Formal Verification of Cyber-Physical Systems in the Industrial Model-Based Design Process

Nikolaos Kekatos

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministérial : 25 mai 2016

Présentée par

Nikolaos KEKATOS

Thèse dirigée par **GORAN FREHSE**,
et codirigée par **THAO DANG**

préparée au sein du **Laboratoire VERIMAG**
dans **l'Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Vérification Formelle des Systèmes Cyber-Physiques dans le Processus Industriel de la Conception Basée sur Modèle

Formal Verification of Cyber-Physical Systems in the Industrial Model-Based Design Process

Thèse soutenue publiquement le **17 Decémbre 2018**,
devant le jury composé de :

Monsieur GORAN FREHSE
MAÎTRE DE CONFÉRENCES, ENSTA ParisTech - PALAISEAU, Directeur de thèse

Monsieur BENOÎT CAILLAUD
DIRECTEUR DE RECHERCHE, INRIA CENTRE RENNES-BRETAGNE ATLANTIQUE, Rapporteur

Monsieur LAURENT FRIBOURG
DIRECTEUR DE RECHERCHE, CNRS ILE-DE FRANCE GIF-SUR YVETTE, Président

Madame THAO DANG
DIRECTRICE DE RECHERCHE, CNRS DELEGATION ALPES, Co-directrice de thèse

Monsieur ALEXANDRE CHAPOUTOT
MAÎTRE DE CONFÉRENCES, ENSTA ParisTech - PALAISEAU, Examinateur

Abstract

Cyber-Physical Systems form a class of complex, large-scale systems of frequently safety-critical nature. Formal verification approaches can provide performance and safety guarantees for these systems. They require two elements, a formal model and a set of formal specifications. However, industrial models are typically non-formal, they are analyzed in non-formal simulation environments, and their specifications are described in non-formal natural language. In this thesis, we aim to facilitate the integration of formal verification into industrial model-based design.

Our first key contribution is a model transformation methodology. Starting with a standard simulation model, we transform it into an equivalent verification model, a network of hybrid automata. The transformation process addresses differences in syntax, semantics, and other aspects of modeling. For this class of formal models, so-called reachability algorithms can be applied to verify safety properties. An obstacle is that scalable algorithms exist for piecewise affine (PWA) models, but not for nonlinear ones. To obtain PWA over-approximations of nonlinear dynamics, we propose a compositional syntactic hybridization technique. The result is a highly compact model that retains the modular structure of the original simulation model and largely avoids an explosion in the number of partitions.

The second key contribution is an approach to encode rich formal specifications so that they can be interpreted by tools for reachability. Herein, we consider specifications expressed by *pattern templates* since they are close to natural language and can be easily understood by non-expert users. We provide (i) formal definitions for select patterns that respect the semantics of hybrid automata, and (ii) monitors which encode the properties as the reachability of an error state. By composing these monitors with the formal model under study, the properties can be checked by off-the-shelf fully automated verification tools.

Furthermore, we provide a semi-automated toolchain and present results from case studies conducted in collaboration with industrial partners.

Résumé

Les systèmes cyber-physiques sont une classe de systèmes complexes, de grande échelle, souvent critiques enlever de sûreté, qui apparaissent dans des applications industrielles variées. Des approches de vérification formelle sont capable de fournir des garanties pour la performance et la sûreté de ces systèmes. Elles nécessitent trois éléments: un modèle formel, une méthode de vérification, ainsi qu'un ensemble de spécifications formelles. En revanche, les modèles industriels sont typiquement informels, ambigus par nature. Ils sont analysés dans des environnements de simulation informels et leurs spécifications sont décrites dans un langage naturel informel. Dans cette thèse, nous visons à faciliter l'intégration de la vérification formelle dans le processus industriel de la conception basée sur des modèles.

Notre première contribution clé est une méthodologie de transformation de modèle. À partir d'un modèle de simulation standard, nous le transformons en un modèle de vérification équivalent, plus précisément en un réseau d'automates hybrides. Pour cette classe de modèle formel, des algorithmes de l'atteignabilité peuvent être appliqués pour vérifier des propriétés de sûreté. Le processus de transformation prend en compte les différences de syntaxe, de sémantique et d'autres aspects de la modélisation. L'un des obstacles rencontré est que des algorithmes d'atteignabilité passent à l'échelle pour des modèles affines par morceaux, mais pas pour des modèles non linéaires. Pour obtenir des surapproximations affines par morceaux des dynamiques non linéaires, nous proposons une technique compositionnelle d'hybridisation syntaxique. Le résultat est un modèle très compact qui retient la structure modulaire du modèle d'origine de simulation, tout en évitant une explosion du nombre de partitions.

La seconde contribution clé est une approche pour encoder des spécifications formelles riches de façon à ce qu'elles puissent être interprétées par des outils d'atteignabilité. Nous prenons en compte des spécifications exprimées sous forme d'un gabarit de motif (pattern template), puisqu'elles sont proches du langage naturel et peuvent être comprises facilement par des utilisateurs non

experts. Nous fournissons (i) des définitions formelles pour des motifs choisis, qui respectent la sémantique des automates hybrides, et (ii) des observateurs qui encodent les propriétés en tant qu'atteignabilité d'un état d'erreur. En composant ces observateurs avec le modèle formel, les propriétés peuvent être vérifiées par des outils standards de vérification qui sont automatisés.

Finalement, nous présentons une chaîne d'outils semi-automatisée ainsi que des études de cas menées en collaboration avec des partenaires industriels.

Contents

Abstract i

Résumé iii

1 Introduction 1
- 1.1 Context . . . 3
- 1.2 Contributions . . . 9
- 1.3 Outline . . . 11

2 Formal Verification using Reachability Analysis 13
- 2.1 Hybrid Automata . . . 14
- 2.2 Networks of Hybrid automata . . . 18
- 2.3 Set-Based Reachability Analysis . . . 19
- 2.4 Formal Verification Tools & SpaceEx . . . 20
- 2.5 Semi-Formal Verification Tools & Breach . . . 22
- 2.6 Temporal Logic . . . 23
- 2.7 Cyber-Physical Systems . . . 24

3 Bringing Formal Verification to Industrial Model-Based Design 29
- 3.1 Industrial Model-Based Design & Tools . . . 31
  - 3.1.1 Simulink . . . 32
  - 3.1.2 SCADE . . . 33
- 3.2 Tool Integration . . . 34

4 From Informal Requirements to Formal Specifications via Pattern Templates 41
- 4.1 Pattern Templates . . . 43
  - 4.1.1 Patterns occurring in Control Systems . . . 48
  - 4.1.2 Patterns found in Industrial Use Cases . . . 49
- 4.2 Formalizing Pattern Templates for Hybrid Automata . . . 51

4.2.1 Preliminaries . . . 51
4.2.2 Formal Definitions . . . 52
4.3 Verifying Pattern Templates using Monitor Automata . . . 56
4.4 Correctness of Monitor Automata . . . 61
4.4.1 Preliminaries . . . 61
4.4.2 Sufficient Conditions . . . 62
4.4.3 Necessary Conditions . . . 65
4.4.4 Buggy Monitors . . . 67
4.5 Application Example . . . 71
4.6 Related Work . . . 73

5 From Simulation Models to Formal Models 75
5.1 Compositional Syntactic Hybridization . . . 77
5.1.1 Syntactic PWA Approximation . . . 78
5.1.2 Compositional Scheme . . . 80
5.1.3 Algorithm for Compositional Syntactic Hybridization 81
5.2 From Simulink to Hybrid Automata . . . 87
5.2.1 Model Adaptation . . . 88
5.2.2 Estimation of the signal range . . . 89
5.2.3 Translation to SX format . . . 90
5.2.4 Hybridization . . . 91
5.2.5 Example . . . 91
5.3 From Stateflow Diagrams to Hybrid Automata . . . 94
5.3.1 Stateflow Semantics . . . 94
5.3.2 Translation Scheme . . . 96
5.3.3 Examples . . . 98
5.4 Urgent Semantics . . . 104
5.4.1 Reach Tubes under Invariant Constraints . . . 104
5.4.2 Examples . . . 107
5.5 Related Work . . . 112

6 Case Studies 115
6.1 Cruise Controller . . . 115
6.2 Wind Turbine . . . 120
6.2.1 Benchmark Model . . . 120
6.2.2 Model Transformation . . . 122
6.2.3 Reachability Results . . . 127
6.3 Lane Change Manoeuvre for Autonomous Vehicles . . . 129
6.3.1 System Description . . . 130
6.3.2 Simulation Results . . . 139

7 Conclusion **143**
7.1 Summary 143
7.2 Future work 145

Bibliography **147**

CHAPTER 1

Introduction

Model-Based Design (MBD), also known as Model-Based Development, is a paradigm that enables the cost-effective and quick development of complex systems, such as control and energy systems. MBD has facilitated the detection and correction of errors in the early design stages and has established a common framework for communication throughout the whole design process [237]. The design process begins with the construction of a high-level model of the system to be developed. Emphasis is placed on abstract, mathematical models which guide further development, simulation and testing. Afterwards, model transformation techniques are applied to transform abstract models into more concrete models accounting for low-level modeling details [185, 273, 276].

However, traditional test-based and simulation-based methods, such as simulation of corner cases or stochastic simulation, have inherent limitations and sometimes fail to detect bad behaviors [35]. Simulation is typically based on unverified numerical computation, links to numerical errors, and has limited precision. Also, simulation cannot deal with the intrinsic non-determinism present in complex systems and with partial or incomplete designs [64]. These matters render simulation prone to incomplete coverage of system behaviors as well as unsound analysis results. As such, existing errors in the design might not be discovered through simulation. If such incorrectly designed and insufficiently verified systems are deployed, there is a high risk of serious system failure [297].

This issue is particularly important within safety-critical fields as well as with the new generation of systems, embedded and cyber-physical. These

systems are complex with various interacting components and frequently have a safety-critical nature [259]. They are difficult to analyze, even numerically simulate, as neighboring states, no matter how close, may exhibit qualitatively different behaviors [127]. In addition, the presence of uncertainties, disturbances, noise may have adverse effects on the performance [111].

Formal verification techniques help to remedy the problems of simulation by formally establishing whether a system satisfies given specifications [297]. There exist various formal verification techniques in the literature [8, 91], and their goal is to decide whether specifications are satisfied or violated via a rigorous mathematical analysis. One of these techniques is reachability analysis and it constitutes an exhaustive verification procedure that replaces an infinite number of simulations. In particular, set-based reachability analysis computes the set of all behaviors of the system (exact or approximative), starting from all possible initial conditions, under all possible disturbances and variations in parameter values [31, 218]. In other words, set-based reachability analysis exhaustively computes a cover of all behaviors and, if precise enough, can show safety of the system as well as provide quantitative measurements of key variables [127].

Over the past years, there have been a lot of efforts to bridge the gap between formal methods and industrial applications. Applying verification tools to industrially relevant models requires three main elements: a formal verification method, a formal model, and a set of formal specifications. Every verification method is normally supported by a formal verification engine. The task of the end user is to input the formal model and the formal specification in the tool. Then, the tool employs the verification engine to find out whether the specifications are satisfied or not. In practice, however, this task is not simple since industries mainly operate on non-formal models and non-formal requirements/specifications. On the one hand, industrial requirements are written and described in natural language. On the other hand, industrial models are expressed and analyzed in non-formal simulation environments.

The goal of this thesis is to assist bridging this gap and facilitate the integration of formal verification into the industrial model-based design process. We employ hybrid automata as the formal model and reachability analysis as the formal method. Our first key contribution is a model transformation methodology. Starting with a standard simulation model, we transform it into an equivalent formal verification model addressing differences in syntax, semantics, and other aspects of modeling. The second key contribution is an approach to encode rich formal specifications so that they can be interpreted by tools for reachability. Starting with natural language requirements ex-

pressed by pattern templates, we construct formally correct monitors which can be directly composed with the formal model. Finally, we provide a semi-automated toolchain and present results from case studies.

1.1 Context

Over the past years, there have been a lot of efforts to bridge the gap between formal methods and industrial applications. Formal methods have evolved from an elegant theory to a vital practice. However, any pathway from theory to practice faces a lot of challenges. New technologies must compete against well established practice and demonstrate the need to replace the old with the new while overcoming the frequent concerns and doubts of engineers [152, 199].

The main focus has been on addressing the issues with the format mismatch and scale of industrially sized models. These challenges are of particular significance with the new generation of systems, embedded and cyber-physical, as they are complex with various interacting components and frequently have a safety-critical nature [259]. Cyber-physical systems (CPS) attempt to bridge the cyber-world of computing and communications with the physical world. In CPS, embedded computers along with networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa [207]. CPS have applications in various domains such as health care, transportation, automation, aerospace, autonomous vehicles, and robotics [258].

System Modeling. An appropriate modeling formalism for the design of such systems is hybrid systems [18]. Hybrid systems demonstrate joint discrete and continuous behaviors by combining the traditional models for discrete systems with classical differential and algebraic equation-based models for dynamical systems [19]. There is a wide range of formal models for hybrid systems, e.g. hybrid automata [20], petri nets [7, 136], hybrid programs [249], and process algebra [52]. These modeling languages have different advantages and serve different purposes. General purpose models should be abstractable, composable, and descriptive [214].

A popular modeling formalism is hybrid automata. Hybrid automata can exhibit nondeterministic behaviors. That means that any given state may lead to different futures. This is a useful feature as incomplete knowledge about initial conditions, parameters or other types of uncertainty can be easily captured within the hybrid automata formalism. In particular, rates of change or variable updates can be specified through bounds and not

necessarily fixed constant numbers. In addition, complex models can be easily constructed through the parallel composition of hybrid automata.The automata can interact with each other by sharing variables and synchronizing events. Hybrid automata are suitable for formal analysis and they form an analysis-friendly modeling formalism [111, 128].

In industrial practice, however, user-friendly modeling formalisms are preferred. The system modeling and control design tasks are typically based on a model, within a simulation environment like MATLAB/Simulink [227], Simplorer [116, 283], Ptolemy [75], or SCADE [2]. The standard technique for conducting system analysis and design validation is numerical simulation. Numerical simulation is a highly scalable technique but relies on a discrete-time approximation of the evolution of the system variables. As such, it is prone to incomplete coverage of open systems and possibly unsound results due to numerical error. It is especially difficult to simulate all representative behaviors of a system. In the case of hybrid systems, this issue can be critical as neighboring states can have significantly different and even chaotic behaviors. In this respect, it is hard to exhaustively test models and critical behaviors may go undetected [35, 298].

Formal Verification. The necessity to provide guarantees of correctness and performance has motivated the development of formal verification techniques and the design of robust verification tools. Their goal is to guarantee that specifications are satisfied through a rigorous mathematical analysis of the system. Various techniques for hybrid system verification have been developed. According to the model checking handbook [111], they could be broadly divided into 3 categories: approaches based on symbolic representations (e.g. reachability analysis), abstraction (e.g. bisimulations), or logic (e.g. theorem proving). Survey papers can be found in the literature, e.g. [18, 23, 111, 250, 282].

Recently, there has been an increased interest in fully automated verification tools for hybrid systems, such as set-based reachability analysis [207]. This is the case as they have demonstrated substantial success in finding bugs in real-world applications and there has been much progress towards efficient and scalable reachability algorithms [18].

Set-based reachability analysis aims at accurately and quickly computing a cover of the system states which can be reached from a given set of initial states. Constructing a cover of all system behaviors is done through the computation of one-step successor states. The computation is exhaustive and terminates if a fixed-point is found. In this way, reachability can guarantee that no critical behaviors are missed. As such, it has been used to formally

show safety and bounded liveness properties [111,128]. Set-based reachability analysis can be seen as a generalization of numerical simulation. In numerical simulation, one picks an initial state and tries to compute a successor state that lies on one of the solutions of the corresponding flow constraint and satisfies one of the jump conditions. Then, one of the successor states of the jump is picked and the process is repeated. Reachability analysis directly follows the transition semantics of hybrid systems, but considers sets of states instead of single states. The reachable set consists of all the states that can be visited by a trajectory of the hybrid system starting in specified initial states [111].

Scalability poses a major challenge in reachability analysis. In general, one-step successors can only be computed approximately and computational costs generally increase sharply with respect to the number of continuous variables. Special attention should be given to the approximation error as it can accumulate, leading to a coarse cover, prohibitive state explosion, or preventing termination. The computational complexity of one-step successors is dependent on the dynamics of the underlying system. Scalable approximations have been developed for certain types of dynamics, such as linear, piecewise constant and piecewise affine dynamics, but this performance comes at a price in accuracy. The trade-off between runtime and accuracy remains a central problem in reachability analysis. Surveys of reachability techniques for hybrid automata can be found in [31, 111, 218].

Specialized techniques exist for the reachability computation of nonlinear systems, which rely on linearization [30, 288], polynomial approximations [82, 263], or Bernstein expansions [98, 261]. A well established abstraction method is *hybridization*. In hybridization procedures, the state-space is partitioned into smaller domains and the nonlinear dynamics are approximated by simpler ones in each distinct domain. An important consideration is the abstraction error, which is typically accounted for via the use of nondeterministic inputs. Note that traditional hybridization schemes, e.g. [33], operate on the flattened model, which results to the number of partitions being exponential in the number of state variables.

Industrial Requirements. Despite the progress made on reachability algorithms of hybrid automata, it is not easy to formalize requirements of hybrid systems such that they can be verified automatically. The main reason concerns the semantic mismatch between industrial requirements and formal requirements. Typically, formal requirements are expressed in temporal logic [252], whereas industrial requirements are described in natural language or controlled natural language [198, 240]. A controlled natural language

(commonly abbreviated as CNL) forms a subset of natural language with a constrained grammar aiming to reduce or eliminate ambiguity. Requirements written in a CNL can be better understood across different disciplines (e.g. technical, legal, marketing).

CNL can be divided into two broad categories: human-oriented and machine-oriented languages [198]. Human-oriented languages, also called simplified or technical, provide general guidelines and restrict the writer/designer by rules such as "Keep sentences short", "Avoid the use of pronouns", and "Use only the active voice" [270]. They are used in the industry to increase the quality of of technical documentation. Among others, Basic English is used as an international auxiliary language for trade; Airbus Warning Language is employed for industrial warnings; and ASD Simplified Technical English for maintenance documentation. IBM has developed its own IBM's Easy English language [53] and Caterpillar has developed the Caterpillar Fundamental English [181]. Human-oriented languages are the most expressive languages and can be easily understood and used by humans. However, they do not have a formal basis and they pose significant translation challenges.

Machine-oriented languages have a formal logical basis, i.e. they have a formal syntax and semantics, and can be mapped to an existing formal language, such as first-order logic. These languages can be used as knowledge representation languages and are typically supported by automatic consistency and redundancy checks [270]. In particular, KANT (Knowledge-based, Accurate Natural-language Translation) is used for machine translation of technical documents; ACE (Attempto Controlled English), PENG (Processable English), and CNL for knowledge acquisition and representation; Rabbit and Lite Natural Language for semantic web; and ACE rules for rule and policy documentation [198, 270].

Several recent machine-oriented languages employ template based natural language specifications (TBNLS) [100], i.e. a grammar that is restricted to a set of simple templates. Such a grammar has been efficient in avoiding ambiguity [118]. For example, CPS Lite utilizes 113 sentence templates that include verb-like, noun-like, and preposition-like relations [270]. However, one pitfall that these languages share is the difficulty to timed relations and operations, restricting their applicability to simple properties.

One promising direction that is closely related to TBNLS and aims to handle this limitation concerns the use of the so-called pattern templates. Pattern templates (also called specification templates) contain predicates which should be substituted by logical expressions so as to encode the actual requirements. Dwyer et al. [114] introduced qualitative specification pattern

templates and translated them into different logic expressions (e.g. linear temporal logic). A milestone was the extension of these pattern templates to capture real-time properties, see [196]. Application of the patterns in the automotive industry can be found in [182, 254]. A generalization to probabilistic pattern templates was proposed in [153].

For discrete systems, there are tools that accept as an input CNL expressions (e.g. in the form of pattern templates) and automatically translate them into formal specifications. Examples of such tools are Stimulus [28], Embedded Specifier [74], AutoFocus3 [173], and SpeAR [57]. Pattern templates for hybrid systems do not differ from pattern templates already available. Yet, no existing tool can translate them into a formal representation that is applicable to hybrid systems and enables the verification of rich properties.

Formal Specifications. Selected pattern templates have been formally defined using temporal logic: linear temporal logic (LTL), computational temporal logic (CTL) and timed CTL (TCTL) in [34], metric temporal logic (MTL) in [44, 196], and probabilistic LTL (PLTL) in [153]. These definitions, however, do not immediately carry over to monitoring with hybrid automata. Note that the hybrid automata cannot distinguish events with the precision of a temporal logic because the intersection between the location invariants and location guards must be non-empty (see Chapter 4).

A formal alternative to temporal logic is the use of monitor automata (also known as observer automata). A monitor automaton has the same syntax and respects the same semantics as the system model. In this vein, it can be composed with the system model and fed into a reachability tool. Such monitor automaton encodes the requirements as the reachability of a designated error state (see [156] for one of the earliest works on monitor automata). In the context of formal verification of hybrid systems, monitor automata have been used for checking whether simple safety requirements are satisfied or violated [260]. On the contrary, for program and software verification purposes, the use of monitor automata has been significant [56]. There exist several tools that automate the composition process, e.g. BLAST [272] and ORION [96].

Verifying Simulink Models. MATLAB is a commercial tool for simulation and model-based design of dynamical and embedded systems. Simulink, integrated within MATLAB, is a data flow graphical programming language tool for modeling, simulating and analyzing multi-domain dynamic systems. Simulink is considered to be an industrial de-facto standard for building executable models of control systems [298]. Stateflow [280] is a Simulink

toolbox that is used to describe decision logic, hybrid systems, and scheduling. A Stateflow diagram can be a block of a bigger Simulink model interconnected with other blocks through signals.

The widespread use of MATLAB/Simulink has instigated research on the verification of Simulink models [24, 297]. One group of approaches can be classified as verification by simulation. Various tools have been developed which employ the original MATLAB/Simulink models and excite the inputs of the models to provide coverage of the system behaviors. Breach [107] tool performs approximate reachability analysis relying on sensitivity analysis. It has been used for monitoring properties and requirements of Simulink models. S-Taliro [27] tool conducts fast and efficient simulations, relies on gridding, and focuses on falsification of Simulink models. C2E2 [121] is another MATLAB/Simulink tool that generalizes simulation trajectories to bundles of trajectories. This is done by identifying an area around the simulated trajectory where all trajectories have similar behavior. Nevertheless, the verification by simulation approaches face a major obstacle, i.e. the initial states have to be sampled. Given that the number of samples is exponential in the number of state variables, these approaches are restricted to systems with low-dimensional initial states. In addition, such techniques can be used to verify bounded-time properties but not propertied defined over unbounded time.

Another direction concerns the translation of Simulink models into modeling languages that enable formal verification. A comprehensive survey can be found in [297]. There are available translators from a fragment of Simulink to Lustre [290], NuSMV model checker [228], and BIP [274]. However, all these tools solely apply discrete verification. Filipovikj et al. [122] provided a transformation scheme from Simulink to the modeling language of UPPAAL Statistical Model checker. UPPAAL is well-suited for timed automata but has some limitations in the support of hybrid automata, e.g. restricting their continuous parts to simple dynamics or applying the Euler integration method.

Zuiliani et al. [299] proposed a statistical model checking methodology that is applicable to Simulink/Stateflow (SLSF) models. Stanley Bak et al. introduced a translation process from SLSF to hybrid automata in [223]. Both papers focus entirely on Stateflow diagrams and necessitate the transformation of the original Simulink model into a single Stateflow model. This transformation is not easy and sometimes impossible for most large-scale Simulink models. Alur et al. [24] translate SLSF models into linear hybrid automata to improve simulation coverage, but they only consider deterministic models. Other translation schemes have been proposed in [42, 248, 264].

The translation of a Simulink/Stateflow model into a hybrid automaton can be undertaken by the tools HyLink [223] and GreAT [3]. These tools, however, do not allow hierarchical modeling and support a small subset of Simulink blocks. The translation of Simulink models into hybrid automata is also supported by SL2SX [231]. This translator supports a larger number of Simulink blocks and replaces the unsupported Simulink blocks with empty components. These blocks should be replaced by the user. However, Stateflow diagrams are not considered.

Note that all the aforementioned methodologies and tools run into a common pitfall. The modeling language of Simulink/Stateflow tool lacks a formal and rigorous definition of its semantics. In particular, Stateflow is a highly complex language and its semantics are described through examples on the MathWorks website without any formal definition [66]. In this respect, much research has been conducted on the formal analysis of SLSF diagrams [297]. The objective is to obtain (i) a convincing formal semantics; (ii) a faithful compilation that preserves the hierarchical structure of the Stateflow model; and (iii) fully automated analysis engine [66]. As for the interpretation and development of formal semantics for SL2SF, several semantic types have been proposed, such as denotational [158], operational [61, 159, 160], continuation-passing style [66], and communicating push-down automata based [287].

Another major obstacle encountered is that simulation models are deterministic[1] and employ transitions with *urgent* semantics (which must be taken as soon as possible, also known as *must* semantics), whereas verification models are nondeterministic and respect more general semantics (called *may* semantics). Urgent semantics are not covered by standard reachability algorithms, since the computation of the states reachable by time elapse is different. An urgent time elapse operator for piecewise constant dynamics is proposed in [229] and for piecewise affine dynamics in [230]. However, these techniques cannot handle high dimensional systems in a computationally efficient manner.

1.2 Contributions

The goal of this thesis is to facilitate the integration of formal verification techniques into model-based design employing hybrid automata as the formal model and reachability analysis as the formal method.

[1] That is the case since the simulator needs to be able to compute what happens in the next step.

Bringing formal verification to industrial practice. In this thesis, we propose a workflow that is based upon current industrial practice, provides incremental improvements and is able to demonstrate value without hampering existing methodologies. In particular, we propose a semi-automated toolchain that enables the formal verification of industrial-sized models employing well-established and recently designed tools. Our starting point is an industrial model, described in MATLAB/Simulink, and a set of industrial requirements, defined in natural language. The entire analysis process is semi-automated via a successive use of tools and translators. The end goal is to formally verify the correctness of the design against pre-specified requirements.

Patterns templates and monitors for verifying rich formal requirements of hybrid systems. We propose a semi–automated, template–based translation of industrial requirements into a formal representation (monitor automata) that enables the algorithmic verification of rich specifications. In particular, we consider specifications expressed in pattern templates which are predefined properties with placeholders for state predicates. Pattern templates can be readily understood by non-expert users as they are close to natural language. We provide (i) formal definitions for select patterns that respect the hybrid automata semantics and (ii) monitors which encode the properties as the reachability of an error state. By composing these monitors with the formal model under study, the property can be checked by off-the-shelf fully automated verification tools.

Compositional syntactic hybridization. Hybridization is an established abstraction method to obtain PWA approximations of nonlinear dynamics. However, existing hybridization procedures operate on the composed (flattened) system and the number of partitions is exponential in the number of variables. As such, even for small systems, this technique can quickly lead to intractably large models. To mitigate this problem, we propose a compositional syntactic hybridization technique. Our hybridization scheme decomposes the original dynamics and carries out the state-space partitioning and PWA approximation on the components. This results in a highly compact model that retains the modular structure of the original simulation model and largely avoids an explosion in the number of partitions.

Simulink translation to hybrid automata. Starting with a standard Simulink/Stateflow model, we transform it into an equivalent formal verification model, a network of hybrid automata. A major obstacle encountered is that simulation models are typically deterministic and employ transitions

with urgent semantics (which must be taken as soon as possible), whereas standard verification models are nondeterministic and respect more general semantics. Our transformation process tackles these modeling, syntactic, and semantic differences. The constructed verification model complies with the SX format, which is a formalism used by several reachability tools.

Handling urgent transitions in hybrid automata. Standard hybrid automata have nondeterministic transitions that respect *may* semantics. However, several physical systems and most simulation models respect *urgent* semantics, i.e. a transition must be taken as soon as the guard condition is satisfied. Urgent semantics are either not covered or cannot be treated efficiently by standard reachability algorithms, since the time elapse operator is different. We present a heuristic model transformation approach to bridge this gap. The urgent transitions are replaced by non-convex invariants, which are subsequently split into convex subsets. To prevent and mitigate the pitfalls of related work, i.e. avoiding over-approximation errors and excessive number of auxiliary locations, we partition the state-space only when required and employ the so-called Lie planes.

Case studies. We present results from three case studies conducted in collaboration with our industrial partners. The first case study concerns a wind turbine system modeled in MATLAB/Simulink. The Simulink model consists of more than 150 blocks and has several nonlinear functions. We employ the suggested model transformation and obtain an equivalent PWA hybrid automaton model. The second case study concerns a lane change maneuver of 4 autonomous vehicles. We model and analyze the system in MATLAB/Simulink. The third case study refers to a cruise controller. Starting from MATLAB/Simulink, we construct a formal model, expressed as a network of hybrid automata. We transform the design requirements into monitor automata and conduct reachability analysis with SpaceEx verification platform. Finally, we analyze the system with SCADE Hybrid.

1.3 Outline

The remainder of the thesis is organized as follows. Section 2 introduces the formalism of hybrid automata and set-based reachability analysis. In Section 3, we present the Moded-Based Design process and propose a semi-automated toolchain that enables the formal verification of industrial-sized models employing well-established and recently constructed tools. In Section 4, we facilitate the verification of hybrid systems against rich formal

requirements by employing pattern templates and monitor automata. Section 5 describes the steps that are required to obtain formal models from simulation models. We propose a methodology to efficient handle nonlinear dynamics and address semantic differences. Special focus is given to MATLAB/Simulink models which are very common in industrial practice. In Section 6, we describe and formally analyze three industrial case studies. Section 7 provides conclusive remarks and recommendation for future work.

CHAPTER 2

Formal Verification using Reachability Analysis

Formal verification is the process of checking whether a design satisfies specific requirements (properties) [90]. In formal verification, the designer first builds a model, with precise mathematical semantics, of the system under design, and then performs extensive analysis with respect to correctness requirements [18]. Formal verification takes place during the design and development process; before the system is up and running. It is often conducted on an abstract model of the system which abstracts away certain data and implementation detail [219].

In literature, there are various terminologies and taxonomies for formal verification techniques [8]. According to a recent verification handbook [91], they can be divided into model checking (e.g. symbolic, enumerative, SAT-based), program analysis (e.g. abstract interpretation, dataflow analysis), and deductive techniques (e.g. theorem provering [137]). Algorithmic verification typically corresponds to automated verification techniques, such as symbolic model checking [219] or regular model checking [112, 139]. Model checking is the exhaustive exploration of the state space of a system, typically to see if an error state is reachable. A key feature is that it can generate concrete counterexamples [90]. Specialized verification techniques and tools have been designed for software, hardware, real-time, probabilistic, hybrid, and distributed systems [8].

In this thesis, we focus on the formal verification of hybrid systems. Hybrid systems demonstrate joint discrete and continuous behaviors by com-

bining the traditional models for discrete systems with classical differential and algebraic equation-based models for dynamical systems [19]. They form an appropriate modeling formalism for the design of cyber-physical and embedded control systems [18]. Various techniques for hybrid system verification have been developed. They could be broadly divided into 3 categories [111]: approaches based on symbolic representations (e.g. reachability analysis), abstraction (e.g. bisimulations), or logic (e.g. theorem proving). There are approaches that lie between several categories like interval analysis [62, 232]. The combination of interval analysis and barrier functions is proposed in [104] and applied in a robotic system in [103]. Several survey papers related to formal verification can be found in the literature, e.g. [18, 23, 111, 250, 282].

Fully automated tools for hybrid systems have gained industrial interest as they can be applied to various read-world applications and haven been able to identify design errors [207]. Automated tools that rely upon set-based reachability analysis have been used in a wider range of of application domains, e.g. automotive control [9, 130], robotics [245], and electronic circuits [134].

2.1 Hybrid Automata

The literature on hybrid systems is very rich and spans different disciplines (see e.g. [148, 209, 213, 235, 250, 282, 294] and references therein). There is a wide range of formal models for hybrid systems, e.g. hybrid automata [20], petri nets [51, 136], hybrid programs [249], hybrid action systems [262], hybrid data-flow languages [46, 49, 50], linear hybrid automata [138] mixed logical dynamical systems [45], timed automata [26] and parametric automata [25, 86], linear complementarity systems [165], max-min-plus-scaling systems [164], and process algebra [52]. These modeling languages have different advantages and put more emphasis on different aspects, depending on the applications and problems they are designed to address. General purpose models should be abstractable, composable, and descriptive [214].

In this part, we concentrate on hybrid automata. A hybrid automation is a dynamical system that describes the evolution over time of the values of a set of discrete and continuous variables. More precisely, it is a formal model that combines finite state automata with continuously evolving variables. A hybrid automaton displays two types of state evolution: discrete transitions that occur instantaneously, and continuous flow that occur when time elapses. A hybrid automaton can also be seen as a finite-state automaton that contains continuous variables [260].

Before we formally define the modeling language of hybrid automata, we introduce some notation for describing real-valued variables.

Preliminaries. Given a set $X = \{x_1, \ldots, x_n\}$ of variables, a *valuation* is a function $v : X \to \mathbb{R}$. Let $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ and $X' = \{x'_1, \ldots, x'_n\}$. The *projection* of v to variables $Y \subseteq X$ is $v{\downarrow_Y} = \{x \to v(x) | x \in Y\}$. Let $V(X)$ denote the set of valuations over X. The *embedding* of a set $U \subseteq V(X)$ into variables $\bar{X} \supseteq X$ is the largest subset of $V(Y)$ whose projection is in U, written as $U|^{\bar{X}}$. Let $const_X(Y) = \{(v, v') | v, v' \in V(X), v{\downarrow_Y} = v'{\downarrow_Y}\}$.

Definition 2.1 (Hybrid automaton)**.** [19, 126, 166] A *hybrid automaton*

$$H = (\mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump})$$

consists of

- a finite set of *locations* $\mathsf{Loc} = \{\ell_1, \ldots, \ell_m\}$, also called discrete states or modes,
- a finite set of *synchronization labels* Lab, which is used to coordinate changes between different automata,
- a finite set of edges $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Lab} \times \mathsf{Loc}$, also called *transitions* that indicates which discrete state changes are possible using which label;
- a finite set of *variables* $X = \{x_1, \ldots, x_n\}$, partitioned into controlled variables Y and uncontrolled variables U; a *state* of H comprises a location ℓ and a numerical value for each variable, it is denoted by $s = (\ell, \mathbf{x})$;
- a set of states $\mathsf{Inv} \subseteq \mathsf{Loc} \times \mathbb{R}^X$ named *invariant conditions*; this set restricts for each location the values that x is allowed to take, also known as *staying conditions* since it dictates how long the system can stay in this location;
- a set of *initial* states $\mathsf{Init} \subseteq \mathsf{Inv}$; every behavior of H has to start in one of these initial states;
- a *flow relation* Flow, where $\mathsf{Flow}(\ell) \subseteq \mathbb{R}^{\dot{X}} \times \mathbb{R}^X$ defines for each state $(\ell, \mathbf{x})$ the set of possible derivatives $\dot{\mathbf{x}}$, for example with a differential equation of the form

  $$\dot{\mathbf{x}} = f(\mathbf{x});$$

  For a location ℓ, a *trajectory* has duration $\delta \geq 0$ and is a continuously differentiable function $\xi : [0, \delta] \to \mathbb{R}^X$ so that for all $t \in [0, \delta]$, $(\dot{\xi}(t), \xi(t)) \in \mathsf{Flow}(\ell)$. The trajectory *satisfies the invariant* if $\forall t \in [0, \delta]$, it is true that $\xi(t) \in \mathsf{Inv}(\ell)$.

– a *jump relation* Jump, where $\mathsf{Jump}(e) \subseteq \mathbb{R}^X \times \mathbb{R}^{X'}$ defines for every transition $e \in \mathsf{Edg}$ the possible successors $\mathbf{x}'$ of $\mathbf{x}$; jump relations are usually described by a *guard* set $\mathcal{G} \subseteq \mathbb{R}^X$ and an *assignment* (also called *reset*) $\mathbf{x}' = r(\mathbf{x})$ as follows

$$\mathsf{Jump}(e) = \{(\mathbf{x}, \mathbf{x}') \mid \mathbf{x} \in \mathcal{G} \wedge \mathbf{x}' = r(\mathbf{x})\}.$$

A jump can be defined as *urgent*, which pinpoints that time cannot pass when the state is inside this guard set.

The behavior of a hybrid automaton is defined with a *run*. Starting from some initial states, the state progresses according to the differential equations while time elapses, and according to the jump relations while taking a transition. The transition is considered to be instantaneous. Special events, which we name *uncontrolled assignments*, model an environment that can arbitrarily make changes to the uncontrolled variables.

Definition 2.2 (Run semantics)**.** An *execution* of a hybrid automaton H is a sequence

$$(\ell_0, \mathbf{x}_0) \xrightarrow{\delta_0, \xi_0} (\ell_0, \xi_0(\delta_0)) \xrightarrow{\alpha_0} (\ell_1, \mathbf{x}_1) \xrightarrow{\delta_1, \xi_1} (\ell_1, \xi_1(\delta_1)) \ldots \xrightarrow{\alpha_{N-1}} (\ell_N, \mathbf{x}_N),$$

with $\alpha_i \in \mathsf{Lab} \cup \{\tau\}$, satisfying for $i = 0, \ldots, N-1$:

1. *Trajectories:* in location ℓ_i, the trajectory ξ_i satisfies the invariant and has duration δ_i with $\xi_i(0) = \mathbf{x}_i$. Note that duration δ_i is 0 indicates that the trajectory goes through urgent guard sets.
2. *Jumps:* in case $\alpha_i \in \mathsf{Lab}$, there is a transition $(\ell_i, \alpha_i, \ell_{i+1}) \in \mathsf{Edg}$ with jump relation $\mathsf{Jump}(e)$ such that $(\xi_i(\delta_i), \mathbf{x}_{i+1}) \in \mathsf{Jump}(e)$ and $\mathbf{x}_{i+1} \in \mathsf{Inv}(\ell_{i+1})$.
3. *Uncontrolled assignments:* if $\alpha_i = \tau$, then $\ell_i = \ell_{i+1}$ and $\xi_i(\delta_i) \downarrow_Y = \mathbf{x}_{i+1} \downarrow_Y$. As mentioned before, these assignments correspond to arbitrary changes that the environment can carry out on the uncontrolled variables $U = X \setminus Y$.

A *run* of H is an execution which starts in one of the initial states, i.e., $(\ell_0, \mathbf{x}_0) \in \mathsf{Init}$. A state $(\ell, \mathbf{x})$ is *reachable* only if there exists a run with $(\ell_i, \mathbf{x}_i) = (\ell, \mathbf{x})$ for some i.

Definition 2.2 prescribes an alternation of trajectories and jumps. However, a strict alternation is not necessary. Two successive trajectories can be

expressed by inserting an uncontrolled assignment jump that does not modify the variables, while two successive jumps can be expressed by inserting a trajectory with duration zero (which always exists), and Note that we opt for an event at the end of the run as it will simplify the notation in the remainder of the Thesis.

May and Must semantics. In Definition 2.2, transitions may be taken when they are enabled but it is not necessary to do so so. The system may remain in a location provided that the invariant is satisfied. These so-called *may* semantics enable the introduction of nondeterminism in the system regarding when a transition will be taken. For the case of *must* semantics (also called urgent or ASAP semantics), the transition must be taken as soon as possible [230]. Must semantics are implemented in simulators such as Simulink [227], Modelica [116], MapleSim [225], since they require deterministic models.

Deadlock Situations. The concept of deadlock has been studied in the computer science literature [36]. Deadlock is typically considered as a pathology and refers to situations where no forward progress can be made. In the discrete community, the focus is on verifying the presence of deadlock situations in programs and ensuring its absence upon their composition. In hybrid systems, deadlock can take one of the following forms [1].

- blocking conditions: states which have no defined "next" state.
- stable equilibria in finite time: equilibria in the continuous dynamics that are reached by a reset operation.
- chattering Zeno: there is switching between two or more discrete locations infinite often, while the continuous variable remain unchanged,
- genuine Zeno: the hybrid trajectory performs an infinite number of transitions in a finite amount of time.

Chattering behaviors are a common issue in hybrid systems and control often rendering simulation results unreliable or incomplete [4]. An efficient approach for robust simulation of hybrid systems with chattering behaviors is proposed in [6] and related work proposed in the literature can be found in [5]. A common issue/error that arises in modeling hybrid systems, commonly causing unintended deadlocks, relates to the hybrid automata semantics. In particular, incorrect models are created as several users are unaware that the intersection between the location invariants and location guards must be non-empty [1]. Let p be a state predicate and $!p$ its negation. Then, adding

p (e.g. $x > 0$) in the invariant of the source location and $!p$ ($x \leq 0$) in the invariant of the target location leads to a deadlock because of the empty intersection. Figure 2.1 illustrates this modeling error.

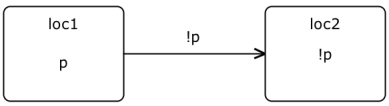


Figure 2.1: Common modeling error in hybrid systems that leads to deadlock (clipped from SpaceEx Model Editor).

2.2 Networks of Hybrid automata

In this Section, we provide a formal definition of the composition operation of two hybrid automata. This operation is later used to couple the system model with a monitor automaton. The operator is analogous to the composition operator in [22].

The jump relations of synchronized transitions rise from the conjunction of the involved transitions. There might be transitions that do not synchronize and can change variables arbitrarily. As for the variables over which their jump relation is not defined, they are are set to remain constant [109].

Definition 2.3 (Composition of HA)**.** The *parallel composition* of hybrid automata H_1 and H_2 generates the hybrid automaton $H = H_1||H_2$

- $Loc = Loc_1 \times Loc_2$,
- $Lab = Lab_1 \cup Lab_2$,
- $Edg = \{((\ell_1, \ell_2), \alpha, (\ell_1', \ell_2')) \mid (\alpha \in Lab_1 \Rightarrow (\ell_1, \alpha, \ell_1')) \wedge (\alpha \in Lab_2 \Rightarrow (\ell_2, \alpha, \ell_2'))\}$,
- $X = X_1 = X_2$ (by assumption), $Y = Y_1 \cup Y_2$, $U = (U_1 \cup U_2) \setminus Y$,
- $Jump((\ell_1, \ell_2), a, (\ell_1', \ell_2'))$ with $\mu = \{(v, v') \in \mu_i\}$ iff for $i = 1, 2$,
  - $a \in Lab_i$ and $(\ell_i, a_i, \mu_i, \ell_i') \in Edg_i$, or
  - $a \notin Lab_i$, $\ell_i' = \ell_i$, and $\mu_i = const_{X_i}(Z_i)$, where $Z_1 = Y_1 \setminus Y_2$ and $Z_2 = Y_2 \setminus Y_1$;
- $Flow(\ell_1, \ell_2) = Flow_1(\ell_1) \cap Flow_2(\ell_2)$;
- $Inv(\ell_1, \ell_2) = Inv_1(\ell_1) \cap Inv_2(\ell_2)$;
- $Init(\ell_1, \ell_2) = Init_1(\ell_1) \cap Init_2(\ell_2)$.

We assume that H_1 and H_2 have the same variables without any loss of generality. If H_2 has a variable not in H_1, e.g., a clock variable that measures the time in between events, we can add it to H_1 without having to restrict it in the invariants, guards, or flows.

A run $r_{H_1||H_2}$ in $H_1||H_2$ is given by locations $\ell_i = (\ell_i^{H_1}, \ell_i^{H_2})$, continuous states $\mathbf{x}_i$, trajectories ξ_i, durations δ_i, and labels α_i. Let r_{H_1} be the projection of the run onto H_1, obtained by replacing ℓ_i with $\ell_i^{H_1}$, and let r_{H_2} be the projection of the run onto H_2, obtained by replacing ℓ_i with $\ell_i^{H_2}$ and α_i with τ. Then by definition, we have that for any run $r_{H_1||H_2}$ in $\text{Runs}(H_1||H_2)$, $r_{H_1} \in \text{Runs}(H_1)$ and $r_{H_2} \in \text{Runs}(H_2)$.

2.3 Set-Based Reachability Analysis

Reachability analysis is concerned with the problem of computing the set of discrete and continuous states that a system can reach, making it possible to verify if a state can avoid a set of unsafe states. There are two kinds of events that can take place in a hybrid automaton: time can pass with the state evolving according to the flow constraints, or a jump can take the system instantaneously to a new state. Starting from the initial states, set-based reachability analysis exhaustively computes the successor states for both time elapse and jumps in alternation until this no longer produces any new states. However, this iterative process might not terminate. In [166], it was shown that the safety verification problem is undecidable for rectangular hybrid automata (and therefore also for linear and affine hybrid automata). As such, in practice, an a priori limit on the search depth (typically counted in the number of discrete transitions) is imposed [111].

Set-based reachability analysis can be seen as a generalization of numerical simulation. In numerical simulation, one picks an initial state and tries to compute a successor state that lies on one of the solutions of the corresponding flow constraint and satisfies one of the jump conditions. Then, one of the successor states of the jump is picked and the process is repeated. Reachability analysis directly follows the transition semantics of hybrid automata, but considers sets of states instead of single states [111].

The reachable set contains all the states that can be visited by a trajectory of the hybrid system starting in specified initial states. Reachability analysis is often motivated by safety verification; checking if the intersection of the computed reachable set with a set of error (forbidden) states is empty. In case the reachable set of a hybrid system cannot be computed exactly, we try to compute an over-approximation so that if it does not intersect the set

of error states, the hybrid system is guaranteed to be safe [31]. Figure 2.2 illustrates the reachability computations on a helicopter model.

Computational costs generally increase sharply with respect to the number of continuous variables. Scalable approximations are available for certain types of dynamics, such as PWA dynamics, but this performance comes at a price in accuracy. The trade-off between runtime and accuracy remains a central problem in reachability analysis. Surveys of reachability techniques for hybrid automata can be found in [31, 111, 218].

Specialized techniques exist for the reachability computation of nonlinear systems, which rely on linearization [30, 288], polynomial approximations [82, 263], or Bernstein expansions [98, 261]. A well established abstraction method is *hybridization*. Hybridization [33] consists in partitioning the state-space into smaller domains and approximating the nonlinear dynamics by simpler ones in each domain. Nondeterministic inputs are added to account for the abstraction error.

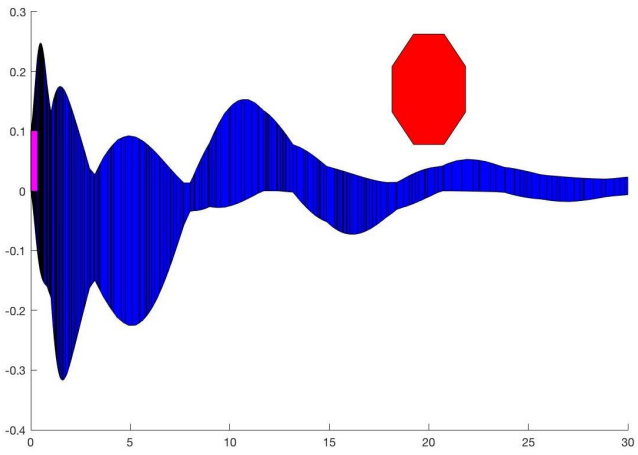


Figure 2.2: Safety verification of a 20-dimensional helicopter model [135]. Conducted reachability analysis with SpaceEx tool over a time horizon of 30 seconds. Blue: reachable sets, red: unsafe area, magenta: initial set.

2.4 Formal Verification Tools & SpaceEx

A variety of tools for formal verification of hybrid systems exists in the literature [85, 179]. In the ARCH workshop, friendly competitions are organized where tools compete against each other on a set of benchmark models [15]. Some prominent tools are Ariadne [93], C2E2 [113], CORA [10], dReal [195],

Flow* [83], HyCreate [37], HyPro [268], HyReach [217], JuliaReach [59], KeYmaera [251], SoapBox [155], SpaceEx [135].

In this Section, we are going to focus on SpaceEx [135] which is is a verification platform for hybrid systems. The name SpaceEx stands for State Space Explorer. SpaceEx and the principal functionality of the tool is to verify that a certain mathematical model of a hybrid system satisfies desired safety properties. SpaceEx operates on symbolic states consisting of a discrete location and a continuous region. Its reachability algorithm can be represented as a fixed point computation: $S_0 := \text{post}_c(Init)$, $S_{i+1} := S_i \cup \text{post}_c(\text{post}_d(S_i))$, where $Init$ refers to the initial states of the hybrid automaton, $\text{post}_c(S)$ is the continuous post operator, and $\text{post}_d(S)$ is the discrete post operator. Note that SpaceEx performs full fixed point computations operating on symbolic states; it should not be confused with the least fixed point often used in program analysis, see e.g. [140].

SpaceEx offers three options for set-based reachability analysis, i.e. the PHAVer, the LGG and the STC, and one simulation option. The PHAVer (Polyhedral Hybrid Automaton Verifyer) scenario is well-suited to linear hybrid automata, i.e. hybrid systems with piecewise constant bounds on the derivatives, and produces exact reachability results. The LGG Support Function scenario implements an adaptation of the Le Guernic-Girard (LGG) algorithm [203]. The STC (Space-Time approximation with Clustering) scenario enhances the LGG algorithm by computing fewer convex sets for a given accuracy. This scenario yields more precise results for discrete transitions.

These algorithms compute an over-approximation of the reachable sets and can be applied to high-dimensional hybrid systems with piecewise affine dynamics and nondeterministic inputs [127, 129, 131]. SpaceEx combines explicit set representations (polyhedra), implicit set representations (support functions) and linear programming to achieve scalability while maintaining high accuracy. SpaceEx engine computes an over-approximation of the reachable states and expresses them as template polyhedra. Template polyhedra are polyhedra whose facets are oriented according the so-called template directions; these directions can be provided by the user. A cover of the continuous evolution is obtained by time-discretization which allows adaptive time-steps. The algorithm guarantees that the approximation error in each template direction stays below a given value [11]. Very recently, zonotope-based reachability algorithms for nonlinear systems from CORA [10] have been implemented inside SpaceEx [14]. SpaceEx is available at `http://spaceex.imag.fr`.

SpaceEx supports hybrid automata models and provides hierarchy, templates, and instantiations of components. SpaceEx models respect the SX modeling language [94] and consist of *base compoments* (single hybrid automata) and *network components* (networks of hybrid automata). Note that the composition of multiple hybrid automata is supported by SpaceEx and it is automatically conducted by the tool, once the model is specified. The design of the formal model is facilitated with the use of SpaceEx Model Editor (MO.E.), a graphical editor for constructing models of complex hybrid systems out of nested components. The user can determine various options to tailor the reachability analysis according to his needs. These options involve analysis algorithms, time horizon, number of iterations, output variables, forbidden states, etc. An informal introduction to SpaceEx and its modeling paradigm is available at [125].

2.5 Semi-Formal Verification Tools & Breach

Model-based analysis and design techniques for complex systems with uncertainty rely mostly on extensive simulation. Efficient techniques and tools exist for verification of hybrid systems with linear continuous dynamics but no tool can be readily scalable for hybrid non-linear dynamics. An alternative to formal verification is to use semi-formal verification techniques.

Semi-formal verification aims to combine formal verification with simulation taking advantage of the benefits of both approaches. On the one hand, formal verification provides complete and sound results, and on the other hand, simulation can be efficient, scalable and easy-to-use. In this way, it is possible leverage formal techniques in a resource-efficient way to identify critical “deep” bugs. Typically, semi-formal techniques augment or guide simulation using formal techniques [102].

In hybrid and control systems community, semi-formal verification typically refers to simulation-based methods, e.g. [97,184]. Their goal is to obtain the same effect as reachability computation by finitely many simulations, not necessarily of extremal points. Simulation-based are useful for nonlinear dynamical systems, black or grey box models as well as programs that do not come with an explicit mathematical model [218].

Breach [107] is a toolbox for simulation-based verification and parameter synthesis of hybrid systems. It is a MATLAB/Simulink based tool that performs approximate reachability analysis and relies on sensitivity analysis. Its primary feature is to conduct efficient signal monitoring of properties and requirements via the robust monitoring of metric interval temporal logic formulas.

2.6 Temporal Logic

Temporal Logic (TL) is a popular formalism, introduced into systems design [252] to specify acceptable behaviors of reactive systems. Traditionally, TL has been used for formal verification, either by deductive methods [224], or algorithmic methods [90, 256]. In this setting, the behaviors under consideration are discrete, i.e. sequences of states or events [108]. Signal Temporal Logic (STL) was proposed in [221,222] as a high-level declarative language for continuous and hybrid systems. In this section, we recall the STL framework of [221] as summarized in [108].

Preliminaries. The set of Boolean values is defined as $\mathbb{B} := \{\bot, \top\}$, with $\bot < \top$, $-\top = \bot$ and $-\bot = \top$. Let $\overline{\mathbb{R}} := \mathbb{R} \cup \mathbb{B}$ be the totally ordered set of real numbers with $\top$ the greatest element and with $\bot$ being the smallest element. A signal is a function $D \to E$ where D is an interval of $\mathbb{R}^+$ and $E \subset \overline{\mathbb{R}}$. Signals with $E = \mathbb{R}$ are real-valued signals, whereas signals with $E = \mathbb{B}$ are Boolean signals. A trace w is a set of real-valued signals $\{x_1^w, \cdots, x_k^w\}$ defined over an interval D of $\mathbb{R}^+$. This interval is called the time domain of w. Note that a trace can be "booleanized" via a set of threshold predicates of the form $x_i \geq 0$. STL can be seen an extension of Metric Temporal Logic where real-valued variables $(x_i)_{i\in\mathbb{N}}$ are transformed into Boolean values through these predicates. The syntax of STL is given by:

$$\varphi := \text{true} \,|\, x_i \geq 0 \,|\, \neg\varphi \,|\, \varphi \wedge \varphi \,|\, \varphi \mathcal{U}_I \varphi$$

with x_i being the variables and I a non-singular interval of $\mathbb{R}^+$. This interval defines both bounded intervals $[a, b]$ and unbounded intervals $[a, +\infty)$ for any $0 \leq a < b$. Let w be a trace over a time domain D. The formula φ is defined over a time interval $\text{dom}(\varphi, w)$ and respects the following rules:

$$\begin{aligned}
&\text{dom}(\text{true}, w) = \text{dom}(x_i \geq 0, w) = D,\\
&\text{dom}(\neg\varphi, w) = \text{dom}(\varphi, w),\\
&\text{dom}(\varphi \wedge \psi, w) = \text{dom}(\varphi, w) \cap \text{dom}(\psi, w),\\
&\text{dom}(\varphi \mathcal{U}_I \psi, w) = \{t \in \mathbb{R}^+ \,|\, t, t{+}\inf(I) \in \text{dom}(\varphi, w), \text{ and}\\
&\qquad\qquad t{+}\inf(I) \in \text{dom}(\psi, w)\}.
\end{aligned}$$

Boolean Semantics. For a given trace w, the validity of an STL formula φ at a given time $t \in \text{dom}(\varphi, w)$ is determined according to the following inductive definition [221].

$$
\begin{array}{ll}
w,t \models \text{true} & \\
w,t \models x_i \geq 0 & \text{iff } x_i^w(t) \geq 0 \\
w,t \models \neg\varphi & \text{iff} w,t \not\models \varphi \\
w,t \models \varphi \wedge \psi & \text{iff } w,t \models \varphi \text{ and } w,t \models \psi \\
w,t \models \varphi \mathcal{U}_I \psi & \text{iff } \exists t' \in t+I \text{ s.t } w,t' \models \psi \text{ and } \forall t'' \in [t,t'], w,t'' \models \varphi
\end{array}
$$

Some common operations can be redefined in a straightforward way:

$$
\begin{array}{cc}
\text{false} := \neg \text{ true} & \varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi) \\
\Diamond_I \varphi := \text{true}\, \mathcal{U}_I\, \varphi & \Box_I \varphi := \neg\Diamond_I \neg\varphi
\end{array}
$$

We use the shorthand notation $\Diamond$, $\Box$, and $\mathcal{U}$ to describe the untimed operators $\Diamond_{[0,+\infty)}$, $\Box_{[0,+\infty)}$ and $\mathcal{U}_{[0,+\infty)}$. For a formula φ and a trace w, we define the satisfaction signal $\chi(\varphi, w, .)$ as follows:

$$
\text{for all } t \in \text{dom}(\varphi, w), \qquad \chi(\varphi, w, t) = \begin{cases} \top \text{ iff } w,t \models \varphi \\ \bot \text{ otherwise} \end{cases}
$$

The monitoring of the satisfaction of φ can be accomplished by computing the entire satisfaction signal $\chi(\psi, w, .)$ for each subformula ψ of φ. This procedure is recursive on the structure of the formula, and moves from the atomic predicates up to the top formula [221].

2.7 Cyber-Physical Systems

Background. The term Cyber-Physical Systems[1] (commonly abbreviated as CPS) emerged around 2006. Over the past few years, this term has rapidly gained popularity. Its use has been evident both in the academic world and in non-technical context. If google trends can form any reliable basis to draw meaningful conclusions, it can be observed[2] that more and more people have been searching for CPS. Figure 2.3 portrays what Google thinks of the popularity of the Cyber-Physical Systems query over time. Even if these data can be considered circumstantial, there has been indeed an increasing interest in CPS.

As for the origin of the term, it was coined in 2006 by Helen Gill at the National Science Foundation in the United States [146]. Etymologically, the term combines the words cyber and physical. Physical stems from the Ancient Greek word $\phi\acute{\upsilon}\sigma\iota\varsigma$ (physis) which means nature. The word cyber is known to be an abbreviation of the word cybernetics, which is derived from

[1] Regarding the debate whether a dash is required in the term, google trends show that people mainly search for cyber physical systems or cyber-physical system.

[2] The google trends can be found at https://tinyurl.com/y8dvsrvy.

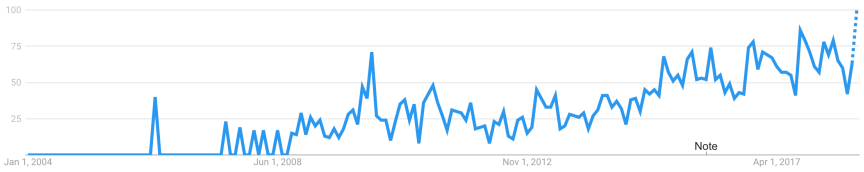


Figure 2.3: Plotting the number Google Search queries of the term "cyber physical system" over time (courtesy of google trends). Data correspond to the period from January 2014 till now.

the Greek word $\kappa\upsilon\beta\epsilon\rho\nu\acute{\eta}\tau\eta\varsigma$ (kybernetes) meaning governor, helmsman or captain. The word cybernetics was first used by Plato in *The First Alcibiades* to signify the governance of people. In the modern era, the French word "cybernétique" was coined in 1834 by the physicist André-Marie Ampère (1775–1836) to denote the sciences of government in his classification system of human knowledge [178].

The English word cybernetics first appeared in Norbert Wiener's book "Cybernetics: Or Control and Communication in the Animal and the Machine" in 1948. A second version of this book was published in 1961 [296]. Gibson theories and findings had great resemblance to contemporary cybernetics by interleaving the fields of control systems, electrical network theory, mechanical engineering, logic modeling, evolutionary biology and neuroscience.

In particular, his work on the automatic aiming and firing of anti-aircraft guns, eventually led Wiener to formulate cybernetics. He investigated the conjunction of information theory and automatic control constructing mechanisms that included analog circuits, mechanical parts and notions of feedback control logic. Albeit he did not employ digital computers, he readily demonstrated the need for unifying physical processes, computation and communication [206].

Following the prior historical and etymological discussion, it is plausible that cyber-physical systems refer to the integration of computation with physical processes [207]. More precisely, CPS bridge the cyber-world of computations and communications with the physical world. In CPS, embedded computers along with networks monitor and control the physical processes, typically with feedback loops where physical processes affect computations and vice versa. In this respect, the design of such systems requires comprehending the joint underlying mechanisms of computers, software, networks, and physical processes.

In [259], the authors define CPS as "physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core".

Opportunities & Applications. Examples of CPS include aerospace systems, transportation vehicles and intelligent highways, agricultural devices, defense systems, biomedical and healthcare systems, process control, factory automation, building and environmental control, robotic systems. CPS interact with the physical world, and must operate dependably, safely, securely, and efficiently and in real-time [258].

The advent of CPS has been facilitated by recent technological achievements and trends, such as low-cost and high performance sensors, nanotechnology breakthroughs, wireless sensor networks, alternative energy sources, and energy efficient computing devices. The industrial interest in CPS technologies has also increased, especially by vendors whose goal is to build large-scale safety-critical CPS correctly, affordably, flexibly and on schedule [259]. Some of the most promising applications for CPS research are as follows [192, 208, 259].

– *"Near-zero automotive traffic fatalities, minimal injuries, and greatly reduced traffic congestion and delays."*

  The National Highway Traffic Safety Administration reports that "on an annual basis, there are more than 5 million car accidents and 2 million injuries or fatalities" in USA [78]. The total number of vehicles worldwide has been increasing over the past decades and the transportation infrastructure has reached its maximum capacity giving rise to congestion and delays on roads. These problems have motivated research on CPS such as automated vehicles, intelligent intersection systems, wireless communication systems for vehicle-to-vehicle and vehicle-to-infrastructure [192].

– *"Sustainable and blackout-free electricity generation and distribution."*

  In the United States, approximately 70% of electricity is generated from fossil fuels causing more than 40% of greenhouse gas emission globally [29]. This problem could be detrimental in the future since demand for electricity is estimated to grow by 75% before 2030. Also, 50% of these power plants are 40 years old making the entire power grid less reliable [35]. These challenges have stimulated CPS research in monitoring, distributed sensing and control of power systems [192].

– *"Smart, reliable, and flexible medical and healthcare systems."*

  One significant challenge is to find ways to provide cost-efficient and effective medical and health care services to the elderly and the disabled people [275]. CPS research areas involve smart sensor systems for real-time patient health condition monitoring, telemedicine systems

which enable remote healthcare service provision, semi-autonomous tele-operated service robots which can assist with patient physical activities. In addition, CPS research could mitigate the large number of medical errors that lead to fatalities (98,000 per year only in USA [246]). Some common medical errors involve surgical errors, diagnostic errors, medication errors. A significant portion of those errors can be resolved through CPS technologies [192].

Research Challenges. Cyber-Physical Systems (CPS) are integrations of computation with physical processes. Embedded computers and networks monitor and control the physical processes, typically with feedback loops where physical processes affect computations and vice versa. In the physical world, a central property of a system is its dynamics, the evolution of its state over time. In the cyber world, dynamics are reduced to sequences of state changes without temporal semantics. In this vein, CPS is about the intersection, not the union, of the physical and the cyber world studying the joint dynamics of physical processes, software, and networks [205].

A new scientific foundation that addresses the interaction between the physical and cyber subsystems is vital. It should provide the grounds for an overall understanding of the design, development, certification, and deployment of cyber-physical systems. It has to integrate the theories of computing and communication systems, sensing and control of physical systems, as well as the interaction between humans and CPS. The science of CPS composition must discover new architecture patterns, hierarchical system composition from components and subsystems, protocol composition, and new modeling languages and tools to specify, analyze, design and simulate different compositions [259].

A key issue is that the formal models that are used for described physical dynamics, e.g. difference and differential equations, do not model well the behavior of software and networks. On the other hand, prevailing cyber abstractions, e.g. state machines, functions, and objects, generally lack temporal dynamics. For this reason, the modeling, design, and analysis of CPS requires effective theory and tools to reason about hybrid systems [20, 220] that combine discrete and continuous dynamics.

Note that the models that include differential equations, imperative programs, synchronous digital logic are typically deterministic. Cyber-physical systems, however, combine these models in such a way that determinism is not preserved. In this respect, nondeterministic modeling formalisms are required to capture the nature of CPS. Nondeterministic models are

suitable for systematically handling unknown or unknowable events and behaviors [206].

In the following, we mention several crucial research challenges for CPS. For detailed descriptions, see [35, 204, 206, 259, 273].

- *Safety, Security, and Robustness of CPS.* The large amount of uncertainties in the system, the environment, security attacks, and errors in physical devices renders the system robustness, security, and safety a crucial challenge [259].
- *Model-based Development of CPS.* Models are used today to generate and test software implementations of control logic. Communications, computing and physical dynamics must be abstracted and modeled at different scales, vicinities, and time granularity [259].
- *Verification, Validation, and Certification of CPS.* The gap between formal methods and testing needs to be bridged. Compositional verification and testing methods that explore the heterogeneous nature of CPS models are essential. V&V must also be incorporated into certification regimes [259].
- *Computational Abstractions and Composition.* Physical properties such as the laws of physics and chemistry, safety, real-time and power constraints, resources, robustness, and security characteristics should be captured in a composable manner by programming abstractions [259]. Ideally, heterogeneous systems to be composed in plug-and-play fashion [35].

CHAPTER 3

Bringing Formal Verification to Industrial Model-Based Design

Technology transfer, also called transfer of technology, is the process by which scientific research and new discoveries are developed into practical and commercially relevant applications and products [68]. Typically, the technology is transferred or disseminated from its original creator to a place with a wider distribution of people or resources; it occurs among universities, from universities to businesses, from large businesses to smaller ones, from governments to businesses, across borders, both formally and informally [151].

In computer science, the transfer may refer to abstract technologies, such as algorithms and data structures, or concrete such as open source software packages. The need to provide a tool-oriented link between academic research and industrial practice has motivated the foundation of specialized societies, such as the International Journal on Software Tools for Technology Transfer. This journal provides a forum discussing all aspects of tools that aid in the development of computer systems [279].

Formal verification is considered to be one of the most prominent applications of logic to computer science and computer engineering. The formal verification community has achieved many breakthroughs, bridging the gap between theoretical computer science and hardware and software engineering. Formal verification has been utilized in the hardware industry and it has been successfully applied to various software types. Recently, it is reaching out to new challenging areas such as system biology and hybrid systems [152].

Formal verification of hybrid systems has evolved from an elegant theory to a vital practice. However, any pathway from theory to practice faces a lot of challenges. New technologies must compete against well established practice and demonstrate the need to replace the old with the new while overcoming the frequent concerns and skepticism of engineers. A classic example concerns the breakthrough in linear optimization that offered a polynomial algorithm in the place of an exponential one. In fact, the new algorithm had a hard time showing that it was indeed faster in practice. Implementations of the exponential algorithm had been so refined that often the polynomial algorithm was not faster on the size of applications that mattered [200]. There are several other interesting examples; the interested reader could also read about the so-called galactic algorithms[1]. The bottomline is that the new, better way has to compete against years of engineering and also show that it is better in enough important cases to justify the trouble of replacing the old with the new. Another obstacle arises that there is typically no way to demonstrate that the new way is actually better without fully replacing the old way in some context. Kurshan provides a stimulating read on his personal experience on verification technology transfer in [199].

The lessons learned from the transfer of of formal verification technology in the hardware industry could benefit the hybrid and cyber-physical systems community. Technology transfer was achieved through a succession of small, incremental steps, each of which moved in the direction of industrial adoption and collectively, over more than a decade, achieved that goal. Through small, incremental steps, excessive disruption of existing industrial design development flows was avoided. However, to be worth the effort of adoption, each small step nonetheless needed to offer some benefit over the current practice. The key point is cost-effectiveness: the small step needs only to provide a short-term benefit greater than its adoption cost. At the same time, longer-term benefits are too hard to predict and thus are generally ignored. Kurshan in [200] summarizes the specific challenges that need to be addressed to successfully transfer a disruptive technology:

- interface the new technology to the client environment;
- limit and then integrate methodology changes into client practice;
- demonstrate the cost-effectiveness of the new technology in the client's environment;
- enable competitive evaluations.

[1] Johnson, https://rjlipton.wordpress.com/2010/10/23/galactic-algorithms

3.1 Industrial Model-Based Design & Tools

Nicolescu and Mosterman illustratively describe model-based design (MBD) as an emerging solution to bridge the gap between computational capabilities that are available but that we are yet unable to exploit. They pinpoint that the use of a computational approach in the design itself allows raising the level of abstraction of the system specification at which novel and differentiating functionalities are captured. Automation can then assist in refining this specification to an implementation. For this to be successful, performance studies of potential implementations at a high level of abstraction are essential, combined with the necessity of traceability and parameterization throughout the refinement process [237].

More concretely, MBD is a paradigm for system design in which the design process starts with the construction of high-level models which are later used to guide further development, simulation, verification, and testing of the system. MBD has found industrial use in the field of embedded systems, particularly in automotive and avionics applications. The MBD approach seeks to place an emphasis on abstract, mathematical modeling as a first step before getting into low-level implementation details. The availability of such models, with associated formal (mathematical) specification of desired/undesired behaviors, can aid in simulation and verification early in the design process, thus identifying bugs in the logic of the system at a point where the cost of finding and fixing them is still relatively low, and improving overall system dependability [185, 273, 276].

In MBD, the designer should follow a standard a sequence of steps. Initially, the designer models the physical plant, relying either on first principles or on system identification; this model captures the dynamical characteristics of the physical parts of the system using mathematical equations. Then, the designer synthesizes a controller that regulates the behavior of the physical system. Subsequently, he performs extensive simulations to check the model behavior under different configuration settings. The aim is to analyze and evaluate the controller design by inspecting the behavior of specific variables over time. The analysis is typically performed with respect to some requirements. In practice, however, these requirements are high-level and often vague or informal. In case the system behavior is not satisfying with respect to these requirements, the designer has to manually modify the controller, e.g. by tuning the parameters or gains, and then repeat the validation step. Through these validation efforts, the design is deemed to be satisfactory or not. The evaluation may also remain inconclusive [177].

Afterwards, model transformation techniques are applied to transform abstract models into more concrete models, and later into source code. It should be noted that at any abstraction level, a complex model (system) can be composed/decomposed from/to several simpler sub-models (subsystems), which can be built by reusing existing verified components. In this respect, the development complexity becomes manageable by separation of concerns, the reliability of the system to be developed can be guaranteed by divide and conquer, and the productivity of the development can be improved by reuse [185, 297].

MDB has been successfully applied in industry, and has become a major approach in the design of computer controlled systems [237, 276, 298]. There is a variety of MBD approaches proposed and used in industry and academia, based on Simulink/Stateflow [227, 297, 298], Modelica [116, 285, 286], SCADE [2, 54], LabVIEW [58], DSpace [277], as well as earlier attempts, e.g., SysML [141, 163], MARTE [271], Metropolis [41], Ptolemy [115], CHARON [21].

To improve the efficiency and reliability of MBD, it is necessary to automate the design process as much as possible. This implies that we must apply design-automation techniques for modeling, simulation, synthesis, and verification to the system-design process. However, automation is not easy if a system-abstraction level is not well-defined, if components on any particular abstraction level are not well known, if system-design languages do not have clear semantics, or if the design rules and modeling styles are not clear and simple. This requires that all models at different abstraction levels should have precise mathematical semantics as well as the transformation between models at different abstraction levels should preserve semantics (ideally, it should be done automatically with tool support) [298].

3.1.1 Simulink

Model-based design is supported by MATLAB/Simulink and has been widely used in the industry. Simulink [227] is s a graphical programming environment for modeling, simulating and analyzing multi-domain systems, such as control systems, signal processing systems, communications systems, and other dynamic systems. It includes a set of block libraries and enables hierarchical, component-based modeling. It is fully integrated within MATLAB and facilitates mixed-signal, multirate and multitasking system simulation. It also offers code generation, e.g. C/C++, VHDL, PLC. A Simulink model comprises a set of inputs, outputs, and blocks.

Stateflow [280] is a toolbox that adds functionalities for modeling and simulating reactive systems with the use of hierarchical state diagrams, known as statecharts. In practice, Stateflow extends Simulink with a design environment for incorporating flow charts, switched actions and logical decisions. As such, it can be used to design supervisory, event-driven, and hybrid control. A Stateflow diagram has an hierarchical structure and is made up of events, variables, discrete states, and transitions. Stateflow states support exclusive (OR) as well as parallel (AND) decomposition. An OR diagram contains states which are linked with transitions and only one state is activated at any time. On the contrary, an AND diagram contains states that are arranged in parallel and are activated at the same time.

Simulink/Stateflow conducts fast simulation but depends on unverified numerical simulation. This fact entails two crucial errors: (i) simulations may provide incomplete coverage of system behaviors (especially, for open systems) and (ii) numerical errors may cause unsound analysis results. As such, validating requirements with Simulink's simulation may not lead to the discovery of critical design errors or bugs. If such insufficiently tested systems are deployed in a real setting, there could be critical system failure [298].

Formal verification techniques aim to ameliorate the inherent problems of simulation. Mathworks has designed a semi-formal tool, named Simulink Design Verifier, in order to complement simulation. The goal of this tool is to identify hidden design errors without requiring extensive simulation runs. Currently, it can low-level errors such as integer overflow, division by zero, dead logic, and array access violations. However, it cannot check the correctness of any system-level properties considering the entire system model or any environmental aspects [298].

3.1.2 SCADE

SCADE has been used in various application domains like avionics, nuclear plants, transportation, and automotive. SCADE was based on on the synchronous data-flow language Lustre [157]. Originally, it was used as a tool that provided a code generator that met several standards for safety critical applications. It also offered graphical notation for Lustre. In 2008, the new language 'SCADE 6' was developed. This language is a combination of the Lustre data-flow style with control structures and compilation and static analyses for ensuring safety properties. This increase in expressiveness together with the qualified code generator have broadened SCADE scope of applications [92].

SCADE originates from an effort to define proper languages to program high-integrity applications. Such applications have two main characteristics: a) they control a process using a read inputs/compute/write outputs cycle, b) a defect can have catastrophic effects involving possible fatalities. SCADE's solution was to employ the synchronous languages family relying on the zero-delay hypothesis. Moreover these languages have a well-defined and unambiguous semantics [110].

SCADE is a data-flow oriented language, close to the applied control engineer practices. Its semantics rely on the Kahn Process Network [147], which is a model of distributed computation.The language is strongly typed and declarative. Moreover, SCADE supports state-machines as well as data-flow and control-flow constructs. The SCADE Suite KCG code generator (i) checks if a SCADE model is correct with respect to the language semantics, ii) generates C or Ada code from the original model and iii) implements static analyses to ensure strong safety properties like determinism. SCADE Suite checks if a model is correct with respect to the language semantics, and can generate certified C code. The code generation and the static checks are qualified/certified for various safety standards, such as DO-178C/DO-330 at TQL-1, IEC 61508 at SIL 3, and ISO 26262. The SCADE Suite code generator is itself developed following the rules of the standards (DO-178, ISO 26262, EN 50128). Every artefact produced during its development is accessible to authorities, which can then assess the correctness of the code generator in more detail [65, 244].

3.2 Tool Integration

In this Section, we provide a workflow that is based upon current industrial practice and acts as an incremental patch that is able to demonstrate value without hampering existing methodologies. In particular, we propose a semi-automated toolchain that enables the formal verification of industrial-sized models employing well-established and recently designed tools. Our starting point is an industrial model, described in MATLAB/Simulink, and a set of industrial requirements, defined in natural language. The entire analysis process is semi-automated via a successive use of tools and translators. The end goal is to formally verify the correctness of the design against the specified requirements. Figure 3.1 indicates the proposed verification workflow. Figure 3.2 emphasizes on the construction of verified code from SpaceEx models and Figure 3.3 depicts all the internal steps that are required within SCADE Suite for the code generation.

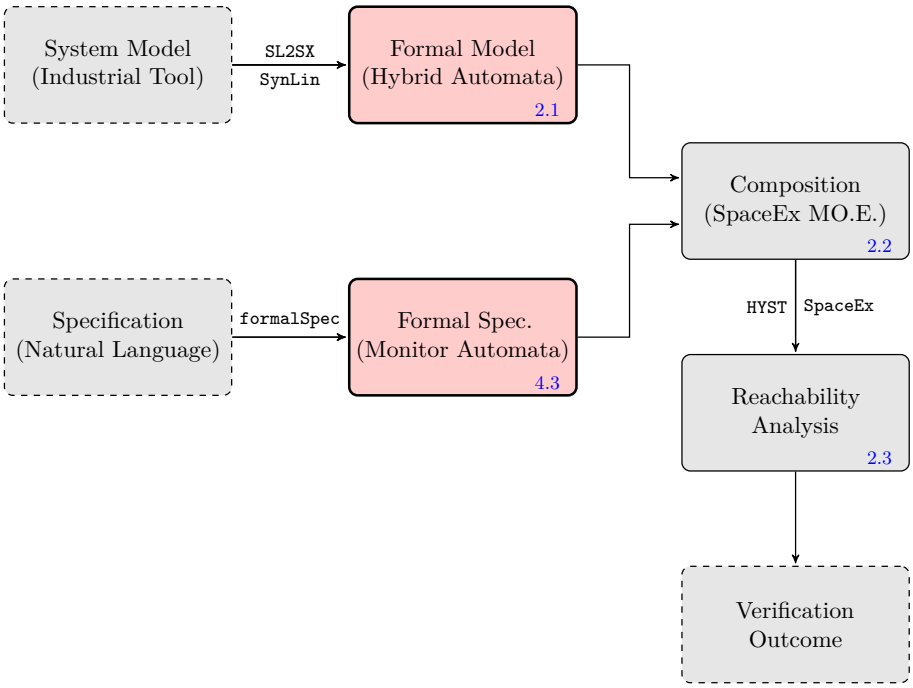


Figure 3.1: Proposed verification workflow: (i) start with an informal model and specs, (ii) formalize the model and the specs via pattern templates, (iii) feed the formal model and the specs into off-the-shelf reachability analysis tools. The contribution (pink) is the formalization using pattern templates tailored for model checking.

Simulink to SpaceEx Translator (SL2SX). Simulink to SpaceEx Translator (SL2SX). SL2SX is a semi-automated translator of Simulink models into equivalent SpaceEx models [231]. The user should select the input of the translator, i.e. a Simulink model saved as an XML file, and the tool builds a SpaceEx model, that is a network of hybrid automata. For hybrid automata, the tool uses the SX format since it is a popular formalism, employed by SpaceEx as well as other formal verification tools. The output model has the same structure with the original Simulink model. The variable names, hierarchical connection and graphical positions are preserved. The tool supports several continuous-time and arithmetical, logic-based blocks, and blocks with discontinuous dynamics, e.g. switches. Finally, SL2SX creates empty components for the unsupported Simulink blocks and the user should manually define them.

SynLin tool. SynLin is a prototype MATLAB tool that facilitates the semi-automated formal verification of Simulink models. Starting from a general Simulink model, it conducts a sequence of steps, i.e. model adaptation, model translation via SL2SX, guided simulation, and over-approximation of

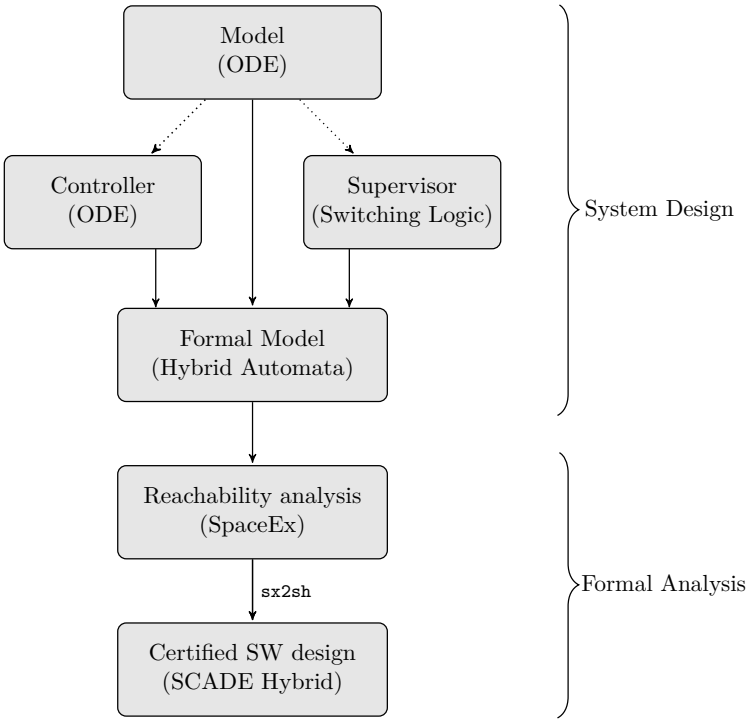


Figure 3.2: Proposed workflow and toolchain for the construction of verified controllers using SpaceEx and SCADE

nonlinear functions and blocks (if need be). Finally, it constructs a formal model, described as a network of hybrid automata. Note that simulation is useful to gain some knowledge regarding the operating conditions of the model states/variables. This information is important when conducting over-approximation/linearization of the nonlinear dynamics. Currently, the tool can conduct corner-case simulations via MATLAB or call Breach [107] to obtain more accurate bounds via falsification. These bounds are enlarged to avoid any out-of-range cases.

SCADE Hybrid. SCADE Hybrid is an extension of the SCADE language with constructs to define flows using *ordinary differential equations* (ODE). The resulting language is an explicit hybrid systems modeling language, like Simulink [227] or LabView [58]. It is different from implicit hybrid systems modeling languages like the ones implemented in Dymola [71] or ANSYS Simplorer [283], which use *differential algebraic equations*.

The language relies on [48] and has the following key ideas:

– The solving of ODEs is handled by an external numerical solver. No hypothesis is made on the method used by this solver.

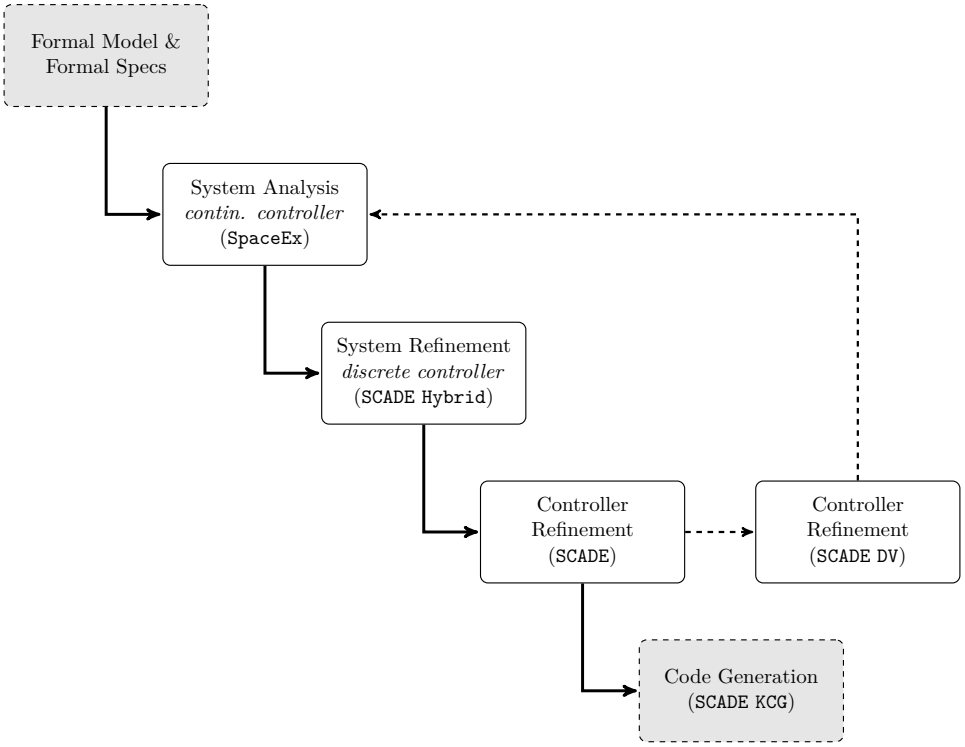


Figure 3.3: Applying SCADE Suite on hybrid automata models and their interaction with a model checker; two inputs: formal model and formal specifications, output: certified code generation of system requirements with formaSpec; SCADE DV stands for SCADE Design Verifier and contin. for continuous [244]

– Discrete-time computations are strictly separated from continuous-time computations. The goal is to ensure that the semantics of a model do not depend on the numerical solver used or its time step. Discrete computations must be triggered by a timer or a *zero-crossing*, which is detected by the solver when a signal crosses zero. Unlike Simulink and other tools that check such properties and display warnings in some cases, SCADE Hybrid only allows program that separate discrete and continuous computations. As such, the behavior of the hybrid programs is fully deterministic for the discrete part.

– The existing compilation process and infrastructure can be reused as much as possible. Discrete nodes are compiled exactly as in regular SCADE. This compilation process and the simulation algorithm is described in details in [67].

For explicit system modelers, an important step in their compilation is to detect algebraic and causality loops. Such loops are critical as they could lead to deadlock and impede the generation of statically scheduled code [47].

SpaceEx to SCADE Hybrid (sx2sh) translator. The `sx2sh` tool implements the translation of hybrid automata, defined as SpaceEx models, into SCADE Hybrid models. SCADE Hybrid deals with discrete and continuous modeling and essentially describes hybrid programs. The objective of the translation is to bridge a gap between high-level description with possible non determinism necessary for space exploration with deterministic low-level description necessary for path to implementation. Indeed, SpaceEx allows for reachability analyses within the environment while SCADE Hybrid allows for embedded software simulation within its environment and can lead to certifiable code generation using SCADE. Note that the current version supports urgent transitions.

formalSpec tool. The formalSpec tool [76] assists with the semi-automatic translation and instantiation of system requirements from controlled natural language into hybrid automata. The resulting hybrid automata are expressed formally as monitors (observer automata). These monitors can be automatically integrated within an existing SpaceEx system model via parallel composition and network components. The tool comes with a database of structured English phrases and the associated monitor automata.

In the following, we describe how the formalSpec tool operates:

- First, the user should specify the SpaceEx model as an input and the specifications that he/she wants to check.
- The tool can identify automatically or manually the top-level component of the system model and create a hierarchical, tree-based view of the complete model.
- The variables that appear in the formal specification do not have to be visible to the top-level component (global variables). They can be local variables accessible by lower-level network or base components. These variables are color-coded for clarity.
- The tool provides syntax highlighting to indicate whether the parameters used in the specifications are unambiguous within the SpaceEx system (green), ambiguous (yellow), or non-existing (red).
- After identifying the corresponding variables and the specification (from a predefined list), the tool instantiates the monitor automaton which encodes the specification. Then, it creates a new base component with

the monitor. The predicates (p, q, s) and associated parameters, e.g. T, (see Chapter 4 for more information) of the monitor are replaced by arithmetic expressions. The tool also offers the option to couple this new base component with the original SpaceEx model by linking the appropriate variables (through binds).

– The user can input more than one specifications to the tool. In this case, multiple monitors will be constructed.

HYST translator. HYST [40], a Hybrid Source Transformer, is a source-to-source conversion tool tailored to hybrid systems. The main functionality of HYST is to translate a hybrid automaton model expressed in SX format into formalisms that are supported by other formal verification tools. The user should select the input model, in SpaceEx format, and the tool converts it to the formats of HyCreate [37], Flow* [83], or dReach [195]. A useful feature of HYST is that it can optimize the input model in order to meet specific characteristics of the target tool. Specifically, the generation of pseudo-invariants is supported by HYST which could lead to reachability improvements.

MATLAB integration. All the aforementioned tools can be called and used via MATLAB scripts. Starting with a mathematical model expressed with ODEs (in MATLAB, Simulink, or hybrid automata) and a set of natural language requirements, we can semi-automatically obtain the formal model (both in discrete and continuous time) and the formal specifications. Then, we have interfaced the SpaceEx analysis core to conduct reachability analysis. Some scripts can be found at [186].

CHAPTER 4

From Informal Requirements to Formal Specifications via Pattern Templates

Parts of this Chapter have been published in [132, 133].

Requirements play an important role in every design process [239]. Requirements broadly describe any necessary or desired function, attribute, characteristic, capability or quality that a system should satisfy and possess. Requirements can have various levels of specificity and may refer to a product or a process [253]. In practice, product and process requirements are closely linked. According to their purpose, they can be divided in functional and non-functional requirements. Non-functional requirements describe how the system works and include performance objectives, whereas functional requirements describe what the system should do. According to [211], requirements that indicate what a "system shall be" are non-functional, whereas requirements that express what "a system must do" are functional. The field of requirements engineering [278] specializes in defining, documenting and maintaining requirements in the engineering design process.

Industrial requirements are typically described in natural language or controlled natural language (CNL) [198]. A CNL is a subset of natural language with a constrained grammar aiming to reduce or eliminate ambiguity. There are various CNL languages available which serve different purposes (see Section 1.1), such as template-based natural languages [100]. Such

languages can be helpful since similar classes of requirements are encountered in industrial practice.

Generally, requirements written in some form of natural language cannot be directly interpreted by tools for formal verification. They first need to be translated into a formal representation, such as temporal logic [252]. However, this task for complicated properties and hybrid systems is prone to errors and not straightforward; especially for users that are not experts in formal methods.

In this respect, the goal of this Chapter is twofold: (i) to bridge the gap between between industrial requirements and formal specifications, and (ii) to encode rich formal specifications so that they can be interpreted by reachability tools and can be verified automatically. In particular, we propose a semi–automated, template–based translation of industrial requirements into a formal representation (monitor automata). This translation assists in mitigating the semantic mismatch and the use of monitor automata that enables the algorithmic verification of rich specifications.

Herein, we consider specifications expressed by *pattern templates* which are predefined properties with placeholders for state predicates. Pattern templates [142, 196] can be readily understood by non-expert users as they use expressions from natural language. In this Chapter, we provide (i) formal definitions for select patterns in the formalism of hybrid automata that are applicable over both bounded and unbounded time and (ii) monitors with correctness proofs which encode these properties as a reachability problem of an error state. The composition of these monitors with the formal model enables the algorithmic verification of the property via off-the-shelf fully automated tools.

A schematic of the proposed workflow is depicted in Figure 5.9. A (safety) requirement in CNL is translated into a monitor automaton using pattern templates [196]. The monitor automaton has the same syntax and respects the same semantics as the system model. As such, it can be composed with the system model and fed into a reachability tool. The monitor automaton encodes the requirements as the reachability of a designated error state (see [156] for one of the earliest works on monitor automata and [72] for analog and mixed-signal applications). The formalSpec tool [76] can assist in automatically instantiating monitor automata from CNL. The tool comes with a database of structured English phrases and the associated monitor automata.

The remainder of the Chapter is organized as follows. In Section 4.1, we introduce the pattern templates, present illustrative examples and describe selected patterns in an intuitive manner. In subsections 4.1.1 and 4.1.2, we

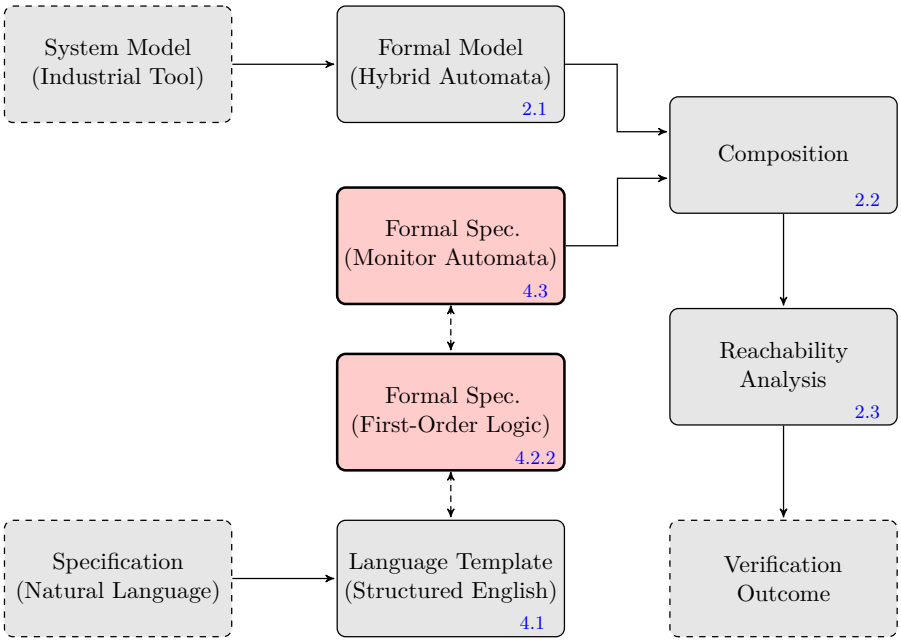


Figure 4.1: Proposed verification workflow: (i) start with an informal model and specs, (ii) formalize the model and the specs via pattern templates, (iii) feed the formal model and the specs into off-the-shelf reachability analysis tools. The contribution (pink) is the formalization using pattern templates tailored for model checking.

elaborate on the applicability of these pattern tempates in control systems and industrial use cases. The formalization of the selected pattern templates in hybrid automata formalism is presented in Section 4.2. The corresponding monitor automata are described in Section 4.3. The formal proofs of correctness can be found in Section 4.4. We apply the introduced monitors to an electromechanical brake use case in Section 4.5. In Section 4.6, we give an overview of the state of the art.

4.1 Pattern Templates

Pattern templates are predefined properties with placeholders for state predicates and were introduced in [142]. Informally, a state predicate refers to a statement/formula that may be true or false depending on the values of its variables. It is commonly understood as a Boolean-valued function, i.e. an operator that returns a value that is either true or false. Background information on boolean functions and temporal logic can be found in Section 2.6. Hereafter, we use p, q, and s to describe state predicates, i.e. a function $\mathbb{R}^X \rightarrow \mathbb{B}$.

The list of patterns considered in this work and the examples are based on [114, 196]. Most pattern intents are borrowed from [34]. Some of the pattern templates that we employ in this thesis have been formally defined using temporal logic: LTL, CTL and timed CTL in [34], MTL in [44,196], and probabilistic LTL in [153]. These definitions, however, do not immediately carry over to monitoring with hybrid automata, since hybrid automata cannot distinguish events with the precision of a temporal logic. In the next section, we formalize the properties such that they can be applied to hybrid automata.

Absence. *"After q, it is never the case that p holds."*

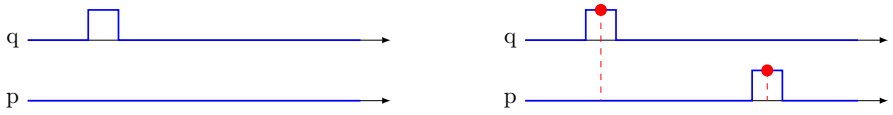


Figure 4.2: Absence pattern: satisfied (left), violated (right).

Definitions. Informally, the absence pattern specifies a state predicate that must not hold. Formally, the absence pattern can be encoded as an STL specification, i.e. $\Box(q \rightarrow \Box(\neg p))$.

Pattern intent. This pattern describes a portion of a system's execution that is free of certain events or states. An example, borrowed from ABS systems, is that "the ABS controller should never allow a wheel skidding".

Absence (timed). *"When T time units are measured, after q was first satisfied, it is never the case that p holds."*

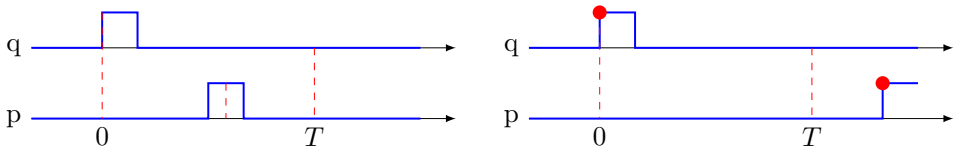


Figure 4.3: Timed absence pattern: satisfied (left), violated (right).

Definitions. Informally, this pattern specifies a state predicate that must not hold after a certain amount of time has passed. Formally, this pattern can be encoded as an STL specification, i.e. $\Box(q \rightarrow \Box_{[T,\infty)}(\neg p))$.

Pattern intent. This pattern describes a portion of a system's execution that is free of certain events or states after a time-bound. An example, borrowed

from cybersecurity, is that "after a login failure, the system must be free of login attempts between 10 to 50 milliseconds".

Universality. *"After q, it is always the case that p holds."*

Figure 4.4: Universality pattern: satisfied (left), violated (right).

Definitions. Informally, the universality pattern specifies a state predicate that must always hold. Formally, the universality pattern can be encoded as the STL specification $\square(q \rightarrow \square p)$.

Pattern intent. This pattern describes a portion of a system's execution which has a desired property. An example, borrowed from ABS systems, is that "the ABS controller should always guarantee vehicle steerability".

Remark 1. Note that the universality can be seen as the dual of the absence pattern.

Minimum duration. *"After q, it is always the case that once p becomes satisfied, it holds for at least T time units ($T > 0$)."*

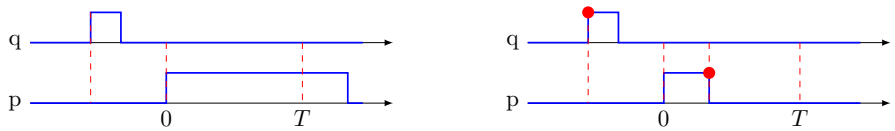


Figure 4.5: Minimum duration pattern: satisfied (left), violated (right).

Definitions. Informally, this pattern describes the minimum amount of time a state predicate has to hold once it becomes true. Formally, this pattern can be encoded as the STL specification $\square(q \rightarrow \square(p \vee (\neg p \; \mathcal{W} \; \square_{[0,T]} p)))$. The temporal operator $\mathcal{W}$ stands for weak until, i.e. $p_1 \mathcal{W} p_2 := (p_1 \mathcal{U} p_2) \vee \square p_2$.

Pattern intent. This pattern captures the property that every time the state formula P switches from false to true, it will remain true for at least t time unit(s). An example, borrowed from engine starter systems, is that "the system has a minimum 'off' period of 120s before it reenters the cranking mode".

Maximum duration. *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*

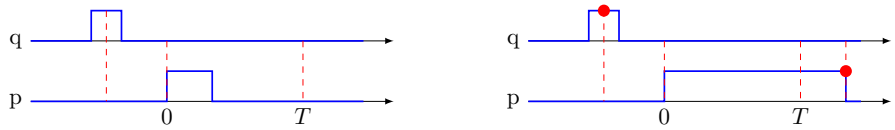


Figure 4.6: Maximum duration pattern: satisfied (left), violated (right).

Definitions. Informally, the maximum duration pattern captures that a state predicate always holds for less than a specified amount of time. Formally, this pattern can be encoded as the STL specification $\Box(q \rightarrow \Box(p \vee (\neg p \ \mathcal{W} \ (p \wedge \Diamond_{[0,T]}(\neg p)))))$.

Pattern intent. This pattern captures the property that every time the state formula P switches from false to true, it will remain true for at most t time unit(s). An example, borrowed from engine starter systems, is that "the system can only operate in engine cranking mode for no longer than 10s".

Bounded invariance. *"After q, it is always the case that if p holds, then s holds for at least T time units."*

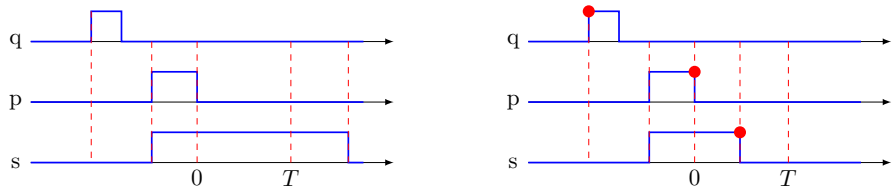


Figure 4.7: Bounded invariance pattern: satisfied (left), violated (right).

Definitions. Informally, the bounded invariance pattern specifies the minimum amount of time a state predicate must hold once another predicate is satisfied. Formally, this pattern matches the STL specification $\Box(q \rightarrow \Box(p \rightarrow \Box_{[0,T]} s))$.

Pattern intent. An example, borrowed from engine starter systems, is that "if the error 502 is sent to the Drive Information System, the braking system is inhibited for 10s".

Bounded recurrence. *"After q, it is always the case that p holds at least every T time units ($T > 0$)."*

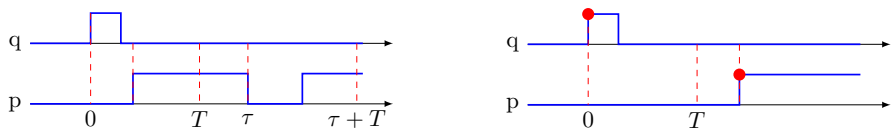


Figure 4.8: Bounded recurrence pattern: satisfied (left), violated (right).

Definitions. Informally, the bounded recurrence pattern denotes the amount of time in which a state formula has to hold at least once. Formally, this pattern can be described by the STL specification $\Box(q \rightarrow \Box(\Diamond_{[0,T]}p))$.

Pattern intent. This pattern requires that P must recurrently hold. An example, borrowed from ABS systems, is that "the ABS controller checks for skidding every 10ms".

Bounded response (persisting). *"After q, it is always the case that if p holds, then s persists (holds for nonzero time) after at most T time units. '*

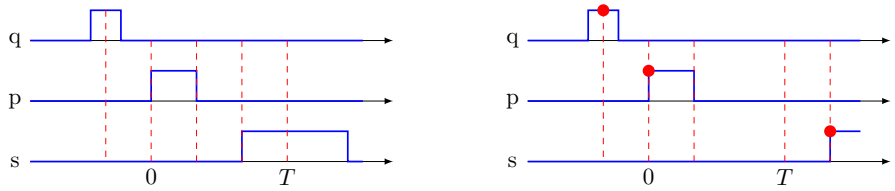


Figure 4.9: Bounded response pattern: satisfied (left), violated (right).

Definitions. Informally, the bounded response pattern restricts the maximum amount of time that passes after a state predicate holds until another predicate becomes true. Formally, this pattern can be formulated as the STL specification $\Box(q \rightarrow \Box(p \rightarrow \Diamond_{[0,T]}s))$.

Pattern intent. To describe cause-effect relationships between a pair of events/states. An occurrence of the first, the cause p, must be followed by an occurrence of the second, the effect s within a time-bound. An example, borrowed from ABS systems, is that "from direct client input, detection and response to rapid deceleration must occur within 0.015s".

4.1.1 Patterns occurring in Control Systems

Some control specifications have been expressed with Temporal Logic [183], Simulink monitors [43], or Modelica language [241]. In this Section, we show how some common control specifications can be expressed with our formalized pattern templates. We consider the untriggered version of the requirements ($q :=$ true). We assume a constant, positive reference signal x_{ref} as well as that $x(0) < x_{ref}$ holds.

Safety. *"The state x of the system should always be inside the acceptable operating range expressed as safe region S."*

This property is matched by the **absence** pattern with $p := \{x \notin S\}$. Equivalently, $p := \{x <= \min(S) \wedge x >= \max(S)\}$.

Target Reachability. *"The state x of the system should be within distance ε of the target (x_{target}) within T time units."*

This property can be encoded as the **bounded response** pattern, where $p :=$ true and $s := \{d(x, x_{target}) \leq \varepsilon\}$, with $d(x, y)$ being a function that computes the distance between states x and y.

Overshoot. *"The state x of the system should not exceed an overshoot of $ov\%$ with respect to the reference x_{ref}".*

This property can be formulated as an **absence** pattern, where $p := \{x > (100 + ov)\% \cdot x_{ref}\}$.

Settling Time. *"The state x of the system should reach and stay within a $per\%$ of the reference x_{ref} within T_{set} time units."*

This property can be described by the **absence (timed)** pattern, where $T := T_{set}$, p:= $\{x \leq (100 - per)\% \cdot x_{ref} \vee x \geq (100 + per)\% \cdot x_{ref}\}$.

Rise-Time. *"The state x of the system should reach 90% of the reference x_{ref} at time T_{rise}".*

This property can be mapped to the **bounded response** pattern, where $p :=$true, $T := T_{rise}$, and $s := \{x \geq 0.9 \cdot x_{ref}\}$.

Undershoot. *"After reaching the reference x_{ref}, the state x of the system should not fall below a threshold of $u\%$ with respect to the reference."*

This property can be expressed with the **bounded invariance** pattern, where $T := \infty$, $p := x \geq x_{ref}$ and $s := \{x \geq (100 - u)\% \cdot x_{ref}\}$.

Remark 2. In several cases above, the monitor can be simplified (when $p :=$true, $T := 0$, etc.) or be expressed with multiple pattern templates. A varying reference signal can be captured with the introduction of predicate q.

4.1.2 Patterns found in Industrial Use Cases

Formalizing practical requirements is a challenging task, even for experts [105, 172]. This Section shows the applicability of the select pattern templates to different application domains. Together with out industrial partners, we have selected requirements from three different application domains, wind turbines [188, 265], automated driving [168, 266] and braking systems [130, 281]. Some of the requirements can be difficult to translate into a formal representation without a given pattern template.

Wind Turbines.
Requirement 1: "The pitch rate of the turbine blades shall not be larger than the maximal pitch rate".

This requirement can be mapped to the *absence* pattern.

Requirement 2: "The generator torque rate shall be between the minimum and the maximum torque rate".

This requirement can be mapped to the *universality* pattern.

Requirement 3: "In partial load, the pitch angle shall be larger than the stall pitch angle".

This requirement can be mapped to the *universality* pattern.

Requirement 4: "The absolute difference between the commanded pitch angle and the measured pitch angle can only be larger than the maximum difference for less than c time units.

This requirement can be mapped to the *maximum duration* pattern.

Requirement 5: "The absolute difference between two individual pitch angles can only be larger than the maximum difference for less than c time units".

This requirement can be mapped to the *maximum duration* pattern.
Requirements 1 - 3 describe time invariant properties, whereas requirements 4 and 5 depend explicitly on time.

Automated Driving.
Requirement 1: "The largest communication sequence flow duration shall be less than *TBD* seconds".

This requirement can be encoded as the *maximum duration* pattern.

Requirement 2: "The maximal waiting time for an *ACK* message should be 20 ms".

This requirement can be encoded as the *maximum duration* pattern.

Requirement 3: "When an acknowledgeable message arrives, an *ACK* message shall be sent to the sender by the receiver within the maximal waiting time".

This requirement can be encoded as the *maximum duration* pattern (triggered).

Requirement 4: "If an *ACK* message exceeds the maximal waiting time, the message being acknowledged shall be considered as lost".

This requirement can be encoded as the *universality* pattern.

Requirement 5: "A vehicle shall send `MVR_FINISHED` messages to its session partner after finishing successfully its planned maneuver".

This requirement can be encoded as the *universality* pattern (triggered).

Requirement 6: "After a vehicle i received a propose message with the information {ID-x, constraints-x, Tfinish-x} and after vehicle i sent an accept message with ID-x, every planned contingency manoeuvre of vehicle i must satisfy constraints-x, until the point of time Tfinish-x is reached".

This requirement can be encoded as the *minimum duration* pattern (triggered).

Requirement 7: "After starting execution of the manoeuvre primitive k it is always the case that if the actual disturbances and measurement errors are below error-bounds-k then the deviation of the state from the reference trajectory is below state-bounds-k for at least duration-k.

This requirement can be encoded as the *bounded response* pattern (triggered).

Brake systems.

Requirement 1: "The caliper must reach $x_0 = 0.05$ dm after the braking request is issued within 20 ms with a precision of 4%".

This requirement can be expressed by the *bounded response* pattern.

Requirement 2: "The caliper speed at contact must be below 2 mm/s".

This requirement can be expressed by the *absence* pattern.

Note that events, boolean signals and messages can be encoded as state predicates.

4.2 Formalizing Pattern Templates for Hybrid Automata

In the previous section, we provided a list of pattern templates along with formal and informal definitions in structured English. These pattern templates were also formally defined using temporal logic, e.g. MITL in [34, 196]. These definitions, however, do not immediately carry over to monitoring with hybrid automata. The hybrid automata cannot distinguish events with the precision of a temporal logic as the intersection between the location invariants and location guards must be non-empty. In addition, hybrid automata traces include left and right closed intervals whereas properties might be defined over open intervals. In this respect, in this section, we define these pattern templates in a formalism that is suitable for hybrid automata. Note that our definitions can differ slightly from the MITL semantics since they are tailored specifically to hybrid automata monitors. Later, in Section 4.3, we are going to see how to use these templates with reachability tools.

Note that the properties in this Chapter refer to predicates that describe states, not events. In addition, the predicates can express timing properties by adding a clock to the monitor, so that time becomes a state variable. We consider *triggered* versions of the properties that only take effect after a predicate q holds. A run, for which $!q$ always holds, satisfies the property.

4.2.1 Preliminaries

We consider p, q, s to be predicates over the state variables, which means functions $\mathbb{R}^X \to \mathbb{B}$. We use the shorthand notation $p(\mathbf{x})$ to indicate that p is true for $\mathbf{x}$. A hybrid automaton run $r \in R$ is given by continuous states $\mathbf{x}_i$, locations ℓ_i, durations δ_i, and trajectories ξ_i. We write the set of runs of a hybrid automaton as $\mathrm{Runs}(H)$. We also use some notation for the timing of states on runs, since it simplifies formalization of the properties. The *event-times* of a run r are denoted by

$$t_i = \sum_{j=0}^{i} \delta_j.$$

That means that the jump number i occurs at time t_i for $i = 0, \ldots, N-1$. For further notational convenience, we assume $t_{-1} = 0$.

We establish a total order on the time points of a run by considering pairs (i, t), with i being an index and t denoting the global time. Formally, the *event-time* is $\mathbb{T} = \mathbb{N}^0 \times \mathbb{R}^{\geq 0}$. To clarify the difference, we define real

time with the symbol t and event-time with $\tau \in \mathbb{T}$. Event-times are ordered lexicographically, formally

$$(i,t) < (i',t') \Leftrightarrow (i < i') \vee (i = i' \wedge t < t').$$

Note that the event-time enables us to uniquely identify continuous and discrete states on the run. The *event-time domain* of a run r is expressed by the set of pairs:

$$\mathrm{dom}(r) = \{(i,t) \mid 0 \leq i \leq N-1, t_{i-1} \leq t \leq t_i\} \cup \{(N, t_{N-1})\},$$

with the last term pointing out that the last state in the run, $(\ell_N, \mathbf{x}_N)$ is taken at time t_{N-1} (total duration of the run).

The *open truncated* event-time domain of a run r which excludes the last T time units is described by the set of pairs

$$\mathrm{dom}_{-T}(r) = \{(i,t) \in \mathrm{dom}(r) \mid t < t_{N-1} - T\}.$$

The truncated domain is used for properties that refer to future events and are not included in the domain of the run. We opt for an optimistic outlook of such cases: in case the property holds on the truncated domain, we assume that it holds on the run.

For a given $\tau = (i,t) \in \mathrm{dom}(r)$, we denote $r(\tau) \in \mathbb{R}^X$ as the continuous state $\xi_i(t - t_i)$ and $r_{\mathsf{Loc}}(\tau) \in \mathsf{Loc}$ as the discrete state ℓ_i. This denotes the time passed between two event-times $\tau = (i,t), \tau' = (i',t')$ as

$$d(\tau, \tau') = t' - t.$$

In certain cases, we only care about the first time when a predicate holds. If the predicate, for example q, is true over a left-open interval, the infimum shall be used. We write the shorthand $\tau_{q.1} = \inf_{\tau \in \mathrm{dom}(r)} q(r(\tau))$. To formally describe that a predicate holds at τ for nonzero time, we define for every run r, a predicate p, and an event-time τ,

$$\mathrm{persists}\,(r,p,\tau) = \quad \exists \delta > 0 \mid \forall \tau', (\tau \leq \tau') \wedge (d(\tau,\tau') \leq \delta) \Rightarrow r(\tau').$$

4.2.2 Formal Definitions

We formally define the properties of a hybrid automaton with its runs. A hybrid automaton H satisfies a property ϕ if and only if all runs $r \in \mathrm{Runs}(H)$ satisfy ϕ. We use the convention: (i) $r \models \phi$ when a run r satisfies the property ϕ, (ii) p following q means that there are τ_q and τ_p with $\tau_p \geq \tau_q$ such that

$p(r(\tau_p))$ and $q(r(\tau_q))$ hold, (iii) τ_q, τ_p, $\tau_{\bar{p}}$, τ'_p respectively imply that $q(r(\tau_q))$, $p(r(\tau_p))$, $\neg p(r(\tau_{\bar{p}}))$, $p(r(\tau'_p))$ hold.

Absence. *"After q, it is never the case that p holds."*

$r \models \phi$ iff for all $\tau_q, \tau \in \mathrm{dom}(r)$ with $\tau \geq \tau_q$, holds $\neg p(r(\tau))$.

Figure 4.10: Absence pattern: satisfied (left), violated (right).

Remark 3. Note that we use the verb *hold* to describe a property that has been always true. On the contrary, *becomes satisfied* corresponds to an edge, which implies that the signal was earlier false and afterwards it became true.

Absence (timed). *"When T time units are measured, after q was first satisfied, it is never the case that p holds."*

$r \models \phi$ iff for all $\tau_q, \tau \in \mathrm{dom}(r)$ with $d(\tau_q, \tau) \geq T$, holds $\neg p(r(\tau))$.

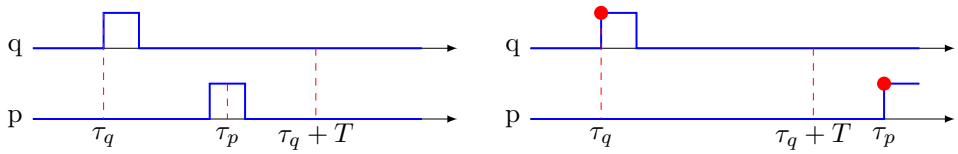


Figure 4.11: Timed absence pattern: satisfied (left), violated (right).

Universality. *"After q, it is always the case that p holds".*

$r \models \phi$ iff for all $\tau_q, \tau \in \mathrm{dom}(r)$ with $q(r(\tau_q))$ and $\tau \geq \tau_q$, holds $p(r(\tau))$.

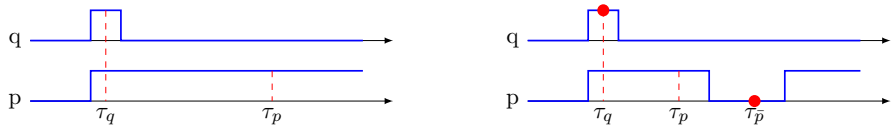


Figure 4.12: Universality pattern: satisfied (left), violated (right).

Minimum duration. *"After q, it is always the case that once p becomes satisfied, it holds for at least T time units."*

$r \models \phi$ iff p following q implies that for $\tau_{q.1}$ holds:

1. for all $\tau_p, \tau_{\bar{p}} \in \mathrm{dom}(r)$ with $\tau_{q.1} \leq \tau_p < \tau_{\bar{p}}$, $d(\tau_{q.1}, \tau_{\bar{p}}) > T$ (p not becoming false within T after $\tau_{q.1}$), and
2. for all $\tau_p, \tau_{\bar{p}}, \tau'_{\bar{p}} \in \mathrm{dom}(r)$ with $\tau_{q.1} \leq \tau_{\bar{p}} < \tau_p < \tau'_{\bar{p}}$, it holds that $d(\tau_{\bar{p}}, \tau'_{\bar{p}}) > T$ (violations of p are more than T apart).

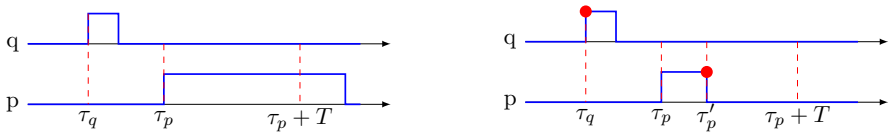


Figure 4.13: Minimum duration pattern: satisfied (left), violated (right).

Maximum duration. *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*

$r \models \phi$ iff p following q implies that for all $\tau_p, \tau'_p \in \mathrm{dom}(r)$ with $\tau_p \geq \tau_q$ one of the following holds:

1. $d(\tau_p, \tau'_p) < T$ (τ'_p is early enough, covering the $\tau_p = \tau'_p$ case), or
2. there is a $\tau_{\bar{p}}$ such that $\tau_p < \tau_{\bar{p}} < \tau'_p$ (p is false in between).

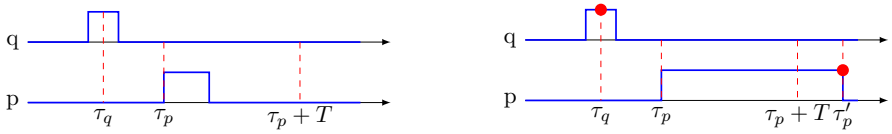


Figure 4.14: Maximum duration pattern: satisfied (left), violated (right).

Bounded invariance. *"After q, it is always the case that if p holds, then s holds for at least T time units."*

$r \models \phi$ iff p following q implies that for all $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \geq \tau_{q.1}$ and for all $\tau \in \mathrm{dom}(r)$ such that $\tau_p \leq \tau$, $d(\tau_p, \tau) < T$, the predicate $s(r(\tau))$ is true.

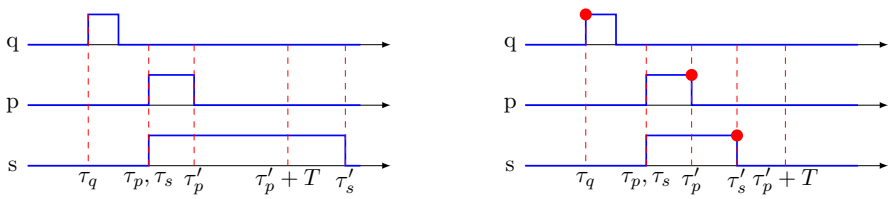


Figure 4.15: Bounded invariance pattern: satisfied (left), violated (right).

Remark 4. It is worth mentioning that in case $s = p$, the predicate p must hold forever (due to recursion), if it becomes true.

Bounded recurrence. *"After q, it is always the case that p holds at least every T time units."*

For the unbounded case, $r \models \phi$ iff for all $\tau_q \in \mathrm{dom}(r)$ both following criteria hold:

(i) for all $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \geq \tau_q$ there is a $\tau_p' \in \mathrm{dom}(r)$ such that $\tau_p < \tau_p'$, $d(\tau_p, \tau_p') \leq T$ (τ_p's with distance less than T),

(ii) there is a $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \geq \tau_q$ such that $d(\tau_q, \tau_p) \leq T$ (distance between τ_q and first τ_p is less than T).

For a bounded time horizon, $r \models \phi$ iff for all $\tau_q \in \mathrm{dom}_{-T}(r)$ both following criteria hold:

(i) for all $\tau_p \in \mathrm{dom}_{-T}(r)$ with $\tau_p \geq \tau_q$ there is a $\tau_p' \in \mathrm{dom}(r)$ such that $\tau_p < \tau_p'$ and $d(\tau_p, \tau_p') < T$,

(ii) there is a $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \geq \tau_q$ and $d(\tau_q, \tau_p) \leq T$.

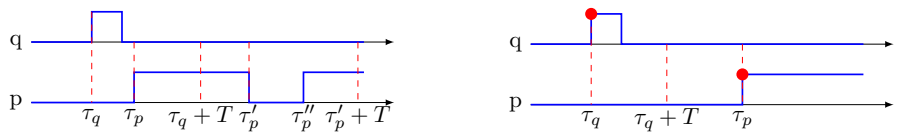


Figure 4.16: Bounded recurrence pattern: satisfied (left), violated (right).

Bounded response (persisting). *"After q, it is always the case that if p holds, then s persists (holds for nonzero time) after at most T time units."*

For an unbounded time horizon, $r \models \phi$ iff p following q implies that for all $\tau_q \in \mathrm{dom}(r)$, it holds that for all $\tau_p \in \mathrm{dom}(r)$ with $\tau_p \geq \tau_q$, there is a $\tau_s \in \mathrm{dom}(r)$ such that $\tau_p \leq \tau_s$, $d(\tau_s, \tau_p) \leq T$, and persists (r, τ_s, s).

For a bounded time horizon, $r \models \phi$ iff p following q implies that for all $\tau_q, \tau_p \in \mathrm{dom}_{-T}(r)$ with $\tau_p \geq \tau_q$, there is a $\tau_s \in \mathrm{dom}(r)$ such that $\tau_p \leq \tau_s$, $d(\tau_p, \tau_s) \leq T$, and persists (r, τ_s, s).

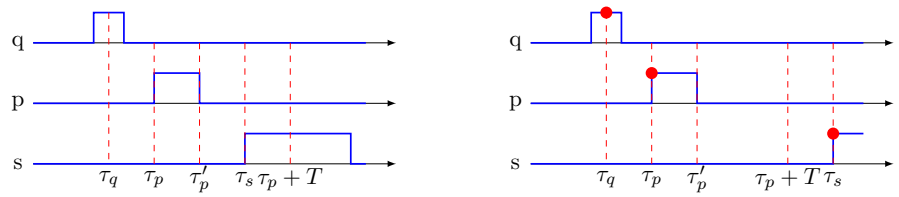


Figure 4.17: Bounded response pattern: satisfied (left), violated (right).

Remark 5. We require $\tau \in \text{dom}_{-T}(r)$ in the bounded time horizon (with the restricted domain being right-open) as we consider an optimistic interpretation of the bounded runs. If there exists a continuation of the run where the system satisfies the property, then the run satisfies the property. Note that selecting the restricted domain to be right-closed would yield a problem. That is the case since a run that with $\neg s$ would violate the property. However, it could have have a continuation that sets s to true (in zero time), which should actually satisfy the property.

Remark 6. We formally require that the predicate s holds for nonzero time, via persists $(\cdot)$. Otherwise, the monitor might generate a false alarm.

4.3 Verifying Pattern Templates using Monitor Automata

A formal alternative to temporal logic is the use of monitor automata (also known as observer automata). A monitor automaton has the same syntax and respects the same semantics as the system model. As such, it can be composed with the system model and fed into a reachability tool. In this way, a monitor automaton encodes the requirements as the reachability of a designated error state (see [156] for one of the earliest works on monitor automata and Figure 4.18 for an illustration). In the context of formal verification of hybrid systems, monitor automata have been used for checking whether simple safety requirements are satisfied or violated [260].

In this Section, we provide monitor automata for the selected pattern templates which express more general and complicated properties. Recall that a hybrid automaton that has a predicate p in the invariant of source location cannot have a target location with an invariant $!p$ because there would be a deadlock (go back to Section 2.1 for more details).The formalSpec [76] tool, presented in Section 3.2, automates the instantiation of monitor automata from controlled natural language English. The tool comes with a database of structured English phrases and the associated template monitors.

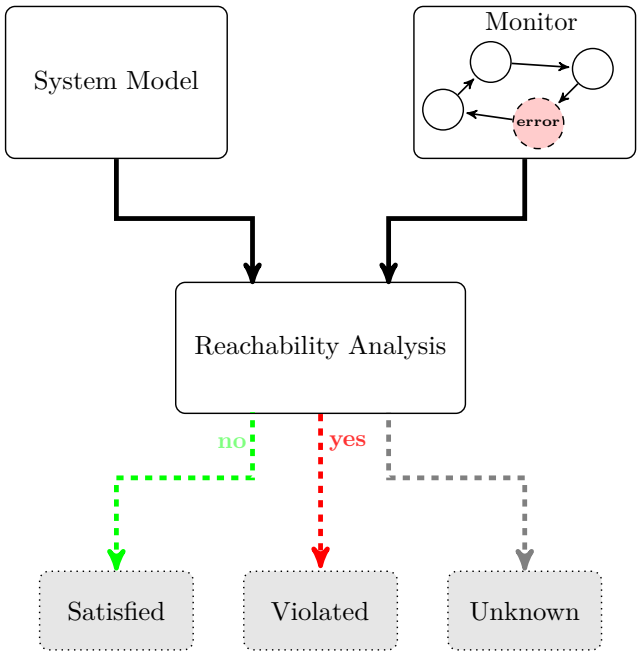


Figure 4.18: Principle of monitor automaton (also called observer automata [156]) - Reachability of an error state.

Absence. *"After q, it is never the case that p holds."*

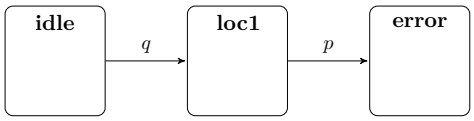


Figure 4.19: Monitor for absence pattern

Remark 7. Note that the monitor automata proposed in this Section are nondeterministic. This is a practical choice since deterministic automata would not be suitable and would yield inefficient monitor constructions. This is the case since the selected pattern templates encode future temporal logic properties. That means that a monitor automaton has to "evolve" together with the system and guess what happens in the future. Later when the monitor has more information, it has to re-check whether these guesses are wrong and if the corresponding runs should be abandoned. Processing all these spurious runs and delaying the output of the verdict until there is sufficient information can be a serious bottleneck. In practice, opting for determinism, would result to inefficient monitor automata with a large number of locations and increased computational costs.

Absence (timed). *"When T time units are measured, after q was first satisfied, it is never the case that p holds."*

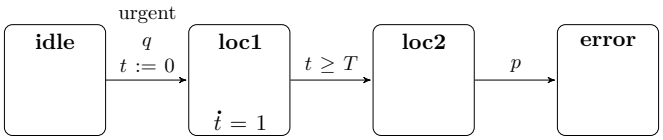


Figure 4.20: Monitor for timed absence pattern

Remark 8. In this Section, we do not discuss how urgent transitions can be treated by standard reachability tools (which do not support urgency); see Section 5.4 for more details on how to efficiently handle this semantics mismatch.

Remark 9. Notice that for the time absence monitor, shown in Figure 4.20, the urgent transition from idle location to loc1 can be replaced by a standard non-urgent transition (see Figure 4.21). Both monitors describe the property correctly. On the one hand, for the monitor with the urgent transition, we only focus on the first time q holds. Once q appears, the clock starts counting and the property is checked for T time units. On the other hand, for the monitor with the non-urgent (may) transition, we do not consider only the first instance of q but all the possible future instances. Also, we re-initialize the clock every time q appears. However, measuring time from a later occurrence of q only goes to the error state if anyway it would go to the error (in a path involving the first q).

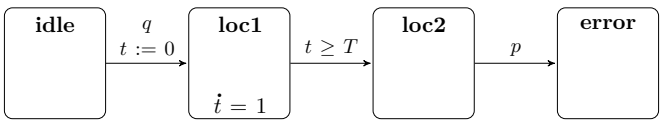


Figure 4.21: Monitor for timed absence pattern

Universality. *"After q, it is always the case that p holds."*

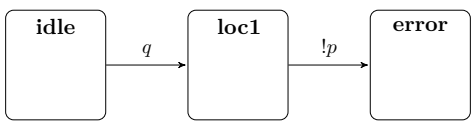


Figure 4.22: Monitor for universality pattern

Minimum duration. *"After q, it is always the case that once p becomes satisfied, it holds for at least T time units."*

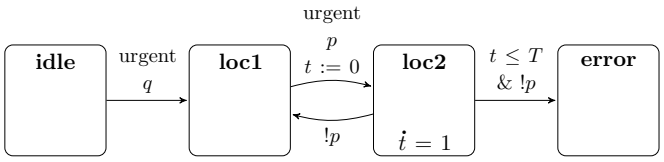


Figure 4.23: Monitor for minimum duration pattern

Maximum duration. *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*

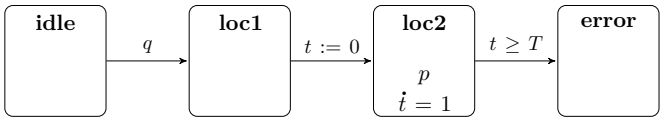


Figure 4.24: Monitor for maximum duration pattern

Remark 10. A semantically equivalent monitor for the maximum duration pattern is presented in Figure 4.25. It can be observed that the only difference between the two monitors is that they have a different guard for the jump from loc2 to error location; the former has $t \geq T$ while the latter has $t \geq T \ \& \ p$. Both monitors have the same invariant conditions for loc2 and error locations. Note that the empty invariant corresponds to a condition that is always true. Thus, the condition for the automata to jump from loc2 and error, defined by the intersection of the guard and the invariants of these locations, is exactly the same, $t \geq T \ \& \ p$, for both monitors. Following the same reasoning, the guard p from loc1 to loc2 could be ignored.

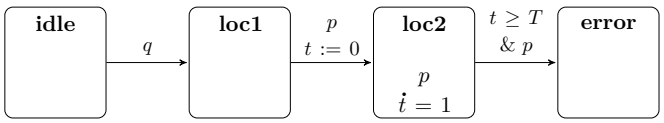


Figure 4.25: Equivalent monitor for maximum duration pattern; there is an extra condition, p, on the guard from loc2 to error location.

Remark 11. The monitors in Figures 4.24 and 4.25 are equivalent. However, we mention both of them since the former one includes less operators and the

latter is more intuitive (as indicated through discussions with our industrial partners).

Bounded invariance. *"After q, it is always the case that if p holds, then s holds for at least T time units."*

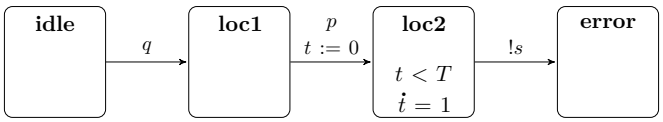


Figure 4.26: Monitor for bounded invariance pattern

Bounded recurrence. *"After q, it is always the case that p holds at least every T time units."*

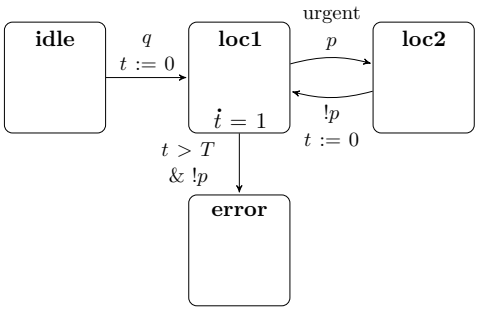


Figure 4.27: Monitor for bounded recurrence pattern (a false positive is avoided by adding the constraint !p on the guard of the ransition from loc1 to error)

Bounded response (persisting). *"After q, it is always the case that if p holds, then s persists (holds for nonzero time) after at most T time units."*

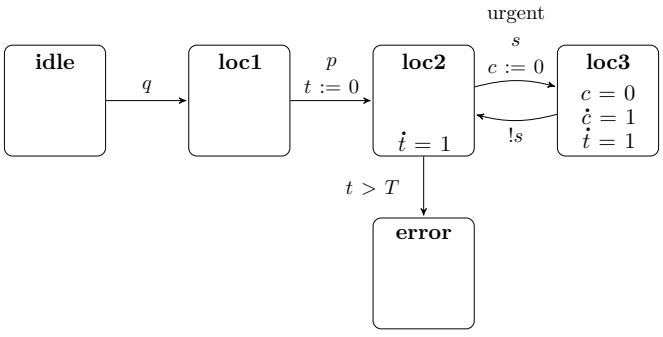


Figure 4.28: Monitor for bounded response pattern

4.4 Correctness of Monitor Automata

In this Section, we formally prove that our monitor automata are correct by providing sufficient and necessary conditions. The proofs are constructive guaranteeing the lack of false negatives, false positives and blocking conditions. We also provide monitors that are buggy despite looking intuitive.

Let assume a hybrid automaton model H and a monitor automaton M. Our goal is to show that H satisfies a property ϕ if and only if the *error* location is unreachable in the automaton produced by the parallel composition $H||M$. We prove correctness of the monitor M by showing that all the violating runs of H have corresponding runs in $H||M$ that reach the error location, and vice versa.

4.4.1 Preliminaries

To improve the proofs readability, we use a shorthand notation. We denote a run by the sequence $(\ell_i, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)_{i=0,\dots,N-1}$. Given a run r and an event-time $\tau^* = (k^*, t^*) \in \mathrm{dom}(r)$, the run can be divided into the *prefix* up to τ^*, and the *postfix* after τ^*. The prefix is enlarged by a silent transition τ, which can be injected anywhere (by definition):

$$\mathrm{prefix}\,(r, (k^*, t^*)) = (\ell_i, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)_{i=0,\dots,k^*-1}; (\ell_{k^*}, \mathbf{x}_{k^*}, t^* - t_{k^*-1}, \xi_{k^*}, \tau), \tag{4.1}$$

$$\begin{aligned}\mathrm{postfix}\,(r, (k^*, t^*)) = \big(\ell_{k^*}, r(k^*, t^*), \delta_{k^*} - t_{k^*-1}, \xi^*, \alpha_{k^*}\big); \\ (\ell_i, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)_{i=k^*+1,\dots,N-1},\end{aligned} \tag{4.2}$$

where $r(k^*, t^*) = \xi_{k^*}(t^* - t_{k^*-1})$, and $\xi^*(t) = \xi_{k^*}(t - t_{k^*-1})$ is the trajectory $\xi_{k^*}(t)$ shifted backwards in time by t_{k^*-1}. Similarly, the *infix* between event-times $\tau_a = (k_a, t_a) \in \mathrm{dom}(r)$, $\tau_b = (k_b, t_b) \in \mathrm{dom}(r)$, with $\tau_a \leq \tau_b$, is

$$\mathrm{infix}\,(r, (k_a, t_a), (k_b, t_b)) = \mathrm{prefix}\,(\mathrm{postfix}\,(r, (k_a, t_a)), (k_b - k_a, t_b - t_a)). \tag{4.3}$$

It is obvious that the concatenation

$$\mathrm{prefix}\,(r, \tau) \;;\; \mathrm{postfix}\,(r, \tau)$$

is a run of H. Similarly, the concatenation

$$\mathrm{prefix}\,(r, \tau_a) \;;\; \mathrm{infix}\,(r, \tau_a, \tau_b) \;;\; \mathrm{postfix}\,(r, \tau_b)$$

is a run of H. Abusing slightly the notation[1], we write $r \times \ell^*$ to define the run $(\ell_i \times \ell^*, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)_{i=0,\dots,N-1}$. This is not necessarily a run of $H||M$, but it can be one under the following condition.

Lemma 4.1. *Let $r = (\ell_i, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)$ be a run of H. If a location ℓ_M in M does not have any (i) invariant constraints, or (ii) urgent outgoing transitions, then $r \times \ell_M$ is a run of $H||M$.*

Lemma 4.2. *Let $r = (\ell_i, \mathbf{x}_i, \delta_i, \xi_i, \alpha_i)$ be a run of H. If a location ℓ_M in M has (i) no invariant constraints, and (ii) one urgent outgoing transition with guard condition p which leads to location ℓ'_M and at the same time location ℓ'_M does not have (iii) invariant constraints or (iv) urgent outgoing transitions, then*

$$\text{prefix}\,(r, \tau_{p.1}) \times \ell_M \;;\; \text{postfix}\,(r, \tau_{p.1}) \times \ell'_M$$

is a run of $H||M$, where $\tau_{p.1} = \text{Infi}\,(r, p)$ is the smallest event time for which p holds.

A monitor M is non-blocking if for any run r_H of H, there is a corresponding run $r_{H||M}$ of $H||M$ so that r_H is the projection of $r_{H||M}$ onto H. In other words, no deadlock can cause an unexpected termination of a run.

4.4.2 Sufficient Conditions

A monitor automaton is correct only if it reaches its error location exactly when the system H violates the given property. Let assume that h be a run of H that violates a property ϕ. First, we show (a) that there exists a run r of $H||M$ that reaches the error location. Second, we show (b) that for any run r of $H||M$ which reaches the error location, the projection of the run onto H violates the property.

Absence. Since $r \not\models \phi$, there exist $\tau_p, \tau_q \in \text{dom}(r)$ with $q(r(\tau_q))$, $\tau_p \geq \tau_q$, and $p(r(\tau_p))$.

Using Lemma 4.1 and the jump definition,

$$\text{prefix}\,(h, \tau_q) \times \text{idle} \;;\; \text{infix}\,(h, \tau_q, \tau_p) \times \text{loc1} \;; \text{postfix}\,(h, \tau_p) \times \text{error}$$

is a run of $H||M$.

[1] More details on the composition can be found at Section 2.1; note that here the symbol $\times$ does not refer to a cartesian product.

Absence (timed). Since $r \not\models \phi$, there exist $\tau_q, \tau_p \in \mathrm{dom}(r)$ with $q(r(\tau_q))$, $d(\tau_q, \tau_p) \geq T$, and $p(r(\tau_p))$.

M can remain in idle location during prefix (h, τ_q), then transition to loc1 and remain there during infix (h, τ_q, τ_p). Due to Lemma 1, M can then transition to loc2. In loc2, M can transition to error, since p holds.

Universality. The proof is analogous to the proof of the absence pattern replacing p by $!p$.

Minimum duration. $r \not\models \phi$, so there is $\tau_p \geq \tau_q$ with $p(r(\tau_p^*))$ and $q(r(\tau_q))$, and one of the following conditions is true:

(a) there are $\tau_p, \tau'_{\bar{p}}$ with $\tau_{q.1} \leq \tau_p < \tau'_{\bar{p}}$, $p(r(\tau_p))$, $\neg p(r(\tau'_{\bar{p}}))$, and $d(\tau_{q.1}, \tau_{\bar{p}}) \leq T$, or

(b) there are $\tau_p, \tau_{\bar{p}}, \tau'_{\bar{p}} \in \mathrm{dom}(r)$ with $\tau_{q,1} \leq \tau_{\bar{p}} < \tau_p < \tau'_{\bar{p}}$, $p(r(\tau_p))$, $\neg p(r(\tau_{\bar{p}}))$, $\neg p(r(\tau'_{\bar{p}}))$, and $d(\tau_{\bar{p}}, \tau'_{\bar{p}}) \leq T$.

In case (a), let $\tau_{p.1} = \mathrm{first}\,(r, \tau_{q.1}, p)$, so $\tau_{p.1} \leq \tau_p$. M can remain in idle location during prefix $(h, \tau_{q.1})$, then take the transition to loc1 and due to Lemma 2 remain there during infix $(h, \tau_q, \tau_{p.1})$. Then, M can take the transition to loc2, setting t to zero (Lemma 1). M can remain in this location during infix $\left(h, \tau_p, \tau'_{\bar{p}}\right)$. Since $d(\tau_{q,1}, \tau'_{\bar{p}}) \leq T$, we have $t \leq T$. Then, M can take the transition to error.

In case (b), we initally show that M can be at loc1 at $\tau_{\bar{p}}$. After τ_q, M can move to loc2 as soon as p is satisfied, and return back to loc1 as soon as p is violated. Consequently, we can assume that M can be loc1 at $\tau_{\bar{p}}$. We match the remainder of the run in analogy to the previous case (a), replacing $\tau_{q.1}$ by $\tau_{\bar{p}}$.

For the remaining proofs, we only show the differences with the previous proofs, i.e. we ignore what happens before τ_q and τ_p.

Maximum duration. $r \not\models \phi$ implies that

(i) there is $\tau_p \geq \tau_q$ with $r(\tau_p)$ satisfying p and $r(\tau_q)$ satisfying q, and

(ii)(a) there is τ'_p with $p(r(\tau'_p))$ and $d(\tau_p, \tau'_p) \geq T$, and

(ii)(b) there is no $\tau_{\bar{p}}$ such that $\neg(p(r(\tau_{\bar{p}}))$ and $\tau_p < \tau_{\bar{p}} < \tau'_p$.

At τ_p, M can be in loc1 or loc2. In loc1, M can transition to loc2, as p holds. Once reached loc2, M can wait there for T time units, since with (ii)(a) and (ii)(b), p still holds. M can then take the transition to error.

Bounded invariance. $r \not\models \phi$ implies that

(i) there is $\tau_p \geq \tau_q$ with $r(\tau_p)$ satisfying p and $r(\tau_q)$ satisfying q, and

(ii) there is a τ with $\tau \geq \tau_p$ such that $d(\tau_p, \tau) < T$ and $s(r(\tau))$ is false.

At τ_p, the monitor M can be in loc1 or in loc2. If it is in loc1, M can immediately transition to loc2 (because p is true). Once it is loc2, M can wait for $t = d(\tau_p, \tau)$ there. Since $\mathrm{d}(\tau_p, \tau) < T$, the monitor can remain in loc2 until τ. Since $\neg s$ at τ ($\neg s(r(\tau)$ holds), the monitor can transition to error.

Bounded recurrence. For the unbounded case (i), there is $\tau_p \in \mathrm{dom}(r)$ with $p(r(\tau_p))$ and $\tau_p \geq \tau_q$, such that there is no $\tau'_p > \tau_p$, with $d(\tau'_p, \tau_p) \leq T$ and $p(r(\tau'_p))$ (τ_p's having distance less than T).

If the monitor M is in loc1 at τ_p, it can then transition from loc1 to loc2. If the monitor is in loc2, there are two subcases:

(a) p doesn't hold within T time units after τ_p, in that case M can go to loc1, wait there for more than T time and then jump to error.

(b) p holds after τ_p, which signifies that it holds at time τ'_p with $d(\tau_p, \tau'_p) > T$. Let $\delta = d(\tau_p, \tau'_p) - T$. At that point, M can wait for $\delta/3$ time in loc2, after which the predicate p is false. Afterwards, M can go to loc1, wait for $T + \delta/3$ time, and because only $T + 2\delta/3$ time units have passed since τ_p, it means p is still false. Since $t > T$, M can go to error.

Bounded response (persisting). For the unbounded time horizon, $r \not\models \phi$ implies that

(i) there is $\tau_p \geq \tau_q$ with $r(\tau_p)$ satisfying p and $r(\tau_q)$ satisfying q, and

(ii) there is no $\tau_s \geq \tau_p$ such that $r(\tau_s)$ satisfies s, $d(\tau_p, \tau_s) \leq T$ and persists (r, τ_s, s).

At τ_p, the monitor M can be either in loc1, loc2 or loc3. In loc1, the monitor can transition immediately to loc2 (because p is true). In loc2, two scenarios are possible. If s is false ($\nexists \tau_s : d(\tau_p, \tau_s) \leq T$), M can stay in loc2 for more than T time units. Then, M can transition to error. In case s is not always false, there is a τ_s so that $\neg$persists (r, τ_s, s). At τ_s, M can instantaneously move to loc3 and then back to loc2 when s doesn't hold. From loc3, if $\neg s$, M can transition to loc2. If s and $\neg$persists (r, τ_s, s), M can take the transition to loc2, since $\neg$persists $(r, \tau_s, s) : \exists \tau'_s > \tau_s$ with $d(\tau_s, \tau'_s) = 0$ and $\neg s(\tau'_s)$.

For the bounded time horizon, $r \not\models \phi$ implies that

(i) there is $\tau_p \geq \tau_q$ with $r(\tau_p)$ satisfying p and $r(\tau_q)$ satisfying q, and

(ii) $\tau_q, \tau_p \in \mathrm{dom}_{-T}(r)$ and there is no $\tau_s \in \mathrm{dom}(r)$ such that $\tau_p \leq \tau_s$, $d(\tau_p, \tau_s) \leq T$ and $\mathrm{persists}\,(r, \tau_s, s)$.

The proof for the bounded case is similar to the unbounded one.

4.4.3 Necessary Conditions

For the necessary conditions, we need to show that a run r in $H||M$ that ends in location error implies the existence of a run in H that violates the property. Let r_H be the projection of the run to H (by removing the locations and clocks of M). It is clear that r_H is a run of H. In the following part, we show that $r_H \not\models \phi$. Note that r starts in idle location and any event-times of r are also event-times of r_H.

Absence. To get from idle to error, M first had to take a transition with guard q and afterwards a transition with guard p. Consequently, there exist τ_q and τ_p with $\tau_q \leq \tau_p$, $q(r_H(\tau_q))$ and $p(r_H(\tau_p))$. τ_p and τ_q are witnesses that violate ϕ.

Absence (timed). To get from idle to error, M first had to take a transition with guard q, wait for T time units, and afterwards take a transition with guard p. Consequently, there exist τ_q and τ_p with $d(\tau_q, \tau_p) \geq T$, $q(r_H(\tau_q))$ and $p(r_H(\tau_p))$. τ_p and τ_q are witnesses that violate ϕ.

Universality. The proof is analogous to the proof of the absence pattern replacing p by $!p$.

Minimum duration. In a similar way to the above proofs, we can stipulate the existence of τ_q, τ_p and $\tau'_{\bar{p}}$ with $\tau_q \leq \tau_p \leq \tau'_{\bar{p}}$, $q(r_H(\tau_q))$, $p(r_H(\tau_p))$ and $\neg p(r_H(\tau'_{\bar{p}}))$. τ_p and τ_q are witnesses that violate case (i).

For case (ii), let $\tau_{q.1} = \mathrm{first}\,(r, 0, q)$, so that $\tau_{q.1} \leq \tau_q$. Without loss of generality, we can presume that τ_p is the last event-time on r when M entered loc2, so $t = d(\tau_p, \tau'_{\bar{p}})$. Due to the transition guard from loc2 to error, $d(\tau_p, \tau'_{\bar{p}}) \leq t \leq T$. There are two subcases:

(a) If there is no $\tau_{\bar{p}}$ with $\tau_{q.1} \leq \tau_{\bar{p}} \leq \tau_p$ and $\neg p(r_H(\tau_{\bar{p}}))$, we can conclude that $\tau_{q.1} = \tau_{p.1}$, where $\tau_{p.1} = \mathrm{first}\,(r, \tau_{q.1}, p)$. In this case, the run in M goes from idle to loc1 to loc2, so $\tau_{q.1} = \tau_{p.1} = \tau_p$. As a rsult, $d(\tau_{q.1}, \tau'_{\bar{p}}) = d(\tau_p, \tau'_{\bar{p}}) \leq T$, violating case (a).

(b) Otherwise, we have $\tau_{q.1} \leq \tau_{\bar{p}} \leq \tau_{p.1} \leq \tau'_{\bar{p}}$. We will manifest that there is a $\tau^* \leq \tau_p$, with $d(\tau^*, \tau_p) = 0$ and where $r(\tau^*)$ violates p. Then $d(\tau^*, \tau'_{\bar{p}}) \leq T$, violating case (b). Now, we show the existence of τ^*, by first recognizing some $\tau' \leq \tau_p$ so that M is in loc1 for all $\tau' \leq \tau \leq \tau_p$, and for which $r(\tau')$ violates p. Consider that we can presume that loc1 was reached either from idle with p being violated (otherwise case (a) applies), or from loc2, which p also being violated. As the transition from loc1 to loc2 is urgent, p cannot hold for any τ with $\tau' \leq \tau < \tau_p$ where $d(\tau, \tau_p) > 0$ (no time can pass while p is true). So there exists a τ^* with $\tau' \leq \tau^* \leq \tau_p$ with $d(\tau^*, \tau_p) = 0$.

Maximum duration. We assume τ_p to be the last event-time on r when M entered loc2. Since the loc2 has invariant p and the transition guard from loc2 to error has the constraint $t \geq T$, we get that at least T time units have passed in loc2. In this vein, there exist τ_p and τ'_p so that $d(\tau_p, \tau'_p) \geq T$ without any $\tau_{\bar{p}}$ in between them. Consequently, τ_q, τ_p and the absence of $\neg p$ witnesses the violation of ϕ.

Bounded invariance. We assume that M is in the error location. Because of the guard conditions of the incoming transitions, we know that at some point τ on the run, s didn't hold. In loc2, we know, from its invariance and the incoming guard conditions, that there is p which held at some point τ_p with $\tau_p \leq \tau$ and $d(\tau_p, \tau) < T$. Consequently, τ_q, τ_p, and τ witness the violation of ϕ.

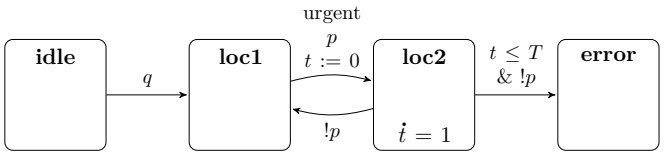
Bounded recurrence. Given the fact time can only elapse in loc2 while $\neg p$ and t is reset on all incoming transitions, we conclude that $\neg p$ holds for more than T time units; which in turn violates the property.

Bounded response (persisting). Similarly to the absence pattern proofs, we can pinpoint the existence of τ_q and τ_p. Let assume that τ_p is the last event-time on r when M entered loc2. Cycles between loc2 and loc3 last zero time: due to the urgent transition from loc2 to loc3, s had to be false during this time, with the only possible exception of switching to true and back to false in zero time (which doesn't comply with the definition of "persists"). Since the transition guard from loc2 to error contains the constraint $t > T$, we can deduce that more than T time units have elapsed in loc2. Consequently, τ_q, τ_p and the absence of s witness the violation of ϕ.

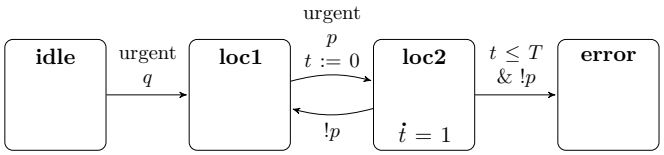
4.4.4 Buggy Monitors

Proving the correctness of the monitors is not easy. In fact, it is an arduous task and it required a lot effort despite collaborating with our academic and industrial partners. Several seemingly equivalent monitors are not correct. Below, we present some examples.

Minimum duration. *"After q, it is always the case that once p becomes satisfied, it holds for at least T time units."*



a) *Incorrect monitor*



b) *Correct monitor*

Figure 4.29: Two seemingly equivalent monitors for the minimum duration pattern

To figure out why the monitor of Figure 4.29a is incorrect, consider the trace portrayed in Figure 4.30. The trace on the right provides a counterexample. Although this trace does not violate the minimum duration property (p holds for less than T time units), the buggy monitor goes to the error location. This happens due to the lack of an urgent guard q from location idle to location loc1.

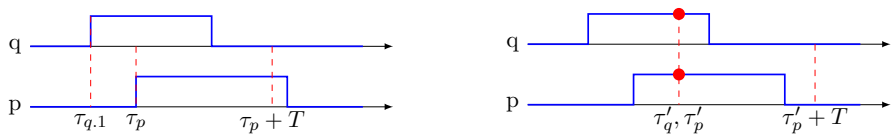
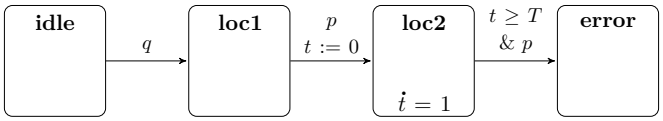
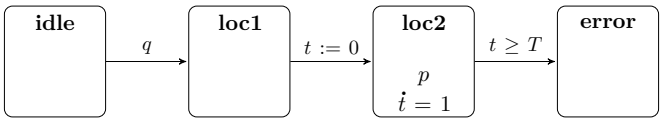


Figure 4.30: Buggy monitor for minimum duration pattern - Traces: satisfied for $\tau_{q.1}$ (left), violated for τ'_q(right).

Maximum duration. *"After q, it is always the case that once p becomes satisfied, it holds for less than T time units."*



a) *Incorrect monitor*



b) *Correct monitor*

Figure 4.31: Two seemingly equivalent monitors for the maximum duration pattern

To figure out why the monitor of Figure 4.31a is incorrect, consider the traces portrayed in Figure 4.32. The trace on the right of the Figure violates the maximum duration property; both correct and buggy monitors arrive at the the error location. However, the trace on the left of Figure acts as a counterexample. Although this trace satisfies the maximum duration property (indeed, p holds for less than T time units), the buggy monitor reaches the error location. This happens due to the lack of an invariant in location loc2.

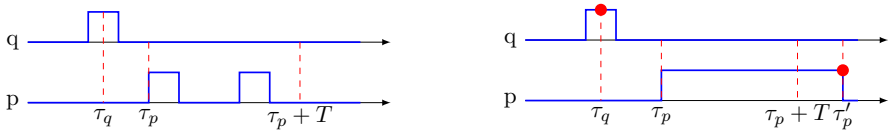
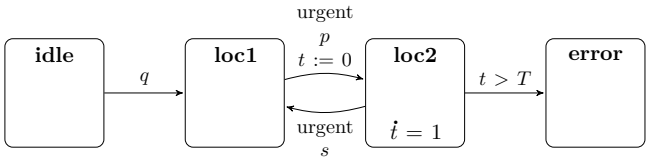


Figure 4.32: Maximum duration pattern. Satisfying trace (left): correct monitor does not go to error location but buggy monitor does; Violating trace (right): both correct and buggy monitor go to the error location.

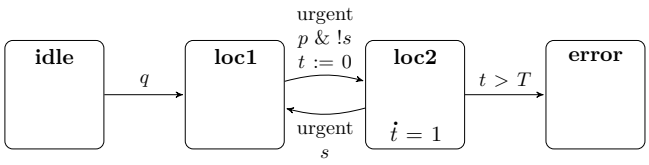
Bounded response (persisting). *"After q, it is always the case that if p holds, then s persists (holds for nonzero time) after at most T time units."*

The problem with the persisting case (monitors: version 1 and version 2) can be observed if there is a trace when p & !s hold at the same time. In that case the trace is automatically accepted, although s should hold for

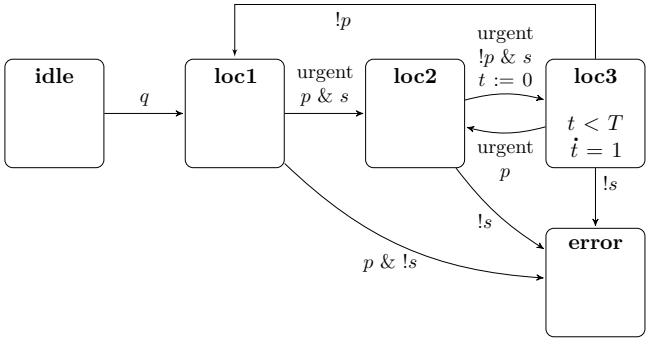
non-zero time. The problem with the non-persisting case is that traces with instantaneous (zero-time) s traces are accepted although they should not.



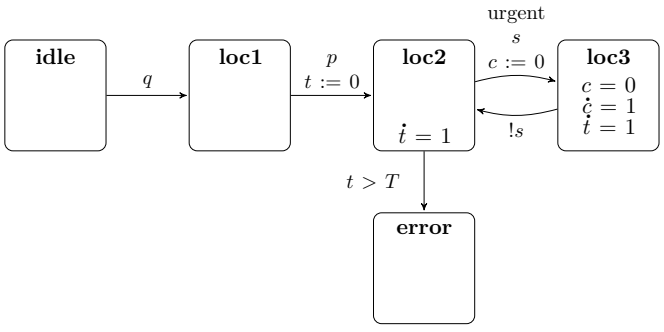
a) *Incorrect monitor for bounded response pattern (persisting - version 1)*



b) *Incorrect monitor for bounded response pattern (persisting - version 2)*



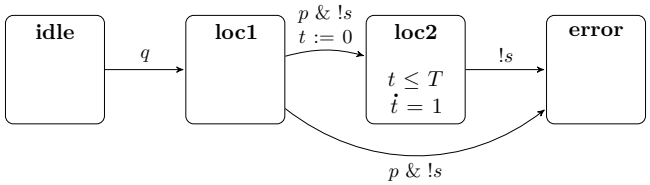
c) *Incorrect monitor for bounded response pattern (non-persisting)*



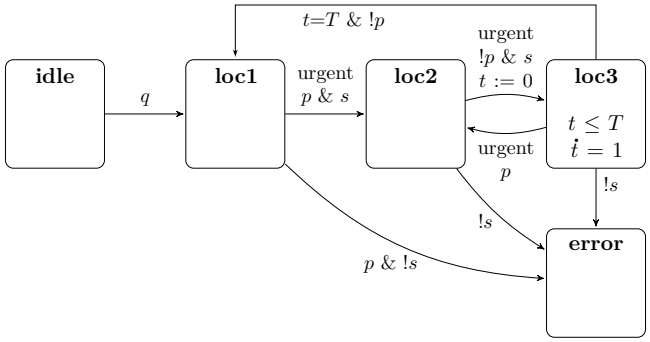
d) *Correct monitor for bounded response pattern*

Figure 4.33: Seemingly equivalent monitors for the bounded response pattern

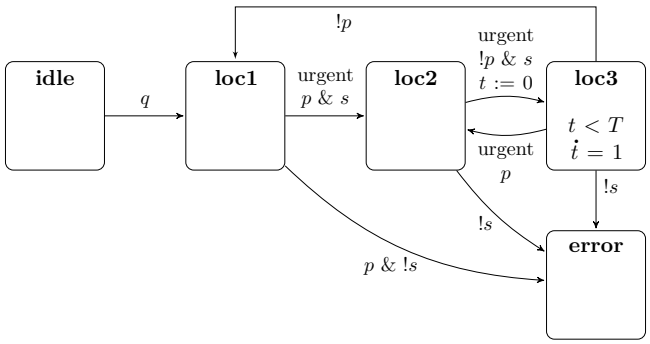
Bounded invariance. *"After q, it is always the case that if p holds, then s holds for at least T time units."*



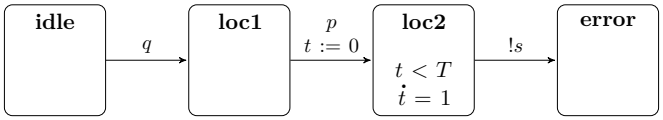
a) *Incorrect monitor for bounded invariance pattern (version 1)*



b) *Incorrect monitor for bounded response pattern (version 2)*



c) *Incorrect monitor for bounded invariance pattern (version 3)*



d) *Correct monitor for bounded invariance pattern*

Figure 4.34: Seemingly equivalent monitors for the bounded invariance pattern

4.5 Application Example

In this Section, we illustrate the workflow on an industrial use case on electro-mechanical brakes (EMB) and highlight how the introduced pattern templates and associated monitor automata can facilitate the verification process. The EMB use case is described in [281]. The requirements that shall be enforced are presented in [130]. The steps of the proposed workflow are as follows.

Industrial Model. The model is designed with Simulink. It consists of an experimental electro-mechanical braking system, a feedforward and a feedback controller.

Formal Model. The Simulink to SpaceEx (SL2SX) translator [231] is used to construct the formal model. The model is expressed in the SpaceEx format [125] and it consists of 8 base components (single HA) and 4 network components (networks of HA). General, nonlinear Simulink systems can be transformed to SpaceEx models in the form of piecewise affine hybrid automata, as shown in [187] and [188].

Specifications. Two braking specifications are provided in [130].

1. "The caliper must reach $x_0 = 0.05$ dm after the braking request is issued within 20 ms with a precision of 4%".
   This property can be mapped to the **bounded response** pattern, where $T := 20$, $q := true$, $p :=$ true (braking request), $s := \{0.96 \cdot x_0 \leq x\}$ and x represents the caliper position.
2. "The caliper speed at contact must be below 2 mm/s".
   This property can be mapped to the **absence** pattern, where $q := true$, $p := \{v \geq 2\}$, and v represents the caliper speed.

Monitor. For the first specification, we use the corresponding monitor automaton for the bounded response pattern. The monitor is generated with formalSpec tool [76]. The formal model and the monitor are expressed as hybrid automata and comply with SpaceEx format.

Composition. This step corresponds to the parallel composition of the formal model with the corresponding monitor. In essence, the variables that appear in the monitor should be connected with the corresponding variables of the formal model. In our case, only the caliper position x should be considered. The variables t and c are local and only used inside the

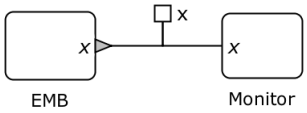


Figure 4.35: Composition of the formal model (EMB) with the monitor automaton (bounded response), shown in SpaceEx Model Editor.

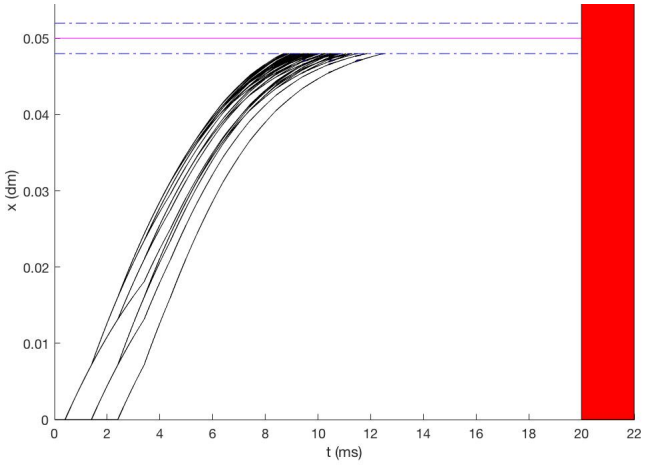


Figure 4.36: Reachable sets of the caliper position computed with SpaceEx.

monitor automaton. Figure 4.35 shows the composed system in the Model Editor [125].

Reachability Analysis. SpaceEx [135] is used for computing the reachable sets. The safety verification problem is tackled by introducing a set of error states and checking whether they are reachable or not. In practice, we check if "loc(Monitor)==error". SpaceEx finds a fixed point after 434 iterations and 31.702s. The computation time for the same model without the monitor is 29.427s. As such, the induced overhead is around 7.73%. Note that SpaceEx composes (flattens) the model on-the-fly during reachability analysis.

Verification Outcome. The error state is not reachable and the property is satisfied. The reachable set for the caliper position x is portrayed in Figure 4.36.

4.6 Related Work

Conducting hybrid system verification against rich formal specifications is a challenging problem. The authors in [99] studied the topological aspects of hybrid systems in the context of propositional modal μ-calculus. Mysore, et al., studied the verification problem of semi-algebraic hybrid systems for TCTL (Timed Computation Tree Logic) properties and proved undecidability [234]. Jeannin and Platzer presented in [176] a differential temporal dynamic logic to specify temporal properties of hybrid systems. This logic complemented with a theorem prover could enable verification of nested temporalities for hybrid systems. The authors in [87] studied the verification of hybrid systems with K-liveness but restricted the system model to a small subclass of hybrid automata.

In this work, we employ a template language to express informal requirements rather than a declarative language based on temporal logic. Signal Temporal Logic, introduced in [221, 222], can describe properties of hybrid systems but is mostly used for monitoring hybrid behaviors [236]. Hybrid extensions of LTL can be found in the liteature, e.g. HyLTL [69, 70] and HRETL [88, 89]. Both, however, utilize untimed operators.

The use of pattern templates for system requirements and their (semi–) automatic translation to formal specifications have been proposed earlier. Dwyer et al. [114] were among the first to introduce qualitative specification pattern templates and their translation into different logic expressions. Among others, Konrad and Cheng [196] extended Dwyer's original patterns to the real–time domain. Application of the patterns in the automotive industry can be found in [182, 254]. A generalization to probabilistic pattern templates was proposed in [153]. For discrete systems, there are tools that accept as an input CNL expressions (e.g. in the form of pattern templates) and automatically translate them into formal specifications. Examples of such tools are Stimulus [28], Embedded Specifier [74], AutoFocus3 [173], and SpeAR [57]. Pattern templates for hybrid systems do not differ from pattern templates already available. Yet, no existing tool can translate them into a formal representation that is applicable to hybrid systems and enables the verification of rich properties.

CHAPTER 5

From Simulation Models to Formal Models

Parts of this Chapter have been published in [187].

Model transformation plays an essential role in bridging the gap between industrially relevant models and verification tools [24]. In model-based design, the physical plant and its controllers are synthesized based on a model, usually within a simulation environment, e.g. MATLAB/Simulink [227, 237]. The presence of nondeterminism is inherent in every dynamical system and can take the form of disturbances, parameter uncertainties, user inputs, measurement noises, or operating conditions. All these nondeterministic elements may have adverse effects on performance and correctness, which can be difficult to predict and analyze in the design step. Consequently, the designer conducts extensive testing by simulating a large number of trajectories, each with a different choice for these nondeterministic quantities, and checking if they satisfy the requirements.

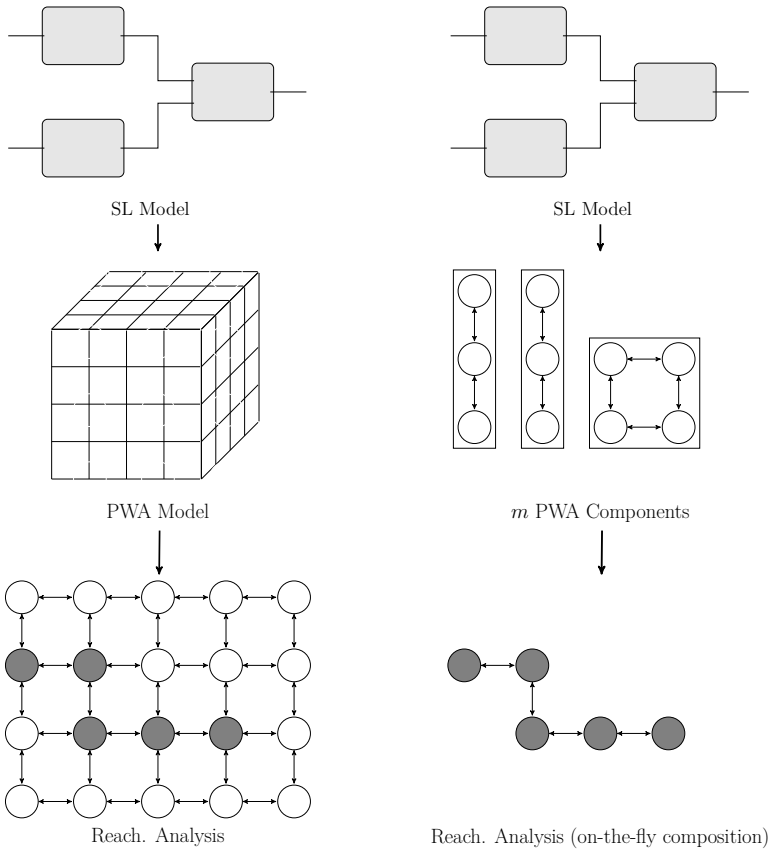
However, this process is incomplete as the number of different choices is prohibitively large or even infinite. As such, it can be hard to say with high confidence if a requirement is really satisfied under all circumstances. Formal verification aims to guarantee that the system requirements are truly satisfied via a rigorous mathematical analysis of the system. A commonly used verification method is set-based reachability analysis, see Section 2.3 for more information.

There are two main challenges to applying reachability analysis in model-based design. Firstly, the simulation model has to be converted to a suitable formal model, e.g. a hybrid automaton, a hybrid petri net, or any language with formal semantics. Secondly, the model has be amenable to existing reachability algorithms, particularly in terms of scale. Highly scalable algorithms exist for piecewise affine (PWA) dynamics but not for complex nonlinearities [111, 218]. Even though a large class of nonlinearities could be approximated arbitrarily well by a PWA system, there might be scalability problems owing to the resulting models being very large with numerous locations.

In this Chapter, we propose an approach to transform a simulation model into a compact, i.e. relatively small verification model described with PWA dynamics. To accomplish this, we decompose the nonlinear system and perform the transformation component-wise. The resulting model can be fed to the SpaceEx verification [135] or converted into formats for other verification tools with the HyST tool [40]. Due to the fact that SpaceEx composes the model on-the-fly during the reachability computations, only the reachable partitions of the PWA approximations are instantiated.

Figure 5.1 displays the difference between classical hybridization techniques and the proposed one; in case the original nonlinear model is designed with Simulink. Traditional hybridization schemes that rely on state-space partitioning [33] create a PWA model with $\mathcal{O}(1/\ell^n)$ locations, where n is the dimension of the state-space and ℓ the mesh size. During the reachability analysis, $\mathcal{O}(T/\delta)$ locations are visited, where T is the global time horizon and δ the minimum dwell time. On the other hand, syntactic hybridization generates m PWA components, where m is the number of nonlinearities, and the total number of locations is $\mathcal{O}(m/\ell^2)$. The locations that are not reachable do not have to be instantiated.

The rest of this Chapter is organized as follows. In Section 5.1, we introduce the compositional syntactic hybridization, present the underlying theory, and provide a step-by-step guide. In Section 5.2, we propose a model transformation methodology to obtain formal models (expressed with hybrid automata) out of Simulink/MATLAB models. A simple Simulink model is used as a running example. Section 5.3 is devoted to Stateflow models and how they can be mapped to hybrid automata. In Section 5.4, we propose a heurisitic method to address the semantic mismatch between Simulink/Stateflow models and hybrid automata, i.e. that the former have urgent transitions while the latter non-urgent transition and apply this method to several examples. Finally, in Section 5.5, we present the related literature.



a) *Traditional Hybridization* **b)** *Syntactic Hybridization*

Figure 5.1: Constructing verification models from a Simulink (SL) system, with $m = 3$ nonlinear blocks. The syntactic approach can lead to approximations of smaller size and less instantiated locations.

5.1 Compositional Syntactic Hybridization

Hybridization is an abstraction method to obtain PWA over-approximations of nonlinear dynamics [33]. Hybridization schemes involve three main steps: (i) partitioning the state-space into domains, (ii) in each domain approximating the nonlinear dynamics by simpler dynamics, and (iii) compute the abstraction error and add nondeterministic inputs to capture it. Classical hybridization methods operate on the flattened system, causing the number of partitions to be exponential in the number of state variables. This approach quickly leads to the construction of intractably large models, even in the case of small systems. Compositional syntactic hybridization performs the PWA over-approximation in a compositional manner and takes advantage of the on-the-fly composition of hybrid systems supported by the

SpaceEx platform. Three main steps are required: *syntactic decomposition*, substituting the nonlinear ODE by a linear ODE and a set of nonlinear algebraic equations; *hybridization*, constructing a PWA approximation for each nonlinear algebraic equation and providing an over-approximation of the original system by adding an error term; and eventually *composition* of the hybrid automata and elimination of the algebraic equations.

5.1.1 Syntactic PWA Approximation

In this part, we explain the technical details, considering a nonlinear differential equation of the form

$$\frac{dx}{dt} = f(x), \qquad x \in \mathbb{R}^n. \tag{5.1a}$$

This ODE is considered to be regular (f is a Lipschitz function of constant $L > 0$ over the state-space $\mathcal{X} \subset \mathbb{R}^n$). This method can be readily extended to semi-explicit differential-algebraic equations (DAEs). Under certain assumptions it could be applied to ODEs with non-uniform (local) Lipschitz continuity. In principle, the idea is to encapsulate the solution of the ODE in a set of solutions of differential inclusions.

Syntactic Decomposition

The decomposition process consists in generating a new system where auxiliary variables replace the nonlinear terms,

$$\frac{dx}{dt} = g(x, y), \qquad y \in \mathbb{R}^m, \tag{5.2a}$$

$$y = h(x, y). \tag{5.2b}$$

Here, y is a vector of auxiliary variables, $g(x, y) \in \mathbb{R}^n$ is linear both in x and y and $h(x, y) \in \mathbb{R}^m$ contains all the nonlinear terms, m, of the original system, as discussed below. Note that we have replaced the original system by a linear ODE of a higher-dimensional space, $\mathbb{R}^{n+m}$, associated with a set of nonlinear algebraic constraints. This step is exact.

Moreover, let $V_i \subseteq \{x_1, \ldots, x_n\}$ for $i \in \{1, \ldots, m\}$ be the variables involved in the i-th nonlinearity and $p_i = |V_i|$ define the number of variables in such expression. Note that with a sufficient number of auxiliary variables, we can always assume that $h_i(x, y)$ satisfies $1 \leq p_i \leq 2$ for all i. Examples 5.1.1 and 5.1.2 could provide further insights for this step.

PWA Approximation

We consider a set of *domains*, R_{ij}, that cover entirely the operating range of the variables in V_i, with j a label referring to each individual domain. For each R_{ij}, we carry out a PWA linearization of h_i. Hence, (5.2a)-(5.2b) is replaced by

$$\frac{dx}{dt} = g(x, y), \qquad y \in \mathbb{R}^m, \tag{5.3a}$$

$$y = \hat{h}(x, y), \tag{5.3b}$$

where $\hat{h}$ forms a vector of PWA functions.

Let assume that *op* denotes the operating point in the domain R_{ij}. Using Taylor's formula with the Lagrange remainder, for each $1 \leq i \leq m$,

$$\hat{h}_i(x, y) = h_{i,op} + \frac{\partial h_i}{\partial x}\Big|_{op}(x - x_{op}) + \frac{\partial h_i}{\partial y}\Big|_{op}(y - y_{op}), \tag{5.4}$$

and

$$h_i(x, y) - \hat{h}_i(x, y) = \frac{1}{2}(x - x_{op})^{\mathrm{T}} \frac{\partial^2 h_i}{\partial x^2}\Big|_{\xi}(x - x_{op}) + \tag{5.5}$$
$$\frac{1}{2}(y - y_{op})^{\mathrm{T}} \frac{\partial^2 h_i}{\partial y^2}\Big|_{\xi}(y - y_{op}) + (x - x_{op})^{\mathrm{T}} \frac{\partial^2 h_i}{\partial x \partial y}\Big|_{\xi}(y - y_{op}),$$

where $\xi = (\xi_x, \xi_y) \in \mathbb{R}^{n+m}$ is a point inside the interval $\xi_x \in \{x_{op} + a(x - x_{op}), a \in [0, 1]\}$ and similarly for ξ_y. The right-hand side of the Equation(5.5) is the Lagrange remainder; its resulting values in each domain R_i are used to estimate the error induced by the approximation process [55]. The linearization errors ϵ_h are computed by evaluating the Lagrange remainder and they satisfy $y = h(x, y) \in \hat{h}(x, y) \oplus \mathcal{B}\epsilon_h$, where $\mathcal{B}$ is the unit ball in a chosen norm $||\cdot||$. In this work, we assume that R_i are boxes. In this case, the point that minimizes the absolute value of the Lagrange remainder is known to be its center [17]. Several interesting alternatives exist, such as simplices [33].

Example 5.1.1. Consider the fourth-order polynomial function $f = [x_1 - x_2x_3x_4, x_1x_2 - x_4, -x_3x_4, x_2 - x_3]$. We introduce the auxiliary variables $y_1 = x_3x_4$, $y_2 = x_1x_2$ and $y_3 = x_2y_1$. As a result, $g(x, y) = [x_1 - y_3, y_2 - x_4, -y_1, x_2 - x_3]$ becomes a linear ODE and $h(x, y) = [x_3x_4, x_1x_2, x_2y_1]$ a second-order nonlinear algebraic equation with $m = 3$ elements. Consider a PWA approximation, $\tilde{f}$, which performs a rectangular partitioning of the state-space, with elements of size ℓ in each dimension. Then, $\tilde{f}$ leads to

$O(1/\ell^4)$ elements, whereas the PWA approximation of h only to $O(m/\ell^2)$ elements.

Example 5.1.2. In this example, we consider a 9-dimensional biological model modified from the one in [84, 193]. The model is described by $f(x) = [3x_3 - x_1x_6, x_4 - x_2x_6, x_1x_6 - 3x_3, x_2x_6 - x_4, 3x_3 + 5x_1 - x_5, 5x_5 + 3x_3 + x_4 - x_6(x_1 + x_2 + 2x_8 + 1), 5x_4 + x_2 - 0.5x_7, 5x_7 - 2x_6x_8 + x_9 - 0.2x_8, 2x_6x_8 - x_9]$. We introduce the auxiliary variables $y_1 = x_1x_6$, $y_2 = x_2x_6$, $y_3 = x_6x_8$. As such, $g(x, y) = [3x_3 - y_1, x_4 - y_2, y_1 - 3x_3, y_2 - x_4, 3x_3 + 5x_1 - x_5, 5x_5 + 3x_3 + x_4 - y_1 - y_2 - 2y_3 - x_6, 5x_4 + x_2 - 0.5x_7, 5x_7 - 2y_3 + x_9 - 0.2x_8, 2y_3 - x_9]$ is a linear ODE and $h(x, y) = [x_1x_6, x_2x_6, x_6x_8]$ is a second-order nonlinear algebraic equation with $m = 3$ elements. Consider a PWA approximation, $\tilde{f}$, which employs a rectangular partitioning of the state-space, with elements of size ℓ in each dimension. Then, $\tilde{f}$ leads to $O(1/\ell^9)$ elements, whereas the PWA approximation of h only to $O(m/\ell^2)$ elements. Instead of gridding a 9-dimensional state-space, we only have to approximate with PWA functions three 2-dimensional state-spaces. These examples illustrate the usability of this syntactic approach especially in cases that repeated nonlinearities appear.

5.1.2 Compositional Scheme

The syntactic PWA approximation yields a linear ODE with PWA algebraic constraints. To feed this model into an available reachability tool, we have describe it as a network of hybrid automata. Each hybrid automaton represents to the PWA approximation of one nonlinearity. Each piece of the PWA approximation represents one location in the corresponding automaton.

In the SX/SpaceEx file format [94], see 2.4, a model contains components that are either hybrid automata or networks of hybrid automata. A component can be instantiated inside a network, potentially replacing variables with constant values or remapping them to other variables or. Notice that an ODE or an algebraic constraint can be simply embedded in a hybrid automaton with a single location. The ODE should be placed in the flow-constraint of the location and the algebraic constraint in its invariant.

Expressing the PWA approximation in this setting, the linear ODE is modeled by a single (trivial) automaton. Each PWA constraint $y_i = \hat{h}_i(x, y)$ represents a hybrid automaton with one location per piece. The locations of adjacent pieces are linked through transitions. The approximation error is defined by extra variables with range ϵ_h, and the error threshold $\mu > 0$ gives an upper bound on the maximum value it can obtain (in some chosen norm $||\cdot||$).

Standard reachability algorithms take as input a single hybrid automaton with ODE dynamics. To go from the multi-component input model to this form, the reachability tool needs to carry out two operations. Firstly, it combines the components through a procedure that is called *parallel composition*. Secondly, it eliminates the algebraic constraints to get an ODE. In principle, parallel composition refers to generating the product automaton, whose locations contain the cross-product of the locations of the components. As such, a model with m components of k locations each has a product automaton with m^k locations. Nonetheless, tools like SpaceEx, build the product automaton on-the-fly and instantiate only the reachable locations. In a similar fashion, the conversion from linear DAE to ODE is only performed on the instantiated locations. The conversion can be carried out efficiently by Gauss-Jordan elimination [109]. The underlying theory is reported in [145]. The outlined procedure allows us to approximate the reachable set of the original system with arbitrary precision. Let $\Phi(t, x)$ determine the trajectory starting from x at time t. The reachable set from a set of initial points $X_0 \subseteq \mathcal{X}$ during the interval $[0, t]$ is described as

$$Reach(t, X_0) = \{y = \Phi(\tau, x) : \tau \in [0, t],\ x \in X_0\}. \tag{5.6}$$

This approximate system converges to the original system, as shown in the following theorem.

Theorem 5.1. *(see [32]) The Hausdorff distance between the reachable set of* (5.2a)-(5.2b) *and the reachable set computed via hybridization,* (5.3a)-(5.3b)*, from time* 0 *to a final time* $T > 0$ *satisfies*

$$d_H\big(Reach_f(T, X_0), Reach_{\hat{f}}(T, X_0)\big) \leq \frac{2\mu}{L}(e^{LT} - 1), \tag{5.7}$$

where L *is the Lipschitz constant of the original nonlinear dynamics,* μ *is the error threshold, and* $\hat{f}$ *is the PWA approximation.*

5.1.3 Algorithm for Compositional Syntactic Hybridization

Motivation. In [233], Tiwari et al. present an approach to construct non-uniform multidimensional partitions. Let $n \geq 1$, and consider the vector function

$$f : X \to \mathbb{R}^n, \tag{5.8}$$

in a domain $X \subseteq \mathbb{R}^n$, with f assumed continuous in its domain. The objective is to find a pair of functions, which provides sound under and over

approximations of f, that is,

$$\hat{f}_{lb}(x) \leq f(x) \leq \hat{f}_{ub}(x) \tag{5.9}$$

for all $x \in X$.
Inspired by their approach, we would like to replace their inequalities by inclusions and restrict these functions to be piecewise-linear. This last restriction is motivated by hybridization methods in reachability analysis. Our objective can be more explicitly formulated as follows. If $\hat{f}$ is PWA, then there is a partition of the domain $\bigcup_k P_k = X$ (with the union being finite, and the P_k's pairwise disjoint), and a set of coefficients associated to each P_k, that we write $\{a_{ij}^{(k)}\}$ so that for all $x \in X$ we can write

$$\hat{f}_i(x) = a_{i0} + \sum_{j=1}^{n} a_{ij} x_j \tag{5.10}$$

for all $i \in 1, \ldots, n$.

Moreover, it is generally known that non-compositional techniques for PWA approximations are not computationally efficient for complex systems. This the case because an acceptable accuracy requires a very large number of pieces (locations) in the piecewise affine approximation [124]. Recently, Deshmukh et al. [101] presented an exploratory comparison of a compositional approach, similar to that presented in this Chapter (called nested approximation there), against a simplex-partitioning PWA hybridization, showing that the former scales much better than the latter for increasing demands on precision. The compositional PWA approximation is presented informally, while the paper neither discusses the implications in terms of the model size nor preserves this compositionality in the generated model. The complexity of the approximation can be further decreased by focusing on a set of reference trajectories, as done in [39].

Step-by-Step. Given our goal to automate the whole process, the steps that we undertake approach are the following:

1. We start from an autonomous system (not necessarily homogeneous), of the form
$$\frac{dx}{dt} = f(x), \qquad x \in \mathbb{R}^n. \tag{5.11}$$
Herein, consider nonlinear systems that are continuous and smooth. It can be straightforwardly extended to models with discrete transitions.
2. We create a dictionary to identify the nonlinear terms. In this way, we can replace the original system by an equivalent one with auxiliary

variables and generate a new system in m dimensions, $m \geq n$ in general,

$$\frac{dx}{dt} = g(x, y), \qquad y \in \mathbb{R}^m, \tag{5.12a}$$

$$y = h(x) \tag{5.12b}$$

such that

$$f(x) = g(x, h(x)). \tag{5.13}$$

The main property of g is that each nonlinear term in g contains at most two variables. For instance, there are only terms like $y_1 \sin y_2$, $y_3^2 y_4$, etc. More formally we have in general for each row $1 \leq k \leq n$,

$$g_k(x, y) = g_{k,0} + \sum_{i=1}^{n}\sum_{j=1}^{m} g_{k,ij}^{(1)}(x_i, y_j) + \sum_{i=1}^{m}\sum_{j=1}^{m} g_{k,ij}^{(2)}(y_i, y_j) + \sum_{i=1}^{n}\sum_{j=1}^{n} g_{k,ij}^{(3)}(x_i, x_j) \tag{5.14}$$

3. Then we select our state space partition for each nonlinear term (auxiliary) in the corresponding variables (which may be up to 2). This involves two steps.

   Firstly, we need the maximum and minimum values of these variables. These could either be introduced by the user (i.e. if they represent physical variables) or computed/over-approximated numerically. Three options are currently available (corner case simulations, star simulations and pseudo random simulations with Breach toolbox). It is also possible to use the initial conditions to derive estimated future values for a given time window.

   Secondly, we should select the partition type, the quantization parameter (for each variable) and the set representation. The simpler option is to use uniform fixed gridding, with different quantization parameters for each variable and set the domains to be boxes/hyper-rectangles.

4. Once, we have selected the linearization domains and we can construct the linear approximation for each domain. This can be done either either by first order Taylor series or symbolically. The linearization/operating points are opted to be the centers of the boxes. In this way, we produce

a new system

$$\frac{dx}{dt} = \hat{g}(x, y), \qquad y \in \mathbb{R}^m, \tag{5.15a}$$

$$y = \hat{h}(x) \tag{5.15b}$$

where both $\hat{h}$ and $\hat{g}$ is a piecewise-affine function.

5. Errors. The idea is that we perform a differential inclusion

$$\frac{dx}{dt} = g(x, y) \in \hat{g}(x, y) \oplus \epsilon_g \tag{5.16}$$

and

$$y = h(x) \in \hat{h}(x) \oplus \epsilon_h \tag{5.17}$$

For each nonlinear term (auxiliary term), the error is computed by solving a nonlinear optimization problems with linear constraints. For univariate functions, there is also the option to use directly the Taylor inequality.

6. Finally, we automatically generate a SpaceEx model file. The idea is that for each nonlinear term (recall they involve only two variables), we introduce a new variable and generate a domain (box or polytope) with two inputs and one output. This corresponds to a (linear) algebraic relation; no dynamics. It is written in the invariant.

   The error is added as an uncontrolled variable with the previously estimated bounds. There are two options for SpaceEx to parse it (either to add as a variable in the invariant or as a constant that belongs to a range). Each nonlinearity is saved as a new XML file and then they are combined with the original file (replacing nonlinear base component with PWA one).

Example. This small example considers a Van der Pol oscillator, which is a two-dimensional system with the following nonlinear dynamics:

$$\dot{x_1} = x_2$$
$$\dot{x_2} = (1 - x_1^2) * x_2 - x_1$$

By expansion, we get

$$\dot{x_1} = x_2$$
$$\dot{x_2} = -x_1 + x_2 - x_1^2 * x_2$$

That means that the state is $x = [x_1, x_2]$ and there is one nonlinear term which should be $u = -x_1^2 * x_2$ and can be identified from the dictionary. The range of of variables x_1, x_2 is $[-4\ , 4]$ and the quantization parameter for both variables is $h_{x_1} = h_{x_2} = 0.5$.

The gridding and quantization can selection can be seen in Figure 5.2. In Figure 5.3, the original function is plotted against the PWA one. A simulation run is portrayed in Figure 5.4.

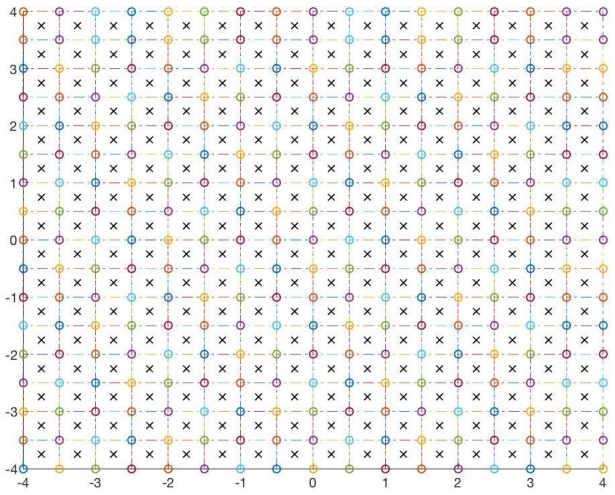


Figure 5.2: Fixed gridding- Van der Pol Oscillator

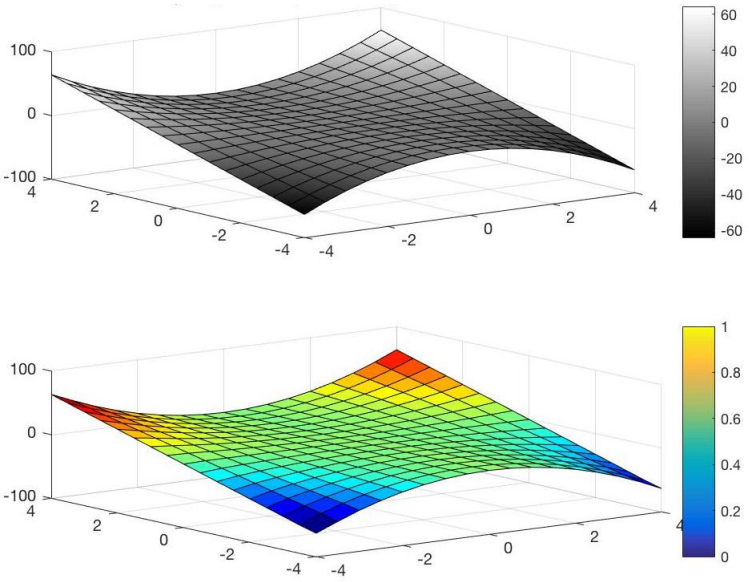


Figure 5.3: 3D- nonlinear vs PWA

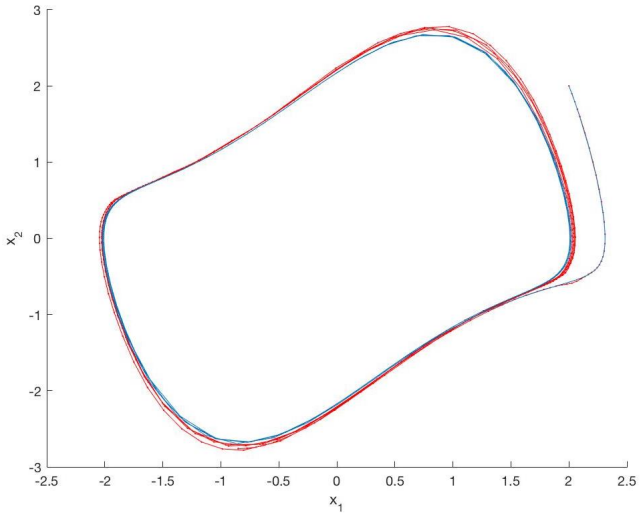


Figure 5.4: Van der Pol oscillator - simulation; original in blue, syntactic in red

The corresponding SpaceEx file consists of three four components; three base and one system components. The first base component is the clock (Figure 5.5), the second is the original dynamics with the auxiliary variable u (Figure 5.6) and the third is the PWA approximation of the nonlinear term including the error which is described by the variable w_1 (Figure 5.7).

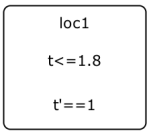


Figure 5.5: SpaceEx model - clock

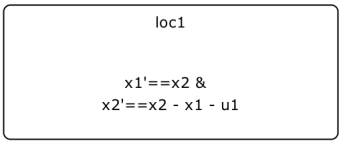


Figure 5.6: SpaceEx model - ODE

For bounded-time reachability (2 seconds), we can find a fixed point after 754 iterations in 2 minutes. The reachable states with SpaceEx (octagonal directions, STC, 0.01 tolerance) are shown in Figure 5.8.

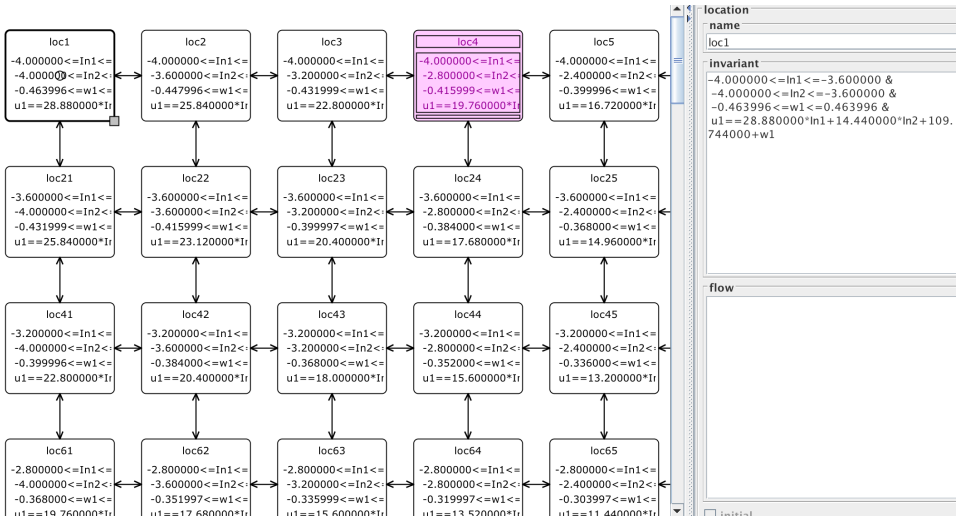


Figure 5.7: SpaceEx model- PWA approximation

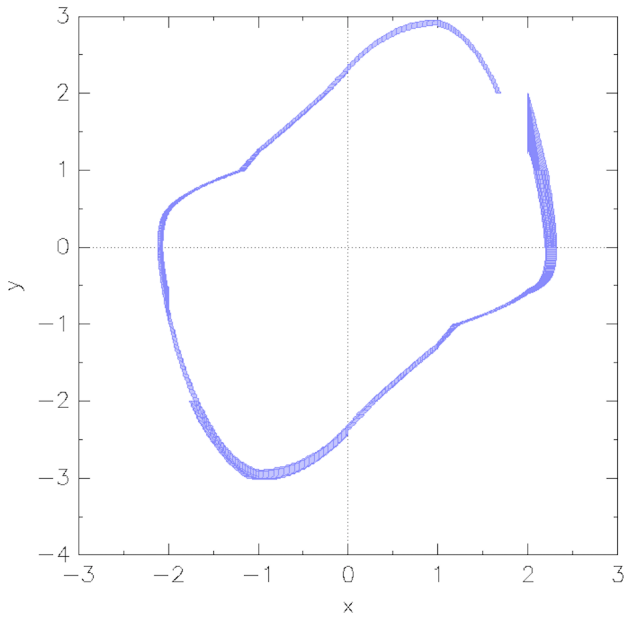
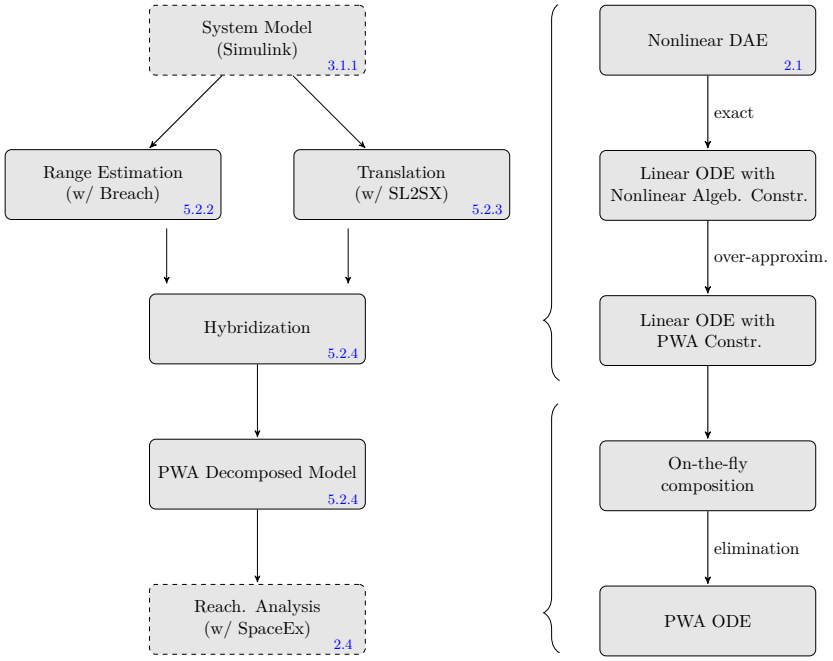


Figure 5.8: SpaceEx model- Reachable Sets

5.2 From Simulink to Hybrid Automata

In this Section, we present a transformation approach to obtain hybrid automata models from Simulink systems (portrayed in Fig. 5.9). To illustrate our methodology, the proposed steps are applied to a rotational pendulum model.



a) *Verification Workflow* **b)** *Compositional Syntactic Hybridization Scheme*

Figure 5.9: Methodology for constructing verification models is presented on the left and the steps of our hybridization scheme on the right.

The modeling and control design tasks are typically undertaken with MATLAB/ Simulink, through the connection of signals, blocks and subsystems. Simulink [227] is a graphical programming environment for modeling, simulating, and analyzing dynamical systems. More information about MATLAB and Simulink can be found in Section 3.1.1.

Example 5.2.1. As a running example, we consider a simple rotational pendulum [284]. The pendulum has a nonlinear term (a sine function) and its Simulink model is shown in Fig 5.10. The system produces simulation traces of the pendulum angle over time, when it is released from rest.

5.2.1 Model Adaptation

Model adaptation relates to the fact that a simulation model typically contains information that should be obscured from the verification model. A verification model could also be enriched with nondeterminism so as to check the behavior of the system under uncertain or varying parameters, disturbances or user inputs.

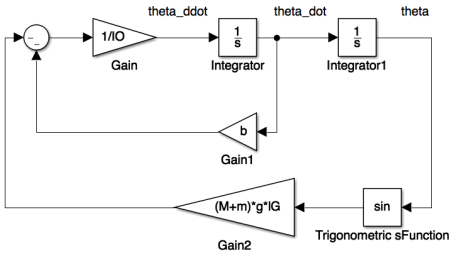
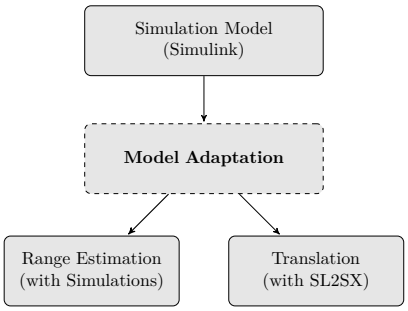


Figure 5.10: Simulink model for the rotational pendulum



5.2.2 Estimation of the signal range

The goal of this step is to obtain bounds on the behavior (minimum and maximum) of specific variables of the Simulink model. These variables typically correspond to Simulink signals. In particular, we care about the variables that appear in the blocks that cannot be directly described by PWA hybrid automata. Such blocks could include nonlinear function or more complicated operations. The smaller the operating ranges of these signals are, the smaller the number of locations required by the PWA abstraction (given a desired error bound) is going to be. There are various ways to estimate these ranges, such as simulations, statistical analysis [79], interval analysis [80], or Monte Carlo methods [298]. In this Chapter, we use corner-case simulations and the Breach [107] toolbox for a (not necessarily conservative) estimation of the signal range.

Note that these signal ranges are considered only as a rough indication for the hybridization procedure (presented in the next Section). The approximation is equipped with out-of-range scopes. Therefore, if the range is shown to be insufficient during reachability analysis, it is accordingly revised (enlarged).

Example 5.2.2. For the rotational pendulum example, we estimate the range of the signal that is as an input in the nonlinear block. The resulting range is enlarged by a percentage. Figure 5.11 shows a set of simulation runs

computed with Breach and displays the evolution of the angle θ as a variable of time. Uncertain initial conditions and a quasi random distribution (Sobol) were selected.

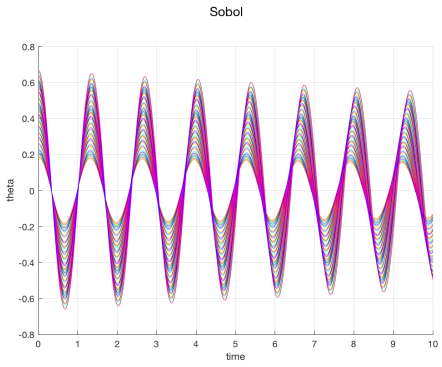


Figure 5.11: Estimating the range of the input signal of the nonlinear block (sin) of the rotational pendulum θ (rad) vs. time (s); conducted with Breach simulations.

5.2.3 Translation to SX format

The next step of the model transformation process is to translate the Simulink model into an equivalent hybrid automaton model. By equivalence, we mean that the new model is the same as the original one with only the syntax changing. The target hybrid automaton model respects the semantics of SX grammar; this format is similar to the standard hybrid automata, syntactically extended with hierarchy and templates. The SX formalism is supported by SpaceEx [135] and other verification tools. In the following, we are going to use SpaceEx and SX interchangeably to simplify the terminology. We use the SL2SX [231] tool to assist with the translation of the Simulink model into a hybrid automaton model expressed in the SX format. The tool outputs empty components for the unsupported Simulink blocks, e.g. nonlinear, sinks, or string manipulation blocks. For more information about SpaceEx, see Section 2.4, and about SL2SX, see Section 3.2.

Example 5.2.3. After applying the SL2SX tool to the rotational pendulum model, we get a SpaceEx model with base and network components. The top-level network component is shown in 5.12. The constructed model can be readily compared to the original one, thanks to the preservation of the block structures, names and variables. In red, we highlight the nonlinear block that cannot be translated by the tool. The way we handle this block is described in the following subsection.

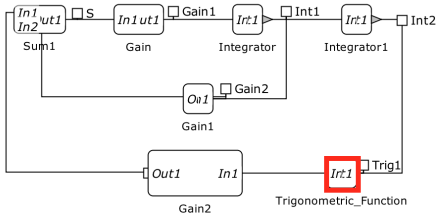


Figure 5.12: SpaceEx model of the rotational pendulum constructed by SL2SX tool. The top-level network component is shown; in red, we indicate the block that represents to a trigonometric function.

5.2.4 Hybridization

The goal of this hybridization step is to construct PWA approximations for the Simulink blocks that cannot be handled by the translator. These blocks may be nonlinear or may not have an exact translation. Also, there blocks for which the translation scheme cannot be applied (e.g. Embedded MATLAB Functions).

Once we perform the hybridization procedure, we express the constructed PWA approximations in SX format and integrate them within the entire model file. In other words, we obtain a SpaceEx model that combines the blocks that can be translated exactly and the blocks that are obtained from the hybridization process. Then, the resulting model can employed for formal analysis.

Example 5.2.4. Getting back to the the rotational pendulum example, the nonlinear function (sin) is over-approximated by a PWA function. We pick boxes/hyper-rectangles as linearization domains and we obtain 40 domains provided a given error bound. Each linearization domain corresponds to one location of the resulting automaton and is coupled with a distinct over-approximation error Note that the approximation errors are different for each domain and are added in the invariants as bounded, nondeterministic inputs. The resulting SpaceEx base component is highlighted in Figure 5.13.

5.2.5 Example

We use SpaceEx to perform the reachability computations. SpaceEx, composing on-the-fly the individual components, instantiates only the relevant parts of the model.

Example 5.2.5. For the reachability analysis of the rotational pendulum model, we select the STC algorithm, uncertain initial conditions (bounded

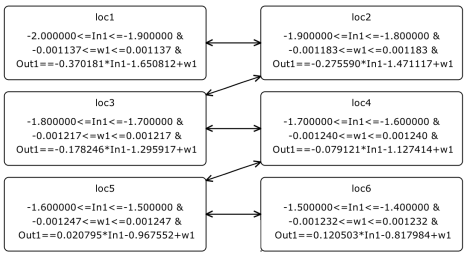


Figure 5.13: PWA of the nonlinear component (sin) of the rotational pendulum. Only 6 locations are shown (out of 40). Here, the nondeterministic input w_1 represents the approximation error.

initial speed), an approximation error of 0.01 and a time horizon of 1s). Figure 5.14 shows the resulting phase portrait when the pendulum is released from its most upward position.

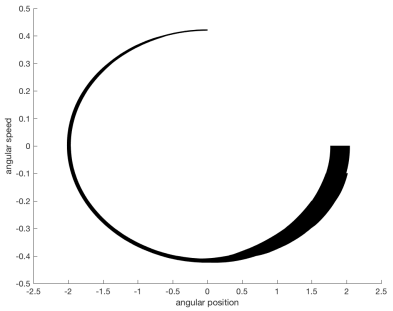


Figure 5.14: Reachability results of the rotational pendulum for a global time horizon of 1s. Phase portrait of the angular position (rad) and angular speed (rad/s).

Example 5.2.6. Getting back to Example 5.1.2, we now model this 9-dimensional biological system in Simulink. Figure 5.15 depicts the resulting model. The blocks that involve nonlinear operators are highlighted in color. The ones that correspond to the same nonlinearity have the same color. In total, there are three nonlinearities and three types of colored blocks.

We follow the aforementioned procedure: (i) translation into hybrid automata, (ii) range estimation, and (iii) compositional syntactic hybridization. As we pointed out in Section 5.1.1, we do not have to partition a nine dimensional space but we can construct two dimensional approximations (three in total). Each approximation corresponds to a single hybrid automaton, i.e. a SpaceEx base component. Note that a base component is essentially a template, i.e.

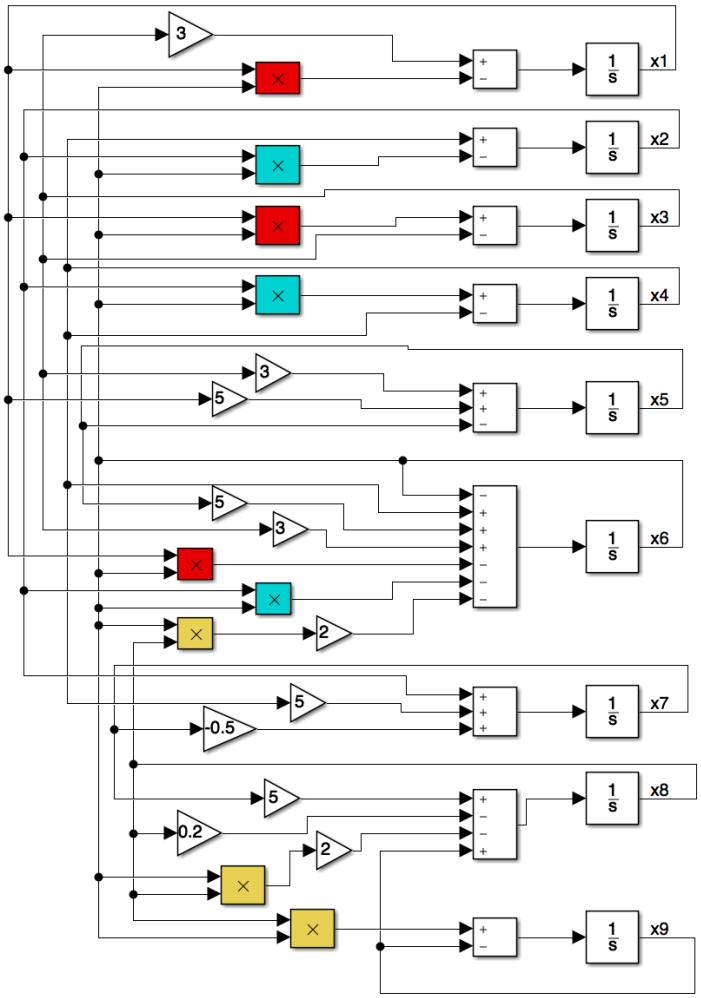


Figure 5.15: Genetic model in Simulink. Colored blocks correspond to nonlinear operators; cyan: $x_2 \cdot x_6$, red: $x_1 \cdot x_6$, and yellow: $x_6 \cdot x_8$.

building block, and it can be reused several times. Therefore, it would suffice to have one base component to describe the three *red* Simulink blocks, one for the *blue* blocks, and another one for the *yellow* blocks.

Finally, we compute the reachable sets of this biological model win SpaceEx. We select the STC scenario, an error of 0.01, octagonal constraints and a time horizon of 1sec. The reachabilty results are shown in Figure 5.16.

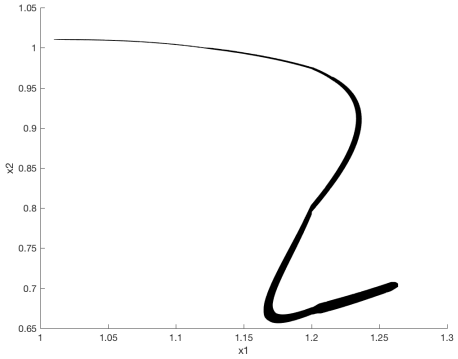


Figure 5.16: Reachable set of the genetic model computed with SpaceEx. The projection of a 9-dimensional set in 2D is displayed; 2D space corresponding to variables x_1 and x_2.

5.3 From Stateflow Diagrams to Hybrid Automata

Simulink/Stateflow (SLSF) has become a widely used industrial tool in the context of model-based design. SLSF relies on numerical simulation and has been applied to large-scale systems with complex dynamics. However, simulation is an inherently incomplete technique since the number of possible configurations is prohibitively large or even infinite. This limitation is mitigated by the use of formal verification techniques. Given that verification tools operate on formal models, it is necessary to transform SLSF models into a formal representation.

In this work, we translate a subset of SLSF models, i.e. continuous-time Stateflow charts, into hybrid automata. The resultant formal model complies with the SX format, which is a formalism used by several reachability tools, such as SpaceEx. We illustrate the verification of the transformed models on several examples.

In this respect, there is the need to translate Stateflow diagrams into hybrid automata. A major issue arises from the fact that Stateflow is a highly complex language with no formal semantics and its blocks and functionalities are described informally by the tool provider, Mathworks. A Stateflow diagram has an hierarchical structure and enables parallel and sequential state execution.

5.3.1 Stateflow Semantics

Simulink is a graphical modeling environment for plants, control design, and software. Stateflow is a graphical modeling language integrated within

Simulink. An SLSF design is represented as a diagram of interconnected Simulink blocks. It illustrates the mathematical relationships between the inputs, outputs, and states of the system. Note that the modeling language of Simulink/Stateflow lacks a formal definition of its semantics. In this section, we present an estimate of the semantics provided in [24].

Stateflow diagrams may take the form of a Chart, State Transition Table, Truth Table or Message Viewer. In this paper, we consider Stateflow charts as they better describe the hybrid behaviors and we define a restricted subclass of continuous-time Stateflow diagrams in the spirit of [38]. Only classic state machines are considered. The other two options (Mealy and Moore state machines) are specific cases, which we do not address.

Definition 5.2 (Stateflow Chart)**.** A Stateflow chart

$$\mathsf{SF} = (\mathsf{St}, \mathsf{Jun}, \mathsf{Var}, \mathsf{Trans}, \mathsf{Action}, \mathsf{Box}, \mathsf{Def})$$

consists of

- a finite set of states St which correspond to locations. States are partitioned into atomic (AND) and exclusive (OR) states. Drawing one state within the boundaries of another state indicates that the inner state is a substate (or child) of the outer state (or superstate).
- a finite set of junctions Jun which represent instantaneous states,
- a finite set of variables Var that is assumed real-valued. The variables are partitioned into input variables Var_I , output variables Var_O , and local variables Var_L.
- a finite set of actions $Action$ which can be performed during different event times and are subdivided into `entry`, `during`, and `exit` actions. Entry actions are executed only once when entering the state, and exit actions are executed only once when exiting the state. The during actions describe the continuous-time evolution of the variables according to a differential equation (this happens strictly between entering and exiting),
- a finite set of discrete transition relations $Trans$ defined by a tuple $(\ell, Guard, Update, TP, \ell')$, where
  - the source state or junction is $\ell \in St \wedge Junc$,
  - the guard is defined by $Guard$ and once satisfied it forces the discrete transition to be taken,
  - the $Update$ defines which variables are modified and to what value,

  - the priority, given by TP, is a natural number that indicates the order in which the transitions are taken if more than one is enabled,
  - the target state or junction is $\ell' \in St \wedge Junc$,

- a finite set of boxes Box which is typically used for grouping states together for visual reasons. It also enables the use of external or internal functions simplifying the model.
- a finite set of default transitions Def tailored for junctions or states. A default transition in exclusive decomposition specifies which exclusive (OR) state to enter when there is ambiguity among two or more neighboring exclusive (OR) states. A default transition to the connective junction defines that upon entering the chart, the destination depends on the condition of each transition segment. Note that a default transition has a destination but no source object.

A transition between states may occur at each simulation time step, whereas multiple junction transitions may occur in a single simulation time step. A continuous-time Stateflow diagram is roughly analogous to a hybrid automaton, but their behaviors differ in several ways [38]. In particular, Stateflow diagrams (i) are deterministic, (ii) have urgent transitions with priorities, and (iii) have events such as enabled transitions that are determined at runtime by zero-crossing detection algorithms.

5.3.2 Translation Scheme

We consider a Simulink model that contains only a single Stateflow chart $SF = (\mathsf{St}, \mathsf{Jun}, \mathsf{Var}, \mathsf{Action}, \mathsf{Box}, \mathsf{Def}, \mathsf{Trans})$. It can be translated to a hybrid automaton $H = (\mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump})$ subject to certain conditions as explained below. Note that we consider the scenario of an (idealized) numerical simulation of an SLSF diagram. In SLSF, the various actions (e.g. `entry`, `during`, and `exit` action, and transitions updates) are evaluated sequentially, while hybrid automaton action are executed concurrently. In practice, SLSF requires the execution of (however small) simulation step in each state.

States. St is a finite set of states containing substates and superstates. The low level substates (at least one) are transformed in locations in base components (single hybrid automata). The higher level substates are mapped to a network components which include all the base components and conducts

parallel composition. Each level higher maps the superstates to a network component.

Junctions. The junctions are transformed into instantaneous states. This means that a junction is mapped to a location of a base component whose invariant is valid for time $t = 0$. If a junction is included in an SLSF without any hierarchy, then a new location should be added in the base component that contains the incoming transition to the junction. If a junction maps superstates, then the base component should be included in a network component.

Real-valued variables. The variables of the SLSF are mapped to the variables of the HA. However, an extra variable is needed for HA that defines the passing of time. Note that the variables that end with *_dot* correspond to derivatives and are not needed in HA. The local variables of the SLSF remain local in HA. The variables of the SLSF that appear in `during` actions and end with *_out* are not local.

Actions. Each state of the SLSF is replaced by three locations in the HA. The `entry` and `exit` actions are instanaeous and occur only once. The `during` action is used to specify an ordinary differential equation and is mapped to flow of the corresponding location/state. The first location has an empty invariant (always true) and the reset equals the predicate of the entry action. The second location concerns the `during` action and the predicate is added in the flow. The third location has an empty invariant and the incoming transition has as a reset that equals the predicate of the `exit` action.

Boxes. The boxes are used for visual reasons and are not translated in HA. In case they contain functions for mathematical operations, these functions should be embedded inside the HA (in the corresponding element, e.g. flow, invariant).

Initial states. The initial states `Def` of SLSF are encoded as initial states of HA (Init).

Discrete transitions. The source and target states of a `Trans` are translated into source and target locations in HA. The guards of SLSF are transformed into urgent transitions. The updates correspond to resets of the HA. The mismatch between the sequential implementation of SLSF and the concurrent implementation of HA can either be ignored or bridged with the

introduction of a new location (for every every reset) in the HA that enables time elapse for a small time duration (equal to the delay, eps, in SLSF). This would lead to new locations (in total, #Updates-1) with invariants $t \leq \text{eps}$, flows $t' = 1$, where t is a controlled variable. Note that we not address inner transitions in this work.

5.3.3 Examples

In this part, we show how the translation scheme works on three benchmarks.

Single Hybrid Automaton. The first example[1] is an ARCH benchmark [13] that describes a motor-transmission system [81]. The Stateflow model is shown in Figure 5.17 and the corresponding hybrid automaton model in Figure 5.18. Note that the transitions of the SpaceEx model are urgent and they are defined according to the urgent semantics presented in [230].

Figure 5.17: Stateflow model of the Motor-Transmission System.

To evaluate the performance of our translation, we simulate the Stateflow model in Simulink and conduct reachability analysis with uncertain initial conditions in SpaceEx. Then, we check if the simulated behavior is enclosed in the reachable set. That is the case as illustrated in Figure 5.19. For the reachable sets, the initial conditions were enlarged by 1%, the flowpipe tolerance was 0.001, the STC scenario was selected and octagonal constraints were used. For a global time horizon of 0.5s, a fixed point was found after 7 iterations and the computation lasted 0.191s.

[1]Note that the latest SLX model has been modified and enhanced with respect to the original published in 2014.

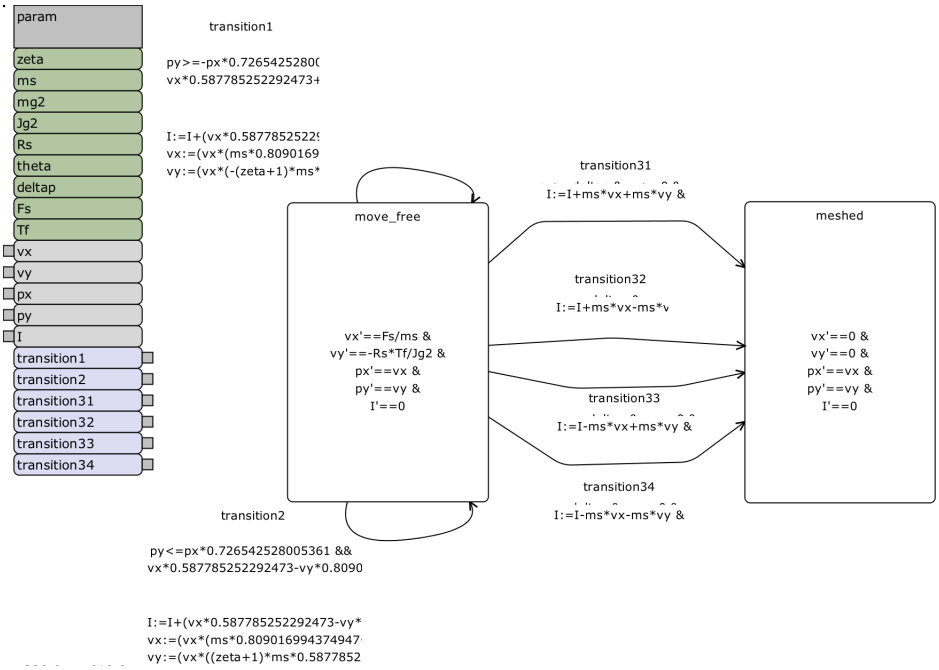


Figure 5.18: SpaceEx model of the Motor-Transmission System.

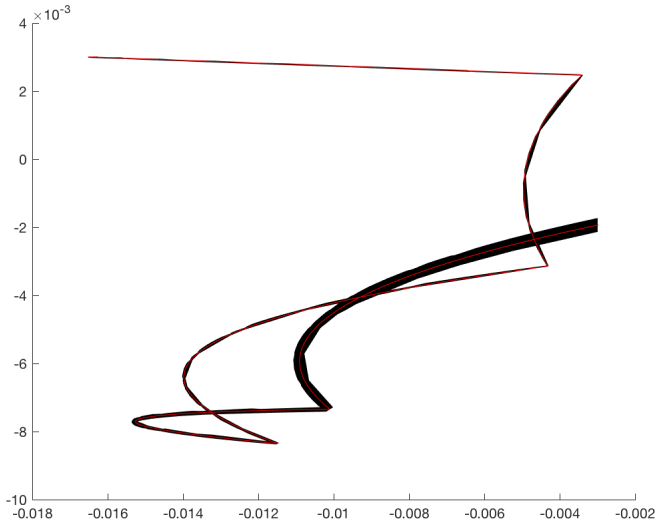


Figure 5.19: Analysis of Motor-Transmission System. Simulation results (Stateflow) in red vs Reachable sets (SpaceEx) in black.

Parallel Composition. The second example describes an SLSF model of two thermostats and forms another reachability benchmark [223]. The Stateflow model is shown in Figure 5.20 and the corresponding SpaceEx

model in Figures 5.21 and 5.22. Note that for this case, we construct two base components (one for each thermostat) and we compose them in parallel inside a network component.

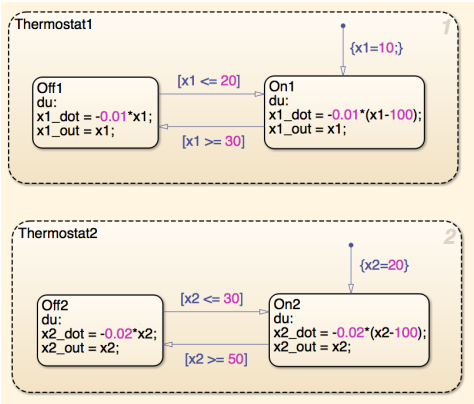


Figure 5.20: Stateflow model of the Thermostats System.

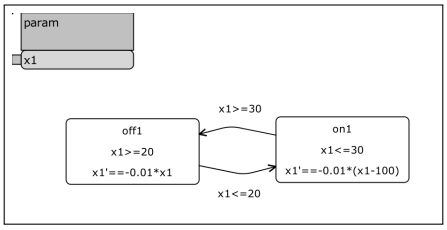


Figure 5.21: SpaceEx model of the Thermostats System - Base Component.

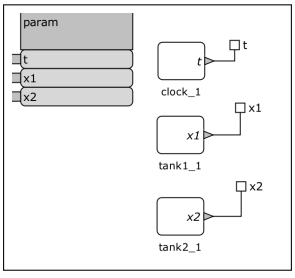


Figure 5.22: SpaceEx model of the Thermostats System - Network Component.

To evaluate the performance of our translation, we follow the same procedure as before. Indeed, the simulated behavior is enclosed in the reachable set. That is the case as illustrated in Figure 5.23. For the reachable sets, the initial conditions were enlarged by 5% and 10% respectively, the flowpipe tolerance was 0.1, the STC scenario was selected and box constraints were

used. For a global time horizon of 180s, a fixed point was found after 32 iterations and the computation lasted 0.221s.

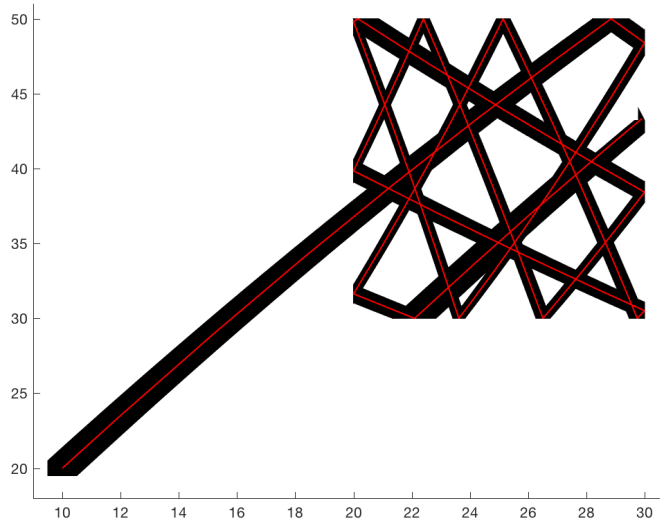


Figure 5.23: Analysis of Thermostats System. Simulation results (Stateflow) in red vs Reachable sets (SpaceEx) in black.

Hierarchy. The third example is a Simulink demo model from Mathworks [226]. It models a permanent-magnet DC motor and employs hierarchy. The SLSF model is shown in Figures 5.24 and 5.25.

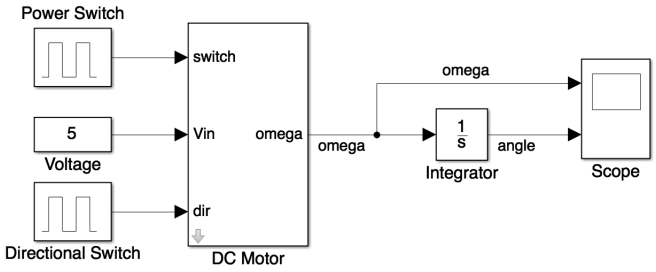


Figure 5.24: SLSF model of the DC motor system.

The SpaceEx model is shown in Figures 5.26 and 5.27. Note that the existence of hierarchy necessitates the use of synchronization labels. The priority is sustained in the HA model by the transition from location `powerOff` to location `up`. Self-loops are critical to avoid deadlocks due to the synchronization. Also, transitions from locations `up` and `down` back to the `powerOff` are required since this location corresponds to a superstate.

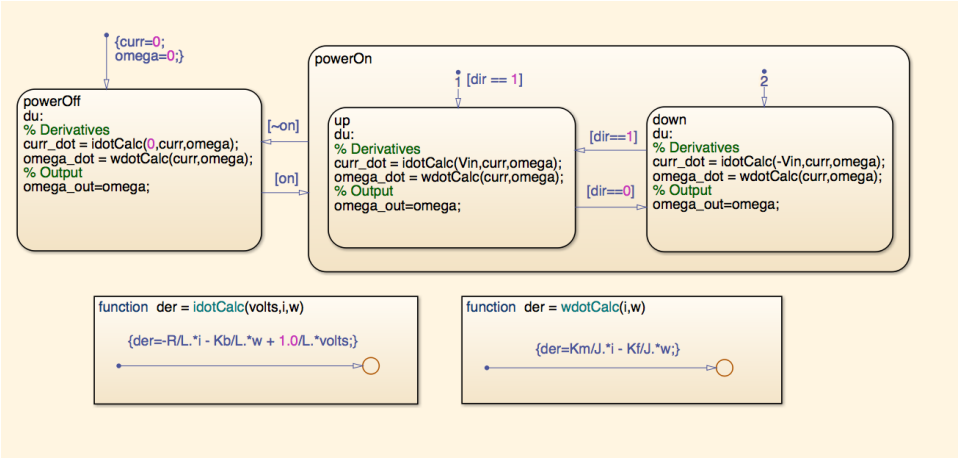


Figure 5.25: SLSF model of the DC motor system.

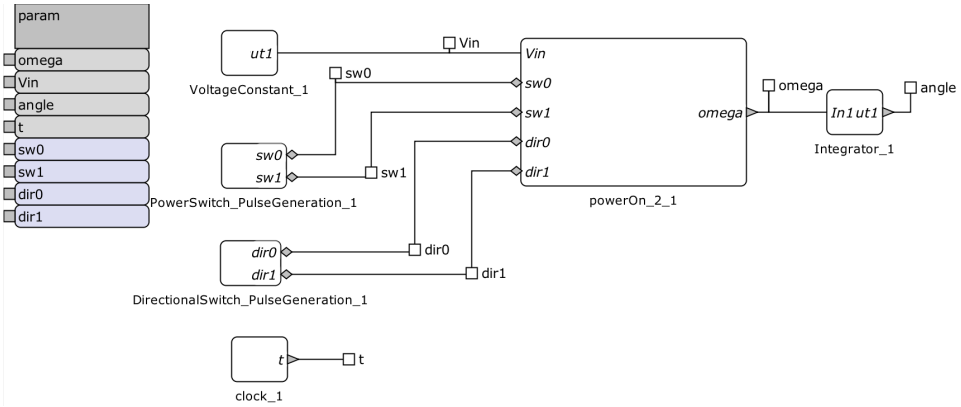


Figure 5.26: SpaceEx model of the DC motor system.

The simulated behavior is enclosed in the reachable set. That is the case as illustrated in Figure 5.28. For the reachable sets, the initial conditions were enlarged, the flowpipe tolerance was 0.01, the STC scenario was selected and box constraints were used. For a global time horizon of 10s, a fixed point was found after 30 iterations and the computation lasted 0.225s.

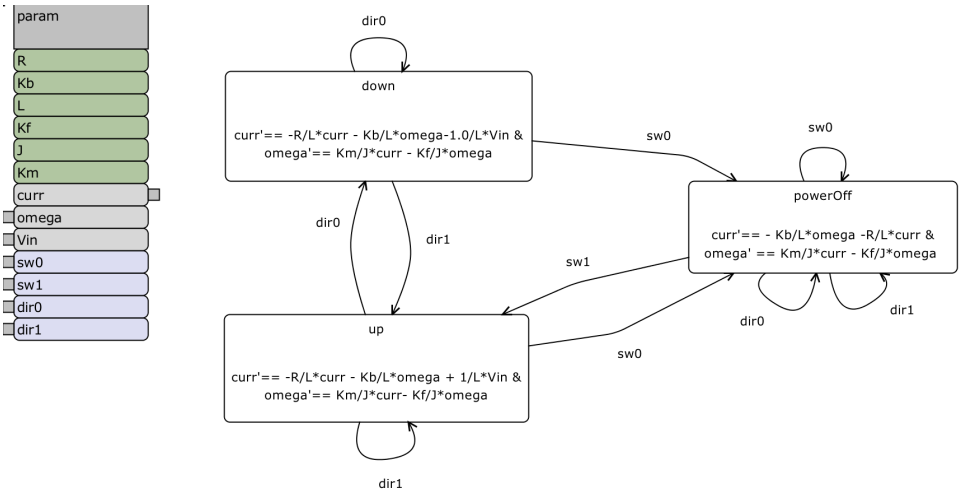


Figure 5.27: SpaceEx model of the DC motor system.

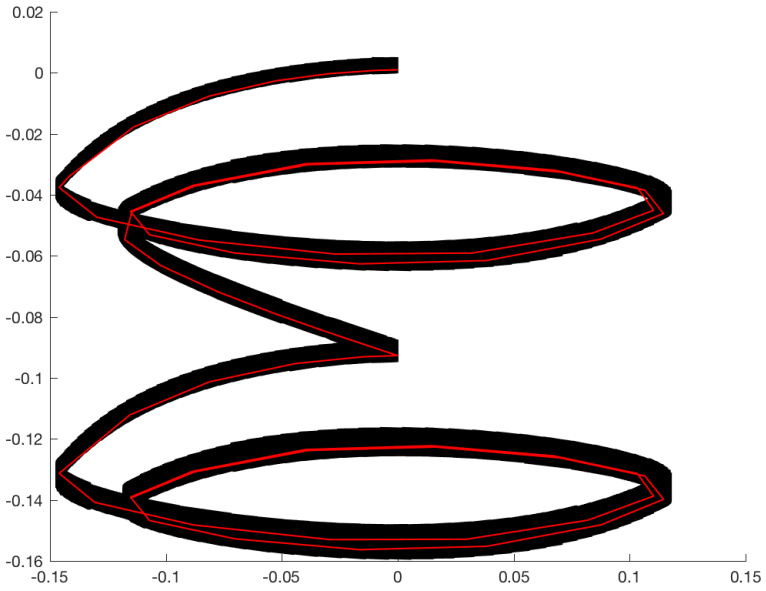


Figure 5.28: Analysis of the DC Motor System. Simulation results (Stateflow) in red vs Reachable sets (SpaceEx) in black.

5.4 Urgent Semantics

Standard reachability algorithms, based on may semantics, cannot handle *urgent semantics*. The main reason is that the computation of the states reachable via time elapse is more complex. A formal description of urgency can be found in [144, 264]. Following the work of [229, 230], the time elapse computation with urgency can be reduced to time elapse with a nonconvex invariant by taking the complement of the negation of the urgent guard. Then, the nonconvex invariant should be partitioned in convex and closed subsets. Finally, the standard time elapse is applied recursively on each of the elements. These approaches, however, may lead to coarse over-approximations and error accumulation. In this respect, we propose a heuristic method that can reduce the over-approximation by splitting the invariant along the so-called Lie Planes.

5.4.1 Reach Tubes under Invariant Constraints

Let the *reach set* $\mathcal{S}(t, X_0, \mathcal{I})$ of $\dot{x} = Ax + b$ be the set of states $x(t)$ such that $x(0) \in X_0$ and for all $0 \leq \tau \leq t$, $x(\tau) \in \mathcal{I}$, where $\mathcal{I}$ denotes the invariant. The *reach tube* $\mathcal{R}(t, X_0, \mathcal{I})$ is the union of the reach sets up to t,

$$\mathcal{R}(t, X_0, \mathcal{I}) = \bigcup_{0 \leq \tau \leq t} \mathcal{S}(\tau, X_0, \mathcal{I}).$$

Both the reach set and the reach tube are difficult to compute in general. Tools like SpaceEx instead compute an overapproximation by first ignoring the invariant and then intersecting with it a-posteriori:

$$\hat{\mathcal{R}}(t, X_0, \mathcal{I}) \stackrel{!}{=} \mathcal{R}(t, X_0, \mathbb{R}^n) \cap \mathcal{I}.$$

In the following, we will examine under which conditions this heuristic is exact, and propose a solution that is more accurate.

Reach Set under Invariant Constraints

We consider nondeterministic affine dynamics of the form

$$\dot{x} = Ax + u, \qquad u \in \mathcal{U}.$$

In the following, we will without loss of generality assume that

$$0 \in \mathcal{U}.^{2}$$

If there are no invariant constraints, then the reach set is [295]

$$\mathcal{S}(t, X_0, \mathbb{R}^n) = e^{At} X_0 \oplus \mathcal{T}(t),$$

where the *input convolution* is independent of the initial states,

$$\mathcal{T}(t) = \int_{\tau=0}^{t} e^{A(t-\tau)} \mathcal{U} d\tau = \lim_{\delta \to 0} \bigoplus_{k=0}^{t/\delta} e^{A\delta k} \delta \mathcal{U},$$

A conservative over- and under-approximation of the input convolution can be obtained by taking a finite time step δ and adding an appropriate bloating term $\mathcal{E}(\delta)$ has been proposed in [202]:

$$\mathcal{T}^{-}(t) \stackrel{!}{=} \bigoplus_{k=0}^{t/\delta} e^{A\delta k} \delta \mathcal{U} \subseteq \mathcal{T}(t) \subseteq \bigoplus_{k=0}^{t/\delta} e^{A\delta k} (\delta \mathcal{U} \oplus \mathcal{E}(\delta)) \stackrel{!}{=} \mathcal{T}^{+}(t). \tag{5.18}$$

As the time step δ goes to zero, both sides converge to the true input convolution. In the sequel, we will need the following *semi-group property* of the reach set, which allows us to move sets forward in time:

$$\forall t, s \geq 0 : \mathcal{S}(t + s, X_0, \mathbb{R}^n) = e^{At} \mathcal{S}(s, X_0, \mathbb{R}^n) \oplus \mathcal{T}(t),$$

Because $0 \in \mathcal{U}$, the input convolution is monotonic:

$$\forall t' \geq t : \mathcal{T}(t) \subseteq \mathcal{T}(t'). \tag{5.19}$$

Invariant. In the presence of invariant constraints, the computation of the reach set is more complicated, as is described in detail in [201]. Let $\mathcal{V}(\delta) = \delta \mathcal{U} \oplus \mathcal{E}(\delta)$. Let $\mathcal{Y}_k$ be the sequence given by $\mathcal{Y}_0 = X_0$,

$$\mathcal{Y}_{k+1} = (e^{A\delta} \mathcal{Y}_k \oplus \mathcal{V}(\delta)) \cap \mathcal{I}.$$

Then the reach set is the limit $\delta \to 0$:

$$\mathcal{S}(t, X_0, \mathcal{I}) = \lim_{\delta \to 0} \mathcal{Y}_{t/\delta}.$$

[2] If $0 \notin \mathcal{U}$, we can construct an equivalent system as follows: Let $u_0 \in \mathcal{U}$. Add an auxiliary variable z whose value is identical to 1 at all times, i.e., $z(0) = 1$ and $\dot{z} = 0$. Then substitute $x \leftarrow [x; z]$, $A \leftarrow [A, u_0; 0, 0]$, $\mathcal{U} \leftarrow \mathcal{U} \oplus \{-u_0\}$.

A Geometric Sufficient Condition

It is known [171] that this approximation is exact, i.e., $\hat{\mathcal{R}}(t, X_0, \mathcal{I}) = \mathcal{R}(t, X_0, \mathcal{I})$, if the invariant is convex and the system follows straight-line trajectories, i.e., $A^2 = 0$ and $A\mathcal{U} = 0$. An algorithm for computing $\mathcal{R}(t, X_0, \mathcal{I})$ has been proposed by Girard and Le Guernic, but it is computationally much more expensive.

We extend this result to a more general case. Let $\mathcal{I}$ be characterized by a differentiable function $h(x)$ such that $\mathcal{I} = \{x \mid h(x) \leq 0\}$. The Lie derivative of h with respect to the vector field $f(x) = Ax + b$ is

$$\partial_f h(x) = \lim_{t \to 0} \frac{h(x + tf(x)) - h(x)}{t}.$$

Lemma 5.3. *If (i) for all $x \in \hat{\mathcal{R}}$, $h(x) = 0 \Rightarrow \partial_f h(x) \leq 0$ or (ii) for all x, $h(x) = 0 \Rightarrow \partial_f h(x) \geq 0$, then $\hat{\mathcal{R}}(t, X_0, \mathcal{I}) = \mathcal{R}(t, X_0, \mathcal{I})$.*

If $\mathcal{I}$ is a polyhedron, i.e., a conjunction of linear constraints $c_i^T x \leq d_i$, we can specialize this condition as follows. The Lie dervative on the border of a constraint $c_i^T x \leq d_i$ is

$$\partial_f h(x) = c_i^T Ax + c_i^T b.$$

The condition $\partial_f h(x) \leq 0$ then defines a halfspace

$$c_i^T Ax \leq -c_i^T b.$$

This leads to the following condition:

Lemma 5.4. *If either (i) for all i,*

$$\hat{\mathcal{R}} \cap \{c_i^T x = d_i\} \subseteq \{c_i^T Ax \leq -c_i^T b\},$$

or (ii) for all i,

$$\hat{\mathcal{R}} \cap \{c_i^T x = d_i\} \subseteq \{c_i^T Ax \geq -c_i^T b\},$$

then $\hat{\mathcal{R}}(t, X_0, \mathcal{I}) = \mathcal{R}(t, X_0, \mathcal{I})$.

It follows from the above that partitioning $\mathcal{I}$ along the hyperplanes $c_i^T Ax = -c_i^T b$ leads to convex partitions inside each of which the approximate reach operator is exact. However, the number of partitions may be large, so that it may be worthwhile to investigate exactly which of those partitions are necessary to take into account.

5.4.2 Examples

In this part, we consider hybrid automata models with different urgent transitions. We indicate how the proposed heuristic of employing the so-called Lie planes help obtaining tighter over-approximations. As noted, on the one hand, simulation tools are deterministic and rely on urgent semantics. On the other hand, reachability tools like SpaceEx use may semantics. Let us consider the example of a hybrid automaton model which is shown in Figure 5.29. The model has two state variables and two locations with different flows (continuous dynamics). There is an urgent transition and the initial condition for y is $y(0) = 0$. If we input this model to SpaceEx, which does not support urgent transitions, it will output the reachable sets shown in Figure 5.30a. It can be seen that after the guard is reached, just consider the hyperplane $y = 2.9$, two flowpipes are generated. The top flowpipe corresponds to the loc2 where $y > 2.9$, while the one on the bottom corresponds to loc1 where $y < 2.9$. However, the system should not be able to produce the bottom flowpipe as it can no longer stay in loc1 and must move to loc2 (due to the urgent transition). The correct reachability results are portrayed in Figure 5.30b. This example aims to streamline that utilizing existing reachability tools with may semantics leads to coarse over-approximative results which can hamper formal verification attempts.

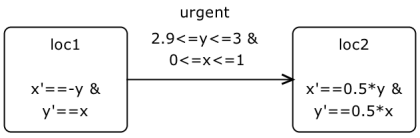
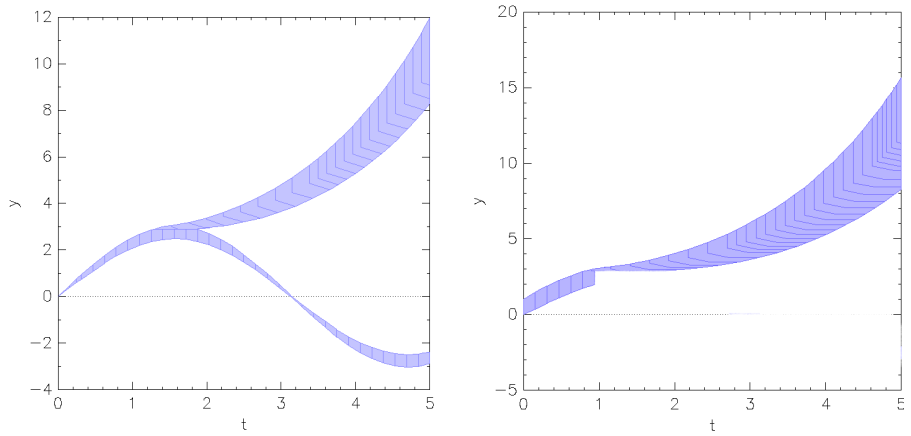


Figure 5.29: Hybrid automaton model with one urgent transition

In the following, we consider other hybrid automata models and we show that our proposed model transformation approach produces tighter reachable sets and avoids large over-approximation errors. We present models of increasing complexity.



a) *May semantics - overapproximation* **b)** *Urgent semantics - correct*

Figure 5.30: Hybrid automaton model with one urgent transition - Reachable sets computed with SpaceEx (may semantics); bottom flowpipe is spurious.

Example with one urgent guard (one constraint and one variable). In this example, we only have one guard condition and when the dynamics jump there is no continuous evolution. This is done by adding a *false* flow. As such, it would be easier to visualize the correct and anticipated behavior vis-a-vis the actual one.

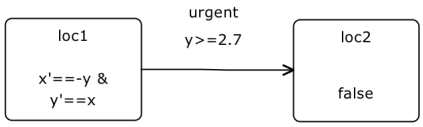


Figure 5.31: Hybrid automaton model with one urgent transition ($y \geq 2.7$).

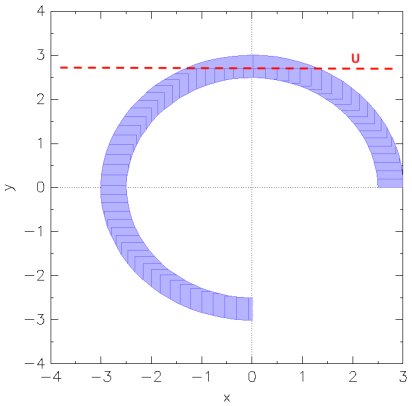


Figure 5.32: Hybrid automaton model with one urgent transition ($y \geq 2.7$) - Reachability analysis with may semantics.

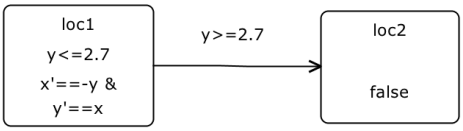


Figure 5.33: Hybrid automaton model with one urgent transition ($y \geq 2.7$).

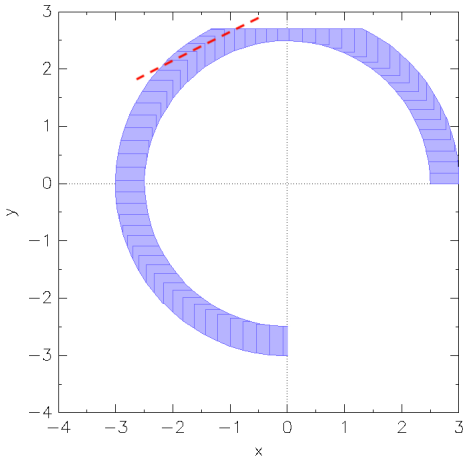


Figure 5.34: New hybrid automaton model with one may transition ($y \geq 2.7$) - Reachability analysis with may semantics.

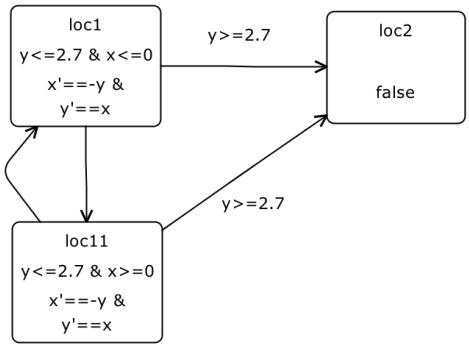


Figure 5.35: New hybrid automaton model - after splitting with the Lie plane $x = 0$.

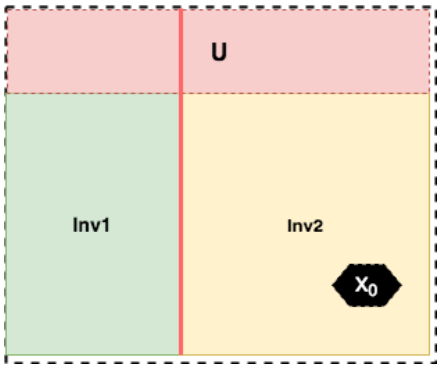


Figure 5.36: New hybrid automaton model - Partitioning with respect to the Lie plane.

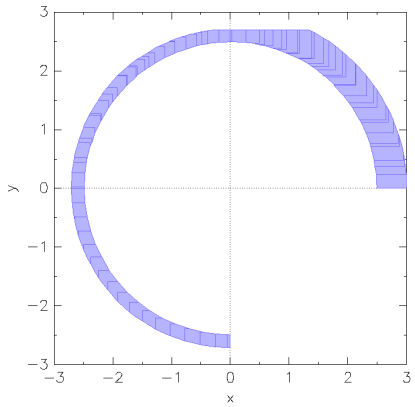


Figure 5.37: New hybrid automaton model with two new locations - Reachability analysis with may semantics.

Example with one urgent guard (four constraints and two variables). In this example, we only have one guard condition and when the dynamics jump there is no continuous evolution. This is done by adding a *false* flow. As such, it would be easier to visualize the correct and anticipated behavior vis-a-vis the actual one.

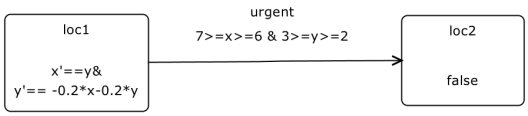


Figure 5.38: Hybrid automata model with one urgent transition (4 constraints).

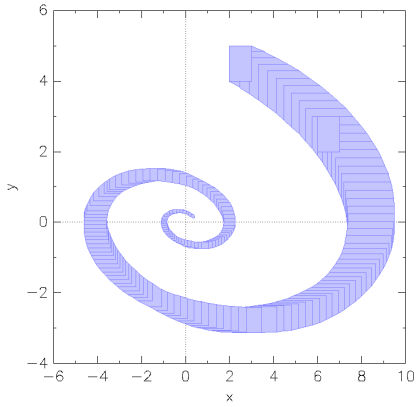


Figure 5.39: Hybrid automata model with one urgent transition (4 constraints) - Reachability analysis with may semantics.

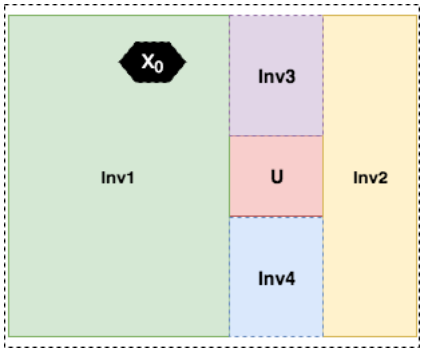


Figure 5.40: Hybrid automata model with one urgent transition (4 constraints).

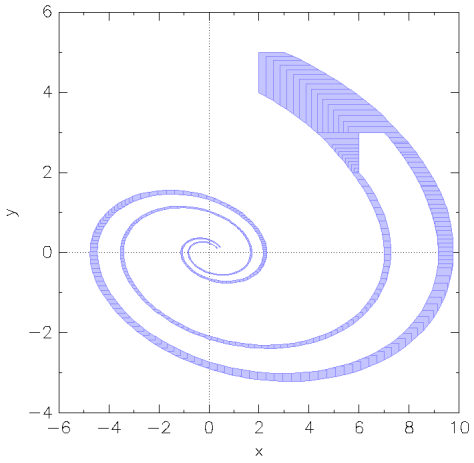


Figure 5.41: Hybrid automata model with one urgent transition (4 constraints) - Reachability analysis with must semantics.

5.5 Related Work

The formal verification of MATLAB/Simulink models has been studied in the literature [297]. There are two main research directions. The first direction is known as simulation-based verification or verification by simulation. Techniques and tools have been developed that operate on the original MATLAB/Simulink model and analyze it by manipulating the model inputs and running extensive simulations. The focus is either on providing coverage of the system behaviors or falsifying given requirements. Typically, the MATLAB engine is used for numerical compuations, a fact that renders the analysis prone to unsound results. Representative tools are Breach [107], S-Taliro [27], and C2E2 [121].

These techniques, however, need the set of initial states to be sampled. Since the number of required samples can grow exponentially with the number of state variables, this can restrict the applicability of such an approach to systems with low-dimensional initial states. Moreover, these techniques can be employed to verify bounded-time properties but cannot be applied for unbounded time. The tool HySon [63, 64] aims to provide complete analysis results by performing set-based simulation directly on a Simulink model and computes a good approximation of the set of all possible executions. However, this tool is not publicly available and the technical details imply that it might have drawbacks while analyzing hybrid systems for an unbounded switching horizon.

The second research direction is to translate the Simulink models into modeling formalisms which are amenable to existing hybrid system verification tools [42, 248, 264] One promising formalism concerns the use of hybrid automata. The translation of Simulink models is supported by the Hylink [223] and GreAT tools [3]. However, these tools do not allow hierarchical modeling and can only be applied to a small subset of Simulink blocks. An original algorithm to translate Simulink models into a hybrid system formalism was proposed in [247]. Alur et al. [24] translated a class of Simulink models into linear hybrid automata to improve simulation coverage, but they only consider deterministic models. Stanley Bak et al. proposed a translation process from Simulink/Stateflow to hybrid automata in [223]. Both papers focus on Stateflow diagrams and necessitate the transformation of the Simulink model into a Stateflow one. This transformation, however, is not straightforward and sometimes it can even be impossible, e.g. in case of large-scale models.

However, the translation of MATLAB/Simulink models into formal modeling languages runs into two obstacles. The first obstacle encountered is the

lack of a formal definition of its underlying semantics [297]. As such, most tools and techniques consider an idealized version, a relaxed version or an estimate of the semantics. In the literature, there has been work realted the interpretation and development of formal semantics for Simulink. Several types have been proposed, such as denotational [158], operational [61,159,160], continuation-passing style [66], and communicating push-down automata based [287].

The second obstacle concerns the presence of urgency in Simulink/Stateflow models, which is usually not supported by formal verification tools. Simulink models are deterministic and utilize urgent transitions, which means that a transition must be taken as soon as the guard is satisfied. Standard reachability algorithms and tools are based on may semantics and cannot handle urgent semantics. The main reason is that the computation of the states reachable by time elapse is different. One way to overcome this problem is to substitute the urgent guard condition by a condition on the invariant of the source location [171]. However, this transformation leads to non-convex invariants and requires splitting of these invariants into convex partitions. Each partition would correspond to a new auxiliary location of the resulting hybrid automaton. Issues with boundary conditions (strict vs. non-strict inequalities) arise which might lead to unsound or over-approximative results [60]. An algorithm for computing exactly the time-elapse of linear hybrid automata with urgency is presented in [229]. It has been extended to PWA hybrid automata in [230]. These approaches, however, still can lead to coarse over-approximation of the reachable sets and are not efficient computationally since they tend to generate many unnecessary auxiliary locations.

Finally, on a slightly different note, there has been work on the translation of a formal model into an Simulink/Stateflow model. Bak et al. [38] have proposed an approach to translate hybrid automata into continuous-time SL2SF. Pajic et al. [243] have worked on the translation of timed automata described in UPPAAL into SLSF diagrams. The translation of Hybrid Communicating Sequential Processes into Simulink diagrams is presented in [298].

CHAPTER 6

Case Studies

Parts of this Chapter have been published in [188, 189].

6.1 Cruise Controller

The area of autonomous vehicles has instigated research interest both from industry and academia [119]. Autonomous vehicles are expected to become the most common means of transportation by 2040 [215]. An autonomous vehicle can fully take over the driving duties leading to a significant prevention of accidents caused by human errors. Human-caused accidents account for 90% of the total accidents [216]. In this vein, automated driving is expected to provide safety and comfort as well as reduce accidents, crashes, and congestion [291]. Nonetheless, autonomous vehicles drive in a dynamic environment and their safe operation has to be guaranteed.

The main components of a modern autonomous vehicle are localization, perception, and control [194]. The literature on vehicle path planning and control is rich, see the volumes [106, 175, 191, 255] and references therein. Control design of autonomous vehicles has been treated, e.g. in [120, 210]. Different vehicle models for autonomous driving are presented in [16, 194]. Efficient trajectory planning with obstacle avoidance is a fundamental task for autonomous vehicles. The approaches can be divided into (i) planning in discrete space and (ii) planning in continuous space [215, 216]. Very recently, an open source tool SPOT [197] was proposed to compute the future occupancy of multiple traffic participants on arbitrary road networks. A set of benchmarks for motion planning on roads can be found at [16].

One of the most common driving assistance systems of modern vehicles, autonomous and non-autonomous alike, is cruise control. The purpose of a cruise control system is to regulate the speed of the vehicle, despite external disturbances. Its basic operation is to measure the actual vehicle speed, compare it to the reference or desired speed and automatically accelerate or decelerate according to a control law [174], [292]. Reachability analysis of a cooperative adaptive cruise controller has been considered in [190]. The authors conduct safety analysis of two adjacent vehicles in a platoon, defined by a linear dynamical model and controller.

In this Section, we apply the verification workflow shown in Figure 3.2 on a cruise control case study, adapted from [190]. In particular, we employ the toolchain to conduct formal verification on Simulink/Stateflow models and perform analysis with SCADE.

Simulink Model. The closed-loop system, shown in Figure 6.1, consists of the physical plant (ODEs) and a PID controller. The speed is measured by a sensor which is assumed to be noisy. The uncertainty is modeled as a bandwidth-limited white noise with a noise power of 0.01 and correlation (sampling) time of 0.1s. The reference signal is time-varying and is modeled with Stateflow, as shown in Figure 6.2.

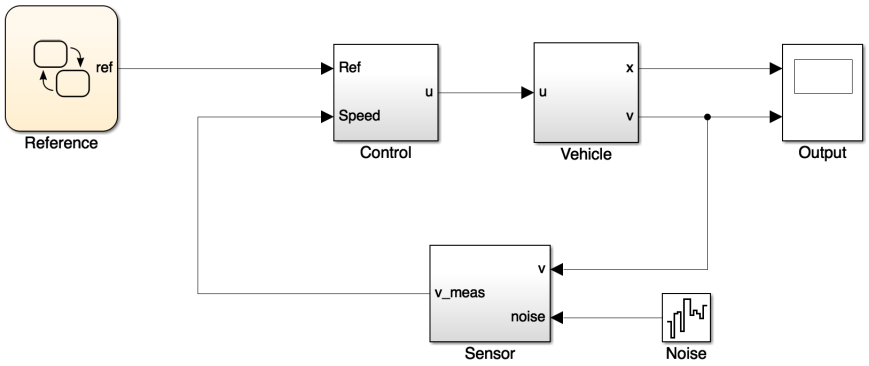


Figure 6.1: Simulink model of the Cruise Control system.

Formal Model. Using SL2SX tool [231] and the proposed SLSF translation, the model is translated into a formal model described by piecewise affine hybrid automata in SpaceEx format. It consists of 13 base components (single HA) and 3 network components (networks of HA). Note that general, nonlinear Simulink systems can be transformed to SpaceEx models in the form of PWA hybrid automata, as shown in [187, 188]. The SpaceEx model is shown in Figures 6.3 and 6.4. Note that the noise is over-approximated by a nondeterminsitic variable that lies in the range $[-1, 1]$.

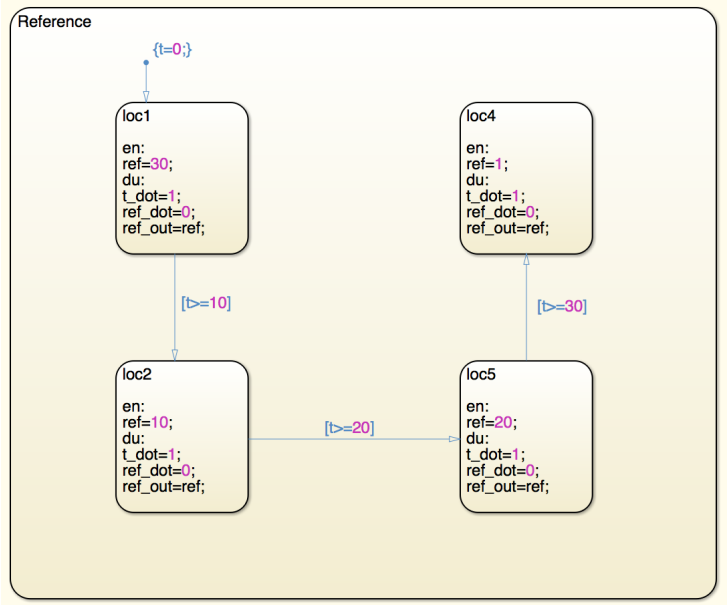


Figure 6.2: Stateflow chart of the Cruise Control system.

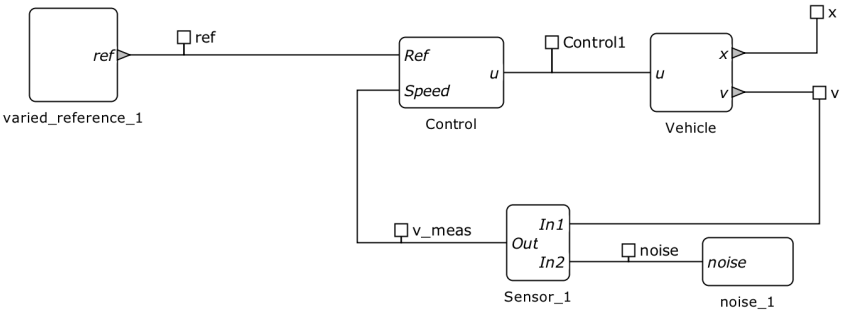


Figure 6.3: SpaceEx model of the Cruise Control system.

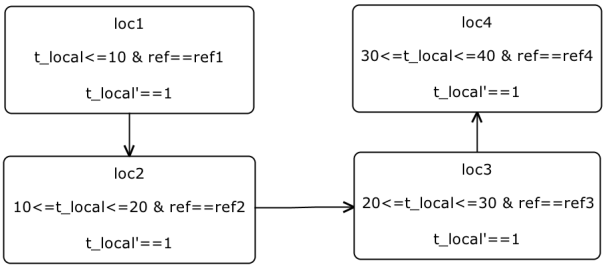


Figure 6.4: SpaceEx model of the Stateflow chart.

Requirements. The control objective is to regulate the vehicle speed while respecting the following design specifications, (i) rise-time < 6s, (ii) settling-time < 9s, and (iii) overshoot $< 10\%$.

Formal Specifications. The control objectives can be translated into formal specifications using pattern templates [133], as shown in Section 4.1.1. The rise-time is mapped to the *bounded response* pattern, the settling-time to the *timed absence* pattern, and the overshoot to the *absence* pattern. Using the `formalSpecs` tool [76], we construct monitor automata for these patterns. These monitor automata, once composed with the system under test, encode the requirements as reachability properties. Considering the first reference speed ($v = 30$) and the settling-time pattern, the monitor automaton is shown in Figure 6.5 and the complete system in Figure 6.6.

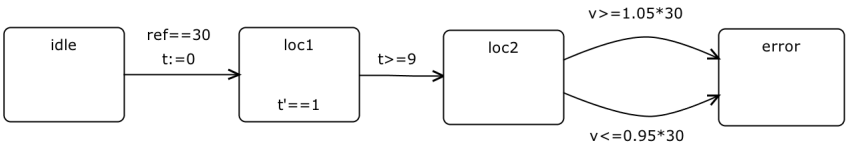


Figure 6.5: Monitor automaton describing the settling-time control objective of the Cruise Control system, shown in SpaceEx Model Editor.

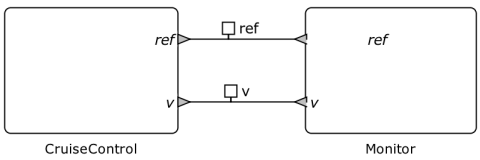


Figure 6.6: Composition of the formal model with the monitor automaton (timed absence), shown in SpaceEx Model Editor.

Formal verification via reachability analysis. SpaceEx [135] is used for computing the reachable sets. The safety verification problem is tackled by introducing a set of forbidden states, i.e. "loc(monitor)==error". Running SpaceEx (STC scenario, box directions) for 105 seconds, we get a fixed point after 4 iterations. The global time horizon is 40s and the initial condition is uncertain $0 \leq v \leq 3$. Since the error state is not reachable, the property is satisfied. Figure 6.7 displays the evolution of the vehicle speed over time.

Analysis with SCADE. The system model includes the plant, the environment and the controller. Through the formal verification, the system is guaranteed to satisfy the specification under the considered uncertainties. The next step is to analyze the system with the SCADE Suite. Initially, we apply the `sx2sh` translator (see Section 3.2) to obtain the equivalent hybrid program that can be employed by SCADE. Both the flattened version of the formal model (single hybrid automaton) and the original model (compositional, network of hybrid automata) can be parsed by the translator. The

resulting SCADE models are equivalent and are deterministic. Then, we use SCADE Hybrid to run simulations and check whether the controller needs to be refined or not. Figure 6.8 shows a simulation trace from specified initial conditions.

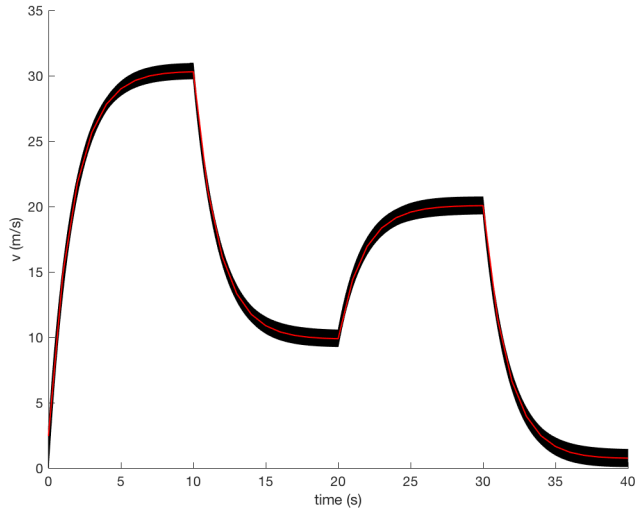


Figure 6.7: Reachable analysis of a Cruise Control system conducted with SpaceEx; black: reachable sets, red: simulation with Simulink (enclosed in the reach set).

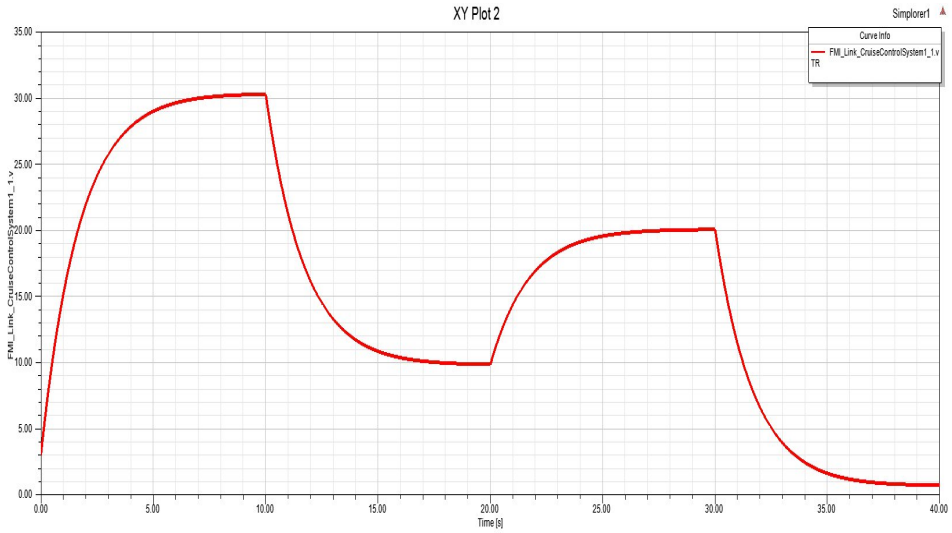


Figure 6.8: Simulation of a Cruise Control system conducted with SCADE Hybrid; input: formal model expressed in SpaceEx format (with or without the monitor automaton); model transformation with SpaceEx to SCADE translator.

6.2 Wind Turbine

The wind turbine systems form one of the fastest-growing industries in renewable energy worldwide. An industrial wind turbine model has been proposed by General Electrics in [267]. This wind turbine is designed with MATLAB/Simulink and it is a large-scale nonlinear model equipped with hybrid controllers.

In this Section, we describe the process of building a PWA hybrid automaton model of the Simulink model. The resulting model is expressed in SX format, used by SpaceEx [135] and other tools. The model transformation process involves four steps: (i) adapt the Simulink model to respect the verification standards, e.g. adding nondeterminism, (ii) build the formal model with the help of SL2SX tool [231], (iii) perform compositional syntactic hybridization (see Section 5.1) to obtain PWA approximations of the nonlinear function/blocks, and eventually (iv) conduct model validation to determine whether the resulting approximations (base components) are non-blocking.

6.2.1 Benchmark Model

The wind turbine modeling is done with MATLAB/Simulink [267]. Figure 6.9 displays the top-level view of the Simulink diagram and Figure 6.10 depicts the plant of the wind turbine. The wind turbine dynamics are nonlinear functions of the operating point and they are determined by the wind speed, the rotor speed, and the blade pitch angle.

The plant comprises three subsystems: an aeroelastic, a servo-elastic, and a pitch-actuator. The servo-elastic subsystem contains linear operators and is linked with the aeroelastic subsystem through signals that represent the aerodynamic torque and thrust. The aeroelastic subsystem has several nonlinear blocks, i.e. two fourth-order *Polynomials*, two *Products*, two *Square* functions, and two *Divisions*. These polynomials represent the aerodynamic torque coefficient (cP) and thrust coefficient (cT). They are computed through a MATLAB regression model. The pitch actuator dynamics can be expressed by a first-order lag, a second-order lag, or a time-delay. The inputs of pitch actuator subsystem are the pitch actuator type and the requested pitch angle. Its output is the actual pitch angle which is connected with to the aeroelastic subsystem. It includes several linear operators as well as a *Multiport Switch*, three *Enable* blocks, three *Compare to constant* blocks, and three *Enabled Subsystems*.

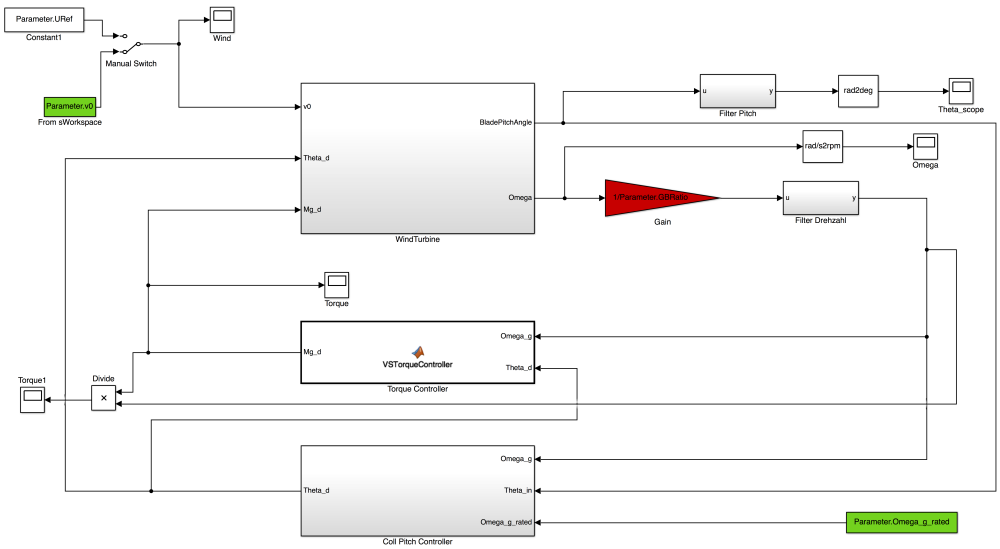


Figure 6.9: Wind turbine - Simulink model - Top Level [267].

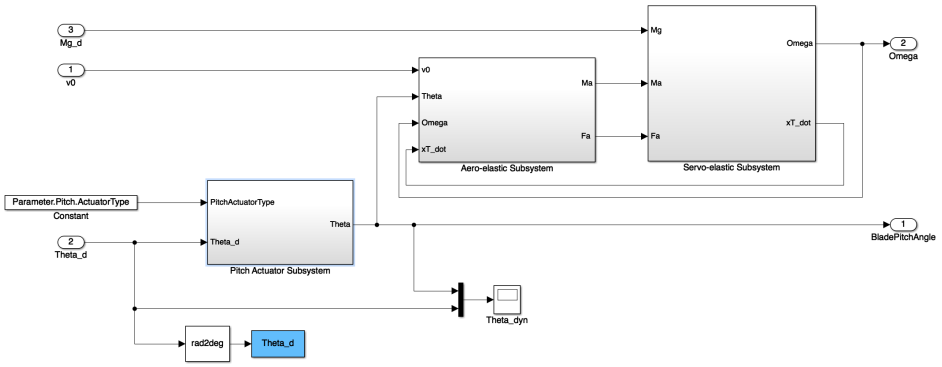


Figure 6.10: Wind turbine - Simulink model - Plant [267].

The wind turbine has two controllers: a collective blade-pitch controller and a generator-torque controller. On the one hand, the collective blade-pitch controller is a gain-scheduled PID and its aim is to minimize the error between the rated and the filtered generator speed. It includes an anti-windup scheme to prevent integration wind-up whenever the actuator is saturated. It contains linear, hybrid blocks and two nonlinear ones, a *Division* and a *Product*. On the other hand, the objective of the generator-torque controller is to maximize the extracted maximum power from the wind via tracking the optimal tip-speed ratio λ_{opt}. In essence, it forms a hybrid controller with 2 inputs and 5 5 discrete states signals. Its inputs correspond to the pitch angle and the filtered generator speed. It is defined as an *Embedded MATLAB Function* and it contains two nonlinearities, a *Square* and a *Division*.

In total, the model contains more than 10 distinct nonlinearities. Excluding the memory blocks, the clocks, and the rate limiters, the model states vary from 5 to 7.

6.2.2 Model Transformation

This section describes the model transformation, presenting the model adaptation, translation, hybridization, and validation steps.

Model Adaptation

The goal of model adaptation is either to abstract or enrich the verification model. For example, let us look at the Simulink block that determines the wind speed profile. The wind profile is an external input to the Simulink model read from the MATLAB workspace. The input is a discrete time signal but it has 33 possible profiles; saved in the the `aeromaps.mat` file. In principle, such a profile can be translated into a hybrid automaton, but this would not be very efficient. As such, we add a nondeterministic variable that covers all possible combinations and is bounded between the minimum and the maximum allowed wind speeds. Note it is also possible we can set bounds on the rate of change of the wind speed. We further modify the Simulink model by replacing or deleting several blocks, e.g. *Scopes*, *Save to workspace Mux*, *Demux*, *Enabled Subsystems*, *Manual Switches*.

Translation

The second step of the proposed model transformation is to translate the Simulink model into an equivalent SpaceEx model using the SL2SX translator. The translation is almost instantaneous and generates an SX file with 18 Network and 89 Base Components. The Simulink blocks that can be automatically translated into corresponding base components are as follows: *Add, Subtract, Divide, Multiply, Constant, Gain, Saturation, Integrator, Subsystem, Inport, Outport*. Nonetheless, there are Simulink blocks that can be exactly translated, because they can not be expressed as PWA hybrid automata.

Note that it is feasible to make the SpaceEx model more compact by reusing base components and replacing the components that are duplicate or serve exactly the same role. Such configuration makes the total number of base components to be reduced approximately up to 30 distinct ones. Figure 6.11 and Figure 6.12, respectively, present the top level block (network

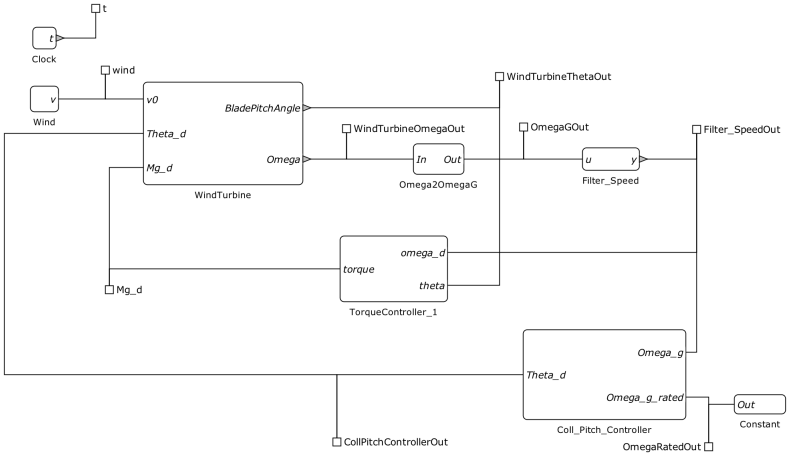


Figure 6.11: Wind turbine - SpaceEx model - Top Level.

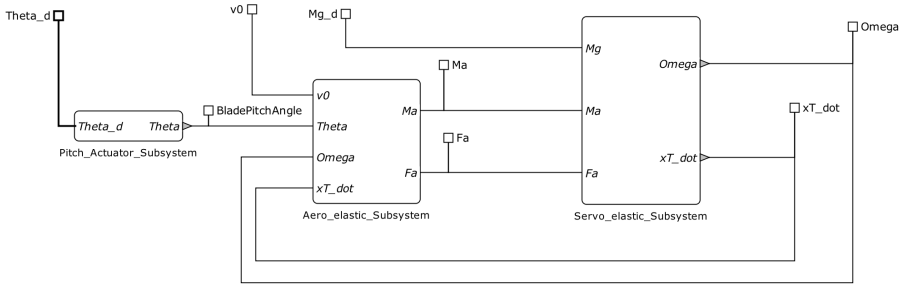


Figure 6.12: Wind turbine - SpaceEx model - Plant.

component) and the plant subsystem of the wind turbine model in SX format, as shown in the Model Editor of SpaceEx.

Hybridization

The third step of the model transformation is to obtain PWA approximations and express them in SX format of all Simulink blocks that cannot be handled automatically. This step concerns cases where no exact translation is available, e.g. nonlinear functions, or the translation procedure cannot be applied, e.g. Embedded MATLAB or S-Functions.

To obtain PWA approximations of nonlinear dynamics, we use the proposed syntactic hybridization method. A useful intermediate step is to obtain bounds on the minimum and maximum behaviors of the inputs of the nonlinear Simulink blocks. Note that the shorter the ranges of the signals, the smaller the number of locations required by the PWA abstraction. In this work, we run Simulink simulations for different scenarios and initial

conditions to obtain a (not necessarily conservative) estimation of the signal range.

For this wind turbine model, the constructed base components only have one or two input signals and one output signal, the state space is partitioned into a set of boxes, the approximations are linearized around the center of each domain, and the abstraction (linearization) error is computed by evaluating the Lagrange remainder. The maximum error value is computed for each location (box). We use a combination of interval arithmetic and global optimization to bound it [17]. All the computations (linearization domains, quantization parameters, operating points, PWA approximations, errors) and the model files integration are conducted with MATLAB.

Notice that the approximations can be conservative or non-conservative. The error is first computed for each distinct location of the base components and then it is added as a nondeterministic input in the respective invariants. It is also possible to ignore the errors for debugging purposes or select a single error value, e.g. average, in order to generate a deterministic model. Table 6.1 presents the Simulink blocks that have to be approximated along with the maximum induced errors and the corresponding number of locations of each approximation. Note that the error corresponds to the maximum error that appears in one of the locations of the PWA approximation. The individual errors in the remaining locations can be significantly smaller.

The resulting model only has 72 locations in all components combined. To conduct reachability analysis, SpaceEx carries out on-the-fly composition of the hybrid automata and instantiates only the reachable parts of the state-space. Notice that the upper bound of the number of locations of the composed model is 16 millions.

Model Validation

The fourth step concerns the model validation. We present an empirical approach to check whether the generated base components are correct to the extent that they yield satisfactory behaviors and do not cause any deadlocks. In this step, we basically test the implementation. In this respect, we have generated a signal library with predefined components that define *step*, *trigonometric*, and *ramp* functions. We have added an extra base component that assists with visualizing the outputs[1]. Once we construct a new approximation, we integrate it within the SX library and obtain a *tester module* (containing both base and network components).

[1]Certain SpaceEx algorithms do not output algebraic variables, so we mapping the algebraic variable to the solution of an ODE.

No	Block Type	# Loc	Error Bounds	Info
1	Product *(Anti windup)*	4	2.00	$x \cdot y$
2	Division (C_p/λ)	10	2.76e-2	x/y
3	Division (*GS factor*)	2	2.01e-2	$\frac{1}{1+x}$
4	Division (λ)	4	6.98e-1	x/y
5	Product ($C_T \cdot v_{ref}^2$)	4	25.6	$x^2 \cdot y$
6	Product ($C_p/\lambda \cdot v_{ref}^2$)	4	3.23	$x^2 \cdot y$
7	Polynomial (*aeromaps*)	6	3.39	4^{th}-order
8	Polynomial (*aeromaps*)	6	12.3	4^{th}-order
9	Embedded MATLAB	28	10.5	x^2 and $1/x$
10	Saturation	3	-	exact
11	Read from workspace	1	-	aux. variable
12	Mux, Scope	-	-	-
13	Save to workspace	-	-	-
14	Enabled Subsystem	-	-	-
15	Multiport Switch	-	-	-
16	Compare to Constant	-	-	-
17	Manual Switch	-	-	-
18	Rate Limiter	–	–	DAE
19	Memory	–	–	cont. delay

Table 6.1: Wind Turbine Benchmark – breakdown into blocks. Blocks, 1-9, are approximated syntactically; the 10^{th} block is exactly translated; 11^{th} is replaced by a nondeterministic input; blocks, 12-17, are not necessary due to semantic differences; and the rest, 18-19, are ignored to mitigate further state space explosion.

The tester modules check that the approximation is non-blocking and is indeed an over-approximation. For the first objective, the input signals are selected in a way that compels the PWA automaton (introduced by the hybridization process) to necessarily visit all of its locations. As soon as the input signals exceed the allowed operating range, the automaton goes out-of-bounds and the analysis terminates. That means that a fixed point has been reached. For the second objective, we perform reachability analysis on the tester module (containing the approximation and the input signals) and then compare the results with random simulations. In this way, we check whether the simulations are included in the reachable sets.

For illustration purposes, let us consider a Simulink block of the aero-elastic subsystem of the wind turbine's plant. Figure 6.13 shows the entire Simulink aero-elastic subsystem. The block under study is the *Divide* block and it is highlighted in red.

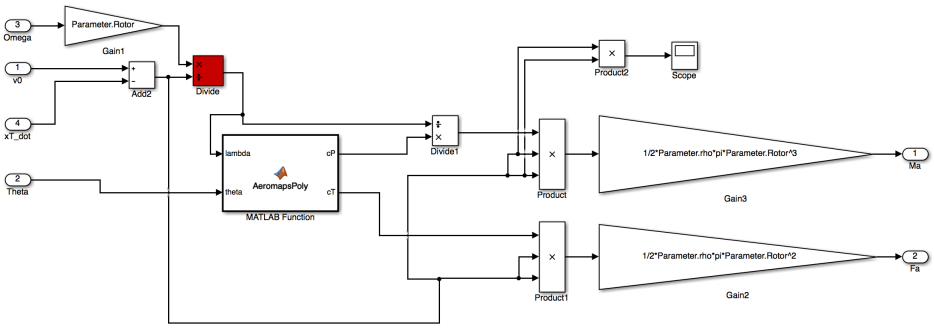


Figure 6.13: Aero-elastic subsystem - Simulink - Nonlinear *Divide* block shown in red.

This block describes a division operator for two input signals that continuously evolve over time. Physically, this block computes the tip-speed ratio, a dimensionless variable described by *lambda* (λ). This block is nonlinear and, as such, is replaced by a PWA over-approximation that contains 4 locations. Figure 6.14 shows the resulting SpaceEx base component. The variables *In1*

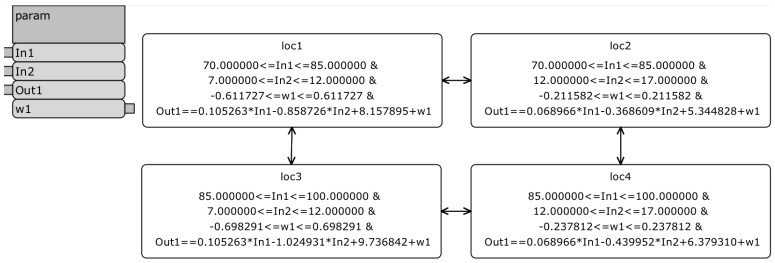


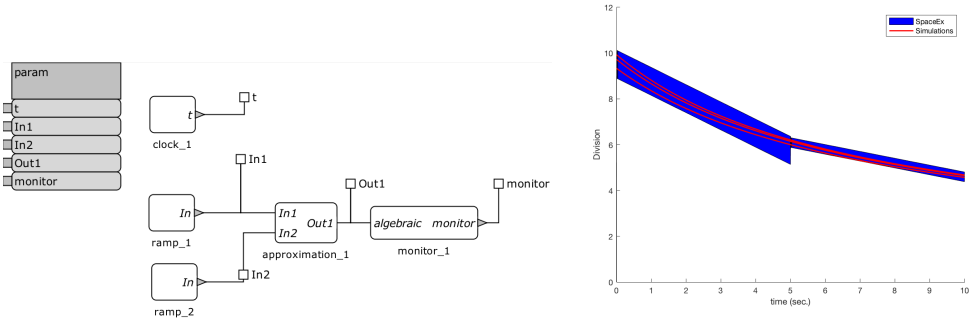
Figure 6.14: *Divide* block - PWA approximation in SpaceEx.

and *In2* denote the two inputs, whereas *Out1* is the PWA approximation of the division operator *In1/In2*. The approximation error $w1$ is included as a nondeterministic disturbance. Noticee that all the variables should be uncontrolled, and the error term should be included in the invariant, denoted as a local variable; see Section 2.4 for detailed information regarding the modeling language of SpaceEx.

After integrating the base component of the *Divide* block with our input library, we want to check the previously mentioned objectives (no deadlocks and over-approximation). We select the inputs to be ramp signals with varying initial conditions. The first input is described by $\frac{\mathrm{d}In1}{\mathrm{d}t} = 1$ with initial conditions in $69 < In1 < 71$ and the second input by $\frac{\mathrm{d}In2}{\mathrm{d}t} = 1$ with initial conditions in $6.9 < In2 < 7.1$. Then, we conduct reachability analysis and compare the results against random simulations of the original nonlinear function. In this way, we observe that for the considered scenario SpaceEx

yields an over-approximation. The tester module is depicted in Figure 6.15a, the reachable sets and the simulation runs are shown in 6.15b.

The same approach, in a hierarchical manner, is employed to guarantee that compositions of multiple base components, network components or larger subsystems operate correctly.



a) *Tester - Division λ.* **b)** *Reachable sets and Simulations.*

Figure 6.15: SpaceEx Base Component - Division λ - Model Validation. The reachable sets of the approximate SpaceEx block are shown in blue and the simulation runs of the nonlinear function in red. The simulation runs are indeed enclosed in the reachable sets.

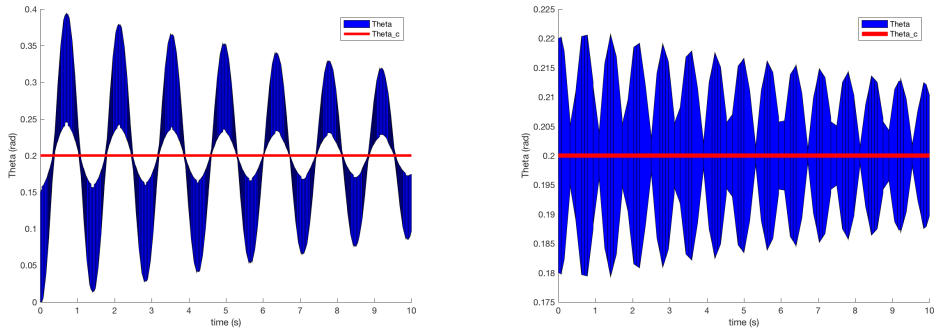
6.2.3 Reachability Results

In this part, we provide validation runs for selected Simulink subsystems, which correspond to SpaceEx network components. We begin with the **pitch-actuator** subsystem of the wind turbine plant. The input of this block is the commanded pitch angle and the output is the actual pitch angle. Opting for the STC (space-time with clustering) scenario, a global time horizon of 10s, and a constant input signal, we obtain a fixed point. Figure 6.16 depicts the reachable sets for different initial conditions. The input signals are depicted in red and the output signals in blue.

Next, we consider the **servo-elastic** subsystem of the wind turbine plant. Considering constant inputs, the STC scenario and a global time horizon of 20s, SpaceEx again terminates successfully. Figure 6.17 illustrates the reachable sets for initial conditions that can be in the following sets: $0 \leq$ Omega ≤ 0.1 and $0 \leq$ xT_dot ≤ 0.1.

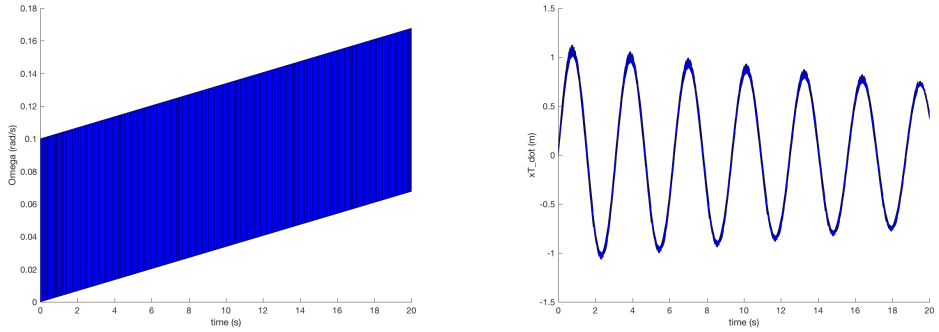
Afterwards, we analyse the **torque controller**. The resulting approximation comprises 28 locations, has two input signals (omega_d, theta) and one output (torque). We select the input signals of Figure 6.19 (a ramp and a sine function), as they cover, when combined, the entire 2-dimensional operating range (from minimum to maximum allowed values). Actually,

with this configuration, SpaceEx visits all the controller locations and is able to find a fixed point after 16 seconds. The reachable sets are computed with the PHAVer scenario; PHAVER stands for polyhedral hybrid automaton verifier and it uses unbounded integer arithmetic and guarantees over-approximations. The reachable sets of the torque controller are shown in Figure 6.18.



a) *Initial Conditions* - $0 \leq \theta \leq 0.15$. **b)** *Initial Conditions* - $0.18 \leq \theta \leq 0.22$.

Figure 6.16: Reachable sets of pitch actuator (blue: Output, red: Input).



a) *Output - Omega* **b)** *Output - Displacement (xT_dot).*

Figure 6.17: Reachable sets servo-elastic subsystem.

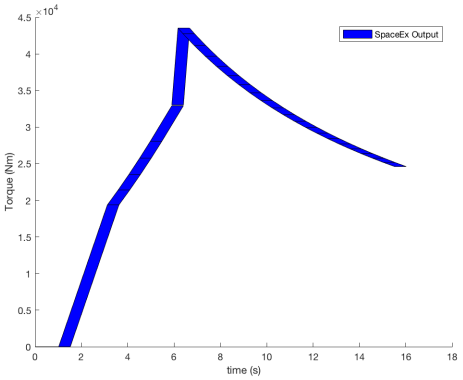
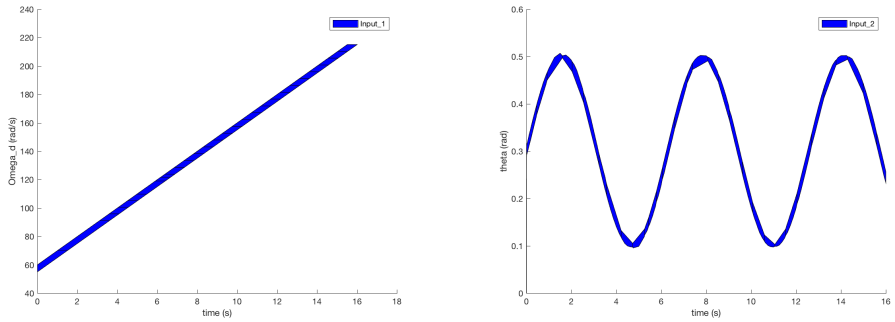


Figure 6.18: Reachable sets of torque controller.



a) *Input 1 - Omega_d.* **b)** *Input 2 - Theta.*

Figure 6.19: Selected Input signals for torque controller.

6.3 Lane Change Manoeuvre for Autonomous Vehicles

Lane changes are considered to be risky manoeuvres both for human drivers and autonomous vehicles. This the case as they require changes in lateral and longitudinal velocities in the presence of other moving vehicles [77]. According to the National Highway Traffic Safety Administration, the main cause of lane change accidents is failure to detect the other vehicle and almost 80% of the accidents occur at speeds smaller than 25km/h [154].

Lane changes manoeuvres can be divided on the basis of the existence of road infrastructure [170] or of a reference trajectory [238]. Predefined trajectories are known as motion primitives [123]. Motion primitives are computed offline and can be used to construct manoeuvre automata. Construction of formally verified manoeuvre automata using reachability analysis

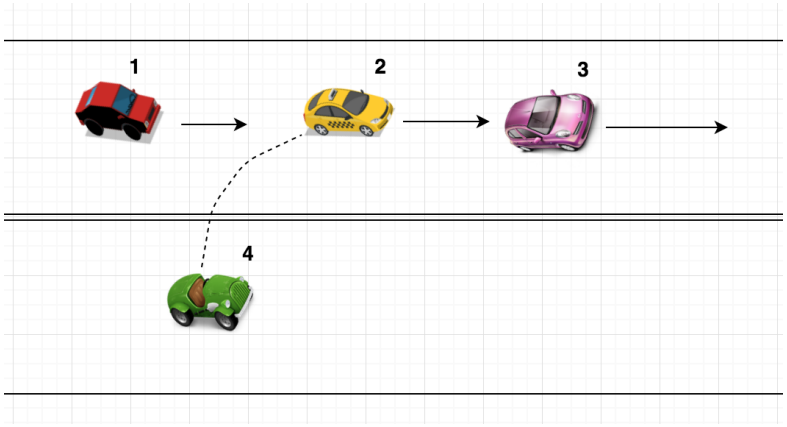


Figure 6.20: Lane Change manoeuvre - Car in the right lane merging to the left lane.

is investigated in [12, 167]. Examples of control designs tailored to lane change manoeuvres are tube-based MPC [143] and convex interpolation control [269]. As for metrics required to validate the suitability of a lane change manoeuvre, one can employ the time to collision, time headway, time to line crossing, or the inter-vehicle traffic gap [150, 238, 289].

In this Section, we present a benchmark modeling a cooperative lane change manoeuvre that includes four fully autonomous vehicles. Three vehicles drive in the left lane and one is in the right lane. The vehicle in the right lane wants to move to the left whereas avoiding a collision (see Figure 6.20). Each vehicle is equipped with sensors and can communicate with its neighboring vehicles. The vehicle dynamics are expressed by a dynamic bicycle model and each vehicle hash a linear low-level controller that regulates its own lateral and longitudinal behavior. In order to guarantee that the manoeuvre is safe and the traffic rules are fulfilled, we use a cooperative driving control scheme (supervisory logic) that determines the actions of each vehicle.

6.3.1 System Description

In this part, we present the lane change scenario, the system specifications, the modeling part, and the control design. We consider that there are three vehicles in the left lane that have already formed a platoon and they move with the same speed. The desired speed is assumed to be predefined and is an external (constant) input to our system. We also assume that the manoeuvre has already been requested, e.g. by a higher layer (road infrastructure) or due to an emergency (obstacle avoidance). The vehicle in the right lane has to change its lane and merge to the left one. We consider that the

vehicle should move in between two specific vehicles, i.e. tail (no. 1) and interior (no. 2). This manoeuvre can be seen as an automated merging manoeuvre [212], i.e. how to insert a vehicle from on-ramp in the middle between two pre-selected vehicles of a platoon in the main lane.

The lane change manoeuvre consists of two phases. The first phase is the preparation of the manoeuvre and the second is the manoeuvre itself. Once the manoeuvre is completed, the platooning formation should be achieved. In other words, the desired inter-vehicle distances and speeds should be reached.

We assume that the vehicles are equipped with orientation, position, and velocity sensors; thus, all the state variables can be measured. Also, the vehicles can communicate their longitudinal position and speed with their neighboring vehicles.

Specifications

The specifications that the vehicle platoon should satisfy are taken from [161, 238]. Each vehicle should

1. maintain safety margins with all the surrounding vehicles,
2. respect the traffic rules, and
3. satisfy physical and design limitations.

More precisely, these specifications can be expressed as

1a. the distance of two neighboring vehicles should never be smaller than a given threshold[2],

1b. the vehicles of the platoon should maintain a constant time gap (t_{gap}) between each other; this gap is analogous to their speed,

2a. the manoeuvre should only be initiated if the time gap (also known as time-to-collision [293])[3] is greater than a given value (t_{gap_m}),

2b. eventually (once the manoeuvre is finished) the vehicles should form a platoon and the velocity of all vehicles should reach $||v_{des} \pm \varepsilon||$, where ε is a user-defined metric,

3a. the cars cannot exceed the practical velocity bounds, e.g. $v_{min} \leq v \leq v_{max}$, and

[2]By vehicle distance, we assume the difference between the longitudinal positions of two vehicles with respect to the center of gravity in their rear axles.

[3]Time-to-collision (TTC) corresponds to the time required for two vehicles to collide if they continue at their present speed and on the same path.

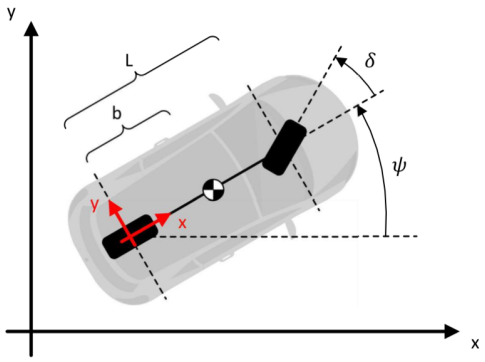


Figure 6.21: Bicycle Model - Physical Interpretation [180]

3b. the control inputs are bounded, i.e. $\alpha_{min} \leq \alpha_x \leq \alpha_{max}$ and $\delta_{min} \leq \delta \leq \delta_{max}$.

Note that these time gaps will be used for the control design part. In addition, we consider a desired platoon speed v_{des} and a constant time gap t_{gap}. As a result, the distance between two vehicles in the platoon should not be smaller than $v_{des} \cdot t_{gap}$. During the lane change, the time gap that the merging vehicle should respect is $t_{gap\_m} < t_{gap}$. The numerical values of these parameters are given in Table 6.5. These requirements can be encoded as reachability/safety problems via the monitors of Section 4.3.

Vehicle Dynamics

For vehicle dynamics, there exists a large variety of models ranging from simpler to more complicated. Typically, the more complicated the model is, the more accurately it captures the physical behavior of the vehicle [257]. Among others, one could select from point-mass, kinematic, dynamic, and multi-body models, see e.g. [16]. In the literature of autonomous vehicles, dynamic and kinematic bicycle models are commonly used [194]. Unlike kinematic models, dynamic bicycle models consider the tire slip. Note that the standard bicycle model is also known as single-track model [257]. As for tire models, there are various alternatives from linear to nonlinear ones [242].

In this work, we consider a dynamic bicycle model with a linear tire model (see Figure 6.21). We select a dynamic model in order to incorporate the tire slip given the fact that it can express important phenomena, such as understeering and oversteering. At the same time, the model is assumed linear to avoid computational complexity. Similar to [257, Chapter 2], the state vector contains:

- the longitudinal position of the rear axle x_r,
- the lateral position of the rear axle y_r,
- the yaw angle ψ,
- the longitudinal velocity v_x,
- the lateral velocity at the center of the rear axle v_y,
- the yaw rate ω.

The model inputs are the longitudinal acceleration a_x and the steering angle δ. The state vector is completely measured and we model additive measurement noise in all state dimensions. The disturbances are defined as three normalized forces, with the error force $e^d_{f_x}$ acting in longitudinal direction, $e^d_{f_{y,f}}$ acting in lateral direction at the front axle and $e^d_{f_{y,r}}$ acting in lateral direction at the rear axle. In vector form, the state vector is expressed as

$$x = [x_r, y_r, \psi, v_x, v_y, \omega]^{\mathrm{T}} \qquad \text{state wrt. rear axle} \qquad (6.1)$$

$$u = [a_x, \delta]^{\mathrm{T}} \qquad \text{input, measured} \qquad (6.2)$$

$$y = [x_r, y_r, \psi, v_x, v_y, \omega]^{\mathrm{T}} \qquad \text{measurement} \qquad (6.3)$$

$$\nu = [e^m_{x_r}, e^m_{y_r}, e^m_{\psi}, e^m_{v_x}, e^m_{v_y}, e^m_{\omega}]^{\mathrm{T}} \qquad \text{meas. err.} \qquad (6.4)$$

$$w = [e^d_{f_x}, e^d_{f_{y,f}}, e^d_{f_{y,r}}]^{\mathrm{T}} \qquad \text{disturbance} \qquad (6.5)$$

Starting from first-principles, as shown in [257], the differential equations of the dynamic bicycle model are defined:

$$f_B(x,u) = \begin{cases} \dot{x}_1 = x_4\cos(x_3) - x_5\sin(x_3) \\ \dot{x}_2 = x_4\sin(x_3) + x_5\cos(x_3) \\ \dot{x}_3 = x_6 \\ \dot{x}_4 = u_1 + x_5 x_6 + w_1 \\ \dot{x}_5 = f_{y,f}(x,u,w) + f_{y,r}(x,w) - x_4 x_6 \\ \dot{x}_6 = a\frac{m}{J}\left(f_{y,f}(x,u,w)\right) - b\frac{m}{J}\left(f_{y,r}(x,w)\right) \end{cases}$$

with the normalized front and rear lateral forces $f_{y,f}(x,u), f_{y,r}(x)$ given as

$$f_{y,f}(x,u) = -c_f\mu g\frac{b}{a+b}\left(\frac{x_5 + (a+b)\cdot x_6}{x_4} - u_2\right) + w_2$$

$$f_{y,r}(x) = -c_r\mu g\frac{a}{a+b}\frac{x_5}{x_4} + w_3.$$

Note that we assume a global coordinate system and the variables are described in absolute coordinates. The state of the system is often expressed

Table 6.2: Parameters for bicycle model

Description	Symbol	Value
wheelbase (m)	L	2.7
gravitational constant (m/s^2)	g	9.81
friction coefficient	μ	0.8
distance from front wheels to center of gravity	a	$(1-\frac{b}{L})\cdot L$
ratio of mass to rotational inertia (m^2/ s^2)	I_z/m	1.57
relative position of center of gravity	b/L	0.57
relative front tire stiffness	c_f	10.8
relative rear tire stiffness	c_r	17.8

in different coordinates [167], e.g. using a polar-coordinate representation of the vehicle velocity, which is defined by the slip-angle $\beta = \arctan(v_y/v_x)$ and the absolute velocity $v_{abs} = \sqrt{v_x^2 + v_y^2}$, with the direction of motion $\theta = \psi + \beta$.

The model parameters that we opt for are taken from [161] and are provided in Table 6.2. More details about the parameter estimation and identification can be found at [95]. The nonlinear model is linearized around a set of operating points using standard point-wise linearization, e.g. [169]. For linearization purposes, we consider $x_{op} = [0, 70/3.6, 0, 0, 0, 0]$, meaning that we linearize around the nominal platoon conditions. Finally, we obtain the state-space representation

$$\begin{aligned} \dot{x} &= A \cdot x + B \cdot u + B_d \cdot w \\ y &= C \cdot x + v \end{aligned} \tag{6.6}$$

$$A \approx \begin{bmatrix} 0 & 0 & 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 19.4444 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0000 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5.5739 & -17.5748 \\ 0 & 0 & 0 & 0 & 1.1909 & -6.7936 \end{bmatrix} \quad B \approx \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1.0000 & 0 \\ 0 & 48.3123 \\ 0 & 35.7265 \end{bmatrix},$$

$$B_d \approx \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1.0000 & 0 & 0 \\ 0 & 1.0000 & 1.0000 \\ 0 & 0.7395 & -0.9803 \end{bmatrix} \text{ and } C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

$\nu_1 : e^m_{x_r}[m]$	$\nu_2 : e^m_{y_r}[m]$	$\nu_3 : e^m_{\psi}[^\circ]$	$\nu_4 : e^m_{v_x}[m/s]$	$\nu_5 : e^m_{v_y}[m/s]$	$\nu_6 : e^m_{\omega}[^\circ/s]$
0.04	0.04	1	0.05	0.05	2

Table 6.3: Maximum values of measurement errors

$w_1 : e^d_{f_x}[m/s^2]$	$w_2 : e^d_{f_{y,f}}[m/s^2]$	$w_3 : e^d_{f_{y,r}}[m/s^2]$
0.1	0.057	0.043

Table 6.4: Maximum values of disturbances

The maximum disturbances and maximum measurement errors are taken from [161] and are given in Tables 6.3 and 6.4. Note that the values of the A and B matrices that we present here are rounded. For the computations, we use the double-precision values that can be found at the associated MATLAB files[4]. We opt for a linear model for computational reasons; such a model is amenable to efficient reachability algorithms for PWA dynamics.

Low-level Control

The objective of the low controller of each vehicle is to regulate its position and velocity in accordance with the behavior of the other vehicles. At the same time, the lane change manoeuvre should be safe. Once the manoeuvre is completed, the vehicle platoon should maintain the predefined vehicle speed v_{des}. The control scheme is shown in Figure 6.22.

As all the vehicles are described by the same physical model, we use the same low-level controller for all vehicles. In particular, we utilize a linear controller

$$u = -K \cdot y = -K \cdot (x + v)$$

that renders the closed-loop system asymptotically stable. We opt for an LQR (Linear Quadratic Regulation) controller since it is a well established design technique that provides practical feedback gains. LQR is an optimal multivariable feedback control approach which minimizes the deviation of the state trajectories of the closed-loop system while requiring minimum controller effort. The behavior of an LQR controller is determined by two parameters: state and control weighting matrices. These two matrices are design parameters and influence the success of the LQR controller synthesis [162].

[4]The source code of this benchmark model has been published online at https://cps-vo.org/group/ARCH/benchmarks.

Employing the Bryson's rule we arrive at the following weighting matrices

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/180 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5/180 \end{bmatrix}, \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 180/\pi \end{bmatrix}, \text{and } N = \mathbf{0}_{6,2},$$

we numerically solve the continuous Algebraic Ricatti Equation with MATLAB and obtain our state-feedback matrix

$$K = \begin{bmatrix} 1.0000 & 0 & 0 & 2.6458 & 0 & 0 \\ 0 & 0.1321 & 1.6970 & 0 & 0.0457 & 0.2829 \end{bmatrix}.$$

Checking the eigenvalues of $A - BK$ matrix have $\text{Re}(\text{eig}(A - BK)) < 0$, we validate that the system is indeed stable.

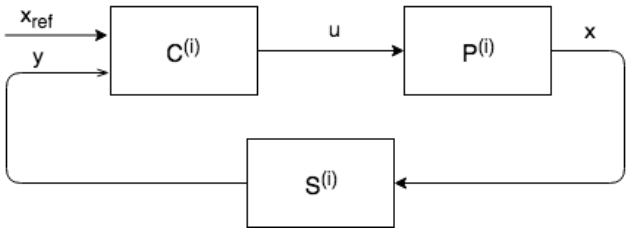


Figure 6.22: Low-level control loop - $C^{(i)}$ is the controller, $S^{(i)}$ is the sensor module and $P^{(i)}$ is the plant of the i^{th} vehicle.

Supervisory Control

The next step is to define the references (set points) x_{ref} for the controller of each vehicle. We employ a consensus-based cooperation scheme where the set points of each vehicle are defined by the states of its neighboring vehicles. As such, the controllers are expressed by

$$\begin{aligned} u &= -K(y - x_{ref}) = -K(x - x_{ref} + v) \text{ where} \\ x_{ref} &= [x_{r,ref}, y_{r,ref}, \psi_{ref}, v_{x,ref}, v_{y,ref}, \omega_{ref}]. \end{aligned} \tag{6.7}$$

Note that the length of a lane is assumed to be 3.5 meters. The control inputs are bounded. In particular, the acceleration remains in the range

$$-3 \leq u_1 \leq 2.$$

For the steering angle, we consider that

$$-\pi/4 \leq u_2 \leq \pi/4.$$

However, the reference values are different for each vehicle.

Leader Vehicle. For the leader vehicle in the left lane (no. 3, pink in figures), we select

$$\begin{aligned} v_{x,ref}^{(3)} &= \max\Big(v_{des}, v_x^{(2)}\Big) \\ x_{r,ref}^{(3)} &= \max\Big(x_r^{(3)}, x_r^{(2)} + t_{gap} \cdot v_x^{(2)}\Big) \\ y_{r,ref}^{(3)} &= 0, \;\; \psi_{ref}^{(3)} = 0, \;\; v_{y,ref}^{(3)} = 0, \text{and } \omega_{ref}^{(3)} = 0. \end{aligned} \tag{6.8}$$

In essence, the leader vehicle should maintain the desired platoon velocity or accelerate when the vehicle behind it starts accelerating. Accordingly, the reference position is defined.

Rear Vehicle. For the vehicle at the tail of the platoon in the left lane (no. 1, red in figures), we select

$$\begin{aligned} v_{x,ref}^{(1)} &= \min\Big(v_x^{(2)}, v_x^{(4)}\Big) \\ x_{r,ref}^{(1)} &= \min\Big(x_r^{(2)} - t_{gap} \cdot \;\; v_x^{(2)}, x_r^{(4)} - t_{gap} \cdot v_x^{(4)}\Big) \\ y_{r,ref}^{(1)} &= 0, \;\; \psi_{ref}^{(1)} = 0, \;\; v_{y,ref}^{(1)} = 0, \text{and } \omega_{ref}^{(1)} = 0. \end{aligned} \tag{6.9}$$

That means that the rear vehicle should maintain the smaller speed between the merging vehicle (no. 4) and the vehicle in front of it (no. 2) to avoid any crashes.

Interior Vehicle. For the vehicle in the middle of the platoon in the left lane (no. 2, yellow in figures), we select

$$\begin{aligned} v_{x,ref}^{(2)} &= v_x^{(3)} \\ x_{r,ref}^{(2)} &= \max\Bigg(\frac{x_r^{(2)} - t_{gap} \cdot v_x^{(2)} + \max(x_r^{(1)} + t_{gap} \cdot v_x^{(1)}, x_r^{(4)} + t_{gap} \cdot v_x^{(4)})}{2}, \\ &\qquad\qquad x_r^{(2)} - t_{gap} \cdot v_x^{(2)}\Bigg) \\ y_{r,ref}^{(2)} &= 0, \;\; \psi_{ref}^{(2)} = 0, \;\; v_{y,ref}^{(2)} = 0, \text{and } \omega_{ref}^{(2)} = 0. \end{aligned} \tag{6.10}$$

That means that this vehicle should accelerate if there is not enough space for the merging vehicle to do the lane change. At the same time, it should respect the velocity speed of the leading vehicle.

Merging vehicle. For the vehicle in the right lane (no. 4, green in figures), which is doing the manoeuvre, we consider two phases. The first one is the pre-processing/preparation step where the vehicle needs to check if and when it is feasible to do the manoeuvre. In essence, the vehicle needs to regulate its velocity with respect to the platoon velocity (while guaranteeing that there is enough space margin). For that case, we choose

$$\begin{aligned} v_{x,ref}^{(4)} &= v_x^{(2)} \\ x_{r,ref}^{(4)} &= \max\left(x_r^{(2)} - t_{gap}\cdot v_x^{(2)}, \frac{x_r^{(2)} - t_{gap}\cdot v_x^{(2)} + x_r^{(1)} + t_{gap}\cdot v_x^{(1)}}{2}\right) \\ \boldsymbol{y_{r,ref}^{(4)} = 0}, \;\; & \psi_{ref}^{(4)} = 0, \;\; v_{y,ref}^{(4)} = 0, \text{and } \omega_{ref}^{(4)} = 0. \end{aligned} \tag{6.11}$$

In the second phase, the vehicle starts the merging manoeuvre. This practically means that its lateral position should change. To do so, the lateral position reference $y_{r,ref}$ should be modified.

$$\begin{aligned} v_{x,ref}^{(4)} &= v_x^{(2)} \\ x_{r,ref}^{(4)} &= \max\left(x_r^{(2)} - t_{gap}\cdot v_x^{(2)}, \frac{x_r^{(2)} - t_{gap}\cdot v_x^{(2)} + x_r^{(1)} + t_{gap}\cdot v_x^{(1)}}{2}\right) \\ \boldsymbol{y_{r,ref}^{(4)} = 3.5}, \;\; & \psi_{ref}^{(4)} = 0, \;\; v_{y,ref}^{(4)} = 0, \text{and } \omega_{ref}^{(4)} = 0. \end{aligned} \tag{6.12}$$

The condition that should be valid to initiate a safe lane change (transition from phase 1 to 2) is

$$\phi_{12} := \{x_r^{(4)} < x_r^{(2)} - t_{gap\_m}\cdot v_x^{(2)} \text{ and } x_r^{(4)} > x_r^{(1)} + t_{gap\_m}\cdot v_x^{(1)}\}. \tag{6.13}$$

The switching logic can be visualized in Figure 6.23. The parameter values and initial conditions used for simulation purposes are shown in Tables 6.5 and 6.6 respectively.

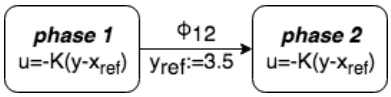


Figure 6.23: Schematic of lane change manoeuvre - 2 phases of merging vehicle movement

Description	Symbol	Value
minimum velocity (km/h)	v_{min}	0
maximum velocity (km/h)	v_{max}	150
minimum acceleration (m/s^2)	α_{min}	-3
maximum acceleration (m/s^2)	α_{max}	2
minimum steering angle (rad)	δ_{min}	$-\pi/4$
maximum steering angle (rad)	δ_{max}	$\pi/4$
desired platoon vehicle (km/h)	v_{des}	70
time gap (sec.)	t_{gap}	1.5
minimum time gap (sec.)	t_{gap_m}	1
constant gap (m)	gap	$\frac{70}{3.6} * 1.5$

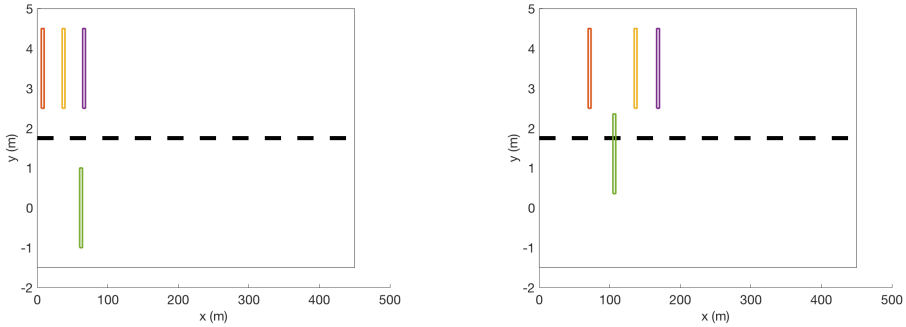
Table 6.5: Parameters for control design

Vehicle	i	$x_1^{(i)}(0)$	$x_2^{(i)}(0)$	$x_3^{(i)}(0)$	$x_4^{(i)}(0)$	$x_5^{(i)}(0)$	$x_6^{(i)}(0)$
Rear	1	0	3.5	0	v_{des}	0	0
Interior	2	gap	3.5	0	v_{des}	0	0
Leader	3	$2 * gap$	3.5	0	v_{des}	0	0
Merging	4	$2 * gap$	0	0	$v_{des}/2$	0	0

Table 6.6: Initial conditions of each vehicle

6.3.2 Simulation Results

The simulation results for the specified scenario are depicted in Figure 6.24. The 2D images have been constructed using the plot and patch commands of MATLAB. Figure 6.25 shows the position of the vehicle doing the lane change and the evolution of all vehicles velocity over time. Notice that in Figure 6.25b the rear vehicle initially needs to decelerate, while the interior and lead vehicles need to accelerate. As soon as there is enough space for the merging vehicle to change lane, the velocities of all vehicles stabilize around the desired platoon speed. For the 3D images, we have used Unity game engine [117]. After setting up the scenario in Unity, we simulate the lane change manoeuvre and get a 3D video [149]. Figure 6.26 illustrates the main instances of the manoeuvre and it is obtained through Unity image capture.



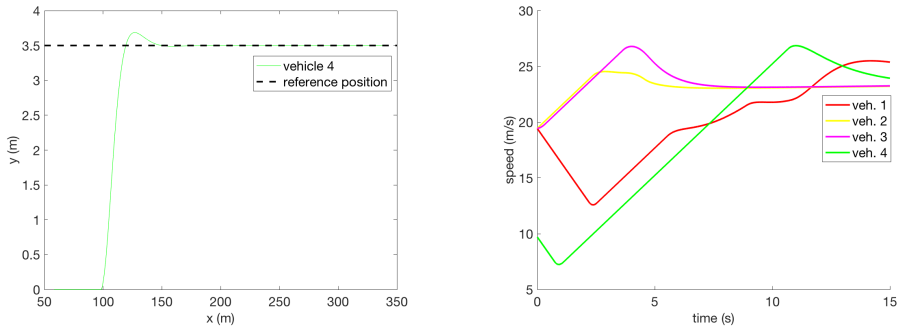
a) *Initial Phase (before manoeuvre)*

b) *Manoeuvre initiated*

c) *Manoeuvre just completed*

d) *Pllatoon stabilized*

Figure 6.24: Graphical illustration of a lane change manoeuvre - Simulation results with MATLAB



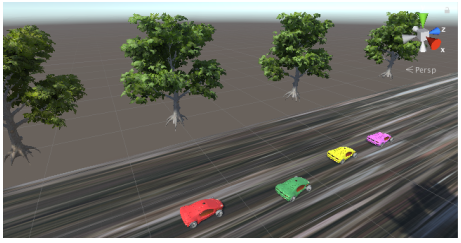
a) *Evolution of maneuvering vehicle position*

b) *Evolution of vehicles velocities over time*

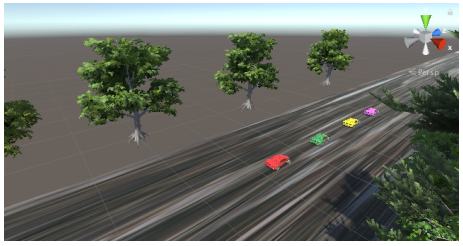
Figure 6.25: Graphical illustration of a lane change manoeuvre - Simulation results with MATLAB

a) *Initial Phase (before manoeuvre)*

b) *Manoeuvre has begun*



c) *Manoeuvre - in progress*



d) *Platoon stabilized*

Figure 6.26: Graphical illustration of a lane change manoeuvre - Simulation results with Unity

CHAPTER 7

Conclusion

7.1 Summary

In this thesis, we facilitate the verification of hybrid systems against rich formal requirements by employing pattern templates. We define selected patterns in a formalism which is suitable for hybrid automata and applicable over both bounded and unbounded time. For these patterns, we give monitor automata with correctness proofs. By composing the monitors with the system model under study, the safety verification problem is transformed into the reachability problem of an error state. Results obtained from an industrial braking use case indicate that monitor automata can facilitate the applicability of hybrid system verification tools to industrial settings while inducing almost negligible computational overhead. These monitors are applicable to the development process of industries that utilize text-based safety requirements and they yield risk reduction when translating requirements into formal specifications.

Model transformation plays a vital role in bridging the gap between industrially relevant models and verification tools. This work aims to assist the application of hybrid system reachability tools to models designed with MATLAB/Simulink. We propose a methodology to construct verification models out of Simulink systems. We employ a translator to handle the mechanical aspects of the translation to hybrid automata and a simulation-guided technique to get bounds on the signal ranges. To address the semantic difference between Simulink/Stateflow models that are deterministic and urgent, and hybrid automata models that are nondeterministic and non-

urgent, we propose a heuristic method that reconstructs the resulting hybrid automaton model in order to comply with *may* semantics. For the Simulink blocks that are not exactly translated, e.g. nonlinearities, we apply a new hybridization method, which we call compositional syntactic hybridization.

Unlike standard state-space hybridization methods, we do not operate over the fully composed (flattened) model but perform the PWA approximations component-wise. In this way, we can obtain a significant reduction in terms of model size. The constructed verification model consists of a network of hybrid automata and is described in the SX format. It can then be fed into the SpaceEx platform or other verification tools through the HYST translator. Using SpaceEx for the reachability computations, we can take advantage of the on-the-fly composition and instantiate only the reachable parts of the approximation. Note that our compositional hybridization can be applied not only to the dynamics but also to algebraic and initial constraints.

On an industrial benchmark, the wind turbine, our approach leads to a very compact model that is orders of magnitude smaller than a standard hybridization model. In particular, the wind turbine benchmark from the ARCH workshop poses a challenging and relevant industrial model [267]. It is designed with MATLAB/Simulink and it is a large-scale model with many nonlinearities. Ten nonlinear blocks were approximated with the proposed hybridization scheme and the resulting model only has 72 locations in all components combined. The standard hybridization would lead to an $\mathcal{O}(1/\ell^n)$ number of locations. Taking into account that the dimension of the state-space is (at least) seven and considering 5 locations per state variable, the standard method would yield $5^7 = 78125$ locations. That indicates a considerable difference between the hybridization methods in terms of the model size.

7.2 Future work

Pattern Templates and Monitor Automata. In this thesis, we considered 8 different pattern templates to specify system properties. Each pattern template has a triggered and untriggered (global) version. The select pattern templates correspond to some of the properties that most common occur in industrial practice. In the future, we would like to construct monitors to encode other patterns, such as bounded existence or constrained precedence [34, 196]. Another direction would be to employ these patterns to describe other control objectives like ringing and spikes [43, 183]. In addition, it would be noteworthy to consider adding quantitative metrics in the monitors in the spirit of [73] and evaluate their efficiency. Finally, optimized versions of our monitors could be provided that aim to mitigate the computational costs of the standard parallel composition operation.

Compositional Hybridization. The next step is to improve the reachability tools so that they can efficiently employ these compositional models. The primary objective is to instantiate as few locations and transitions as possible during the analysis. There are three issues to address. The first is the on-the-fly composition and instantiation of the models, which can reduce the number of instantiated locations of the product automaton. The second is a compositional pre-processing of the components, where we utilize the pre-image of the target invariant when checking which transitions are enabled. The third direction would be to perform compositional mapping of the initial states. In the case of SpaceEx platform, the identification of the initial conditions is done through enumeration. However, enumeration of the locations of the product automaton is an operation that does not scale. Applying compositional reasoning would allow us to identify the initial locations and instantiate as few locations as possible.

Syntactic Hybridization. Our future work is targeted towards improvements in all three steps of the syntactic approach. As for the decomposition, we plan to broaden the syntactic method for a larger class of nonlinear functions and we would like to find out what is the best method to break down nonlinearities that includes a large amount of variables. As far as the PWA approximation is concerned, we intend to perform refinement of the discretization domains in order to reduce their total number and keep the most relevant regions. Finally, we would like to explore the importance of adding non-uniform, multidimensional grid methods as well as specialized approaches to handle non-smooth functions.

Lance Chage Maneuver. Autonomous vehicles have stimulated both industrial and academic interest, and have induced major advances in several research areas, such as perception, sensing, control theory, verification, and testing. Verification efforts have been directed towards the validation of vehicle modules and subsystems, such as cruise control and lane-keeping. The current model can serve as a basis for lane change maneuvers of autonomous vehicles tailored for automated verification tools. It is not very complicated but still realistic and expressive. Disturbances and measurement noise increase the computational efforts during analysis with standard simulation tools. This benchmark model can be extended in several directions. Different controllers and vehicle models could be straightforwardly used and analysed.

Bibliography

[1] Alessandro Abate, Alessandro D'Innocenzo, Giordano Pola, Maria Domenica Di Benedetto, and Shankar Sastry. The concept of deadlock and livelock in hybrid control systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 628–632. Springer, 2007. (*Cited on page 17.*)

[2] Parosh Aziz Abdulla, Johann Deneux, Gunnar Stalmarck, Herman Agren, and Ove Akerlund. Designing safe, reliable systems using Scade. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 115–129. Springer, 2004. (*Cited on pages 4 and 32.*)

[3] Aditya Agrawal, Gyula Simon, and Gabor Karsai. Semantic translation of Simulink/Stateflow models to hybrid automata using graph transformations. *ENTCS Journal*, 109:43–56, 2004. (*Cited on pages 9 and 112.*)

[4] Ayman Aljarbouh and Benoît Caillaud. On the regularization of chattering executions in real time simulation of hybrid systems. In *Baltic Young Scientists Conference*, page 49, 2015. (*Cited on page 17.*)

[5] Ayman Aljarbouh and Benoit Caillaud. Robust simulation for hybrid systems: Chattering path avoidance. In *Proceedings of the 56th Conference on Simulation and Modelling (SIMS 56), October, 7-9, 2015, Linköping University, Sweden*, number 119, pages 175–185. Linköping University Electronic Press, 2015. (*Cited on page 17.*)

[6] Ayman Aljarbouh, Yingfu Zeng, Adam Duracz, Benoît Caillaud, and Walid Taha. Chattering-free simulation for hybrid dynamical systems semantics and prototype implementation. In *Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed*

Computing and Applications for Business Engineering (DCABES), 2016 IEEE Intl Conference on, pages 412–422. IEEE, 2016. (*Cited on page 17.*)

[7] Hassane Alla and René David. Continuous and hybrid petri nets. *Journal of Circuits, Systems, and Computers*, 8(01):159–188, 1998. (*Cited on page 3.*)

[8] José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simao Melo de Sousa. An overview of formal methods tools and techniques. In *Rigorous Software Development*, pages 15–44. Springer, 2011. (*Cited on pages 2 and 13.*)

[9] Matthias Althoff. Reachability analysis and its application to the safety assessment of autonomous cars. *Technische Universität München*, 2010. (*Cited on page 14.*)

[10] Matthias Althoff. An introduction to CORA 2015. In *ARCH Work.*, 2015. (*Cited on pages 20 and 21.*)

[11] Matthias Althoff, Stanley Bak, Dario Cattaruzza, Xin Chen, Goran Frehse, Rajarshi Ray, and Stefan Schupp. Arch-comp17 category report: Continuous and hybrid systems with linear continuous dynamics. In *ARCH Workshop*, volume 48 of *EPiC Series in Computing*, pages 143–159, 2017. (*Cited on page 21.*)

[12] Matthias Althoff and John M Dolan. Set-based computation of vehicle behaviors for the online verification of autonomous vehicles. In *ITSC Conf.*, 2011. (*Cited on page 130.*)

[13] Matthias Althoff and Goran Frehse. Benchmarks of the ARCH workshop. `http://cps-vo.org/group/ARCH/benchmarks`, 2014. (*Cited on page 98.*)

[14] Matthias Althoff and Goran Frehse. Combining zonotopes and support functions for efficient reachability analysis of linear systems. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 7439–7446. IEEE, 2016. (*Cited on page 21.*)

[15] Matthias Althoff and Goran Frehse. ARCH Workshop. EPiC Series in Computing, 2018. (*Cited on page 20.*)

[16] Matthias Althoff, Markus Koschi, and Stefanie Manzinger. CommonRoad: Composable benchmarks for motion planning on roads. In *IV Symposium*, 2017. (*Cited on pages 115 and 132.*)

[17] Matthias Althoff, Olaf Stursberg, and Martin Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *CDC Conference*. IEEE, 2008. (*Cited on pages 79 and 124.*)

[18] Rajeev Alur. Formal verification of hybrid systems. In *EMSOFT*, 2011. (*Cited on pages 3, 4, 13, and 14.*)

[19] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 1995. (*Cited on pages 3, 14, and 15.*)

[20] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*, pages 209–229. Springer, 1993. (*Cited on pages 3, 14, and 27.*)

[21] Rajeev Alur and Thomas A Henzinger. Modularity for timed and hybrid systems. In *International Conference on Concurrency Theory*, pages 74–88. Springer, 1997. (*Cited on page 32.*)

[22] Rajeev Alur, Thomas A Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *ITSE*, 1996. (*Cited on page 18.*)

[23] Rajeev Alur, Thomas A Henzinger, Gerardo Lafferriere, and George J Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000. (*Cited on pages 4 and 14.*)

[24] Rajeev Alur, Aditya Kanade, S Ramesh, and KC Shashidhar. Symbolic analysis for improving simulation coverage of Simulink/Stateflow models. In *EMSOFT Conference*. ACM, 2008. (*Cited on pages 8, 75, 95, and 112.*)

[25] Étienne André, Thomas Chatain, Laurent Fribourg, and Emmanuelle Encrenaz. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(05):819–836, 2009. (*Cited on page 14.*)

[26] Étienne André and Laurent Fribourg. Behavioral cartography of timed automata. In *International Workshop on Reachability Problems*, pages 76–90. Springer, 2010. (*Cited on page 14.*)

[27] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *TACAS Conference*. Springer, 2011. (*Cited on pages 8 and 112.*)

[28] Argosim. Stimulus. http://argosim.com, 2015. (*Cited on pages 7 and 73.*)

[29] George W Arnold. Challenges and opportunities in smart grid: A position article. *Proceedings of the IEEE*, 99(6):922–927, 2011. (*Cited on page 26.*)

[30] Eugene Asarin and Thao Dang. Abstraction by projection and application to multi-affine systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 32–47. Springer, 2004. (*Cited on pages 5 and 20.*)

[31] Eugene Asarin, Thao Dang, Goran Frehse, Antoine Girard, Colas Le Guernic, and Oded Maler. Recent progress in continuous and hybrid reachability analysis. In *Computer Aided Control System Design*, 2006. (*Cited on pages 2, 5, and 20.*)

[32] Eugene Asarin, Thao Dang, and Antoine Girard. Reachability analysis of nonlinear systems using conservative approximation. In *HSCC Workshop*. Springer, 2003. (*Cited on page 81.*)

[33] Eugene Asarin, Thao Dang, and Antoine Girard. Hybridization Methods for the Analysis of Nonlinear Systems. *Acta Informatica*, 2007. (*Cited on pages 5, 20, 76, 77, and 79.*)

[34] Marco Autili, Lars Grunske, Markus Lumpe, Patrizio Pelliccione, and Antony Tang. Property Specification Patterns. http://ps-patterns.wikidot.com/. (*Cited on pages 7, 44, 51, and 145.*)

[35] Radhakisan Baheti and Helen Gill. Cyber-physical systems. *The Impact of Control Technology*, 2011. (*Cited on pages 1, 4, 26, and 28.*)

[36] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008. (*Cited on page 17.*)

[37] Stanley Bak. HyCreate: A tool for overapproximating reachability of hybrid automata. 2013. (*Cited on pages 21 and 39.*)

[38] Stanley Bak, Omar Ali Beg, Sergiy Bogomolov, Taylor T Johnson, Luan Viet Nguyen, and Christian Schilling. Hybrid automata: from verification to implementation. *International Journal on Software Tools for Technology Transfer*, 2017. (*Cited on pages 95, 96, and 113.*)

[39] Stanley Bak, Sergiy Bogomolov, Thomas A. Henzinger, Taylor T. Johnson, and Pradyot Prakash. Scalable static hybridization methods for analysis of nonlinear systems. In *HSCC'16*, pages 155–164, 2016. (*Cited on page 82.*)

[40] Stanley Bak, Sergiy Bogomolov, and Taylor T. Johnson. HYST: a source transformation and translation tool for hybrid automaton models. In *[HSCC'15]*, pages 128–133, 2015. (*Cited on pages 39 and 76.*)

[41] Felice Balarin, Yosinori Watanabe, Harry Hsieh, Luciano Lavagno, Claudio Passerone, and Alberto Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *Computer*, 36(4):45–52, 2003. (*Cited on page 32.*)

[42] Daniel Balasubramanian, Corina S Păsăreanu, Michael W Whalen, Gábor Karsai, and Michael Lowry. Polyglot: modeling and analysis for multiple statechart formalisms. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, pages 45–55. ACM, 2011. (*Cited on pages 8 and 112.*)

[43] Alessio Balsini, Marco Di Natale, Marco Celia, and Vassilios Tsachouridis. Generation of Simulink monitors for control applications from formal requirements. In *SIES Symposium*, 2017. (*Cited on pages 48 and 145.*)

[44] Pierfrancesco Bellini, Paolo Nesi, and Davide Rogai. Expressing and organizing real-time specification patterns via temporal logics. *Journal of Systems and Software*, 82(2):183–196, 2009. (*Cited on pages 7 and 44.*)

[45] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999. (*Cited on page 14.*)

[46] Albert Benveniste, Timothy Bourke, Benoit Caillaud, Jean-Louis Colaço, Cédric Pasteur, and Marc Pouzet. Building a hybrid systems modeler on synchronous languages principles. *Proceedings of the IEEE*, 106(9):1568–1592, 2018. (*Cited on page 14.*)

[47] Albert Benveniste, Timothy Bourke, Benoit Caillaud, Bruno Pagano, and Marc Pouzet. A type-based analysis of causality loops in hybrid systems modelers. *Nonlinear Analysis: Hybrid Systems*, 26:168–189, 2017. (*Cited on page 38.*)

[48] Albert Benveniste, Timothy Bourke, Benoit Caillaud, and Marc Pouzet. Divide and recycle: types and compilation for a hybrid synchronous language. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES'11)*, 2011. (*Cited on page 36.*)

[49] Albert Benveniste, Timothy Bourke, Benoit Caillaud, and Marc Pouzet. Non-standard semantics of hybrid systems modelers. *Journal of Computer and System Sciences*, 78(3):877–910, 2012. (*Cited on page 14.*)

[50] Albert Benveniste, Benoit Caillaud, and Marc Pouzet. The fundamentals of hybrid systems modelers. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 4180–4185. IEEE, 2010. (*Cited on page 14.*)

[51] Béatrice Bérard and Laurent Fribourg. Reachability analysis of (timed) petri nets using real arithmetic. In *International Conference on Concurrency Theory*, pages 178–193. Springer, 1999. (*Cited on page 14.*)

[52] Jan A Bergstra, Cornelis A Middelburg, et al. Process algebra for hybrid systems. *Theoretical Computer Science*, 335(2-3):215–280, 2005. (*Cited on pages 3 and 14.*)

[53] Arendse Bernth. EasyEnglish: a tool for improving document quality. In *Proceedings of the fifth conference on Applied natural language processing*, pages 159–165. Association for Computational Linguistics, 1997. (*Cited on page 6.*)

[54] Gérard Berry. SCADE: Synchronous design and validation of embedded control software. In *Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems*, pages 19–33. Springer, 2007. (*Cited on page 32.*)

[55] Martin Berz and Georg Hoffstätter. Computation and application of Taylor polynomials with interval remainder bounds. *Reliable Computing*, 4(1):83–97, 1998. (*Cited on page 79.*)

[56] Dirk Beyer, Sumit Gulwani, and David Schmidt. *Combining Model Checking and Data-Flow Analysis*. Springer, 2018. (*Cited on page 7.*)

[57] A. Bita. SpeAR — specification and analysis for requirements tool. https://github.com/AFifarek/SpeAR, 2016. (*Cited on pages 7 and 73.*)

[58] Rick Bitter, Taqi Mohiuddin, and Matt Nawrocki. *LabVIEW: Advanced programming techniques*. Crc Press, 2006. (*Cited on pages 32 and 36.*)

[59] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Frédéric Viry, Andreas Podelski, and Christian Schilling. Reach set approximation through decomposition with low-dimensional sets and high-dimensional matrices. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pages 41–50. ACM, 2018. (*Cited on page 21.*)

[60] Sergiy Bogomolov, Daniele Magazzeni, Stefano Minopoli, and Martin Wehrle. Pddl+ planning with hybrid automata: Foundations of translating must behavior. 2015. (*Cited on page 113.*)

[61] Olivier Bouissou and Alexandre Chapoutot. An operational semantics for simulink's simulation engine. In *ACM SIGPLAN Notices*, volume 47, pages 129–138. ACM, 2012. (*Cited on pages 9 and 113.*)

[62] Olivier Bouissou, Alexandre Chapoutot, Adel Djaballah, and Michel Kieffer. Computation of parametric barrier functions for dynamical systems using interval analysis. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 753–758. IEEE, 2014. (*Cited on page 14.*)

[63] Olivier Bouissou, Samuel Mimram, and Alexandre Chapoutot. HySon: Set-based simulation of hybrid systems. In *RSP Symposium*. IEEE, 2012. (*Cited on page 112.*)

[64] Olivier Bouissou, Samuel Mimram, Baptiste Strazzulla, and Alexandre Chapoutot. Set-based simulation for design and verification of Simulink models. In *Embedded Real Time Software and Systems (ERTS2)*, 2014. (*Cited on pages 1 and 112.*)

[65] Jean-Louis Boulanger, François-Xavier Fornari, Jean-Louis Camus, and Bernard Dion. SCADE: Language and applications. 2015. (*Cited on page 34.*)

[66] Hamza Bourbouh, Pierre-Loic Garoche, Christophe Garion, Arie Gurfinkel, Kahsai Temesghen, and Thirioux Xavier. Automated analysis of stateflow models. In *21st International Conference on Logic for*

Programming, Artificial Intelligence and Reasoning, 2017. (*Cited on pages 9 and 113.*)

[67] Timothy Bourke, Jean-Louis Colaço, Bruno Pagano, Cédric Pasteur, and Marc Pouzet. A synchronous-based code generator for explicit hybrid systems languages. In *CC Conference*, pages 69–88. Springer, 2015. (*Cited on page 37.*)

[68] Barry Bozeman. Technology transfer and public policy: a review of research and theory. *Research policy*, 29(4-5):627–655, 2000. (*Cited on page 29.*)

[69] Davide Bresolin. HyLTL: a temporal logic for model checking hybrid systems. In *Hybrid Autonomous Systems Workshop*, 2013. (*Cited on page 73.*)

[70] Davide Bresolin. Improving HyLTL model checking of hybrid systems. In *GandALF Symposium*, 2013. (*Cited on page 73.*)

[71] Dag Brück, Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. Dymola for multi-engineering modeling and simulation. In *Proceedings of modelica*, volume 2002. Citeseer, 2002. (*Cited on page 36.*)

[72] A. A. Bruto da Costa and Pallab Dasgupta. Formal interpretation of assertion-based features on ams designs. *IEEE Design & Test*, 2015. (*Cited on page 42.*)

[73] Antonio Anastasio Bruto da Costa, Goran Frehse, and Pallab Dasgputa. Formal feature interpretation of hybrid systems. *TCAD Transactions*, 2018. (*Cited on page 145.*)

[74] BTC. Embedded specifier. `http://www.btc-es.de`, 2015. (*Cited on pages 7 and 73.*)

[75] Joseph T Buck, Soonhoi Ha, Edward A Lee, and David G Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. 1994. (*Cited on page 4.*)

[76] Axel Busboom, Simone Schuler, and Alexander Walsch. FORMALSPEC: Semi-automatic formalization of system requirements for formal verification. In *ARCH Workshop*, 2016. (*Cited on pages 38, 42, 56, 71, and 118.*)

[77] Vadim A Butakov and Petros Ioannou. Personalized driver/vehicle lane change models for adas. *IEEE Transactions on Vehicular Technology*, 64(10):4422–4431, 2015. (*Cited on page 129.*)

[78] Quick Facts by National Highway Traffic Safety Administration. Available at http://www-nrd.nhtsa.dot.gov, 2011. (*Cited on page 26.*)

[79] Alexandre Chapoutot, Laurent-Stéphane Didier, and Fanny Villers. Range estimation of floating-point variables in Simulink models. In *Design and Architectures for Signal and Image Processing (DASIP), 2012 Conference on*, pages 1–8. IEEE, 2012. (*Cited on page 89.*)

[80] Alexandre Chapoutot and Matthieu Martel. Abstract simulation: a static analysis of simulink models. In *Embedded Software and Systems, 2009. ICESS'09. International Conference on*, pages 83–92. IEEE, 2009. (*Cited on page 89.*)

[81] Hongxu Chen, Sayan Mitra, and Guangyu Tian. Motor-transmission drive system: a benchmark example for safety verification. In *ARCH@ CPSWeek*, 2014. (*Cited on page 98.*)

[82] Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *RTSS Symposium*, pages 183–192. IEEE, 2012. (*Cited on pages 5 and 20.*)

[83] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *CAV Conference*, 2013. (*Cited on pages 21 and 39.*)

[84] Xin Chen and Sriram Sankaranarayanan. Decomposed reachability analysis for nonlinear systems. In *Real-Time Systems Symposium (RTSS)*, 2016. (*Cited on page 80.*)

[85] Xin Chen, Stefan Schupp, Ibtissem Ben Makhlouf, Erika Ábrahám, Goran Frehse, and Stefan Kowalewski. A benchmark suite for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*, pages 408–414. Springer, 2015. (*Cited on page 20.*)

[86] Remy Chevallier, Emmanuelle Encrenaz-Tiphene, Laurent Fribourg, and Weiwen Xu. Timed verification of the generic architecture of a memory circuit using parametric timed automata. *FORM METHOD SYST DES*, 2009. (*Cited on page 14.*)

[87] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Verifying LTL properties of hybrid systems with K-Liveness. In *CAV Conf.*, 2014. (*Cited on page 73.*)

[88] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. Requirements validation for hybrid systems. In *CAV Conference*, 2009. (*Cited on page 73.*)

[89] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. HRELTL: A temporal logic for hybrid systems. *Inf. Comput.*, 2015. (*Cited on page 73.*)

[90] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999. (*Cited on pages 13 and 23.*)

[91] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. (*Cited on pages 2 and 13.*)

[92] Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet. Scade 6: a formal language for embedded critical software development. In *TASE Symposium*, pages 1–11. IEEE, 2017. (*Cited on page 33.*)

[93] Pieter Collins, Davide Bresolin, Luca Geretti, and Tiziano Villa. Computing the evolution of hybrid systems using rigorous function calculus. *Proc. of ADHS*, 12:284–290, 2012. (*Cited on page 20.*)

[94] Scott Cotton, Goran Frehse, and Olivier Lebeltel. The SpaceEx modeling language, 2010. (*Cited on pages 22 and 80.*)

[95] Javier Sanchez Cubillo, Simone Schuler, Daniel Heß, Maria Prandini, and Mark Burgin. UnCoVerCPS Deliverable D5.1: Report on Application Models. `https://cps-vo.org/node/24201`, 2015. (*Cited on page 134.*)

[96] Dennis R Dams and Kedar S Namjoshi. Orion: High-precision methods for static error analysis of C and C++ programs. In *International Symposium on Formal Methods for Components and Objects*, pages 138–160. Springer, 2005. (*Cited on page 7.*)

[97] Thao Dang and Tarik Nahhal. Coverage-guided test generation for continuous and hybrid systems. *FMSD Journal*, 34(2):183–213, 2009. (*Cited on page 22.*)

[98] Thao Dang and Romain Testylier. Reachability Analysis for Polynomial Dynamical Systems Using the Bernstein Expansion. *Reliable Computing*, 17(2):128–152, 2012. (*Cited on pages 5 and 20.*)

[99] Jennifer Mary Davoren and Anil Nerode. Logics for hybrid systems. *IEEE Proceedings*, 2000. (*Cited on page 73.*)

[100] Kees Van Deemter, Mariët Theune, and Emiel Krahmer. Real versus template-based natural language generation: A false opposition? *Computational Linguistics*, 31(1):15–24, 2005. (*Cited on pages 6 and 41.*)

[101] Jyotirmoy V Deshmukh, Hisahiro Ito, Xiaoqing Jin, James Kapinski, Ken Butts, Jürgen Gerhard, Behzad Samadi, Kevin Walker, and Yuzhen Xie. Piecewise-affine approximations for a powertrain control verification benchmark. In *Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015. (*Cited on page 82.*)

[102] David L Dill. What's between simulation and formal verification? In *Design Automation Conference, 1998. Proceedings*, pages 328–329. IEEE, 1998. (*Cited on page 22.*)

[103] Julien Alexandre dit Sandretto, Alexandre Chapoutot, and Olivier Mullier. Formal verification of robotic behaviors in presence of bounded uncertainties. In *Robotic Computing (IRC), IEEE International Conference on*, pages 81–88. IEEE, 2017. (*Cited on page 14.*)

[104] Adel Djaballah, Alexandre Chapoutot, Michel Kieffer, and Olivier Bouissou. Construction of parametric barrier functions for dynamical systems using interval analysis. *Automatica*, 78:287–296, 2017. (*Cited on page 14.*)

[105] Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos. Metric interval temporal logic specification elicitation and debugging. In *MEMOCODE*, 2015. (*Cited on page 49.*)

[106] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010. (*Cited on page 115.*)

[107] Alexandre Donzé. Breach: a toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *CAV Conference*, pages 167–170. Springer, 2010. (*Cited on pages 8, 22, 36, 89, and 112.*)

[108] Alexandre Donzé, Thomas Ferrere, and Oded Maler. Efficient robust monitoring for STL. In *CAV Conference*, pages 264–279. Springer, 2013. (*Cited on page 23.*)

[109] Alexandre Donzé and Goran Frehse. Modular, hierarchical models of control systems in SpaceEx. In *Control Conference (ECC), 2013 European*. IEEE, 2013. (*Cited on pages 18 and 81.*)

[110] Francois-Xavier Dormoy. Scade 6: a model based solution for safety critical software development. In *Proceedings of the 4th European Congress on Embedded Real Time Software (ERTS?08)*, pages 1–9, 2008. (*Cited on page 34.*)

[111] Laurent Doyen, Goran Frehse, George Pappas, and A. Platzer. *Verification of hybrid systems*. Handbook of Model Checking, 2017. (*Cited on pages 2, 4, 5, 14, 19, 20, and 76.*)

[112] Marie Duflot, Laurent Fribourg, and Ulf Nilsson. Unavoidable configurations of parameterized rings of processes. In *International Conference on Concurrency Theory*, pages 472–486. Springer, 2001. (*Cited on page 13.*)

[113] Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. C2E2: A verification tool for stateflow models. In *TACAS Conference*, 2015. (*Cited on page 20.*)

[114] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE Conference*, 1999. (*Cited on pages 6, 44, and 73.*)

[115] Johan Eker, Jörn W Janneck, Edward A Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming heterogeneity-the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003. (*Cited on page 32.*)

[116] Hilding Elmqvist, Sven Erik Mattsson, and Martin Otter. Modelica – a language for physical system modeling, visualization and interaction. In *Computer Aided Control System Design, 1999. Proceedings of the 1999 IEEE International Symposium on*, pages 630–639. IEEE, 1999. (*Cited on pages 4, 17, and 32.*)

[117] Unity Game Engine. Official website. *http://unity3d. com*, 2008. (*Cited on page 139.*)

[118] MW Esser and Peter Struss. Obtaining models for test generation from natural-language-like functional specifications. In *International workshop on principles of diagnosis*, pages 75–82, 2007. (*Cited on page 6.*)

[119] Daniel J Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 2015. (*Cited on page 115.*)

[120] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat. Predictive active steering control for autonomous vehicle systems. *IEEE CST Transactions*, 2007. (*Cited on page 115.*)

[121] Chuchu Fan, Bolun Qi, Sayan Mitra, Mahesh Viswanathan, and Parasara Sridhar Duggirala. Automatic reachability analysis for nonlinear hybrid models with C2E2. In *CAV Conference*, 2016. (*Cited on pages 8 and 112.*)

[122] Predrag Filipovikj, Nesredin Mahmud, Raluca Marinescu, Cristina Seceleanu, Oscar Ljungkrantz, and Henrik Lönn. Simulink to UPPAAL statistical model checker: Analyzing automotive industrial systems. In *FM Symposium*. Springer, 2016. (*Cited on page 8.*)

[123] Emilio Frazzoli. *Robust hybrid control for autonomous vehicle motion planning*. PhD thesis, Massachusetts Institute of Technology, 2001. (*Cited on page 129.*)

[124] Goran Frehse. *Compositional Verification of Hybrid Systems Using Simulation Relations*. PhD thesis, Radboud University Nijmegen, 2005. (*Cited on page 82.*)

[125] Goran Frehse. An introduction to SpaceEx v0.8. `http://spaceex.imag.fr/sites/default/files/introduction_to_spaceex_0.pdf`, 2010. (*Cited on pages 22, 71, and 72.*)

[126] Goran Frehse. An introduction to hybrid automata, numerical simulation and reachability analysis. In *Formal Modeling and Verification of Cyber-Physical Systems*, pages 50–81. Springer, 2015. (*Cited on page 15.*)

[127] Goran Frehse. Reachability of hybrid systems in space-time. In *EMSOFT*, pages 41–50. IEEE, 2015. (*Cited on pages 2 and 21.*)

[128] Goran Frehse. *Scalable Verification of Hybrid Systems*. PhD thesis, Univ. Grenoble Alpes, 2016. (*Cited on pages 4 and 5.*)

[129] Goran Frehse, Sergiy Bogomolov, Marius Greitschus, Thomas Strump, and Andreas Podelski. Eliminating spurious transitions in reachability with support functions. In *HSCC Conference*. ACM, 2015. (*Cited on page 21.*)

[130] Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *RTSS Symposium*, 2014. (*Cited on pages 14, 49, and 71.*)

[131] Goran Frehse, Rajat Kateja, and Colas Le Guernic. Flowpipe approximation and clustering in space-time. In *HSCC Conference*. ACM, 2013. (*Cited on page 21.*)

[132] Goran Frehse, Nikolaos Kekatos, and Dejan Nickovic. Formally correct monitors for hybrid automata. *Verimag Research Report*, 2017. (*Cited on page 41.*)

[133] Goran Frehse, Nikolaos Kekatos, Dejan Nickovic, Jens Oehlerking, Simone Schuler, Alexander Walsch, and Matthias Woerhle. A Toolchain for Verifying Safety Properties of Hybrid Automata via Pattern Templates. In *ACC Conference*, 2018. (*Cited on pages 41 and 118.*)

[134] Goran Frehse, Bruce H Krogh, and Rob A Rutenbar. Verifying analog oscillator circuits using forward/backward abstraction refinement. In *DATE Conference*. European Design and Automation Association, 2006. (*Cited on page 14.*)

[135] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *CAV Conference*. Springer, 2011. (*Cited on pages 20, 21, 72, 76, 90, 118, and 120.*)

[136] Laurent Fribourg. Petri nets, flat languages and linear arithmetic. Citeseer. (*Cited on pages 3 and 14.*)

[137] Laurent Fribourg. A superposition oriented theorem prover. *Theoretical Computer Science*, 35:129–164, 1985. (*Cited on page 13.*)

[138] Laurent Fribourg and Ulrich Kühne. Parametric verification and test coverage for hybrid automata using the inverse method. In *International*

Workshop on Reachability Problems, pages 191–204. Springer, 2011. (*Cited on page 14.*)

[139] Laurent Fribourg and Hans Olsén. Reachability sets of parameterized rings as regular languages. *Electronic Notes in Theoretical Computer Science*, 9:40, 1997. (*Cited on page 13.*)

[140] Laurent Fribourg and Julian Richardson. Symbolic verification with gap-order constraints. In *International Workshop on Logic Programming Synthesis and Transformation*, pages 20–37. Springer, 1996. (*Cited on page 21.*)

[141] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014. (*Cited on page 32.*)

[142] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns–elements of reusable object-oriented software. 1995. (*Cited on pages 42 and 43.*)

[143] Yiqi Gao, Andrew Gray, H. Eric Tseng, and Francesco Borrelli. A tube-based robust nonlinear predictive control approach to semiautonomous ground vehicles. *Vehicle System Dynamics*, 2014. (*Cited on page 130.*)

[144] Biniam Gebremichael and Frits Vaandrager. Specifying urgency in timed i/o automata. In *Software Engineering and Formal Methods, 2005. SEFM 2005. Third IEEE International Conference on*, pages 64–73. IEEE, 2005. (*Cited on page 104.*)

[145] Markus Gerdin. *Identification and estimation for models described by Differential-Algebraic Equations*. PhD thesis, Institutionen för systemteknik, 2006. (*Cited on page 81.*)

[146] Helen Gill. From vision to reality: Cyber-physical systems. 2008. (*Cited on page 24.*)

[147] KAHN Gilles. The semantics of a simple language for parallel programming. 1974. (*Cited on page 34.*)

[148] Rafal Goebel, Ricardo G Sanfelice, and Andrew R Teel. Hybrid dynamical systems. *IEEE Control Systems*, 29(2):28–93, 2009. (*Cited on page 14.*)

[149] Deepthi Gopalakrishna. Simulating autonomous vehicles in Unity. https://github.com/deepkrishna/AV-simulation, 2018. (*Cited on page 139.*)

[150] Andrew Gray, Yiqi Gao, J Karl Hedrick, and Francesco Borrelli. Robust predictive control for semi-autonomous vehicles with an uncertain driver model. In *IV Symposium*, 2013. (*Cited on page 130.*)

[151] Robert Grosse. International technology transfer in services. *Journal of international business studies*, 27(4):781–800, 1996. (*Cited on page 29.*)

[152] Orna Grumberg and Helmut Veith. *25 years of model checking: history, achievements, perspectives*, volume 5000. Springer, 2008. (*Cited on pages 3 and 29.*)

[153] L. Grunske. Specification patterns for probabilistic quality properties. In *ICSE Conference*, 2008. (*Cited on pages 7, 44, and 73.*)

[154] Feng Guo, Brian M Wotring, and Jonathan F Antin. Evaluation of lane change collision avoidance systems using the national advanced driving simulator. Technical report, 2010. (*Cited on page 129.*)

[155] Willem Hagemann, Eike Möhlmann, and Astrid Rakow. Verifying a PI controller using SoapBox and Stabhyli: Experiences on establishing properties for a steering controller. In *ARCH Workshop*, 2014. (*Cited on page 21.*)

[156] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In *AMAST Conference*, 1993. (*Cited on pages 7, 42, 56, and 57.*)

[157] Nicholas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991. (*Cited on page 33.*)

[158] Grégoire Hamon. A denotational semantics for stateflow. In *Proceedings of the 5th ACM international conference on Embedded software*. ACM, 2005. (*Cited on pages 9 and 113.*)

[159] Grégoire Hamon and John Rushby. An operational semantics for stateflow. *STTT Journal*, 9(5):447–456, 2007. (*Cited on pages 9 and 113.*)

[160] Grégoire Hamon, John Rushby, et al. An operational semantics for stateflow. In *FASE*, volume 2984. Springer, 2004. (*Cited on pages 9 and 113.*)

[161] Marianne Hartung, Daniel Heß, Ray Lattarulo, Jens Oehlerking, Joshué Pérez, and Alexander Rausch. UnCoVerCPS Deliverable D5.2: Report on Conformance Testing of Application Models. https://cps-vo.org/node/39012, 2016. (*Cited on pages 131, 134, and 135.*)

[162] Kaveh Hassani and Won-Sook Lee. Multi-objective design of state feedback controllers using reinforced quantum-behaved particle swarm optimization. *Applied Soft Computing*, 41:66–76, 2016. (*Cited on page 135.*)

[163] Matthew Hause et al. The SysML modelling language. In *Fifteenth European Systems Engineering Conference*, volume 9, pages 1–12. Citeseer, 2006. (*Cited on page 32.*)

[164] Wilhemus PMH Heemels, Bart De Schutter, and Alberto Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, 2001. (*Cited on page 14.*)

[165] WPMH Heemels, Johannes M Schumacher, and S Weiland. Linear complementarity systems. *SIAM journal on applied mathematics*, 60(4):1234–1269, 2000. (*Cited on page 14.*)

[166] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of computer and system sciences*, 57(1):94–124, 1998. (*Cited on pages 15 and 19.*)

[167] Daniel Heß, Matthias Althoff, and Thomas Sattel. Formal verification of maneuver automata for parameterized motion primitives. In *IROS Conf.*, 2014. (*Cited on pages 130 and 134.*)

[168] Daniel Heß, Christian Löper, and Tobias Hesse. Safe cooperation of automated vehicles, 2017. (*Cited on page 49.*)

[169] Daniel Heß, Bastian Schürmann, Marcelo Forets, and Goran Frehse. UnCoVerCPS Deliverable D3.2: Report on Precomputation of Reachable Sets and Advances in Reachability Analysis. https://cps-vo.org/node/39010, 2017. (*Cited on page 134.*)

[170] ML Ho, PT Chan, and AB Rad. Lane change algorithm for autonomous vehicles via virtual curvature method. *Journal of Advanced Transportation*, 43(1):47–70, 2009. (*Cited on page 129.*)

[171] Pei-Hsin Ho. Automatic analysis of hybrid systems. Technical report, Cornell University, 1995. (*Cited on pages 106 and 113.*)

[172] B Hoxha, N Mavridis, and G Fainekos. VISPEC: a graphical tool for easy elicitation of MTL requirements. In *IROS*, 2015. (*Cited on page 49.*)

[173] Franz Huber, Bernhard Schätz, Alexander Schmidt, and Katharina Spies. AutoFocus– a tool for distributed systems specification. In *FTRTFT Symposium*, 1996. (*Cited on pages 7 and 73.*)

[174] Petros A Ioannou and Cheng-Chih Chien. Autonomous intelligent cruise control. *IEEE Transactions on Vehicular technology*, 42(4):657–672, 1993. (*Cited on page 116.*)

[175] Jonas Jansson and Fredrik Gustafsson. A framework and automotive application of collision avoidance decision making. 2008. (*Cited on page 115.*)

[176] Jean-Baptiste Jeannin and André Platzer. dTL2: Differential temporal dynamic logic with nested temporalities for hybrid systems. In *IJCAR*, 2014. (*Cited on page 73.*)

[177] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V Deshmukh, and Sanjit A Seshia. Mining requirements from closed-loop control models. *TCAD Journal*, 2015. (*Cited on page 31.*)

[178] Barnabas D Johnson. The cybernetics of society: The governance of self and civilization, 2012. (*Cited on page 25.*)

[179] Taylor T Johnson. ARCH-COMP17 Repeatability Evaluation Report. 2017. (*Cited on page 20.*)

[180] Michael Kaiser. *Untersuchung zur online abgesicherten Fahrer-Fahrzeug Interaktion in Kollisionsvermeidungs-Szenarien*. PhD thesis, Technische Universität Braunschweig, 2017. (*Cited on page 132.*)

[181] Christine Kamprath, Eric Adolphson, Teruko Mitamura, and Eric Nyberg. Controlled language for multilingual document production: Experience with caterpillar technical english. In *Proceedings of the Second International Workshop on Controlled Language Applications*, 1998. (*Cited on page 6.*)

[182] Aaron Kane. Runtime monitoring for safety-critical embedded systems. *Dissertation*, 2015. (*Cited on pages 7 and 73.*)

[183] James Kapinski, Xiaoqing Jin, Jyotirmoy Deshmukh, Alexandre Donze, Tomoya Yamaguchi, Hisahiro Ito, Tomoyuki Kaga, Shunsuke Kobuna, and Sanjit Seshia. ST-Lib: A library for specifying and classifying model behaviors. *SAE Technical Report*, 2016. (*Cited on pages 48 and 145.*)

[184] Jim Kapinski, Bruce H Krogh, Oded Maler, and Olaf Stursberg. On systematic simulation of open continuous systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 283–297. Springer, 2003. (*Cited on page 22.*)

[185] Gabor Karsai, Janos Sztipanovits, Akos Ledeczi, and Ted Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, 2003. (*Cited on pages 1, 31, and 32.*)

[186] Nikolaos Kekatos. A guide on interleaving MATLAB modeling and SpaceEx reachability computations. `https://github.com/nikos-kekatos/Matlab-SpaceEx`, 2018. (*Cited on page 39.*)

[187] Nikolaos Kekatos, Marcelo Forets, and Goran Frehse. Constructing Verification Models of Nonlinear Simulink Systems via Syntactic Hybridization. In *CDC Conference*, 2017. (*Cited on pages 71, 75, and 116.*)

[188] Nikolaos Kekatos, Marcelo Forets, and Goran Frehse. Modeling the wind turbine benchmark with PWA hybrid automata. In *ARCH Workshop*, 2017. (*Cited on pages 49, 71, 115, and 116.*)

[189] Nikolaos Kekatos, Daniel Heß, and Goran Frehse. Lane change maneuver for autonomous vehicles. In *ARCH Workshop*, volume 54 of *EPiC Series in Computing*, pages 229–241. EasyChair, 2018. (*Cited on page 115.*)

[190] Roozbeh Kianfar, Paolo Falcone, and Jonas Fredriksson. Reachability analysis of cooperative adaptive cruise controller. In *Conference on Intelligent Transportation Systems*. IEEE, 2012. (*Cited on page 116.*)

[191] Uwe Kiencke and Lars Nielsen. *Automotive control systems: for engine, driveline, and vehicle*. Springer Science & Business Media, 2005. (*Cited on page 115.*)

[192] Kyoung-Dae Kim and PR Kumar. An overview and some challenges in cyber-physical systems. *Journal of the Indian Institute of Science*, 93(3):341–352, 2013. (*Cited on pages 26 and 27.*)

[193] Edda Klipp, Ralf Herwig, Axel Kowald, Christoph Wierling, and Hans Lehrach. *Systems biology in practice: concepts, implementation and application*. John Wiley & Sons, 2008. (*Cited on page 80.*)

[194] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *IV Symposium*, 2015. (*Cited on pages 115 and 132.*)

[195] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dreach: δ-reachability analysis for hybrid systems. In *TACAS Conference*, pages 200–205. Springer, 2015. (*Cited on pages 20 and 39.*)

[196] S. Konrad and B. Cheng. Real-time specification patterns. *ICSE*, 2005. (*Cited on pages 7, 42, 44, 51, 73, and 145.*)

[197] Markus Koschi and Matthias Althoff. Spot: A tool for set-based prediction of traffic participants. In *IV Symposium*. IEEE, 2017. (*Cited on page 115.*)

[198] Tobias Kuhn. A survey and classification of controlled natural languages. *Computational Linguistics*, 2014. (*Cited on pages 5, 6, and 41.*)

[199] Robert P Kurshan. Verification technology transfer. In *25 Years of Model Checking*, pages 46–64. Springer, 2008. (*Cited on pages 3 and 30.*)

[200] Robert P Kurshan. Transfer of model checking to industrial practice. In *Handbook of Model Checking*, pages 763–793. Springer, 2018. (*Cited on page 30.*)

[201] Colas Le Guernic. *Reachability analysis of hybrid systems with linear continuous dynamics*. PhD thesis, University of Joseph-Fourier Grenoble, 2009. (*Cited on page 105.*)

[202] Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In *CAV Conference*. Springer, 2009. (*Cited on page 105.*)

[203] Colas Le Guernic and Antoine Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 2010. (*Cited on page 21.*)

[204] Edward A Lee. Cyber physical systems: Design challenges. In *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369. IEEE, 2008. (*Cited on page 28.*)

[205] Edward A Lee. CPS foundations. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 737–742. IEEE, 2010. (*Cited on page 27.*)

[206] Edward A Lee. The past, present and future of cyber-physical systems: A focus on models. *Sensors*, 15(3):4837–4869, 2015. (*Cited on pages 25 and 28.*)

[207] Edward Ashford Lee and Sanjit A Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. MIT Press, 2016. (*Cited on pages 3, 4, 14, and 25.*)

[208] Insup Lee, Oleg Sokolsky, Sanjian Chen, John Hatcliff, Eunkyoung Jee, BaekGyu Kim, Andrew King, Margaret Mullen-Fortino, Soojin Park, Alexander Roederer, et al. Challenges and research directions in medical cyber–physical systems. *Proceedings of the IEEE*, 100(1):75–90, 2012. (*Cited on page 26.*)

[209] Daniel Liberzon. *Switching in systems and control*. Springer Science & Business Media, 2003. (*Cited on page 14.*)

[210] Sarah M Loos, André Platzer, and Ligia Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In *International Symposium on Formal Methods*, 2011. (*Cited on page 115.*)

[211] Pericles Loucopoulos and Vassilios Karakostas. *System requirements engineering*. McGraw-Hill, Inc., 1995. (*Cited on page 41.*)

[212] Xiao-Yun Lu and J Karl Hedrick. Longitudinal control algorithm for automated vehicle merging. *International Journal of Control*, 2003. (*Cited on page 131.*)

[213] Jan Lunze and Françoise Lamnabhi-Lagarrigue. *Handbook of hybrid systems control: theory, tools, applications*. Cambridge University Press, 2009. (*Cited on page 14.*)

[214] John Lygeros, Karl Henrik Johansson, Slobodan N Simic, Jun Zhang, and Shankar S Sastry. Dynamical properties of hybrid automata. *IEEE Transactions on automatic control*, 48(1):2–17, 2003. (*Cited on pages 3 and 14.*)

[215] Silvia Magdici and Matthias Althoff. Fail-safe motion planning of autonomous vehicles. In *ITSC Conf.*, 2016. (*Cited on page 115.*)

[216] Silvia Magdici, Zhenzhang Ye, and Matthias Althoff. Determining the maximum time horizon for vehicles to safely follow a trajectory. In *ITSC Conf.*, 2017. (*Cited on page 115.*)

[217] Ibtissem Ben Makhlouf, Norman Hansen, and Stefan Kowalewski. HyReach: A reachability tool for linear hybrid systems based on support functions. In *ARCH Workshop*, 2016. (*Cited on page 21.*)

[218] Oded Maler. Algorithmic verification of continuous and hybrid systems. In *Infinity*, 2013. (*Cited on pages 2, 5, 20, 22, and 76.*)

[219] Oded Maler. Some thoughts on runtime verification. In *International Conference on Runtime Verification*, pages 3–14. Springer, 2016. (*Cited on page 13.*)

[220] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*, pages 447–484. Springer, 1991. (*Cited on page 27.*)

[221] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *FORMATS Conference*, 2004. (*Cited on pages 23, 24, and 73.*)

[222] Oded Maler and Dejan Nickovic. Monitoring properties of analog and mixed-signal circuits. *STTT Journal*, 2013. (*Cited on pages 23 and 73.*)

[223] Karthik Manamcheri, Sayan Mitra, Stanley Bak, and Marco Caccamo. A step towards verification and synthesis from Simulink/Stateflow models. In *HSCC Conference*. ACM, 2011. (*Cited on pages 8, 9, 99, and 112.*)

[224] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety*. Springer Science & Business Media, 2012. (*Cited on page 23.*)

[225] MapleSoft. MapleSim: Advanced System-Level Modeling. (*Cited on page 17.*)

[226] Mathworks. Modeling a dc motor in stateflow. https://fr.mathworks.com/help/stateflow/examples/modeling-a-dc-motor-in-stateflow.html. (*Cited on page 101.*)

[227] MATLAB 9.0 and Simulink 8.7. The MathWorks, Inc., Natick, Massachusetts, United States. (*Cited on pages 4, 17, 32, 36, 75, and 88.*)

[228] B Meenakshi, Abhishek Bhatnagar, and Sudeepa Roy. Tool for translating Simulink models into input language of a model checker. In *International Conference on Formal Engineering Methods*, pages 606–620. Springer, 2006. (*Cited on page 8.*)

[229] Stefano Minopoli and Goran Frehse. Non-convex invariants and urgency conditions on linear hybrid automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 176–190. Springer, 2014. (*Cited on pages 9, 104, and 113.*)

[230] Stefano Minopoli and Goran Frehse. From simulation models to hybrid automata using urgency and relaxation. In *HSCC Conference*, pages 287–296. ACM, 2016. (*Cited on pages 9, 17, 98, 104, and 113.*)

[231] Stefano Minopoli and Goran Frehse. SL2SX translator: From Simulink to SpaceEx models. In *HSCC Conference*, pages 93–98. ACM, 2016. (*Cited on pages 9, 35, 71, 90, 116, and 120.*)

[232] Ramon E Moore. *Methods and applications of interval analysis*, volume 2. Siam, 1979. (*Cited on page 14.*)

[233] Sergio Mover, Alessandro Cimatti, Ashish Tiwari, and Stefano Tonetta. Time-aware relational abstractions for hybrid systems. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–10. IEEE, 2013. (*Cited on page 81.*)

[234] Venkatesh Mysore, Carla Piazza, and Bud Mishra. Algorithmic algebraic model checking II: Decidability of semi-algebraic model checking and its applications to systems biology. In *ATVA*, 2005. (*Cited on page 73.*)

[235] Anil Nerode and Wolf Kohn. Models for hybrid systems: Automata, topologies, controllability, observability. In *Hybrid systems*, pages 317–356. Springer, 1993. (*Cited on page 14.*)

[236] Dejan Nickovic. Monitoring and measuring hybrid behaviors. A tutorial. In *RV Conference*, 2015. (*Cited on page 73.*)

[237] Gabriela Nicolescu and Pieter J Mosterman. *Model-based design for embedded systems*. CRC Press, 2009. (*Cited on pages 1, 31, 32, and 75.*)

[238] Julia Nilsson, Mattias Brännström, Erik Coelingh, and Jonas Fredriksson. Lane change maneuvers for automated vehicles. *ITS*, 2017. (*Cited on pages 129, 130, and 131.*)

[239] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46. ACM, 2000. (*Cited on page 41.*)

[240] Sharon O'Brien. Controlling Controlled English: An analysis of several Controlled Language Rule Sets. *Proceedings of EAMT-CLAW*, 2003. (*Cited on page 5.*)

[241] Martin Otter, Nguyen Thuy, Daniel Bouskela, Lena Buffoni, Hilding Elmqvist, Peter Fritzson, Alfredo Garro, Audrey Jardin, Hans Olsson, Maxime Payelleville, et al. Formal requirements modeling for simulation-based verification. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, number 118, pages 625–635. Linköping University Electronic Press, 2015. (*Cited on page 48.*)

[242] Hans Pacejka. *Tire and vehicle dynamics*. Elsevier, 2005. (*Cited on page 132.*)

[243] Miroslav Pajic, Zhihao Jiang, Insup Lee, Oleg Sokolsky, and Rahul Mangharam. From verification to implementation: A model translation tool and a pacemaker case study. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, pages 173–184. IEEE, 2012. (*Cited on page 113.*)

[244] Cédric Pasteur, Jean-Louis Colaço, and Goran Frehse. UnCoVerCPS Deliverable D4.2: Extension of the Scade language for continuous modeling. `https://cps-vo.org/node/39011`, 2017. (*Cited on pages 34 and 37.*)

[245] Aaron Pereira and Matthias Althoff. Safety control of robots under computed torque control using reachable sets. In *ICRA Conference*. IEEE, 2015. (*Cited on page 14.*)

[246] Julius Cuong Pham, Monica S Aswani, Michael Rosen, HeeWon Lee, Matthew Huddle, Kristina Weeks, and Peter J Pronovost. Reducing medical errors and adverse events. *Annual review of medicine*, 63:447–463, 2012. (*Cited on page 27.*)

[247] Alessandro Pinto, Luca P Carloni, Roberto Passerone, and Alberto Sangiovanni-Vincentelli. Interchange format for hybrid systems: Abstract semantics. In *International Workshop on Hybrid Systems: Computation and Control*, pages 491–506. Springer, 2006. (*Cited on page 112.*)

[248] Alessandro Pinto, Alberto Sangiovanni-Vincentelli, Luca Carloni, and Roberto Passerone. Interchange formats for hybrid systems: Review and proposal. *Hybrid Systems: Computation and Control*, pages 526–541, 2005. (*Cited on pages 8 and 112.*)

[249] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 216–232. Springer, 2007. (*Cited on pages 3 and 14.*)

[250] André Platzer. *Logical analysis of hybrid systems: proving theorems for complex dynamics*. Springer Science & Business Media, 2010. (*Cited on pages 4 and 14.*)

[251] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *International Joint Conference on Automated Reasoning*, pages 171–178. Springer, 2008. (*Cited on page 21.*)

[252] Amir Pnueli. The temporal logic of programs. In *FOCS*, 1977. (*Cited on pages 5, 23, and 42.*)

[253] Klaus Pohl. *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010. (*Cited on page 41.*)

[254] A. Post, I. Menzel, and A. Podelski. Applying restricted English grammar on automotive requirements–does it work? a case study. In *RREFSQ Workshop*, 2011. (*Cited on pages 7 and 73.*)

[255] Zhihua Qu. *Cooperative control of dynamical systems: applications to autonomous vehicles*. Springer Science & Business Media, 2009. (*Cited on page 115.*)

[256] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on programming*, pages 337–351. Springer, 1982. (*Cited on page 23.*)

[257] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011. (*Cited on pages 132 and 133.*)

[258] Ragunathan Rajkumar. A cyber–physical future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1309–1312, 2012. (*Cited on pages 3 and 26.*)

[259] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: the next computing revolution. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 731–736. IEEE, 2010. (*Cited on pages 2, 3, 25, 26, 27, and 28.*)

[260] Jean-François Raskin. An introduction to hybrid automata. In *Handbook of networked and embedded control systems*, pages 491–517. Springer, 2005. (*Cited on pages 7, 14, and 56.*)

[261] Alexandre Rocca. *Formal methods for modelling and validation of biological models*. PhD thesis, Université Grenoble Alpes, 2018. (*Cited on pages 5 and 20.*)

[262] Mauno Rönkkö, Anders P Ravn, and Kaisa Sere. Hybrid action systems. *Theoretical Computer Science*, 290(1):937–973, 2003. (*Cited on page 14.*)

[263] Sriram Sankaranarayanan, Thao Dang, and Franjo Ivančić. Symbolic model checking of hybrid systems using template polyhedra. In *TACAS Conference*. Springer, 2008. (*Cited on pages 5 and 20.*)

[264] Peter Schrammel and Bertrand Jeannet. From hybrid data-flow languages to hybrid automata: A complete translation. In *HSCC Conference*. ACM, 2012. (*Cited on pages 8, 104, and 112.*)

[265] S. Schuler, F. D. Adegas, and A. Anta. Benchmark problem: hyrid modelling of a wind turbine. In *ARCH Workshop*, 2016. (*Cited on page 49.*)

[266] S. Schuler, A. Walsch, and M. Woehrle. Unifying control and verification of cyber–physical systems (UnCoVerCPS), Deliverable D1.1 — Assessment of languages and tools for the automatic formalisation of system requirements. `http://cps-vo.org/node/24197`, 2016. (*Cited on page 49.*)

[267] Simone Schuler, Fabiano Daher Adegas, and Adolfo Anta. Hybrid modelling of a wind turbine (benchmark proposal). Applied Verification for Continuous and Hybrid Systems (ARCH), 2016. (*Cited on pages 120, 121, and 144.*)

[268] Stefan Schupp, Erika Abraham, Ibtissem Ben Makhlouf, and Stefan Kowalewski. HyPro: A C++ library of state set representations for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*, pages 288–294. Springer, 2017. (*Cited on page 21.*)

[269] Bastian Schürmann and Matthias Althoff. Convex interpolation control with formal guarantees for disturbed and constrained nonlinear systems. In *HSCC*, 2017. (*Cited on page 130.*)

[270] Rolf Schwitter. Controlled natural languages for knowledge representation. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 1113–1121. Association for Computational Linguistics, 2010. (*Cited on page 6.*)

[271] Bran Selic and Sébastien Gérard. *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems*. Elsevier, 2013. (*Cited on page 32.*)

[272] Ondřej Šerỳ. Enhanced property specification and verification in BLAST. In *International Conference on Fundamental Approaches to Software Engineering*, pages 456–469. Springer, 2009. (*Cited on page 7.*)

[273] Sanjit A Seshia, Shiyan Hu, Wenchao Li, and Qi Zhu. Design automation of cyber-physical systems: challenges, advances, and opportunities. *TCAD Journal*, 36(9):1421–1434, 2017. (*Cited on pages 1, 28, and 31.*)

[274] Vassiliki Sfyrla, Georgios Tsiligiannis, Iris Safaka, Marius Bozga, and Joseph Sifakis. Compositional translation of Simulink models into synchronous BIP. In *Industrial Embedded Systems (SIES), 2010 International Symposium on*, pages 217–220. IEEE, 2010. (*Cited on page 8.*)

[275] Lui Sha, Sathish Gopalakrishnan, Xue Liu, and Qixin Wang. Cyber-physical systems: A new frontier. 2008. (*Cited on page 26.*)

[276] Sigurd Skogestad and Ian Postlethwaite. *Multivariable feedback control: analysis and design*, volume 2. Wiley New York, 2007. (*Cited on pages 1, 31, and 32.*)

[277] MacKenzie Smith, Mary Barton, Mick Bass, Margret Branschofsky, Greg McClellan, Dave Stuve, Robert Tansley, and Julie Harford Walker. DSpace: An open source dynamic digital repository. 2003. (*Cited on page 32.*)

[278] Ian Sommerville and Pete Sawyer. *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997. (*Cited on page 41.*)

[279] Springer-Verlag. International Journal on Software Tools for Technology Transfer (STTT). (*Cited on page 29.*)

[280] Stateflow 8.7. The MathWorks, Inc., Natick, Massachusetts, United States. http://www.mathworks.com/products/stateflow/. (*Cited on pages 7 and 33.*)

[281] Thomas Strathmann and Jens Oehlerking. Verifying properties of an electro-mechanical braking system. In *ARCH Workshop*, 2015. (*Cited on pages 49 and 71.*)

[282] Paulo Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009. (*Cited on pages 4 and 14.*)

[283] Esterel Technologies. ANSYS Simplorer. www.ansys.com, 2014. (*Cited on pages 4 and 36.*)

[284] Russ Tedrake. Underactuated robotics: Learning, planning, and control for efficient and agile machines course notes for MIT 6.832, 2009. (*Cited on page 88.*)

[285] Bernhard Amadeus Thiele. *Framework for Modelica Based Function Development*. PhD thesis, Technische Universität München, 2015. (*Cited on page 32.*)

[286] Michael Tiller. *Introduction to physical modeling with Modelica*, volume 615. Springer Science & Business Media, 2012. (*Cited on page 32.*)

[287] Ashish Tiwari. Formal semantics and analysis methods for simulink stateflow models. Technical report, Technical report, SRI International, 2002. (*Cited on pages 9 and 113.*)

[288] Ashish Tiwari. Abstractions for hybrid systems. *Formal Methods in System Design*, 32(1):57–83, 2008. (*Cited on pages 5 and 20.*)

[289] Martin Treiber, Arne Kesting, and Dirk Helbing. Understanding widely scattered traffic flows, the capacity drop, and platoons as effects of variance-driven time gaps. *Physical review E*, 2006. (*Cited on page 130.*)

[290] Stavros Tripakis, Christos Sofronis, Paul Caspi, and Adrian Curic. Translating discrete-time Simulink to Lustre. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(4):779–818, 2005. (*Cited on page 8.*)

[291] Torgeir Vaa, Merja Penttinen, and I Spyropoulou. Intelligent transport systems and effects on road traffic accidents: state of the art. *IET Intelligent Transport Systems*, 1(2):81–88, 2007. (*Cited on page 115.*)

[292] Ardalan Vahidi and Azim Eskandarian. Research advances in intelligent collision avoidance and adaptive cruise control. *IEEE transactions on intelligent transportation systems*, 4(3):143–153, 2003. (*Cited on page 116.*)

[293] Richard van der Horst and Jeroen Hogema. *Time-to-collision and collision avoidance systems*. 1993. (*Cited on page 131.*)

[294] Arjan J Van Der Schaft and Johannes Maria Schumacher. *An introduction to hybrid dynamical systems*, volume 251. Springer, 2000. (*Cited on page 14.*)

[295] Pravin Varaiya. Reach set computation using optimal control. In *Verification of Digital and Hybrid Systems*, pages 323–331. Springer, 2000. (*Cited on page 105.*)

[296] Norbert Wiener. *Cybernetics or Control and Communication in the Animal and the Machine*, volume 25. MIT press, 1961. (*Cited on page 25.*)

[297] N. Zhan, S. Wang, and H. Zhao. *Formal verification of Simulink/Stateflow diagrams: a deductive approach*. Springer International Publishing, 2016. (*Cited on pages 1, 2, 8, 9, 32, 112, and 113.*)

[298] Liang Zou, Naijun Zhan, Shuling Wang, and Martin Fränzle. Formal verification of simulink/stateflow diagrams. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2015. (*Cited on pages 4, 7, 32, 33, 89, and 113.*)

[299] 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. (*Cited on page 8.*)

Formal Verification of Cyber-Physical Systems in the Industrial Model-Based Design Process

Nikolaos Kekatos

Thesis Directed By **Goran Frehse and Thao Dang**

Cyber-Physical Systems form a class of complex, large-scale systems of frequently safety-critical nature. Formal verification approaches can provide performance and safety guarantees for these systems. They require two elements, a formal model and a set of formal specifications. However, industrial models are typically non-formal, they are analyzed in non-formal simulation environments, and their specifications are described in non-formal natural language. In this thesis, we aim to facilitate the integration of formal verification into industrial model-based design.

Our first key contribution is a model transformation methodology. Starting with a standard simulation model, we transform it into an equivalent verification model, a network of hybrid automata. The transformation process addresses differences in syntax, semantics, and other aspects of modeling. For this class of formal models, so-called reachability algorithms can be applied to verify safety properties. An obstacle is that scalable algorithms exist for piecewise affine (PWA) models, but not for nonlinear ones. To obtain PWA over-approximations of nonlinear dynamics, we propose a compositional syntactic hybridization technique. The result is a highly compact model that retains the modular structure of the original simulation model and largely avoids an explosion in the number of partitions.

The second key contribution is an approach to encode rich formal specifications so that they can be interpreted by tools for reachability. Herein, we consider specifications expressed by *pattern templates* since they are close to natural language and can be easily understood by non-expert users. We provide (i) formal definitions for select patterns that respect the semantics of hybrid automata, and (ii) monitors which encode the properties as the reachability of an error state. By composing these monitors with the formal model under study, the properties can be checked by off-the-shelf fully automated verification tools.

Furthermore, we provide a semi-automated toolchain and present results from case studies conducted in collaboration with industrial partners.

Les systèmes cyber-physiques sont une classe de systèmes complexes, de grande échelle, souvent critiques enlever de sûreté, qui apparaissent dans des applications industrielles variées. Des approches de vérification formelle sont capable de fournir des garanties pour la performance et la sûreté de ces systèmes. Elles nécessitent trois éléments: un modèle formel, une méthode de vérification, ainsi qu'un ensemble de spécifications formelles. En revanche, les modèles industriels sont typiquement informels, ambigus par nature. Ils sont analysés dans des environnements de simulation informels et leurs spécifications sont décrites dans un langage naturel informel. Dans cette thèse, nous visons à faciliter l'intégration de la vérification formelle dans le processus industriel de la conception basée sur des modèles.

Notre première contribution clé est une méthodologie de transformation de modèle. À partir d'un modèle de simulation standard, nous le transformons en un modèle de vérification équivalent, plus précisément en un réseau d'automates hybrides. Pour cette classe de modèle formel, des algorithmes de l'atteignabilité peuvent être appliqués pour vérifier des propriétés de sûreté. Le processus de transformation prend en compte les différences de syntaxe, de sémantique et d'autres aspects de la modélisation. L'un des obstacles rencontré est que des algorithmes d'atteignabilité passent à l'échelle pour des modèles affines par morceaux, mais pas pour des modèles non linéaires. Pour obtenir des surapproximations affines par morceaux des dynamiques non linéaires, nous proposons une technique compositionnelle d'hybridisation syntaxique. Le résultat est un modèle très compact qui retient la structure modulaire du modèle d'origine de simulation, tout en évitant une explosion du nombre de partitions.

La seconde contribution clé est une approche pour encoder des spécificatio-ns formelles riches de façon à ce qu'elles puissent être interprétées par des outils d'atteignabilité. Nous prenons en compte des spécifications exprimées sous forme d'un gabarit de motif (pattern template), puisqu'elles sont proches du langage naturel et peuvent être comprises facilement par des utilisateurs non experts. Nous fournissons (i) des définitions formelles pour des motifs choisis, qui respectent la sémantique des automates hybrides, et (ii) des observateurs qui encodent les propriétés en tant qu'atteignabilité d'un état d'erreur. En composant ces observateurs avec le modèle formel, les propriétés peuvent être vérifiées par des outils standards de vérification qui sont automatisés.

Finalement, nous présentons une chaîne d'outils semi-automatisée ainsi que des études de cas menées en collaboration avec des partenaires industriels.

Communauté UNIVERSITÉ Grenoble Alpes

Verimag