanced users as it allows for customization. However, this flexibility comes at a cost of lower performance compared to state-of-the-art SMC tools. As future work, we intend to enhance this performance. We also believe that providing a graphical modeling interface would have a positive impact on the tool usability, and would make it easier to handle for beginner users and also for more advanced ones.

In the next chapters, we illustrate the usefulness of the tool in different domains and for several purposes. In Chapter 6, we present six case studies that we model with $\mathcal{S}$BIP and for which we focus on analyzing the performance. We also take a closer look at the tool's overall performance for which we provide insight into practical causes and solutions. In Chapters 7 and 8, we show how SMC in general and $\mathcal{S}$BIP in particular can be applied in the proposition of more complex methodologies in the context of safety and security risk assessment, respectively.

Chapter 6

Analysis of System Performance with $\mathcal{S}$BIP

Designing correct and efficient systems is a complex task that requires both functional verification and performance analysis. While the former ensures that the system behavior conforms to its description and requirements, the latter addresses serious questions about energy consumption, response time, etc. It provides quantitative measurements that guide the designer at every stages of the design towards well-founded decision making. This analysis is not straightforward to achieve, particularly on systems for which the interoperative effect of software and hardware is crucial. In this chapter, we show how the $\mathcal{S}$BIP framework, presented in the previous chapter, can be used to achieve this purpose on various case studies.

The $\mathcal{S}$BIP framework relies on an expressive modeling formalism, and scalable and fast verification techniques. This framework is supported by a tool that provides modeling, simulation and analysis capabilities. This tool must undergo testing in order to validate its development and to evaluate its performance. To do so, besides unitary testing, we confront $\mathcal{S}$BIP with several case studies in concrete utilization conditions, enabling us to see how it behaves in complex configurations, and to assess its usefulness and scalability for large size problems. These experiments helped to identify and fix numerous bugs, and guided the development of the tool by raising needs related to the design process and analysis workflows, such as traces visualization and debugging.

In this chapter, we address the performance analysis of real-life case studies in a variety of application domains using $\mathcal{S}$BIP. More specifically, we study communication and clock synchronization protocols in Section 6.1, in addition to complex embedded systems in Section 6.2. In Section 6.3, we show the importance of rare events analysis on a concurrency model with a shared resource. Finally, Section 6.4 presents an analysis of the tool performance. For reproducibility, we provide the model and requirement files for all the case studies in the tool distribution.

6.1 Communication Protocols Case Studies

6.1.1 FireWire – IEEE 1394

FireWire is a high-performance serial communication bus dedicated for hot plug-and-play multimedia devices. These devices can be organized in arbitrary, yet acyclic, topologies, where each pair of nodes is connected by two unidirectional channels. The internal representation of topologies is a tree where the root (*leader*) arbitrates the access to the bus. The designation of the leader is performed through a leader election protocol, namely, the *tree identification protocol*. Whenever the topology changes, i.e., a device joins/leaves, a reset occurs, and a new election is triggered.

The tree identification protocol is initiated by the leaf nodes of the topology. They send requests asking their neighbors to become their parents. A *parent request* sending mode is probabilistically determined to be $fast$ or $slow$. It indicates the amount of time to wait before sending. Internal nodes of the topology keep on listening to parent requests until they receive exactly $n-1$ requests, n being their number of neighbors. Then, they send a parent request to their remaining neighbor. When receiving a parent request, a node either sends an acknowledgment, or detects a *contention* in the case it has also sent a parent request and it is still waiting for an acknowledgment. Intuitively, a contention means that two neighbors are mutually asking to be leader. This situation is resolved by forcing both nodes to send new requests after a random waiting time.

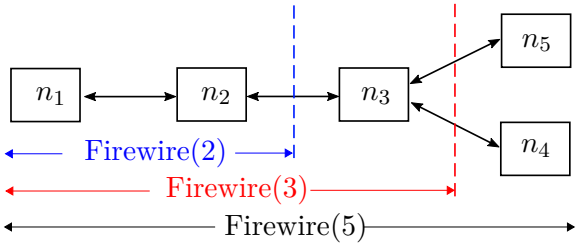


Figure 6.1: Considered FireWire topologies

We implemented a FireWire model inspired from the case study in [52], where the considered topology is made of two devices. Our model is parametric, with m possible devices. We considered three particular topologies with $2, 3$ and 5 devices (Figure 6.1). The models for a device and a channel are shown in Figure 6.2. On the left, the Device component is essentially a timed automaton. On the right, the Channel component contains in addition a stochastic port *rcv_ack* defined by a normal density function, i.e., its scheduling time is sampled with respect to a normal distribution with a mean of 10 and a standard deviation of 2.

We studied the expected convergence time for the three topologies with ($\phi_1(t)$) and without ($\phi_2(t)$) contentions. We also investigated the topology impact on the probability of contentions (ϕ_3) and on the probability for each device (regarding its position) to be elected ($\phi_4(i)$). We provide the detailed MTL specifications of the properties verified on the FireWire model:

- the leader election procedure converges within t time units. It states that one of the nodes eventually becomes a leader and all the other nodes become slaves. The parametric exploration is used to find the expected time t^* when the process is guaranteed to converge

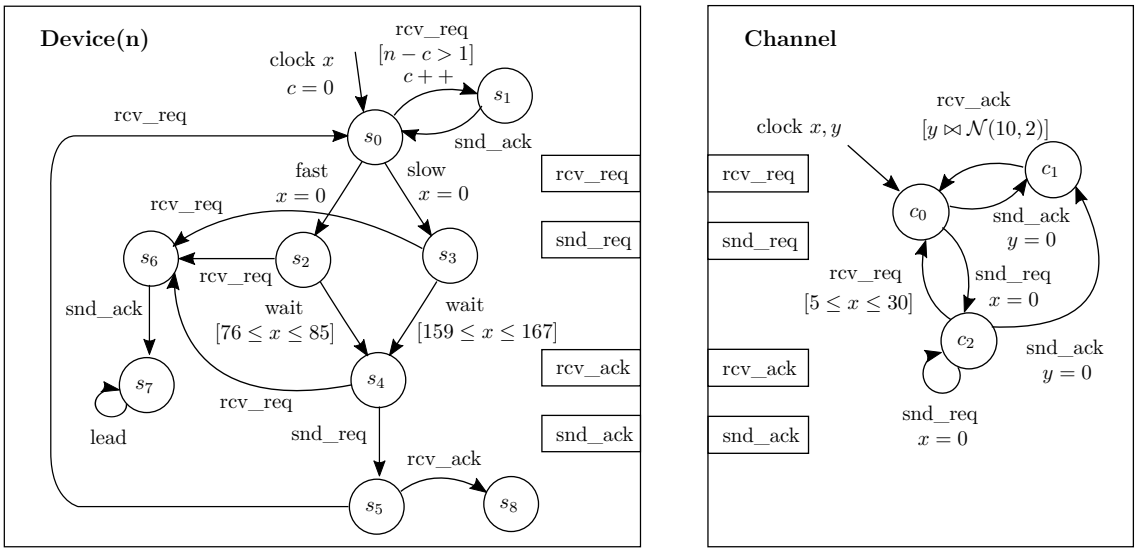


Figure 6.2: Stochastic real-time BIP: Components of the FireWire Protocol.

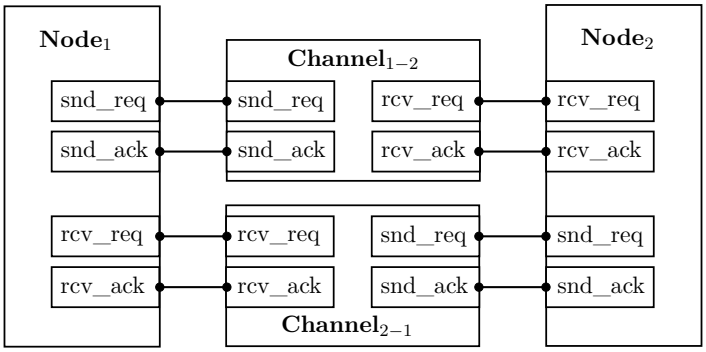


Figure 6.3: Firewire component composition

(with probability 1).

$$\phi_1(t) \equiv \Diamond_{[0,t]} \bigvee_{i=1}^{m} [(node_i.s = 7) \bigwedge_{j=1, j \neq i}^{m} (node_j.s = 8)]$$

- the leader election procedure converges within t time units if no contention occurs. The property is basically an implication written as a disjunction of two parts. The first part of the disjunction is a conjunction between ϕ_1 and a second property stating that always no contention occurs during the election phase ($[0, t]$). The second handles the cases where a contention eventually occurs in $[0, t^*]$, where t^* is computed in ϕ_1. Note that the election and the contentions are detected at the level of the nodes.

$$\phi_2(t) \equiv (\Diamond_{[0,t]} \bigvee_{i=1}^{m} [(node_i.s = 7) \bigwedge_{j=1, j \neq i}^{m} (node_j.s = 8)] \wedge \Box_{[0,t]} \bigwedge_{i=1}^{m} [\neg node_i.contention])$$

$$\vee \left(\Diamond_{[0,t^*]} \bigvee_{j=1}^{m} [node_i.contention]\right)$$

- a contention eventually occurs during the election phase (t^* computed in ϕ_1):

$$\phi_3 \equiv \Diamond_{[0,t^*]} \bigvee_{i=1}^{m} (node_i.contention)$$

- the i^{th} device eventually becomes the leader (t^* computed in ϕ_1):

$$\phi_4(i) \equiv \Diamond_{[0,t^*]} (node_i.s = 7)$$

We used probability estimation with $(\alpha = 5 \times 10^{-11}, \delta = 5 \times 10^{-2})$ for all the analyses and relied on the parametric exploration to analyze properties $\phi_1(t)$ and $\phi_2(t)$.

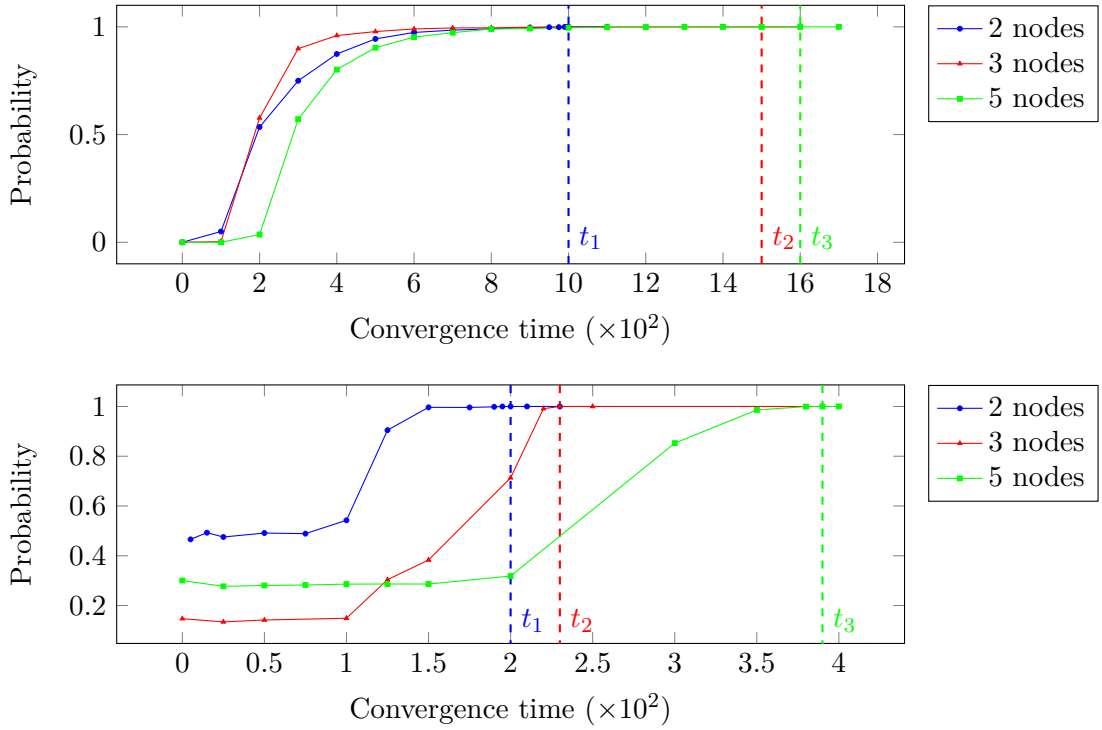


Figure 6.4: Probability of ϕ_1 (top) and ϕ_2 (bottom) for different FireWire topologies

We observed that the expected convergence time increases with larger topologies, as shown in Figure 6.4 for $\phi_1(t)$ (top) and $\phi_2(t)$ (bottom). For $\phi_1(t)$, the expected time (in time units) was respectively 1000, 1500 and 1600 for the three considered topologies. When no contention occurs ($\phi_2(t)$), the expected time drops to 200, 230 and 390. The protocol spends more than 80% of the time resolving contentions. The analysis results for ϕ_3 and $\phi_4(i)$ are summarized in Table 6.1. We noticed that in a two-device topology, both nodes send parent requests almost

simultaneously and thus have equal chances to become leader, but leads to $\sim 50\%$ chance of contention. In larger topologies, leaf nodes initiate the election protocol, hence they have less chance to become leaders (nodes $n_{1,3}$ in FireWire(3) and $n_{1,4,5}$ in FireWire(5)). In contrast, inner nodes are more likely to become leader and this increases proportionally to the number of their neighbors. Moreover, we observed that the probability of contention in FireWire(3) is lower than the other topologies. Actually, contentions do not only depend on the number of nodes in the network but also on the way they are interconnected.

FireWire	ϕ_3	$\phi_4(1)$	$\phi_4(2)$	$\phi_4(3)$	$\phi_4(4)$	$\phi_4(5)$
(2)	0.493	0.507	0.493	-	-	-
(3)	0.137	0.042	0.92	0.038	-	-
(5)	0.289	0	0.4	0.6	0	0

Table 6.1: Results for properties ϕ_3 and ϕ_4

6.1.2 Bluetooth – Device Discovery

Bluetooth is a short-range wireless communication protocol for data exchange that promises low-energy consumption. A serious challenge in this protocol is interference. The Bluetooth standard relies on frequency hopping to tackle this issue. It allows devices to rapidly alternate among predefined frequency bands in a (pseudo-)random fashion. In order to perform data transfer, nodes in the network initially organize themselves into *piconets*, that is, small groups of one master and up to 7 slaves, where frequency hopping are synchronized. The *device discovery* phase lets one of the devices (called *inquiring*) become the master of the piconet by broadcasting messages to discover scanning devices, i.e., potential slaves. During the discovery phase, each node of the network can be in one of two modes: (1) *active*, where it permanently looks to send or receive messages, and (2) *sniff*, where it alternates between sleeping and listening phases.

We built a model of the Bluetooth protocol (see Figure 6.6), precisely the device discovery mechanism, based on the implementation in [15] that considers one receiver (slave) and one sender (master), where the receiver is set to the sniff mode. We improved [15] by considering a parametric model where the receiver can be in addition in the active mode. Figure 6.5 represents the model of the Bluetooth components. The top figures show the receiver in active (left) and sniff (right) modes. The figures on the bottom show the frequency and the sender components from left to right respectively.

In active mode, the receiver sends a start signal to wake the sender component up and starts scanning the different frequencies through the frequency component. The receiver switches to a state where it is ready to receive *inquiries* whenever it hears a transmission attempt on its frequency. Finally, the receiver commands the sender (and the frequency) component to stop by sending a stop signal and goes back to its initial state where other transmissions can be initiated by taking the dashed Stop transition.

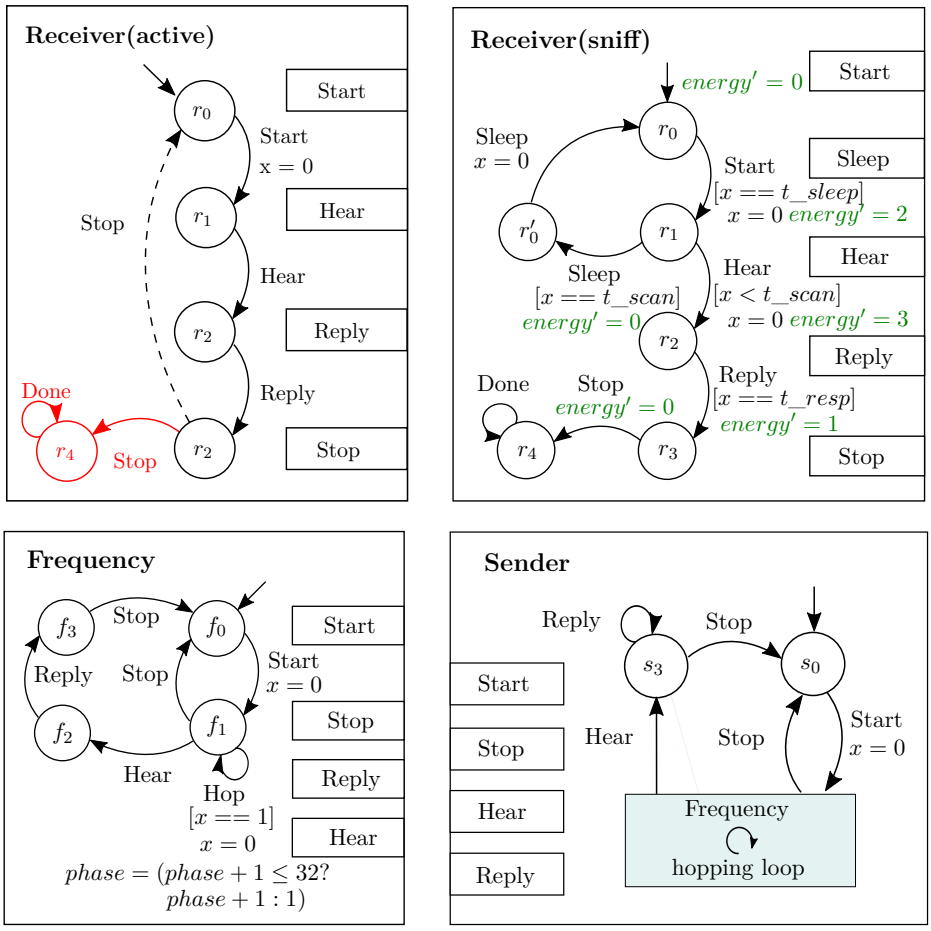


Figure 6.5: Components of the Bluetooth model

In the sniff mode, the receiver alternates between sleeping and scanning states. The sleeping phases of *t_sleep* (2012) time slots are followed by a scanning phase of *t_scan* (36) time slots, where the time slot is 0.3125 ms. During this scanning phase, the receiver can detect a transmission then replies to the *inquiry* that requires *t_resp* (2) time slots to terminate. The receiver is also enriched with a cost clock that computes the amount of consumed energy (in energy units). In the sniff mode, no energy is consumed in sleep states (r_0, r_0'), whereas the receiver consumes 2 energy units per time slot in scan state r_1, and 3 units/time slot during the *reply* (at state r_2).

We measure the impact of the different modes on the delays of discovery and on the receiver's energy consumption. In our model, the discovery process successfully terminates when the sender receives one reply (*sender.rec* = 1). We further

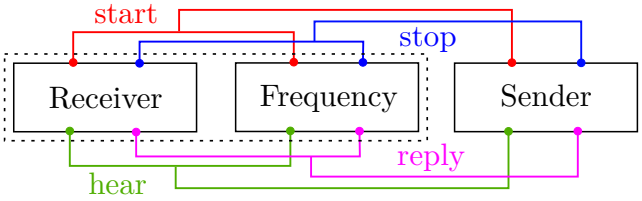


Figure 6.6: Bluetooth model with two devices

model energy consumption of the receiver through the cost clock denoted *energy*. The first requirement is expressed in MTL as $\phi_5(t) \equiv \Diamond_{[0,t]}(sender.rec = 1)$ and states that the discovery must eventually occur within t time units. Our goal is to identify t^* that satisfies this requirement with probability 1. The second requirement is expressed as $\phi_6(e) \equiv \Box_{[0,t^*]}(receiver.energy \leq e)$. It states that the energy consumed by the receiver, during the discovery phase, is always under some threshold e. Again, the goal is to determine e^* that satisfies the requirement with probability 1. Both properties are expressed as parametric MTL and are assessed by using the parametric exploration.

Figure 6.7 summarizes the results obtained by applying the probability estimation algorithm with parameters $(\alpha = 5 \times 10^{-7}, \delta = 5 \times 10^{-2})$ for both modes. As expected, the active mode leads to a shorter discovery phase. On the left, we see that $t^* = 350$ ensures a convergence with probability 1. In the sniff mode (middle), the required time jumps to $t^* = 17000$. Regarding energy (right), in the active mode, the expected energy consumption of the receiver is $e^* = 700$ units, whereas it drops to $e^* = 600$ units when the receiver works in the sniff mode. That is, an energy saving of more than 14% compared to the active mode.

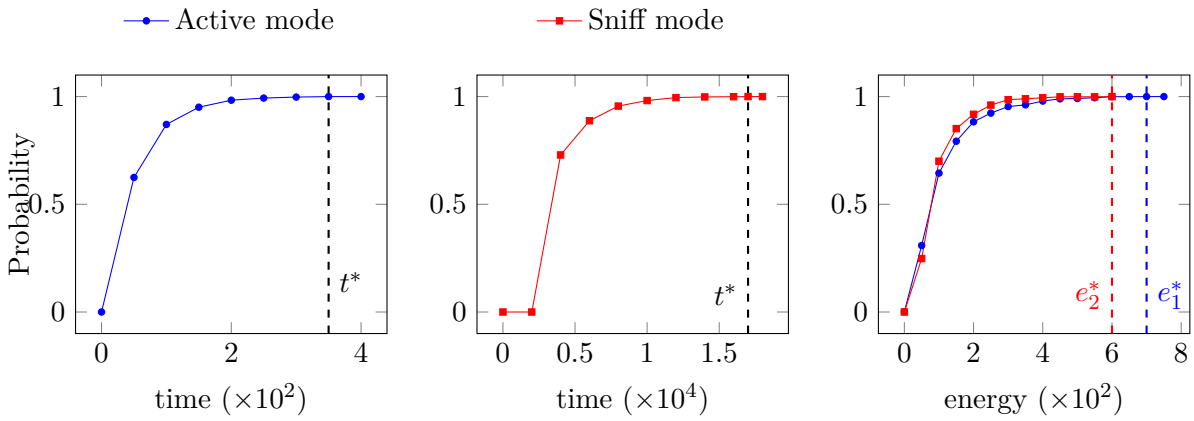


Figure 6.7: Probability of properties ϕ_5 (left and middle) and ϕ_6 (right)

6.1.3 Precision Time Protocol – IEEE 1588

In this study, the Precision Time Protocol (PTP) is deployed as part of a distributed heterogeneous communication system in an aircraft [23] to synchronize the clocks of the different devices of the system. The reference clock is given by a specific device in the network called Master. This synchronization is essential to guarantee a correct behavior of the whole system.

We consider an abstract stochastic model of the PTP protocol shown in Figure 6.8. It is composed of a master and a slave in addition to two communication channels. The considered model is parametric as it represents the communication of the master with different slaves of the actual system. This is expressed through different stochastic communication delays of the

channels (see Figure 6.9a). Concretely, we use different probability density functions, depending on the position of the slave in the network. Additional details on the models can be found in [23].

An important property to verify on the system is that the drift between the clock of the master denoted tm and the clock of any slave denoted ts is always bounded by a threshold Δ. This property is expressed as $\phi_7(\Delta) \equiv \Box_{[0,t]}\ (abs(master.tm - slave.ts) \leq \Delta)$, where t is the simulation time that we fixed to 25 times the period of the PTP protocol, and $abs()$ is the absolute value function. Figure 6.9b shows that the smallest bound Δ guaranteeing the property ϕ_7 is $\Delta^* = 70$. We used the probability estimation algorithm with $\alpha = 5 \times 10^{-11}, \delta = 5 \times 10^{-2}$ combined with the parametric exploration for the analysis of this property.

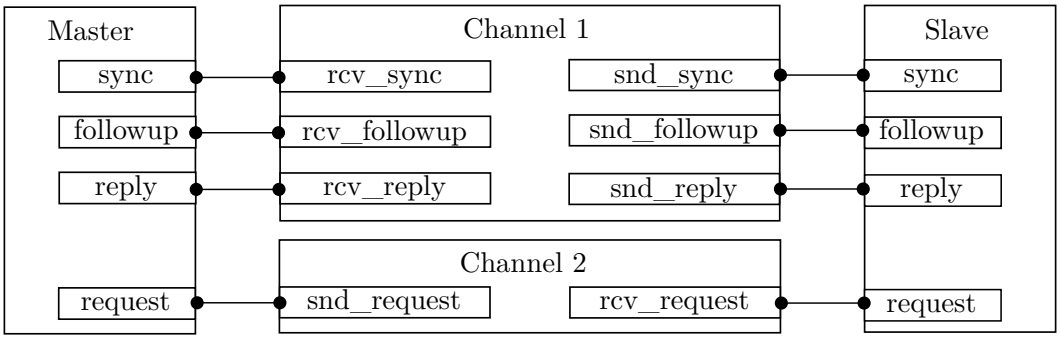
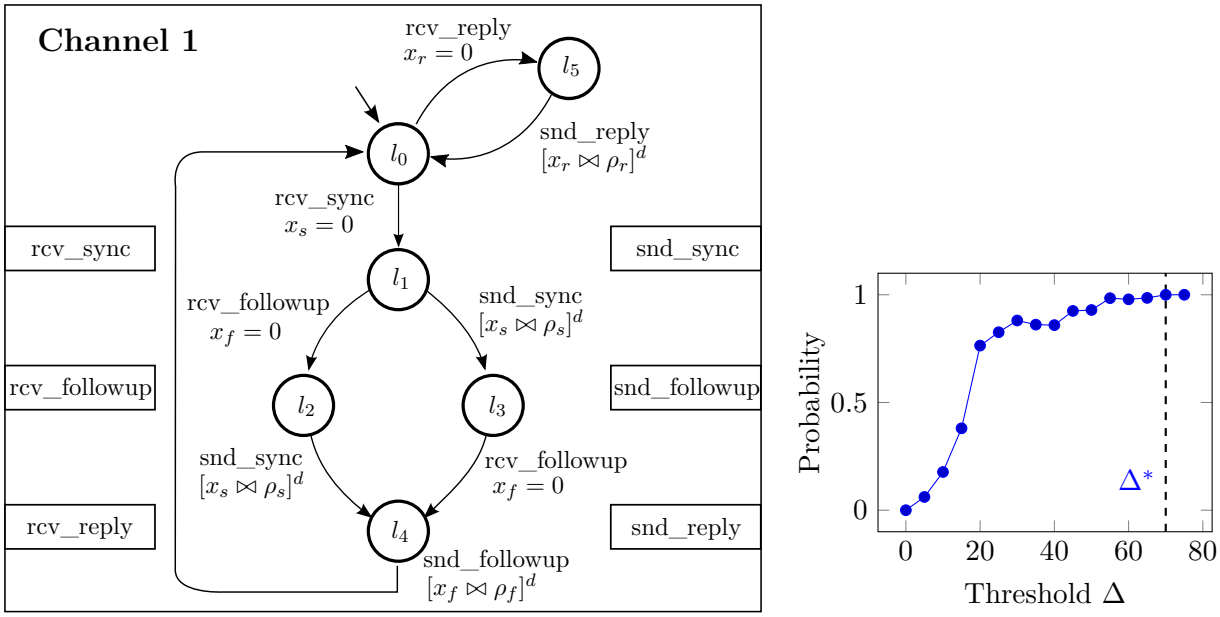


Figure 6.8: The abstract PTP model.



(a) Stochastic model of Channel 1

(b) Probability of ϕ_7

Figure 6.9: Stochastic model and analysis results

6.2 Embedded Systems Case Studies

6.2.1 A Vehicle Gear Controller

The gear controller system is a real-time component embedded in modern cars. It is responsible for implementing the actual gear change requests issued by the driver (or by an algorithm) and transmitted through a communication network. The correctness and performance of the gear controller are important to guarantee a safe behavior of the vehicle. For instance, an excessive time for performing a gear change makes driving unpleasant but may also lead to serious safety problems.

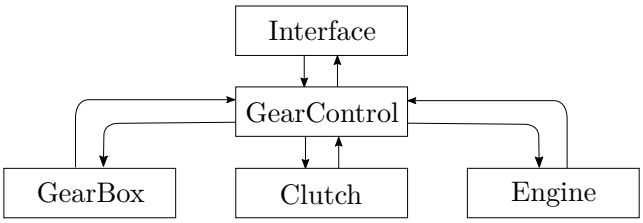


Figure 6.10: Gear controller model

We consider a stochastic real-time BIP model of this system based on the work in [107]. The model is composed of the gear controller component and its environment: a gear change request interface, a gear box, a clutch, and an engine (Figure 6.10). Each of these components obeys to specific timing requirements. For instance, the Clutch can open or close within $100 - 150$ ms, the Gear box, which is electrically controlled, can set (resp. release) a gear in $100 - 300$ ms (resp. in $100 - 200$ ms). The engine operates in 3 modes with different constraints, i.e., normal, zero torque and synchronous speed. In the first mode, the engine gives the requested torque, whereas in the second (resp. third) it tries to find a zero torque (resp. speed) difference with the transmission (resp. the wheels). The maximum allowed time for searching a zero torque (resp. synchronous speed control) is 400 ms (resp. 500 ms). Missing any of these constraints raises errors in the model.

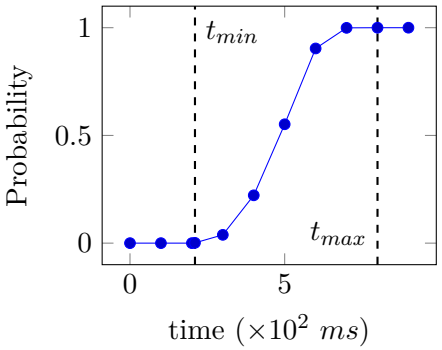


Figure 6.11: Probability of $\phi_8(t)$

In the original work, the authors used reachability analysis to prove several requirements concerning functional and performance aspects. We consider a subset of the original requirements (29 MTL properties, see Appendix A for the complete set of considered requirements). Here, we focus on those concerning the system performance. We provide results for one parametric property $\phi_8(t)$, which states that a complete gear change is always performed within t time units in the case of no errors. It is expressed in MTL as follows:

$$\phi_8(t) \equiv \Diamond_{[0,t]}[(\neg(gb.ErrStat = 0) \vee \neg(c.ErrStat = 0)$$

$$\vee\neg(e.UseCase = 0) \vee \ (gc.GearChanged) \vee (gc.Gear)) \wedge \neg(gc.SysTimer = 0)]$$

We used the probability estimation algorithm with parameters ($\alpha = 5 \times 10^{-7}, \delta = 5 \times 10^{-2}$). The obtained results using the parametric exploration are summarized in Figure 6.11. We observe that the largest time value required to implement a gear change with probability 1 is 800 ms. In the same figure, we can also see that the shortest time with a non-zero probability for a gear change is 210 ms.

6.2.2 Pacemaker Model

A pacemaker is a device implanted on a human heart to cope with malfunctions due to aging or diseases. Its function is to guarantee the temporal relations between atrial and ventricular contractions. This device (see Figure 6.12) acts as a monitor for these atrial and ventricular events and generates electrical pulses to compensate missing/late events and hence, prevents the heart's malfunctions.

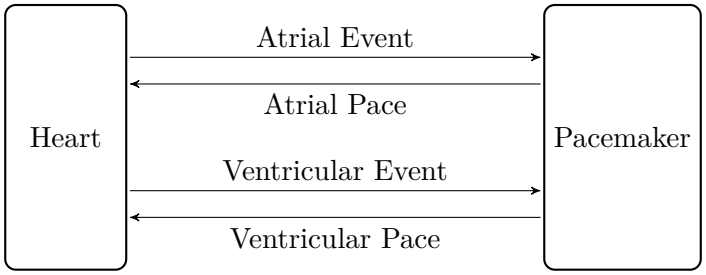


Figure 6.12: Heart and Pacemaker interactions

Our model is a BIP translation of the case study in [91]. In this model, the heart is represented by a component that periodically sends atrial and ventricular contraction events, respectively denoted AS and VS. These events are handled by the pacemaker that may deliver atrial pacing (AP) or ventricular pacing (VP) to regulate the heart timed behavior. The pacemaker is a compound component composed of four components: (*i*) *Lower Rate Interval (LRI)* ensures that the heart rate is above a minimum value by monitoring the elapsed time between ventricular events (VS, VP), generating AP if a time limit of TLRI-TAVI is reached. (*ii*) *Atrio-Ventricular Interval and Upper Rate Interval (AVIURI)* guarantees the delay between an atrial event and a ventricular one by delivering a ventricular pacing in the case where no ventricular contraction is detected within TAVI. This module also tracks delays between ventricular events to avoid pacing the ventricle too fast. (*iii*) *Post Ventricular Atrial Refractory Period (PVARP) and Post Ventricular Atrial Blanking*

Parameter	Value
TLRI	1000
TAVI/TVPR	150
TURI/A_{min}	400
TPVAB	50
TPVARP/V_{min}	100
V_{max}	200
A_{max}	1100

Table 6.2: Parameters for the pacemaker and the heart models

(PVAB) filters noise by ignoring atrial events occurring in TPVARP. (*iv*) *Ventricular Refractory Period (VRP)* ensures a minimum delay TVRP between ventricular events. The parameters for the pacemaker and the heart models are summarized in Table 6.2.

On the one hand, the pacemaker has to monitor the heart rate and verify that the interval between ventricular events, denoted *delta_Vx*, is bounded by TLRI (property ϕ_9). On the other hand, it must not deliver a VP too fast. This amounts to verify that the interval between a ventricular event and a VP, denoted *delta_Vp*, is above TURI (property ϕ_{10}). These properties are formalized as follows:

$$\phi_9 \equiv \Box_{[0,500000]}[delta\_Vx \leq TLRI]$$

$$\phi_{10} \equiv \Box_{[0,500000]}[delta\_Vp \geq TURI]$$

We checked both properties on $\mathcal{S}$BIP using the probability estimation algorithm with parameters ($\alpha = 5 \times 10^{-11}, \delta = 5 \times 10^{-2}$). The analysis required 4883 execution traces, each one representing a simulation time of approximatively 8 minutes. Both properties have been proven true ($P(\phi_9) = P(\phi_{10}) = 1$) in less than $1h30$ per property.

6.3 Concurrency with a Shared Resource

Concurrency is a key concept in systems in general, and programs in particular. Concurrent systems are the ones that can execute independently which can lead to improve the overall execution-time of the systems tasks. One of the most common way to synchronize/communicate this kind of systems is through shared resources. However, one wants to study the fairness in the access to these shared resources.

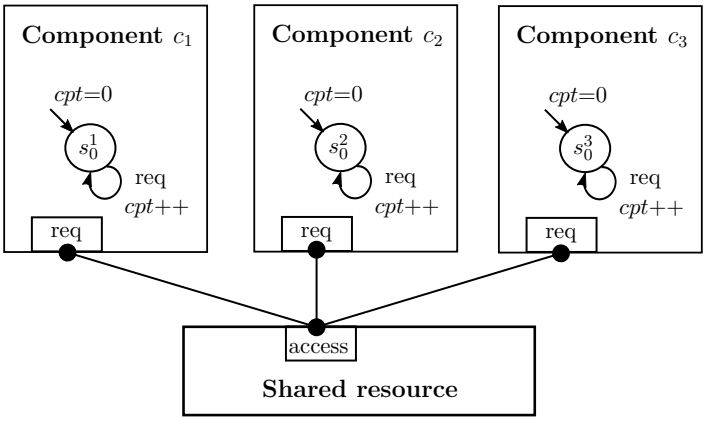


Figure 6.13: A concurrency model with three components sharing a single resource

In this case study, we consider three concurrent components that share a common resource, as depicted in Figure 6.13. Each component c_i memorizes the number of times it accessed the critical

Probability	$P(l_1 \mid l_0)$	$P(l_2 \mid l_1)$	$P(l_3 \mid l_2)$	$P(l_4 \mid l_3)$	$P(l_5 \mid l_4)$
Estimate	0.215	0.234	0.217	0.222	0.227

Probability	$P(l_6 \mid l_5)$	$P(l_7 \mid l_6)$	$P(l_8 \mid l_7)$	$P(l_9 \mid l_8)$	$P(l_{10} \mid l_9)$	$P(\phi_{11})$
Estimate	0.218	0.194	0.209	0.199	0.243	2.35×10^{-7}

Table 6.3: Results of IP on the concurrency model

resource in an integer variable *cpt*. The considered LTL property $\phi_{11} \equiv \Diamond_{\{30\}}(\bigwedge_{i=1}^{3}(c_i.cpt > 9))$ states that, after 30 system steps, each component accesses the shared resource more than 9 times. For ϕ_{11} to be evaluated to true, each component must have exactly 10 accesses out of the overall 30 accesses, corresponding to the 30 system steps, which makes the property rare. The decomposition of this rare property into $n = 10$ levels can be done in a straightforward manner, that is, $l_k \equiv \bigwedge_{i=1}^{3}(c_i.cpt > k - 1), k \in [1, 10]$. This comes from the fact that we want the component to access the resource exactly the same number of times. The stop condition is hence set to 3 steps (one for each component).

We first tried to estimate the probability of property ϕ_{11} using PE with the parameters $(\delta = 1.5 \times 10^{-2}, \alpha = 5 \times 10^{-7})$. This lead us to simulate 33782 traces in which the rare event has never been met $(P(\phi_{11}) = 0)$, in an overall execution time of 3m 37s. However, by using IP with $M = 1000$ traces we were able to estimate the probability of each level to occur and hence $P(\phi_{11})$, in less than 13s. Table 6.3, summarizes the results of IP on the concurrency model. We can see that the probability that concurrent components access exactly the same number of times the shared resource is very low $P(\phi_{11}) = 2.35 \times 10^{-7}$ but not null. It is interesting to see that the conditional probabilities are very close which indicates that the actual decomposition in levels is suitable, and hence reduces the relative variance of the final estimate.

6.4 Tool Performance Analysis

We now provide performance measures of $\mathcal{S}$BIP, mainly regarding time (Table 6.4). The first three columns show respectively the considered case study, the number of components in the associated model, and the properties under test. The two remaining columns illustrate the number of SMC loops in case of parametric exploration and the average SMC time. We observe that depending on the model size and the property complexity, the SMC time can reach a dozen of minutes, which is relatively large compared to other SMC tools.

To get more insights on the reasons of the observed times, we investigated the individual tasks within an SMC loop, i.e., property parsing, trace simulation, trace parsing, and monitoring. We considered the processing time of a single execution trace (with lengths ranging between 10^5 and 2.10^5) of the PTP model and one MTL property (see Section 6.1.3).

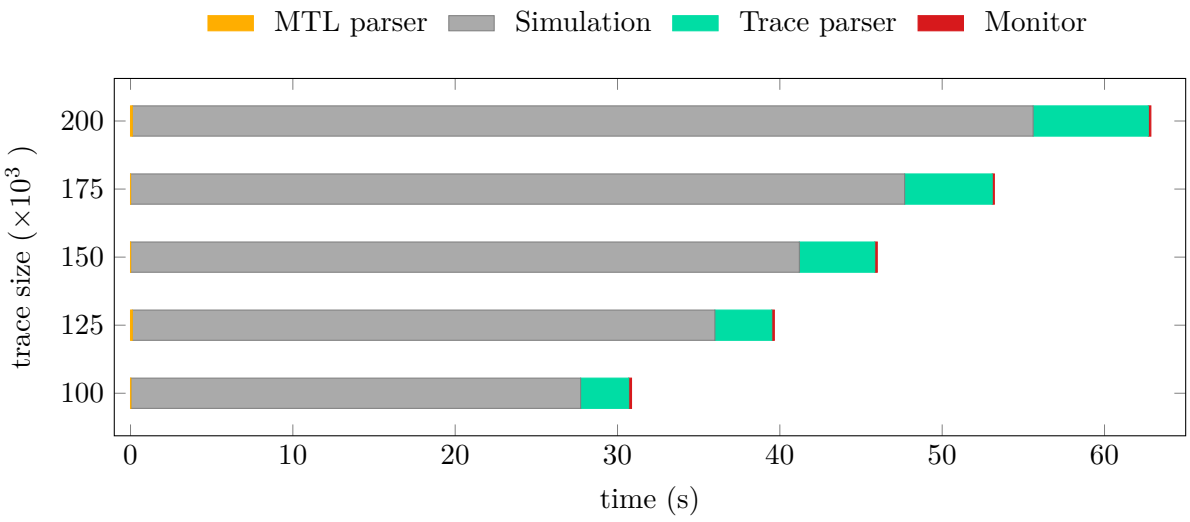


Figure 6.14: Detailed processing times for different trace sizes

Figure 6.14 summarizes the obtained results, which show that the overall processing time grows linearly with the trace size. A noticeable observation is that the simulation time takes almost 90% of the whole analysis. This is mainly due to current prototype implementation of the stochastic real-time engine. Moreover, in this experiment, we systematically logged model variables, which considerably increased the simulation time. Related to this matter, we observe that the trace parsing (including instantiation) is also substantial. The reason for that is mainly strings manipulation. It grows proportionally with the size of the trace. Finally, we notice that the MTL parsing and monitoring require relatively short time and are almost constant in this case, since we considered the same property.

The tool performance can be enhanced at different levels but mainly at the level of the interfacing with the engine. Currently, this latter is done through the standard output stream. Indeed, the engine first writes infor-

Case study	$\#C$	ϕ	#smc loops	avg smc time
Firewire(2)	4	ϕ_1	11	1m 21s
		ϕ_2	9	1m 59s
		ϕ_3	-	2m 28s
		ϕ_4	2	3m 27s
Firewire(3)	7	ϕ_1	17	1m 53s
		ϕ_2	11	3m 34s
		ϕ_3	-	3m 38s
		ϕ_4	3	4m 43s
Firewire(5)	13	ϕ_1	18	3m 54s
		ϕ_2	17	12m 36s
		ϕ_3	-	7m 23s
		ϕ_4	5	10m 16s
Bluetooth v1	3	ϕ_5	9	2m 27s
		ϕ_6	16	3m 11s
Bluetooth v2	3	ϕ_5	11	3m 0s
		ϕ_6	14	13m 05s
PTP	4	ϕ_7	15	8m 42s
Gear Control	5	ϕ_8	11	54s
Pacemaker	5	ϕ_9	-	1h 28m
		ϕ_{10}	-	1h 30m

Table 6.4: Summary of performance

mation on this latter stream. Then at the level of $\mathcal{S}$BIP, this stream is redirected to a string variable that is later parsed to extract a high-level trace. This interfacing significantly slows the tool at two levels: (1) the simulation takes more time due to the stream writes, and (2) the trace consumption is preceded by stream reads and string parsing. As a solution, the engine could be extended to generate serialized trace objects that can be directly retrieved during analysis through the deserialization mechanism.

6.5 Conclusion

In this chapter, we considered several case studies to showcase the tool's analysis capabilities and to assess its performance. In these case studies, we were able to extract useful quantitative information and to verify a variety of properties with high precision and confidence levels. Moreover, we analyzed the tool performance and identified through these investigations a bottleneck due to the current version of the simulation engine. We are currently working on an optimized version which can significantly improve the whole performance of the tool.

In the next chapter, we present a methodology for the design of real-time safety-critical systems. This methodology is developed around $\mathcal{S}$BIP and relies on quantitative risk assessment using SMC, to guarantee system resilience through the introduction of fault detection, isolation and recovery capabilities.

Chapter 7

Quantitative Risk Assessment in the Design of Resilient Systems

The correct system design is in general a hard problem that depends on many factors such as system complexity, requirements satisfaction, tool-chain constraints, etc. In the case of real-time safety-critical systems, uncertainties at runtime also need to be taken into account at design time. Indeed, these situations may have serious implications on the system's execution and mission. Corrective measures to handle them after the system deployment can prove to be very costly, and in some cases impossible to implement.

The type of uncertainties considered in this chapter are faults and failures that occur at system execution. To address these cases and allow for resilience, systems are equipped with *fault detection, isolation and recovery* (*FDIR*) capabilities. FDIR is usually designed as software components that detect whether a fault has happened and apply a predefined sequence of actions that bring the system in a safe mode. The current practice for designing FDIR components follows an ad-hoc process based on the engineers expertise. This process takes into account requirements expressed in natural language and produces implementations that are integrated in the system. We identify several difficulties within this process mainly related to its completeness and automation: (i) faults are identified from informal specification and requirements, (ii) the impact of faults is evaluated manually, and (iii) FDIR implementations are validated by unitary and integration testing with no formal guarantees.

Formal methods have been recently leveraged for correct-by-construction FDIR components [146] in the frame of untimed [31] and real-time [59] systems. In this context, synthesis algorithms are used for building the two parts of the FDIR components: the diagnoser for the fault detection and the controller for the recovery. Two limitations are identified in [59] with respect to the proposed approach: (iv) a diagnoser is devised for each detectable fault, and (v) the controller is manually modeled being left for verification by model-checking techniques. Firstly, synthesizing a diagnoser for each detectable fault has inconveniences since not all faults have an impact on the requirements to satisfy, and the system analysis and performance can

be greatly degraded due to the large number of unnecessary components. Therefore, it is important to synthesize diagnosers only for those faults that are relevant with respect to the system requirements and objectives. This limitation is emphasized by the manual activity of evaluating faults, as described by item (ii) above. Secondly, the controller validation problem is hard and often unfeasible since model-checking techniques suffer from scalability issues.

In this chapter, we tackle the limitations described above by proposing a model-based development approach that relies on quantitative risk assessment and formal methods. The aim of this approach is to design systems resilient to faults in an incremental manner based on model transformation. We consider risk to be any system-related changes that may alter the system nominal behavior or its performance. In our approach, risk is introduced through model transformation, explicitly by modeling faults and implicitly by integrating new FDIR capabilities. Quantitative risk assessment is used to study the impact of such changes and to improve the FDIR design. To tackle limitations (ii) and (iv), we use probabilities to automatically measure risk and to evaluate whether it is tolerable or not. When risk is deemed unacceptable, mitigation would be either to synthesize a new FDIR component or to enhance the existing ones, e.g., with a more appropriate recovery strategy.

In this work, we automate probabilistic risk measurement by using SMC and we leverage its scalability to validate manually designed controller components as described by limitation (v). More precisely, we define an iterative and incremental process for the design of resilient systems equipped with FDIR capabilities. This process is model-based and integrates quantitative risk assessment and system validation which are partially automated using SMC as described in Section 7.1. Moreover, we apply the proposed design process on a real-life robotics case study presented in Section 7.2. We devise three system designs at different levels of granularity on which we perform quantitative risk assessment. For each design, we propose FDIR behavior that we validate against the system's requirements. We use the $\mathcal{S}$BIP framework, presented in Chapter 5, for modeling and quantitative analysis. The obtained results are presented in Section 7.3. Finally, we discuss the advantages and limitations of this process on industrial applications in Section 7.5.

7.1 A Model-based Approach Integrating Quantitative Risk Assessment

The proposed approach, illustrated in Figure 7.1, is based on the idea of iterative and incremental transformation of models Γ. Each model transformation can introduce new risks, for example due to relaxing environment assumptions. The idea depicted in this approach is to perform at each step of the development two assessments. First, *quantitative risk assessment* allows one to measure the impact different risks have with regard to the system requirements, and perform a model upgrade if deemed necessary. Secondly, *validation* ensures that the upgrade is

consistent with respect to the system requirements. Please note that the proposed approach is general enough to be applied to different types of systems, e.g., untimed, real-time. Moreover, the notion of *risk* can have different interpretations, e.g., safety, security. Our setting consists of stochastic real-time systems designed and analyzed with the $\mathcal{S}$BIP framework, where risks are understood and modeled as faults.

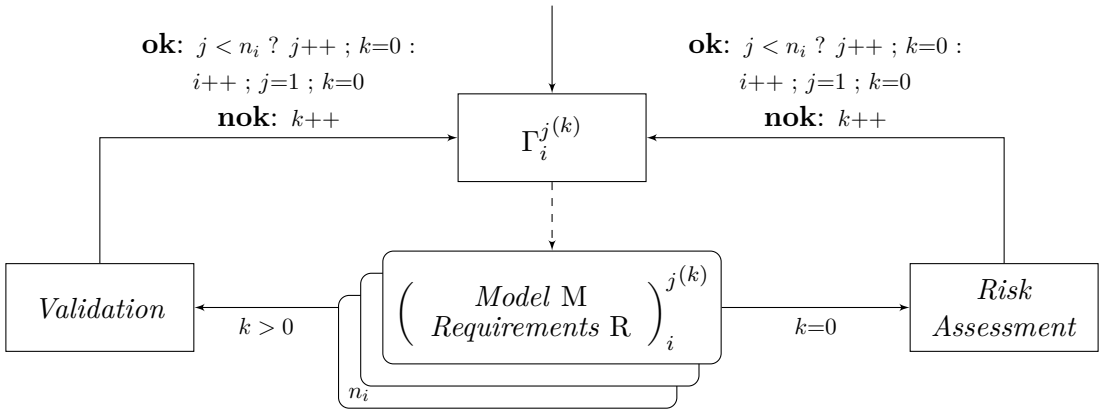


Figure 7.1: Design approach based on formal methods integrating quantitative risk assessment where: Γ denotes model transformation, i is the index of the number of performed steps, j is the index for the number of explored models within a step bounded by n_i, and k is the number of iterative transformations performed on a model. Initially i is set to 0, and j and k to 1.

Initially, system specifications and informal general requirements are analyzed ($\Gamma_0^{1\,1}$) to build a nominal application model ($M_0^{1\,(1)}$) and a set of formal requirements ($R_0^{1\,(1)}$). The only assessment performed at this step is the validation (i.e., $k = 1$): the model should satisfy the formal requirements. While this condition is not satisfied, the model is iteratively transformed, as denoted by the **nok** label and index k in Figure 7.1. When the model is judged valid, one can proceed with the next model transformation step.

The model is incrementally transformed towards the concrete implementation as represented by the i index. Transformation concerns different aspects of the system and may introduce new risks. Transformation examples include integrating new behavior, correction of bugs in the model or legacy code, instantiation of the model's parameters. For the latter, one obtains a family of models indexed with $j \in \{1, \ldots, n_i\}$ in Figure 7.1. Similarly, the system requirements are modified based on the purpose of the performed model transformations. By system requirements we mean those expected to be fulfilled by the system model in the current stage of the design, that is, during the step i the exploration j and the iteration k.

The first analysis to perform on the system model $M_i^{j\,(0)}$ is the *risk assessment*. It implies computing the probability for requirements $R_i^{j\,(0)}$ to hold on the model. Based on this measurement, the risk can be appreciated. If they are acceptable, represented with the **ok** label, one

can continue with inspecting a new model either from the same family if $j < n_i$ or by moving to the next step $i + 1$. If the risks are high, represented with the **nok** label, a *decision* on how to mitigate is taken, which usually involves the transformation of the model architecture and/or behavior. Once all the risks have been dealt with, the obtained model $M_i^{j(1)}$ and its requirements are subject to the iterative validation described above.

In the following sections, we show how to apply this approach on a robotics control system case study. We distinguish four levels of granularity for this case study. At level 0, we design a nominal application model that we validate with respect to initial requirements. At level 1, we introduce risks in the form of faults and perform risk assessment. The decision consists of introducing FDIR functionality in one of the components of the model, which we then validate. At level 2, a performance-related model transformation is applied that impacts only the set of requirements. We show that the FDIR behavior introduced is necessary but not sufficient with respect to performance and we propose an improvement that is again validated. Finally, at level 3, we introduce in the design the deployment model. We instantiate the deployed model that considers 3 aspects and we explore several deployments using risk assessment with respect to performance-specific requirements. All models are described in the stochastic real-time BIP formalism where time evolves probabilistically following uniform density functions. The risk assessment and validation activities are automated using the $\mathcal{S}$BIP tool.

7.2 Planetary Robotics Case Study

The case study considered in this work is an excerpt of a Bridget Rover demonstrator control system developed for the validation of the ESROCOS environment [3]. The Bridget Rover (see Figure 7.2) is a representative of Martian rovers aiming for planetary exploration while providing a modular and configurable payload bay. The payload consists at least of a panoramic camera for image acquisition aiming for biological signatures detection and autonomous driving via map construction. The rover uses 6 wheels to drive and steer, where the motors communicate with the locomotion system via a CAN-bus.

Figure 7.2: The Bridget Rover (courtesy of Airbus Defense and Space UK).

7.2.1 System and Requirements Overview

The control system developed with ESROCOS [2] aims to remotely drive the rover using a joystick and acquire images. In consequence, the developed system interfaces with the low level control of the locomotion system and the CAN-bus node. We consider in the following the drive with a joystick functionality of the developed control system as case study [58].

More precisely, the case study consists of the software chain between the joystick and the locomotion software. The main prerequisite for the system is that the rover is moving according to the requested motions. This feature is formalized by many requirements based on the granularity of the system design and assumptions made over the environment, as listed in Table 7.1. For example, ϕ_0 describes that the requests sent by joystick are received by the locomotion software, while ϕ_1 describes that the locomotion software receives requests regularly (with a given period of $100ms$).

ID	Formal specification
Requirements on the nominal system	
ϕ_0	$\square_{[0,10000]}\ (is\_sent\ \Rightarrow \Diamond_{[0,100]}\ is\_received\_c)$
ϕ_1	$\square_{[0,10000]}\ (is\_received\_c\ \Rightarrow \Diamond_{[1,100]}\ is\_received\_c)$
Requirements on the FDIR system	
ϕ_2	$\square_{[0,10000]}\ (is\_sent\ \Rightarrow \Diamond_{[0,110]}\ is\_received)$
ϕ_3	$\square_{[0,10000]}\ (is\_received\ \Rightarrow (\Diamond_{[1,110]}\ is\_received)\ \vee\ (\Diamond_{[110,200]}\ is\_timeout))$
ϕ_4	$\square_{[0,10000]}\ (\Diamond_{[0,200]}\ nb\_received = nb\_sent\ +\ nb\_timeout)$
ϕ_5	$\square_{[0,10000]}\ (cnbt \geq MNBT\ \Rightarrow \Diamond_{[0,200]}\ is\_reset)$
ϕ_6	$\square_{[0,10000]}\ (is\_reset\ \Rightarrow \Diamond_{[0,100]}\ is\_received)$
Requirements on the system performance	
$\phi_7(n)$	$\square_{[0,10000]}\ (\Diamond_{[0,200]}\ nb\_timeout - (nb\_received - nb\_sent) \leq n)$
$\phi_8(n)$	$\square_{[0,10000]}\ (nb\_timeout \leq n)$
$\phi_9(n)$	$\square_{[0,10000]}\ (cnbt \leq n)$
$\phi_{10}(n)$	$\square_{[0,10000]}\ (nb\_overflow \leq n)$
$\phi_{11}(x)$	$\square_{[0,10000]}\ (nb\_chosen\_timeout \leq x)$
$\phi_{12}(y)$	$\square_{[0,10000]}\ (nb\_chosen\_read \leq y)$
$\phi_{13}(n)$	$\square_{[0,10000]}\ (\Diamond_{[0,100]}\ period\_id - cmd\_id \leq n)$

Table 7.1: Requirements of the planetary robotics case study at the different levels of granularity of system design.

7.2.2 Nominal Software Design

7.2.2.1 Model

As mentioned above, the case study tackles the communication of the joystick with the locomotion control system. The software architecture is given is Figure 7.3.

The software design relies on a library containing two types of components: *triggers* and *queues*. The trigger, illustrated in Figure 7.4a, is a component that with a customizable period P activates the component to which it is attached (action *sig_out*). Once activated, the trigger waits for the completion of the component's associated behavior (action *sig_return*) before a deadline D. In case the associated behavior does not finish before D, an issue is raised during

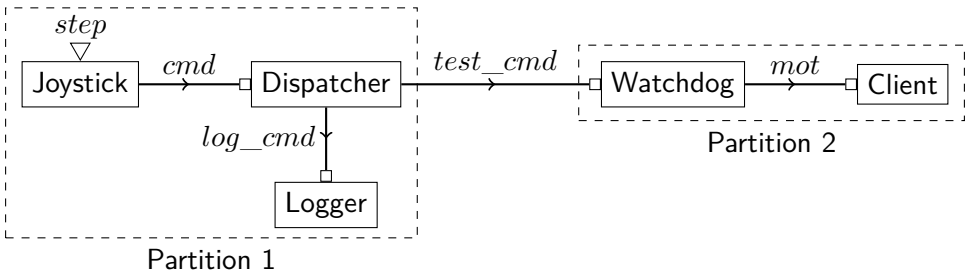


Figure 7.3: Overview of the case study software architecture.

the analysis. This corresponds to a timelock in the system behavior, that is a modeling error and has to be corrected.

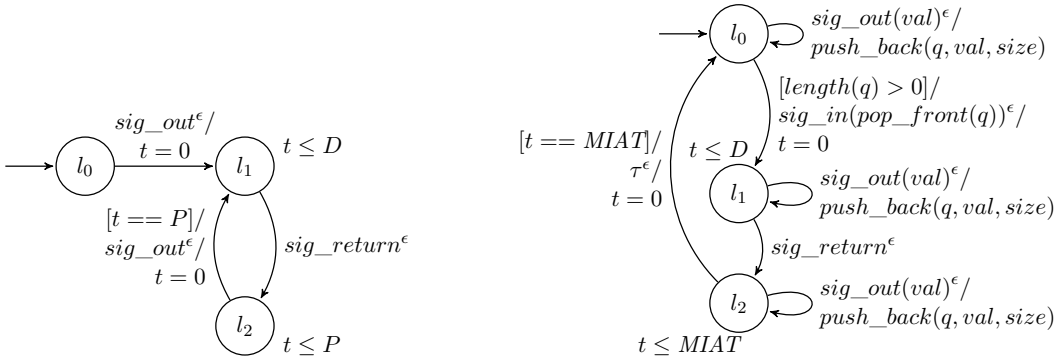
The queue, illustrated in Figure 7.4b, is a component that models the asynchronous communication between two components: a sender and a receiver. It is associated with the receiver component and it contains a buffering structure of fixed *size* to store received requests (on action *sig_out*). If the limit has been reached, the incoming requests are discarded. When available, the receiver consumes the requests stored in its queue, using action *sig_in*, with a fixed minimal time between two reads, called *MIAT* (minimal inter-arrival time). Similarly to the trigger, the queue waits for the completion of the receiver's associated behavior (action *sig_return*), that must happen before a deadline D.

The Joystick component regularly sends a motion command denoted *cmd* to the rover locomotion software to be executed. This behavior is depicted in Figure 7.5a by the states l_0 to l_4, represented in black. The command sending is activated by the *step* trigger. For this case study, the *step* period and deadline are set to $100ms$ and $15ms$ respectively. The *cmd* request has multiple fields to describe the motion to be executed: an *id* of type integer records the package number, and *motion* records the actual data package sent to the locomotion software consisting of translation, rotation and heading of type float.

The *cmd* action is first sent to a Dispatcher, provided in Figure 7.5b. The Dispatcher transfers this request to two software components: first the Logger via the *log_cmd* request and then to the Watchdog via the *test_cmd* request (states l_0, l_3, l_4 and l_6 in black). Both *log_cmd* and *test_cmd* contain the data package received from the Joystick. The Logger records the received requests such that they can be reused later for the validation of the system through replaying.

The Watchdog interfaces the Dispatcher (and subsequently the Joystick) with a wrapper of the locomotion control software called Client, as illustrated in Figure 7.5c by the states l_0 to l_3 in black. Whenever a *test_cmd* request is received, the Watchdog transfers the data package in the *mot* request to the Client. For simplicity, we do not consider the behavior of the Client, which is abstracted to discarding all received *mot* requests.

All these components communicate asynchronously via queues. All queues have the *MIAT* and D set to $50ms$ and $15ms$ respectively, while they can record only one element, i.e., $size = 1$.



(a) Generic behavior of triggers with parameters P and D

(b) Generic behavior of queues with parameters $MIAT$, D and $size$

Figure 7.4: Library of components and their behavior: triggers represented with triangle ($\triangledown$) and queues represented with square ($\square$) in Fig. 7.3.

7.2.2.2 Validation Requirements

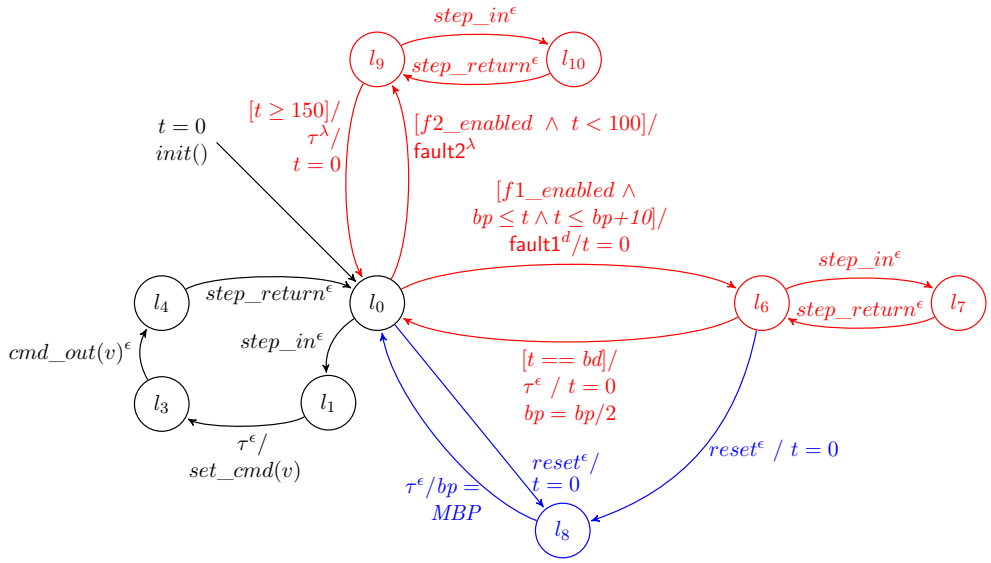
For this case study, we are interested first in validating the nominal behavior of the system when the environment assumptions are satisfied:

1. the Joystick issues periodically a *cmd* request,
2. no requests (data packages) are lost, and
3. all executions including queue writing and reading take $0ms$.

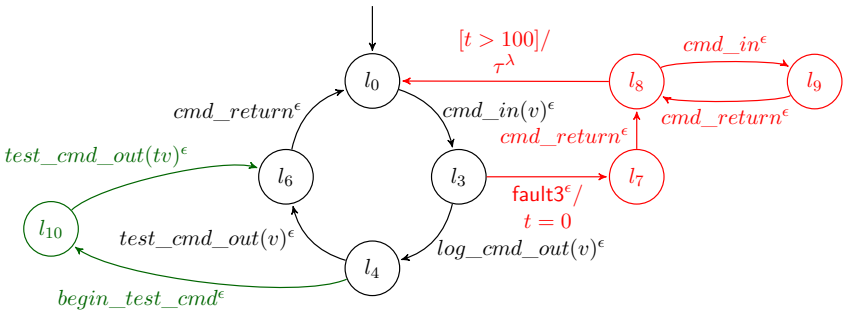
The system's prerequisite – the locomotion system executes the received motion commands – is expressed on the nominal application by the requirements ϕ_0 and ϕ_1. ϕ_0 describes that all *cmd* requests sent by the Joystick, modeled with the Boolean variable is_sent, are received within $100ms$ by the locomotion system (Client component), modeled with the Boolean variable $is\_received\_c$. ϕ_1 describes that the Client regularly receives a request *mot*, within $100ms$. Please note that these requirements and their formalization are relaxed with respect to assumption 3 as they describe cyclic behavior within periods.

7.2.2.3 Validation Results

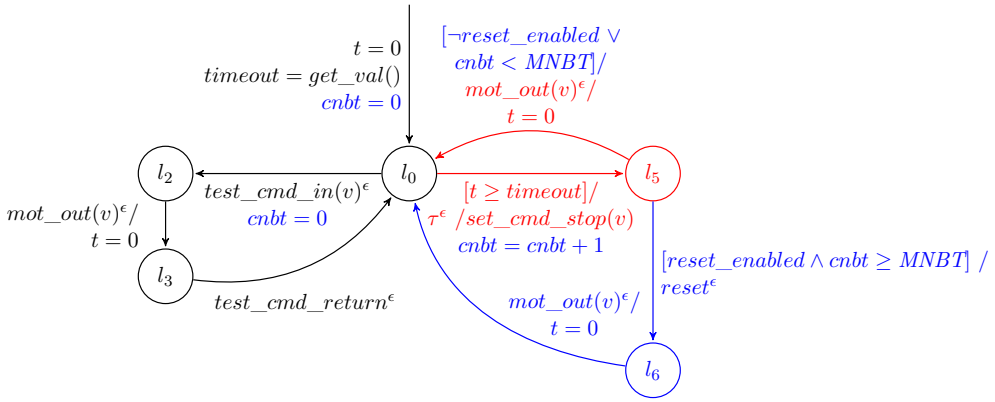
We use the $\mathcal{S}$BIP tool with the confidence parameters $\alpha = 0.005$ and $\delta = 0.05$ for all our experiments (at this step and after). These confidence parameters require the evaluation of 1199 system executions to come up with a global verdict, using the probability estimation technique. The computed satisfaction probability is 1 for both ϕ_0 and ϕ_1. The computations took for each requirement independently roughly 4 minutes.



(a) Behavior of Joystick



(b) Behavior of Dispatcher



(c) Behavior of Watchdog

Figure 7.5: Behavior of the main components from Fig. 7.3 represented as timed automata in $\mathcal{S}$BIP, where faults, fault detection and standard recovery action are represented in red, more complex recovery strategy in blue, and deployment-specific actions in dark green.

7.2.2.4 Conclusion

As the requirements on the nominal application are satisfied, we can proceed with the next step of the approach.

7.3 Risk Assessment of the Planetary Robotics System

In the following, we describe two models at different levels of granularity and the results of the risk assessment and validation phases on them: one system level design including faults modeling relaxed assumptions and one deployment level design additionally modeling the hardware platform.

7.3.1 On Robustness to Faults

7.3.1.1 Model with Faults

The first transformation of the nominal model ($i = 1$) tackles the assumptions 1 and 2 described in Section 7.2.2. We relax these assumptions in the new model by incorporating faults. In order to have a systematic way of evaluating the impact of faults and their combinations on the system and its requirements, we define Boolean constants for each fault to control their injection.

Assumption 1 describes that the Joystick sends periodically a request. We break this hypothesis by modeling the stop of request sending for a certain amount of time with two faults as follows. The fault1 action is related to the external code that can be embedded in the model. More specifically in this case, the software of the Joystick component does not send *cmd* for a certain duration. This behavior is represented in Figure 7.5a by the red states l_6 and l_7, as follows. The fault is injected at state l_0 with the Boolean constant $f1\_enabled$ and can be executed any time between bp and $bp + 10$. (bp is an integer variable that describes the break period of not sending *cmd* requests.) Once the transition is picked for execution, in a uniform way, the clock t that measures the break duration bd is initialized to 0. During the $[0, bd[$ time interval, any step triggers are discarded by the Joystick as modeled with the $l_6 - l_7$ cycle. Once the break duration bd has passed, the Joystick recovers, the clock is reset and the break period is updated. For this example, bd is equal to $20ms$ and bp is set to $190ms$. Please note that we model here a persistent fault since the break period is decreasingly converging to $0ms$, and therefore the Joystick will eventually be continuously failing.

The fault2 action is motivated by the risks in the hardware connections, where the Joystick can be unplugged non-deterministically for a certain moment. This behavior is modeled in Figure 7.5a by the red states l_9 and l_{10}. As before, this fault is injected at state l_0 by the Boolean variable $f2\_enabled$ and can be executed as long as $100ms$ have not passed since the last clock reset. If the fault occurs, the Joystick could recover after $150ms$ since last time reset

(the transition from l_9 to l_0). However, the recover action is defined as *lazy*, which implies that the Joystick could fail for large time periods. During the fail, any step triggers are discarded as above.

Assumption 2 describes that no motion command is lost. We relax this hypothesis in the Dispatcher, where we model the message loss with the fault3 action. This behavior is represented in Figure 7.5b by the red states l_7, l_8, and l_9. As above, we inject the fault with the Boolean constant $f3\_enabled$ and we consider that this fault can happen after a *cmd* is received. Then the Dispatcher has the choice of forwarding the data package (transition between l_3 and l_4) or losing the data package (transition l_3 to l_7). If a *cmd* is lost, the Dispatcher will continue to loose packages (cycle $l_8 - l_9$) unless it recovers. The recovery cannot happen before $100ms$ since fault3. Again, the recovery is *lazy* (transition from l_8 to l_0), which means that the Dispatcher can loose commands for large time periods.

7.3.1.2 Risk Assessment Requirements

For the risk assessment, we use the requirements ϕ_0 and ϕ_1 defined above, and we do not introduce other ones. These requirements are sufficient to quantify and assess the impact of all faults – fault1, fault2, and fault3 – have on the nominal behavior of the system.

7.3.1.3 Risk Assessment Results

With the $\mathcal{S}$BIP tool, we obtain the following results. Regarding fault1 and fault2, ϕ_0 is satisfied with probability 1. Indeed, if a command is not sent the implication evaluates to *true*. However, this probability drops to 0 when injecting fault3. In this case, some sent commands are lost by the Dispatcher. Therefore, they are not received by the Client. Property ϕ_1 is not satisfied regardless of the faults occurring, independently or combined, since commands are either not sent or lost within the $[1, 100]ms$ time interval. The computed probability is equal to 0. The complete results, including the time needed for the experiments, are given in Table 7.2.

Given these results, we conclude that these faults (i.e., risks) have a great impact on the system and an FDIR behavior needs to be added such that the rover operates safely. Indeed, without a recovery strategy, the rover's locomotion system would keep executing the last received command and this can have important consequences. For example, the rover could be stuck into a harmful environment and therefore not achieve its mission. More specifically, we are interested in stopping the rover whenever such faults are detected.

7.3.1.4 Model with FDIR Behavior

In order to stop the rover consistently when risks are present, we equip the Watchdog with a data package validity checking before transferring the request to the Client. By validity we do not mean the checking of the command content (even though this could be achieved if necessary), but ensuring that *the package must be received before a* timeout *event*. This

	fault1 (without reset)		fault1 (with reset)		fault2		fault3		fault4 ($MTD = 0/MTD = 5$)	
	Probability / Parameter	Time (sec)	Probability / Parameter	Time (sec)	Probability / Parameter	Time (sec)	Probability / Parameter	Time (sec)	Probability / Parameter	Time (sec)
Nominal system										
ϕ_0	1	240	1	196	1	229	0	14	-	-
ϕ_1	0	14	0	13	0	14	0	14	-	-
System with FDIR functionality										
ϕ_2	1	240	1	196	1	229	0	14	0.05 / 0	14 / 96
ϕ_3	1	202	1	242	1	211	1	180	1	300
ϕ_4	1	192	1	246	1	215	0	14	0.05 / 0	14 / 105
ϕ_5	-	-	1	194	-	-	-	-	-	-
ϕ_6	-	-	1	217	-	-	-	-	-	-
$\phi_7(n)$	$n^*_{\phi_7(n)}$=0	140	$n^*_{\phi_7(n)}$=0	140	$n^*_{\phi_7(n)}$=0	140	$n^*_{\phi_7(n)}$=88	900	$n^*_{\phi_7(n)}$=4	2160
$\phi_8(n)$	$n^*_{\phi_8(n)}$=88	720	$n^*_{\phi_8(n)}$=61	197	$n^*_{\phi_8(n)}$=88	720	$n^*_{\phi_8(n)}$=88	900	$n^*_{\phi_8(n)}$=49	2160
$\phi_9(n)$	$n^*_{\phi_9(n)}$=88	720	$n^*_{\phi_9(n)}$=5	180	$n^*_{\phi_9(n)}$=35	1800	$n^*_{\phi_9(n)}$=30	1080	$n^*_{\phi_9(n)}$=1	480

Table 7.2: Results obtained with the $\mathcal{S}$BIP framework on the system design with faults and with respect to requirements from Table 7.1. n^*_ϕ refers to the parameter value for which $\phi(n)$ is satisfied with probability 1.

corresponds to the diagnoser part of FDIR components, and it is based on the property that the Client awaits periodically for motion requests.

If this does not happen due to faults in the system, the Watchdog will ensure that the rover still operates safely with respect to the locomotion system: *the motion is stopped*. To achieve this, the Watchdog sets the data package to stop – translation, rotation and heading are set to 0 – and sends *mot* with stop as value. This corresponds to the controller part of FDIR components.

For simplicity[1], we model this FDIR behavior directly in the Watchdog as illustrated in Figure 7.5c. While waiting for a *test_cmd*, the Watchdog checks the time elapse with the clock t. If the timeout duration has been observed (transition from l_0 to l_5 in red), the stop data package is set, the corresponding *mot* command is sent and the clock is reset (transition from l_5 to l_0 in red). We configure the value of the timeout to $110ms$ in order to account for possible delays in the system execution. As a consequence, the requirements listed in Table 7.1 for the FDIR component and described below use this new time bound in their formalization.

We can now proceed with validating the FDIR behavior of the Watchdog component.

7.3.1.5 Validation Requirements

We define a new set of requirements ϕ_{2-4} specific to the FDIR behavior of the Watchdog. ϕ_2 describes that whenever a motion command is sent, it is received by the Watchdog. This requirement is the transformation of ϕ_0 from the Client to the Watchdog as receiver. Indeed, there is no need to check ϕ_1 from now on, as the new FDIR behavior should guarantee it. With ϕ_3 we are interested to check that the Client should receive *mot* requests periodically. The *mot* could contain either the data package sent by the Joystick (modeled with the *is_received* variable) or the stop data package sent by the Watchdog (modeled with the *is_timeout* variable). Note that ϕ_3 is also a transformation of ϕ_1 from the Client to the Watchdog by including the FDIR part. Requirement ϕ_4 models the consistency of the data package reception: all the data packages received by the Client are either commands generated by the Joystick or by the Watchdog.

7.3.1.6 Validation Results

The Watchdog is robust with respect to faults fault1, fault2, and fault3: the probabilities computed for ϕ_3 are 1, as showed in Table 7.2. In addition, ϕ_2 and ϕ_4 are also satisfied when considering the faults of the Joystick, namely fault1 and fault2. However, in the presence of fault3 of the Dispatcher, the probability of satisfaction for ϕ_2 and ϕ_4 is 0. This result is expected since fault3 models message loss. Whichever combination of faults between the two components is considered, the results are similar: ϕ_3 is satisfied, while ϕ_2 and ϕ_4 are not.

[1]The system architecture and specification, Watchdog included, have been provided in the frame of this case study such that the used resources (e.g., number of components and threads) are minimal.

7.3.1.7 Conclusion

This step of the approach consisted of the following actions. The system model was enriched with faults and the need of FDIR behavior was highlighted by the risk analysis and evaluation results. The decision was to include the detection and recovery capabilities into the Watchdog component which now also checks the validity of the received commands. We showed that the Watchdog is robust with respect to the modeled faults. As the obtained results are satisfying, we go to the next step $i = 2$ in our approach.

7.3.1.8 On System Performance

7.3.1.9 Model for Performance Measurement

At this step we are interested in the performance of the FDIR behavior of the Watchdog in the system. Therefore we do not perform any model transformation – $\Gamma_2^{1(0)}$ is the identity function. However, we enrich the set of requirements with ones evaluating the system performance ϕ_{7-9}, as described below.

7.3.1.10 Risk Assessment Requirements

ϕ_7 explores the different bounds for inconsistency in the number of packages. This requirement makes sense to be checked when the system looses commands, i.e., when fault3 is present or $P(\phi_4) \neq 1$. ϕ_8 explores the maximal number of stop commands the Watchdog issues for the given time period, while ϕ_9 considers the number of consecutive stop commands. We are interested in this case to have a low number of stop commands such that the rover operates smoothly.

7.3.1.11 Risk Assessment Results

The results obtained from the risk assessment are given in Table 7.2. We remark that fault1 leads the Watchdog to issue a large number of (consecutive) stop commands ($\phi_{8,9}$) due to its persistence. As we will see in the next refinement, we tackle this aspect by introducing a reset mechanism. fault2 and fault3 imply the same large number of stop commands from the Watchdog as fault1. However, we remark that in these cases, the system recovers for longer periods and acts consistently since the consecutive number of stop commands is bounded to 35 and 30, respectively, as illustrated in Figure 7.6 and 7.7. The maximal failure period is approximately equal to $35 \times 100ms$ and $30 \times 100ms$, respectively. Therefore, the Watchdog is able to keep the system safe, even with long failure periods.

7.3.1.12 Model with *reset* Mechanism for the Joystick

Since fault1 is persistent and the rover will be mostly not moving due to the stop commands issued by the Watchdog, we add a reset mechanism in the Joystick that will allow this component

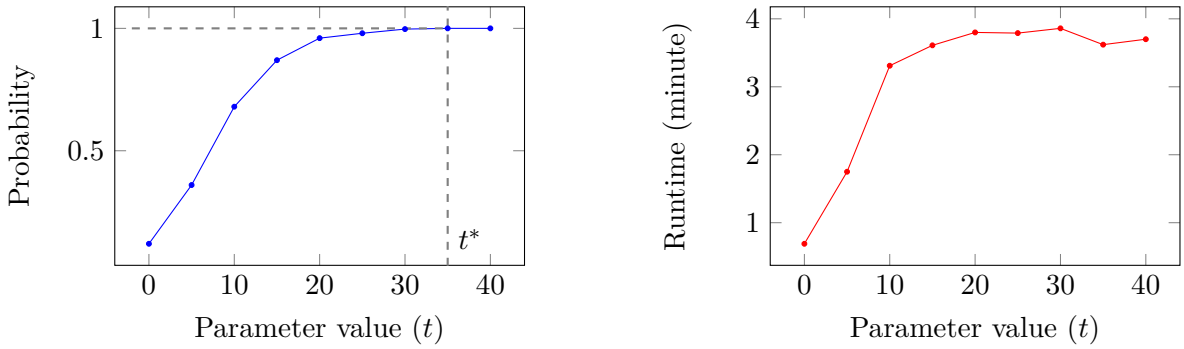


Figure 7.6: Probability and runtime of ϕ_9 for the model including fault2.

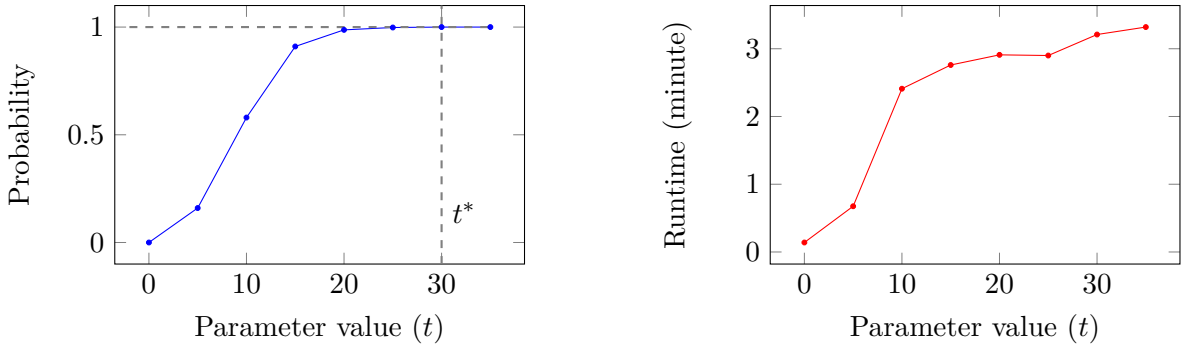


Figure 7.7: Probability and runtime of ϕ_9 for the model including fault3.

to go back to its nominal behavior. This mechanism, illustrated in Figure 7.5a in blue, consists of an action *reset* (leading to state l_8) which sets the break period *bp* back to the maximal allowed duration *MBP* (transition from l_8 to l_0). Then the Joystick will again issue motion commands, with fault1 enabled.

The reset mechanism is controlled by the Watchdog since it implements FDIR behavior and it is enabled with a Boolean constant *reset_enabled* (as for fault injection). As illustrated in Figure 7.5c (in blue), the Watchdog defines a variable *cnbt* that stores the consecutive number of stop commands issued. If *cnbt* is below the *MNBT* threshold, the Watchdog will issue a stop command (transition from l_5 to l_0). Otherwise, the Watchdog will first trigger the reset mechanism (transition from l_5 to l_6) and then will issue the stop command (transition from l_6 to l_0). Please note that the behavior of the Watchdog described in Section 7.3.1 is identical to the one described in this figure, when *reset_enabled* is set to *false*.

7.3.1.13 Validation Requirements

The efficiency of the reset mechanism is additionally validated by requirements ϕ_5 and ϕ_6. ϕ_5 describes that whenever the Joystick starts repeatedly failing, the Watchdog triggers the

reset action. ϕ_6 validates the efficiency of the reset mechanism modeled by the receiving of a command by the Watchdog after a reset is triggered. We also check and compare the performance of the reset mechanism with requirements ϕ_{7-9} described above.

7.3.1.14 Validation Results

For this model, we configure the Watchdog to tolerate a maximum number of 5 consecutive timeouts before triggering a Joystick reset ($MNBT = 5$).

In the second column of Table 7.2, we see that both ϕ_5 and ϕ_6 are satisfied with probability 1. From the performance point of view, we observe an improvement of order of magnitude for the number of stop commands issued by the Watchdog. More specifically, the total number of issued stop commands is reduced by 31%, whereas the number of consecutive stop commands is bounded to 5. The latter corresponds in general to the bound $MNBT$ for which the reset mechanism is implemented, instead of the computed bound of 88 without the reset mechanism.

An interesting result is obtained when combining fault1 implementing the reset mechanism and fault3. In this case, requirement ϕ_6 does not hold. The reason is that the reset mechanism does not guarantee the receiving of commands by the Watchdog at the next cycle. This is mainly due to the fact that the consecutive timeouts could be caused by the Dispatcher and in that case, resetting the Joystick will not change anything. Another refinement is necessary in order to develop a more resilient system by implementing more complex recovery strategies.

7.3.1.15 Conclusion

At this step, we checked the performance measurements of Watchdog in the system. We observe that in some cases the Watchdog is efficient. In others, as for example fault1, a more complex recovery mechanism based on the *reset* of the Joystick is implemented and validated. It is showed that the robustness property (ϕ_3) is preserved by the model with the reset mechanism, and moreover this mechanism reduces the overhead of the Watchdog on the system performance. Therefore, we proceed by transforming and studying a deployment model of our system.

7.3.2 On Deployment Impact

7.3.2.1 Deployed Model

The software is deployed on two partitions due to the rover architecture. The locomotion control software, and therefore its Client wrapper, are deployed on a partition which communicates with the other software components via a CAN-bus. Since the Watchdog component aims to check the validity of the requests sent to the Client, it will be deployed on the same partition with the Client – Partition2 in Figure 7.3. The remaining components – Joystick, Dispatcher, and Logger – are deployed on another partition – Partition1 in Figure 7.3. Between the two partitions a Channel component is considered.

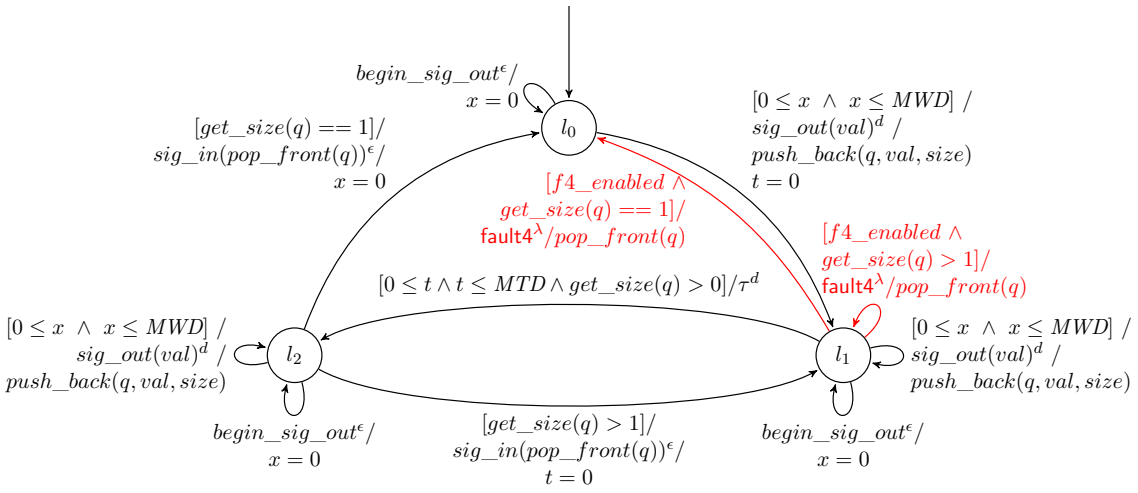


Figure 7.8: Behavior of the communication channel between the two partitions of Fig. 7.3.

The Channel is a component added to the system model and connected to the Dispatcher in writing mode and to the Watchdog in reading mode. This component, illustrated in Figure 7.8, has a similar behavior to the queues. Once a data package is received, it is written in a buffer of a predefined *size*. If the Channel buffer is full, the received data package is discarded. Finally, the recorded data package is removed from the buffer and transferred to its target when possible.

We relax here assumption 3 and we model *maximal transmission* and *writing delays* for the Channel with variables *MTD* and *MWD*, respectively. The maximal transmission delay describes how much time a data package transfer can take at most. For example, such a variation in the transmission time can depend on the network load. This behavior is modeled by the transition from l_1 to l_2 in Figure 7.8: a transfer can finish at any moment between 0 and *MTD* (the delayable d urgency). Similarly, the maximal writing delay describes the time the Dispatcher can take to write a motion command in the Channel buffer before continuing its behavior (here, informing the Dispatcher queue that another request can be handled). Usually writing on a channel is not instantaneous. We consider that the writing can take between 0 and *MWD* as described by the guard on transition from l_0 to l_1 and loop transitions in states l_1 and l_2. The writing process is initialized by the Dispatcher with the *begin_sig_out* action. This action can always be performed on the Channel, as modeled by the self-loop transitions labeled with *begin_sig_out* in all states (Figure 7.8).

Moreover, we enable the loss of command requests with fault4. More precisely, a package stored in the buffer (state l_1) can be lost at any moment (the lazy λ urgency). Similarly to the other faults, we use the Boolean constant $f4\_enabled$ to inject it. This fault implies the removal of the oldest data package from the buffer. This can result in an empty buffer (transition from l_1 to l_0) or a buffer with at least one data package (loop transition in l_1).

In this setting, fault1 to fault3 are disabled. These faults can be considered in a further step together with the current deployment model.

7.3.2.2 Risk Assessment Requirements

We consider for evaluation requirements ϕ_{2-13} from Table 7.1. Please note that ϕ_5 and ϕ_6 are not checked since they make sense only when fault1 is present. ϕ_{10} explores the number of *mot* requests lost by the Client queue: incoming requests are lost if the queue is full. ϕ_{11-13} tackle the freshness of the data package received by the Client. We want to determine with ϕ_{11} how many times the Watchdog triggers a stop command when a motion command is present in its queue. ϕ_{12} is the dual of ϕ_{11}: how many times the Watchdog handles a command received at timeout. Finally, ϕ_{13} explores the discrete time difference (in terms of periods) between when a command is received and when it is issued.

7.3.2.3 Risk Assessment Results

For this analysis, we consider 3 aspects of the Channel in the deployed model: (1) $M_3^{1(0)}$ with transmission delays, (2) $M_3^{2(0)}$ with writing delays, and (3) $M_3^{3(0)}$ with command losses.

Transmission delays. This exploration concerns the *MTD* parameter of the model, with $MWD = 0$. The results obtained with $\mathcal{S}$BIP for this model on requirements $\phi_{2-4,7-10}$ are represented in Figure 7.9. The evolution of the probability estimation for ϕ_{2-4} and $\phi_{10}(0)$ is plotted on the left, while the evolution of the optimal parameter value for ϕ_{7-10} is displayed on the right. Notice that both the probabilities and the optimal parameter values are functions of the maximal transmission delay (*MTD*) represented on the x-axis, and which is a parameter of the model as described above. The optimal parameter value for a property ϕ is the smallest parameter value for which the property is satisfied with probability 1. We write the optimal parameter as $n^*_\phi = min_{n\in\mathbb{N}}\{\mathcal{P}(\phi(n)) = 1\}$.

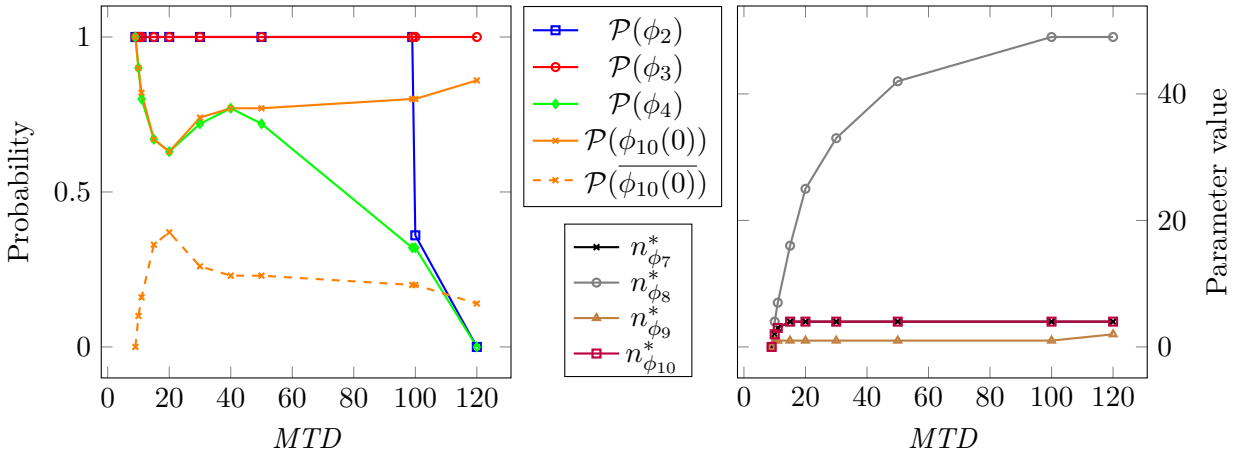


Figure 7.9: $\mathcal{S}$BIP results for the deployed model including transmission delays.

We start by presenting the parametric exploration results on requirements ϕ_{7-10}. The total number of stop commands as described by ϕ_8 is in general bounded to 49. The number of

consecutive stop commands, as expressed by ϕ_9, is bounded to 1 for MTD values below $100ms$. Similarly, the number of lost commands due to the Client queue overflow modeled in ϕ_{10} is bounded to 4 motion commands. We remark that $n^*_{\phi_7} = n^*_{\phi_{10}}$, which shows that the Client queue overflow is the only source of command losses in the system.

The Client queue can discard a message due to being full if all of the following three conditions are satisfied in the given order: (i) a *test_cmd* arrives at exactly timeout for the Watchdog, (ii) the Watchdog chooses to first issue a stop command, (iii) the Watchdog immediately transfers the *test_cmd* as *mot*, without the Client handling the previous stop data package. Then, the queue of the Client which size is 1 will not be able to store the *mot* and this data package is dropped. This situation evolves with respect to MTD as follows. When MTD increases, the number of stop commands issued also increases as showed for ϕ_{11} up to a certain limit. However, the probability for the *test_cmd* sent by the Dispatcher to take exactly timeout $- P$ to be delivered to the Watchdog is decreasing (i.e., it boils down to generating a single value in the growing interval $I = [0, MTD]$ of transmission delays).

Therefore, we observe that the probability to loose commands, represented by the negation of $\phi_{10}(0)$ on the left hand side of Figure 7.9, first increases until $MTD < 20ms$ and then decreases. The increase is justified by the higher impact the choice of sending stop commands instead of transferring the motion requests (see the results for ϕ_{11} in Figure 7.10) has on the property. After $20ms$, the aforementioned choice stabilizes and the probability to generate a single time value in I decreases, leading $\mathcal{P}(\overline{\phi_{10}(0)})$ to also decrease.

Finally, on the left hand side plot of Figure 7.9, we observe that ϕ_2 is satisfied when MTD is lower than $100ms$. However, in the cases where the transmission delay is greater than the $100ms$ period of the Joystick, the delivery of commands is no longer guaranteed in the same period. It is worth mentioning that the Watchdog is able to detect that phenomenon and act accordingly, as reflected by requirement ϕ_3 that is satisfied with probability 1. With respect to requirement ϕ_4, we remark that the interleaving of command generation and reception also can have an effect on the probability of satisfaction, besides the command loss. The interleaving can happen when the MTD is greater than $40ms$ (as expected from the timeout value) and therefore the satisfaction probability of ϕ_4 decreases. Up to $40ms$, ϕ_4 and $\phi_{10}(0)$ are identical, while after $40ms$ they diverge.

Next, we are interested in quantifying the number of times the Watchdog has the non-deterministic choice between issuing a stop command (ϕ_{11}) or reading a command from its queue (ϕ_{12}). These results are showed in Figure 7.10. We observe that the number of non-deterministic choices, represented by the sum of x^* and y^* values, varies with the MTD then stabilizes at the value of 6 when the transmission delay is above $20ms$. Also, the Watchdog chooses fairly between the two options. An interesting observation is that the number of chosen stop commands is relatively low in comparison with the total number of issued stop commands, as reflected in Table 7.3.

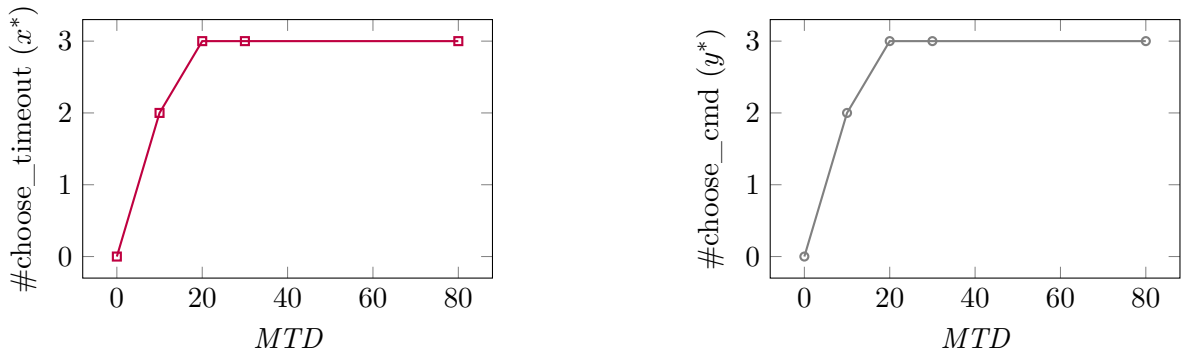


Figure 7.10: Parametric exploration of ϕ_{11} (left) and ϕ_{12} (right) on the deployed model with transmission delays

MTD	Max_timeouts ($n^*_{\phi_8}$)	#choose_timeout (x^*)	Proportion $x^*/n^*_{\phi_8}$
10	4	2	0.5
20	25	3	0.12
30	33	3	0.09
80	45	3	0.07

Table 7.3: Proportion of non-deterministic stop commands when increasing *MTD*.

Lastly, we focus on analyzing motion command and period identifiers to evaluate data freshness. Due to transmission delays, a command can be received in a different period than the one it was issued in. We express this behavior with ϕ_{13} and the results are shown in Figure 7.11. The left plot represents the variation of the probability estimation of $\phi_{13}(n)$ for different instances of n when varying the *MTD*. The right plot represents the optimal value of n for different values of *MTD*. We can see that the difference between the period of issued and received command identifiers is bounded and increases with the growth of *MTD*. Hence the transmission delays have an important impact on data freshness.

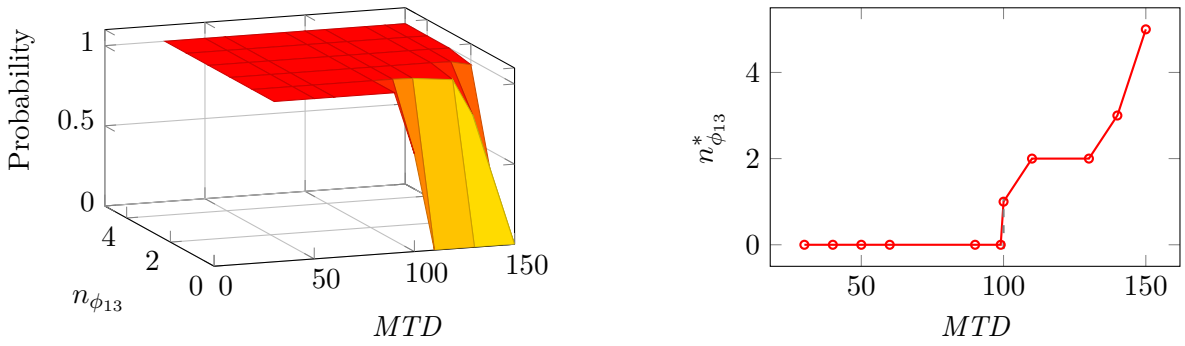


Figure 7.11: Parametric exploration of ϕ_{13} on the deployed model with transmission delays.

In order to deal with lost commands in the Client queue, a refinement could be performed by increasing the size of this queue. A binary search can be used to determine the minimal size. In our case, a queue of size 2 is sufficient to avoid overflows, as shown in Figure 7.12 for acceptable ($30ms$) and high ($100ms$) values of MTD. The probability to not loose any commands in the Client's queue ($\phi_{10}(0)$) is equal to 1. In the case where $MTD = 100ms$, the difference in the number of commands received by the Client and those sent by the Joystick and Watchdog expressed by ϕ_4 is only due to the interleaving of actions.

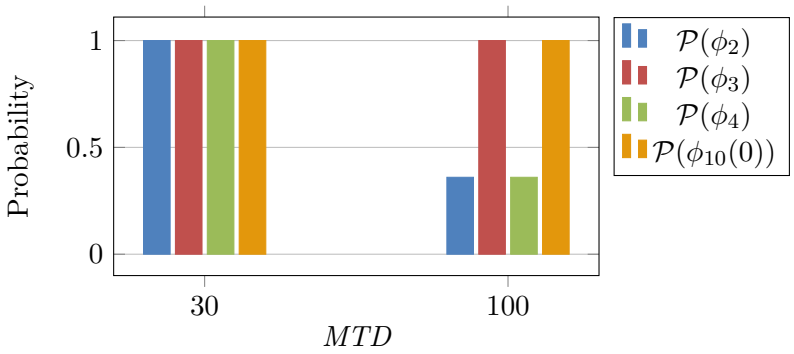


Figure 7.12: Results on the corrected deployed model.

Writing delays. This exploration concerns the MWD parameter of the model, with $MTD = 0$. Recall that the D (deadline) of the Dispatcher queue is set to $15ms$. Indeed, the Dispatcher has to transfer the cmd request as log_cmd (which takes $0ms$) and as $test\_cmd$ via the Channel (which takes at most $MWDms$). All these actions must happen before the D implemented by the queue, otherwise the Dispatcher timelocks. Timelocks are modeling errors that can be detected during model analysis and can be subject to model transformation: for example, either set a new worst case execution time for the component and in consequence the system scheduling is recomputed, or a recovery mechanism (similar to the reset mechanism proposed) is implemented in the Dispatcher.

The results are provided in Figure 7.13. In the left hand side plot, we illustrate the probabilities estimated for ϕ_{2-4}. When $MWD < 15ms$, we observe that the Watchdog is robust (ϕ_3). Additionally, analysis results for $\phi_{2,4}$ in this setting are similar to the results in the transmission delay setting. When $MWD \geq 15ms$, most requirements do not hold anymore due to the timelock of the Dispatcher. However, the Watchdog keeps on guaranteeing the system's safety, as shown by the probability of ϕ_3 evaluated to 1. Note that $\phi_{10}(0)$ is equivalent to ϕ_4 because there is no interleaving possible between the command generation and reception when the $MWD < 40ms$.

On the right hand side of Figure 7.13, we illustrate the optimal parameter values obtained on properties ϕ_{7-9}. These results are similar to those obtained in the transmission delay setting

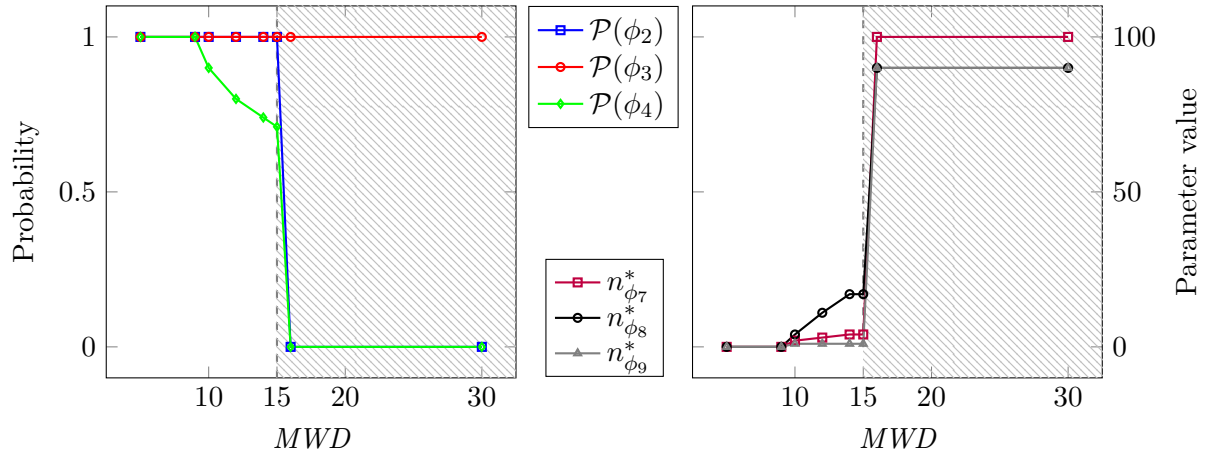


Figure 7.13: SMC results for the deployed model with writing delays

for $MWD < 15ms$. Note that in this case $\phi_{10}(n)$ is identical to $\phi_7(n)$, and therefore it is not explored.

Command losses. Finally, we consider the command loss aspect of the Channel. For our exploration we set $MTD \in \{0ms, 5ms\}$ and $MWD = 0ms$. The results with respect to $\phi_{2-4,8-10}$ are given in Figure 7.14. The results for requirements ϕ_{11-13} are not relevant in this case since MTD should be greater than $10ms$ and $40ms$ for a relevant exploration of ϕ_{11-12} and ϕ_{13}, respectively.

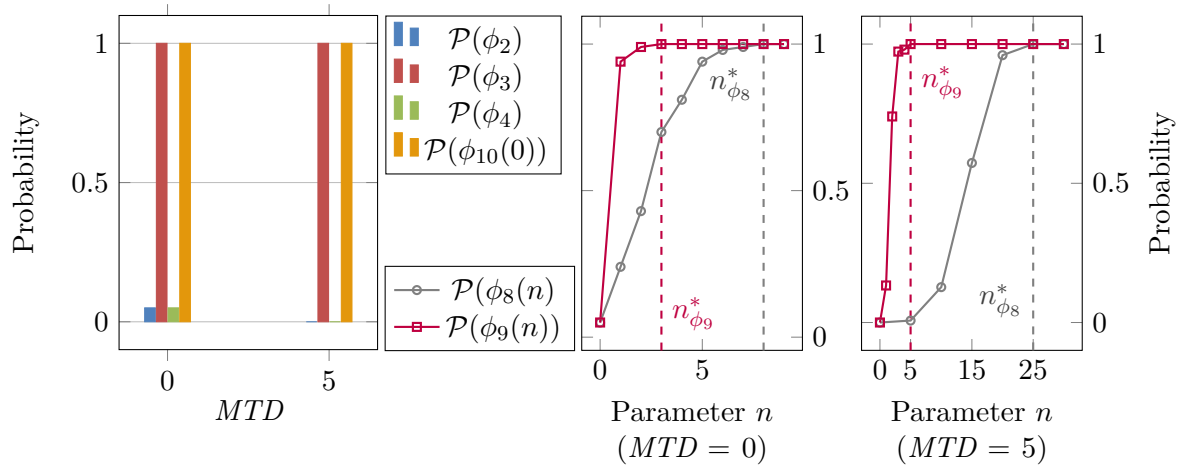


Figure 7.14: SMC results for the deployed model with command losses

We remark that ϕ_3 is always satisfied. Requirements $\phi_{2,4}$ reflect the occurrence of commands losses: the satisfaction probability is 0.05 when $MTD = 0ms$, and 0 when $MTD = 5ms$. Property $\phi_{10}(0)$ is satisfied with both values of the transmission delay. This shows that commands are not lost in the client's queue due to an overflow; the only source of command losses in this

model is fault4. Consequently, the number of timeouts encodes the number of times the fault occurred. When there is no transmission delay, the number of lost commands is bounded to a value of 8 ($n^*_{\phi_8}$) with a maximum of 3 consecutive losses ($n^*_{\phi_9}$), as shown in Figure 7.14 for $MTD = 0ms$. These numbers drastically increase when $MTD > 0ms$. For instance when $MTD = 5ms$ illustrated in Figure 7.14, the Channel can loose up to 25 commands ($n^*_{\phi_8}$) with a maximum of 5 consecutive losses ($n^*_{\phi_9}$).

7.3.2.4 Conclusion

We have modeled a deployment for our case study taking into account three risk cases: transmission delays, writing delays and command losses. For each of these aspects we have explored ϕ_{2-13} to evaluate the impact of these risks on the behavior of the rover. Based on the shown results, 2 scenarios are further possible depending on other high level requirements. If the hardware constraints obtained through exploration are implementable, the designer can continue with the deployment. Otherwise, a model transformation and a new exploration are required to correct the undesired behaviors and identify the needed hardware for the deployment, as briefly described for the transmission delay risk above.

In this design, we note that whatever the risk is, the Watchdog is robust and the system's safety with respect to the motion functionality is guaranteed. This is also true for the timelock modeling error present in the Dispatcher that occurs for writing delays starting from $15ms$. Before $15ms$, the transmission and writing delays have a similar impact on the system behavior. For higher values, however, these risks have an impact on different aspects of the system. Transmission delays can imply the non-satisfaction of data freshness, meaning that commands are no longer received in the $100ms$ period since their generation. Writing delays can lead to command losses and the generation of numerous stop data packages. In the latter case, the system enters a degraded mode where generated data packages are no longer delivered.

A shared effect of these risks on the locomotion system is the overflow in the Client queue. To address this, a refinement can be necessary, such as the one proposed above where the Client queue size is increased to 2. Therefore, we can safely state that the model does not loose any command and the timelock in the Dispatcher is avoided as long as $MWD < 15ms$. Additionally, if $MTD < 100ms$, the data freshness property is satisfied. Given the deployed system model and the explorations we performed, the values obtained here are optimal in the sense that they guarantee all requirements desired for the system.

7.4 Related Work

The risk assessment activity defined in [84] can be applied to different application domains, including computer-based systems. A survey about the challenges and current practices for risk assessment in computer-based systems is presented in [154]. In the literature, safety risk

assessment is studied from two points of view: qualitative or quantitative. Qualitative safety assessment determines what scenarios lead the system from a nominal mode to a degraded mode where safety requirements do not hold. The practice consists of building safety artifacts such as fault trees or timed failure propagation graphs and analyzing them in order to certify the system safety. For example, automated safety analysis for fault trees is described in [25] for the AltaRica dataflow language. In [30] the xSAP tool is presented for the analysis of fault trees and timed failure propagation graphs in the context of symbolic transition systems à la nuXmv.

Quantitative safety assessment provides probabilistic measures of the risks in the systems, such as the likelihood of failure. Probabilistic computations are usually done manually on the safety artifacts build a priori, on which the probability distributions for faults are added. Some safety assessment tools automate this analysis, such as xSAP for probabilistic fault trees. The work presented in this chapter contributes to this class of risk assessment methods. In our case we use SMC in order to compute the probability for the system to fail as described by its requirements.

Safety risk assessment can be seen as an optimization in the design of FDIR components in general and diagnosers in particular. The correct design of FDIR components from complete system specifications has been studied from methodological point of view in [31, 59, 146]. Implementations are provided in [59] for timed systems with partial observability and in [31] for untimed systems à la nuXmv. While [31] includes the safety assessment mechanism implemented in [30] for user-modeled timed failure propagation graphs, this question is left open in [59]. Our contribution completes the work from [59] by defining and automating a quantitative safety assessment method for (stochastic) timed systems allowing for the efficient design of FDIR components.

Finally, our case study tackles the rigorous design of a robotics control system. In [6, 26], (RT)BIP and (RT)DFinder tool are used to model and verify safety and performance requirements such as causality in service executions, mutual exclusion and data freshness. The TINA model checker is used in [67] to verify schedulability properties of a robotics application tasks on a given hardware platform. In [141] safety properties for modular robots such as conflicting commands, self-collision, etc., are checked and simulated for different configurations. Diagnosers are implemented using formal models for robotics control software in [57] with respect to safety properties and using the P language. In [119], differential dynamic logic is used to model and verify the behavior of ground robots, while a diagnoser to ensure the nominal behavior is synthesized with ModelPlex and added to the system. The contribution presented in this chapter has been used for the development of the FDIR components in the robotics systems scenarios presented in [121, 125].

To the best of our knowledge, this work is the first to use statistical model-checking for quantitative risk assessment in the design of resilient systems in general, and for FDIR behavior in particular.

7.5 Discussion

In this chapter, we propose a model-based design approach that relies on formal methods to develop real-time resilient systems. The method is incremental: it starts from the nominal model, then transformations are applied to take into account different sources of risks. The impact of the considered risks is evaluated using a quantitative risk assessment method and FDIR components are introduced accordingly. These are then validated against safety and performance properties. The approach was successfully used for the design and validation of the control software of a planetary rover. The proposed approach is centered around the $\mathcal{S}$BIP tool in order to perform quantitative risk assessment and validation activities automatically.

Approach. Following a model-based approach for the design of FDIR components and their validation provides a lot of flexibility and allows one to explore various situations rapidly. Combined with formal methods, it provides more confidence in the obtained results given that the built models are faithful, which is not trivial and requires some expertise. Finally, the use of statistical model checking automates quantitative risk analysis, and helps to deal with real-life system models. However, both the identification and the evaluation of risks remain manual and subject to the designer's interpretation.

Case study. The results presented in this chapter are part of the work realized for the validation of the ESROCOS environment [121] with a real-life robotics case study. Although the approach was successfully applied and the designed system is currently being tested in field trials, we wish to share some of the challenges we faced. Building faithful models is by far the most challenging. The choice of the appropriate abstractions to perform and the probability distributions to use requires a deep knowledge of the system under analysis. Using risk assessment helped to take well founded decisions in order to build robust FDIR components. However, the notion of risk is large and several times we found ourselves analyzing risks at different levels, such as risks due to faults then risks due to adding new FDIR behavior, etc. Moreover, managing the transformed models and the associated requirements can quickly become cumbersome if not methodically performed.

Tools. Risk analysis automation is primordial for the design of complex systems as the design space is substantial and proceeding manually is not feasible. In our case, once we built a model it becomes almost straightforward to analyze it using $\mathcal{S}$BIP. Nevertheless, some difficulties remain to use the tool properly, like the correct formalization of requirements in MTL or the instrumentation of the model in order to perform SMC.

Future work. In this work, we only considered quantitative risk assessment. Using qualitative assessment before may help a lot in filtering irrelevant risks with respect to the requirements

of interest. Moreover, risk identification could be done in a knowledge-based manner by using machine-learning techniques for instance. Finally, we are also interested to evaluate the applicability of the approach to security risk assessment. Indeed, we believe that our approach is general enough to also handle security requirements.

In the next chapter, we present a security risk assessment approach for the analysis of systems defenses, that we built around the $\mathcal{S}$BIP tool. This approach aims at identifying defense configurations making sophisticated attacks harder to achieve in terms of required resources and their success probability. Defense configurations are rated based on the increase they introduce in the cost of an optimal attack strategy. This latter is explored using a variant of Genetic Algorithms (GA), denoted Intensified Elitist GA (IEGA), that relies on SMC to compute its fitness function.

Chapter 8

Assessing Systems Security with SMC

Modern organizations strongly rely on information and communication technologies in their daily activities. This reliance raises serious questions about the security threats that may be occasioned because of their inherent vulnerabilities and the way to mitigate the risks accompanying them. The damages that a cyberattack exploiting such vulnerabilities might cause, e.g. [82], highlight the urgent need for organizations to integrate risk assessment activities as part of their main processes. Security risk assessment consists of analyzing and evaluating systems vulnerabilities in order to design reliable security policies.

Cyberattacks usually combine various techniques that exploit different vulnerabilities to circumvent deployed defense configurations. Such combinations are generally referred to as *Attack Strategies*. Reasoning at this level turns out to be more suitable than trying to fix individual vulnerabilities, especially that these are difficult to detect. *Offensive security* aims at identifying reliable defense configurations for a system by exploring attacks exploiting its vulnerabilities.

All is about resources. Both attack and defense actions require resources in order to be achieved. For instance attack actions require equipment and take time to be set up. Accordingly, they have some probability of success, i.e., actions that require a limited amount of resources generally have lower probability of success and conversely. Similarly, defense actions are subject to budgetary considerations (equipment, tools, training, etc.) and do generally provide overlapping protection mechanisms, hence they are not required to be deployed simultaneously. Therefore, it is primordial for organizations to be able to quantitatively analyze and evaluate potential defense actions in order to design configurations that prevent cyberattacks while involving a sufficient set of defense mechanisms.

Diverse attackers profiles can be observed in practice with regard to resources utilization. Some would settle for attack actions requiring limited resources, accepting a low probability

of success, while others would privilege actions with high probability of success and allocate resources for that. These profiles are generally the product of various human factors such as experience, budget and motivations. A sophisticated attack strategy would try to optimize these criteria, namely, to find trade-offs requiring an affordable (within a given budget) amount of resources with an acceptable probability of success.

In this work, we propose a risk assessment approach to synthesize defense configurations making sophisticated attacks harder to achieve. Concretely, we consider resources (e.g., the cost) required by an attack to be the hardness criterion. The rationale is that since a sophisticated attack tries to optimize the cost with respect to the probability of success, defense actions that increase this cost are expected to prevent those attack strategies from being achieved with high probability. Relevant defense configurations are hence those involving a sufficient set of defenses with the highest impact on the attack cost.

As opposed to [69] that relies on reinforcement learning, our approach combines Statistical Model Checking (SMC) [79, 151] with Genetic Algorithms (GA) [115], paired with a model learning algorithm (IOALERGIA) to synthesize advanced attack strategies, which serve as a basis for exploring relevant defense configurations. The proposed approach uses Attack-Defense Trees [95] as a representation of the organization's security breaches, the potential attacks that could exploit them and the deployed defense configuration.

The methodology introduced in this chapter analyzes the defenses deployed in a corporation's information/organizational system exhibiting an *a priori* known set of vulnerabilities at different stages of its design. At early design stages, the system is only described by its requirements from which security experts can extract (1) potential vulnerabilities and (2) their probabilities to succeed (based on their experience and security statistics, e.g. [48]), and can suggest (3) generic defense mechanisms for each vulnerability. These information constitute an abstracted representation of the system. Later on in the design process, when the system nominal behavior and vulnerabilities are modeled in more details, a more precise risk assessment can be performed by substituting the abstracted system model with the detailed one.

Depending on the level of abstraction of the system, we consider different types of attackers, namely, *unstructured* and *structured*. Indeed, for abstracted systems, the approach [118] takes into account a model of an attacker that arbitrarily simulates attack actions targeting the systems. When dealing with such unstructured attackers, the goal is to find *probabilistic* attack strategies that solve the non-determinism in the arbitrary selection of attack actions while optimizing the attack cost and success probability. For detailed system models, we consider a structured attacker that takes advantage of the system structure to discover existing dependencies between the system's vulnerabilities. Consequently, we focus on exploring *deterministic* strategies that indicate which attack action to perform at each step of an attack.

The remainder of the chapter is organized as follows. In Section 8.1, we formally introduce the considered models for risk assessment. The proposed framework is presented in Section 8.2. The technique for the synthesis of an impactful defense configuration and for attack strategies

exploration are respectively presented in Sections 8.3 and 8.5, together with a recall of the IOALERGIA algorithm in Section 8.4. In Section 8.6, we evaluate the proposed methods on several case studies. We discuss related work in Section 8.7. Finally, Section 8.8 concludes the chapter and presents a discussion.

8.1 Modeling Systems with Vulnerabilities and Defenses

In this section, we formally introduce definitions and notations used in the remainder of the report. We first introduce the models for an attacker and a defender. Then, we recall the definition of an attack-defense tree, and finally, we describe the model used for risk assessment.

For the following definitions, we consider Σ_A to be a set of attack actions, Σ_D is a set of defense actions, and $\Sigma = \Sigma_A \cup \Sigma_D$ the set of all actions. Furthermore, we consider that each attack action $a \in \Sigma_A$ is associated with 1) a time interval $[l_a, u_a]$ that represents lower and upper time bounds allowed to perform a, 2) a cost $c_a \in \mathbb{R}$ which models needed resources to perform a and 3) a probability of success p_a that represents the likelihood for a to succeed when performed. We call environment, denoted env, the success probabilities of attack actions in Σ_A.

8.1.1 Attacker, Defender and Attack-Defense Tree

8.1.1.1 Attacker

The attacker model represents all possible attack combinations, given the alphabet of attack actions Σ_A. It is syntactically defined as follows:

Definition 8.1.1 (Attacker)**.** An attacker $\mathcal{A}$ is a tuple $\langle L, l_0, T \rangle$ where :

- $L = \{l_0, \ldots\}$ is a set of locations, where l_0 is the initial location,
- $T \subseteq L \times \Sigma_a \times L$ is a set of labeled transitions of the form (l_i, a, l_j).

Intuitively, an attacker $\mathcal{A}$ performs a sequence of attack actions by choosing each time among the enabled ones. At a given state, an attack action a may succeed, leading to a new state where a is no more enabled [1] and where all other actions remain unchanged. In case a fails, the state of the attacker does not change. The success or failure of a selected attack action is not controlled by the attacker, but is determined by the environment env. We formally define the behavior of an attacker as follows.

Definition 8.1.2 (Attacker semantics)**.** The semantics of an attacker $\mathcal{A} = \langle L, l_0, T \rangle$ is the labeled transition system $\langle S, s_0, R \rangle$, where:

[1] This reflects a realistic behavior expressing the monotony of an attack.

- $S = L \times V_{\Sigma_A}$, where $v \in V_{\Sigma_A}$ is a state vector that contains the status of all the attack actions in Σ_A (succeeded or not), i.e., $V_{\Sigma_A} = \{v : \Sigma_A \longrightarrow \{0,1\}\}$,

- $s_0 = (l_0, v_0)$ is the initial state, where $v_0 = [0, \ldots, 0]$ is the initial status of all the attack actions in Σ_A,

- $R \subseteq S \times \Sigma_A \times S$ is a set of transitions of the form (s_i, a, s'_i) respecting the following rules:

  1. Success: $\frac{(l_i,\ a,\ l'_i) \in T,\ v_i(a) = 0,\ v'_i(a) = 1,\ \forall a' \neq a\ v'_i(a') = v_i(a')}{((l_i,\ v_i),\ a,\ (l'_i,\ v'_i))}$
  2. Failure: $\frac{(l_i,\ a,\ l'_i) \in T,\ v_i(a) = 0}{((l_i,\ v_i),\ a,\ (l_i,\ v_i))}$

We use the notation $status(a, s)$ to denote the status of the attack action a at state $s = (l, v)$, i.e., $status(a, s) = status(a, (l, v)) = v(a)$.

Note that the attacker semantics above is non-deterministic, that is the choice of an attack action at each state is performed non-deterministically. When performing an attack, an attacker has to cope with two kinds of non-determinism: (i) the non-determinism on the attack action choice, and (ii) the non-determinism on the success/failure of the chosen attack action. We introduce the notion of attack *strategy* to cope with the former non-determinism, while the latter is enforced by the system, and more precisely, it is resolved by the environment *env*. In this work, we manipulate two kinds of strategies, namely, deterministic and probabilistic ones.

Deterministic attack strategy. An attack strategy $\mathcal{S}_d : S \longrightarrow \Sigma_A$ is said deterministic if it associates each attacker state with an attack action to be selected by the attacker. The strategy solves the non-determinism by a priori determining which action to select in the case where several alternatives are possible.

Probabilistic attack strategy. A probabilistic attack strategy $\mathcal{S}_p : \Sigma_A \longrightarrow [0, 1]$ is defined as a mass probability function that associates each attack action with a probability of being selected by the attacker. In this work, we restrict to static strategies, i.e., the same in any state. Considering dynamic strategies is a future work. Thus, the probability $P : S \times \Sigma_A \longrightarrow [0, 1]$ to select an attack action a at any state s_i is defined as

$$P(s_i, a) = \begin{cases} 0 & \text{if } status(a, s_i) = 1 \\ \frac{\mathcal{S}_p(a)}{\sum\limits_{a' \in \Sigma_A} \mathcal{S}_p(a') \times (1 - status(a', s_i))} & \text{otherwise} \end{cases}$$

Given a deterministic/probabilistic attack strategy $\mathcal{S}$, we denote by $\mathcal{A}|_{\mathcal{S}}$ the attacker $\mathcal{A}$ that applies the strategy $\mathcal{S}$.

8.1.1.2 Defender

A defender models the deployed set of defense actions. In this work, it represents a static defense configuration, where a defense action $d \in \Sigma_D$ is either enabled or not in all the states of the system. It is defined as follows:

Definition 8.1.3 (Defender)**.** A defender $\mathcal{D} \subseteq \Sigma_D$ is the subset of enabled defense actions in Σ_D.

We define a predicate $enabled : \Sigma_D \rightarrow \{0, 1\}$ that tells if a defense action is currently enabled. Formally, $enabled(d) = 1$ when $d \in \mathcal{D}$, and 0 otherwise.

8.1.1.3 Attack-Defense Tree

It represents some knowledge about the system under analysis. For instance, it includes the attack combinations (with respect to the analyzed system vulnerabilities) that may lead to the success of an attack, along defense mechanisms available in the system. In this work, we define it as a Boolean combination of attack and defense actions as follows:

Definition 8.1.4 (Attack-Defense Tree)**.** An attack-defense tree $\mathcal{T}$ is defined by the following inductive grammar:

$$\phi, \phi_1, \phi_2 ::= true \mid ap \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid (\phi), \ where\ ap \in \Sigma$$

The evaluation of the attack-defense tree considers the attacker and the defender models. This evaluation is performed as part of the risk analysis procedure based on a Risk Assessment Model introduced below.

8.1.2 Risk Assessment Model

We now explain how the previous models, namely Attacker, Defender and Attack-Defense Tree are used together to build a complete view for analysis, called *Risk Assessment Model (RAM)*.

Definition 8.1.5 (Risk Assessment Model)**.** A risk assessment model $\mathcal{M}$ is a composition of:

- $\mathcal{A}|_{\mathcal{S}}$ is an attacker following a strategy $\mathcal{S}$,
- $\mathcal{B}$ is the system under study characterized by its environment $env : \Sigma_A \longrightarrow [0, 1]$,
- $\mathcal{D}$ is a defender,
- $\mathcal{T}$ is an attack-defense tree,
- $c_{max}, t_{max} \in \mathbb{R}$ are the maximal attacker cost and time resources.

A RAM simulates attacks represented by an attacker $\mathcal{A}|_{\mathcal{S}}$ – under the constraints c_{max} and t_{max} – on the system $\mathcal{B}$ against a fixed defense configuration (modeled by $\mathcal{D}$). The system

under study is a BIP model that describes the nominal behavior of the system together with its vulnerabilities weighted with their success probabilities (the environment *env*). The system $\mathcal{B}$ can be abstracted by *env* when the nominal behavior is not designed yet. Recall that BIP models, and consequently the system $\mathcal{B}$, are probabilistic while an attacker model $\mathcal{A}$ is non-deterministic. The composition of $\mathcal{A}$ and $\mathcal{B}$ has an underlying MDP semantics where $\mathcal{A}$ non-deterministically generates attack actions (representing the input symbols of the MDP) that probabilistically drag $\mathcal{B}$ to specific locations depending on the action success/failure verdicts (representing the output symbols of the MDP) encoded as probabilities in the environment *env*.

The status of an attack is given by the current status of the attack-defense tree $\mathcal{T}$. The evaluation of the status of an attack using the attack-defense tree $\mathcal{T}$ is twofold:

1. the defense configuration $\mathcal{D}$ is used to evaluate the defense part of the tree, (i.e., *ap* of $\mathcal{T}$ such that $ap \in \Sigma_D$). This phase is done statically since the defense is fixed in our case. For each $ap \in \mathcal{T}$, where *ap* is a defense action, *ap* is evaluated to $true$ (respectively $false$) whenever $enabled(ap) = 1$ (respectively $enabled(ap) = 0$).

2. second, the attacker $\mathcal{A}|_S$ is used dynamically to sequentially generate attack actions a_i that may succeed or fail according to the states of $\mathcal{B}$ and the environment vector *env*. Whenever an attack a_i succeeds, the corresponding atomic proposition in $\mathcal{T}$ is evaluated to $true$. Attack actions in $\mathcal{T}$ are either evaluated to $true$ or not yet.

An execution trace ω of the risk assessment model $\mathcal{M}$ (denoted *attack trace*) is a sequence of timed attack actions (a_i, τ_i), where $\tau_i \in [l_{a_i}, u_{a_i}]$ is the duration of action a_i. We call $\Omega_{\mathcal{M}}$ the set of all attack traces generated by $\mathcal{M}$. It is worth mentioning that actions relative to the nominal behavior of the system are considered unobservable and hence do not appear in ω. Remark that the attacker model is constrained by c_{max} and t_{max} which define a budget of available resources and time to perform a sequence of attack actions. Hence, an attack trace is finite and ends in one of the following scenarios. Let us first introduce the *attack cost* and the *attack duration* as follows. Given a trace $\omega \in \Omega_{\mathcal{M}}$ of length n, the attack cost is $cost(\omega) = \Sigma_{i=1}^{n} c_{a_i}$, where c_{a_i} is the cost associated with action a_i. Similarly, the attack duration is $duration(\omega) = \Sigma_{i=1}^{n} \tau_i$. Thus, an attack trace ends when:

- the attack-defense tree $\mathcal{T}$ is evaluated to $true$ or $false$,
- the attacker has exhausted his resources or time budget, i.e., when $cost(\omega) > c_{max}$ or $duration(\omega) > t_{max}$,
- the attacker cannot select more attack actions based on the strategy $\mathcal{S}$.

It is worth mentioning that the attack-defense tree $\mathcal{T}$ is evaluated to $false$ only when the defense configuration $\mathcal{D}$ prevents all the tree branches from simplifying to $true$. In contrast, the tree evaluates to $true$ when the attacker's goal is fulfilled. The third situation happens when

the attacker cannot choose an action according to the strategy $\mathcal{S}$ that could have simplified the attack-defense tree.

Given a trace ω, we interpret it as a successful attack whenever the attack-defense tree is simplified to *true* in addition to having $cost(\omega)$ and $duration(\omega)$ below the c_{max} and t_{max} respectively, and as a failed attack otherwise.

8.2 Proposed Workflow

The purpose of this approach is to synthesize the most effective defense configuration against optimized attackers. The proposed workflow relies on the iterative process of IO-Def presented in Section 8.3, together with a combination of IOALERGIA and IEGA to identify the best attackers, respectively presented in Sections 8.4 and 8.5.

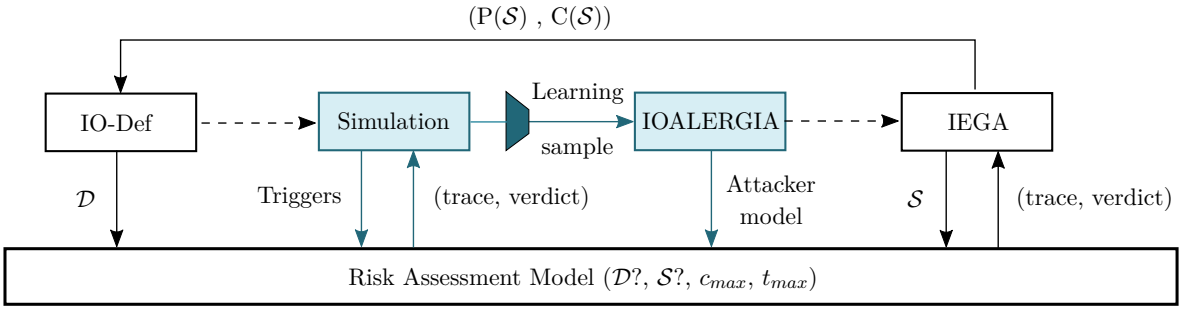


Figure 8.1: Proposed workflow for risk assessment based on model learning and strategies exploration.

Figure 8.1 illustrates the proposed workflow for risk assessment based on model learning and strategies exploration. Given a RAM as input, IO-Def evaluates defense configurations against optimized attackers obtained by the strategy synthesis process in order to determine the most impactful defense configuration. This strategy synthesis process is composed of a simulation module responsible for the construction of learning samples, IOALERGIA module to learn attacker models, and IEGA for the exploration of attack strategies.

For each considered defense, a uniform naive attacker is used to generate attack traces. Successful traces are projected on the set of attack actions and kept to constitute the learning sample. Then, IOALERGIA is applied on this latter to infer an attacker model that encodes the vulnerabilities dependencies, together with additional knowlegde on the success probabilities of individual attack actions (*env*). This augmented attacker model, represented as an MDP, captures the successful attack scenarios observed in the sample generation phase (handled by the simulation module). The role of IEGA is then to identify a strategy for the attacker to optimize his attack cost and success probability.

In this workflow, the role of the simulation and IOALERGIA modules (represented in green in Figure 8.1) is to construct the attacker model. This model represents viable combinations

of vulnerabilities that are later on explored by IEGA to identify the best attack strategies. However, this viability strongly depends on the amount of information available to the system analyst. When performing a security risk assessment at early stages of the system design, only few information is available which makes the viability of attack strategies challenging to state. In that case, an exhaustive combination of all the vulnerabilities is considered. Consequently, the strategy synthesis process varies depending on the abstraction level of the system $\mathcal{B}$ in the input RAM. When dealing with detailed system models (including the nominal behavior), this process starts by inferring the attacker model before exploring it with IEGA. At contrary, for RAM with abstracted systems, the attacker model is defined *a priori* to allow all possible combinations of attack vulnerabilities, ignoring any potential dependencies. In this latter case, the simulation and IOALERGIA modules are bypassed and the combinatorial attacker model is the subject of the IEGA exploration.

In the complex system setting, we intend to study a type of attackers that can extract viable attack strategies from random attack attempts on the system. These attackers exhibit a structured behavior that encapsulate attack action dependencies. We use IOALERGIA as a means to learn such structured behaviors given a set of generated attack traces. In addition, IOALERGIA allows the attacker to learn the probability for each attack action to succeed, i.e., the environment *env*. These two information are encoded in the resulting MDP and are referred to as learned (augmented) attacker model.

IEGA aims at exploring a large range of attack strategies, seeking for a near-optimal solution to this reachability problem, namely, strategies that minimize the attack cost and maximize its probability of success. Recall that the type of explored strategy depends on the level of abstraction of $\mathcal{B}$, that is, IEGA returns a deterministic strategy when the system's nominal behavior is explicitly modeled, and a probabilistic one if this behavior is abstracted.

8.3 Identifying Impactful Defenses

In this section, we explore defense configurations that make the system harder to attack, in the sense that the best attacker – obtained with IOALERGIA and IEGA – needs more resources to achieve his attack. More precisely, we aim at identifying the defense actions that have the largest impact on the attack cost.

We propose a heuristic, denoted Impact-Optimal Defense (IO-Def), that evaluates the impact of the defenses on the attack cost. A naive approach to security would be to enable all available defense actions. However, some of them may not significantly increase the attack cost. A more pragmatic approach is to look for a good balance between defenses and their impact on the attack cost. This is particularly important if the organization's defense budget is limited.

The heuristic implicitly builds an exploration tree where the root is the defense configuration with all the actions enabled, i.e., $D_1^1 = \Sigma_D$. The defense D_j^i at the i^{th} level of the tree is obtained by disabling the defense action j that was enabled in its parent node. For example, the third

child of D_1^1 is $D_3^2 = D_1^1 \setminus \{d_3\}$. Each defense configuration D_j^i is characterized by the cost C_j^i and the success probability P_j^i of the attack strategy obtained with IEGA. The tree is explored in a breadth-first order. For each level $i > 1$, we identify the defense configuration with the minimal impact on the attack cost, and select it for further exploration in the case its impact is lower than a given threshold ϵ.

The impact g_j^i is a measure that scores a defense D_j^i by computing the relative decrease in the attack cost due to the deactivation of the j^{th} defense. It is defined as $(C_*^{i-1} - C_j^i) \ / \ C_*^{i-1}$, where C_*^{i-1} is the attack cost of the selected parent node. The exploration ends whenever all the impacts of level $i+1$ are greater than or equal ϵ, or no more defenses are available, i.e., $D_1^{i+1} = \emptyset$. Finally, the most impactful defense configuration $\mathcal{D}$ is the one in which no defense can be disabled. Algorithm 7 presents the IO-Def heuristic, that identifies the subset $\mathcal{D}$ of defense actions Σ_D such that the individual impact of each enabled defense is above ϵ.

Data: a set of defense actions Σ_D, a threshold ϵ
Result: the optimal subset $\mathcal{D}$ of enabled defenses
$\mathcal{D} = \Sigma_D$;
Boolean *improved* = true;
Integer i = 1;
while *improved* **do**
  i++;
  improved = false;
  Compute the minimal attack cost C_*^{i-1} against $\mathcal{D}$ using $IOALERGIA + IEGA$;
  foreach $d_j \in \mathcal{D}$ **do**
    Compute the minimal attack cost C_j^i against $\mathcal{D} \setminus \{d_j\}$ using $IOALERGIA + IEGA$;
    Compute the impact $g_j^i = \frac{C_*^{i-1} - C_j^i}{C_*^{i-1}}$;
  end
  Find the defense $d_{min} \in \mathcal{D}$ having the lowest impact g_{min}^i;
  if $g_{min}^i < \epsilon$ **then**
    $\mathcal{D} = \mathcal{D} \setminus \{d_{min}\}$;
    improved = true;
  else
    return $\mathcal{D}$;
  end
end

Algorithm 7: Impact-Optimal Defense heuristic for defense exploration

Figure 8.2 illustrates the exploration of the best defense configuration given three defense actions $\Sigma_D = \{a, b, c\}$, using IO-Def. In this example, the three defense actions are initially enabled, represented in the root node ($i = 1$). Then we disable one defense action at a time, resulting in three new defense configurations $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$, that constitute level $i = 2$. Their impacts are then computed and compared to identify the smallest value, in this case g_2^2. Since $g_2^2 < \epsilon$, $\{a, c\}$ is selected as the new best defense and the exploration is resumed from it. Again, we disable defenses one by one to generate defense configurations of level $i = 3$, and g_1^3 is identified as the smallest impact value. However, in this case, $g_1^3 \geq \epsilon$, which leads to the end of the exploration. Therefore, $\mathcal{D} = \{a, c\}$ is considered to be the most impactful defense configuration.

In the worst case, Algorithm 7 executes the while loop $n + 1$ times where $n = |\Sigma_D|$. Each iteration computes $m + 1$ attack strategies using IEGA, where $m = |\mathcal{D}|$ (initially $m = n$),

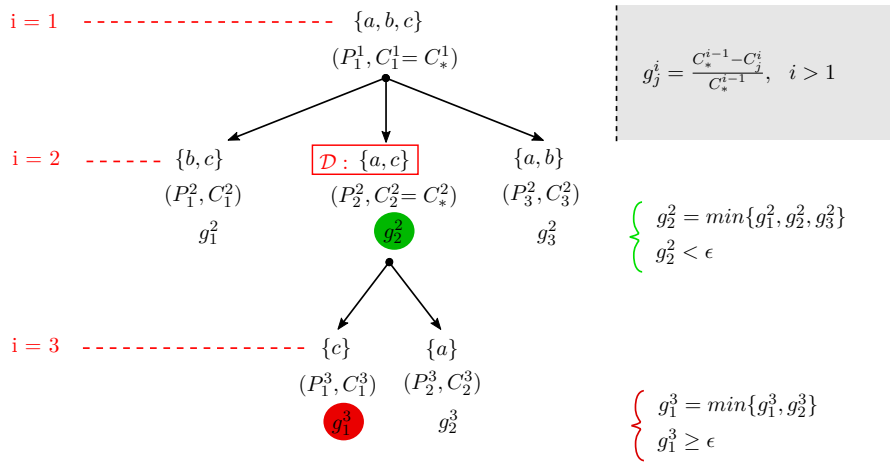


Figure 8.2: Illustration of the IO-Def heuristic

and evaluates the impact of defenses. Remark that each iteration decreases m by one (the deactivated defense action). Hence, IEGA is executed, at worst, $\frac{(n+1)(n+2)}{2}$ times. This happens when all the available defenses fail to prevent the identified strategy. Therefore, they are all disabled.

8.4 Learning Attacker Models

In this section, we recall the IOALERGIA [111] algorithm for learning MDPs. IOALERGIA is a state-merging algorithms that proposes to infer MDPs from a Learning Sample ($\mathcal{LS}$). In this algorithm, this learning sample is first represented in a compact manner as a Input and Output Frequency Prefix Tree Acceptor (IOFPTA). This IOFPTA is a trivial representation of the language since it accepts all the words in $\mathcal{LS}$. Nevertheless, this is not a suitable representation for several reasons. Among those reasons, one can point out the huge size of the model that is proportional to the number of symbols in $\mathcal{LS}$, and the lack of generality since the IOFPTA accepts only words in $\mathcal{LS}$.

To cope with these drawbacks, the IOFPTA building phase is followed by a reduction process. During this computation, compatible states are detected in that IOFPTA and merged in order to obtain a more compact representation of the target language. This operation introduces generalization which leads the model after merge to accept more words.

The exploration is done following a lexicographical order and state compatibility is computed over transition frequencies. The algorithm manipulates three types of states:

1. Red states : Identified to be part of the final MDP.
2. Blue states : States to evaluate at the current iteration. They are the non-red successors of red states.

3. Uncolored states : States to evaluate later on in the process. They belong to the subtree whose root is a blue state.

Initially, the IOFPTA root is marked as red and all its successors are marked as blue, the rest remaining uncolored. At each iteration, the algorithm tries to merge all the pair composed of one red state and one blue state. The compatibility criterion is calculated to evaluate each merge. If a pair of red-blue states is *compatible*, the merge operation is performed. Otherwise, if no possible merge is found, one blue state is marked as red and its successors are marked as blue.

8.4.1 IOFPTA

The IOFPTA is the trivial representation of the learning sample $\mathcal{LS}$. Words in $\mathcal{LS}$ are sequences of symbols $\omega = (Act \bullet \Sigma \cup \{err\})^*$. *Act* are input actions performed by the system environment and are selected non-deterministically, whereas Σ is a set of available actions that the system can output as a probabilistic response to the input actions. Note that the system can trigger an *err* if no suitable response to a specific input action exists at a given system state.

From a model perspective, the nodes L in the IOFPTA are labeled by output actions, i.e., $\mathcal{L}(l) \in \Sigma, l \in \mathrm{L}$. And each edge $e \in \mathrm{L} \times Act \times \mathbb{N} \times \mathrm{L}$ is labeled by input actions defined over *Act*. Hence, each branch of the tree represents one (or more) word in $\mathcal{LS}$ and common prefixes are merged into a single path. Given the definition of *err*, error edges are added as self-loops on the nodes where the error has been detected. Finally, each edge $e = \langle l, a, n, l' \rangle$ is annotated by its frequency $f(e) = n$, namely, the number of words in $\mathcal{LS}$ that share that edge.

An IOFPTA is the first hypothesis of MDP, albeit a simplified one where no cycles are allowed, except error self-loops. Both models share the same structure in terms of states, transitions, alphabets and labeling functions, but they differ in the transition probabilities (respectively edge frequencies) in the MDP (respectively the IOFPTA). The MDP relative to an IOFPTA can be extracted by simply converting edge frequencies to transition probabilities through a normalization process.

8.4.2 Elementary Operations of IOALERGIA

The main goal of merging states is to generalize the learned languages and build a more compact representation of them. At each iteration, the merges are evaluated in a lexicographical order and compatible ones are performed. A blue state is promoted if no compatible states are identified.

The promote operation is a coloring operation that does not affect the structure of the MDP. When no merge is feasible, a blue state is colored in red and its uncolored successors are colored in blue. Since no other operation is possible, this means that the promoted blue state behaves significantly differently from the red states identified so far. This newly identified red state will be considered, in next iterations, when searching for new merges to perform. Hence,

the promote operation is the one responsible for identifying states that belong to the final MDP, colored in red.

8.4.3 Compatibility Criterion

Compatibility criteria are used to evaluate the relevance of a merge operation. They are statistical tests that estimate how equivalent the sub-languages generated by two states are. IOALERGIA implements a compatibility criterion based on the Hoeffding bound. The test – parametrized by $\epsilon \in (0, 1]$ – is composed of three different steps and returns true if all the steps are evaluated to true. Two states, namely, a red state l_r and a blue state l_b, are compatible if:

1. Both states have the same label, i.e., $\mathcal{L}(l_r) = \mathcal{L}(l_b)$
2. For each input symbol $a \in Act$ and output symbol $\sigma \in \Sigma$, the Hoeffding function $H(f_r, n_r, f_b, n_b, \epsilon)$, returns true. The parameter $f_r = f(\langle l_r, a, n', l' \rangle)$ (respectively $f_b = \langle l_b, a, n'', l'' \rangle$) represents the frequency of the outgoing transition from l_r (respectively l_b) labeled by a and leading to a state l' (respectively l'') labeled by σ, i.e., $\mathcal{L}(l') = \mathcal{L}(l'') = \sigma$. The parameter n_r (respectively n_b) is the sum of the frequencies of the outgoing transitions from l_r (respectively l_b) labeled by a, for all the output symbols. It can be written as $n_r = \sum_{\sigma_1 \in \Sigma} f(\langle l_r, a, n_{\sigma_1}, l_{\sigma_1} \rangle),\ with\ \mathcal{L}(l_{\sigma_1}) = \sigma_1$, and similarly for n_b.
3. For each input symbol $a \in Act$ and output symbol $\sigma \in \Sigma$, $succ(l_r, a, \sigma)$ and $succ(l_b, a, \sigma)$ are compatible, where $succ(l, a, \sigma) \in \mathrm{L}$ is a successor function such that $l' = succ(l, a, \sigma)$ iff: (1) $\mathcal{L}(l') = \sigma$, and (2) there exists an edge $e = \langle l, a, n', l' \rangle$.

The Hoeffding function H returns true if the probability to exit l_r and l_b with symbol a and reach a state labeled by σ is sufficiently close. It is computed as follows:

- If n_r or n_b equals zero then return true
- Else, return $\left| \frac{f_r}{n_r} - \frac{f_b}{n_b} \right| < (\sqrt{\frac{1}{n_r}} + \sqrt{\frac{1}{n_b}}) \times \sqrt{\frac{1}{2} ln \frac{2}{\epsilon}}$

8.5 Synthesizing Attack Strategies

In this section, we present our approach to explore attack strategies. As explained earlier, our goal is to identify the most cost-effective strategy under which an attack is most likely to succeed. Our proposal is based on a hybrid variant of GA and Local Search (LS), called Intensified Elitist Genetic Algorithm (IEGA), for the identification of a near-optimal attack strategy.

A Genetic Algorithm (GA) is an evolutionary algorithm inspired from natural selection and genetics. It provides an efficient way to explore large solution spaces to select high-quality solutions for optimization and search problems. An important requirement to achieve an

exploration is to be able to quantify solutions in order to establish an order over them. In this work, we rely on SMC to fulfill this goal as explained hereafter.

8.5.1 Overview

We consider as input a risk assessment model $\mathcal{M}$ composed of an attacker model $\mathcal{A}$, with a system model $\mathcal{B}$ including the environment env, a defender model $\mathcal{D}$, an attack-defense tree $\mathcal{T}$ and the constraints t_{max} and c_{max}.

In our approach (IEGA), an individual denoted $\mathcal{I} = \langle \mathcal{S}, c, p \rangle$ is an attack strategy $\mathcal{S}$ annotated with an expected cost c and a probability p of success of an attack when applying it. The cost c and the probability p of success for an individual are computed using SMC. More precisely, the probability estimation algorithm (PE) [79] is used to check the risk assessment model against the property $\phi = \Diamond_{t<t_{max}}^{c<c_{max}} \mathcal{T}$. Recall that the precision of PE and its confidence are respectively controlled by the parameters α and δ.

IEGA starts by randomly generating N initial strategies (individuals) to constitute the initial population P_0, evolving over M generations, as depicted in Figure 8.3. For each generation, $N/2$ new children strategies are generated as follows:

1. **Selection for breeding**: we randomly choose two parent individuals in the current population as candidates for the cross-over operation,
2. **Cross-over operation**: a child individual is built by performing a single-point cross-over,
3. **Intensification with LS**: the resulting individual is intensified using LS, i.e., a heuristic aiming at improving it by exploring its neighbor solutions,
4. **Mutation**: an individual has a p_m probability to be mutated, i.e., altering a randomly selected information i of the strategy $\mathcal{S}$, namely, changing the value of $\mathcal{S}[i]$.

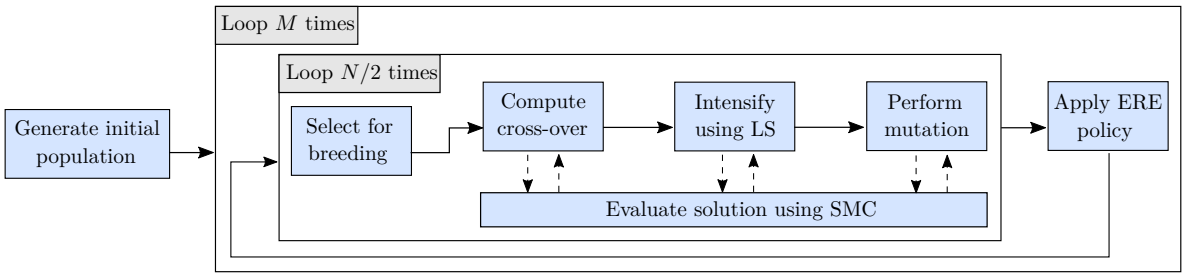


Figure 8.3: Workflow of IEGA with a population of N individuals over M generations

The Local Search (LS) operation aims at improving a solution by repeatedly moving to better solutions residing in its neighborhood, until no improvement is possible. A neighbor solution I_i is said to improve the current one I if it has a better fitness value. The latter is computed using the fitness function $Score$ which is a weighted sum of the solution cost c and its probability of

success p. Formally, the fitness function is defined as $Score(c, p) = a \times p - (1 - a) \times c$, where $a \in [0, 1]$ represents a linearization factor, used for weighting and scaling the two parameters.

The last phase of the outer loop (ERE) in Figure 8.3 identifies among parent individuals in population P_i and their $N/2$ children, the ones to keep in the next generation $i + 1$. We use Extreme Ranking Elitism (ERE) [115] as a replacement policy, which aims at selecting the best individuals while keeping some diversity in the population. Concretely, in addition to the best solutions, bad ones are kept to prevent early convergence.

We use IEGA to explore both probabilistic and deterministic strategies. Obviously, the strategy encoding and its manipulation is strongly dependent on the strategy type. In the next section, we further detail the cross-over, the neighborhood construction, the mutation and ERE operations, for the two types of strategies. Note that selection for breeding is performed by random sampling and will therefore not be further detailed.

8.5.2 IEGA Operations Description

8.5.2.1 Cross-over Operation

It consists of building a child $\mathcal{I} = \langle \mathcal{S}, c, p \rangle$ by combining two randomly selected parents $\mathcal{I}_1 = \langle \mathcal{S}_1, c_1, p_1 \rangle$ and $\mathcal{I}_2 = \langle \mathcal{S}_2, c_2, p_2 \rangle$. $\mathcal{I}$ is obtained by performing a single-point cross-over, i.e., inherits the first half of its genes from $\mathcal{I}_1$ and the second half from $\mathcal{I}_2$. Formally, the description of this operation varies in function of the strategy type since they have different sizes. A deterministic strategy identifies an action to perform at each state of the attacker model $\mathcal{A}$ and therefore, is of size $|\mathcal{A}|$. A cross-over between two deterministic strategies $\mathcal{S}_1$ and $\mathcal{S}_2$ is described as:

$$\mathcal{S}[i] = \begin{cases} \mathcal{S}_1[i], & i \leq |\mathcal{A}|/2 \\ \mathcal{S}_2[i], & \text{otherwise} \end{cases}$$

A stochastic strategy assigns a probability of occurrence to each attack action in Σ_A, which determines its size to $|\Sigma_A|$. Given two probabilistic strategies $\mathcal{S}_1$ and $\mathcal{S}_2$, a cross-over constructs a child solution described by its strategy $\mathcal{S}$ as follows:

$$\mathcal{S}[i] = \begin{cases} \mathcal{S}_1[i], & i \leq |\Sigma_A|/2 \\ \mathcal{S}_2[i], & \text{otherwise} \end{cases}$$

It is worth mentioning that, with probabilistic strategies, cross-over is followed by a normalization operation to ensure that the obtained probabilistic strategy $\mathcal{S}$ is a valid mass function, i.e., $\Sigma_i(\mathcal{S}[i]) = 1$.

8.5.2.2 Neighborhood Construction

The individuals resulting from the cross-over are intensified, i.e. improved, using a local search (LS) over a set of neighbor solutions. In the following, we define the notion of individual's neighborhood for deterministic and probabilistic strategies.

Deterministic strategy's neighbors. Individuals are said to be neighbors when their respective deterministic strategies differ by the decision at only one state of the attacker model. More formally, given an individual $\mathcal{I} = \langle \mathcal{S}, c, p \rangle$, the set of neighbor solutions $V(\mathcal{I}) = \bigcup_{i=1}^{|\mathcal{A}|} V(\mathcal{I}, s_i)$ to individual $\mathcal{I}$ is identified by changing the attack action at state s_i of the attacker by another enabled input action, as follows:

- $V(\mathcal{I}, s_i) = \{\langle \mathcal{S}', c', p' \rangle : \forall j \neq i,\ \mathcal{S}'(s_j) = \mathcal{S}(s_j) \wedge \mathcal{S}'(s_i) \in enabled\_input(s_i) \setminus \mathcal{S}(s_i)\}$

where $enabled\_input(s_i)$ is the set of all input actions that are enabled at state s_i. It is worth mentioning that an individual has at most $(|\Sigma_A| - 1) \times |\mathcal{A}|$ neighbors.

Probabilistic strategy's neighbors. Individuals are said to be neighbors when their respective probabilistic strategies are slightly different. More formally, given an individual $\mathcal{I} = \langle \mathcal{S}, c, p \rangle$, the set of neighbor solutions $V(\mathcal{I}) = \{\mathcal{I}_i = \langle \mathcal{S}_i, c_i, p_i \rangle\}$ to individual $\mathcal{I}$ is identified by disabling a single attack action a_i, as follows:

- if $\mathcal{S}[i] = 1$ or $\mathcal{S}[i] = 0$ then the i^{th} neighbor individual $\mathcal{I}_i$ does not exist. In the first case, it is because a_i is the only enabled action and disabling it makes $\mathcal{S}$ an invalid mass function. In the second case, a_i is already disabled.
- otherwise, individual $\mathcal{I}_i$ is identified by a strategy $\mathcal{S}_i$ such that:

$$\mathcal{S}_i[j] = \begin{cases} 0, & j = i \\ \frac{\mathcal{S}[j]}{\Sigma_k(\mathcal{S}[k]) - \mathcal{S}[i]}, & \text{otherwise} \end{cases} \tag{8.1}$$

The normalization in the second case of Equation 8.1 is again to ensure well-formedness of the synthesized strategy (probability mass function). Remark that an individual has at most $|\Sigma_A|$ neighbors. Figure 8.4 illustrates the computation of the neighbors of an individual with a probabilistic strategy $\mathcal{S} = [0.3, 0.5, 0.2]$, over 3 attack actions. For example, the first neighbor is obtained by disabling the first attack action in $\mathcal{S}_1$ and then normalizing it.

8.5.2.3 Mutation Operation

Mutation consists in randomly altering the i^{th} information of a strategy $\mathcal{S}$. This operation concerns intensified individuals obtained after the local search and has a p_m probability to be performed. In other words, individuals are not systematically mutated. Mutating a deterministic

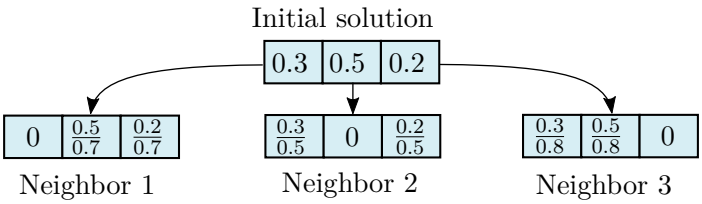


Figure 8.4: Illustration of a neighborhood construction for a probabilistic strategy

strategy $\mathcal{S}_d$ substitutes the chosen attack action a of a randomly selected state s_i by another enabled input action, where $a \in \mathit{enabled\_input}(s_i) \setminus \mathcal{S}_d[i]$. This operation takes a different meaning for a probabilistic strategy $\mathcal{S}_p$ since it consists in the modification of the selection probability of a randomly chosen attack action, and requires a normalization.

8.5.2.4 ERE Replacement Policy

A genetic algorithm maintains a population of size N over M generations. The replacement operation rules the survival of individuals through generations. Extreme Ranking Elitist replacement is a balanced solution to provide elitism while avoiding early convergence.

Given a population P_i of N parents and their $N/2$ children, an Extreme Ranking Elitist replacement policy identifies the N candidate individuals for the next generation's population P_{i+1}. This policy is parametrized by p_{ere}, that represents the proportion of the population to be selected by elitism. More precisely, the replacement is performed as follows:

1. We consider an intermediate population P'_i of size $\frac{3N}{2}$ composed of the N parents and their $N/2$ children. Individuals in this population are ranked based on the Pareto dominance principle, and sorted in an ascending order. In the Pareto dominance principle, a solution I_j is known as dominated by another solution I_k if the latter is better for every criterion, in our case, $c_j \geq c_k \wedge p_j \leq p_k$ excluding the case where they are all equal. Considering this definition, the ranking consists of assigning rank 1 to non-dominated solutions of the population. Iteratively, we temporarily remove the non-dominated ones and identify the new non-dominated solutions that we assign the next rank, until all the solutions are ranked. Figure 8.5 is an example of Pareto ranking on a population of 10 individuals.

2. To select the N individuals to be part of generation $(i+1)$, we compute the number of best (elite) individuals $N_b = N \times p_{ere}$, and the number of worst individuals $N_w = N \times (1 - p_{ere})$ kept for diversification. Population P_{i+1} is computed as:

$$P_{i+1} = \bigcup_{j=1}^{N_b} \{P'_i(j)\} \cup \bigcup_{k=\frac{3N}{2}-N_w+1}^{\frac{3N}{2}} \{P'_i(k)\}$$

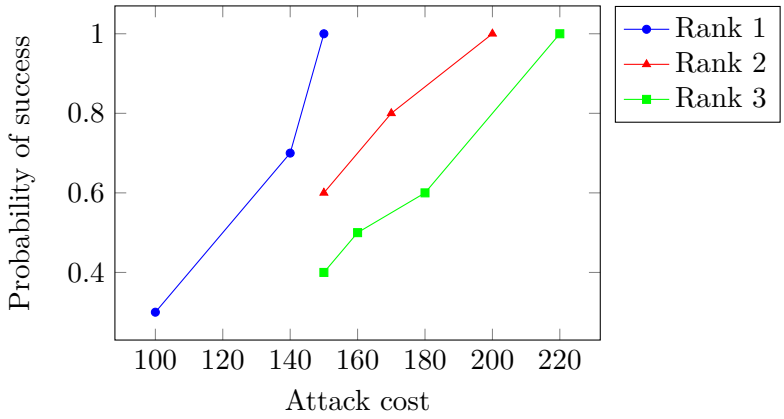


Figure 8.5: Illustration of Pareto dominance ranking on a population of 10 individuals

where $P'_i(j)$ is the j^{th} individual in population P'_i. Therefore, we select the N_b first (best) individuals and the N_w last (worst) solutions in P'_i.

8.6 Experiments

In this section, we present the experiments for the validation of the proposed security risk assessment approach. We validate this proposal in the two configurations, namely, on abstracted and on detailed concrete systems.

8.6.1 Experiments on Abstracted Systems

Here, we present the experiments performed using IEGA and IO-Def heuristics. We consider case studies addressing security issues at the level of organizations (ORGA, MI), gateway protocols (BGP), and sensor network infrastructures (SCADA). The comparison with the state-of-the-art technique using UPPAAL STRATEGO [69] shows that our approach performs better in most of the cases.

8.6.1.1 Overview and Experimental Setting

In the following, we give an overview of the considered case tudy and we explain our experimental setting.

Overview. In our experiments, we considered four case studies briefly discussed below[2]:

1. ORGA. In this study, eight cyber and social attack actions can be combined to infiltrate an organization. To prevent such actions, the organization considers different defense

[2]Further details are provided in Appendix B

actions, namely, train employees for thwart ($t1$) and for tricks ($t2$), threaten to fire them (tf) and authenticate tags (at).

2. Resetting a BGP session. In this case study, an attacker can execute six attack actions to reset a BGP session. The system is protected by three defense actions, i.e. check TCP sequence number by MD5 authentication (au), check trace-route by using randomized sequence numbers (rn), and secure routers with firewall alert (sr).

3. Supervisory Control And Data Acquisition system (SCADA). On these systems, the attacker tries to access some of the thirteen system components and provoke hardware failures in order to disturb the system. The system considers four defense mechanisms: switch the Human-Machine Interface (sw) or restart one of the three system agents ($rst1$, $rst2$, $rst3$).

4. A Malicious Insider attack (MI). In this case study, an insider tries to attack an organization system from inside by exploiting seventeen identified vulnerabilities. The system sets up protections by deploying an anti-virus (dva) and a mechanism to track the number of tries on passwords (tpt).

Experimental approach. For each of the case studies, we performed two kind of experiments. In the first, we manually tried all the possible combinations of available defense actions, synthesized sophisticated attack strategies for them and evaluated their induced costs and probabilities of success. We proceeded as follows: each time, we fixed a defense configuration and applied IEGA in order to synthesize a near-optimal attack strategy. Since IEGA relies on SMC, which is an estimation technique, to synthesize strategies, we performed 25 runs of IEGA each time and measured the expected values and standard deviations of the cost and the probability of success (reported in Table 8.1). Furthermore, for this first experiment, we compared the results obtained by IEGA with the ones of STRATEGO on the ORGA case study. As stated earlier, our technique synthesizes better attack strategies in terms of cost as reported in Table 8.2.

The second kind of experiments aims at identifying the most impactful defense configurations against a near-optimal attack strategy obtained in the first experiments. To do so, we rely on the IO-Def heuristic that automatically explores the defense configurations as explained in Section 8.3. The results of this set of experiments are reported in Figure 8.7.

For all the experiments, we considered the same budgetary constraints $c_{max} = 50000$ and $t_{max} = 300$ and we set the threshold $\epsilon = 0.05$ for the experiments with IO-Def. We also investigated the performance (exploration time) of the proposed heuristics (IEGA and IO-Def). We observed that IEGA shows a linear growth with respect to the size of Σ_A while IO-Def grows polynomially in the size of Σ_D.

8.6.1.2 Results and Discussion

Manual exploration of defenses. We first report in Table 8.1 the results of IEGA on the BGP, SCADA and MI case studies. In this table, the first column corresponds to the deployed defense configuration, the second and third columns report respectively the average cost $\bar{x}_{cost}$ over 25 runs of IEGA and standard deviation σ_{cost}, the last column shows the average execution time of IEGA. We omit reporting the average probability of success (resp. standard deviation) as it is always 1 (resp. 0) [3]. Note that for each study, we also investigated the setting where no defense action is deployed which allows us to see the impact of different defense actions on the attack cost when enabled.

Defense	$\bar{x}_{cost}$	σ_{cost}	Runtime (s)
BGP			
au rn sr	50000	0.00	2.65
au sr	50000	0.00	2.54
rn sr	50000	0.00	2.71
au rn	284.31	2.83	3.95
au	285.00	2.38	4.02
sr	428.95	3.60	4.99
rn	284.45	1.97	3.93
none	283.96	1.94	4.09
SCADA			
*sw rst*1 *rst*2 *rst*3	327.71	3.85	40.74
*sw rst*1 *rst*2	328.68	3.61	39.49
*sw rst*1 *rst*3	328.69	3.00	41.63
*sw rst*2 *rst*3	329.20	3.20	42.63
*rst*1 *rst*2 *rst*3	328.57	2.87	42.67
*sw rst*1	328.09	3.63	39.46
*sw rst*2	328.48	3.07	38.32
*sw rst*3	328.29	3.29	39.90
*rst*1 *rst*2	327.87	2.91	41.68
*rst*1 *rst*3	328.52	4.47	39.43
*rst*2 *rst*3	327.78	3.68	39.20
sw	329.03	4.16	38.64
*rst*1	327.96	3.43	39.29
*rst*2	326.60	4.38	40.26
*rst*3	326.95	3.32	42.30
none	330.21	3.11	41.35
MI			
dva tpt	328.83	3.53	49.62
dva	163.04	3.66	48.60
tpt	331.08	3.42	47.84
none	159.85	2.69	49.26

Table 8.1: IEGA results with various defense configurations on BGP, SCADA and MI.

For BGP, we observed that the first three defense configurations lead inevitably to exceed the maximum allowed cost c_{max}. That is, no attack strategy can be synthesized within this budget, whereas in the case of the remaining defense configurations, strategies requiring lower cost can be synthesized. Moreover, one can see that the cost growth is minor when using *rn* or *au* compared to the case when no defense is used. For SCADA, we notice that the computation of the near-optimal strategy results almost in the same cost for all defense configuration. This can be explained by the existence of a low cost strategy that can always be applied, regardless of the implemented defenses. Furthermore, we observed that the cost induced by using any combination of defense actions does not significantly improve compared to the

[3] Except for the first three cases in BGP where the probability of success is 0.

		IEGA			STRATEGO	Improvement
		$\bar{x}_{cost}$	σ_{cost}	Runtime (s)	$\bar{x}'_{cost}$	(%)
Defenses	$t1\ t2\ tf\ at$	968.08	5.30	9.6	1038.33	7
	$t2\ tf\ at$	237.97	1.39	10.2	410.52	42
	$t1\ t2\ at$	238.37	1.55	10.6	309.35	23
	$at\ t2$	237.92	1.27	10.1	359.48	34
	$t1\ tf\ t2$	967.05	7.90	9.8	1000.90	3
	$tf\ t2$	238.18	1.58	10.2	288.53	17
	$t1\ t2$	238.20	1.29	10.2	295.70	19
	$t2$	238.21	1.59	10.6	298.67	20
	$t1\ tf\ at$	96.19	1.14	9.4	112.17	14
	$tf\ at$	96.04	1.08	9.7	103.37	7
	$t1\ at$	96.35	0.98	9.5	133.60	28
	at	96.15	0.98	9.4	110.00	13
	$t1\ tf$	96.08	1.29	9.8	121.07	21
	tf	96.27	1.14	9.8	105.97	9
	$t1$	95.99	0.67	9.4	109.33	12
	none	96.48	0.91	10.2	110.57	13

Table 8.2: IEGA results with various defense configurations on ORGA benchmark.

defenseless case. For MI, we obtained different costs depending on the defenses used. We noticed that defense action *dva* insignificantly increases the attack cost as opposed to *tpt*. The results for the ORGA case study are reported in Table 8.2 for the sake of comparison with STRATEGO. Except the last two columns, the table presents the same information as Table 8.1. For this study, we observed that varying the enabled defenses significantly affects the minimal attack cost and that the defense action *at* does not have a great impact on the cost. We actually observed that the attack strategies blocked by this defense action can be also blocked by $t2$.

Detailed results regarding the runtime performance of IEGA are reported in the Table 8.1 and summarized in Figure 8.6. The latter shows a linear evolution of the runtime when increasing the size of Σ_A, i.e., the number of available attack actions. The measures in Figure 8.6 correspond respectively to the average runtime on BGP (6 actions, 3.6s), ORGA (8 actions, 9.9s), SCADA (13 actions, 40.8s) and MI (17 actions, 48.8s). We also observed that IEGA shows a certain stability of the synthesized attack strategy over different runs as testified by the small standard deviation observed in the different experiments.

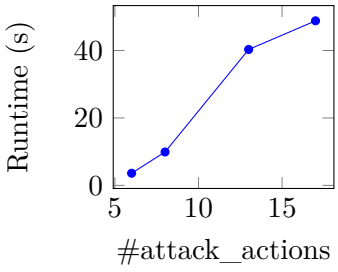


Figure 8.6: IEGA runtime variation

Finally, we compared the results obtained by IEGA with STRATEGO [69] on the ORGA case study. Comparison results are shown in the last two columns of Table 8.2 which respectively

Defense Actions	$t1$	$t2$	tf	at
Status	On	On	On	Off
Impact on cost	+75%	+90%	+75%	-
Exploration time	1min 25s			

(a) Results on ORGA

Defense Actions	au	rn	sr
Status	Off	On	On
Impact on cost	-	+99%	+99%
Exploration time	23s		

(b) Results on BGP

Defense Actions	sw	$rst1$	$rst2$	$rst3$
Status	Off	Off	Off	Off
Impact on cost	-	-	-	-
Exploration time	9min 11s			

(c) Results on SCADA

Defense Actions	dva	tpt
Status	Off	On
Impact on cost	-	+50%
Exploration time	4min 7s	

(d) Results on MI

Figure 8.7: Results obtained with IO-Def on different case studies

present the average cost obtained using STRATEGO and the percentage of improvement provided by our approach. This improvement is measured as $\frac{\bar{x}'_{cost} - \bar{x}_{cost}}{\bar{x}'_{cost}}$ where $\bar{x}'_{cost}$ (respectively $\bar{x}_{cost}$) is the minimal cost returned by STRATEGO (respectively IEGA). The obtained results show that our method is able to find attack strategies with lower attack costs than STRATEGO within the specified cost budget. In this case study, the improvement induced by our approach –in term of cost reduction– compared to STRATEGO ranged from 3% to 42% depending on the deployed defense configuration.

Automatic exploration of defenses. We report in Figure 8.7 exploration results using IO-Def for the different case studies. For each of them, we present the identified most impactful defense configuration $\mathcal{D}$ in a separate table showing respectively, the defense actions, their status (on/off), their impact on the attack cost (in percentage) in the context of $\mathcal{D}$ and the IO-Def exploration time.

We recall that identifying a defense action to be impactful or not, is done by comparing its impact to the threshold $\epsilon = 0.05$. We observed that the best defense configuration for ORGA (Table 8.7a) is $\mathcal{D} = \{t1, t2, tf\}$. In this setting, the role played by at was found to be negligible, while the highest impact (+90%) is brought by $t2$. The exploration results for BGP (Table 8.7b) show that the deployment of both rn and sr defenses is mandatory. Both of them have an impact of +99%, i.e., disabling any of them leads to a heavy decrease of the attack cost. In contrast, in the case of SCADA (Table 8.7c), none of the defenses has a significant impact on the attack cost. Basically, this means that the available defenses are useless against the synthesized cost-effective attack strategy. Table 8.7d shows the best defense obtained in the MI case study. In this defense configuration, only tpt plays a significant role in increasing the attack cost, with a +50% impact.

Regarding the exploration time of IO-Def, the main observation is that it does not only depend on the size of Σ_D but also on the nature of the system to explore and the IEGA runtime

(i.e., the size of Σ_A). In spite of the fact that ORGA and SCADA have the same number of defense actions, they are explored in significantly different amounts of time (respectively 1min 25s and 9min 11s). This is due to the inefficient available defense actions in the case of SCADA, leading to the worst case exploration time of IO-Def where all the defenses have to be disabled. Moreover, even though MI has the smallest number of defense actions to explore, it is not the fastest. This is explained by the time required for a single run of the IEGA algorithm (48.8s in average) in comparison to the cases of ORGA and BGP (respectively 3.6s and 9.9s in average).

8.6.2 Experiments on Detailed Systems

We ran our approach on a case study inspired from [149]. In this case study, an intruder wants to access an organization's network to corrupt its database. The target network is composed of an IIS web server, a Windows machine and a Linux server running three services: "I Seek You" chat software, a Squid web proxy, and a database (see Figure 8.8). This network is protected by two firewalls ruling the distant access to the internal network, together with the internal traffic itself. In addition, the organization deploys an intrusion detection system (IDS) to monitor the traffic between the internal network and the public network.

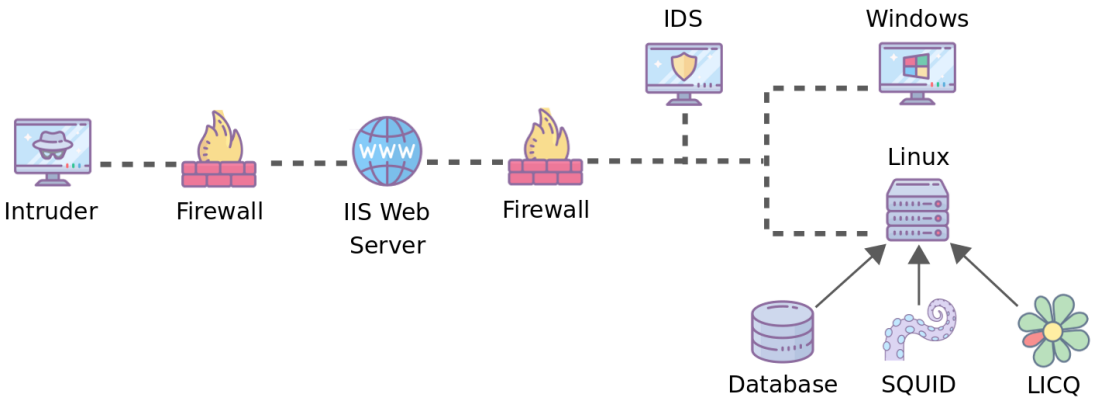


Figure 8.8: Topology of the organization's network

The intruder can perform 5 different attack actions in order to obtain a root privilege on the Linux machine, which allows him to fulfill his malicious goal. The attack actions can be performed by the intruder from any machine of the network as long as he has the appropriate privilege on it, and they are of the following kind:

- IIS buffer overflow attack to remotely gain root privilege on the IIS web server,
- Squid port scan on the Linux machine exploiting a misconfiguration of the SQUID web proxy to perform a port scan of the machines in the neighborhood,
- User privilege gaining on the Linux machine using the LICQ remote-to-user action, only visible after scanning ports,

- HTML scripting exploit on the Windows machine to obtain user privileges on the Windows machine,
- Local buffer overflow on the Linux machine to acquire root privileges on the server.

However, these actions are restricted by the firewalls and are monitored by the IDS. Hence, the difficulty is to complete an attack without being noticed by the IDS. In this case study, we aim at identifying the optimal attack strategy for the intruder to perform a successful attack with a minimal cost and a high probability of success.

We implemented a BIP model for the network security case study. We simulated this model several times to collect execution traces for the learning phase. To do so we used a Java implementation that triggers and collects several runs (in parameter) of the system then projects the traces on the set of attack actions. Note that we consider only execution traces where the malicious goal has been fulfilled, namely, successful attack traces. The result of this process is a set of traces encoding possible successful combinations of attack actions.

In this case study, we first construct a baseline attacker model by reducing the complexity of the learning problem to the case where all the attack actions succeed with probability 1. Secondly, we use IOALERGIA to infer the attacker model under the environment uncertainty. This model is assessed in comparison with the baseline attacker in order to evaluate the ability to identify complex attack action combinations in a probabilistic context, and to find an adequate output alphabet to encode the attack status. Then, given the learned attacker model we compute the optimal deterministic attack strategies with IEGA on manually explorated the defense configurations. The quality of these strategies is evaluated with respect to solutions obtained using Prism. Finally, we synthesize the most impactful defense configuration, that is, identify the impact of the deployed IDS mechanism and its ability to effectively protect the organization network.

8.6.2.1 Learning an Attacker Model Without Uncertainty

In this experiment, the success probability of an attack action is set to 1. The goal of this experiment is to obtain a scenario graph representing all the possible attack combinations. Attack actions are selected by the attacker in a uniform manner. Hence, we know that the probabilities to generate attack scenarios are to the same order, that is, we do not encounter the problem of rare scenarios. The obtained scenario graph can be considered as a comparison basis since it includes all the ways an intruder can use to attack the organization, for that given topology/configuration.

Figure 8.9 represents the scenario graph obtained with ALERGIA. In this graph, we can see different paths to perform successful attacks, identified by the execution of a local buffer overflow on the Linux machine (*local_lin*) that grants the intruder the root privileges. However, most of the paths contain a transition labeled with *licq_iis_lin_ids* indicating that the attack has been detected. The only undetected attack is the once where the intruder first gains access

to the Windows machine and performs the rest of his attack from it, since the internal traffic is not monitored by the IDS. It is worth mentioning that ALERGIA failed to merge states $s_{12,13,15}$ that are all final states with similar incoming transition (labeled *local_lin*).

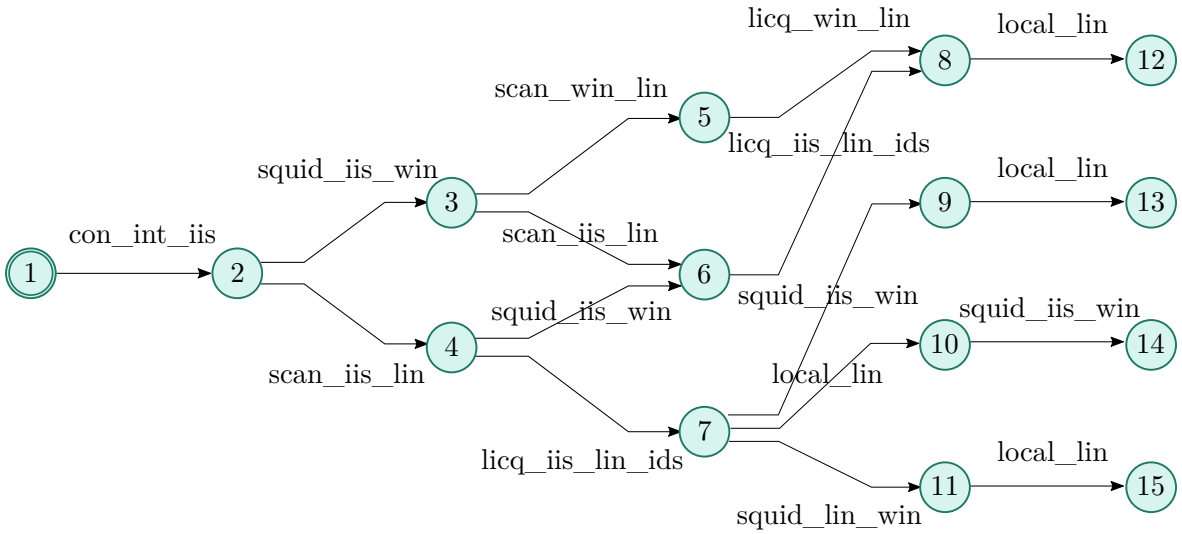


Figure 8.9: Scenario graph of the network intrusion obtained with ALERGIA

To be able to apply IOALERGIA, we define the output alphabet to represent the current status of an attack, encoded with a single digit number. An attack is in one of the three following statuses: running (0), succeeded (1), detected (2). Using this encoding, we update our sample generator to produce a learning sample for IOALERGIA. Note that the input alphabet represents the set of available attack actions. Figure 8.10 shows the scenario graph of the network intrusion obtained with IOALERGIA. One can see a more compact model with a small number of states. However, the resulting MDP introduces a lot of generalization. For example, according to the learned model, the attacker could scan the Linux machine from the IIS web server without first connecting to that server, which is obviously incorrect. This generalization is mostly due to the fact that the single-digit status encoding does not carry enough information about the status of single attacks. Hence, no distinction is made between the cases where no attack action has been performed and when several actions already succeeded.

8.6.2.2 Learning an Attacker Model With Probabilities

As a result of the previous experiment, one can see the inefficiency of the firstly proposed encoding of the output actions. To cope with this, we propose a more explicit encoding that reflects the status of the intruder's attacks on the network's machines and services labeled by "0" for "not succeeded yet" and "1" for "succeeded". The format of the output symbol is composed of 5 boolean values where:

- the first represents the status of the IIS buffer overflow attack,

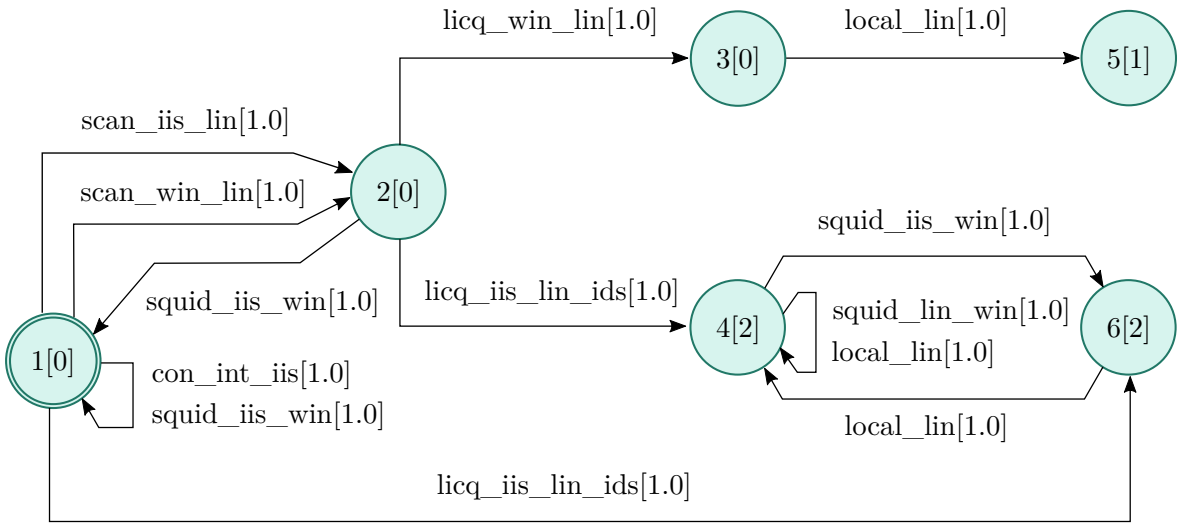


Figure 8.10: Scenario graph of the network intrusion obtained with IOALERGIA

- the second is for Squid port scan on the Linux machine,
- the third concerns the LICQ user privilege gaining on the Linux machine,
- the fourth encodes the HTML scripting exploit on the Windows machine,
- and the last is for the local buffer overflow on the Linux machine.

In addition to the new output encoding, we added some uncertainty on the success of each attack action (environment *env*). By setting the probability of the success of each attack to 0.5, we increase the length of traces (due to unsuccessful attempts) leading to successful attacks, which makes the scenario graph harder to identify.

Figure 8.11 shows the intruder's model, obtained with IOALERGIA, in the case of uncertainty and with the explicit encoding. In this model, states where the acquisition of root privileges on the Linux machine succeeded are identified by the success of the local buffer overflow on the Linux machine, i.e., the last digit of the encoding takes the value 1. In the learned attacker model, one can distinguish a clear pattern in the transitions. That is, given a state, an input symbol probabilistically leads back to the same state through a self-loop or takes to a new state, and the probability for each transition is approximately 50%. The model handles perfectly the uncertainty such that, the self-loop models the unsuccessful attempts while the second transition of the pattern encodes the success of the attack action. It is worth mentioning that the input symbol 'H' at location s_7 does not follow this pattern due to the absence of traces where 'H' failed in the small learning sample given to IOALERGIA.

Besides, the learned model includes all the scenarios discovered in the baseline experiment in Figure 8.9, without introducing undesired generalization, compared to figure 8.10. This shows that the explicit encoding is more adequate to represent attack progression than just the attack status.

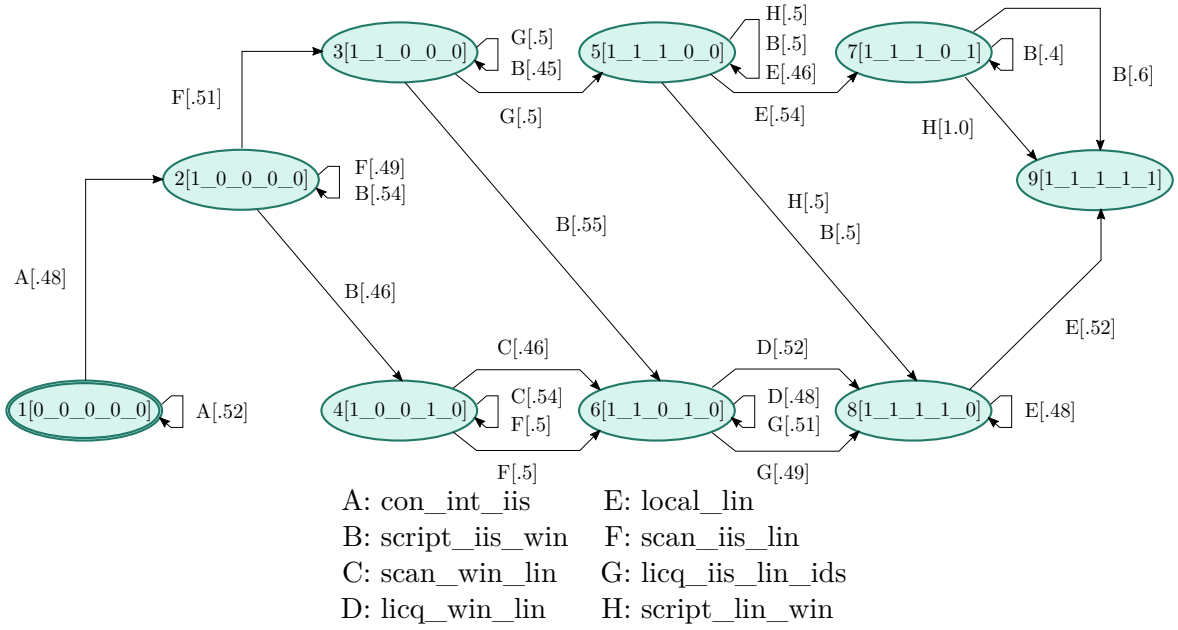


Figure 8.11: Scenario graph of the network intrusion with uncertainty using explicit output encoding

8.6.2.3 Deterministic Strategy Exploration with IEGA

The input of this experiment is the intruder's model previously learned using IOALERGIA and represented in Figure 8.11, with the objective of identifying optimal strategies using IEGA. We set the SMC parameters to ($\delta = 0.01$, $\alpha = 0.1$), which results in the evaluation of more than 14000 traces.

We define a cost structure that takes into account the depth the intruder has to reach to initiate an attack action. Hence, an attack action performed from the IIS web server consumes 1 cost unit while an attack from the Linux or Windows machine has a cost of 2 units, since the intruder has to also pass by the IIS web server. Similarly we assign a cost of 1 unit to the IIS buffer overflow attack on the IIS web server. This cost structure is also extended to carry the impact of the IDS by assigning a cost penalty of 100 units to the attacker when the intrusion is detected. We also set the cost budget to $c_{max} = 20$, that is, any attack that is detected by the IDS would exhaust the attacker resources. Regarding time, attack actions duration is uniformly selected in the interval $[0, 20]$ and the attacker time budget is set to $t_{max} = 300$.

To evaluate the quality of the synthesized strategies, we encoded the attacker model in the Prism dialect that we enriched with the attack cost structure presented above. We then ran Prism to construct optimal strategies minimizing the attack cost, that we finally compared to our strategies. Note that Prism only supports single-objective optimization.

Table 8.3 shows the strategies (and their cost and success probabilities) obtained with both IEGA and Prism with and without the activation of the IDS defense mechanism. We can see

Method	IDS Penalty	Deterministic Strategy	(Cost - Probability)
PRISM	No	$A \rightarrow F \rightarrow G \rightarrow E$	(9.75 - ?)
	Yes	$A \rightarrow F \rightarrow B \rightarrow D \rightarrow E$	(13.55 - ?)
IEGA	No	$A \rightarrow F \rightarrow G \rightarrow E$	(9.60 - 97%)
	Yes	$A \rightarrow F \rightarrow B \rightarrow D \rightarrow E$	(13.17 - 90%)

Table 8.3: Strategy synthesis using PRISM and IEGA with/without IDS penalty

that IEGA identifies the same strategies as PRISM, obtained by an exhaustive search. The slight difference in the score is due to the statistical estimation error of SMC. It is worth mentioning that PRISM strategies only optimize a single objective that is minimizing the cost structure. At contrary, IEGA allows for a multi-criteria exploration of both the cost and the probability to succeed an attack together with additional maximal time and cost bounds.

In terms of execution times, both techniques require few seconds to compute the result. However, this runtime grows exponentially when increasing the size of the model in the case of PRISM, since it relies on an exhaustive exploration. IEGA is more scalable given its small solution encoding and the simplicity of its basic operations.

8.6.2.4 Impactful Defense Synthesis

We use IO-Def to synthesize the most impactful defense configuration by evaluating the role that the IDS mechanism plays in the protection of the network. In this heuristic, the impact of a defense action d in a defense configuration D is computed as:

$$g(D, d) = \frac{C(D) - C(D \setminus \{d\})}{C(D)}$$

where $C(X)$ is the attack cost of the best attacker identified by $IOALERGIA + IEGA$ against the defense configuration X. In this case study, the network is guarded by a single defense, namely, IDS. We can compute the impact of the IDS based on IO-Def as follows:

$$g(\{IDS\}, IDS) = \frac{C(\{IDS\}) - C(\emptyset)}{C(\{IDS\})} = \frac{13.17 - 9.60}{13.17} = 31.13\%$$

The activation of IDS increases the attack cost of the best strategy by a significant ratio of 31.13%. This defense plays an important role in the organization security policy. However, this policy is not sufficient to guarantee a full protection of the network. This weakness is reflected by the very high success probability of 90% for the obtained strategy when IDS is activated. In conclusion, the organization should further prospect to deploy more complex defense mechanisms in order to protect the integrity of its database.

8.7 Related Works

Attack Trees (AT) [112] are widely used in security to model system vulnerabilities and the different combinations of threats to address a malicious goal. Attack-Defense Trees (ADT) [95] extend ATs with defense measures, also known as countermeasures, to include the organizations defenses and bring into consideration the impact of attacks on these organizations. These defense actions try to prevent an attacker from reaching its final goal. More recently, Attack-Countermeasure Trees (ACT) [135] were introduced to model defense mechanisms that are dynamically triggered upon attack detection.

Different types of analysis are proposed on these variants of trees. In [62] authors focus on the probabilistic analysis of ATs, through the computation of the probability, cost, risk and impact of an attack. A similar analysis is performed on ADTs in [147], called Threat Risk Analysis (TRA), and applied to the security assessment of cloud systems. In addition to the aforementioned probabilistic analysis, Roy et al. [135] make use of the structural and Birnbaum importance measure to prioritize attack events and countermeasures in ACTs.

Authors of [69] propose a reinforcement learning method on ADTs to find a near-optimal attack strategy. In this work, an attacker with a complex probabilistic and timed behavior is considered which makes it more difficult to perform a static analysis. The authors propose to address the security analysis problem from the attacker's viewpoint by synthesizing the stochastic and timed strategy that minimizes the attack cost using Uppaal stratego tool. The strategy indicates the attack action to perform in each state in order to realize a successful attack with a minimal cost.

In the previous approach, attack actions are also characterized by time duration as intervals. It identifies the sequence of attack actions and associated duration towards satisfying a specified time budget. However, it is not always the case that an attacker can control the duration of an attack action, eg. the time necessary for a brute-force attack. In [118], we consider time as a characteristic of an attack action, i.e., not controlled as it depends on the system, environment, etc. We consider the maximum time bound as a global success condition of an attack, and we propose IEGA, a hybrid Genetic Algorithm to find the stochastic strategy minimizing the attack cost while maximizing the probability of success. This strategy schedules attack actions and tells the attacker which action to perform when a choice is required.

The proposed approach is an extension of [118] in two points. First, the previous approach studied naive attackers that randomly select any attack action that did not succeed yet. The behavior of the considered attacker has been enhanced to be able to identify possible action dependencies and existing patterns. Second, the previous approach focuses on exploring pure stochastic strategies that define the probability for a naive attacker the select an attack action at each state of that attack, whereas these strategies were extended to deterministic ones, in which the attacker knows which action to perform at every step of the attack in order to optimize its resources and success probability.

8.8 Discussion.

In this chapter, we were interested in addressing the security risk assessment of organizations defenses following an offensive security framework. For this purpose we introduced a workflow combining machine learning techniques and SMC to explore optimal strategies in terms of cost and success probability. The proposed workflow is applicable on systems with different granularities. The worthiness of deployed defense mechanisms is evaluated against abstracted or detailed systems depending on the availability and the complexity of their nominal behavior model.

This workflow relies on the IO-Def heuristic to synthesize defense configurations and IEGA to explore optimal attack strategies. IO-Def varies the defense configurations of the system and quantifies them according to the score of the best attacker strategy, returned by IEGA, in order to find the most impactful defense configuration. IEGA allows for the resolution of a bi-objective optimization problem, that is, the identification of attack strategies with a minimal cost and a maximal success probability. These strategies are either probabilistic or deterministic, in function of the adopted system abstraction level.

We experimented our workflow on several case studies addressing security issues in a variation of domains. We were able to effectively identify the best attack scenarios according to different defense configurations. This proposal also allowed us to give quantitative information about the current systems security and point out weaknesses in the deployed security policies.

In the following, we discuss the proposed workflow and its possible extensions. First, we address questions about the need of a refinement loop and a strategy validation in the workflow. Then, we give insight into the impact of system observability on the risk assessment approach and how structured probabilistic strategies could help in this context. Finally, we conclude with future work.

Should we check the validity of the synthesized strategies? The validity of a strategy refers to its ability to effectively lead the attacker to reach his win condition. A strategy is invalid in two cases: either some of the recommended choice are wrong, e.g., the attacker ends up entering in a cycle where the win condition is never satisfied, or the strategy fails to be played against the black-box system. This latter case is encountered when a strategy is learned on an over-approximated model of an attacker. Consequently, some attack actions that are necessary to perform an attack can be missing. We recall that all the considered strategies by IEGA are quantified using SMC, that requires to collect execution traces obtained by playing these strategies against the actual system. Consequently, invalid strategies would result in a probability of success of 0 and hence would be filtered out by the genetic operators through generations.

Is there a need for an iterative process to refine the learned MDP? In most cases, iterative processes intend to refine the learned model / synthesized strategy. The refinement of the learned model intervenes in two cases: (1) some valid execution traces are not recognized by the learned model, or (2) the process ends with a very generalizing model. The first case can be fixed by increasing the size of the initial learning sample. The probably approximately correct (PAC) learning framework is a possible alternative to provide a sufficient bound on the number of required traces. A bigger and more varied learning sample also has an impact on reducing the generalization in the model by increasing the accuracy of the compatibility criteria. However, the worst case of generalization is when the number of states in the learned model has less states than the original unknown model. This impacts the synthesized strategy. Recall that a deterministic strategy assigns a single action per state of the system. Having less states than the original model means that one learned state is playing several roles which is inconsistent with any deterministic strategy. Note that in practice, the system is a black-box and we cannot know the number of states in the target model.

In our approach there is no need for the process to be iterative as long as the learning sample is big enough. The only remaining reason to iterate would be the generalization. But even this is not a concern because (i) the traces accepted by the learned model and not belonging to the target model represent invalid strategies that are filtered later on (as stated above), and (ii) with an adequate output encoding, where each output symbol encodes the status of individual attacks, we cannot then encounter the situation where an attacker state plays several roles. This comes from the fact that IOALERGIA does not merge two states with different output labels. The same restriction is made on states with error loops on input symbols with respect to states where those input symbols are enabled.

What about probabilistic strategies? Probability distributions are another way to solve non-determinism in MDPs. Applied to a structured attacker model, this would give room for the attacker to probabilistically select an action to perform and not to be restricted to a single action per state, while guaranteeing a mean cost. Probabilistic strategies give more flexibility to the attacker w.r.t the chosen actions. We believe that synthesizing a probabilistic strategy would result in a similar deterministic strategy when it come to minimizing cost since the attacker has no interest in trying different actions at the same location knowing that it implies additional cost.

How does the system's observability affect the whole process? Observability refers here to the ability for an attacker to know whether his attack action succeeded or failed. In our work we assumed that the attacker either receives an answer from the system (e.g. success/failure of a brute force attack on an authentication system) , or sees the effect of his action of on the system (DDoS on a server then it does not respond anymore). But in practice, attack actions can sometimes be non-observable. We can give an example of sending an e-mail

with a hidden malware in a picture. In that example, the success of the action happens when the target opens the picture and the malware is secretly installed. However, the attacker is not notified of it and hence cannot observe the success of his attack action.

In the case of fully observable actions, the proposed output encoding allows us to effectively identify the current status on the attack, resulting into accurate learned models and efficient deterministic strategies. With unobservable actions, the attacker has to manipulate longer substrings in order to keep track of his current status. Indeed an attacker can try to remotely control the target's computer through the sent malware. This action would only work if the unobservable action succeeded. In this context of uncertainty, it may be useful to explore probabilistic strategies where the probabilities can cope with the missing knowledge of the unobservable actions. For example, after sending his e-mail, the attacker reaches an uncertain state where he can either re-send another e-mail (with a nicer picture) or try to exploit the malware. It is obvious that the winning action (e.g. accessing a database) must be observable to guarantee the termination of the attack process.

Future work. The IO-Def heuristic can be adapted for risk assessment from the defense perspective. This can be easily done by extending it to consider a maximal defense budget, which would enable a more realistic analysis. Other criteria, such as the return on investment (ROI) [135], can be also used to evaluate defense actions.

Regarding IEGA, one can study the impact of the different parameters such as the population size, the number of generations, the probability of mutation, the linearization factor and the proportion of best individuals to keep. These parameters together with the intensification using LS could be tweaked in order to reduce the exploration time while synthesizing near-optimal strategies. We also want to experiment different cross-over policies instead of just a fixed single-point cross-over.

In future work, we intend to apply our method to larger models with a higher number of defense mechanisms and vulnerabilities. Also, studying partial observability is an interesting purpose to consider more realistic systems.

Chapter 9

Conclusion and Future Work

9.1 Conclusions

This thesis addresses the problem of designing complex systems. This complexity arises from the heterogeneity of their components, the uncertainty in their environment and the real-time constraints. To overcome this complexity, we adopted a rigorous model-based and component-based approach relying on formal methods. We analyzed the existing challenges, in model-based approaches in general, and identified the need for: (*i*) a modeling formalism that handles real-time and stochasticity in a compositional manner, (*ii*) a faithful construction of performance models to enable performance analysis, (*iii*) automation techniques to ease the design especially for code generation and bug finding, and (*iv*) the formal verification of system correctness with respect to functional and extra-functional requirements.

We introduced a modeling formalism that conciliates the accuracy of a continuous time representation with the flexibility of general probability distributions and probabilistic branching. SRT-BIP is built on the basis of the BIP framework from which it inherits its component-based and real-time capabilities. For probabilistic behaviors, SRT-BIP is formally defined to enable the modeling of probabilistic branching and time stochasticity. To allow for more flexibility when describing this latter, SRT-BIP supports general probability distributions, resulting in an underlying semantics of GSMP. In addition, our implementation provides simulation and C++ code generation capabilities from SRT-BIP models.

For the construction of performance models, we proposed the use of model learning techniques to infer probabilistic timed models from execution traces. Applying such an approach copes with limitations of the ASTROLABE state-of-the-art technique regarding data independence, plot and charts interpretation, and manual model calibration. We adapted and improved a state-of-the-art algorithm to the inference of SRT-BIP models. Results have shown significant improvement in the accuracy of the learned models in terms of precision and recall. However, work still has to be done in this domain to enhance the learning of timed models.

We introduced the $\mathcal{S}$BIP framework for the modeling and the analysis of complex systems exhibiting probabilistic and timed behaviors. This framework is the composition of the SRT-BIP formalism for the modeling, simulation and code generation, and statistical model checking techniques for verification and performance analysis. It enables us to build system models with underlying semantics of DTMC, CTMC, or GSMP. Requirements in this framework are formally expressed using the bounded versions of LTL and MTL. Regarding analysis techniques, $\mathcal{S}$BIP provides support for qualitative and quantitative analyses using classical SMC algorithms, in addition to importance splitting for the analysis of rare properties and an automated technique to explore property parameters. This framework is equipped with a user-friendly graphical interface that centralizes and simplifies the tool's usage. We applied $\mathcal{S}$BIP to the performance evaluation of real-life case studies ranging from communication protocols and concurrent systems to embedded systems. We also built two risk assessment approaches around $\mathcal{S}$BIP to illustrate the interest of SMC in complex workflows.

We proposed a model-based design approach that relies on formal methods to develop real-time resilient systems equipped with FDIR behavior. The proposed approach is centered around $\mathcal{S}$BIP and consists to incrementally refine a nominal model, expressed in SRT-BIP, through model transformations. The impact of safety risks is evaluated through a quantitative risk assessment method using SMC. The approach was successfully used for the design and validation of the control software of a planetary rover.

We addressed the security assessment of organization defenses by proposing an approach for the synthesis of the most impactful defense configurations. This approach explores the efficiency of the defense configurations with respect to sophisticated attackers that aim at fulfilling a malicious goal with the least resources (attack cost) and the highest success probability. We defined the impact of defense configurations as the relative decrease in the attack cost due to the deactivation of a single defense mechanism. Optimized strategies are constructed by combining model learning and genetic algorithms in which strategies cost and probabilities are computed using SMC.

Throughout our experiments, we highlighted the interest of relying on statistical model checking for the verification and performance evaluation of stochastic and real-time systems. This method has the advantage to easily scale to large-scale models while providing an adjustable trade-off between analysis speed and result accuracy. However, in this method, the number of required runs drastically increases when aiming for very high precision and confidence.

Our analyses being based on concrete executions of automatically generated C++ code, a code instrumentation phase is necessary before any analysis. This task is left to the appreciation of the designer and is highly error-prone. Further investigation into the automation of this operation would substantially simplify the analysis process.

9.2 Future Work

In the above contributions, we proposed an answer to the identified challenges and we improved the state-of-the-art in modeling and analyzing complex systems. This work opens the way to several directions for future work, as summarized hereafter.

9.2.1 The SRT-BIP Formalism

Modeling Features. Currently, the SRT-BIP formalism has a purely stochastic semantics that handles both continuous time dynamics and probabilities. This formalism can be extended to handle the modeling of non-deterministic, dynamic and continuous systems.

In practice, some aspects of the system may be under-specified due a lack in the specification or as a result of environment unpredictability. In these cases, non-determinism is a necessary feature of the modeling formalism. Resolving the non-determinism with probabilities would lead to consider only one possible (probabilistic) behavior, and may alter the representativeness of the performance analysis. It would be interesting to extend our semantics by allowing for non-determinism, aiming for an underlying semantics of Stochastic Timed Automata (STA). This would enable us to take benefits from the advances in the SMC techniques on MDPs [51] and extend them to STAs.

Regarding system dynamism, an extension of BIP to model self-reconfiguration in a model-based and architecture-based manner has been proposed in [63]. This formalism, denoted DR-BIP, defines a system as a composition of motifs to describe the architecture, and expresses dynamism in terms of interaction and reconfiguration rules. Analyzing such models using SMC with respect to dynamic properties described in the DynBLTL logic would be interesting to investigate, but one has first to define the stochastic semantics of these systems at three different levels, namely, the atomic behavior, the composition and the reconfiguration.

In contrast with discrete-event simulation, continuous simulation allows one to study phenomena in which the system state changes continuously, usually described with differential equations. Few works have addressed the verification of systems with continuous behaviors [54, 60] using SMC. An interesting extension of our work would be to make use of the SRT-BIP encapsulation and flexibility regarding the execution of external code to combine traditional SRT-BIP components with others having continuous behavior. Simulating such a system can be obtained by relying on Simulink for the continuous part and the SRT-BIP engine for the discrete part, and the global orchestration. Implementing such a simulation procedure in $\mathcal{S}$BIP would enable the use of SMC techniques against properties expressed in a continuous logic, such as, Signal Temporal Logic (STL) [109]. However, the main difficulty is to define the semantics behind the resulting model, and whether it is fully stochastic to permit us to use SMC algorithms.

Engine implementation. The current implementation of SRT-BIP does not actually support the assignment of weights to port/interactions, and hence does not natively support the notion of probabilistic branching. To date, this notion has to be explicitly described in the model by the use of external libraries providing sampling capabilities. We intend to provide a native support for probabilistic branching in a short term perspective. We also plan to extend the collection of available distributions for expressing time stochasticity, such as, normal and gamma distributions, by adding support for the Weibull distribution that is often used to describe software execution times.

9.2.2 Learning Performance Models

Machine learning has promising applications in system design in general, and model-based approaches in particular. As a first step in this direction, we illustrated the usage of these techniques to learn performance models. The main drawbacks of the method we adapted are the inference of a monolithic model, namely the loss of the system decomposition into components, and the reasoning over discretized time intervals. Considering recent work on learning GSMPs [56] should help improve this latter point.

9.2.3 The $\mathcal{S}$BIP Framework

GUI features. The GUI of $\mathcal{S}$BIP represents a significant improvement in the user experience, compared to the previous version of the tool. Adding a graphical modeling interface to the GUI would have a positive impact on the tool usability, and would make it easier to handle for beginner users and also for more advanced ones. This would make the system modeling more convenient and would enable graphical visualization of models.

Specification languages. The modularity and extensibility of $\mathcal{S}$BIP make it easier to extend its specification capabilities with additional classes of properties, such as, quantified (DynBLTL) and continuous (STL) logics. In terms of specification languages for SMC, work still has to be done on the consideration of temporal operators with unbounded horizon. This is crucial for the expression of safety properties and system invariance, using unbounded globally, which cannot be expressed in the bounded LTL and MTL.

Analysis techniques. The tool presents a collection of analysis techniques that can be enriched by further investigating new SMC analyses, and useful automation techniques.

In the literature, rare properties are analyzed using either importance sampling or slitting. However, importance sampling distributions fail to be accurately determined for properties requiring long execution traces, while, in importance splitting, a proper decomposition of a rare property into intermediate levels can be challenging to find, whenever this decomposition exists. We identified the need for new algorithms to analyze rare properties, and we presented

a possible direction to estimate rare properties using monitoring-based guided simulation. We believe that this method could provide interesting results and we intend to further investigate in this direction.

Similarly to the work that we did on security risk assessment, one can make use of genetic algorithms to explore the scheduler space for the statistical model checking of MDPs. The optimization problem consists, like in [51], to identify the highest, respectively the lowest, probability for a non-deterministic system to satisfy a given property.

Regarding automation techniques, we implemented a method to explore parametrized properties. This automated exploration facilitates the identification of optimal parameter values for which a property is satisfied. Performing such a parameter exploration becomes less trivial when dealing with multiple parameters since one has to define a strategy to efficiently explore the set of valuation combinations. An idea would be to adopt an approach similar to the gradient descent in order to iteratively converge to the optimal vector of parameter valuations.

Tool performance. As stated in Chapter 6, the overall performance of the tool would strongly benefit from the enhancement of the SRT-BIP simulation engine. This amelioration can either be addressed by optimizing the required computations by the engine for the identification of the enabled interactions, the sampling of their remaining lifetime and the management of the planning memory, or by refining the interfacing between the engine and $\mathcal{S}$BIP.

The analysis time can also be reduced by considering SMC algorithms that guarantee high confidence with fewer simulations. For example, such an algorithm has been proposed in [90], based on the Massart bounds. Providing a distributed implementation of the SMC algorithms remains the obvious manner to reduce analysis time and we plan to provide such implementations in future releases of $\mathcal{S}$BIP.

In design project, it is often the case that there are more than just one property to evaluate. Hence, a useful automation technique would be to enable the parallel analysis of a set of properties, either from the same family, using a parametrized property, or distinct ones. For each generated trace, one would instantiate a monitor and an SMC core per property, resulting in a 1-producer-n-consumers architecture. The execution of such a technique would terminate when all the SMC cores finish their computation. This technique would significantly reduce the overall analysis time by making use of the parallelism of the analysis platform. This method is particularly interesting when the trace simulation is the most time consuming operation since the same trace is reused for the analysis of the n properties, and hence avoid the simulation of $(n-1) \times M$ traces, with M representing the number of required simulations by SMC to guarantee the desired precision and confidence level.

Congratulations, you have just finished reading this thesis.

List of Figures

List of Tables

Bibliography

[1] ANTLR Web page. http://www.antlr.org/. Accessed: 2019-02-05.

[2] ESROCOS Planetary Exploration Demonstrator. https://github.com/ESROCOS/plex-demonstrator-record.

[3] ESROCOS Project Github Repository. https://github.com/ESROCOS.

[4] Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug 1986.

[5] Roger Aarenstrup. *Managing model-based design*. The MathWorks, Inc., 2015.

[6] Tesnim Abdellatif, Saddek Bensalem, Jacques Combaz, Lavindra de Silva, and Félix Ingrand. Rigorous design of robot software: A formal component-based approach. *Robotics and Autonomous Systems*, 60(12):1563–1578, 2012.

[7] Tesnim Abdellatif, Jacques Combaz, and Joseph Sifakis. Rigorous implementation of real-time systems - from theory to application. *Mathematical Structures in Computer Science*, 23(4):882–914, 2013.

[8] Gul Agha and Karl Palmskog. A survey of statistical model checking. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 28(1):6, 2018.

[9] Bernhard K. Aichernig, Priska Bauerstätter, Elisabeth Jöbstl, Severin Kann, Robert Korošec, Willibald Krenn, Cristinel Mateis, Rupert Schlick, and Richard Schumi. Learning and statistical model checking of system response times. *Software Quality Journal*, Jan 2019.

[10] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and computation*, 104(1):2–34, 1993.

[11] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.

[12] Rajeev Alur, Tomas Feder, and Thomas A Henzinger. The benefits of relaxing punctuality. Technical report, Cornell University, 1994.

[13] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[14] Shiraj Arora, Axel Legay, Tania Richmond, and Louis-Marie Traonouez. Statistical model checking of incomplete stochastic systems. In *International Symposium on Leveraging Applications of Formal Methods*, pages 354–371. Springer, 2018.

[15] Shivangi Arry and Amardeep Kaur. Article: Formal verification of device discovery mechanism using uppaal. *International Journal of Computer Applications*, 58(19):32–37, November 2012.

[16] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking continuous-time markov chains by transient analysis. In *International Conference on Computer Aided Verification*, pages 358–372. Springer, 2000.

[17] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[18] George H Baker and Allan Berg. Supervisory control and data acquisition (scada) systems. *The Critical Infrastructure Protection Report*, 1(6):5–6, 2002.

[19] Thomas Ball and Sriram K Rajamani. The slam toolkit. In *International Conference on Computer Aided Verification*, pages 260–264. Springer, 2001.

[20] Paolo Ballarini, Benoît Barbot, Marie Duflot, Serge Haddad, and Nihal Pekergin. Hasl: A new approach for performance evaluation and model checking from concepts to experimentation. *Performance Evaluation*, 90:53–77, 2015.

[21] Boehm Barry et al. Software engineering economics. *New York*, 197, 1981.

[22] Davide Basile, Maurice H. ter Beek, and Vincenzo Ciancia. Statistical model checking of a moving block railway signalling scenario with uppaal smc. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Verification*, pages 372–391, Cham, 2018. Springer International Publishing.

[23] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Caillaud, Benoît Delahaye, and Axel Legay. Statistical abstraction and model-checking of large heterogeneous systems. In *Formal Techniques for Distributed Systems*, pages 32–46. Springer, 2010.

[24] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in bip. In *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, SEFM'06, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[25] Michel Batteux, Tatiana Prosvirnova, Antoine Rauzy, and Leïla Kloul. The AltaRica 3.0 project for model-based safety assessment. In *11th IEEE International Conference on Industrial Informatics, INDIN 2013, Bochum, Germany, July 29-31, 2013*, pages 741–746, 2013.

[26] Saddek Bensalem, Lavindra de Silva, Andreas Griesmayer, Félix Ingrand, Axel Legay, and Rongjie Yan. A Formal Approach for Incremental Construction with an Application to Autonomous Robotic Systems. In *Software Composition - 10th International Conference, SC 2011, Zurich, Switzerland, June 30 - July 1, 2011. Proceedings*, pages 116–132, 2011.

[27] Saddek Bensalem, Benoît Delahaye, and Axel Legay. Statistical model checking: Present and future. 2010.

[28] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Größer, and Marcin Jurdzinski. Stochastic timed automata. *arXiv preprint arXiv:1410.2128*, 2014.

[29] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, Yunshan Zhu, et al. Bounded model checking. *Advances in computers*, 58(11):117–148, 2003.

[30] Benjamin Bittner, Marco Bozzano, Roberto Cavada, Alessandro Cimatti, Marco Gario, Alberto Griggio, Cristian Mattarei, Andrea Micheli, and Gianni Zampedri. The xSAP Safety Analysis Platform. In *TACAS 2016*, pages 533–539, 2016.

[31] Benjamin Bittner, Marco Bozzano, Alessandro Cimatti, Regis De Ferluc, Marco Gario, Andrea Guiotto, and Yuri Yushtein. An Integrated Process for FDIR Design in Aerospace. In *IMBSA 2014*, pages 82–95, 2014.

[32] Henrik Bohnenkamp, Pedro R d'Argenio, Holger Hermanns, and J-P Katoen. Modest: A compositional modeling formalism for hard and softly timed systems. *IEEE Transactions on Software Engineering*, 32(10):812–830, 2006.

[33] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of nfa. In *IJCAI*, volume 9, pages 1004–1009, 2009.

[34] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. On expressiveness and complexity in real-time model checking. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 124–135, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[35] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 298–302. Springer, 1998.

[36] Tomáš Brázdil, Jan Krčál, Jan Křetínský, and Vojtěch Řehák. *Fixed-Delay Events in Generalized Semi-Markov Processes Revisited*, pages 140–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[37] Lei Bu, Doron Peled, Dachuan Shen, and Yuan Zhuang. Genetic synthesis of concurrent code using model checking and statistical model checking. In *International Symposium on Model Checking Software*, pages 275–291. Springer, 2018.

[38] Peter E Bulychev, Alexandre David, Kim G Larsen, Axel Legay, Guangyuan Li, and Danny Bøgsted Poulsen. Rewrite-based statistical model checking of wmtl. *RV*, 7687:260–275, 2012.

[39] Jonathan W Butts, Robert F Mills, and Rusty O Baldwin. Developing an insider threat model using functional decomposition. In *International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 412–417. Springer, 2005.

[40] Rafael C Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *International Colloquium on Grammatical Inference*, pages 139–152. Springer, 1994.

[41] Everton Cavalcante, Jean Quilbeuf, Louis-Marie Traonouez, Flavio Oquendo, Thais Batista, and Axel Legay. Statistical model checking of dynamic software architectures. In *European Conference on Software Architecture*, pages 185–200. Springer, 2016.

[42] Frédéric Cérou and Arnaud Guyader. Adaptive multilevel splitting for rare event analysis. *Stochastic Analysis and Applications*, 25(2):417–443, 2007.

[43] Yingke Chen and Thomas Dyhre Nielsen. Active learning of markov decision processes for system verification. In *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, volume 2, pages 289–294. IEEE, 2012.

[44] Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, pages 52–71. Springer, 1981.

[45] Edmund M Clarke, E Allen Emerson, Somesh Jha, and A Prasad Sistla. Symmetry reductions in model checking. In *International Conference on Computer Aided Verification*, pages 147–158. Springer, 1998.

[46] Edmund M. Clarke, E Allen Emerson, and A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. volume 8, pages 244–263. ACM, 1986.

[47] Jamieson M Cobleigh, Dimitra Giannakopoulou, and Corina S Păsăreanu. Learning assumptions for compositional verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346. Springer, 2003.

[48] WASC: The Web Application Security Consortium. Web application security statistics, 2008. Accessed on 20 February 2019.

[49] S. Convery, D. Cook, and M. Franz. An attack tree for the border gateway protocol. *Cisco Internet Draft*, 2002.

[50] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings 4th ACM Symp. Principles Prog. Lang*, pages 238–252, 1977.

[51] Pedro D'argenio, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Smart Sampling for Lightweight Verification of Markov Decision Processes. *International Journal on Software Tools for Technology Transfer*, 17(4):469–484, August 2015.

[52] Alexandre David, Kim Larsen, Axel Legay, Marius Mikučionis, and Zheng Wang. Time for statistical model checking of real-time systems. In *Computer Aided Verification*, pages 349–355. Springer, 2011.

[53] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.

[54] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for biological systems. *International Journal on Software Tools for Technology Transfer*, 17(3):351–367, 2015.

[55] Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[56] André de Matos Pedro, Paul Andrew Crocker, and Simão Melo de Sousa. *Learning Stochastic Timed Automata from Sample Executions*, pages 508–523. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[57] Ankush Desai, Shaz Qadeer, and Sanjit A Seshia. Programming Safe Robotics Systems: Challenges and Advances. In *International Symposium on Leveraging Applications of Formal Methods*, pages 103–119. Springer, 2018.

[58] Iulia Dragomir. ESROCOS Planetary Exploration Demonstrator: the Watchdog component in TASTE and BIP. https://github.com/ESROCOS/control-mc_watchdog.

[59] Iulia Dragomir, Simon Iosti, Marius Bozga, and Saddek Bensalem. Designing Systems with Detection and Reconfiguration Capabilities: A Formal Approach. In Bernhard Steffen and Tiziana Margaria, editors, *Leveraging Applications of Formal Methods, Verification and Validation - 8th International Symposium, ISoLA 2018, Lymassol, Cyprus, November 5-9, 2018*, Lecture Notes in Computer Science. Springer, november 2018.

[60] Yann Duplouy. *Applying Formal Methods to Autonomous Vehicle Control*. Theses, Université Paris-Saclay, November 2018.

[61] Pedro R D'Argenio, Arnd Hartmanns, and Sean Sedwards. Lightweight statistical model checking in nondeterministic continuous time. In *International Symposium on Leveraging Applications of Formal Methods*, pages 336–353. Springer, 2018.

[62] Kenneth S Edge, George C Dalton, Richard A Raines, and Robert F Mills. Using attack and protection trees to analyze threats and defenses to homeland security. In *Military Communications Conference, 2006. MILCOM 2006. IEEE*, pages 1–7. IEEE, 2006.

[63] Rim El Ballouli, Saddek Bensalem, Marius Bozga, and Joseph Sifakis. Four exercises in programming dynamic reconfigurable systems: methodology and solution in dr-bip. In *International Symposium on Leveraging Applications of Formal Methods*, pages 304–320. Springer, 2018.

[64] E Allen Emerson and Kedar S Namjoshi. Verification of a parameterized bus arbitration protocol. In *International Conference on Computer Aided Verification*, pages 452–463. Springer, 1998.

[65] Eugene A Feinberg and Adam Shwartz. *Handbook of Markov decision processes: methods and applications*, volume 40. Springer Science & Business Media, 2012.

[66] Bernd Finkbeiner and Henny Sipma. Checking finite traces using alternating automata. *Formal Methods in System Design*, 24(2):101–127, 2004.

[67] Mohammed Foughali, Bernard Berthomieu, Silvano Dal Zilio, Pierre-Emmanuel Hladik, Félix Ingrand, and Anthony Mallet. Formal Verification of Complex Robotic Systems on Resource-Constrained Platforms. In *FormaliSE: 6th International Conference on Formal Methods in Software Engineering*, 2018.

[68] Susumu Fujiwara, G v Bochmann, Ferhat Khendek, Mokhtar Amalou, and Abderrazak Ghedamsi. Test selection based on finite state models. *IEEE Transactions on software engineering*, 17(6):591–603, 1991.

[69] Olga Gadyatskaya, René Rydhof Hansen, Kim Guldstrand Larsen, Axel Legay, Mads Chr Olesen, and Danny Bøgsted Poulsen. Modelling attack-defense trees using timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 35–50. Springer, 2016.

[70] Antonio-Javier Gallego, Damián López, and Jorge Calera-Rubio. Grammatical inference of directed acyclic graph languages with polynomial time complexity. *Journal of Computer and System Sciences*, 95:19–34, 2018.

[71] Pedro Garcia, Enrique Vidal, and José Oncina. Learning locally testable languages in the strict sense. In *ALT*, pages 325–338, 1990.

[72] Sebastian Gerwinn, Eike Möhlmann, and Anja Sieper. *Statistical Model Checking for Scenario-Based Verification of ADAS*, pages 67–87. Springer International Publishing, Cham, 2019.

[73] Peter W Glynn. A gsmp formalism for discrete event systems. *Proceedings of the IEEE*, 77(1):14–23, 1989.

[74] Patrice Godefroid. Using partial orders to improve automatic verification methods. In *International Conference on Computer Aided Verification*, pages 176–185. Springer, 1990.

[75] E Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

[76] Joseph Y Halpern and Moshe Y Vardi. Model checking vs. theorem proving: a manifesto. *Artificial intelligence and mathematical theory of computation*, 212:151–176, 1991.

[77] Mor Harchol-Balter and Allen B Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems (TOCS)*, 15(3):253–285, 1997.

[78] Thomas A Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Software verification with blast. In *International SPIN Workshop on Model Checking of Software*, pages 235–239. Springer, 2003.

[79] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'04*, pages 73–84, January 2004.

[80] Gerard J. Holzmann. The model checker spin. *IEEE Transactions on software engineering*, 23(5):279–295, 1997.

[81] Falk Howar and Bernhard Steffen. Active automata learning in practice. In *Machine Learning for Dynamic Software Analysis: Potentials and Limits*, pages 123–148. Springer, 2018.

[82] SANS ICS. Analysis of the cyber attack on the ukrainian power grid, march 2016. Accessed on 25 April 2018.

[83] Malte Isberner, Falk Howar, and Bernhard Steffen. The ttt algorithm: a redundancy-free approach to active automata learning. In *International Conference on Runtime Verification*, pages 307–322. Springer, 2014.

[84] Risk management - guidelines. Standard, International Organization for Standardization, Geneva, CH, February 2018.

[85] Cyrille Jegourel. *Rare event simulation for statistical model checking.* PhD thesis, Université Rennes 1, 2014.

[86] Cyrille Jegourel, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. *Importance Sampling for Stochastic Timed Automata*, pages 163–178. Springer International Publishing, Cham, 2016.

[87] Cyrille Jegourel, Axel Legay, and Sean Sedwards. Cross-entropy optimisation of importance sampling parameters for statistical model checking. In *International Conference on Computer Aided Verification*, pages 327–342. Springer, 2012.

[88] Cyrille Jegourel, Axel Legay, and Sean Sedwards. A platform for high performance statistical model checking — plasma. In *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'12, pages 498–503, Berlin, Heidelberg, 2012. Springer-Verlag.

[89] Cyrille Jegourel, Axel Legay, and Sean Sedwards. Importance splitting for statistical model checking rare properties. In *CAV*, volume 13, pages 576–591. Springer, 2013.

[90] Cyrille Jegourel, Jun Sun, and Jin Song Dong. Sequential schemes for frequentist estimation of properties in statistical model checking. In *International Conference on Quantitative Evaluation of Systems*, pages 333–350. Springer, 2017.

[91] Zhihao Jiang, Miroslav Pajic, Salar Moarref, Rajeev Alur, and Rahul Mangharam. Modeling and verification of a dual chamber implantable pacemaker. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 188–203, 2012.

[92] Max Kanovich, Tajana Ban Kirigin, Vivek Nigam, Andre Scedrov, and Carolyn Talcott. Discrete vs. dense times in the analysis of cyber-physical security protocols. In Riccardo Focardi and Andrew Myers, editors, *Principles of Security and Trust*, pages 259–279, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[93] Joost-Pieter Katoen, Ivan S Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N Jansen. The ins and outs of the probabilistic model checker mrmc. *Performance evaluation*, 68(2):90–104, 2011.

[94] Robert M Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[95] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of attack–defense trees. In *International Workshop on Formal Aspects in Security and Trust*, pages 80–95. Springer, 2010.

[96] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, Nov 1990.

[97] Dirk P Kroese and Reuven Y Rubinstein. Monte carlo methods. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(1):48–58, 2012.

[98] Vidyadhar G Kulkarni. *Introduction to Modeling and Analysis of Stochastic Systems*. Springer New York, 2011.

[99] Vidyadhar G Kulkarni. *Modeling and analysis of stochastic systems*. Chapman and Hall/CRC, 2016.

[100] Nirman Kumar, Koushik Sen, José Meseguer, and Gul Agha. A rewriting based model for probabilistic distributed object systems. In Elie Najm, Uwe Nestmann, and Perdita Stevens, editors, *FMOODS*, pages 32–46, 2003.

[101] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: verification of probabilistic real-time systems. In *Proceedings of the 23rd international conference on Computer aided verification*, CAV'11, pages 585–591, Berlin, Heidelberg, 2011. Springer-Verlag.

[102] Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.

[103] Marta Kwiatkowska, Gethin Norman, Jeremy Sproston, and Fuzhi Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7):1027–1077, 2007.

[104] Kevin J Lang, Barak A Pearlmutter, and Rodney A Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *International Colloquium on Grammatical Inference*, pages 1–12. Springer, 1998.

[105] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *International conference on runtime verification*, pages 122–135. Springer, 2010.

[106] Axel Legay, Dirk Nowotka, Danny Bøgsted Poulsen, and Louis-Marie Tranouez. Statistical model checking of llvm code. In Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik de Vink, editors, *Formal Methods*, pages 542–549, Cham, 2018. Springer International Publishing.

[107] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal design and analysis of a gear controller. *International Journal on Software Tools for Technology Transfer (STTT)*, 3(3):353–368, 2001.

[108] Alexander Maier, Asmir Vodencarevic, Oliver Niggemann, Roman Just, and Michael Jaeger. Anomaly detection in production plants using timed automata. In *8th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 363–369, 2011.

[109] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[110] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D Nielsen, Kim G Larsen, and Brian Nielsen. Learning probabilistic automata for model checking. In *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, pages 111–120. IEEE, 2011.

[111] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D Nielsen, Kim G Larsen, and Brian Nielsen. Learning markov decision processes for model checking. *arXiv preprint arXiv:1212.3873*, 2012.

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

[113] Franz Mayr and Sergio Yovine. Regular inference on artificial neural networks. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl, editors, *Machine Learning and Knowledge Extraction*, pages 350–369, Cham, 2018. Springer International Publishing.

[114] Kenneth L McMillan. Symbolic model checking. In *Symbolic Model Checking*, pages 25–60. Springer, 1993.

[115] Braham Lotfi Mediouni, Smail Niar, Rachid Benmansour, Karima Benatchba, and Mouloud Koudil. A bi-objective heuristic for heterogeneous mpsoc design space exploration. In *Design & Test Symposium (IDT), 2015 10th International*, pages 90–95. IEEE, 2015.

[116] Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, and Saddek Bensalem. Improved learning for stochastic timed models by state-merging algorithms. In *NASA Formal Methods Symposium*, pages 178–193. Springer, 2017.

[117] Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Mahieddine Dellabani, Axel Legay, and Saddek Bensalem. $\mathcal{S}$BIP 2.0: Statistical Model Checking Stochastic Real-Time Systems. In *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, pages 536–542, 2018.

[118] Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Axel Legay, and Saddek Bensalem. Mitigating security risks through attack strategies exploration. *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, 2018.

[119] Stefan Mitsch, Khalil Ghorbal, David Vogelbacher, and André Platzer. Formal verification of obstacle avoidance and navigation of ground robots. *The International Journal of Robotics Research*, 36(12):1312–1340, 2017.

[120] Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michał Szynwelski. Learning nominal automata. *arXiv preprint arXiv:1607.06268*, 2016.

[121] Miguel Munoz, Giuseppe Montano, Malte Wirkus, Kilian Hoeflinger, Daniel Silveira, Nikolaos Tsiogkas, Jerome Hugues, Herman Bruyninckx, Iulia Dragomir, and Ali Muhammad. ESROCOS: a Robotic Operating System for Space and Terrestrial Applications. In *Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA) 2017, Leiden, Netherlands, June 20-22, 2017*, june 2017.

[122] Ayoub Nouri, Saddek Bensalem, Marius Bozga, Benoit Delahaye, Cyrille Jegourel, and Axel Legay. Statistical model checking QoS properties of systems with SBIP. *Int. J. Softw. Tools Technol. Transf. (STTT)*, 17(2):171–185, April 2015.

[123] Ayoub Nouri, Marius Bozga, Anca Molnos, Axel Legay, and Saddek Bensalem. Astrolabe: A rigorous approach for system-level performance modeling and analysis. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(2):31, 2016.

[124] Ayoub Nouri, Braham Lotfi Mediouni, Marius Bozga, Jacques Combaz, Saddek Bensalem, and Axel Legay. Performance evaluation of stochastic real-time systems with the SBIP framework. *International Journal of Critical Computer-Based Systems*, 8(3-4):340–370, 2018.

[125] Jorge Ocon, Francisco Colemenero, Joaquin Estremera, Karl Buckley, Mercedes Alonso, Enrique Heredia, Javier Garcia, Amanda Coles, Andrew Coles, Moises Martinez, Emre Savas, Florian Pommerening, Thomas Keller, Spyros Karachalios, Mark Woods, Iulia Dragomir, Saddek Bensalem, Pierre Dissaux, Arnaud Schach, Robert Marc, and Piotr Weclewski. The ERGO framework and its use in planetary/orbital scenarios. In *International Astronautical Congress (IAC) 2018, Bremen, Germany, October 1-5, 2018*, october 2018.

[126] Masashi Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the institute of Statistical Mathematics*, 10(1):29–35, 1959.

[127] José Oncina and Pedro Garcia. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, pages 99–108. World Scientific, 1992.

[128] Joël Ouaknine and James Worrell. Safety metric temporal logic is fully decidable. In Holger Hermanns and Jens Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 411–425, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[129] Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems*, pages 1–13, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[130] Doron Peled, Moshe Y Vardi, and Mihalis Yannakakis. Black box checking. In *Formal Methods for Protocol Engineering and Distributed Systems*, pages 225–240. Springer, 1999.

[131] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977.

[132] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on programming*, pages 337–351. Springer, 1982.

[133] Jean Quilbeuf, Mathieu Barbier, Lukas Rummelhard, Christian Laugier, Axel Legay, Blanche Baudouin, Thomas Genevois, Javier Ibañez-Guzmán, and Olivier Simonin. Statistical model checking applied on perception and decision-making systems for autonomous driving. In *10th Workshop on Planning, Perception and Navigation for Intelligent Vehicles at the IEEE International Conference on Intelligent Robots and Systems, october 2018*, 2018.

[134] Jean Quilbeuf, Everton Cavalcante, Louis-Marie Traonouez, Flavio Oquendo, Thais Batista, and Axel Legay. A Logic for the Statistical Model Checking of Dynamic Software Architectures. In *ISoLA*, volume 9952 of *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*, pages 806 – 820, Corfou, Greece, October 2016. Springer.

[135] Arpan Roy, Dong Seong Kim, and Kishor S Trivedi. Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. *Security and Communication Networks*, 5(8):929–943, 2012.

[136] Enno Ruijters, Dennis Guck, Peter Drolenga, and Mariëlle Stoelinga. Fault maintenance trees: reliability centered maintenance via statistical model checking. In *2016 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–6. IEEE, 2016.

[137] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Learning continuous time markov chains from sample executions. In *Proceedings of the The Quantitative Evaluation of Systems, First International Conference*, QEST '04, pages 146–155, Washington, DC, USA, 2004. IEEE Computer Society.

[138] Koushik Sen, Mahesh Viswanathan, and Gul A. Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. In *International Conference on the Quantitative Evaluation of Systems, QEST'05*, pages 251–252, 2005.

[139] Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 256–296. Springer, 2011.

[140] Franck Thollard, Pierre Dupont, Colin de la Higuera, et al. Probabilistic dfa inference using kullback-leibler divergence and minimality. In *ICML*, pages 975–982, 2000.

[141] Tarik Tosun, Gangyuan Jing, Hadas Kress-Gazit, and Mark Yim. Computer-aided compositional design and verification for modular robots. In *Robotics Research*, pages 237–252. Springer, 2018.

[142] Sicco Ewout Verwer. *Efficient identification of timed automata: Theory and practice*. PhD thesis, TU Delft, Delft University of Technology, 2010.

[143] Abraham Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.

[144] Abraham Wald. Statistical decision functions. 1950.

[145] Neil Walkinshaw, Ramsay Taylor, and John Derrick. Inferring extended finite state machine models from software executions. *Empirical Software Engineering*, 21(3):811–853, 2016.

[146] Alexandra Wander and Roger Förstner. Innovative Fault Detection, Isolation and Recovery Strategies On-board Spacecraft: State of the Art and Research Challenges. Deutscher Luft- und Raumfahrtkongress, 2012.

[147] Ping Wang, Wen-Hui Lin, Pu-Tsun Kuo, Hui-Tang Lin, and Tzu Chia Wang. Threat risk analysis for cloud security based on attack-defense trees. In *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*, volume 1, pages 106–111. IEEE, 2012.

[148] Yu Wang, Siddhartha Nalluri, Borzoo Bonakdarpour, and Miroslav Pajic. Statistical model checking for probabilistic hyperproperties. *arXiv preprint arXiv:1902.04111*, 2019.

[149] Jeannette M Wing et al. Scenario graphs applied to network security. *Information assurance: survivability and security in networked systems*, pages 247–277, 2008.

[150] Nan Yang, Kousar Aslam, Ramon Schiffelers, Leonard Lensink, Dennis Hendriks, LGWA Cleophas, and Alexander Serebrenik. Improving model inference in industry by combining active and passive learning. In *IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2018.

[151] Håkan LS Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon, 2005.

[152] Håkan LS Younes, Marta Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 8(3):216–228, 2006.

[153] Håkan L. S. Younes. Ymer: A statistical model checker. In *Computer Aided Verification, CAV'05*, pages 429–433. Springer, 2005.

[154] E. Zio. The future of risk assessment. *Reliability Engineering & System Safety*, 177:176 – 190, 2018.

[155] Paolo Zuliani, André Platzer, and Edmund M Clarke. Bayesian statistical model checking with application to stateflow/simulink verification. *Formal Methods in System Design*, 43(2):338–367, 2013.

Appendix A

A Vehicle Gear Controller: Extended Case Study

A.1 The Complete Set of Considered Requirements

The complete set of verified requirements is listed below (the required verification time is given between brackets). Note that some requirements are expressed as several MTL properties. For instance, requirement P_{13} induces 6 MTL properties. We also point out the fact that requirements $P_{9,10}$ were not considered since they address events that are very unlikely to occur. Note that Importance Splitting also fails to analyze them due to the impossibility to identify suitable levels (that are less rare).

- P_1. The gear can be changed. *[41s]* $\Diamond_{[0,1500]}$ $(gc.GearChanged)$
- P_2. The gear can be set to gear 5 and the reverse gear. *[16m 11s]*
  a. $\Diamond_{[0,1000000]}$ $(inf.gear = 5)$
  b. $\Diamond_{[0,1000000]}$ $(inf.gear = -1)$
- P_3. The switch gear can be performed in 1000 ms. *[42s]* $\Diamond_{[0,1500]}$ $(gc.GearChanged \wedge gc.SysTimer \leq 1000)$
- P_4. When the gearbox is in position N, the gear is zero. *[5m 16s]* $\Box_{[0,10000]}$ $((gb.Neutral \wedge inf.gear = 0) \vee \neg gb.Neutral \vee \neg inf.stableGear)$
- P_5. If the gearbox is idle then the gear is never N. *[10m 47s]*
  a. $\Box_{[0,10000]}$ $(\neg gb.Idle \vee \neg inf.N)$
  b. $\Box_{[0,10000]}$ $(\neg inf.N \vee gb.Neutral)$

- P_6. If there are no errors in gear and clutch and the engine is in normal mode, a gear switch is guaranteed in 900 ms, a switch gear can never be performed in less than 150 ms, and if the switch is not from/to gear N, a switch gear cannot be done in less than 400 ms. *[1m 26s]*

  a. a gear switch is guaranteed in 900 ms
  $\Diamond_{[0,900]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg e.UseCase = 0 \vee gc.GearChanged)$

  b. A switch gear can never be performed in less than 150 ms
  $\Diamond_{[0,150]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg\ e.UseCase = 0 \vee \neg\ gc.GearChanged)$

  c. If the switch is not from/to gear N, a switch gear cannot be done in less than 400 ms
  $\Box_{[0,400]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg e.UseCase = 0 \vee \neg(inf.FromGear > 0\ \vee\ \neg inf.ToGear > 0\ \vee\ \neg gc.GearChanged)$

- P_7. If there are no errors in gear and clutch but engine in zero torque mode, a gear switch is guaranteed in 1055 ms, a switch gear can never be performed in less than 550 ms, and if the switch is not from/to gear N, a switch gear cannot be done in less than 700 ms. *[1m 30s]*

  a. a gear switch is guaranteed in 1055 ms
  $\Diamond_{[0,1055]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg e.UseCase = 1 \vee gc.GearChanged)$

  b. switch gear can never be performed in less than 550 ms
  $\Diamond_{[0,550]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg e.UseCase = 1 \vee (\neg gc.GearChanged \wedge \neg gc.Gear))$

  c. If the switch is not from/to gear N, a switch gear cannot be done in less than 700 ms
  $\Box_{[0,700]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg e.UseCase = 1 \vee \neg inf.FromGear > 0\ \vee\ \neg inf.ToGear > 0\ \vee\ \neg gc.GearChanged\ \vee\ \neg gc.Gear)$

- P_8. If there are no errors in gear and clutch but engine in synchronous speed mode, a gear switch is guaranteed in 1205 ms, a switch gear can never be performed in less than 450 ms, and if the switch is not from/to gear N, a switch gear cannot be done in less than 750 ms. *[1m 31s]*

  a. a gear switch is guaranteed in 1205 ms
  $\Diamond_{[0,1205]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg e.UseCase = 2 \vee gc.GearChanged)$

  b. switch gear can never be performed in less than 450 ms
  $\Diamond_{[0,450]}\ (\neg\ e.UseCase = 2\ \vee\ (\neg gc.GearChanged\ \wedge\ \neg gc.Gear))$

  c. If the switch is not from/to gear N, a switch gear cannot be done in less than 750 ms
  $\Box_{[0,750]}\ (\neg gb.ErrStat = 0 \vee \neg c.ErrStat = 0 \vee \neg e.UseCase = 2 \vee \neg inf.FromGear > 0\ \vee\ \neg inf.ToGear > 0\ \vee\ \neg gc.GearChanged\ \vee\ \neg gc.Gear)$

- P_{11}. The engine is guaranteed to find synchronous speed in the case where no error occurs in it. *[5m 52s]*

  $\Box_{[0,10000]}\ (\neg gb.ErrStat = 0\ \vee\ \neg c.ErrStat = 0\ \vee\ \neg e.isError)$

- P_{12}. If the gear is N, the engine is either in initial or going to initial (i.e. ToGear = 0 and engine in zero). *[5m 22s]*

  $\Box_{[0,10000]}\ (\neg inf.N\ \vee\ (inf.ToGear = 0\ \wedge\ e.Zero)\ \vee\ e.Initial)$

- P_{13}. Torque is always indicated in the engine when the gear controller has a gear set. *[31m 17s]*

  a. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = -1\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

  b. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = 1\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

  c. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = 2\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

  d. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = 3\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

  e. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = 4\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

  f. $\Box_{[0,10000]}\ (\neg gc.Gear\ \vee\ \neg\ inf.gear = 5\ \vee\ \neg inf.stableGear\ \vee\ e.Torque)$

- P_{14}. The controller is in predefined locations depending on the clutch state. *[10m 31s]*

  a. If clutch is open

  $\Box_{[0,10000]}\ (\neg c.Open \vee gc.ClutchOpen \vee gc.ClutchOpenTwo \vee gc.CheckGearSetTwo \vee gc.ReqSetGearTwo \vee gc.ClutchClose \vee gc.CheckClutchClosed \vee gc.CheckClutchClosedTwo \vee gc.CheckGearNeuTwo)$

  b. If clutch is closed

  $\Box_{[0,10000]}\ (\neg c.Closed \vee gc.ReqTorqueC \vee gc.GearChanged \vee gc.Gear \vee gc.Initiate \vee gc.CheckTorque\ \vee\ gc.ReqNeuGear\ \vee\ gc.CheckGearNeu\ \vee\ gc.ReqSyncSpeed\ \vee\ gc.CheckSyncSpeed\ \vee\ gc.ReqSetGear\ \vee\ gc.CheckGearSet)$

- P_{15}. The controller is in predefined locations depending on the gearbox status. *[10m 32s]*

  a. If gear is idle

  $\Box_{[0,10000]}\ (\neg c.Open \vee gc.ClutchOpen \vee gc.ClutchOpenTwo \vee gc.CheckGearSetTwo \vee gc.ReqSetGearTwo \vee gc.ClutchClose \vee gc.CheckClutchClosed \vee gc.CheckClutchClosedTwo \vee gc.CheckGearNeuTwo)$

  b. If gear is neutral

  $\Box_{[0,10000]}\ (\neg gb.Neutral \vee gc.ReqSetGear \vee gc.CheckClutchClosedTwo \vee gc.ReqTorqueC \vee gc.GearChanged \vee gc.Gear \vee gc.Initiate \vee gc.ReqSyncSpeed \vee gc.CheckSyncSpeed \vee gc.ReqSetGear\ \vee\ gc.CheckClutch\ \vee\ gc.ClutchOpen\ \vee\ gc.ReqSetGearTwo)$

- P_{16}. If engine regulates on torque, then the clutch must be closed. *[6m 08s]*

  $\Box_{[0,10000]}\ (\neg e.Torque\ \vee\ c.Closed)$

Appendix B

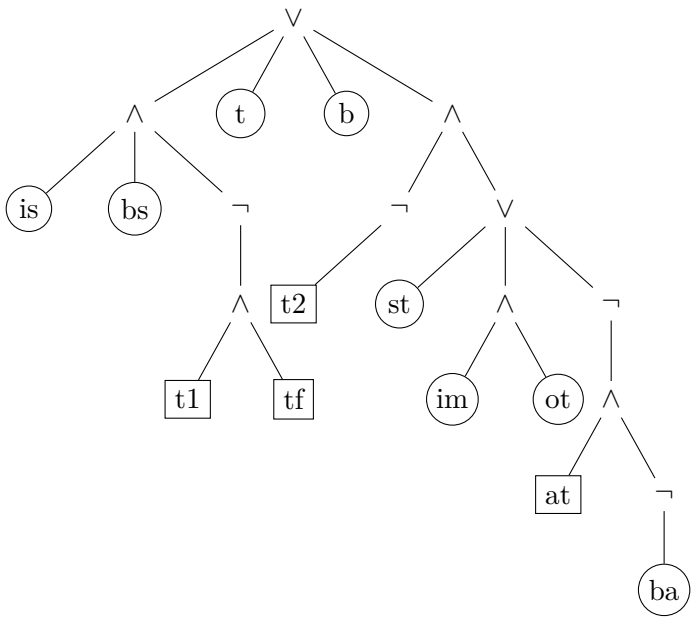
Assessing Systems Security with SMC: Case Studies Description

In the following case study descriptions, attack actions are characterized by their lower (LB) and upper (UB) time bounds, the required resources (Cost) and their probability to succeed (Env). In the ADTs, attack actions are represented by ellipses and defense actions by rectangles. Please note that the system models are abstracted.

B.1 An Organization System Attack (ORGA)

Figure B.1 describes an Organization System Attack (ORGA) taken from [69]. In this case study, an attacker aims at removing an RFID-tag from a warehouse. To achieve this task, he can proceed in different ways among which he can, for instance, infiltrate management and order a new tag, then in the second phase send a false tag.

The purpose of this case study is mainly to use it as a comparison basis between our proposed approach based on genetic algorithms and the reinforcement learning approach proposed in STRATEGO. The application of this latter method on this case study is reported in [69]. However, due to technical problems related to the released UPPAAL-STRATEGO, the authors of this paper kindly accepted to rerun the experiments for us to be able to give a faithful comparison. In fact, the reported results in the STRATEGO column of Table 8.2 are provided by the development team of that tool.



(a) Attack-Defense Tree

Action	LB	UB	Cost	Env
Identify Subject (is)	0	20	80	0.8
Bribe Subject (bs)	0	20	100	0.7
Threaten (t)	0	20	700	0.7
Blackmail (b)	0	20	700	0.7
Send false Tag (st)	0	20	50	0.5
Break Authentication (ba)	0	20	85	0.6
Infiltrate Management (im)	0	20	70	0.5
Order Tag replacement (ot)	0	20	0	0.6

(b) Attack actions characteristics

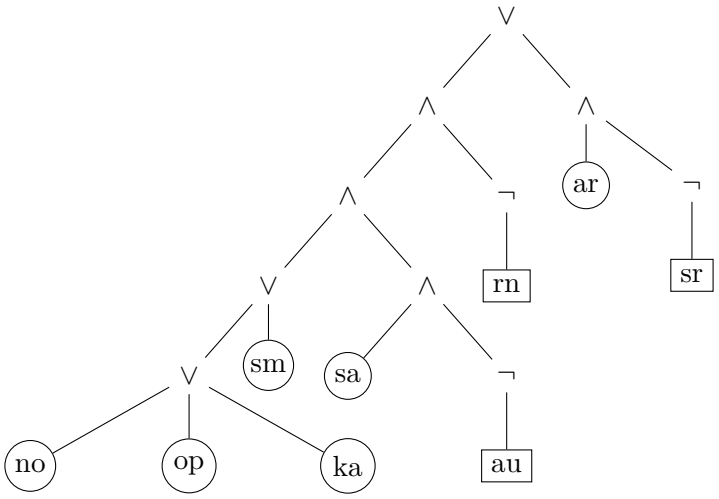
Defense action	Label
t1	Training for thwart
tf	Threaten to Fire employees
t2	Training for trick
at	Authenticate Tag

(c) Defense actions labels

Figure B.1: ORGA case study description

B.2 Resetting a BGP Session (BGP)

We constructed this case study based on [135], in which detection and mitigation events are attached with success probabilities (resp. P_D and P_M). We transpose these probabilities to the attack actions in a straightforward manner: the probability of an attack action to succeed is computed as the probability that all the implemented countermeasures set to block it, fail. For example, the attack action *sa* can be blocked by both defense actions *au* and *rn*. So, the probability of *sa* to succeed equals $Env(sa) = (1 - P_{D1} \times P_{M1}) \times (1 - P_{D2} \times P_{M2})$, where P_{D1}, P_{D2}, P_{M1} and P_{M2} are given in [135]. Note that, in our case, a pair of detection-mitigation events is combined is a single defense action. For example, P_{D1} and P_{M1} are merged into a defense *au*, and, P_{D2} and P_{M2} into the defense action *rn*. Also, the defense mechanisms are fixed before starting an analysis and have a probability 1.



(a) Attack-Defense Tree

Action	LB	UB	Cost	Env
Send RST message to TCP stack (sm)	0	20	50	0.7
Send BGP message: notify (no)	0	20	60	0.7
Send BGP message: open (op)	0	20	70	0.7
Send BGP message: keep alive (ka)	0	20	100	0.7
TCP sequence number attack (sa)	0	20	150	0.42
Alter config. via router (ar)	0	20	190	0.65

(b) Attack actions characteristics

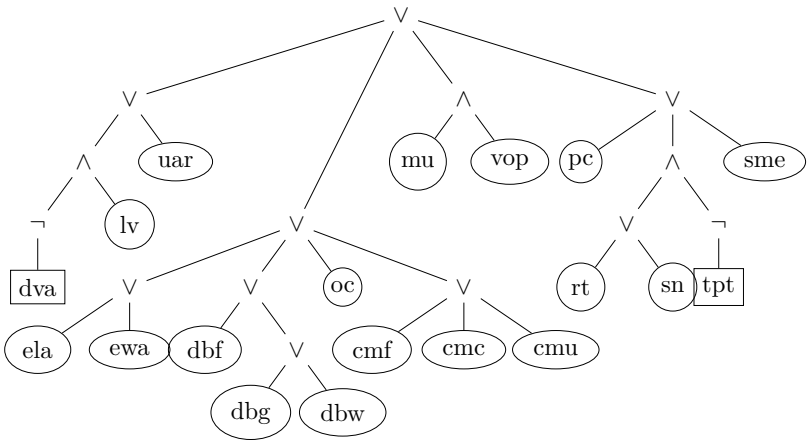
Defense action	Label
au	Check TCP sequence number by MD5 authentication
rn	Check Trace-route by using randomized sequence numbers
sr	Secure routers with firewall alert

(c) Defense actions labels

Figure B.2: Resetting a BGP session description [49]

B.3 A Malicious Insider Attack (MI)

In what follows, we describe a Malicious Insider attack (MI) [39] . It is presented in [135] and is adapted to our context in a similar way to BGP.



(a) Attack-Defense Tree

Action	LB	UB	Cost	Env
Unauthorized alternation of registry (uar)	0	20	50	0.08
Launch virus (lv)	0	20	60	0.07
Email local account (ela)	0	20	70	0.15
Email web-based account (ewa)	0	20	100	0.2
Drop-box: FTP to file server (dbf)	0	20	150	0.1
Drop-box: post to new group (dbg)	0	20	190	0.4
Drop-box: post to website (dbw)	0	20	100	0.1
Online chat (oc)	0	20	110	0.1
Copy to media: Floppy disk (cmf)	0	20	90	0.1
Copy to media: CD-ROM (cmc)	0	20	250	0.25
Copy to media: USB drive (cmu)	0	20	275	0.3
Misuse (mu)	0	20	100	0.2
Violation of organization policy (vop)	0	20	120	0.15
Poor configuration (pc)	0	20	100	0.15
Sniff Network (sn)	0	20	30	0.18
Root Telnet (rt)	0	20	40	0.12
Sendmail exploit (sme)	0	20	170	0.5

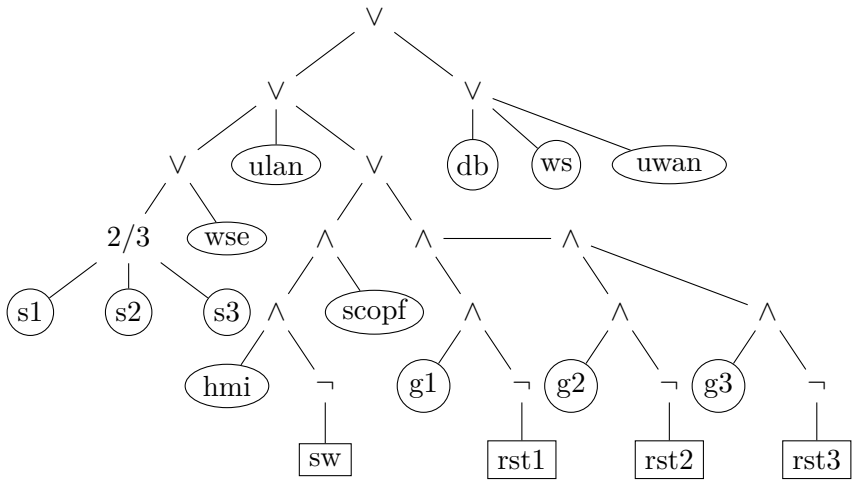
(b) Attack actions characteristics

Defense action	Label
dva	Detect viruses with anti-virus
tpt	Track number of tries at password

(c) Defense actions labels

Figure B.3: A Malicious Insider attack (MI) description

B.4 Supervisory Control And Data Acquisition System (SCADA)



(a) Attack-Defense Tree

Action	LB	UB	Cost	Env
Sensor one (s1)	0	20	100	0.1
Sensor two (s2)	0	20	110	0.1
Sensor three (s3)	0	20	90	0.1
Wrong estimation (wse)	0	20	250	0.25
Unavailable network LAN (ulan)	0	20	275	0.3
Control server one (hmi)	0	20	100	0.15
Control server two (scopf)	0	20	120	0.15
Controlling agent one (g1)	0	20	100	0.09
Controlling agent two (g2)	0	20	30	0.15
Controlling agent three (g3)	0	20	40	0.08
Database (db)	0	20	170	0.5
Unavailable network (uwan)	0	20	160	0.35
Workstation (ws)	0	20	150	0.4

(b) Attack actions characteristics

Defense action	Label
sw	Switch
rst1	Restart agent one if an attack is detected on it
rst2	Restart agent two if an attack is detected on it
rst3	Restart agent three if an attack is detected on it

(c) Defense actions labels

Figure B.4: SCADA system description [18]

Similarly to BGP, SCADA is inspired from [135]. This case study represents an example of how attack trees are used to answer the failure assessment problem where attack actions represent the possible hardware/software failures. Since we are interested to identify what an attacker can do to reach a malicious goal on a system, we then interpret these attack actions as an attacker trying to trigger a hardware/software failure.

In addition to the transposition from probabilities of successful defenses to probabilities of successful attack actions, Env also scales with the probability of failures. For example, the probability of $g1$ to succeed, provided it is guarded by a defense $rst1$, is computed as: $Env(g1) = P_{g1} \times (1 - P_D \times P_M)$, where the probabilities of a failure of the controlling agent one P_{g1}, the detection of its failure P_D and its restarting P_M are given in [135].

In Figure B.4a, the operator "2/3" is a shortcut designating the case where at least two events s_i and s_j occur, with $i \neq j$. It is equivalent to the boolean expression $\phi = (s1 \wedge s2) \vee (s1 \wedge s3) \vee (s2 \wedge s3)$.