Analysis and Synthesis of Hybrid Systems in Engineering Applications

Von der Fakultät für Mathematik,
Informatik und Naturwissenschaften
der RWTH Aachen University
zur Erlangung des akademischen Grades
einer Doktorin der Naturwissenschaften
genehmigte Dissertation

vorgelegt von

Diplom-Informatikerin
Johanna Nellen

aus Düren

Berichter:
Univ.-Prof. Dr. Erika Ábrahám
Priv.-Doz. Dr. Walter Unger

Tag der mündlichen Prüfung: 12.12.2016

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.

Abstract

This thesis deals with analysis and synthesis methods and their application in the area of engineering. In the first part, we show the application of formal methods in chemical plant control to verify safety critical properties. In general, not only the control program but also the controlled system are relevant to prove a set of requirements. Thus, we show how a hybrid automaton can be generated automatically, that models the controller, its cyclic execution, and the dynamic plant behavior.

We propose verification techniques based on counterexample-guided abstraction refinement (CEGAR) to keep the model sizes moderate. However, existing analysis tools do not compute counterexamples for models beyond timed automata. Thus, we present different methods to over-approximate the set of counterexamples for hybrid automata and we employ simulation techniques to validate counterexamples.

Afterwards, we develop two CEGAR approaches for our application scenario. The first one starts with an analysis on a purely discrete model. For each discrete counterexample, a reachability analysis that is guided along the discrete counterexample is computed on the hybrid system model. The second approach uses reachability analysis for hybrid systems and abstracts away parts of the dynamic plant behavior.

Finally, we show that some special characteristics of our models can be exploited during the analysis to reduce the computation time and to increase the accuracy of the computed reachable state set.

In the second part of this thesis, we switch to the synthesis of control strategies for parallel hybrid vehicles where an internal combustion engine and an electrical motor are coupled on the same axis. A control strategy distributes the requested torque over the available engines.

We implement a control strategy that optimizes the control using a genetic algorithm (GA). The basis of this control strategy is our GA library GENEIAL. We analyze the control strategy to determine real-time capable configurations with good optimization results. For promising configurations, we compare the GA-based control strategy with common control strategies. Moreover, we report on the integration of this set of control strategies into a learning-based control strategy. A learning-based control strategy optimizes the fuel consumption using a set of control strategies as experts. We can show that the fuel consumption of the learning-based control strategy is comparable to the fuel consumption of the best expert. On all benchmarks, GA-based control strategies turn out to be the best experts.

Zusammenfassung

Diese Doktorarbeit befasst sich mit Methoden der Analyse und Synthese und deren Anwendung im Ingenieurswesen. Im ersten Teil zeigen wir die Anwendung formaler Methoden zur Verifikation sicherheitskritischer Eigenschaften in der Steuerung von chemischen Anlagen. Im Allgemeinen ist nicht nur das Steuerungsprogramm sondern auch das gesteuerte System relevant, um eine Anforderungsmenge zu überprüfen. Daher zeigen wir, wie ein hybrider Automat, der das Steuerungsprogramm, seine zyklische Ausführung und das dynamische Verhalten der Anlage modelliert, automatisch erzeugt werden kann.

Wir entwickeln Verifikationstechniken, die Gegenbeispiel-gesteuerte Abstraktionsverfeinerung (CEGAR) verwenden, um die Modellgröße moderat zu halten. Allerdings berechnen existierende Analysetools keine Gegenbeispiele für Modelle jenseits von Zeitautomaten. Daher präsentieren wir verschiedene Methoden, um die Menge der Gegenbeispiele für hybride Automaten zu überapproximieren, und nutzen Simulationstechniken, um Gegenbeispiele zu validieren.

Anschließend präsentieren wir für unser Anwendungsszenario zwei CEGAR-Verfahren. Der erste Ansatz führt eine Analyse auf einem rein diskreten Modell durch. Für jedes diskrete Gegenbeispiel wird eine Erreichbarkeitsanalyse auf dem hybriden System berechnet, die entlang des diskreten Gegenbeispiels gesteuert wird. Der zweite Ansatz verwendet Erreichbarkeitsanalyse für hybride Systeme und abstrahiert Teile des dynamischen Anlagenverhaltens.

Abschließend zeigen wir, dass spezielle Eigenschaften unserer Modelle während der Analyse ausgenutzt werden können, um die Berechnungszeit zu verkürzen und die Genauigkeit der erreichbaren Zustandsmenge zu verbessern.

Im zweiten Teil der Doktorarbeit widmen wir uns der Synthese von Betriebsstrategien für parallele Hybridfahrzeuge, bei denen ein elektrischer Motor mit einem Verbrennungsmotor auf einem Antriebsstrang mechanisch gekoppelt ist. Eine Betriebsstrategie verteilt das geforderte Drehmoment auf die verfügbaren Motoren.

Wir implementieren eine Betriebsstrategie, die mit Hilfe von genetischen Algorithmen (GA) die Kontrolle optimiert. Die Basis dieser Strategie bildet unsere GA Bibliothek GeneiAL. Wir analysieren die Betriebsstrategie, um echtzeitfähige Konfigurationen mit guten Optimierungsergebnissen zu ermitteln und vergleichen die GA-basierte Strategie mit herkömmlichen Betriebsstrategien. Außerdem berichten wir über die Integration dieser Strategien in eine lernbasierte Betriebsstrategie, die den Kraftstoffverbrauch unter Nutzung einer Menge von Betriebsstrategien als Experten optimiert. Wir zeigen, dass der Kraftstoffverbrauch der lernbasierten Betriebsstrategie vergleichbar zum Kraftstoffverbrauch des besten Experten ist. In unseren Experimenten erweist sich die GA-basierte Strategie als bester Experte.

Acknowledgements

First of all, I want to thank Erika Ábrahám for all her advice and during the last six years! She gave me the opportunity of doing my PhD in her research group and guided me all the way to the final goal.

Second on my list is Walter Unger, who agreed to become my second examiner. I am grateful, that even in times of heavy workload he managed to find time for my thesis!

Looking back, there have been collaborations with many people during the last six years. I am grateful for all the fruitful discussions on my research. Special thanks goes to Sascha Geulen for the close collaboration on the OASys project and for his endurance during the paper writing days and nights. To Ulrich Loup who did valuable preparatory work on the OASys project, to Martin Neuhäußer and Kai Driessen for the collaboration on the bounded model checking approach, and to Stefan Schupp who implemented our ideas for improved verification results in HYPRO and who provided the color package that I used in my thesis.

Moreover, I want to thank all the students that assisted me during my research. Especially Kai Driessen, Lukas Netz, Thomas Osterland, and Benedikt Wolters have earned a big thank you for their valuable work.

Then of course, there are my colleagues Florian, Gereon, Nils, Stefan, Ulrich, and Xin from the research group *Theory of Hybrid Systems*. It was great being part of the team and there have been some legendary *Hybrid Systems Evenings* I will never forget!

Last but not least, I want to thank my family and my friends for their support and for always believing in me. The final thank you is for Christopher. For always being at my side, for his patience, and for his colorful parrots.



Für Marianne



Contents

Introduction xvii

Contribution of the Thesis xxi

Outline xxvii

I. Analysis of Automated Control in Chemical Plants 1

1. Introduction 3

2. Related work 9
   2.1. Reachability Analysis of Hybrid Automata 9
      2.1.1. State Set Representation 10
      2.1.2. Tools 10
      2.1.3. CEGAR-based Analysis 11
   2.2. Bounded Model Checking 11
   2.3. CEGAR 11
   2.4. Verification of SFC-Controlled Plants 12

3. Preliminaries 13
   3.1. Numbers, Formulas, and Predicates 13
   3.2. Chemical Plants 14
   3.3. Sequential Function Charts (SFCs) 16
   3.4. Hybrid Automata (HA) 21
      3.4.1. Reachability Analysis of HA 27
   3.5. Bounded Model Checking 28
   3.6. Counterexample-Guided Abstraction Refinement (CEGAR) 30

4. Modeling SFC-Controlled Plants 33
   4.1. Interface 34
   4.2. Plant Specifications 34
   4.3. Hybrid Automata Models for SFC Controller 36
   4.4. Modeling the PLC Cycle Synchronization 39
   4.5. Modeling the Plant Dynamics 40
   4.6. Parallel Composition 42
   4.7. Outlook 43

5. Counterexamples 45
5.1. Generating Traces for Presumable Counterexamples . . . 47
5.1.1. Approach I: Model Augmentation . . . 48
5.1.2. Approach II: Parsing the Output of SpaceEx . . . 53
5.1.3. Approach III: Extending the Functionality of Flow* . . . 55
5.2. Generating a Presumable Counterexample . . . 55
5.3. Simulation . . . 57
5.3.1. Finding Candidate Initial States . . . 57
5.3.2. Simulating the Dynamics . . . 58
5.3.3. Checking Invariants . . . 58
5.3.4. Dynamic Search for Suitable Jump Time Points . . . 59
5.4. Outlook . . . 61

6. CEGAR-based Verification I: Bounded Model Checking 63
6.1. Discrete Analysis . . . 63
6.2. Hybrid Analysis . . . 65
6.2.1. Model Generation . . . 65
6.2.2. Hybrid Analysis . . . 67
6.2.3. Generating Explanations . . . 67
6.3. Enhancements . . . 67
6.4. Outlook . . . 68

7. CEGAR-based Verification II: SpaceEx Integration 69
7.1. Abstraction and Abstraction Refinement . . . 70
7.1.1. Abstraction . . . 70
7.1.2. Counterexample-Guided Abstraction Refinement . . . 70
7.1.3. Building the Model at a Given Level of Abstraction . . . 71
7.1.4. Dealing with Urgency . . . 72
7.1.5. Dealing with Zeno Paths . . . 73
7.1.6. CEGAR Iterations . . . 74
7.2. Integrating CEGAR into the Reachability Analysis . . . 74
7.2.1. Adapting the Reachability Analysis Algorithm . . . 74
7.2.2. Implementation . . . 76
7.3. Outlook . . . 79

8. Advanced Verification Techniques for PLC-Controlled Plants 81
8.1. Discrete Variables . . . 82
8.2. Urgency . . . 82
8.3. Automata Models . . . 84
8.4. Outlook . . . 84

9. Experimental Results 89
9.1. Benchmarks . . . 89
9.1.1. Thermostat . . . 89

9.1.2. Leaking Water Tank . . . 91
9.1.3. Two Tank System . . . 93
9.2. CEGAR-based Verification I: Bounded Model Checking . . . 95
9.3. CEGAR-based Verification II: SpaceEx Integration . . . 96
9.4. Advanced Verification Techniques for PLC-Controlled Plants . . . 99
9.5. Outlook . . . 101

10. Summary 105

II. Synthesis of Control Strategies for Hybrid Vehicles 109

11. Introduction 111

12. Preliminaries 115
12.1. Vehicle Model . . . 115
12.2. Energy Management . . . 117
12.3. Genetic Algorithms (GAs) . . . 118
12.3.1. GeneiAL . . . 119

13. Basic Control Strategies 121
13.1. Non-predictive Control Strategies . . . 121
13.1.1. ICE and EM Control Strategies . . . 122
13.1.2. Equivalent Consumption Minimization Strategy (ECMS) . . . 122
13.1.3. Adaptive ECMS (A-ECMS) . . . 122
13.1.4. Telemetry ECMS (T-ECMS) . . . 123
13.2. Predictive Control Strategies . . . 123
13.2.1. Receding Dynamic Programming (RDP) . . . 124
13.2.2. Receding Dynamic Programming with Cost-to-Go Approximation (ADP) . . . 124
13.2.3. Genetic Algorithm (GA) . . . 124
13.3. Outlook . . . 128

14. Learning-based Control Strategies 129
14.1. Structure of the Learning-based Control Strategies . . . 129
14.2. Shrinking Dartboard . . . 131
14.3. Weighted Fractional . . . 131
14.4. Outlook . . . 132

15. Experimental Results 133
15.1. The Benchmark Settings . . . 133
15.2. Comparison of the Control Strategies . . . 136
15.3. Evaluation of the GA-based Control Strategy . . . 137
15.3.1. Fitness Evaluation and Drivability . . . 137
15.3.2. Real-time Capability . . . 139

15.4. Evaluation of the Learning-based Control Strategies 139
15.5. Outlook 141

16. Summary 143

Conclusion 147

Literature 149

Notations 157
Part I – Analysis of Automated Control in Chemical Plants 157
Part II – Synthesis of Control Strategies for Hybrid Vehicles 163

Glossary 167

Introduction

In industry, technology has become an indispensable component. Looking back to the second half of the 18th century and the beginning of the 19th century, the first industrial revolution introduced mechanization as well as water and steam power. The second industrial revolution already made mass production possible by the use of electricity, e. g. to run assembly lines. Computers gave way to automation by controlling the production, e. g. via *programmable logic controllers*, which was the third industrial revolution. Nowadays, *Industry 4.0* is a commonly used buzzword in industrial production and is sometimes called the fourth industrial revolution. It bases on the idea of smart factories where *cyber-physical systems* organize the production process with as little human intervention as possible. Cloud computing and data analysis play an essential role in Industry 4.0 to make massive amounts of data available via the *Internet of Things*. For example, the necessary processing steps for a product can be stored on a chip together with the current manufacturing status, such that the machines can evaluate which steps they have to perform on the given piece.

With the increasing use of computers in production, safety aspects become more and more important. Contamination of the environment, machine damage, and faulty products that occur due to errors in the control software can often be avoided by a careful analysis of the plant specification. To find bugs in the control software, simulation is extensively used nowadays. However, it is limited to the subset of possible system behavior that is covered by the test cases. In contrast to that, verification covers the complete behavior of a system model and gives a mathematical proof that the model exhibits a specific behavior. However, it is only applicable to small systems since the analysis is time consuming and suffers from state space explosion. A thorough examination should not only consider the control program which is often purely discrete but also the controlled environment where physical quantities evolve continuously. Thus, we get *hybrid* system models, which are models that exhibit both discrete and continuous behavior. These models typically have an infinite state space which can in general not be analyzed exactly. However, in this thesis we take the challenge and propose verification methods that are capable of analyzing small- and mid-scale applications from the area of plant control.

When looking at the *automotive sector*, both the digitalization and environmental characteristics play a big role in vehicle production. Not only Industry 4.0 is increasingly used, also the vehicles themselves are equipped with a multitude of chips and sensors. To ensure safety, the driver is supported by digital systems that help in dangerous situations. One example is the anti-lock braking system. With the introduction of the first autonomously driving vehicles, the need for sophisti-

cated localization techniques and the abilities to detect and respond to unexpected influences from the environment within a split second become important.

Also environmental aspects have to be considered by manufacturers. To reduce pollutant emissions, alternative propulsion systems have been examined. However, in matters of driving range, maximal performance, size of the energy storage, and safety for the passengers and the environment, it is hard to rule out conventional combustion engines. Hybrid vehicles seem to be a compromise, as they can exploit and combine the advantages of different propulsion systems. However, there is a need for control programs that ensure that the benefits of the hybrid propulsion system are noticeable.

In the digital age computer science offers valuable knowledge in form of specialized algorithms that are applicable to manifold different application areas with much potential for the *synthesis* of innovative products. We had a closer look at some optimization algorithms and developed a control strategy that distributed the requested torque to the available engines.

Thus, in this thesis we show examples how both synthesis and analysis can be applied in the area of engineering. *Synthesis* is important to develop new products by combining existing components as algorithms, hardware components, mathematical models, etc. However, especially for safety critical systems, a thorough and detailed *analysis* of both software and hardware is advisable, where a system is divided into subcomponents. This has the potential of ruling out bugs that might have caused accidents or costly recalls.

Analysis of Automated Control in Production

Automated control consists of two parts: the control program and the controlled system. While the control program is usually purely discrete, the controlled system exhibits continuous behavior in most cases. Physical quantities like temperature, pressure, or the filling level of storage containers are continuous values. A common language to specify control programs are *sequential function charts (SFCs)*. Since SFCs are a graphical programming language, they allow a fast understanding of the control flow. The controlled system can behave differently depending on the current system state. Typically, the controlled system can be modeled as a set of physical quantities, whose behavior is given in form of a set of differential equations together with the conditions under which the specific behavior is enabled.

The combination of an SFC with the controlled system yields a hybrid system, i. e. a system with combined discrete and continuous behavior. For the analysis, formal methods can be used. However, the verification of hybrid systems is a challenging problem, as the reachability problem for hybrid systems is in general undecidable. Existing tools use over-approximative methods such that it is possible to verify a system, but falsification would only be possible with under-approximation which is not supported by state-of-the-art tools. A second problem is the state space. Even relatively small systems yield complex models where the verification is time consuming.

Thus, we propose a *CEGAR* approach. Starting with a model of the discrete control program and abstracting away the continuous behavior of the controlled system, counterexamples are analyzed to refine the model. Thus, in the end, those parts of the dynamics that are relevant to verify the safety property have been added to the abstract system model.

This approach relies on the following components: First of all, a formal semantics for SFCs is needed. Therefore, we extended the work from [Bau04, BHLE04] and developed an automated translation from SFCs and the plant dynamics to hybrid automata [1]. In a CEGAR approach, counterexamples are the basis for the refinement decision. Unfortunately, existing analysis tools perform a complete analysis before computing the intersection of the reachable states with the unsafe states. The output is also tool dependent: whereas SpaceEx [Spa] returns the set of reachable (bad) states and no counterexamples, Flow* [CÁS13] returns both the set of reachable states and the detected counterexamples. However, due to the over-approximative computation of the reachable states, those counterexamples might be spurious, i. e. they might not be realizable in the original system. We proposed simulation to further analyze counterexamples for spuriousness [2].

Being able to extract counterexamples, we developed a first CEGAR-based approach for plant control models. This approach combines discrete bounded model checking (BMC) with reachability analysis of hybrid systems. Since the hybrid system model should be as small as possible, a complete analysis is performed on the discrete controller model. This allows to guide the reachability analysis for the hybrid model along discrete counterexamples. This method has the potential to exploit the fast analysis techniques for discrete models. Since this CEGAR-based approach did not yield satisfactory running time, we developed a second CEGAR-based approach. This operates on hybrid models only and is directly integrated into an existing analysis tool [3, 4]. We extended SpaceEx such that the analysis is interrupted if a counterexample is found. The refinement can be done manually for more flexibility. However, automated refinement strategies could easily be implemented.

In a last step, we analyzed our models to support the reachability analysis by advanced verification techniques that accelerate the analysis of models from the area of plant control without extending the expressiveness of the modeling language. We support discrete variables, which have been modeled as continuous variables with constant zero derivatives so far. Moreover, we integrated direct support for urgency into the analysis tool HyPro. Experiments show that these techniques improve both the running time and the accuracy of the analysis which enables the verification of bigger system models.

Synthesis of Control Strategies for Hybrid Vehicles

The second part of this thesis considers parallel hybrid electric vehicles (PHEVs). Analysis in this field is complicated, since the engine specifications are often given by characteristic diagrams which are difficult to express in a closed form. Instead,

we concentrate on synthesis now. The goal is to develop control strategies that distribute the requested torque over the available engines. These control strategies can either be used as stand-alone strategies or in a learning-based control strategy as one strategy in a set of control strategies.

First of all, it is essential to have a vehicle model to which a developed control strategy can be applied. In the context of the DFG research project *OASys*, a `MATLAB/Simulink` model has been developed for the simulation of a hybrid vehicle. Based on this model, we generated `C++` code and developed a `C++` vehicle model that is used by our optimization-based control strategies to compute the outcome for possible solutions during the optimization process. We focus on optimization algorithms from the field of artificial intelligence.

Our control strategy uses *genetic algorithms (GAs)* for optimization [7] which perform a heuristic search on a set of solution candidates over several iterations. These candidates evolve from iteration to iteration imitating the principles of evolution. Thus, new solution candidates are built by combining two existing ones, candidates are mutated to create diversity, and the elite survives in the next iteration to avoid regression in the optimization process. We implemented our own library for genetic algorithms [9] which was designed to our needs.

The main focus of the OASys project is a learning-based algorithm that uses a set of control strategies as experts. The control sequences of these experts are combined to a new control strategy by the learning-based algorithm [11, 6]. We tested our GA-based control strategy both as a standalone strategy and as an expert in the learning-based algorithm.

Contribution of the Thesis

In this thesis, we present our approaches to apply analysis and synthesis to applications scenarios in engineering. Over the years, I collaborated with many different people from the areas of computer and engineering science. Numerous fruitful discussions with professors, PhD, master, and bachelor students led to the publications that are relevant for this thesis. Below I clarify my own contributions to this work and those parts that other people contributed. I do this separately for the first part of the thesis that deals with the analysis of automated control in chemical plants and for the second part about the synthesis of control strategies for hybrid vehicles. Afterwards, the list of publications that are relevant for this thesis is given.

Analysis of Automated Control in Chemical Plants

When I started my PhD, we concentrated on applying formal methods in the application area of engineering by developing approaches to verify safety properties of control programs considering the program itself and the controlled plant in our analysis. This work has been supported by the DFG (Deutsche Forschungsgemeinschaft, engl. German Research Foundation) in form of a scholarship for the research training group AlgoSyn (Algorithmic Synthesis of Reactive and Discrete-Continuous Systems).

After many discussions with Ulrich Epple and David Kampert from the chair of Process Control Engineering at RWTH Aachen University, we decided to consider SFCs running on a programmable logic controller (PLC) as control programs and started to develop a transformation from SFCs to hybrid automata. For [1], I developed such a transformation together with Erika Ábrahám. Therefore, we slightly adapted the semantics of SFCs from [Bau04, BHLE04].

Knowing that a CEGAR-based analysis approach was needed to keep the model size as small as possible during the process of verification, we faced the problem that existing tools for the reachability analysis of hybrid automata do compute the set of reachable states and the intersection with the unsafe states, but do not return a counterexample in form of a system run starting in an initial state and ending in an unsafe state. In discussions together with Erika Ábrahám, my colleague Xin Chen, and Pieter Collins from Maastricht University, we tackled the problem of generating and validating counterexamples for hybrid systems. Together, we developed the theoretical background for several methods to compute over-approximations of counterexamples by either augmenting the model, by parsing information from the output of existing analysis tools, or by extending the functionality of an existing tool

such that counterexamples are computed. Moreover, we discussed, how possibly spurious counterexamples could be validated, i. e. we tried to find unsafe paths in the over-approximation. Although we developed the ideas collaboratively, I created the models for benchmarking and implemented the first two approaches to generate counterexamples and the validation method. Xin Chen as one of the authors of the analysis tool FLOW*extended the functionality of his tool, which is our third approach to generated counterexamples. We presented our results in [2].

With the new knowledge on the generation of counterexamples for hybrid systems, we started to develop a first CEGAR-based analysis approach. In cooperation with Martin Neuhäußer from Siemens AG, Erika Ábrahám, and Kai Driessen (as master and later as PhD student), I worked on the first CEGAR-based approach that combines BMC for discrete controller models with a reachability analysis for hybrid models. We developed the theoretical aspects collaboratively. The modeling and implementation work was divided into a bounded model checker part and into a part for the hybrid analysis of discrete counterexample paths. While Martin Neuhäußer provided the bounded model checker and the discrete controller models, Kai Driessen implemented the automated creation of a hybrid model for a discrete counterexample and the generation of explanations for the bounded model checker under my co-supervision. First results have been presented in the master thesis of Kai Driessen [Dri14]. Later, we improved our approach and together with Kai Driessen, I designed the parameters for our benchmarks and performed the analysis. The results are presented in [5].

Since the results of this first CEGAR-based approach have not been satisfactory, together with Erika Ábrahám, I worked out the theoretical background for a second analysis approach. We planned to integrate a CEGAR loop directly in the analysis tool SPACEEX. I extended SPACEEX by an implementation of a CEGAR loop together with Benedikt Wolters, who worked as a student assistant under my supervision. Together, we performed the benchmarks that have been published in [3, 4, 5]. The extended SPACEEX version is provided on our web page [8].

Although this second CEGAR approach is much faster, the analysis is still time consuming. I analyzed our models together with Erika Ábrahám and we found out that the analysis of our model would benefit from tool support for discrete variables and urgency. While Stefan Schupp, an author of the analysis tool HYPRO, implemented those features in his analysis tool, I prepared three benchmarks for the evaluation and worked out some further optimizations of our modeling approach. Together with Stefan Schupp, I performed the benchmarks and analyzed the results. This contribution to my thesis is ongoing work and has not been published yet.

During my work on the analysis of automated control in chemical plants, I supervised Kai Driessen, Kim Haps, Thomas Osterland, and Benedikt Wolters as scientific student assistants. Moreover, I assisted in the supervision of the Bachelor theses of Kai Driessen [Dri12], Kim Haps [Hap13], and Thomas Osterland [Ost13], as well as in the supervision of the Master thesis of Kai Driessen [Dri14]. The above-mentioned students contributed to the implementation of data structures and enhancements of the analysis for models from the area of plant control.

Synthesis of Control Strategies for Hybrid Vehicles

The second part of this thesis originates from my work on the interdisciplinary DFG research project OASys (Online Algorithms for Optimal Control of Hybrid Propulsion Systems, AB 461/2-1). On this project, we collaborated with Dirk Abel, Rainer Gasper, Frank-Josef Heßeler, Matthias Hoppe, Martina Joševski, and Jan Maschuw from the institute of Automatic Control at RWTH Aachen University and with Berthold Vöcking, Walter Unger, Janosch Fuchs, Sascha Geulen, and Melanie Winkler from the chair of Computer Science 1 at RWTH Aachen University. The aim of OASys was to combine optimization methods from the areas of control theory and artificial intelligence with online learning. Whereas the engineers from the institute of Automatic Control provided a shimulation model and different control strategies, the main contribution of the computer scientists from the chair of Computer Science 1 was the development of two learning-based control strategies. Our contribution to the project was to synthesize control strategies using methods from the area of artificial intelligence.

During the writing of the project proposal, I participated in many discussions on our contributions to the project with Erika Ábrahám and my colleague Ulrich Loup. When the project was granted by the DFG, we started with the development of a `MATLAB/Simulink` simulation and vehicle model. This task was completed in form of a master thesis [Ion12] by Alin-Dragoș Ionașcu. Together with my colleague Ulrich Loup I assisted in the supervision of the student. This first model was later replaced by an advanced `MATLAB/Simulink` model that has been developed by the engineers Frank-Josef Heßeler and Martina Joševski. This model has been adapted by Ulrich Loup such that the *Simulink Coder* could be used to generate `C++` code for the building blocks of the model. Around this generated code, he built a library that could be used as a vehicle model by `C++`-based control strategies or as a simulation model to test control strategies independently from `MATLAB/Simulink`. When Ulrich Loup left the group in 2014, I took over the model development and maintenance.

For the OASys project, several control strategies have been developed: Different equivalent consumption minimization strategies (ECMS, A-ECMS, T-ECMS) have been developed by Sascha Geulen, Martina Joševski, Michael Tegethoff, and Melanie Winkler. Two control strategies that are based on dynamic programming (RDP, ADP) have been implemented by Martina Joševski. Although I did not contribute to the development of those control strategies, they are presented in this thesis since they are used in the benchmark section for a comparison with our control strategy. My main contribution to the OASys project has been the development of a control strategy that uses genetic algorithms for the optimization of the control sequence. This work was assisted by the research students Lukas Netz and Benedikt Wolters. First of all, GeneiAL [9], a new library for genetic algorithms, was implemented by them under my supervision. Afterwards, we worked together on the development of a GA-based control strategy for PHEVs using GeneiAL and the `C++` vehicle model. We extensively tested the GA-based control strategy in our `C++`-based simulation

environment to find real-time capable parameters that yield good optimization results. Once, we found good parameter sets, we integrated the control strategy in our `MATLAB/Simulink` model. There, we compared our control strategy with the other control strategies that have been developed for OASys by our cooperation partners.

Janosch Fuchs, Sascha Geulen, Michael Tegethoff, and Melanie Winkler developed two learning-based control strategies and integrated them in our `MATLAB/Simulink` simulation model. Together with Benedikt Wolters, I supported Sascha Geulen and Janosch Fuchs with the integration of the `C++` vehicle model library in the learning-based control strategies. We tested the learning-based control strategies on different sets of basic control strategies collaboratively. Our GA-based control strategy has been among the set of basic control strategies. The results are presented in [6].

Not part of this thesis is the work of two bachelor students. I assisted in the supervision of Rebecca Haehn [Hae16] and Patricia Wessel [Wes16] who applied different methods to approximate basic control strategies to reduce the computation time. For their theses, they precomputed the control of different basic control strategies for discretized points of the state space. Rebecca Haehn used this data to train a neural network that can afterwards be used in a control strategy. Patricia Wessel used the precomputed data to interpolate the control for an arbitrary state from the neighboring precomputed control values.

Relevant Publications

In this section, the publications that resulted from my work as a PhD student, are presented. For the papers [1, 2, 3, 4, 5, 7], where I am listed as the first author, I did the main writing. However, Xin Chen provided the section on the FLOW* extension to compute counterexamples in [2]. Martin Neuhäußer wrote the preliminaries concerning the bounded model checking and explained the discrete part of the first CEGAR-based analysis approach in [5]. Kai Driessen contributed to the description of the hybrid part of the analysis and to the experimental results of [5]. Martina Joševski wrote the description of the vehicle model and the energy management problem in [7]. The experimental results of [7] have been written collaboratively with Sascha Geulen and Benedikt Wolters. For the publication [6], I worked as a co-author and contributed mainly to the experimental results. The implementation of a CEGAR loop in the analysis tool SPACEEX [8] was done collaboratively with the student research assistant Benedikt Wolters, and the student research assistants Benedikt Wolters and Lukas Netz implemented the library GENEIAL [9] for genetic algorithms under my supervision.

In this thesis, I will use text from those papers where I have been the first author without marking those parts explicitly by citations. However, I will clearly emphasize in the beginning of a chapter, when I use text from those papers, where I have been a co-author.

In the following, I list the publications that build the foundation of this thesis together with the information which chapters are based on which reference.

Peer-Reviewed Publications

[1] Johanna Nellen and Erika Ábrahám. Hybrid sequential function charts. In *Proc. of MBMV'12*, pages 109–120. Verlag Dr. Kovac, 2012.

Basis of Part I, Chapter 4.

[2] Johanna Nellen, Erika Ábrahám, Xin Chen, and Pieter Collins. Counterexample generation for hybrid automata. In *Proc. of FTSCS'13*, volume 419 of *CCIS*, pages 88–106. Springer, 2014.

Basis of Part I, Chapter 5.

[3] Johanna Nellen and Erika Ábrahám. A CEGAR approach for the reachability analysis of PLC-controlled chemical plants. In *Proc. of IRI'14*, pages 500–507. IEEE, 2014.

Basis of Part I, Chapters 4, 7, 9.

[4] Johanna Nellen, Erika Ábrahám, and Benedikt Wolters. A CEGAR tool for the reachability analysis of PLC-controlled plants using hybrid automata. In *Formalisms for Reuse and Systems Integration*, volume 346 of *AISC*, pages 55–78. Springer, 2015.

Basis of Part I, Chapters 4, 7, 9.

[5] Johanna Nellen, Kai Driessen, Martin Neuhäußer, Erika Ábrahám, and Benedikt Wolters. Two CEGAR-based approaches for the safety verification of PLC-controlled plants. *Information Systems Frontiers*, 18(5):927–952, 2016.

Basis of Part I, Chapters 4, 6, 7, 9.

[6] Sascha Geulen, Martina Josevski, Johanna Nellen, Janosch Fuchs, Lukas Netz, Benedikt Wolters, Dirk Abel, Erika Ábrahám, and Walter Unger. Learning-based control strategies for hybrid electric vehicles. In *Proc. of CCA'15*, pages 1722–1728. IEEE, 2015.

Basis of Part II, Chapters 12, 13, 14, 15.

[7] Johanna Nellen, Benedikt Wolters, Lukas Netz, Sascha Geulen, and Erika Ábrahám. A genetic algorithm based control strategy for the energy management problem in PHEVs. In *Proc. of GCAI'15*, volume 36 of *EPiC Series in Computer Science*, pages 196–214. EasyChair, 2015.

Basis of Part II, Chapters 12, 13, 15.

Tools

The following tools originated from my research.

[8] SpaceEx with CEGAR. http://ths.rwth-aachen.de/research/tools/spaceex-with-cegar/.

Implemented for Part I, Chapter 7.

[9] Genetic algorithm library GeneiAL. http://geneial.org.

Implemented for Part II, Chapters 12, 13.

Further Publications

I was a co-author of the following publications that are not part of this thesis.

[10] Erika Ábrahám, Nadine Bergner, Philipp Brauner, Florian Corzilius, Nils Jansen, Thiemo Leonhardt, Ulrich Loup, Johanna Nellen, and Ulrik Schroeder. On collaboratively conveying computer science to pupils. In *Proc. of KOLI'11*, pages 132–137. ACM, 2011.

[11] Sascha Geulen, Martina Josevski, Johanna Nellen, Janosch Fuchs, Lukas Netz, Benedikt Wolters, Erika Ábrahám, Walter Unger, and Dirk Abel. Online lernen als Kontrollstrategie in Hybridfahrzeugen. In *Proc. of AUTOREG'15*, volume 2233 of *VDI-Berichte*, pages 101–112. VDI Verlag, 2015.

Outline

This thesis is split into two parts which cover formal analysis in the application area of automated control in production (Part I) and synthesis of control strategies for vehicles with parallel hybrid propulsion systems (Part II).

Part I starts with an introduction into analysis of automated control in chemical plants in Chapter 1, followed by related work in Chapter 2, and preliminaries in Chapter 3. Our modeling approach for controller programs, the cyclic execution of a set of controllers on a programmable logic controller, and the plant dynamics are presented in Chapter 4. In Chapter 5, different approaches to extract and validate counterexamples during the reachability analysis of hybrid automata are presented. The main work of this first part are the two CEGAR-bases approaches for the verification of chemical plant control that are shown in Chapter 6 and Chapter 7. To improve the analysis for our special type of models, we developed advanced techniques for the verification that do not extend the expressiveness but yields better running time and improved accuracy. These techniques are presented in Chapter 8, before we show the experimental results for the two CEGAR approaches and the advanced verification techniques in Chapter 9. We conclude this part in Chapter 10.

An introduction for Part II of this thesis, which deals with the synthesis of control strategies for hybrid vehicles, is given in Chapter 11. Afterwards, the vehicle model, the energy management problem, and genetic algorithms, which build the basis of our control strategy, are introduced in Chapter 12. The basic control strategies that have been implemented for the OASys-project are explained in Chapter 13. Among them is our GA-based control strategy which is the main contribution to this second part of the thesis. All presented basic control strategies can be used either as standalone strategies or in combination with the learning-based control strategy from Chapter 14. The experimental results for this part of the thesis are shown in Chapter 15, before the synthesis part is concluded in Chapter 16.

In the back matter of this thesis, an overall summary and outlook of the presented topics in this thesis can be found on page 147 and the bibliography starts on page 149. The thesis is concluded with an alphabetically sorted list of notations used in Part I and in Part II (starting on page 157 and on page 163, respectively). A glossary where the acronyms are listed is given on page 166.

PART I

Analysis of Automated Control in Chemical Plants

1. Introduction

In automation *programmable logic controllers (PLCs)* are widely used to control the behavior of plants. The industry standard IEC 61131-3 [Int03] specifies several languages for the programming of PLCs. In this thesis, we consider *sequential function charts (SFCs)*, a graphical programming language which allows the structuring of control sequences into several steps or into branches that are executed in parallel. Since PLC-controlled plants are often safety-critical, SFC *verification* has been extensively studied [FL00].

Several approaches exist which consider either an SFC in isolation or the combination of an SFC with a model of the plant (see Section 2.4). As the combination of the controller and the plant has a mixed discrete-continuous behavior, the latter approaches usually define a timed or hybrid automaton that specifies the SFC, and a hybrid automaton that specifies the plant. The composition of these two models gives a hybrid automaton model of the controlled plant. Theoretically, this composed model can be analyzed using existing tools for hybrid systems reachability analysis. However, in practice, there are some issues that have to be tackled that come along with the special characteristics of the composed system models.

We propose to specify the dynamic behavior of plant components by sets of *conditional ordinary differential equation (ODE) systems*. The condition expresses assumptions about the current state of the system. For example, the dynamic change of the water level in a tank can be given as the sum of the flows through the pipes that fill and empty the tank. This sum may vary depending on valves being open or closed, pumps being on or off, and other tanks being empty or not. Each conditional ODE system specifies the behavior of a physical quantity of a plant for different conditions. Thus, for a given system state, always the first conditional ODE, whose condition is satisfied, is applied and specifies the dynamic behavior of the physical quantity.

This enables us to build separated hybrid automata models for each SFC controller and for different components of the physical plant (see [1, 3, 4, 5]). Our formalism allows a modular definition of the dynamic continuous behavior of plant elements with different levels of details for different control modes. This simplifies the application of analysis approaches that use counterexample-guided abstraction refinement (CEGAR).

A wide range of reachability analysis techniques and software tools have been developed for hybrid systems. In this thesis, we make use of reachability analysis techniques that are based on *flowpipe construction*. Since the reachability problem for hybrid systems is in general undecidable, most of these techniques compute *over-approximations* of the set of reachable states of hybrid systems. Starting from

a set of initial states, the system trajectories result from alternating time evolution (*flowpipes*) and discrete steps (*jumps*). Due to the over-approximation, those analysis tools can only be used to prove safety, i. e. that a given set of unsafe states cannot be reached from a set of initial states in a given model. However, if the over-approximation of the reachable states contains unsafe states, no conclusive answer can be given. For more information on the reachability analysis of hybrid automata we refer to Section 2.1.

Although many different analysis tools for hybrid systems have been developed, the verification of PLC-controlled chemical plants still remains a challenging problem. One particularity is that the size of global models for controlled plants, being the composition of (possibly several) controller programs and the plant, grows exponentially in the number of components. Both the control programs and the plant can be analyzed separately, however, their combination is highly challenging. The composed system models exhibit further characteristics that harden their safety analysis. For example, state-of-the-art hybrid systems reachability analysis tools do not support discrete variables, only continuous ones which leads to unnecessarily high-dimensional models. Another issue is the urgency of transitions, which is inherent to the controller models but is not yet covered by most of the analysis tools.

This was our motivation to develop dedicated abstraction techniques for this special class of (highly safety-critical) systems. As the models we consider are often too large to be handled by state-of-the-art tools, we focus on abstraction techniques to reduce the verification effort, simplifying the model in an over-approximative manner. Originating in the successful application of abstraction and *CEGAR* techniques to discrete models [CGJ+00], also several methods have been proposed to apply CEGAR to hybrid automata (see Section 2.3). However, none of the existing approaches exploits the special properties of hybrid models for chemical plant control. They either have been designed for control-dominated models with little dynamic behavior or they do not consider the dynamic plant behavior at all. A CEGAR-based abstraction technique for the verification of safety properties of PLC-controlled plants was published in [ELS05]. Given a hybrid automaton model of the controlled plant, the method abstracts away from parts of the continuous dynamics. However, instead of refining the dynamics in the hybrid model to exclude spurious counterexamples, their method adds information about enabled and disabled transitions.

Counterexamples in form of system runs leading to unsafe states would be extremely valuable, even if they are *spurious*, i. e. if they were considered in the analysis but are not possible in the given model. For safe models they could help to reduce the approximation error in the analysis efficiently, whereas for unsafe models they could provide important information about the source of the critical system behavior. Counterexamples would enable the application of *CEGAR* techniques and could also play an important role in controller synthesis.

Unfortunately, only very few of the available tools for the reachability analysis of hybrid automata with continuous time domain compute counterexamples. This is surprising since *internally* they possess sufficient information to generate at least a coarse over-approximation of a counterexample in form of a sequence of jumps (i. e.

changes in the discrete part of the system state), augmented with time intervals over-approximating the time durations between the jumps. In this thesis, we address the lack of counterexamples and provide solutions to generate counterexamples by either augmenting a given automaton model or by extending the functionality of existing tools *internally* [2]. Moreover, we develop a simulation-based approach to *validate* the counterexample over-approximations, i. e. to determine unsafe paths in the over-approximation.

We chose SpaceEx and Flow* for our experiments because on the one hand SpaceEx is one of the most popular analysis tools for hybrid automata and on the other hand we had good contact to some of the authors that belong to the implementation team of Flow*. Thus the modification of the source code of Flow* could be done safely. Unfortunately, counterexample generation without tool extension is unsatisfactory: we need either expensive additional analysis runs for enlarged systems or parsing hidden information from debug output. The results demonstrate the need to extend the functionality of available analysis tools to generate counterexamples internally. However, even if that task is done, the results strongly over-approximate counterexamples, whose existence can be indicated but not proven. Thus we need novel methods to refine and validate the results, posing highly challenging problems in both theory and practice.

As mentioned before, PLC-controlled chemical plants are hybrid systems, however, they have certain special characteristics that distinguish them from general hybrid systems. Moreover, PLC-controlled plants can strongly differ in, e. g. the complexity of their dynamics or the size and the control structure of their PLC programs. Thus if we want to employ abstraction techniques to such problems, we should not aim at a single general approach, but rather on different approaches, each of them dedicated to a special type of problems.

In Part I of this thesis we present two different CEGAR-based approaches, published in [3, 4, 5], that are suitable for the safety verification of PLC-controlled plants. The first one combines *bounded model checking (BMC)* on discrete models with reachability analysis on hybrid models that are restricted to specific control paths. The second methodology uses a hybrid abstraction and reachability analysis for hybrid systems. We developed prototypical implementations of both methods to prove the applicability of the presented ideas in the application area of plant control.

The first CEGAR-based analysis approach combines *BMC* for the controller behavior with a reachability analysis on the plant model. First, we abstract away the plant dynamics and perform BMC on the discrete control program. In particular, the sensor data received during each cycle remain unspecified in this discrete analysis. Based on this coarse abstraction, the algorithm enters a refinement loop: Each discrete counterexample is analyzed for spuriousness by performing a reachability analysis on a hybrid plant model whose discrete behavior is governed by the counterexample path.

If the hybrid reachability analysis refutes the discrete counterexample as being spurious, it generates an *explanation* for the refutation which is used to refine the over-approximation of the discrete control model and to exclude the counterexam-

ple from the BMC search. Otherwise, if the hybrid analysis confirms the reachability of an unsafe state along the discrete counterexample, the analysis stops and reports a possible safety violation (again, due to over-approximative computations, we cannot disprove safety). Finally, if no safety-violating paths are left in the abstraction, the method reports (bounded) safety.

The main advantage of this method is that it first analyzes the behavior of the discrete controller employing highly efficient solver technologies for BMC, and allows thereby to *restrict the computationally expensive reachability analysis in the hybrid model to certain potentially safety-critical paths*, found by BMC. Therefore, this method is dedicated to systems with complex controller components but rather simple dynamic structure.

The second CEGAR-based analysis approach maintains a single *hybrid abstraction* modeling both the controller and the controlled plant. Our goal is to consider only safety-relevant parts of the complex system dynamics in the verification process. Starting from an initial abstraction which models the SFC control program but allows arbitrary plant dynamics, we apply *reachability analysis* to check the abstract model for safety. Each time when a potential counterexample path leading to an unsafe state is detected, we *refine* our abstract model step-wise by adding some pieces of information about the dynamics along the abstract counterexample path.

As both the abstraction and the reachability analysis are over-approximative, this tool can report safety (when the abstraction is fine enough to detect that a set of safety-critical states is not reachable), but is not able to prove that a system is unsafe (if unsafe states are found to be reachable along fully concretized paths, we do not know whether it happened due to the over-approximative computations).

This method is dedicated to the reachability analysis of PLC-controlled plants with rather simple discrete PLC programs but complex dynamic behavior. The main reason for the applicability of this method is that we do not start the reachability analysis procedure anew after each refinement step. Instead, we closely *embed the refinement into the reachability analysis procedure*. Thereby we prevent the algorithm from repeatedly re-checking the same model behavior.

As mentioned above, our models obtain some special properties which are not yet supported by most of the existing analysis tools. The main issue is the extensive use of *discrete variables*. If we have to model them as continuous variables, our models are high-dimensional and as a consequence, the analysis is expensive. Another need is the support of *urgency*. Due to the cyclic execution of the control programs on a PLC at the beginning of each cycle, the input is read from the plant and at the end of a cycle, the computed control is sent back to the plant. However, we assume that these actions are performed instantaneously and do not need any time. Thus we can model these actions as being urgent, i. e. we force that some transitions have to be taken immediately if they become enabled or that a location is left immediately without any time elapse. The result is a more accurate computation of the reachable states since we avoid unnecessary bloating operations in urgent locations.

The authors of the analysis tool HYPRO extended their tool by the support of *discrete variables* and *timed transitions* to model urgency. We could show that the

running time of the analysis could be reduced by both features. However, the impact of discrete variables is stronger than the impact of timed transitions. Furthermore, we could demonstrate on some benchmarks that the accuracy of the reachable state set is enhanced, i. e. we have less over-approximation if we make use of timed transitions.

Outline Part I of this thesis is organized as follows. In Chapter 2 and Chapter 3 the related work and the preliminaries for the analysis of automated control in chemical plants are presented. Afterwards, we introduce our modeling approach from [1, 3, 4, 5] in Chapter 4. Chapter 5 is dedicated to our approaches for the generation and validation of counterexamples for hybrid reachability analysis [2], which is fundamental for our two CEGAR-based analysis approaches [3, 4, 5] that are presented in Chapter 6 and Chapter 7. Since both analysis approaches are time consuming, we examine in Chapter 8, how special characteristics of our models can be exploited to enhance the running time of the reachability analysis. These results have not been published yet. Finally, we present the experimental results for our analysis approaches in Chapter 9 and conclude the first part of this thesis in Chapter 10. A lists of the acronyms, variables, and constants that are used in this part are given in the glossary in the back matter of this thesis.

2. Related work

In this chapter, we give an overview of previous work that is related to the analysis of automated control in chemical plants.

Outline In the first part of this thesis, we focus on the reachability analysis for hybrid automata. Related work on reachability analysis, different state set representations, and existing analysis tools for hybrid systems is given in Section 2.1. We will present previous work on bounded model checking (BMC) in Section 2.2, which has originally been introduced as an analysis approach for discrete systems. However, it has already been extended to hybrid systems as well. Some basic work on counterexample-guided abstraction refinement (CEGAR)-based analysis is presented in Section 2.3, before we present some previous work on the analysis of sequential function chart (SFC)-controlled plants in Section 2.4.

2.1. Reachability Analysis of Hybrid Automata

In the last two decades, a wide range of reachability analysis techniques and software tools have been developed for hybrid systems. In this thesis we focus on reachability analysis techniques for continuous-time hybrid automata that apply a fixed-point-based forward-reachability iteration [ACH+95]. Starting from a set of initial states, the system trajectories result from alternating time evolution (*flowpipes*) and discrete steps (*jumps*). Since the reachability problem for hybrid systems is in general undecidable, most of the techniques for reachability analysis compute *over-approximations* of the set of reachable states of hybrid systems. As a consequence, they can prove safety if a set of unsafe states is not reachable, but they cannot disprove safety.

Fixed-point-based forward-reachability iteration algorithms need two main ingredients: (a) A technique to *represent* state sets and to compute certain operations on them like union, intersection, Minkowski sum, etc. and (b) a method to compute *one-step-successors* of state sets both for continuous flows and discrete jumps. The choice of the representation is crucial, as it strongly influences the *approximation error* and the *efficiency* of the computations.

A *flowpipe* is an over-approximation of the states that are reachable from a given initial set of states by letting time progress within a certain maximal time horizon. To compute a flowpipe, the maximal time horizon is often divided into smaller intervals and the flowpipe is represented as a (finite) union of state sets (flowpipe segments), each covering one of the smaller intervals [CK98].

The variety of analysis techniques for hybrid systems is complemented by approximation techniques that increase the applicability of hybrid systems verification. Examples are hybridization (e. g. [ADG07, TD13]), linearizion (e. g. [ASB08]), and other abstraction methods (e. g. [ADI02, GP07b]).

2.1.1. State Set Representation

Popular approaches use for the state set representation either geometric objects like hyperrectangles [SK03], convex polyhedra [CÁ11, CK98], zonotopes [Küh98, Gir05], orthogonal polyhedra [BMP99], or ellipsoids [KV02]. Others use symbolic representations like support functions [Le 09, LG09] or Taylor models [CÁS12, CBGV12]. In [MT00], the state sets are over-approximated by level sets.

To mention some analysis tools, HyTech [HHWT97], PHAVer [Fre08], and the Multi-Parametric Toolbox [HKJM13] use convex polyhedra for the over-approximative representation of state sets, SpaceEx [FLD+11] additionally allows the usage of support functions. The analysis tool d/dt [ADM02] uses grid paving as over-approximations. MATISSE [GP07b] over-approximates state sets by zonotopes. The MATLAB Ellipsoidal Toolbox [KV06] supports the over-approximative representation of sets by ellipsoids, Flow* by Taylor models. In Ariadne [BCC+06], the state sets are over-approximated by Taylor models or grid pavings.

2.1.2. Tools

Some of the most popular *flowpipe-construction-based* tools for the reachability analysis of hybrid automata are Cora [AD14], d/dt [ADM02], the MATLAB Ellipsoidal Toolbox [KV06], Flow* [CÁS13], HyCreate [HyC], HyReach [HyR], HySon [BCM13], SoapBox [HMR14], and SpaceEx [Fre08, FLD+11].

In contrast to the above approaches, logical formalizations are used in *theorem-proving-based* analysis tools like KeYmaera [PQ08], Ariadne [BCC+06], or the Isabelle/HOL-based tool [Imm15]. Other analysis tools like dReach [KGCC15], HSolver [RS05], and HySAT [FH06] with its successor iSAT [SKB13] and its extension iSAT-ODE [Egg14] also use logical characterizations but in combination with *interval arithmetic and satisfiability-modulo-theories (SMT) solving*.

There are further approaches in use for the reachability analysis of hybrid systems. Just to mention a few, the tool C2E2 [DMVP15] uses validated numerical simulation, Bernstein expansion is implemented in NLTOOLBOX [TD13], and level sets were used in [MT00].

Among the flowpipe-construction-based analysis tools, Flow* supports the analysis of non-linear hybrid automata (with non-linear differential equations). Also the tools based on theorem proving (Ariadne, Isabelle/HOL, KeYmaera), some of the SMT-based tools (dReach, HSolver, iSAT-ODE), C2E2, and NLTOOLBOX can solve non-linear differential equations.

In this thesis, we will concentrate on flowpipe-construction-based analysis tools.

2.1.3. CEGAR-based Analysis

When we started our work on [2], the generation of counterexamples was not supported by any of the above mentioned flowpipe-construction-based tools for the reachability analysis of hybrid automata. There are some works [PDMV13, DM11] related to counterexample generation for hybrid systems, but they are mostly devoted to CEGAR approaches for restricted classes of hybrid automata like, e. g. (initialized) rectangular automata.

2.2. Bounded Model Checking

To check bounded reachability, some of the above mentioned tools for the reachability analysis of hybrid systems apply *BMC*, originally developed for discrete systems [Bie09, BCCZ99, CK03, CBRZ01], but also extended to hybrid systems [ÁBKS05, GPB05]. BMC requires a logical characterization of the initial states and the transition relation of the model, and a set of target states. The method generates logical formulas φ_k^{BMC} for increasing values of k, such that φ_k^{BMC} is satisfiable if and only if there is a system trajectory of length k that reaches a target state. Depending on the underlying logic, a suitable propositional satisfiability (SAT) or satisfiability-modulo-theories (SMT) solver can be used to check the generated formulas for satisfiability, thereby deciding bounded reachability of the target set.

2.3. CEGAR

Originating in the successful application of abstraction and *CEGAR* techniques to discrete models [CGJ+00], also several methods have been proposed to apply CEGAR to hybrid automata [ADI03, CFH+03a, CFH+03b]. The work [FCJK05] extends the research on CEGAR for hybrid automata by restricting the analysis to fragments of counterexamples. Other works [JKWC07, PDMV13, RPV16] are restricted to the classes of rectangular, linear, or affine hybrid automata. Linear programming for the abstraction refinement is used in [JKWC07], whereas rectangular automata that over-approximate the affine dynamics are used in [RPV16] for the abstraction refinement. However, none of the above approaches exploits the special properties of hybrid models for chemical plant control.

The work in [DKL07] proposes a CEGAR verification approach for programmable logic controller (PLC) programs using timed automata. Starting with the coarsest abstraction, the model is refined with variables and clocks. However, this work does not consider the dynamic plant behavior.

A CEGAR approach on step-discrete hybrid models is suggested in [Seg07], where system models are verified by learning reasons for spurious counterexamples and excluding them from further search. However, this method was originally designed for control-dominated models with little dynamic behavior.

A CEGAR-based abstraction technique for the verification of safety properties of PLC-controlled plants was published in [ELS05]. Given a hybrid automaton model of the controlled plant, the method abstracts away from parts of the continuous dynamics. However, instead of refining the dynamics in the hybrid model to exclude spurious counterexamples, their method adds information about enabled and disabled transitions.

2.4. Verification of SFC-Controlled Plants

In automation, *programmable logic controllers* are widely used to control the behavior of plants. The industry standard IEC 61131-3 [Int03] specifies several languages for the programming of PLCs. In this work, we consider the graphical language of *SFCs*.

Since PLC-controlled plants are often safety-critical, SFC *verification* has been extensively studied [FL00]. Several approaches exist which consider either an SFC in isolation or the combination of an SFC with a model of the plant [BCMP98, HKD98]. As the combination of the controller and the plant has a mixed discrete-continuous behavior, the latter approaches usually define a timed or hybrid automaton that specifies the SFC controller, and a hybrid automaton that specifies the plant. Building the composition of these two models gives a hybrid automaton model of the controlled plant that can be analyzed using an existing tool for the reachability analysis of hybrid systems.

3. Preliminaries

In this chapter, we introduce notations that are used in this thesis and the basic knowledge that is needed to understand the work that is presented in this first part of the thesis.

Outline We introduce our notation for numbers, formulas, and predicates in Section 3.1. Afterwards, we present chemical plants and and their controllers in Section 3.2 and Section 3.3. We present sequential function charts as a programming language to control a plant and explain the cyclic execution of a controller on a programmable logic controller. After some background information on our application scenarios, we switch to formal analysis and present hybrid automata and the reachability analysis for hybrid automata in Section 3.4.1. With bounded model checking, a second analysis approach is presented in Section 3.5, that we will use later for an efficient analysis of discrete systems. The text from this section about bounded model checking is not my contribution but has been written by Martin Neuhäußer in our joint publication [5]. We conclude this chapter with some details on counterexample-guided abstraction refinement which allows to keep the model sizes moderate during the analysis of a given property.

3.1. Numbers, Formulas, and Predicates

In this thesis, the set of Boolean values and the sets of natural (including 0), integer, and real numbers are denoted by $\mathbb{B}$, $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}$, respectively. $\mathbb{R}_{\geq 0}$ is the set of non-negative reals and $\mathbb{N}_{>0} := \mathbb{N} \setminus \{0\}$ is the set of natural numbers excluding 0.

For some $n \in \mathbb{N}_{>0}$, let $V = \{x_1, \ldots, x_n\}$ be an ordered set of variables over $\mathbb{R}$. We also use the vector notation $x = (x_1, \ldots, x_n)$, and denote by V' and $\dot{V}$ the renamed variable sets $\{x'_1, \ldots, x'_n\}$ and $\{\dot{x}_1, \ldots, \dot{x}_n\}$, respectively.

We will use quantifier-free real-arithmetic formulas (sometimes also called predicates) over V with their standard syntax and semantics. Especially, the truth value of such a formula φ over V under a given variable valuation $v \in \mathbb{R}^n$ is determined by evaluating the result of the substitution $\varphi\,[v/x]$ of the values v for the variables x. Given a real-arithmetic formula φ over V, its satisfaction set is $[\![\varphi]\!] = \{v \in \mathbb{R}^n \mid \varphi\,[v/x] = \mathit{true}\}$. A formula is *linear* if it is a quantifier-free linear real-arithmetic formula. We call φ *convex* if $[\![\varphi]\!]$ is convex. By $\Phi(V)$ we denote the set of linear predicates over V. The set of linear convex predicates over the set of variables V is denoted by $\Phi_\zeta(V)$.

3.2. Chemical Plants

In this part of the thesis, we focus on the verification of *chemical plants*. A *chemical plant* is a physical system that consists of different units which contain some material. The units are interconnected by piping. *Actuators* can influence the state of the material, e. g., cause the movement of material along the pipes between the units. *Sensors* allow some observations about the state of the plant.

Example 3.2.1

A simple example plant that we use as a running example is depicted in Figure 3.1. It consists of two cylindrical tanks U_1 and U_2 that are connected by pipes in a closed system. The tanks have equal diameters but different heights F_1, F_2, respectively. The variables h_1 and h_2 denote the water level in U_1 and U_2, respectively. Each tank U_i is equipped with two sensors $\texttt{low}_i$ at height L_i and $\texttt{high}_i$ at height H_i ($0 < L_i < H_i < F_i$) that detect low and high water levels, respectively. The sensors $\texttt{full}_i$ and $\texttt{empty}_i$ are logical annotations that indicate a tank being full ($h_i = F_i$) or empty ($h_i = 0$) and are used for the BMC methodology only.
The water flow into and out of tank i is controlled by the valves $\texttt{V}_i^{in}$ and $\texttt{V}_i^{out}$. If $\texttt{V}_i^{in}$ is open, water can flow into U_i and $\texttt{V}_i^{out}$ controls the flow out of tank U_i analogously. Moreover, the plant is equipped with two pumps $\texttt{P}_1$ and $\texttt{P}_2$ which can pump water when the adjacent valves are open. $\texttt{P}_1$ pumps water from U_1 to U_2, decreasing h_1 and increasing h_2 by k_1 per time unit. $\texttt{P}_2$ pumps water through a second pipeline in the other direction, causing a height increase of k_2 per time unit for h_1 and a height decrease of k_2 per time unit for h_2.
The pumps are manually controlled by the operator panel which allows to switch the pumps on ($\texttt{P}_i^+$) or off ($\texttt{P}_i^-$). The control receives this input request along with the state of the pumps and the sensor values, and computes some output values. These are sent to the environment and cause actuators to be controlled accordingly, by switching the pumps on or off. The pumps are coupled with the adjacent valves $\texttt{V}_2^{in}/\texttt{V}_1^{out}$, $\texttt{V}_1^{in}/\texttt{V}_2^{out}$, which will automatically be opened or closed when the pump is switched on or off, respectively.
The control program prevents the pumps from working if they cannot pump; thus a pump should be switched off before the tank to which it pumps is full (maintaining $h_i < F_i, i = 1, 2$) or before the tank from which it pumps water is dry (maintaining $h_i > 0$). If the water level of a tank U_i reaches the position H_i of the $\texttt{high}_i$ sensor, the pump that fills this tank is switched off and the connected valves are closed. Additionally, the pump $\texttt{P}_i$ is switched on and the connected valves are opened. Analogously, if the water level of tank U_i decreases below the position L_i of the $\texttt{low}_i$ sensor, the pump $\texttt{P}_i$ is switched off and the connected valves are closed. The other pump that fills U_i with water is switched on and the connected valves are opened. For simplicity, we neglect the valves in the following.

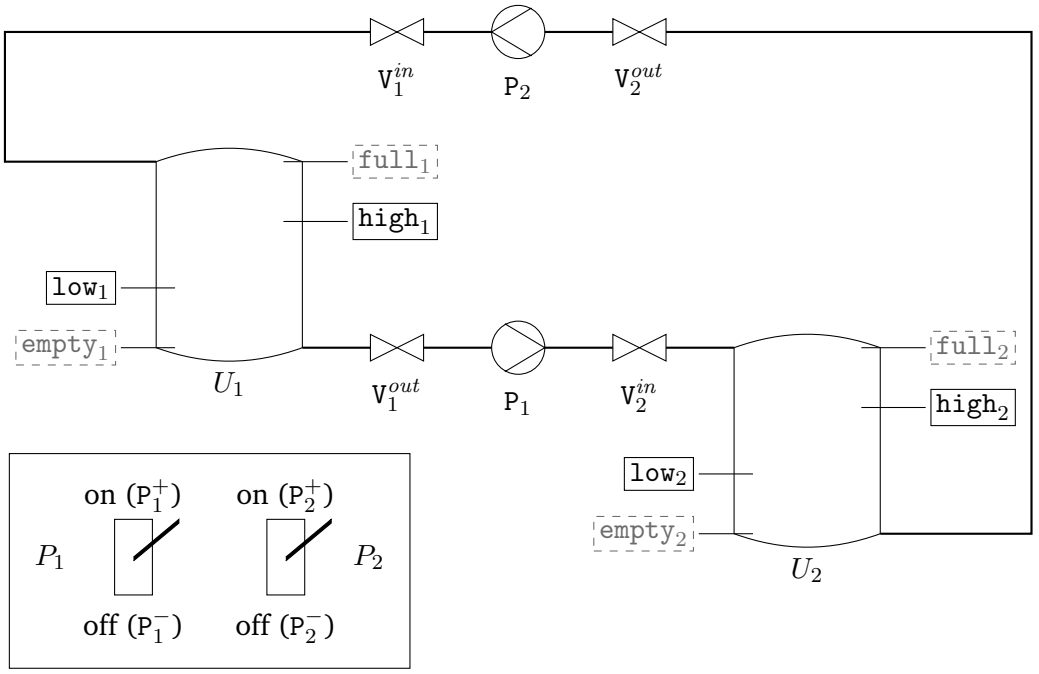


Figure 3.1.: An example plant and its operator panel.

Figure 3.3 on page 18 illustrates the plant, the PLC, and the interface between them; additionally, it shows which variable sets are used in our models to describe those components and the communication between them. For the plant, the current states of the actuators and sensors are described by the variable sets V_{act} and V_{sen}, respectively. The states of the physical quantities are specified by the values of the variables from a set V_{cont}; these values evolve continuously over time. We assume the variable sets to be ordered and use also vector notation to list their elements. The plant variables are thus given by the set $V_{plant} := V_{act} \cup V_{sen} \cup V_{cont}$.

Example 3.2.2

For the example plant in Figure 3.1, we overload the meaning of the sensors $\mathtt{low}_i$ and $\mathtt{high}_i$ to also represent the *state* of the sensor. $\mathtt{low}_i \equiv (h_i \geq L_i)$ and $\mathtt{high}_i \equiv (h_i \geq H_i)$. Similarly, we overload the meaning of $\mathtt{P}_i$ and use it also to represent the state of pump i ($\mathtt{P}_i$ for the ith pump being on, and $\neg\mathtt{P}_i$ for the ith pump being off). Analogously for the operator panel, we use $\mathtt{P}_i^+$ and $\mathtt{P}_i^-$ to denote the states of the control panel buttons. Thus we use the following variables in our encodings:

- $V_{act} = (V_{act}^1, V_{act}^2)$ with $V_{act}^i = (\mathtt{P}_i^+, \mathtt{P}_i^-, \mathtt{P}_i)$ are the variables storing the states of the actuators. The control panel requests $\mathtt{P}_i^+$ and $\mathtt{P}_i^-$ encode user requests to switch a pump on ($\mathtt{P}_i^+ = 1$) or off ($\mathtt{P}_i^- = 1$). If no user

input is received, we have $\mathtt{P}_i^+ = \mathtt{P}_i^- = 0$. The states $\mathtt{P}_i$ of the pumps are given by the values 0 (off) or 1 (on), respectively;

- $V_{sen} = (V_{sen}^1, V_{sen}^2)$ with $V_{sen}^i = (\mathtt{low}_i, \mathtt{high}_i)$ model the values of the sensors at L_i respectively H_i;
- for the values of the physical quantities, the variables $V_{cont} = (V_{cont}^1, V_{cont}^2)$ with $V_{cont}^i = (h_i)$ are used to model the water levels in the tanks.

In this example, the actuators and sensors are assumed to be purely discrete. In general, the states of actuators and sensors could also change continuously, e. g. when slowly opening a valve or when continuously sensing temperature values.

3.3. Sequential Function Charts (SFCs)

To specify controllers, we use the graphical programming language *sequential function charts (SFCs)* as given by the industry norm IEC 61131-3 [Int03]. We use the formal semantics for SFCs as specified in [1]. This semantics is based on [BHLE04, Bau04, Luk05] with slight adaptations to a certain PLC application.

Example 3.3.1

Figure 3.2 shows a possible control program for the example plant. We specify only the control of pump $\mathtt{P}_1$ which runs in parallel with an analogous SFC for the second pump $\mathtt{P}_2$. The components of this SFC are described in the following.

SFC execution is based on a main loop. It starts with getting the current state of the sensors and actuators as input from the plant. After the execution of some computations, the output of the SFC is sent back to the plant to control the actuators. Accordingly, an SFC has a finite set $V_\mathcal{C} = V_{in} \cup V_{loc} \cup V_{out}$ of typed *variables* which is a (not necessarily disjoint) union of sets of input variables V_{in}, local variables V_{loc}, and output variables V_{out}. The input variables serve to store the current state of the sensors and actuators, local variables are used in computations, and the output variables are used to determine the new actuator states.

The IEC norm supports – among others – the elementary data types integer, real, Boolean, string, time, and date. Let D be the union of the data type domains of all variables $v \in V_\mathcal{C}$. A *valuation* $\nu \in \mathcal{V}^{V_\mathcal{C}}$ is a function $\nu : V_\mathcal{C} \to D$ that assigns to each variable $v \in V_\mathcal{C}$ a value from its domain. Given a valuation ν, a predicate from the set $\Phi(V_\mathcal{C})$ evaluates to *true* (1) or *false* (0) in ν.

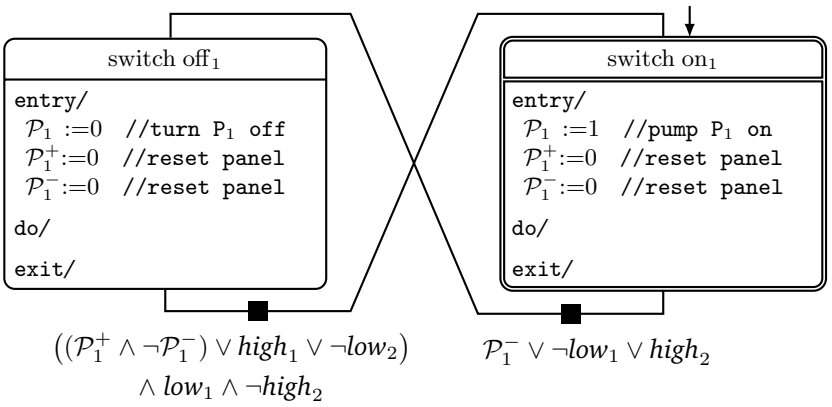


Figure 3.2.: SFC for pump $\mathtt{P}_1$.

Example 3.3.2

The controller from Example 3.2.1 consists of two SFCs, one for each pump (see Figure 3.2 for pump $\mathtt{P}_1$). The specification of this controller uses the following variables:

$$\begin{aligned} V_{in} &= (\mathcal{P}_1^+, \mathcal{P}_1^-, \mathcal{P}_1, low_1, high_1, \\ &\qquad \mathcal{P}_2^+, \mathcal{P}_2^-, \mathcal{P}_2, low_2, high_2) \\ V_{loc} &= () \\ V_{out} &= (\mathcal{P}_1^+, \mathcal{P}_1^-, \mathcal{P}_1, \mathcal{P}_2^+, \mathcal{P}_2^-, \mathcal{P}_2)\,. \end{aligned}$$

The Boolean input variables $\mathcal{P}_i^+$ and $\mathcal{P}_i^-$ store the states of the control panel buttons $\mathtt{P}_i^+$ and $\mathtt{P}_i^-$ for the user requests to switch pump $\mathtt{P}_i$ on or off, respectively (0 for not pressed, 1 for pressed); they are also used as output variables to reset the panel state when a request has been served. The Boolean variables $\mathcal{P}_i$ are used as input variables to read the pump states as well as output variables to control the new states of the pumps. The value of the Boolean input variable low_i ($high_i$) is true if the sensor $\mathtt{low}_i$ ($\mathtt{high}_i$) senses water, i. e., if the water level in tank i is at least L_i (H_i).

The control is specified using a finite set of *steps* and *guarded transitions* between them, connecting the bottom of a source step with the top of a target step. A distinguished initial step is *active* at start. A transition is *enabled* if its source step is active and its transition guard which is a predicate over the SFC variables V_C evaluates to *true* for the current variable valuation; taking an enabled transition moves the activation from its source to its target step. Apart from transitions that connect single steps also parallel branching can be specified by defining sets of source/target steps.

A partial order on the transitions defines *priorities* for concurrently enabled tran-

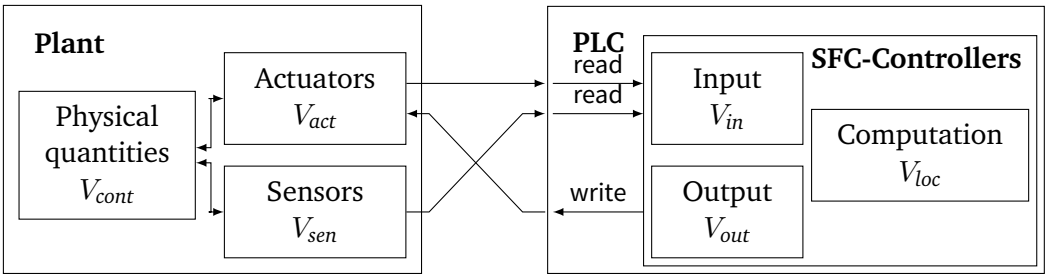


Figure 3.3.: The interface of a plant and a PLC.

sitions that have a common source step. For each step, the enabled transition with the highest priority is taken. Transitions are *urgent*, i. e., a step is active only as long as no outgoing transition is enabled.

Example 3.3.3

The example SFC in Figure 3.2 has two steps named switch off_1 and switch on_1, where switch on_1 is active initially. The activity moves from switch on_1 to switch off_1 as soon as the guard $\mathcal{P}_1^- \vee \neg \mathit{low}_1 \vee \mathit{high}_2$ of the corresponding transition is true, i. e., if the user requests to turn $\mathtt{P}_1$ off, if the water level in the first tank falls below L_1, or if the water level in the second tank reaches H_2.

Each step contains a set of prioritized *action blocks* specifying the actions that are performed during the activation period of the step. An action block $b = (q, a)$ is a tuple of an *action qualifier* q and an *action* a. The set of all action blocks using actions from the set $\mathcal{A}$ is denoted by $B_{\mathcal{A}}$.

The action qualifier $q \in \{\texttt{entry}, \texttt{do}, \texttt{exit}\}$[1] specifies when the corresponding action is performed. When control enters a step, its `entry` and `do` actions are executed once. As long as the step is active, its `do` actions are executed repeatedly. The `exit` actions are executed only once upon deactivation.

An action a is either a variable assignment or an SFC. Executing a variable assignment changes the value of a variable. Executing an SFC means activating it and thus performing the actions of its active step. For completeness, we introduce the *history* flag of an SFC. It determines which step becomes active when the SFC is activated. If the flag is set to *false*, the initial step of the nested SFC becomes active. Otherwise its last active step is re-activated. In our analysis approaches, we assume the history flags of all SFCs to be set to *false*. However, we could extend our system models to support arbitrary history flags for nested SFCs.

[1] In the IEC standard, the qualifiers $P1, N$, and $P0$ are used instead of `entry`, `do`, and `exit`. The remaining qualifiers of the industry standard are not considered in this thesis.

Example 3.3.4

The step switch off_1 has three action blocks; we assume them to be listed in decreasing priority order. The first action block $(\texttt{entry}, \mathcal{P}_1{:=}0)$ sets the value of the output variable $\mathcal{P}_1$ to 0 each time the activity enters the step switch off_1. When the activity moves from switch off_1 to switch on_1, as the steps have neither `do` nor `exit` actions, only the `entry` actions of switch on_1 are executed.

The formal syntax of SFCs is specified as follows:

Definition 3.3.1 – SFC Syntax

An *SFC* is a tuple $(V_\mathcal{C}, \mathcal{S}, \mathcal{A}, s_0, \mathit{Trans}, \mathcal{B}, \sqsubset, \prec, \mathit{Hist})$, where

- $V_\mathcal{C} = \{x_1, \ldots, x_n\} = V_{in} \cup V_{loc} \cup V_{out}$ is a finite ordered set of variables which is the union of the finite sets of *input*, *local*, and *output variables*; A *valuation* $\nu \in \mathcal{V}$ with $\nu : V_\mathcal{C} \to \mathbb{R}$ assigns a value to each variable $x_i \in V_\mathcal{C}$.
- $\mathcal{S}$ is a finite set of *steps*;
- $\mathcal{A}$ is a finite set of *actions* which are either variable assignments referring to variables from $V_\mathcal{C}$, or SFCs whose variable and action sets are subsets of $V_\mathcal{C}$ and $\mathcal{A}$, respectively, and whose action order is a subset of $\sqsubset$;
- $s_0 \in \mathcal{S}$ is the *initial step*;
- $\mathit{Trans} \subseteq (2^\mathcal{S} \setminus \{\emptyset\}) \times \Phi(V_\mathcal{C}) \times (2^\mathcal{S} \setminus \{\emptyset\})$ is a finite set of guarded *transitions*;
- $\mathcal{B} : \mathcal{S} \to 2^{B_\mathcal{A}}$ is a function which assigns a set of *action blocks* to each step;
- $\sqsubset \subseteq \mathcal{A} \times \mathcal{A}$ is a *total order on the actions*;
- $\prec \subseteq \mathit{Trans} \times \mathit{Trans}$ is a *partial order on the transitions*; for all transitions with a common source step, this order is defined;
- $\mathit{Hist} \in \{0, 1\}$ is the *history* flag ($\mathit{Hist} = 1$: SFC with history, $\mathit{Hist} = 0$: SFC without history).

Given a set $\mathit{Trans}' \subseteq \mathit{Trans}$ of transitions, we use the functions $\texttt{source}(\mathit{Trans}')$ and $\texttt{target}(\mathit{Trans}')$ to denote the union of all source and target steps, respectively, of transitions in Trans'. We use $\overline{\mathcal{C}}$ for the set containing the top level SFC $\mathcal{C}$ and its nested SFCs at all depths, $\mathcal{S}(\overline{\mathcal{C}})$ for the union of all steps of the SFCs in $\overline{\mathcal{C}}$, and $\mathit{Trans}^{\overline{\mathcal{C}}}$ for the union of all transitions of the SFCs in $\overline{\mathcal{C}}$.

Let $\mathcal{C} = (V_\mathcal{C}, \mathcal{S}, \mathcal{A}, s_0, \mathit{Trans}, \mathcal{B}, \sqsubset, \prec, \mathit{Hist})$ be an SFC. Its execution on a PLC performs the following steps in a cyclic way:

1. Get the input data from the environment and update the values of the input variables accordingly.
2. Collect the set of transitions that have to be taken and execute them.
3. Determine the set of actions that have to be performed and execute them in priority order.
4. Send the output data (the values of the output variables) to the environment.

Items 1. and 4. of the PLC cycle implement the communication with the environment, e. g. with plant sensors and actuators, whereas 2. and 3. execute the control. The interface for the communication of an SFC with the environment is shown in Figure 3.3.

Each PLC cycle takes δ time units, which we assume to be constant for all cycles. However, our approach could be extended to varying cycles times. After items 1. – 3. are executed, the control waits until δ time units have passed, before executing item 4. We assume that δ is specified such that items 1. – 3. can always be executed within this time limit; in real systems, the execution is interrupted if it is not completed within δ time units.

In order to specify the control, we need to define *configurations* of SFCs. A *configuration* ρ of an SFC $\mathcal{C}$ is a tuple $(\nu, \mathcal{S}_r, \mathcal{S}_a, \mathcal{A}_a)$. It stores the valuation $\nu \in \mathcal{V}^{Vc}$ of $\mathcal{C}$ and the following elements that are formalized in Definition 3.3.2:

- The set $\mathcal{S}_r \subseteq \mathcal{S}(\overline{\mathcal{C}})$ of *ready steps* contains all active steps of the top-level and the nested SFCs. Moreover, the last active steps of all currently inactive nested SFCs with history belong to $\mathcal{S}_r$. We say that control resides in these steps.
- The set $\mathcal{S}_a \subseteq \mathcal{S}(\overline{\mathcal{C}})$ contains the *active steps* of the top-level SFC and the ready steps of those nested SFCs to which an active action points.
- *Active actions* are the `do` actions of active steps and the `exit/entry` actions of the source/target steps of the taken transitions. The sequence $\mathcal{A}_a$ is the list of the active actions sorted according to decreasing action priorities. It specifies the actions that are executed in the next PLC cycle.

The set *Conf* denotes the set of all SFC configurations. The initial configuration ρ_0 of an SFC is $(\nu_0, \mathcal{S}_r^0, \mathcal{S}_a^0, \mathcal{A}_a^0)$. The initial valuation ν_0 assigns the initial values to the variables. The default value for Booleans is *false*, whereas numerical variables are set to zero if no initial value is specified. The initial set of ready steps $\mathcal{S}_r^0 := \{s_0 \mid \exists \mathcal{C}_i \in \overline{\mathcal{C}} \ . \ s_0 \text{ is the initial valuation of } \mathcal{C}_i\}$ contains the initial steps of all SFCs in $\overline{\mathcal{C}}$. The initial active steps and actions are empty, i. e., $\mathcal{S}_a^0 = \emptyset$ and $\mathcal{A}_a^0 = \emptyset$.

We define the set $\mathcal{E}$ of enabled transitions and the set $\mathcal{T}$ of taken transitions for an SFC $\mathcal{C}$ and a configuration $\rho = (\nu, \mathcal{S}_r, \mathcal{S}_a, \mathcal{A}_a)$ of $\mathcal{C}$ as follows:

$$\mathcal{E}(\mathcal{C}, \rho) = \{(S, g, S') \in \mathit{Trans}_{\overline{\mathcal{C}}} \mid S \subseteq \mathcal{S}_a \wedge \nu \vDash g\}$$

$$\mathcal{T}(\mathcal{C}, \rho) = \{d = (S, g, S') \in \mathcal{E}(\mathcal{C}, \rho) \mid \forall d_1 = (S_1, g_1, S_1') \neq d \in \mathcal{E}(\mathcal{C}, \rho) \\ (S \cap S_1 = \emptyset \vee d_1 \prec d)\}$$

We define the operational semantics of SFCs by configuration changes during PLC cycles. In this thesis, we do not formalize the communication with the environment (steps 1. and 4. of the PLC cycle), but focus on the computation steps in between: the computation starts with a configuration $(\nu, \mathcal{S}_r, \mathcal{S}_a, \mathcal{A}_a)$ where ν has already been updated with the new input data from the environment. We show how to compute the configuration $(\nu', \mathcal{S}'_r, \mathcal{S}'_a, \mathcal{A}'_a)$ that is obtained after the execution of the second and the third step of the PLC cycle. The output data that is sent to the environment in the last step of the PLC cycle can be extracted from the updated valuation ν'.

If we remove the source steps of all taken transitions and add the target steps of those transitions afterwards, we obtain the new set of ready states. The new sets of active steps and active actions are computed recursively by Algorithm 1. It starts to compute the active sets for the top-level SFC and then recursively adds the active sets for each nested active SFC. Afterwards, the active actions are sorted according to the given action order and executed in the computed order.

The formal semantics for SFCs is given below:

Definition 3.3.2 – SFC Semantics

The semantics of an SFC $\mathcal{C}$ is defined by the relation of configuration changes $\rightarrow_{Conf} \subseteq Conf \times Conf$ with $\rho \rightarrow_{Conf} \rho'$ for $\rho = (\nu, \mathcal{S}_r, \mathcal{S}_a, \mathcal{A}_a)$ and $\rho' = (\nu', \mathcal{S}'_r, \mathcal{S}'_a, \mathcal{A}'_a)$ if and only if

- $\mathcal{S}'_r = (\mathcal{S}_r \setminus \texttt{source}(\mathcal{T}(\mathcal{C}, \rho))) \cup \texttt{target}(\mathcal{T}(\mathcal{C}, \rho))$,
- $(\mathcal{S}'_a, \text{unsorted}\mathcal{A}'_a) = \text{computeActiveSets}(\mathcal{S}'_r, \emptyset, \emptyset, \mathcal{C}, \rho, \mathcal{A}_a \cap \overline{\mathcal{C}})$,
- $\mathcal{A}'_a = \mathit{sort}(\text{unsorted}\mathcal{A}'_a, \sqsubset)$, and
- $\nu' = (a_1 \circ \ldots \circ a_m)(\nu)$ where $\mathcal{A}'_a = a_m \circ \ldots \circ a_1$.

Algorithm 1 shows how the active steps and actions are computed. Note that the algorithm is recursively called for each active nested SFC with the already computed active steps and actions.

3.4. Hybrid Automata (HA)

A popular modeling language for systems with mixed discrete-continuous behavior are *hybrid automata (HA)*. They are based on discrete transition systems, which consist of a set of *locations (control modi)*, a set of *variables*, and a set of *discrete transitions (jumps)*. The current location and the current variable values specify the current state. Discrete state changes can happen if the enabling condition (*guard*) of a jump is true in the current state. The jump might change the current location as well as the values of the variables according to a *reset function*.

Hybrid automata extend these models with a dynamic continuous behavior. While the control stays in a location, *time steps* (*flows*) let the values of the variables evolve

Algorithm 1:
computeActiveSets($\mathcal{S}'_r, \mathcal{S}_a, \mathcal{A}_a, \mathcal{C}, \rho, SFC_a$)

let $\mathcal{C} = (V_\mathcal{C}, \mathcal{S}, \mathcal{A}, s_0, \mathit{Trans}, \mathcal{B}, \sqsubset, \prec, \mathit{Hist})$
`/* Add local active steps of` $\mathcal{C}$ `to` $\mathcal{S}'_a$`. */`
if $\mathit{Hist} = 1 \vee \mathcal{C} \in SFC_a$ **then**
    $\mathcal{S}'_a := \mathcal{S}_a \cup (\mathcal{S} \cap \mathcal{S}'_r)$
else
    $\mathcal{S}'_a := \mathcal{S}_a \cup \{s_0\}$
`/* Collect local active actions and their qualifiers. */`
$\mathcal{A}'_a := \mathcal{A}_a$
foreach $s \in \mathcal{S}, b = (q, a) \in \mathcal{B}(s)$ **do**
    if $(q{=}\texttt{exit} \wedge s{\in}\texttt{source}(\mathcal{T}(\mathcal{C}, \rho))) \vee (q{=}\texttt{entry} \wedge s{\in}\texttt{target}(\mathcal{T}(C, \rho))) \vee (q{=}\texttt{do} \wedge s{\in}\mathcal{S}'_a)$ **then**
        $\mathcal{A}'_a := \mathcal{A}'_a \circ a$
`/* Compute` $\mathcal{A}'_a$ `and` $\mathcal{S}'_a$ `for each active nested SFC */`
foreach $s \in \mathcal{S} \cap \mathcal{S}'_a, b = (q, a) \in \mathcal{B}(s)$ **do**
    if $a \in \overline{\mathcal{C}}$ **then**
        $(\mathcal{S}'_a, \mathcal{A}'_a) := \text{computeActiveSets}(\mathcal{S}'_r, \mathcal{S}'_a, \mathcal{A}'_a, a, \rho, SFC_a)$
return $(\mathcal{S}'_a, \mathcal{A}'_a)$

continuously according to some ordinary differential equations (ODEs). *Invariants* are used to restrict the time evolution and force the control to change the modus before the invariants get violated.

Extending the original definition of hybrid automata without changing the expressivity, we distinguish in our modeling language between *urgent* and *non-urgent* locations and jumps. Time can evolve only in non-urgent locations. For jumps, urgency means that the control must leave a location as soon as the guard of an outgoing urgent transition is satisfied.

The syntax of hybrid automata is given below:

Definition 3.4.1 – Syntax of Hybrid Automata [ACH+95]

A *hybrid automaton (HA)* is a tuple $\mathcal{H} = (\mathit{Loc}, \mathit{Lab}, V_\mathcal{H}, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init}, \mathit{Urg})$, where

- *Loc* is a finite set of *locations* or *modes*;
- *Lab* is a finite set of synchronization *labels*;
- $V_\mathcal{H}$ is a finite ordered set $\{x_1, \ldots, x_n\}$ of real-valued *variables*. A *valuation* $\nu \in \mathcal{V}^{V_\mathcal{H}}$ with $\nu : V_\mathcal{H} \to \mathbb{R}$ assigns a value to each variable $x_i \in V_\mathcal{H}$. A *state* $\sigma = (l, \nu) \in \Sigma^{V_\mathcal{H}} = \mathit{Loc} \times \mathcal{V}^{V_\mathcal{H}}$ is a location-valuation pair;

- *Edge* $\subseteq$ *Loc* $\times$ *Lab* $\times\, 2^{\mathcal{V}^{V_\mathcal{H}}} \times (\mathcal{V}^{V_\mathcal{H}} \to 2^{\mathcal{V}^{V_\mathcal{H}}}) \times$ *Loc* is a finite set of *discrete transitions (jumps)*; for an edge $e = (l, \alpha, g, r, l') \in$ *Edge*, we call g the *guard* and $r : \mathcal{V}^{V_\mathcal{H}} \to 2^{\mathcal{V}^{V_\mathcal{H}}}$ the *reset function*; the locations l and l' are called the *source* and *target* mode of the discrete transition.
  We define a function `enabled` : *Loc* $\times\, \mathcal{V}^{V_\mathcal{H}} \to 2^{Edge}$ to get the subset of outgoing edges of the current location whose guards evaluate to *true* under the current valuation;
- *Act* is a function assigning a set of time-invariant *activities* $f : \mathbb{R}_{\geq 0} \to \mathcal{V}^{V_\mathcal{H}}$ to each location, i. e., for all $l \in$ *Loc*, $f \in$ *Act* (l) implies $(f + t) \in f\,(l)$ where $(f + t)\,(t') = f\,(t + t')$ for all $t, t' \in \mathbb{R}_{\geq 0}$;
- *Inv* : *Loc* $\to 2^{\mathcal{V}^{V_\mathcal{H}}}$ is a function assigning an *invariant* to each location;
- *Init* $\subseteq$ *Loc* $\times\, \mathcal{V}^{V_\mathcal{H}}$ is a set of *initial states* such that $\nu \in$ *Inv*(l) for each $(l, \nu) \in$ *Init*;
- *Urg* : (*Loc* $\cup$ *Edge*) $\to \mathbb{B}$ (with $\mathbb{B} = \{0, 1\}$) is a function defining the locations and discrete transitions with function value 1 to be *urgent*.

The activity sets are usually specified by *ODE systems*, whose solutions build the activity set. Discrete variables can be modeled by adding differential equations with constant derivative zero for these variables to each location. It is standard to require the invariants, guards, and initial sets to be specified by convex polyhedral sets: if they are not linear, they can be over-approximated[2] by a linear set; if they are not convex, they can be expressed as a finite union of convex sets (corresponding to the replacement of a transition with non-convex guard by several transitions with convex guards, and similarly for initial sets and location invariants). In the following, we syntactically allow linear non-convex conditions and use such a transformation to eliminate them from the models. Furthermore, we assume that all considered reset functions are linear.

The semantics of HA distinguishes between discrete steps (jumps) and time steps (flows). A jump follows a transition $e = (l, \alpha, g, r, l')$, transforming the current state (l, ν) to (l', ν') with $\nu' \in r(\nu))$. This transition, which has a synchronization label α (used for parallel composition), must be *enabled*, i. e., the guard g is true in ν and *Inv*(l') is true in ν'. Time steps model time elapse; from a state ν, the values of the continuous variables evolve according to an activity $f \in$ *Act*(l) with $f(0) = \nu$ in the current location l. Time cannot elapse in *urgent locations* l, identified by *Urg*$(l) = 1$, instead an outgoing transition must be taken immediately. Control can stay in a non-urgent location as long as the invariant of the location is satisfied. Furthermore, if an *urgent transition* e, identified by *Urg*$(e) = 1$, is enabled, time cannot further elapse in the location and an outgoing transition must be taken.

[2] For over-approximative reachability analysis; otherwise under-approximated.

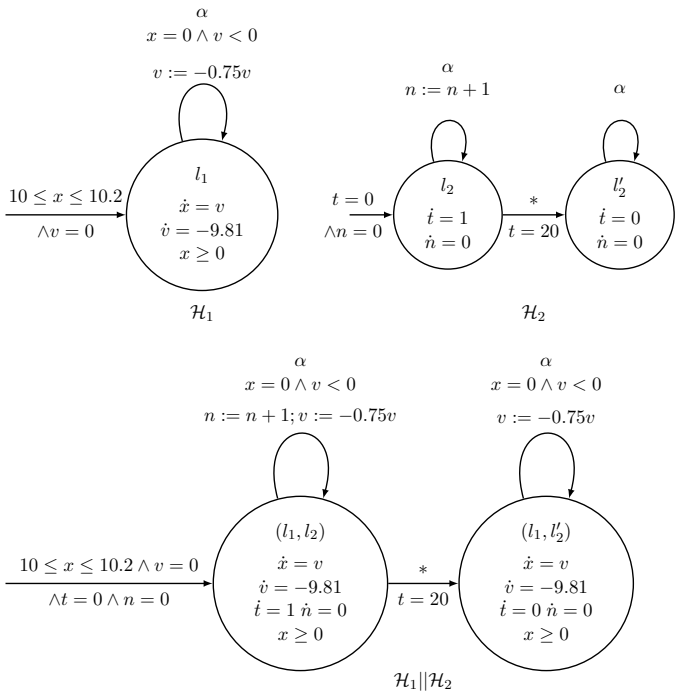


Figure 3.4.: Two example hybrid automata and their parallel composition.

Example 3.4.1

Hybrid automata are often specified graphically, as illustrated in Figure 3.4. A star $*$ in a location indicates that the location is urgent. Similarly, transitions labeled with a star $*$ are urgent.
In Figure 3.4 (top left) a hybrid automaton model $\mathcal{H}_1$ for the classical bouncing ball example is shown. A ball is dropped from some initial height with zero initial velocity. Due to gravity, the ball has an acceleration pointing towards the earth. Therefore the ball falls until it hits the ground. Then it bounces back into the air, raises until its velocity gets zero, and starts to fall again. Upon bouncing, the ball loses a fraction of its kinetic energy.
The dynamics of raising and falling is modeled in a single mode l_1 using two variables x and v, where x models the vertical position (height) and v the vertical velocity of the ball. The dynamics in l_1 is specified by the ODEs $\dot{x} = v$ and $\dot{v} = -9.81$ with the gravitational force as the only influence on the velocity of the ball. The invariant in l_1 is $x \geq 0$ which enforces that the ball bounces when it reaches the ground. This bouncing is represented by the only jump (l_1, α, g, r, l_1) with guard g given by $x = 0 \wedge v < 0$ (that means bouncing only occurs when the ball falls from above and reaches the ground) and reset r specified by $v := -0.75v$ (i. e., the sign of the velocity gets inverted and the velocity is dampened by a constant factor 0.75). The initial states are $\{(l_1, \nu) | 10 \leq \nu(x) \leq 10.2 \wedge \nu(v) = 0)\}$.

$$\frac{\begin{array}{c} l \in \mathit{Loc} \quad \mathit{Urg}(l) = 0 \quad \nu, \nu' \in \mathbb{R}^n \\ t \in \mathbb{R}_{\geq 0} \quad f \in \mathit{Act}(l) \quad f(0) = \nu \quad f(t) = \nu' \\ \forall t' \in [0,t].\ f(t') \in \mathit{Inv}(l) \quad \forall t' \in [0,t).\ \forall e \in \texttt{enabled}(l, f(t')).\ \mathit{Urg}(e) = 0 \end{array}}{(l,\nu) \to_{\mathcal{H}}^{t} (l,\nu')}\ \textsc{Time}$$

$$\frac{\begin{array}{c} l, l' \in \mathit{Loc} \quad \nu, \nu' \in \mathbb{R}^n \\ e = (l, \alpha, g, r, l') \in \mathit{Edge} \quad \nu \in g \quad \nu' \in r(\nu) \cap \mathit{Inv}(l') \end{array}}{(l,\nu) \to_{\mathcal{H}}^{e} (l',\nu')}\ \textsc{Jump}$$

Figure 3.5.: Operational semantics rules for hybrid automata.

The operational semantics of hybrid automata is given below:

Definition 3.4.2 – Semantics of Hybrid Automata

The *operational semantics* of an HA $\mathcal{H} = (\mathit{Loc}, \mathit{Lab}, V_{\mathcal{H}}, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init}, \mathit{Urg})$ with $V_{\mathcal{H}} = \{x_1, \ldots, x_n\}$ is given by the rules of Figure 3.5. The first rule specifies time evolution *(time steps)*, the second one discrete mode changes *(jumps)*.
Let $\to_{\mathcal{H}} = \bigcup_{t \in \mathbb{R}_{\geq 0}} \to_{\mathcal{H}}^{t} \cup \bigcup_{e \in \mathit{Edge}} \to_{\mathcal{H}}^{e}$. A *path* of the hybrid automaton $\mathcal{H}$ is a (finite or infinite) sequence $(l_0,\nu_0) \to_{\mathcal{H}} (l_1,\nu_1) \to_{\mathcal{H}} \ldots$. For an *initial path* we additionally require $\nu_0 \in \mathit{Init}(l_0)$. A state $(l,\nu) \in \mathit{Loc} \times \mathcal{V}^{V_{\mathcal{H}}}$ is called *reachable* in $\mathcal{H}$ if there is an initial path $(l_0,\nu_0) \to_{\mathcal{H}} (l_1,\nu_1) \to_{\mathcal{H}} \ldots$ of $\mathcal{H}$ and an index $i \geq 0$ such that $(l_i,\nu_i) = (l,\nu)$.

Please note that each reachable state (l,ν) of an HA can be reached via an initial path of the form $(l_0,\nu_0) \to_{\mathcal{H}}^{t_0} (l_0,\nu_0') \to_{\mathcal{H}}^{e_0} \ldots \to_{\mathcal{H}}^{e_{n-1}} (l_n,\nu_n) \to_{\mathcal{H}}^{t_n} (l_n,\nu_n') = (l,\nu)$ with alternating time steps and jumps for some $n \in \mathbb{N}$. In the following we consider only paths of this form.

A *trace* $e_0, e_1, \ldots$ describes a sequence of jumps with $e_i \in \mathit{Edge}$ such that the target mode of e_i is equal to the source mode of e_{i+1} for all $i \in \mathbb{N}$. If we can assume that there is at most one jump between each mode pair, we also identify traces by the sequence $l_0, l_1, \ldots$ of modes visited. Such a trace *represents* the set of all paths $(l_0,\nu_0) \to_{\mathcal{H}}^{t_0''} (l_0,\nu_0') \to_{\mathcal{H}}^{e_0} (l_1,\nu_1) \to_{\mathcal{H}}^{t_1''} (l_1,\nu_1') \to_{\mathcal{H}}^{e_1} \ldots$. We say that those paths are *contained* in the trace.

A *timed trace* $e_0, [t_0,t_0'], e_1, [t_1,t_1'], \ldots$ annotates a trace $e_0, e_1, \ldots$ with time intervals and *represents* the set of all paths $(l_0,\nu_0) \to_{\mathcal{H}}^{t_0''} (l_0,\nu_0') \to_{\mathcal{H}}^{e_0} (l_1,\nu_1) \to_{\mathcal{H}}^{t_1''} (l_1,\nu_1') \to_{\mathcal{H}}^{e_1} \ldots$ with $t_i'' \in [t_i,t_i']$ for all $i \in \mathbb{N}$. We say that $e_0, [t_0,t_0'], e_1, [t_1,t_1'], \ldots$ is a timed trace of the represented paths, which are *contained* in the timed trace.

If an HA is unsafe, a *counterexample* is an initial path of the HA leading to a state from a set B of unsafe states. For models with weaker expressivity, for example hybrid automata defined by linear predicates and constant derivatives (i. e., dynamics of the form $\bigwedge_{x \in V_{\mathcal{H}}} \dot{x} = c_x$ with $c_x \in \mathbb{Z}$ for all $x \in V_{\mathcal{H}}$), the *bounded* reachability problem is decidable and, for unsafe models, counterexamples can be generated (e. g.,

by bounded model checking using SMT solving with exact arithmetic). However, the computation of counterexamples for general hybrid automata is hard. Theoretically, it could be done by (incomplete) *under-approximative* reachability computations, however currently there are no techniques available for this task.

In Section 5.1, we propose an approach to generate and refine *presumable counterexamples*. Presumable counterexamples are timed traces that *might* contain a counterexample; presumable counterexamples that do *not* contain any counterexample are called *spurious*.

The parallel composition is defined for two HA; for a set of HA it is computed iteratively. For simplicity, we assume that the components have the same variable set (otherwise we extend the models to use the union of the variable sets and allow arbitrary behavior for the new variables). In the parallel composition $\mathcal{H} = (Loc, Lab, V^{\mathcal{H}}, Edge, Act, Inv, Init, Urg)$ of two hybrid automata $\mathcal{H}_i = (Loc_i, Lab_i, V^{\mathcal{H}_i}, Edge_i, Act_i, Inv_i, Init_i, Urg_i)$, $i = 1, 2$, each location $l = (l_1, l_2) \in Loc = Loc_1 \times Loc_2$ is a pair of locations, one from each component. The continuous behavior in the composition is the intersection of the continuous behaviors in the components, i. e., $Act(l) = Act_1(l_1) \cap Act_2(l_2)$ and $Inv(l) = Inv_1(l_1) \cap Inv_2(l_2)$. A location in the parallel composition is urgent if at least one of the component locations is urgent, i. e., $Urg(l) = Urg_1(l_1) \vee Urg_2(l_2)$. The jump set of the composition over the labels $Lab = Lab_1 \cup Lab_2$ contains for each non-synchronizing jump $e_1 = (l_1, \alpha_1, g_1, r_1, l_1') \in Edge_1$ of $\mathcal{H}_1$ with $\alpha_1 \notin Lab_2$ and for each $l_2 \in Loc_2$ a jump $e = ((l_1, l_2), \alpha_1, g_1, r_1, (l_1', l2))$; this holds analogously for non-synchronizing transitions of $\mathcal{H}_2$. Additionally, the jump set contains the compositions $e = ((l_1, l_2), \alpha, g_1 \cap g_2, \lambda\nu.r_1(\nu) \cap r_2(\nu), (l_1', l_2'))$ of all synchronizing jump pairs $e_i = (l_i, \alpha, g_i, r_i, l_i') \in Edge_i$. Analogously, a jump of the composition is urgent if it is a non-synchronizing urgent jump in one of the components, or at least one of the synchronizing component jumps is urgent. Finally, also the initial sets are determined by intersection, i. e. $Init = \{((l_1, l_2), \nu) | (l_i, \nu) \in Init_i,\ i = 1, 2\}$.

For more information on the formal semantics and the parallel composition of hybrid automata we refer to [ACH+95].

Example 3.4.2

An example for the parallel composition of hybrid automata is shown in Figure 3.4. We compose the bouncing ball model $\mathcal{H}_1$ (top left) with a second automaton $\mathcal{H}_2$ (top right), which counts the number of bounces that happen within the first 20 time units. The composition is shown at the bottom of the figure. Note that this composition exhibits non-determinism: if a bouncing happens at time point 20, the jump for bouncing can be taken before or after the urgent jump for stopping counting.

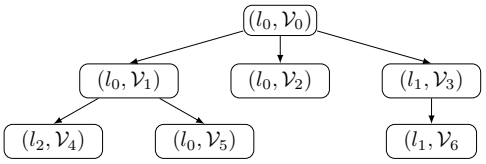


Figure 3.6.: Example search tree.

3.4.1. Reachability Analysis of HA

Though the *reachability problem* for HA is undecidable in general [HKPV98], there exist several approaches to compute an *over-approximation* of the set of reachable states. In this thesis we use *flowpipe-computation-based reachability analysis* methods. These use geometric objects (e. g. convex polytopes, zonotopes, ellipsoids, etc.) or symbolic representations (e. g. support functions or Taylor models) for the over-approximative representation of state sets. The efficiency of certain operations (i. e. intersection, union, linear transformation, projection, and Minkowski sum) on such objects determines the efficiency of their application in the reachability analysis of HA.

The basic idea of the reachability analysis is as follows: Let $\mathcal{R}_0 = (l_0, \mathcal{V}_0)$ specify an initial location $l_0 \in \mathit{Loc}$ and a set $\mathcal{V}_0 \subseteq \mathbb{R}^n$ of initial valuations (in some representation). We assume furthermore a step size $\tau \in \mathbb{R}_{>0}$ and a time horizon $\Delta = m\tau$ ($m \in \mathbb{N}_{>0}$). First the so-called *flowpipe*, i. e., the set of states reachable from $\mathcal{R}_0$ within time Δ in l_0, is computed. To reduce the approximation error, this is done by computing a sequence $\mathcal{V}_1, \ldots, \mathcal{V}_m$ of *flowpipe segments*, where for each $0 < i \leq m$ the set $\mathcal{V}_i$ over-approximates the set of states reachable from $\mathcal{V}_0$ in time $[(i-1)\tau, i\tau]$ according to the dynamics in l_0. The intersection of these sets with the invariant of l_0 gives us the time successors of $\mathcal{V}_0$ within time Δ. Finally, we also need to compute for each of the flowpipe segments (intersected with the invariant) all possible jump successors. This latter computation involves the intersection of the flowpipe segments with the transition guards, state set transformations for applying the reset functions, and an intersection computation with the target location's invariant; special attention has to be payed to urgent locations and transitions.

The whole computation of flowpipe segments and jump successors is applied in later iterations to each of the above computed successor sets (for termination usually both the maximal time delay in a location and the number of jumps along paths are bounded). Thus the reachability analysis computes a *search tree*, where each node is a pair of a location and a set of valuations, whose children are its time and jump successors (see Figure 3.6).

Different heuristics can be applied to determine the node whose children will be computed next. Nodes, whose children still need to be computed, are marked to be *non-completed*, the others are marked *completed*. When applying a fixed-point check, only those nodes which are not included in other nodes are marked as non-completed.

As the reachability problem for hybrid systems is undecidable in general, all approaches for reachability analysis are inherently incomplete. If the computation of the reachable state sets is over-approximative, the tool can show safety, however, it is not well-suited to prove that a system is unsafe. Validation techniques as for example suggested in Section 5.1 can sometimes show that a path detected in the over-approximation is not executable in the model, but in general they cannot give a proof of executability.

In our approaches, we use the SpaceEx tool [FLD+11, Spa] which is available as a standalone tool with a web interface as well as a command-line tool that provides the analysis core and is easy to integrate into other projects. To increase efficiency, SpaceEx can compute the composition of HA on-the-fly during the analysis.

3.5. Bounded Model Checking

We use *BMC* [BCCZ99, CK03] to analyze the discrete models of PLC programs. Given a set of target states which are usually unsafe, BMC generates incrementally for $k = 0, 1, \ldots$ formulas φ_k^{BMC} that are satisfiable if and only if there is an execution path of length k of the program that leads to a target state. These formulas encode all execution steps in the PLC cycle. For the encoding of the input from the environment, the input variables remain unspecified, i. e., we consider arbitrary environment behavior both for user input as well as for the sensor and actuator states. If a counterexample path is found, we will apply a refinement cycle: The discrete counterexamples obtained by the BMC algorithm are checked for realizability in the plant using hybrid systems reachability analysis techniques. If the counterexamples are spurious, i. e. not realizable, the refinement step excludes them from future search.

To proceed along these lines, we need to formalize the set of unsafe states at the PLC level. To do so, we might need to augment the SFC program to shift some information about the plant state to the PLC state.

Example 3.5.1

We logically extend our tank example with sensors $\texttt{empty}_i$ and $\texttt{full}_i$ for an empty and a full tank, whose values are read into the input variables empty_i or full_i, respectively, i. e., $\mathit{empty}_i \equiv (h_i = 0)$ and $\mathit{full}_i \equiv (h_i = F_i)$ for tanks $U_i,\ i = 1, 2$. Safety is specified within the SFC by asserting $\neg\mathit{empty}_i \wedge \neg\mathit{full}_i$ at the beginning of each scan cycle.

In the next step, the program is transformed into a symbolic representation. Let V_C be the set of all SFC variables[3], $V_C{}' = \{x' | x \in V_C\}$ a set of renamed SFC variables, and analogously $V_C^i = \{x_i | x \in V_C\}$. The semantics of the PLC program can be encoded by a triple $\mathit{Prog} = (I, \mathit{Tr}, \mathit{Prop})$, where I and Prop are quantifier-free arithmetic

[3] Additionally to the previously introduced SFC variables, V_C contains also variables to encode the currently active steps.

formulas over the variables V_C, and *Tr* is a quantifier-free arithmetic formula over $V_C \cup V_C'$. The satisfying valuations of the formulas *I* and *Prop* are the sets of initial and safe states, respectively. The formula *Tr* encodes the transition relation such that it is satisfied by exactly those valuations $\nu(x)$ to $x \in V_C$ and $\nu'(x)$ to $x' \in V_C'$ for which it holds that starting in the state ν and executing a PLC cycle results in the state ν'.

For simplicity, in the following we use I_0 to denote the formula resulting from *I* after substituting x_0 for each $x \in V_C$; we define $Prop_i$ analogously, whereas in $Tr_{i,i+1}$ ($i \geq 0$) we substitute x_i for x and x_{i+1} for x' for all $x \in V_C$.

A k-bounded safety analysis of the discrete control program is performed by unrolling the transition relation k times into a formula φ_k^{BMC} over variables $\bigcup_{i=0}^{k} V_C^i$, and stating that the encoded paths of length k reach an unsafe state:

$$\varphi_k^{BMC} = I_0 \wedge \bigwedge_{i=0}^{k-1} Tr_{i,i+1} \wedge \neg Prop_k \,.$$

The formula φ_k^{BMC} is satisfiable if and only if an unsafe state is reachable in k PLC cycles. Moreover, any satisfying assignment μ of φ_k^{BMC} corresponds to a discrete counterexample.

Once a discrete counterexample μ has been discovered, a hybrid reachability analysis checks its feasibility in the hybrid plant environment. If the hybrid analysis discharges μ as being spurious, we construct an *explanation* as a formula ψ that excludes the sequence of inputs and outputs produced by the unconstrained discrete analysis. Before the next satisfiability check, ψ is conjoined to φ_k^{BMC}. In this way, the over-approximation of the discrete model is refined in each iteration. Once all discrete counterexamples up to bound k have been discharged, we increase k.

Several SAT and SMT solvers (like Z3 or MATHSAT5) support incremental satisfiability checking. Incrementality allows to store the current solver state (push), to assume that certain formulas hold (assert), to check satisfiability of the currently asserted statements (checkSatisfiability), and to undo assertions by re-storing previous solver states (pop). The structure of the incremental BMC approach is shown in Algorithm 2.

First we set k to 0 and Ψ_k to *true*. Ψ_k will be used to collect all explanations at BMC level k, i. e. the reasons why the unsafe PLC program paths of length k, which were found so far during the BMC analysis, are not realizable in the plant. We assert I_0, encoding paths of length 0, store the solver's state by pushing it to a stack, and assert $\neg Prop_k$. At this point, where we enter the main loop, we will maintain the invariant that (1) in the last stored solver state the assertions encode all (safe or unsafe) paths of length k and all explanations found for $i < k$, and (2) the assertions added since the last push assure that solution paths of length k end in unsafe states and moreover exclude all previously found φ_k^{BMC} solutions by asserting their explanations.

In the loop we first check the satisfiability of the currently stated assertions. If they are unsatisfiable, we move to the next BMC level: We restore the last solver

Algorithm 2: IncrementalBMC

```
1  k:=0;
2  Ψ_k:=true;
3  assert(I_0);
4  push();
5  assert(¬Prop_k);
6  while true do
7      (result,solution):=checkSatisfiability();
8      if result=UNSAT then
9          pop();
10         k:=k+1;
11         Ψ_k:=true;
12         assert(Ψ_{k−1});
13         assert(Tr_{k−1,k});
14         push();
15         assert(¬Prop_k);
16     else
17         (plantResult,ψ):=checkRealisability(solution);
18         if plantResult=UNSAT then
19             Ψ_k:=Ψ_k ∧ ψ;
20             assert(ψ);
21         else
22             return solution;
```

state, increase k, initialize Ψ_k to *true*, assert the explanations found at the last level (which were removed by the last pop operation), and add a new unwinding of the transition relation. We store this solver state, encoding paths of length k and all explanation found at BMC levels less than k. We assert now that the kth state is unsafe and continue with the next loop iteration.

Otherwise, if we get a satisfying solution, we check whether it is realizable in the plant. If not, we assert the received explanation, and remember it also in our collection Ψ_k. If the solution is realizable, we report possible violation of the safety property.

3.6. Counterexample-Guided Abstraction Refinement (CEGAR)

Reachability analysis for HA can be used to prove that no state from a given *unsafe* set is reachable from a set of initial states. However, for complex models the analysis might take such a long time that the approach becomes practically inapplicable. In such cases, *abstraction* can be used to reduce the complexity of the model at the cost of over-approximating the system behavior. If the abstraction can be proven to be

safe then also the concrete model is safe. If the abstraction is too coarse to satisfy the required safety property, it can be *refined* by adding more detailed information about the system behavior. This iterative approach is continued until either the refinement level is fine enough to prove the specification correct or the model is fully concretized. In *CEGAR*, the refinement step is guided by a counterexample path that leads from an initial state to an unsafe one in the abstraction. The counterexample is called *spurious* if it exists only in the abstraction but not in the concrete model; the refinement step adds additional information from the concrete model to the abstract one along the spurious counterexample path.

4. Modeling SFC-Controlled Plants

In automation *PLCs* are widely used to control the behavior of a plant. In the industry standard IEC 61131-3 [Int03] several languages are specified which can be used for the programming of a PLC. Commonly used in process control are *SFCs*, a graphical language which allows the structuring of control sequences into several steps or into branches that are executed in parallel.

In this chapter we first describe an approach to model the cyclic execution of a controller on a PLC in the modeling language of hybrid automata.

We model SFC-controlled plants by a single HA, which is the composition of several HA for the different system components [1, 3, 4, 5]: We use one HA to model each SFC controller (see Figure 4.1). We specify a further HA to model timing aspects that come with the cyclic execution on a PLC and the user input (see Figure 4.3). Additional HA model the plant dynamics according to a given conditional ODE system (see Figure 4.4). The resulting hybrid automaton allows to extract behaviors of single plant components, which is very helpful for abstraction and refinement. The parallel composition of the above mentioned hybrid automata gives us a model for the SFC-controlled plant.

We specify the dynamic behavior of the plant's physical quantities by sets of *conditional ODE systems*. Each conditional ODE system specifies the behavior of a physical quantity for different conditions. The condition expresses assumptions about the current state of the system. For example, the dynamic change of the water level in a tank can be given as the sum of the flows through the pipes that fill and empty the tank. This sum may vary depending on valves being open or closed, pumps being on or off, and other tanks being empty or not.

Next we show how a HA model can be built from a conditional ODE system. Our formalism allows a modular definition of the dynamic continuous behavior of plant elements with different levels of details for different control modes. This is useful for applying abstraction refinement, e. g. for CEGAR-based analysis.

Using the above mentioned transformation, we can build the parallel composition of the HA models, modeling the PLC-controlled plant, and apply existing tools for hybrid automata analysis to check properties of the hybrid models.

Outline The foundations of this chapter, namely an introduction to SFCs and hybrid automata, can be found in the Sections 3.3 and 3.4. This chapter is structured as follows: We start introducing the interface between the PLC, the SFC controllers, and the plant in Section 4.1. Afterwards, we show how the plant dynamics can be specified (Section 4.2). We continue with the modeling, describing the transformation of an SFC-controller into a hybrid automaton model (Section 4.3), a HA model

for the PLC cycle synchronization (Section 4.4), and an HA model for each physical quantity of the plant (Section 4.5). Section 4.6 shows how the complete model is built using parallel composition before we conclude this chapter in Section 4.7.

4.1. Interface

A plant can be described by the current states of the actuators, sensors, and physical quantities. The actuator and sensor states are given by the current variable valuation of the variables in the sets V_{act} and V_{sen}, respectively. The states of the physical quantities are specified by the values of the variables from a set V_{cont}; these values evolve continuously over time. The plant variables are thus given by the set $V_{plant} := V_{act} \cup V_{sen} \cup V_{cont}$. As described in Section 3.3, the controller state can be described by the valuation of the input (V_{in}), output (V_{out}), and local (V_{loc}) variables of the SFC, i. e. $V_{\mathcal{C}} := V_{in} \cup V_{out} \cup V_{loc}$. We assume the variable sets to be ordered and use vector notation to list their elements.

Example 4.1.1

The variable sets for the two tank system from Section 3.2 is given below:

$$\begin{aligned}
V_{act} &= (\mathtt{P}_1^+, \mathtt{P}_1^-, \mathtt{P}_1, \mathtt{P}_2^+, \mathtt{P}_2^-, \mathtt{P}_2) \\
V_{sen} &= (\mathtt{low}_1, \mathtt{high}_1, \mathtt{low}_2, \mathtt{high}_2) \\
V_{cont} &= (h_1, h_2) \\
\\
V_{in} &= (\mathcal{P}_1^+, \mathcal{P}_1^-, \mathcal{P}_1, low_1, high_1, \\
&\quad\ \mathcal{P}_2^+, \mathcal{P}_2^-, \mathcal{P}_2, low_2, high_2) \\
V_{loc} &= () \\
V_{out} &= (\mathcal{P}_1^+, \mathcal{P}_1^-, \mathcal{P}_1, \mathcal{P}_2^+, \mathcal{P}_2^-, \mathcal{P}_2)\,.
\end{aligned}$$

At the beginning of each PLC cycle the PLC reads the current state of the actuators and sensors from the plants. The corresponding values are stored in variables from the set V_{in} of the controller. During the PLC cycle, the controller uses the input variables and some local variables from V_{loc} to compute the output, which is stored in V_{out}. At the end of the PLC cycle these output variables are written back to the PLC which updates the actuators of the plant accordingly. Figure 3.3 on page 18 illustrates the interface between the plant and the PLC.

4.2. Plant Specifications

The plant specification defines a set of *initial* states. Starting from the initial states, we model the dynamics of the evolution by a set of conditional *ODE systems*, one conditional ODE system for each continuous variable in V_{cont}. The conditional ODEs

for a plant can be derived from the hardware specifications and the set-up of the plant.

A *conditional ODE system* is a set of *conditional ODEs*. A conditional ODE is a pair of a condition and an ordinary differential equation. The condition expresses assumptions about the overall system state of the plant, whereas the ODE describes the dynamic behavior of a physical quantity of the plant under the assumption that the condition holds. The semantics allows arbitrary behavior for those physical quantities whose evolution is not fixed by a conditional ODE.

Definition 4.2.1 – Conditional ODE System

Let $c \in \Phi^{V_{plant}}$ be a predicate over the set of plant variables V_{plant} and let o be a differential equation for a variable $x \in V_{cont}$ over the plant variables V_{plant}. The pair (c, o) is called a *conditional ODE* for the physical quantity x. The differential equation o specifies the dynamics for x in those states that satisfy the condition c.

A *conditional ODE system* for a physical quantity $x \in V_{cont}$ is an ordered set of conditional ODEs for x. The order of the conditional ODEs in a system specifies priorities: The first conditional ODE, whose condition is satisfied by the current state, defines the current dynamics for x; if none of the conditions of a conditional ODE system apply, we assume arbitrary behavior.

Example 4.2.1

For the example plant from Section 3.2, in the initial states both pumps are on and the water levels are between the low and high sensors ($\wedge_{i=1,2}\, \mathtt{P}_i \wedge L_i < h_i < H_i$). We define the following conditions over the variable set V_{plant}:

$$\begin{aligned}
\varphi_{2\not\to 1} &\equiv \neg \mathtt{P}_2 \vee h_1 = F_1 \vee h_2 = 0 \\
\varphi_{2\to 1} &\equiv \mathtt{P}_2 \wedge h_1 \leq F_1 \wedge h_2 \geq 0 \\
\varphi_{1\not\to 2} &\equiv \neg \mathtt{P}_1 \vee h_1 = 0 \vee h_2 = F_2 \\
\varphi_{1\to 2} &\equiv \mathtt{P}_1 \wedge h_2 \leq F_2 \wedge h_1 \geq 0
\end{aligned}$$

If the condition $\varphi_{i\to j}$ is satisfied by the current system state then water is pumped from tank i to tank j without over-flooding the latter. Otherwise, either no water is pumped from tank i to tank j or tank j is full, and the condition $\varphi_{i\not\to j}$ is satisfied.

The conditional ODE system for the physical quantity h_1 is given by:

$$\begin{aligned}
(c_1^{h_1}, \text{ODE}_1^{h_1}) &= (\varphi_{1\not\to 2} \wedge \varphi_{2\not\to 1}, \quad \dot{h}_1 = 0) \\
(c_2^{h_1}, \text{ODE}_2^{h_1}) &= (\varphi_{1\not\to 2} \wedge \varphi_{2\to 1}, \quad \dot{h}_1 = k_2) \\
(c_3^{h_1}, \text{ODE}_3^{h_1}) &= (\varphi_{1\to 2} \wedge \varphi_{2\not\to 1}, \quad \dot{h}_1 = -k_1) \\
(c_4^{h_1}, \text{ODE}_4^{h_1}) &= (\varphi_{1\to 2} \wedge \varphi_{2\to 1}, \quad \dot{h}_1 = k_2 - k_1)
\end{aligned}$$

The conditional ODE system for h_2 is analogous:

$$\begin{aligned}
(c_1^{h_2}, \text{ODE}_1^{h_2}) &= (\varphi_{1\not\to 2} \wedge \varphi_{2\not\to 1}, \quad \dot{h}_2 = 0) \\
(c_2^{h_2}, \text{ODE}_2^{h_2}) &= (\varphi_{1\not\to 2} \wedge \varphi_{2\to 1}, \quad \dot{h}_2 = -k_2) \\
(c_3^{h_2}, \text{ODE}_3^{h_2}) &= (\varphi_{1\to 2} \wedge \varphi_{2\not\to 1}, \quad \dot{h}_2 = k_1) \\
(c_4^{h_2}, \text{ODE}_4^{h_2}) &= (\varphi_{1\to 2} \wedge \varphi_{2\to 1}, \quad \dot{h}_2 = -k_2 + k_1)
\end{aligned}$$

As mentioned above, if the conditions of two conditional ODEs in a system overlap, the conditional ODE with the higher priority determines the behavior. To simplify the notations in later sections, we apply a preprocessing step to each conditional ODE system. The sequence $(c_1, \text{ODE}_1), \ldots, (c_k, \text{ODE}_k)$ is transformed to the conditional ODE system $(c_1', \text{ODE}_1), \ldots, (c_k', \text{ODE}_k)$ with modified conditions $c_i' = c_i \wedge \bigwedge_{j=1}^{i-1} cl(\neg c_j)$, where $cl(S)$ denotes the closure of the set S. After this transformation, all conditions in a conditional ODE will overlap only at their boundaries. In our example the conditions are not changed by this transformation, as the conditions already overlap only at their boundaries.

If the condition c of a conditional ODE (c, ODE) is not conjunctive, in our implementation we transform c to disjunctive normal form $c_1 \vee \ldots \vee c_l$, and replace the conditional ODE (c, ODE) by a sequence of conditional ODEs $(c_1, \text{ODE}), \ldots, (c_l, \text{ODE})$ having conjunctive predicates.

Though in this work changes in the dynamics are determined by conditions and priorities only, it is also possible to extend the modeling approach by specifying transitions between conditional ODEs. Such transitions restrict the switching from one conditional ODE to another one by a switching condition over V_{plant}, and possibly specifying also a reset function that is applied upon switching. Such transitions are important to model discrete changes that are caused by changes in the physical quantities, e. g., to model discrete changes in the aggregation state.

4.3. Hybrid Automata Models for SFC Controller

Our HA model $\mathcal{H}$ for an SFC controller is illustrated in Figure 4.1. The set of variables $V_{\mathcal{H}} := V_{in} \cup V_{loc} \cup V_{out}$ is the union of the input, the local, and the output

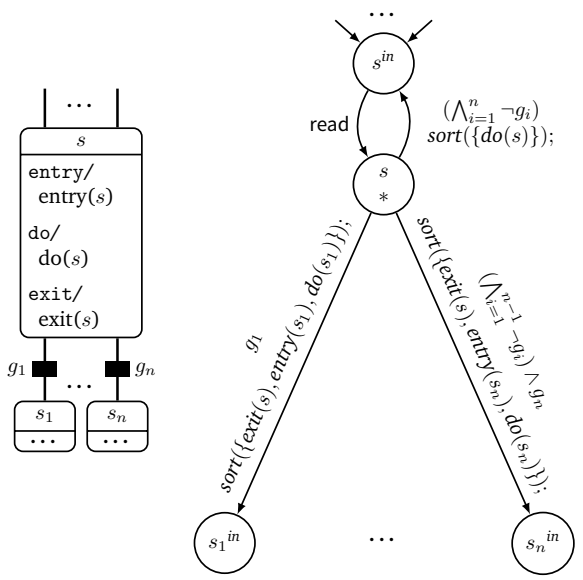


Figure 4.1.: Hybrid automaton for an SFC. The actions are sorted according to a specified priority order.

variables of the SFC. For each step s of the SFC there is a corresponding location pair in the HA: the location s^{in} is entered upon the activation of the step and it is left for location s when the input is read. The execution of the actions is modeled to happen at the beginning of the PLC cycle by defining location s to be urgent. The outgoing transitions of s represent the cycle execution: If s remains activated then its `do` actions else its `exit` actions and both the `entry` and the `do` actions of the next step are executed in their priority order. The location ${s_0}^{in}$ that corresponds to the initial step s_0 defines the initial location of the HA.

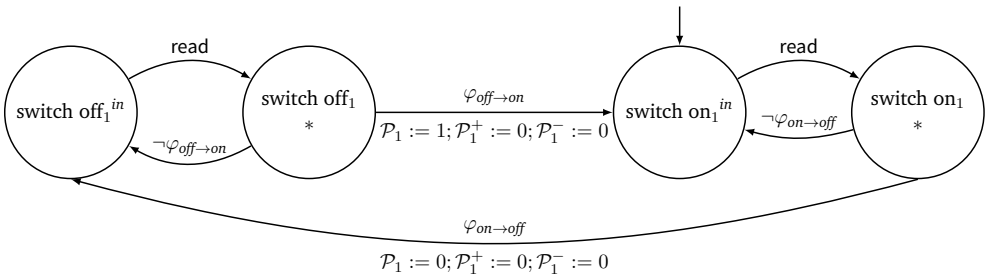


Figure 4.2.: Hybrid automaton model of the SFC controller for pump $\mathtt{P}_1$.

Definition 4.3.1 – Controller Automaton

Let $\mathcal{C} = (V_\mathcal{C}, \mathcal{S}, \mathcal{A}, s_0, \mathit{Trans}, \mathcal{B}, \sqsubset, \prec, \mathit{Hist})$ be an SFC without nested components and parallel transitions. Its transformation to a hybrid automaton model $\mathcal{H} = (\mathit{Loc}, \mathit{Lab}, V_\mathcal{H}, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init}, \mathit{Urg})$ is given as follows:

- $\mathit{Loc} = \bigcup_{s \in \mathcal{S}} \{s^{in}, s\}$,
- $\mathit{Lab} = \{\mathsf{read}, \tau\}$ with τ being a fresh label for local non-synchronizing steps,
- $V_\mathcal{H} = V_\mathcal{C}$,
- $\mathit{Edge} = \bigcup_{s \in \mathcal{S}} \mathit{Edge}_s$ where for each step $s \in \mathcal{S}$ with outgoing transitions $t_1, \ldots, t_n \in \mathit{Trans}$, $t_i = (s, g_i, s_i)$, and ordering $t_1 \succ \ldots \succ t_n$ we define $\mathit{Edge}_s = (\bigcup_{i=1}^{n} \{(s, \tau, g', r_i, {s_i}^{in})\}) \cup \{(s, \tau, g'', r_s, s^{in}), (s^{in}, \mathsf{read}, \mathit{true}, \lambda\nu.\{\nu\}, s)\}$ with
  - $g' := g_i \wedge \bigwedge_{j=1}^{i-1} \neg g_j$,
  - r_i with $r_i(\nu) = \{\nu'\}$ where ν' results from ν by applying the exit actions of s and the entry and do actions of s_i in decreasing priority order,
  - $g'' := \bigwedge_{j=1}^{n} \neg g_j$, and
  - r_s with $r_s(\nu) = \{\nu'\}$ where ν' results from ν by applying the do actions of s in decreasing priority order,
- $\mathit{Act}(s)$ is specified for all $s \in \mathit{Loc}$ by $\dot{v} = 0$ for all $v \in V_\mathcal{H}$,
- $\mathit{Inv}(s)$ is the set of all valuations for $V_\mathcal{C}$ in all locations $s \in \mathit{Loc}$,
- $\mathit{Init} = \{({s_0}^{in}, \nu_0)\}$ with ν_0 assigning initial values to the variables, and
- for each $e \in \mathit{Edge}$, $\mathit{Urg}(e) = 0$, and for each step $s \in \mathcal{S}$, the urgency of the corresponding locations $s, s^{in} \in \mathit{Loc}$ is $\mathit{Urg}(s) = 1$ and $\mathit{Urg}(s^{in}) = 0$, respectively.

Example 4.3.1

For the two tank system from Section 3.2, the SFC for pump $\mathtt{P}_1$ (cf. Figure 3.2 on page 17) is modeled by the hybrid automaton in Figure 4.2.

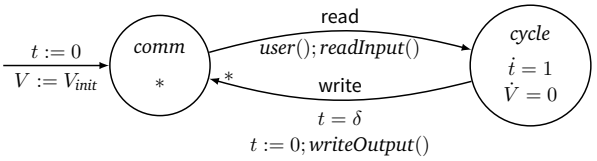


Figure 4.3.: Hybrid automaton modeling the user input, the sensors, and the PLC cycle synchronization.

4.4. Modeling the PLC Cycle Synchronization

SFCs running parallel on a PLC synchronize in each cycle on reading the input and writing the output. These synchronization aspects of the SFC execution are modeled by a HA that is shown in Figure 4.3.

We use a clock t with initial value 0 to measure the cycle time. The initial location *comm* is urgent, represented by a star $*$, thus the outgoing transition to location *cycle* will be taken immediately, representing the beginning of a cycle. This transition has a synchronization label read, which forces all SFCs to synchronize on this step.

Before reading the input upon this synchronizing transition, we simulate the user and the sensors by a function $user()$. (Though we could model the sensors also in the plant model, this function allows us to reduce the model sizes.) Sequentially, reading the input is modeled by the function $readInput()$, which stores the current state (V_{act}, V_{sen}) of the actuators and sensors to the input variables V_{in} of the SFC.

The transition from *cycle* to *comm* is urgent, again represented by a star $*$, forcing the writing to happen at the end of each cycle after a constant cycle time δ. The values of the output variables are sent to the environment to control the actuators; we model the output by writing the values of V_{out} to V_{act} in *writeOutput()*. We assume that the commands to control the actuators of the plant are immediately executed. The synchronization label write will assure the synchronization with the plant model on the actuator updates.

While time elapses in location *cycle*, the SFCs perform their active actions and the dynamic behavior of the environment evolves according to the specified differential equations. The ODE $\dot{V} = 0$ expresses that the derivatives of *all* involved *discrete* variables appearing in $V_{act}, V_{sen}, V_{in}, V_{loc}$, or V_{out} are zero. (For simplicity, here we specify the derivative 0 for all discrete variables in the PLC synchronizer model; in our implementation the SFC variables are handled in the corresponding SFC models.)

Definition 4.4.1 – PLC Cycle Synchronization Automaton

The HA model for the PLC cycle synchronization is given by $\mathcal{H} = (Loc, Lab, V_{\mathcal{H}}, Edge, Act, Inv, Init, Urg)$ with

- $Loc = \{comm, cycle\}$,

- $Lab = \{\mathsf{read}, \mathsf{write}\}$,
- $V_{\mathcal{H}} = V_{plant} \cup V_{\mathcal{C}} \cup \{t\}$,
- $Edge = \{e_1, e_2\}$ where
  - $e_1 := (comm, \mathsf{read}, true, r_1, cycle)$ with $r_1(\nu) = \{\nu'\}$ where ν' results from ν by applying the assignments specified by $user(); readInput()$, and
  - $e_2 := (cycle, \mathsf{write}, t = \delta, r_2, comm)$ with $r_2(\nu) = \{\nu'\}$ where ν' results from ν by applying the assignments specified by $t := 0; writeOutput()$,
- $Act(l)$ is specified by $\dot{t} = 1$ and $\dot{x} = 0$ for all discrete variables $x \in V_{\mathcal{H}} \setminus (\{t\} \cup V_{cont})$ in all $l \in Loc$,
- $Inv(l) = true$ for all $l \in Loc$,
- $Init = \{(comm, \nu_0)\}$ with ν_0 assigning initial values to the variables, and
- $Urg(comm) = 1$, $Urg(cycle) = 0$, $Urg(e_1) = 0$, and $Urg(e_2) = 1$.

Example 4.4.1

For the tank example, we allow arbitrary (type-correct) user input, where we use $\star$ to represent a non-deterministically chosen value.
To model the user and the sensors, $user()$ sets $\mathtt{P}_i^+ := \star$, $\mathtt{P}_i^- := \star$, $\mathtt{low}_i := (h_i \geq L_i)$ and $\mathtt{high}_i := (h_i \geq H_i)$ for $i = 1, 2$.
The function $readInput()$ executes $\mathcal{P}_i^+ := \mathtt{P}_i^+$, $\mathcal{P}_i^- := \mathtt{P}_i^-$, $\mathcal{P}_i := \mathtt{P}_i$, $low_i := \mathtt{low}_i$ and $high_i := \mathtt{high}_i$ for $i = 1, 2$.
Writing the output $writeOutput()$ updates $\mathtt{P}_i^+ := \mathcal{P}_i^+$, $\mathtt{P}_i^- := \mathcal{P}_i^-$ and $\mathtt{P}_i := \mathcal{P}_i$.

4.5. Modeling the Plant Dynamics

The plant's dynamics is described by ordered sets of conditional ODEs, one ordered set for each involved physical quantity. We define a HA for each such quantity (see Figure 4.4); their composition gives us a hybrid automaton model for the plant. This automaton uses the variables $V_{\mathcal{H}} = V_{cont} \cup V_{act} \cup V_{sen}$, and contains one location l_i ($i = 1, \ldots, n$) for each of the conditional ODEs in the system $(c_1, \mathrm{ODE}_1), \ldots, (c_n, \mathrm{ODE}_n)$, and one location $l_\perp$ for arbitrary dynamic behavior (for the case that none of the ODE conditions hold). The conditions specify the locations' invariants, the ODEs the activities; the location with arbitrary dynamics has the invariant *true*.

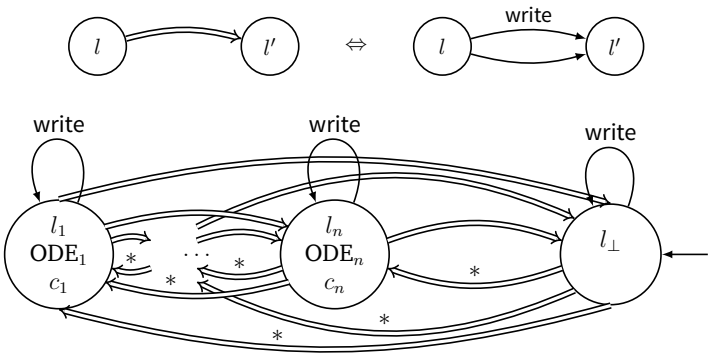


Figure 4.4.: Hybrid automaton for the plant dynamics (bottom) using the conditional ODEs $\{(c_1, \text{ODE}_1), \ldots (c_n, \text{ODE}_n)\}$ and a special notation "⇒" which is explained in the top of the Figure.

Each pair of locations is connected by discrete transitions. The "⇒" arrows in Figure 4.4 denote a pair of discrete transitions, a non-synchronizing one and a transition that synchronizes on write. To correctly model the priorities of the conditional ODEs, all transitions from locations l_i to l_j with $i > j$, and all transitions leaving $l_\perp$ are urgent.

The initial location (the right one in Figure 4.4) allows arbitrary behavior for the physical quantities; note that since there is an outgoing transition with guard *true*, this location is immediately left if there are other locations with applicable concrete dynamics for the current state.

Definition 4.5.1 – Plant Automaton

The hybrid automaton model for a conditional ODE system $(c_1, \text{ODE}_1), \ldots, (c_n, \text{ODE}_n)$ is given by $\mathcal{H} = (\mathit{Loc}, \mathit{Lab}, V_\mathcal{H}, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init}, \mathit{Urg})$ with

- $\mathit{Loc} = \{l_1, \ldots, l_n, l_\perp\}$,
- $\mathit{Lab} = \{\text{write}, \tau'\}$ with τ' being a fresh non-synchronizing label,
- $V_\mathcal{H} = V_{\mathit{cont}} \cup V_{\mathit{act}} \cup V_{\mathit{sen}}$,
- $\mathit{Edge} = \{e_{l,l'} \mid l, l' \in \mathit{Loc} \wedge l \neq l'\} \cup \{e^w_{l,l'} \mid l, l' \in \mathit{Loc}\}$ where
  - $e_{l,l'} := (l, \tau', \mathit{true}, r, l')$ with $r(\nu) = \{\nu\}$,
  - $e^w_{l,l'} := (l, \text{write}, \mathit{true}, r^w, l')$ with $r^w(\nu) = \{\nu' \mid \forall x \in V_{\mathit{cont}}.\ \nu'(x) = \nu(x)\}$,
- $\mathit{Act}(l_i)$ is specified as the solution set of ODE$_i$ for $i = 1, \ldots, n$, whereas $\mathit{Act}(l_\perp)$ allows arbitrary behavior for the continuous variables[a] from V_{cont},

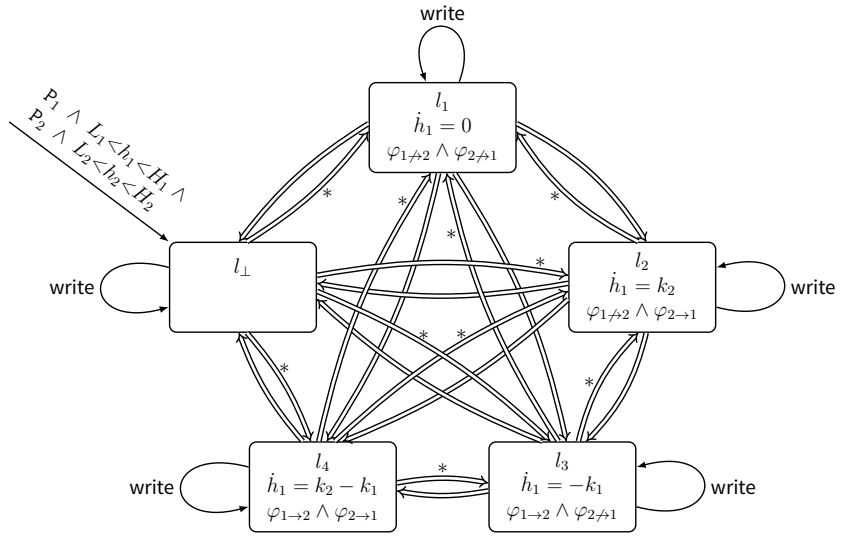


Figure 4.5.: Hybrid automaton model of the plant dynamics for tank U_1.

- *Inv*$(l_\perp) =$ *true* and *Inv*$(l_i) = [\![c_i]\!]$ for $i = 1, \ldots, n$,
- *Init* $= \{(l_\perp, \nu_0)\}$ with ν_0 assigning initial values to the variables, and
- *Urg*$(l) = 0$ for all $l \in$ *Loc*, and for $e \in$ *Edge* with source l and target l' we set *Urg*$(e) = 1$ if $l = l_\perp$ or $l = l_i$, $l' = l_j$ and $i > j$, and *Urg*$(e) = 0$ otherwise.

[a] Constant values for discrete variables from $V_{act} \cup V_{sen}$ are assured by the cycle synchronization automaton, but in our implementation we include these conditions also here.

Example 4.5.1

The plant dynamics for the first tank in our running example is modeled by the hybrid automaton in Figure 4.5. If we additionally eliminate non-convex invariants, we get a total of 17 locations for the specification of the dynamics of the first tank, and 49 for both tanks (after eliminating locations with *false* invariants).

4.6. Parallel Composition

The hybrid automaton model for an SFC controlled plant is the parallel composition of the HA models for the SFC controller, the PLC cycle synchronization, and the plant dynamics. Due to the parallel composition, the models can be very large. Though some simple techniques can be used to reduce the model size, the remaining model

might still be too large to be analyzed. For example we can remove from the model all locations with *false* invariants, transitions between locations whose invariants do not intersect when projected to the plant's sub-space, and non-initial locations without any incoming transitions.

4.7. Outlook

In this chapter, we showed how SFCs controllers, their cyclic execution on a PLC and the dynamic behavior of a plant can be modeled by hybrid automata. This transformation to HA models allows the analysis of the SFC-controlled plant. However, the resulting models can be very large, even for small systems. Therefore, in the following chapters we propose a CEGAR verification framework that allows to handle larger systems.

5. Counterexamples

CEGAR based analysis highly depends on the generation of counterexamples. For discrete and even timed automata there is a variety of tools that provide counterexamples if a given set of critical states is found to be reachable. However, for hybrid automata the reachability problem is undecidable in general. Nevertheless, a lot of effort has been spent on the development of reachability analysis techniques for hybrid automata, driven by the fact that most hybrid systems in industrial context are safety-critical. State-of-the-art tools like SpaceEx [FLD+11] and Flow* [CÁS13] compute an over-approximation of the reachable state space and can therefore be used to prove safety, i. e., that a given set of unsafe states cannot be reached from a set of initial states in a given model. However, if the over-approximation of the reachable states contains unsafe states then no conclusive answer can be given.

Counterexamples in form of system runs leading to unsafe states would be extremely valuable, even if they are *spurious*, i. e., if they were considered in the analysis but are not possible in the given model. For safe models they could help to reduce the approximation error in the analysis efficiently, whereas for unsafe models they could provide important information about the source of the critical system behavior. Counterexamples would enable the application of counterexample-guided abstraction refinement (CEGAR) techniques and could also play an important role in controller synthesis.

Unfortunately, at the time this work was done none of the available tools for hybrid automata reachability analysis with continuous time domain compute counterexamples. It is surprising since *internally* they possess sufficient information to generate at least a coarse over-approximation of a counterexample in form of a sequence of jumps (i. e., changes in the discrete part of the system state), augmented with time intervals over-approximating the time durations between the jumps. In this chapter, we examine whether it is possible to either use *augmented* system models or to extract information from the *output* of the analysis tool SpaceEx such that we can *synthesize* over-approximations of counterexamples. Afterwards, we study how the *efficiency* can be improved by extending the functionality of the tool Flow* *internally*, i. e., by making modifications to the source code. Finally, we develop a simulation-based approach to *validate* the counterexample over-approximations, i. e., to determine unsafe paths in the over-approximation.

We have chosen SpaceEx and Flow* for our experiments because on the one hand SpaceEx is one of the most popular hybrid automata reachability analysis tools and on the other hand we collaborated with a member of the implementation team of Flow*, i. e., the modification of the source code of Flow* could be done safely. For all SpaceEx experiments, we used version v0.9.7c, the support function scenario

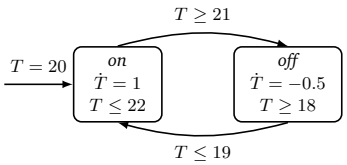


Figure 5.1.: The thermostat example.

with boxes, `chull` aggregation, and no clustering. For our FLOW* experiments, version 1.0.0 but with the proposed extensions has been used. All experiments have been performed on a i4 (3210M) machine with 8GB RAM running Ubuntu 12.04 LTS, 64-bit. Unfortunately, counterexample generation without tool extension is unsatisfactory: we need either expensive additional analysis runs for enlarged systems or parsing hidden information from debug output. The results demonstrate the need to extend the functionality of available analysis tools to generate counterexamples internally. However, even if that task is done, the results strongly over-approximate counterexamples, whose existence can be indicated but not proven. Thus we need novel methods to refine and validate the results.

As they are not central to the given problem, in this chapter we restrict ourselves to closed hybrid automata without urgent locations and jumps, and skip the label components and the urgency functions from the HA definitions; urgent locations and transitions can be eliminated by a semantically equivalent transformation as discussed in Chapter 7, but the proposed approaches could also be extended in a straightforward manner to cover urgency.

We use a toy example of a *thermostat* which is shown in Figure 5.1. It keeps the temperature of a room between $18°C$ and $22°C$. For further experiments, we use the *navigation benchmark* [FI04] (see Figure 5.2). It models an object moving in the $\mathbb{R}^2$ plane. The velocity (v_1, v_2) of the object depends on its position (x_1, x_2) in a grid. For some experiments we add a parameter ε to the navigation benchmark to enlarge the satisfaction sets of guards and invariants by replacing all upper bounds *ub* (lower bounds *lb*) by $ub + \varepsilon$ ($lb - \varepsilon$).

Outline The background information that is necessary to understand this chapter can be found in Sections 3.1 and 3.4. This chapter is structured as follows: In the Sections 5.1 and 5.2 we describe how over-approximations of counterexamples can be computed for unsafe models. Subsection 5.1.3 has been written by my colleague Xin Chen in our joint publication [2] and I used his text without marking it explicitly as a citation. The validation of possible spurious counterexamples is discussed in Section 5.3, before this chapter is concluded in Section 5.4.

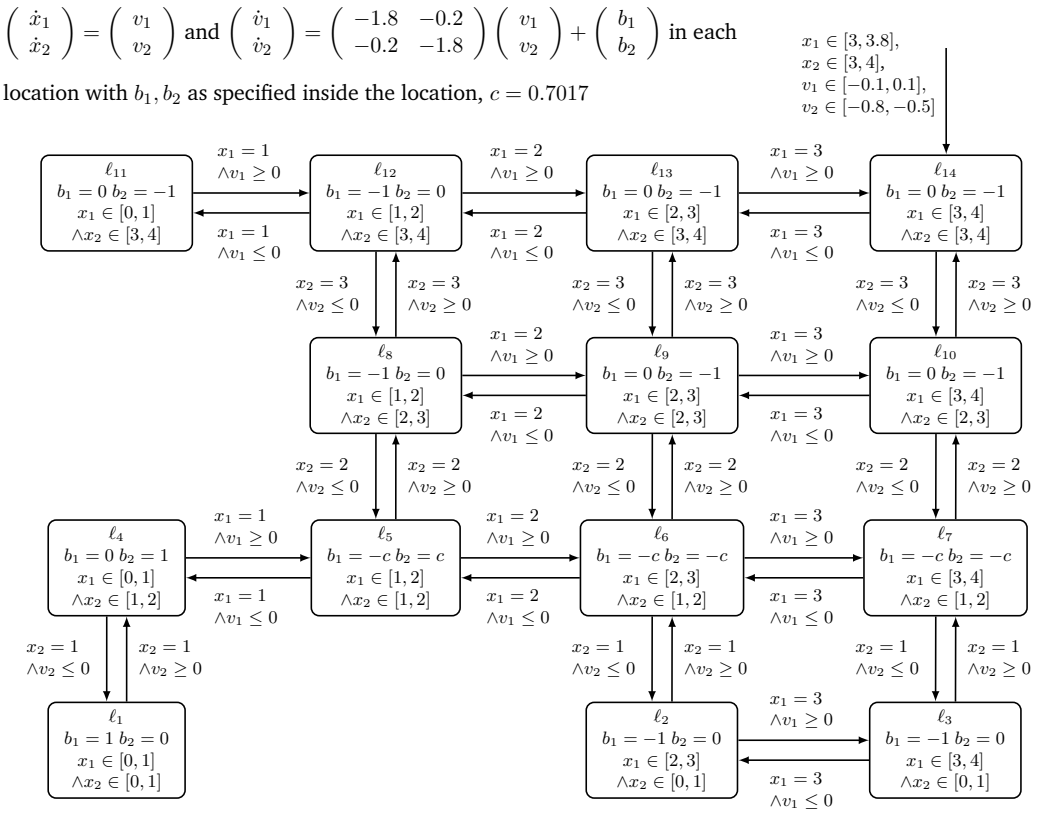


Figure 5.2.: The navigation benchmark.

5.1. Generating Traces for Presumable Counterexamples

Existing hybrid automata analysis tools like SpaceEx offer as output options either the computed over-approximation of the reachable state space, its intersection with the unsafe states, or just the answer whether unsafe states are reachable or not (in the over-approximation). However, in contrast to tools for discrete automata, at the time at which this work was done, none of the tools for hybrid automata provided counterexamples (latest developments for the Flow* tool led to the generation of possible counterexamples in form of counterexample traces and their computed flowpipes, following the third approach in this section).

In this section we show how a *trace* explaining the reachability of unsafe states can be computed. We present three different approaches: The first approach augments hybrid automata with auxiliary variables to make observations about the computation history of the analysis. The second approach can be used if the analysis tool outputs sufficient information about the paths that have been processed during the analysis. The third approach suggests to implement some new functionalities efficiently in existing tools.

5.1.1. Approach I: Model Augmentation

We extend the hybrid automaton model with new variables for book-keeping about traces that lead to unsafe states in the reachability analysis. First we augment the model and analyze the augmented system to observe the *number of jumps* until an unsafe state is reached. Then we augment and analyze an unrolled model to observe unsafe *traces*.

5.1.1.1. Determining the Counterexample Length

We augment the model and analyze it to gain information about the length of paths leading to unsafe states. We introduce a counter *tr* with initial value zero, define $\dot{tr} = 0$ in each mode, and let each jump increase the counter value by one.

However, the unboundedness of *tr* would render the fixed-point analysis to be non-terminating. To bound *tr* from above, we define a constant max_{tr} and either extend the invariants or the edge guards to forbid higher values.

The value of max_{tr} must be guessed, and in case the analysis of the augmented model reports safety, increased. A possible guess could be the number of iterations during the fixed-point analysis of the original model, which is reported by SpaceEx and specifies how many times the tool computed a (time+jump) successor of a state set. To get a smaller value (and thus shorter counterexamples with less computational effort), the reachability analysis could be stopped when an unsafe state is reached. Unfortunately, SpaceEx does not offer this option.

Definition 5.1.1 – Guard and Invariant Augmentation

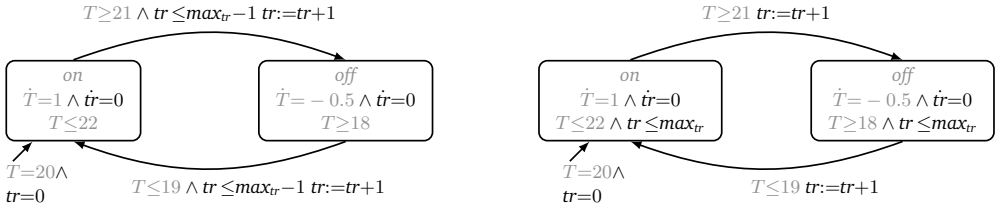
Let $max_{tr} \in \mathbb{N}$ and $\mathcal{H} = (Loc, V, Edge, Act, Inv, Init)$ be an HA.

The *guard augmentation* of the hybrid automaton $\mathcal{H}$ is the HA $\mathcal{H}_{guard} = (Loc, V', Edge', Act', Inv, Init')$ with

- $V' = V \cup \{tr\}$;
- $Edge' = \{(l, g', r', l') \mid (l, g, r, l') \in Edge\}$ with $g' = \{\nu \in \mathcal{V}^{V'} \mid \nu \downarrow_V \in g \wedge \nu(tr) \leq max_{tr} - 1\}$ and $r' = \{(\nu, \nu') \in \mathcal{V}^{V'} \times \mathcal{V}^{V'} \mid (\nu, \nu') \downarrow_V \in r \wedge \nu'(tr) = \nu(tr) + 1\}$, where $\downarrow$ denotes projection;
- $Act'(l)$ contains those functions $f : \mathbb{R}_{\geq 0} \to \mathcal{V}^{V'}$ whose projection to V are in $Act(l)$ and which satisfy $\dot{tr} = 0$, for each $l \in Loc$;
- $Init' = \{(l, \nu) \in Loc \times \mathcal{V}^{V'} \mid (l, \nu \downarrow_V) \in Init \wedge \nu(tr) = 0\}$.

The *invariant augmentation* of the hybrid automaton $\mathcal{H}$ is the HA $\mathcal{H}_{inv} = (Loc, V', Edge'', Act', Inv'', Init')$ with V', Act' and $Init'$ as above and

- $Edge'' = \{(l, g, r', l') \mid (l, g, r, l') \in Edge\}$ with r' as above;
- $Inv''(l) = \{\nu \in \mathcal{V}^{V'} \mid \nu \downarrow_V \in Inv(l) \wedge \nu(tr) \leq max_{tr}\}$ for each $l \in Loc$.



(a) Guard annotation.

(b) Invariant annotation.

Figure 5.3.: Guard and invariant augmentation for the thermostat model.

Table 5.1.: Evaluation of the guard and invariant augmentations with a sampling time of 0.1 and a time horizon of 30. The analysis of the original model without augmentation was limited to the specified number of iteration, i. e. the analysis was interrupted before a fixed-point was found. For this thesis, we recomputed the following experiment from [2] on a i5 (3210M) machine with 8GB Ram with Ubuntu 12.04 LTS, 64-bit with the current SpaceEx release (v0.9.8f).

model	augment.	max_{tr}	#iter.	fixed-point	running time [secs]	tr
thermostat example	none	-	5/11/31	no/no/no	0.13/0.40/1.24	-
	guard	4/10/30	5/11/31	yes/yes/yes	0.36/1.55/11.57	$[1,4]/[1,10]/[1,30]$
	invar.	4/10/30	5/11/31	yes/yes/yes	0.40/1.56/10.49	$[1,4]/[1,10]/[1,30]$
navigation benchmark	none	-	26/92/819	no/no/no	1.03/4.81/57.03	-
	guard	4/10/30	26/92/819	yes/yes/yes	1.34/6.51/157.50	$[4,4]/[4,10]/[4,30]$
	invar.	4/10/30	26/92/819	yes/yes/yes	1.43/6.17/138.69	$[4,4]/[4,10]/[4,30]$
navigation2 benchmark, $\varepsilon = 0.1$	none	-	27/318	no/no	4.35/125.26	-
	guard	4/10	27/318	yes/yes	4.25/153.40	$[4,4]/[4,10]$
	invar.	4/10	27/318	yes/yes	4.59/158.54	$[4,4]/[4,10]$

Figure 5.3 illustrates the augmentation on the thermostat example. Note that, apart from restricting the number of jumps, the above augmentation does not modify the original system behavior. The size of the state space is increased by the factor $max_{tr}+1$, since the value domain of tr is $[0, max_{tr}] \subseteq \mathbb{N}$ for the constant max_{tr}.

When we analyze the augmented model, SpaceEx returns for each mode in the over-approximated set of reachable unsafe states an over-approximation $[l, u]$ for the values of tr. Since tr takes integer values only, the lower and upper bounds are not approximated, i. e., during analysis both after l and after u (over-approximative) jump computations unsafe states have been reached, but we do not have any information about the values in between. Therefore we fix the number k, describing the length of counterexamples we want to generate, to be either l or u.

We made some experiments for the thermostat example with unsafe states $T \leq 19$, and for the navigation benchmark with unsafe states $(x_1, x_2) \in [1,2] \times [0,1]$. Table 5.1 compares for different max_{tr} values the number of SpaceEx iterations, the

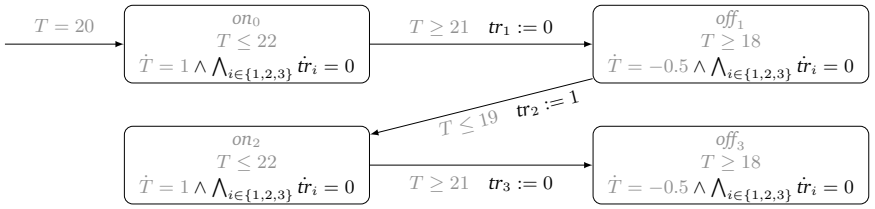


Figure 5.4.: Naive trace encoding of the thermostat example with depth 3.

running times, and the resulting *tr* values for the original models, the guard and the invariant augmentations. For the same number of iterations, the augmented models need in average some more (but still comparable) time for the analysis than the original models; the invariant and guard augmentations seem to be similar in terms of running time. Note that the augmented models find a fixed-point since the number of jumps is limited by max_{tr}. The original models do not find a fixed-point within the restricted number of iterations.

5.1.1.2. Trace Encoding

In order to observe the traces leading to unsafe states, we need to remember the jumps in the order of their occurrences. We achieve this by unrolling the transition relation of the original model k times, where k is the counterexample length determined in the previous step.

We could define the unrolling by copying each mode $k+1$ and each edge k times, and let the ith copy of an edge connect the ith-copy of the source mode with the $(i+1)$st copy of the target mode. To remember the jumps taken, we introduce k auxiliary variables $tr_1, \ldots, tr_k$ and store on the ith copy of an edge the edge's identity in tr_i. Such an unrolling would cause a polynomial increase in the size of the model.

However, in such an unrolling there might be different traces leading to the same mode. SpaceEx would *over-approximate* these trace sets by a mode-wise closure, such that we cannot extract them from the result. E. g. for two traces l_1, l_2, l_4, l_5, l_7 and l_1, l_3, l_4, l_6, l_7, respectively, also the trace l_1, l_2, l_4, l_6, l_7 would be included in the over-approximation. Therefore, we copy each mode as many times as the number of different traces of length up to k leading to it. This yields an exponential growth in k for the number of locations and transitions.

We augment the unrolled model to observe the traces to unsafe states. In a naive encoding we identify edges $e_1, \ldots, e_d$ by numbers $1, \ldots, d$, and introduce new variables tr_i for i=$1, \ldots, k$ to store the edges taken.

An example unrolled model for the thermostat with naive trace encoding is shown in Figure 5.4.

Definition 5.1.2 – k-unrolling

Assume an HA $\mathcal{H} = (\mathit{Loc}, V, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init})$ with an ordered set $\mathit{Edge} = \{e_1, \ldots, e_d\}$ of edges.
The *k-unrolling* of $\mathcal{H}$ is the HA $\mathcal{H}_u = (\mathit{Loc}_u, V, \mathit{Edge}_u, \mathit{Act}_u, \mathit{Inv}_u, \mathit{Init}_u)$ with

- $\mathit{Loc}_u = \bigcup_{i=1,\ldots,k+1} \mathit{Loc}^i$;
- $\mathit{Edge}_u = \{((l_1, \ldots, l_i),\, g,\, r,\, (l_1, \ldots, l_i, l_{i+1})) \mid 1 \leq i \leq k \wedge (l_i, g, r, l_{i+1}) \in \mathit{Edge}\}$;
- $\mathit{Act}_u((l_1, \ldots, l_i)) = \mathit{Act}(l_i)$ for all $(l_1, \ldots, l_i) \in \mathit{Loc}_u$;
- $\mathit{Inv}_u((l_1, \ldots, l_i)) = \mathit{Inv}(l_i)$ for all $(l_1, \ldots, l_i) \in \mathit{Loc}_u$;
- $\mathit{Init}_u = \{((l), \nu) \in \mathit{Loc}_u \times \mathcal{V}^V \mid (l, \nu) \in \mathit{Init}\}$.

We also define an advanced encoding which needs less auxiliary variables. If the domain of a variable includes $[0, |\mathit{Edge}|^n]$ for some $n \in \mathbb{N}$ then we can use it to store a *sequence* of n edges: Each time an edge is taken, we multiply the current value by $|\mathit{Edge}|$ and add the identity of the taken edge. This way we need $\lceil \frac{k}{n} \rceil$ auxiliary variables to encode a path of length k.

Definition 5.1.3 – Naive Trace Encoding

Assume an HA $\mathcal{H} = (\mathit{Loc}, V, \mathit{Edge}, \mathit{Act}, \mathit{Inv}, \mathit{Init})$ with an ordered set $\mathit{Edge} = \{e_1, \ldots, e_d\}$ of edges.
The *naive trace encoding of the hybrid automaton $\mathcal{H}$ with depth k* is the hybrid automaton $\mathcal{H}_1 = (\mathit{Loc}_u, V_1, \mathit{Edge}_1, \mathit{Act}_1, \mathit{Inv}_1, \mathit{Init}_1)$ with

- $V_1 = V \cup \{\mathit{tr}_1, \ldots, \mathit{tr}_k\}$;
- $\mathit{Edge}_1 = \{((l_1, \ldots, l_i), g, r_{i,j}, (l_1, \ldots, l_i, l_{i+1})) \mid 1 \leq i \leq k \wedge e_j = (l_i, g, r, l_{i+1}) \in \mathit{Edge} \wedge$
  $r_{i,j} = \{(\nu, \nu') \in \mathcal{V}^{V_1} \times \mathcal{V}^{V_1} \mid (\nu, \nu') \downarrow_V \in r \wedge \nu'(\mathit{tr}_i) = j \wedge \bigwedge_{m \neq i} \nu'(\mathit{tr}_m) = \nu(\mathit{tr}_m)\}\}$;
- $\mathit{Act}_1((l_1, \ldots, l_i))$ contains those functions $f : \mathbb{R}_{\geq 0} \to \mathcal{V}^{V_1}$ whose projection to V are in $\mathit{Act}(l_i)$ and that satisfy $\bigwedge_{j=1}^{k} \dot{\mathit{tr}}_j = 0$, for all $(l_1, \ldots, l_i) \in \mathit{Loc}_u$;
- $\mathit{Inv}_1((l_1, \ldots, l_i)) = \{\nu \in \mathcal{V}^{V_1} \mid \nu \downarrow_V \in \mathit{Inv}(l_i)\}$, for all $(l_1, \ldots, l_i) \in \mathit{Loc}_u$;
- $\mathit{Init}_1 = \{((l), \nu) \in \mathit{Loc}_u \times \mathcal{V}^{V_1} \mid (l, \nu \downarrow_V) \in \mathit{Init} \wedge \bigwedge_{j=1}^{k} \nu(\mathit{tr}_j) = 0\}$.

Let $n \in \mathbb{N}_{>0}$ such that $[0, d^n]$ is included in the domain of each tr_i and let $z = \lceil \frac{k}{n} \rceil$.

Table 5.2.: Evaluation of the naive and advanced trace encodings for the thermostat example and the navigation benchmark ($k = 4$, time step 0.1, time horizon 30) using π_1=$l_{14}, l_{13}, l_9, l_6, l_2$, π_2=$l_{14}, l_{10}, l_7, l_6, l_2$, π_3=$l_{14}, l_{10}, l_7, l_3, l_2$ and π_4=$l_{14}, l_{10}, l_9, l_6, l_2$.

model	trace	#locs	#trans	#vars	n	time [secs]	solutions
thermostat example	none	2	2	1	-	0.136	-
	naive	5	4	5	4	0.312	on, off, on, off, on
	adv.	5	4	2	4	0.147	on, off, on, off, on
navigation benchmark	none	14	36	5	-	1.372	-
	naive	81	80	9	3	1.777	$\pi_1; \pi_2; \pi_3; \pi_4$
	adv.	81	80	7	3	1.503	$\pi_1; \pi_2; \pi_3; \pi_4$

Definition 5.1.4 – Advanced Trace Encoding

Assume an HA $\mathcal{H} = (Loc, V, Edge, Act, Inv, Init)$ with an ordered set $Edge = \{e_1, \ldots, e_d\}$ of edges.

The *advanced trace encoding of the hybrid automaton $\mathcal{H}$ with depth k* is the HA $\mathcal{H}_2 = (Loc_u, V_2, Edge_2, Act_2, Inv_2, Init_2)$ with

- $V_2 = V \cup \{tr_1, \ldots, tr_z\}$;
- $Edge_2 = \{((l_1, \ldots, l_i), g, r', (l_1, \ldots, l_i, l_{i+1})) \mid 1 \leq i \leq k \wedge e_j = (l_i, g, r, l_{i+1}) \in Edge$
  $\wedge\, r' = \{(\nu, \nu') \mid (\nu, \nu') \downarrow_V \in r \wedge \nu'(tr_{\lceil i/n \rceil}) = \nu(tr_{\lceil i/n \rceil}) \cdot d + j$
  $\wedge \bigwedge_{j \neq \lceil i/n \rceil} \nu'(tr_j) = \nu(tr_j)\}\}$;
- $Act_2((l_1, \ldots, l_i))$ contains those functions $f : \mathbb{R}_{\geq 0} \to \mathcal{V}^{V_2}$ whose projection to V are in $Act(l_i)$ and which satisfy $\bigwedge_{j=1}^{z} \dot{tr}_j = 0$, for all $(l_1, \ldots, l_i) \in Loc_u$;
- $Inv_2((l_1, \ldots, l_i)) = \{\nu \in \mathcal{V}^{V_2} \mid \nu \downarrow_V \in Inv(l_i)\}$, for all $(l_1, \ldots, l_i) \in Loc_u$
- $Init_2 = \{((l), \nu) \in Loc_u \times \mathcal{V}^{V_2} \mid (l, \nu) \in Init \wedge \bigwedge_{j=1}^{z} \nu(tr_j) = 0\}$.

Note that depending on the chosen trace encoding, up to k auxiliary variables are added to the system.

Using our implementation for the proposed trace encodings in Table 5.2 we compare the model sizes and the analysis running times for the thermostat example and the navigation benchmark. Compared to the original model, the analysis running times for the trace encodings increase only slightly. The last column lists the computed traces, which are (as expected) the same for both encodings.

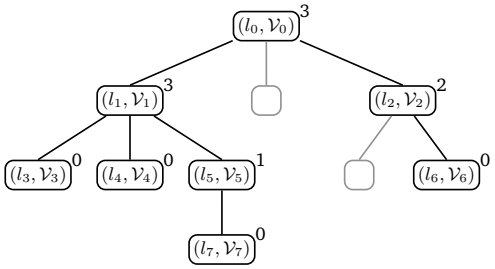


Figure 5.5.: SPACEEX search tree.

5.1.2. Approach II: Parsing the Output of SpaceEx

The approach introduced above does not scale for large systems, since the unrolling blows up the models too much. If the verification tool offers enough information about the analyzed system traces, it is perhaps also possible to extract from the tool's output the same information we gathered by system augmentation and additional analysis runs. We are interested in determining traces that led to unsafe states during the analysis, since they are candidates for presumable counterexamples. Without loss of generality, we assume that unsafe states are restricted to a single mode.

SPACEEX stores in a first-in, first-out (FIFO) list a sequence of *symbolic states*, each of them consisting of a mode and a valuation set, whose successors still have to be computed in the forward reachability analysis algorithm. This so-called *waiting list* contains initially each mode with its initial valuation set (if not empty). In each iteration the next element from the list is taken. Its flowpipe for a user-defined time horizon and all possible (non-empty) jump successors of the flowpipe segments are computed and those that were not yet processed are added to the list. As illustrated in Figure 5.5, this computation hierarchy corresponds to a tree whose nodes are processed in a breadth-first manner. Each node corresponds to a pair of a mode and a valuation set, which was found to be reachable. The upper indices on the nodes show the number of computed successor sets, whereas gray nodes in the Figure represent successors that are contained in another already processed set in the same mode and are therefore not added to the tree.

SPACEEX does not output the structure of this tree. However, using a verbosity of *Debug 2*, we can make use of more verbose console outputs to get additional informations.

- When an iteration starts, SPACEEX outputs a text from which we can extract the iteration number i ("`Iteration 5...`").
- SPACEEX starts the flowpipe computation and outputs the mode of the current symbolic state ("`applying time elapse in location loc()==l14`").
- The computation of jump successors follows, which is edge-wise. For each edge, whose source is the current mode, its label, source, and target is printed

("`applying discrete post of transition with label navigation.trans from location loc()==l14 to location loc()==l13`").

- SpaceEx determines which of the previously computed flowpipe segments intersect with the guard ("`found 1 intervals intersecting with guard`").
- The jump successors for the intersecting flowpipe segments are computed and, if not yet processed, put to the waiting list. Before switching to the next outgoing edge, some information on the computation time is given ("`Discrete post done after 0.098s, cumul 0.098s`").
- When all outgoing edges have been handled, the iteration is done. The following output gives us the total number of processed symbolic states and the current size of the waiting list ("`1 sym states passed, 2 waiting`").
- After termination of the analysis some general analysis results are printed, e. g., the number of iterations, whether a fixed-point has been found or not, the analysis time, and whether unsafe states are reachable.

If we would succeed to re-construct the search tree (or at least the involved mode components and their hierarchy) using the above information, we could extract traces that led to unsafe states in the tree.

The good news is that from the above outputs we can extract quite some information regarding the search tree, such that in some cases we can construct counterexamples. The bad news is that it is not sufficient to reconstruct all details. E. g. since the waiting list size is reported after each iteration, we can determine the number of new waiting list elements added during the last iteration (the new list size minus the old list size minus 1). If this number equals the total number of all intersecting intervals over all analyzed edges then we can determine the mode components of the waiting list elements. However, if some of the successors are already processed and therefore not added to the queue then we cannot know for sure which sets were added. For example, if out of two sets having the same mode component only one was added to the queue then we cannot know which of them. To avoid wrong guesses those cases are skipped and not considered further in our implementation.

Without model augmentation it is not possible to restrict the SpaceEx search to paths of a given length, therefore we cannot directly compare this method to the results of Table 5.2. We made experiments with the navigation benchmark using the verbosity *Debug 2* of SpaceEx. For 50 iterations with a computation time of 32.28 seconds, we found 11 traces leading to unsafe states in l_2. When considering only 25 iterations, the computation time is 12.69 seconds and only 4 traces are found. The increase of running time for using SpaceEx with verbosity level *Debug 2* instead of the default value *Medium* was negligible in our experiments.

A single analysis run suffices to extract traces of counterexamples thus this method seems to be superior to the augmentation approaches if the analysis tool communicates enough information about the system traces. However, if not all relevant details are accessible, not all traces can be rebuilt safely.

Table 5.3.: Trace generation using FLOW* ($k = 4$, time step 0.1, time horizon 30, π_1=$l_{14}\, l_{13}\, l_9\, l_6\, l_2$ π_2=$l_{14}\, l_{10}\, l_7\, l_6\, l_2$ and π_3=$l_{14}\, l_{10}\, l_7\, l_3\, l_2$).

model	**running time [secs]**	**solutions**
thermostat example	0.23	on, off, on, off, on
navigation benchmark	8.37	π_1, π_2, π_3

5.1.3. Approach III: Extending the Functionality of Flow*

Extracting information from the textual output of a tool is an overhead, since the information has already been computed during analysis. Moreover, it might be imprecise if we do not have access to all needed information.

Instead, we could generate counterexample traces on-the-fly by attaching to each symbolic state in the waiting queue the trace that led to it during the search. The waiting queue initially contains initial symbolic states, to which we attach their location components. Time successors are labeled as their predecessors. If we add a new symbolic state as a jump successor of another symbolic state, then we attach to the new state the label of the predecessor state extended with the jump whose successor the new state is. The reachability computation will stop when the tree is complete till depth k (the maximal jump depth). Next, FLOW* intersects each tree node with the unsafe set. If a non-empty intersection is detected, the tool dumps the trace corresponding to the label of the unsafe node.

To implement the above functionality, only minor changes had to be made in FLOW*, but it saves us the time of augmenting the system or parsing tool output. We made experiments in line with Table 5.2 for the thermostat example and the navigation benchmark. The results are shown in Table 5.3. Please note that FLOW* does not compute the trace l_{14}, l_{10}, l_9, l_6, l_2, which is spurious. We additionally analyzed the navigation benchmark with $k = 8$, where FLOW* generated 8 traces to unsafe states in l_2 with initial valuation $x_1 \in [3, 3.5]$, $x_2 \in [3, 4]$, $v_1 \in [-0.1, 0.1]$, and $v_2 \in [-0.8, -0.5]$.

5.2. Generating a Presumable Counterexample

In this section we present the generation of presumable counterexamples by extending the previously computed traces to *timed* traces. Given a trace, we compute a reduced model with the jumps of the trace only. This model is augmented with a clock *timer* and variables *tstamp*$_i$, $i = 1, \ldots, k$, one for each jump in the trace. The clock is initialized with 0 and has the derivative 1. Whenever a jump is taken, the clock value is stored in the *tstamp* variable of the jump and the clock is reset to 0. Figure 5.6 illustrates the above transformation.

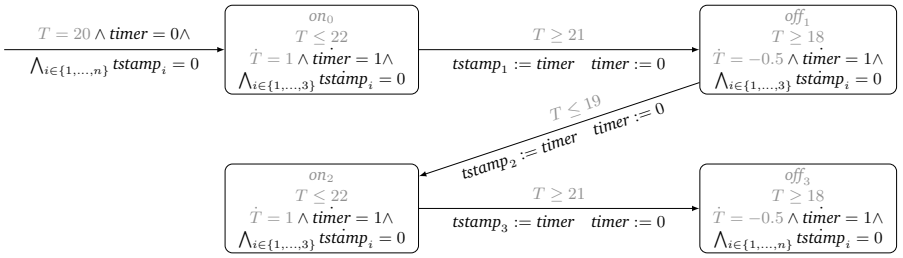


Figure 5.6.: Trace model of the thermostat example for $k = 3$.

Definition 5.2.1 – Trace Model

Assume a HA $\mathcal{H} = (Loc, V, Edge, Act, Inv, Init)$ and a finite trace $e_1, \dots, e_k$ of $\mathcal{H}$ with $e_i = (l_i, g_i, r_i, l_{i+1})$.

The *trace model of $\mathcal{H}$ and the trace $e_1, \dots, e_k$* is the hybrid automaton $\mathcal{H}' = (Loc', V', Edge', Act', Inv', Init')$ with

- $Loc' = \{(l_1, 1), \dots, (l_k, k), (l_{k+1}, k+1)\}$;
- $V' = V \cup \{timer, tstamp_1, \dots, tstamp_k\}$;
- $Edge' = \{((l_i, i), g_i, r_i', (l_{i+1}, i+1)) \mid i \in \{1, \dots, k\} \wedge r_i' = \{(\nu, \nu') \in \mathcal{V}^{V'} \times \mathcal{V}^{V'} \mid (\nu, \nu') \downarrow_V \in r \wedge \nu'(tstamp_i) = \nu(timer) \wedge \nu'(timer) = 0 \wedge \bigwedge_{j\neq i} \nu'(tstamp_j) = \nu(tstamp_j)\}\}$;
- $Act'((l_i, i))$ contains those functions $f : \mathbb{R}_{\geq 0} \to \mathcal{V}^{V'}$ whose projection to V are in $Act(l_i)$ and which satisfy $timer = 1 \wedge \bigwedge_{j=1}^{k} \dot{tstamp}_j = 0$, for all $(l_i, i) \in Loc'$;
- $Inv'((l_i, i)) = \{\nu \in \mathcal{V}^{V'} \mid \nu \downarrow_V \in Inv(l_i)\}$ for all $(l_i, i) \in Loc'$;
- $Init' = \{((l_1, 1), \nu) \in Loc' \times \mathcal{V}^{V'} \mid (l_1, \nu \downarrow_V) \in Init \wedge \nu(timer) = 0 \wedge \forall i = 1, \dots, k.\ \nu(tstamp_i) = 0\}$.

Another method to get timing information is as follows. Both in SpaceEx and in Flow*, the time horizon $[0, \Delta]$ of a flowpipe is divided into smaller time intervals $[0, \tau], [\tau, 2\tau], \dots, [(m-1)\tau, m\tau]$ with $m\tau = \Delta$. The flowpipe is computed as a union of flowpipe segments, one for each smaller interval. Thus the tools have internal information about the timestamps of the symbolic states in the waiting list. We make use of this fact and label the symbolic states in Flow* with the timed traces which lead to them. This way we get the timing information for free. Please note that this would also be possible for SpaceEx. In Flow* an additional backward refinement of the time intervals of the timed trace would be also possible, which we do not describe here.

Table 5.4.: Comparison of the timed traces for the navigation benchmark computed by SpaceEx and Flow*($k = 4$, time step 0.1, time horizon 30, traces from Table 5.2 and 5.3).

Initial states: l_{14}, $x_1 \in [3.0, 3.8], x_2 \in [3.0, 4.0], v_1 \in [-0.1, 0.1], v_2 \in [-0.8, -0.5]$

SpaceEx result in $6.46s$:

π_1 :	l_{14},	$[0.0, 0.6]$,	l_{13},	$[0.0, \mathbf{1.4}]$,	l_9,	$[1.5, \mathbf{1.8}]$,	l_6,	$[2.3, 2.5]$,	l_2
π_2 :	l_{14},	$[0.0, 1.9]$,	l_{10},	$[1.5, 1.9]$,	l_7,	$[0.2, 2.5]$,	l_6,	$[0.0, 2.4]$,	l_2
π_3 :	l_{14},	$[0.0, 1.9]$,	l_{10},	$[1.5, 1.9]$,	l_7,	$[2.3, 2.5]$,	l_3,	$[0.0, 1.0]$,	l_2
π_4 :	l_{14},	$[0.0, 1.9]$,	l_{10},	$[0.0, 0.6]$,	l_9,	$[0.9, 1.4]$,	l_6,	$[0.0, 0.0]$,	l_2

Flow* result in $8.37s$:

π_1 :	l_{14},	$[0.000, \mathbf{0.566}]$,	l_{13},	$[0.000, 1.420]$,	l_9,	$[\mathbf{1.531}, 1.880]$,	l_6,	$[\mathbf{2.421}, \mathbf{2.422}]$,	l_2
π_2 :	l_{14},	$[0.000, \mathbf{1.854}]$,	l_{10},	$[\mathbf{1.534}, \mathbf{1.836}]$,	l_7,	$[\mathbf{0.310}, \mathbf{2.371}]$,	l_6,	$[0.000, \mathbf{2.385}]$,	l_2
π_3 :	l_{14},	$[0.000, \mathbf{1.854}]$,	l_{10},	$[\mathbf{1.534}, \mathbf{1.836}]$,	l_7,	$[\mathbf{2.415}, \mathbf{2.416}]$,	l_3,	$[0.000, \mathbf{0.912}]$,	l_2

Table 5.4 shows some experimental results for the navigation benchmark. We compute timed extensions of the previously computed counterexample traces to build presumable counterexamples. The running times include for Flow* a complete reachability analysis up to jump depth 4, and for SpaceEx the generation of the traces with Approach II and extracting timing information by building and analyzing the trace models. Both tools have their advantages: SpaceEx computes the results faster, Flow* gives sometimes better refinements.

5.3. Simulation

To gain counterexamples, we identify suitable *candidate initial states (CISs)* from the initial state set and *refine* the timed trace separately for each CIS by restricting the timing informations.

Then we apply simulation to each CIS to find concrete counterexamples starting in the given CIS and being contained in the corresponding refined timed trace. Based on the refined timed trace of a CIS each jump can take place within a bounded but dense time interval. We let the simulation branch on a finite set of jump time points chosen from those intervals. The choice is guided by the invariant and guard satisfaction and uses a heuristic which iteratively discretizes time intervals with dynamic step sizes to drive the selection towards hitting conditions, e. g. in the presence of strict equations. Furthermore, the heuristic tries to abort simulation paths that do not lead to a counterexample as early as possible.

5.3.1. Finding Candidate Initial States

The task of identifying CISs for simulation is non-trivial, since the timed traces overapproximate counterexamples, such that not all initial states necessarily lead to un-

safe states within the given time bounds. W. l. o. g. we assume that the initial set is given as a hyperrectangle (otherwise we over-approximate the initial set by a hyperrectangle and use in the following the conjunction of the hyperrectangle with the initial set). We obtain CISs by applying a binary search on the initial set combined with a reachability analysis run to check whether the unsafe states are still reachable. As long as unsafe states are detected to be reachable from a hyperrectangle, the corner points of the hyperrectangle are added to the set of CISs. If in at least one dimension the width of the hyperrectangle is larger than a specified parameter ε, the interval is splitted (in this dimension) in the middle and both halves are analyzed again. The binary search stops if either the specified number of CISs are computed or if all hyperrectangles reach the minimal width in each dimension.

The current implementation could be improved by removing the corner points of those hyperrectangles from which no unsafe states are reachable from the set of CISs.

The user can choose between a depth-first search (DFS) and a breadth-first search (BFS) to generate CISs. DFS computes closely clustered points fast, BFS searches for widely spread points at a higher computation time.

For the trace $l_{14}\, l_{10}\, l_7\, l_6\, l_2$ of the navigation benchmark, our implementation needs 19ms to create the trace model. For the DFS, SpaceEx has to be run 42 times until 10 CISs are computed from which the unsafe state l_2 is reachable in the SpaceEx over-approximation. The corresponding computation time is 7.14s. The BFS finds the first 10 CISs within 29.90s and with 133 SpaceEx calls.

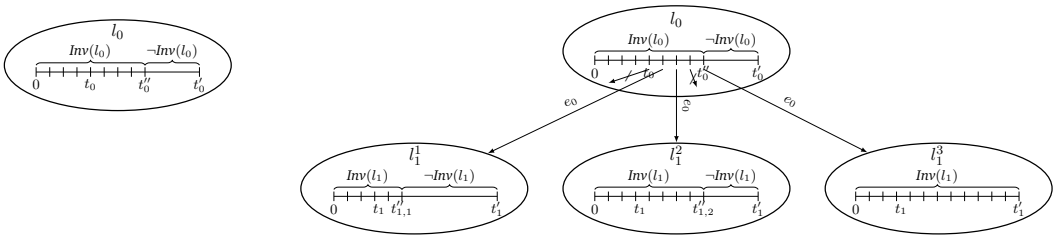
For each selected CIS we determine a refined timed trace using the same method as before for computing presumable counterexamples but now restricted to the given CIS as initial state.

5.3.2. Simulating the Dynamics

For linear differential equations the initial value problem is solvable, i. e., we can compute for each state the (unique) state reachable from it in time t. Thus, for linear differential equations we use the matrix exponential (e. g. in the homogeneous case, $\dot{x} = Ax$ is solved by $x(t) = x_0 e^{At}$, where t is the time and x_0 is the initial value of x), whereas for non-linear differential equations numerical methods (e. g. Runge-Kutta) can be used. However, since either exponential function values must be determined or numerical methods are used, the computation is not always exact.

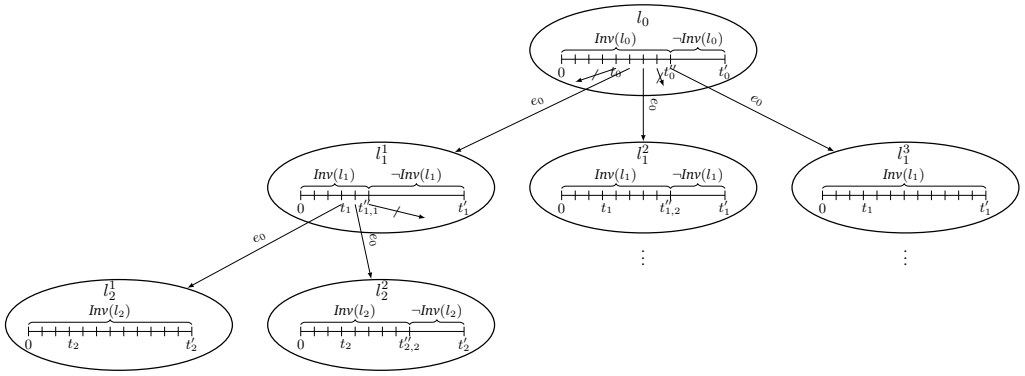
5.3.3. Checking Invariants

Along a simulated timed trace, time can pass in a location only as long as the location's invariant is satisfied. The timed trace provides us for each location l_i a time interval $[t_i, t_i']$, within which a jump to a successor location should be taken. We have to assure that the invariant is constantly fulfilled from the time where a location was entered till the time point where the jump is taken. Therefore, we compute the time successors for a set of sample time points homogeneously distributed (with



(a) Checking the invariant up to time t_0'.

(b) Taking the enabled edges during the time interval from t_0 to t_0'' to l_1.



(c) Expanding the next level.

Figure 5.7.: Simulation.

a user-defined distance) within the time interval $[0, t_i']$. We check the invariant for those sample time points in an increasing order. If the invariant is violated at a sample time point $t \in [0, t_i']$, no further time can elapse in the current location. Thus all simulation paths via time points from $[t, t_i']$ are cut off and the time interval for jumps is restricted to $[t_i, t_i'']$, where t_i'' is the time point before t.

5.3.4. Dynamic Search for Suitable Jump Time Points

Non-determinism (at which time point a jump is taken) is handled by branching the simulation for those previously selected sample time points that lie inside $[t_i, t_i']$. If the edge's guard is fulfilled at a given sample, the jump successor is computed and the corresponding simulation branch is explored in a depth-first search. The first steps of a simulation are shown in Figure 5.7.

The naive discretization of the dense time intervals has sometimes problems to hit guard conditions. Especially hard for the simulation are guards containing equations. To allow simulation for guards defined by equations, we enlarge the model behavior be replacing the guard equations by inequations, allowing values from a small box around a point instead of hitting the point exactly.

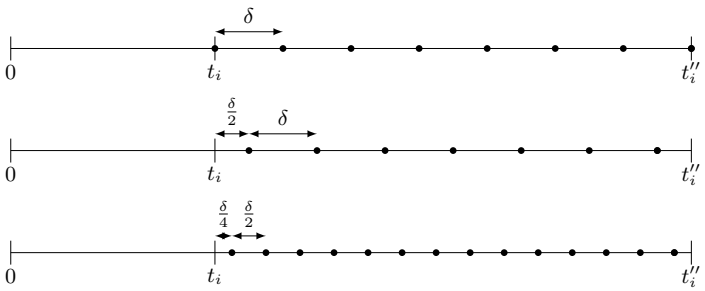


Figure 5.8.: Adaptive-step-size simulation.

However, even with such an enlarging it can happen that the guard is not fulfilled at any of the selected sample time points, or from the states after the jump no counterexamples can be simulated. In this case we dynamically determine new sample time points for the jump as follows. We use two parameters, an offset and a step size, specifying a set $\{t_i + \mathit{offset} + j \cdot \mathit{stepsize} \in [t_i, t_i'] \mid j \in \mathbb{N}\}$ of sample points. Initially (as described above), the offset is 0 and the step size has the value δ. If the simulation for these sample time points does not succeed, we set the offset to $\delta/2$ and let the step size unchanged. If those points are also not successful, we iteratively cut in half both parameter values. This adaption terminates if either the target location of the timed trace is reached (i. e., a counterexample is found) or the step size reaches some predefined lower bound. The dynamic step-size-adaption is visualized in Figure 5.8.

If a single counterexample suffices, the simulation can be stopped as soon as the unsafe location was reached. However, by heuristically searching for further counterexamples, it is also possible to provide additional information about a counterexample: Instead of the time points of the jumps along the simulation path the widest time intervals can be computed, such that the unsafe state is still reachable.

Table 5.5 shows the simulation results for some timed traces, each with a single initial state. Note that we find counterexamples (i. e., we reach the maximal jump depth) only in the two middle cases. We additionally run SpaceEx analyses with the given initial point for the first trace and could not reach the bad state with a time step of 0.001, i. e., the first timed trace is spurious. The last trace was not computed by Flow* and is therefore also spurious.

Note that for an enlarged ε-environment that are considered for guards and invariants, the over-approximation increases. As a result, more paths can be computed such that for trace π_3 the number of counterexamples and for the traces π_1 and π_4 the maximal reached jump depth increases.

Table 5.5.: Simulation results for the navigation benchmark with ε-enlarging
$\pi_1 = l_{14}, [0.0, 0.2], l_{13}, [0.0, 0.4], l_9, [1.5, 1.6], l_6, [2.4, 2.5], l_2$
with initial state $(3.0084472656250005, 3.21875, -0.1, -0.8)$
$\pi_2 = l_{14}, [1.834, 1.835], l_{10}, [1.779, 1.78], l_7, [1.934, 1.936], l_6, [0.511, 0.514], l_2$
with initial state $(3.2, 4.0, 0.1, -0.5)$
$\pi_3 = l_{14}, [0.000, 0.001], l_{10}, [1.569, 1.570], l_7, [2.429, 2.431], l_3, [0.514, 0.517], l_2$
with initial state $(3.8, 3.0, -0.1, -0.8)$
$\pi_4 = l_{14}, [0.0, 0.1], l_{10}, [0.0, 0.5], l_9, [1.0, 1.3], l_6, [2.3, 2.5], l_2$
with initial state $(3.0125, 3.0, -0.1, -0.8)$.

timed trace	step size	ε	maximal jump depth	unsafe	time [secs]
π_1	0.0005	0.0005	2	0	6.34
	0.05	0.5	3	0	1.92
π_2	0.0005	0.5	4	96	4.95
π_3	0.0005	0.0005	4	50	3.34
	0.0005	0.05	4	96	4.72
π_4	0.0005	0.0005	2	0	4.79
	0.05	0.05	3	0	0.66

5.4. Outlook

The approach to find presumable counterexamples for hybrid automata based on existing reachability tools can be further improved. Possible improvements are (1) a backward refinement of the time intervals on timed paths, (2) a rigorous simulation technique for hybrid automata, (3) giving a better heuristics to select the initial points for simulation, and (4) using several tools and take the best results to minimize the overestimation in a presumable counterexample. Preliminary results suggest that the function calculus of the tool ARIADNE [BCC+06] could be also embedded to improve the counterexample validation.

6. CEGAR-based Verification I: Bounded Model Checking

Our first CEGAR approach for the reachability analysis of SFC controlled plants implements a refinement loop that alternates between a discrete and a hybrid analysis. First, a bounded model checker analyses the discrete control program without constraining the inputs and outputs that connect the PLC program with the plant environment. Its discrete counterexamples trigger a hybrid reachability analysis which checks their feasibility with respect to the hybrid plant model. Once the hybrid reachability analysis has proven a counterexample to be spurious, it is excluded from the discrete analysis, and the search for the next discrete counterexample proceeds.

While this approach requires a multitude of hybrid reachability queries, each of them may operate on a considerably simpler hybrid automaton model where discrete control is restricted to operate along the discrete counterexample trace.

The naive approach where the discrete counterexample is returned as an explanation will not bring much benefit. However, when combining the approaches from this chapter and from Chapter 7 and exploiting some properties of our plant model, we expect good performance.

Outline The preliminaries for this chapter are described in Section 3.5, 3.4, and 3.6. This chapter starts with the discrete analysis of the controller in Section 6.1 which has mainly been written by Martin Neuhäußer for our joint publication [5]. The reachability analysis of discrete counterexamples including the model generation and the explanations for the discrete analysis is given in Section 6.2. This part was written by both Kai Driessen and me for [5]. Some enhancements for this methodology that originates from my writing can be found in Section 6.3. An outlook is given in Section 6.4.

6.1. Discrete Analysis

Following the description in Section 3.5, we assume a symbolic representation of the SFC control program and a set of safe SFC states given by $\mathit{Prog} = (I, \mathit{Tr}, \mathit{Prop})$. The formula *Prop* defines the safe states. The formula *Tr* logically encodes one PLC cycle, i. e., one reading, one SFC computation block, and one writing. Assuming a set $V_{\mathcal{C}}$ of SFC variables, the solutions μ of $\mathit{Tr}_{i,i+1}$ specify state pairs (ν_i, ν_{i+1}) with $\nu_i(x) = \mu(x_i)$ and $\nu_{i+1}(x) = \mu(x_{i+1})$ for all $x \in V_{\mathcal{C}}$, such that ν_{i+1} is reachable from ν_i via the execution of one PLC cycle.

This encoding is used to generate bounded model checking formulas

$$\varphi_k^{BMC} = I_0 \wedge \bigwedge_{i=0}^{k-1} Tr_{i,i+1} \wedge \neg Prop_k$$

for increasing path length $k = 0, 1, \ldots$, such that φ_k^{BMC} is satisfiable if and only if there is a path of the SFC program consisting of k complete cycles and reaching an unsafe state.

At this level, we abstract away from the plant behavior and assume that arbitrary input values from the PLC environment are possible, with one exception. At the end of each cycle the actuator outputs are written to the environment, and at the beginning of the next cycle the states of the actuators and sensors are read. We assume that the actuators are controlled according to the output immediately, therefore the read actuator state will have the previously written actuator values. Therefore, in the following we focus on writing the actuator control values and reading the sensor values.

Let $V_{act} = (act^1, \ldots, act^n)$ be the SFC output variables for controlling the actuators, and let the SFC input variables $V_{sen} = (sen^1, \ldots, sen^m)$ store the sensor values at the beginning of each cycle. Then each solution μ of φ_k^{BMC} specifies an input-output-trace

$$trace(\mu) = sen_0, act_1, sen_1, act_2, sen_2, \ldots$$

with input $sen_i{=}(\mu(sen_i^1), \ldots, \mu(sen_i^m))$ read at the beginning and with the output $act_{i+1}{=}(\mu(act_{i+1}^1), \ldots, \mu(act_{i+1}^n))$ written at the end of cycle i.

Example 6.1.1

In our plant example, for simplicity let us focus on the sensor inputs $sen^1 = full_1$, $sen^2 = high_1$, $sen^3 = low_1$, $sen^4 = empty_1$, $sen^5 = full_2$, $sen^6 = high_2$, $sen^7 = low_2$, and $sen^8 = empty_2$, and the actuator outputs $act^1 = \mathcal{P}_1$ and $act^2 = \mathcal{P}_2$ for the two pumps. A possible counterexample that spans three cycles of the controlling SFC is as follows:

$$\underbrace{(0,0,0)}_{full_1}, \underbrace{(0,0,0)}_{high_1}, \underbrace{(1,1,0)}_{low_1}, \underbrace{(0,0,1)}_{empty_1},$$
$$\underbrace{(0,0,0)}_{full_2}, \underbrace{(0,0,0)}_{high_2}, \underbrace{(1,1,0)}_{low_2}, \underbrace{(0,0,1)}_{empty_2},$$
$$\underbrace{(1,1,1)}_{\mathcal{P}_1}, \underbrace{(1,0,1)}_{\mathcal{P}_2}.$$

Once a discrete counterexample μ has been discovered by the BMC algorithm, $trace(\mu)$ is used in a hybrid reachability analysis to check whether it is also feasible in the hybrid plant model. More precisely, we use $trace(\mu)$ to drive the discrete

control in the hybrid plant model during the search for unsafe states. If an unsafe state is reachable according to the hybrid analysis, the possible safety violation is reported and the algorithms terminates. Otherwise, the discrete counterexample is spurious. This occurs if the sequence $trace(\mu)$ of input and output values produced by the discrete model checker is infeasible according to the dynamic behavior of the hybrid plant model.

Once a spurious discrete counterexample has been discovered, we construct an explanation formula ψ from $trace(\mu)$ that excludes the spurious trace. Note that depending on the information obtained from the hybrid analysis, ψ may exclude many discrete counterexamples at once; in particular, ψ can rule out a sub-trace of $trace(\mu)$ that is already infeasible in itself. Finally, the discrete analysis continues after additionally asserting ψ.

Once all counterexamples of at most k steps have been discharged, we increase bound k by one and continue searching for longer counterexamples.

6.2. Hybrid Analysis

6.2.1. Model Generation

The hybrid model for the discrete counterexample trace $trace(\mu)$ is the parallel composition of the dynamic plant model, one hybrid automaton modeling the trace $trace(\mu)$, and one automaton to model the PLC cycle synchronization. The latter two automata replace the SFC model and the PLC synchronizer model, which were used in the first approach to build a global model of the controlled plant.

In contrast to the later approaches, the HA models can be reduced: Firstly, we do not model the whole PLC execution, but just a single trace of it. Secondly, to achieve smaller models, we model the writing at the end of a PLC cycle i and the reading at the beginning of the next cycle $i + 1$ as an atomic step. We can do so because they happen instantaneously and, as explained above, we neglect the reading of the actuator values such that the writing and the reading become independent. As such, also their order does not matter. Consequently, we can model them using a single jump, having as guard the condition that the values of the sensors are sen_i and as reset function setting the actuator state to act_{i-1}. To synchronize with the plant model on this read-write jump, we use the label io.

As a last reduction, we also avoid the explicit modeling of sensors. Instead, we encode the physical meaning of a sensor s via a predicate $\gamma(s)$. In the case of water level sensors, a true sensor value means that the water level is at least as high as the sensor position, a false value means that the water level is lower than the sensor position.

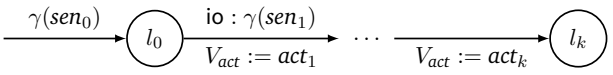


Figure 6.1.: Hybrid automaton for a discrete counterexample trace.

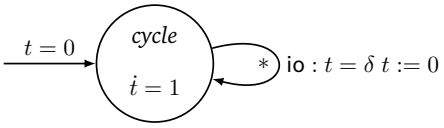


Figure 6.2.: Hybrid automaton for PLC cycle synchronization.

Example 6.2.1

In our two-tanks example we have the following sensor encodings:

$$\begin{aligned} \gamma(full_i) &= h_i = F_i \\ \gamma(high_i) &= h_i \geq H_i \\ \gamma(low_i) &= h_i \geq L_i \\ \gamma(empty_i) &= h_i = 0. \end{aligned}$$

Using this semantic encoding, we define for the sensor value vectors sen_i in our traces:

$$\gamma(sen_i) = \bigwedge_{j=1}^{m} (sen_i^j \Leftrightarrow \gamma(sen^j))$$

The resulting models are shown in the Figures 6.1 and 6.2.

Example 6.2.2

If the value of the sensor $sen^1 = low_1$ is 1 and the value of the sensor $sen^2 = high_1$ is 0 then the sensor value vector $(1, 0)$ would evaluate to $\gamma((1, 0)) = (h_1 \geq L_1 \wedge h_1 < H_1)$.

The dynamic plant behavior is modeled as shown in Figure 4.4 on page 41 with the only modification that the synchronization label write is replaced by the read-write label io.

The parallel composition of the three hybrid automata yields the model for the hybrid analysis. Before applying reachability analysis, we apply the same transformations as proposed in Chapter 7 to eliminate urgent locations and jumps to avoid Zeno behavior, etc.

6.2.2. Hybrid Analysis

For the reachability analysis of the hybrid model we again use SpaceEx, however, in this approach we are not limited to a special scenario for the analysis. If reachability of the unsafe state set is detected, the original tool provides only the intersection of the reachable states with the unsafe states. We use a slightly modified version of SpaceEx that outputs the complete set of reachable states independently of the outcome of the analysis.

In our experiments we will use the *Support Function Scenario*. We restrict the analysis to the set of states reachable in the time interval $[0, k\delta]$, where k is the length of the counterexample. The flowpipes are constructed using a sampling time t_s.

The reachability analysis for a given counterexample trace of length k has two possible outcomes: either the last location l_k of the counterexample trace model (see Figure 6.1) is reachable or not. If it is reachable then we have found a possible violation of the safety property. Note that due to the over-approximation of the reachability analysis for hybrid systems the detected counterexample might be spurious. Otherwise, if l_k is not reachable then the counterexample trace is not realizable, i. e. the dynamic plant behavior prevents the system from the given communication trace between controller and plant. In this case we compute an *explanation*.

6.2.3. Generating Explanations

An explanation is an unrealizable part of an unrealizable discrete counterexample trace, which allows the discrete BMC analysis to ignore solutions with such subtraces in its further search. To exclude as many unrealizable solutions as possible, we aim at computing possibly small explanations. In the current implementation we do not yet have any smart explanation generation implemented: we exclude the whole unrealizable trace, corresponding to the longest path in the search tree. However, unrealizability is usually due to a sub-set of the sensors and actuators only. Also, the initial states are not always necessary requirements for trace unrealizability. It will be the content of future work to find better explanation generation techniques.

6.3. Enhancements

The current implementation of the modular verification is a prototype and can be improved significantly.

Currently, for each discrete BMC counterexample we build the hybrid model described above, perform reachability analysis on it, generate an explanation (in case of unrealizability), and continue the BMC search. Due to the overhead for rebuilding models and re-starting reachability analysis from scratch, the current implementation is far from being efficient. However, with a few enhancements, this methodology has the potential of being highly efficient.

Currently, upon each call a model is generated and parsed by SpaceEx. Instead, it is possible to extend the model step-wise according to new traces, similar to the refinement step in the second CEGAR method in Chapter 7, or even to use the fully concretized model and guide the search along the currently analyzed trace internally. The implementation would benefit not only from reduced model generation and parsing effort, but also from re-using previous reachability analysis runs.

The application scenario of chemical plants with a fixed cycle time of δ allows further improvements concerning the reachability analysis. Since the discrete control is fixed for the duration of a PLC cycle, mode switches within the current cycle are only possible due to changes in the physical quantities. Thus, within a PLC cycle, the reachability analysis can be restricted to non-synchronizing jumps in the plant model. After the cycle time has passed, the communication phase with the environment takes place and leads to a set of mode changes via synchronizing jumps. These changes will lead to a more efficient reachability analysis, as the number of jumps that we need to consider will be reduced remarkably.

Given an unrealizable counterexample μ, the corresponding explanation is added to the formula that is used by the BMC algorithm. If the explanation contains the whole counterexample trace, the explanation can be used for the current BMC iteration only, i. e., we do not need to add it to Ψ (see Algorithm 2). If an explanation involves the initial but not the last state, it will be useful also for later iterations. Explanations computed in a BMC iteration k involving the last state but not the initial state cannot be used unmodified in later BMC iterations. However, we could shift them by one time instance and re-use them in the next iteration $k + 1$. Finally, if an explanation involves neither the initial nor the last state, we could apply shifting to exclude such trace segments in general at all path positions. More details on this BMC acceleration technique can be found in [Str04].

Last but not least, good explanations are at the heart of an efficient implementation. The currently generated unrealizable trace prefixes should be further analyzed to find smaller explanations. E. g., we could try to determine whether the trace remains unrealizable when relaxing the conditions on the initial states. Additionally, we could try to drop certain assumptions about the sensor values to further reduce the explanation.

6.4. Outlook

In this first CEGAR approach we extract the discrete behavior of the controller models. We make use of efficient analysis tools for discrete models and restrict the reachability analysis on the hybrid model to the control sequences that are given by the discrete counterexamples. Some experimental results will be presented in Chapter 9. Although currently our prototypical implementation is not competitive concerning the running time, we presented several ideas to improve this methodology. We strongly believe that this will remarkably enhance the running time of this second approach.

7. CEGAR-based Verification II: SpaceEx Integration

In this chapter, we explain our second CEGAR approach for the verification of plants that are controlled by SFCs. Our method could also be adapted to other kinds of applications for hybrid systems, e. g. to cyber-physical systems.

Given a concrete hybrid model for the controlled plant, we build (and iteratively refine) an abstraction that over-approximates the behavior of the concrete model and analyze it with over-approximative reachability analysis techniques. Therefore, if unsafe states are not found to be reachable in the abstraction then we know that unsafe states are neither reachable in the concrete model. However, if the analysis detects a fully refined counterexample path in the abstraction, we do not know whether the concrete model is unsafe or whether the counterexample was found due to the over-approximative property of the reachability analysis and is therefore spurious.

One of the main barriers in the application of CEGAR in the reachability analysis of hybrid systems is the complete re-start of the analysis after each refinement. To overcome this problem, we propose an *embedding of the CEGAR approach into the HA reachability analysis algorithm*: our algorithm refines the model on-the-fly during analysis and thus reduces the necessity to re-compute parts of the search tree that are not affected by the refinement. Besides this advantage, our method also supports the handling of *urgent locations* and *urgent transitions*, which is not supported by most of the HA analysis tools. Last but not least, our algorithm can be used to extend the functionalities of currently available tools to generate (at least presumed) *abstract counterexamples*.

Outline The background information that is needed to understand this chapter has been presented in the Sections 3.1, 3.2, 3.3, 3.4, and 3.6. In this Chapter, we explain our abstraction and how a model can be refined if a counterexample has been detected in Section 7.1. Afterwards, we show in Section 7.2 how we integrated a CEGAR-loop into an existing tool for the reachability analysis of hybrid automata. A conclusion for this second CEGAR-based analysis approach is given in Section 7.3.

7.1. Abstraction and Abstraction Refinement

The basis for a CEGAR approach is the abstraction technique and the generation and usage of counterexamples to refine the abstract model. We first explain the mechanism for the abstraction and the model refinement before we describe how we embed the refinement into the reachability algorithm to avoid restarts.

7.1.1. Abstraction

Intuitively, for the method described in this chapter, the abstraction of the hybrid model of an SFC-controlled plant consists of removing information about the plant dynamics and assuming arbitrary behavior instead. Initially, the dynamics is assumed to be completely arbitrary; the refinement steps add back more and more information. The idea is that the behavior is refined only along such paths on which the controller's correctness depends on the plant dynamics. Therefore, the abstraction level for the physical quantities (plant variables) of the plant will depend on the controller's state.

Let $\mathcal{H}$ be the HA composed from the PLC-cycle-synchronizer and the SFC model *without* the plant dynamics. The abstraction level is determined by a function *active* that assigns to each pair (l, x_i) of a location l in $\mathcal{H}$ and a variable $x_i \in V_{cont}$ of the plant model a subset of the conditional ODEs for variable x_i. For a given pair (l, x_i), the abstraction level specifies the currently considered dynamics $\mathit{active}(l, x_i) = \{(c_{l,i,1}, \mathrm{ODE}_{l,i,1}), \ldots, (c_{l,i,k_{l,i}}, \mathrm{ODE}_{l,i,k_{l,i}})\}$ for variable x_i in location l, where the conditional ODEs with lower indices have higher priorities. The global model of the controlled plant will define x_i to evolve according to ODE_j if the conditions $\neg c_1, \ldots, \neg c_{j-1}, c_j$ hold, and to evolve arbitrarily if none of the conditions $c_{l,i,1}, \ldots, c_{l,i,k_{l,i}}$ holds. A refinement step *extends* some of the sets $\mathit{active}(l, x_i)$ by adding new conditional ODEs to some variables in some locations.

7.1.2. Counterexample-Guided Abstraction Refinement

The refinement is counterexample-guided. If the analysis of the abstract model can show that no unsafe states are reachable in the abstraction then the concrete model is also provably safe. Otherwise, the search tree of the reachability analysis contains an abstract (symbolic) counterexample path $(\hat{l}_0, \mathcal{V}_0) \ldots (\hat{l}_k, \mathcal{V}_k)$. Recall that $\hat{l}_i = (l_i, l_{i,1}, \ldots, l_{i,n})$ is a location in the parallel composition. l_i is the location of the controller and synchronizer, whereas the remaining locations originate from the plant models for the physical quantities (cf. Definition 4.5.1). Thus, location $l_{i,j}$ represents that the dynamics for x_j follows ODE_j in location l_i for $l_{i,j} \leq k_{l,i}$ and $l_{i,j} = k_{i,j}$ represents the case that none of the ODE conditions $c_1, \ldots, c_{k_{l,i}}$ holds and that the dynamics can therefore be arbitrary. $\mathcal{V}_i$ are sets of valuations such that $\mathcal{V}_k$ has a non-empty intersection with the unsafe states, meaning that there might exist a concrete (explicit) path of the concrete model leading to an unsafe state. Note that, since the reachability analysis is over-approximative, we can state that such

abstract counterexamples are *presumed*, but in general we cannot decide if they are concretizable or spurious. In our refinement strategy, we choose the first presumed counterexample that is detected during the analysis using a BFS, i. e. we find *shorter* presumed counterexamples first. However, other heuristics are possible, too.

We use presumed counterexamples, carrying information about the visited locations and valuation sets, to guide the abstraction refinement process. If we wanted to use some refinement heuristics that requires more information (e. g. discrete transitions taken or time durations spent in a location), we could further *annotate* the search tree nodes with additional bookkeeping about the computation history.

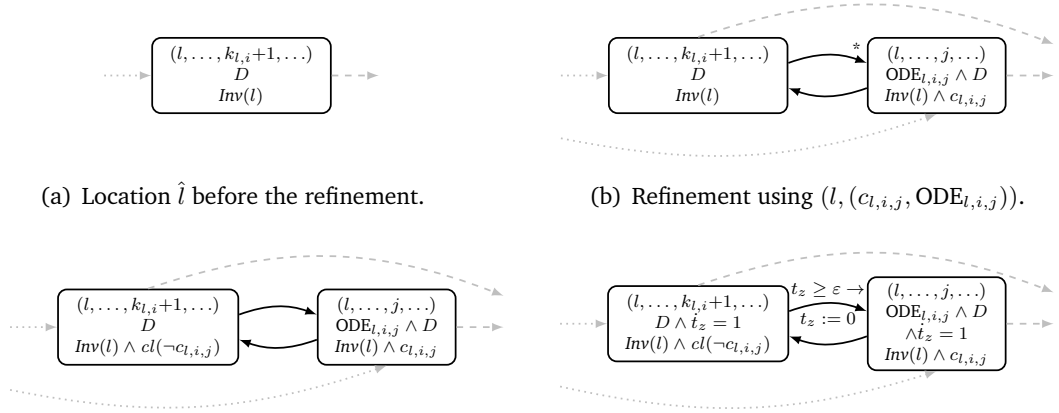
We refine the abstraction by extending the specification of the (initially arbitrary) plant dynamics with some conditional ODEs from the concrete model, which determines the plant dynamics along a presumed counterexample path $(\hat{l}_0, \mathcal{V}_0) \dots (\hat{l}_k, \mathcal{V}_k)$. Our refinement heuristics makes use of *refinement tuples* $(l_i, x, (c, \text{ODE}))$, with l_i being the first component in $\hat{l}_i$ (and as such, it is a location of the model composed from the synchronizer and the SFC models *without* the plant model), $x \in V_{cont}$ is a continuous variable of the plant, and $(c, \text{ODE}) \notin \mathit{active}(l_i, x)$ a conditional ODE for x from the plant dynamics that was not yet considered in location l_i such that there is a valuation in $\mathcal{V}_i$ satisfying c as well as the negation of the conditions of all ODEs for x with higher priorities. In other words, refinement tuples specify conditional ODEs whose inclusion will cause the removal of the given presumed counterexample from the search tree.

We can use such a refinement tuple $(l, x_i, (c, \text{ODE}))$ to refine the model by adding (c, ODE) to $\mathit{active}(l, x_i)$. A refinement step of our model is illustrated in Figure 7.1(a) and 7.1(b). We can but do not necessarily need to use all possible refinement tuples. Related work on possible refinement heuristics in a CEGAR approach have been discussed in [PDMV13] and [BFG+14] in a rectangular respectively affine setting. We experimented with different heuristics, refining either the first possible location on the counterexample path, or the last possible one, or all possible locations. As to the conditional ODEs, we can use for each selected location one, several or all available refinement tuples with the given location component. In the current implementation, we determine the first non-urgent location in the counterexample for which refinement tuples exist, and choose one such tuple for each variable, if it exists.

If no refinement is possible any more (i. e. if there are no refinement tuples) then the counterexample path is *fully refined* and the algorithm terminates with the result that the model is possibly unsafe.

7.1.3. Building the Model at a Given Level of Abstraction

Again let $\mathcal{H}$ be the HA composed from all HA models *without* the plant dynamics. Let $V_{cont} = (x_1, \dots, x_n)$ be the continuous plant variables and let *active* be a function that assigns to each location l of $\mathcal{H}$ and to each continuous plant variable x_i an ordered subset $\mathit{active}(l, x_i) = \{(c_{l,i,1}, \text{ODE}_{l,i,1}), \dots, (c_{l,i,k_{l,i}}, \text{ODE}_{l,i,k_{l,i}})\}$ of the conditional ODEs for x_i with decreasing priorities. We build the global HA model $\mathcal{H}'$ for the controlled plant, induced by the given *active* function, as follows:



(a) Location $\hat{l}$ before the refinement.

(b) Refinement using $(l, (c_{l,i,j}, \text{ODE}_{l,i,j}))$.

(c) Modeling the urgency (over-approximated).

(d) Zeno path exclusion.

Figure 7.1.: Refinement of location $\hat{l}$.

- The locations of $\mathcal{H}'$ are tuples $\hat{l} = (l, l_1, \ldots, l_n)$ with l a location of $\mathcal{H}$ and $l \leq l_i \leq k_{l,i} + 1$ for each $1 \leq i \leq n$. We set $\mathit{Urg}'(\hat{l}) = \mathit{Urg}(l)$.
- The variable set is the union of the variable set of $\mathcal{H}$ and the variable set of the plant.
- For each transition $e = (l, \alpha, g, r, l')$ in $\mathcal{H}$, the automaton $\mathcal{H}'$ has a transition $e' = (\hat{l}, \alpha, g, r, \hat{l}')$ with $\mathit{Urg}'(e') = \mathit{Urg}(e)$ for all locations $\hat{l}$ and $\hat{l}'$ of $\mathcal{H}'$ whose first components are l and l', respectively. Additionally, all locations $\hat{l} = (l, l_1, \ldots, l_n)$ and $\hat{l}' = (l, l'_1, \ldots, l'_n)$ of $\mathcal{H}'$ with identical first components are connected; these transitions have no guards and no effect; they are urgent iff $l_j > l'_j$ for some $1 \leq j \leq n$.
- The activities in location $\hat{l} = (l, l_1, \ldots, l_n)$ are the solutions of the differential equations $\{\text{ODE}_{l,i,l_i} \mid 1 \leq i \leq n, l_i \leq k_{l,i}\}$ extended with the ODEs of $\mathcal{H}$ in l.
- The invariant of a location $(l, l_1, \ldots, l_n)$ in $\mathcal{H}'$ is the conjunction of the invariant of l in $\mathcal{H}$ and the conditions c_{l,i,l_i} for each $1 \leq i \leq n$ with $l_i \leq k_{l,i}$.
- The initial states of $\mathcal{H}'$ are those states $((l, l_1, \ldots, l_n), \nu)$ for which the pair of l and ν projected to the variable set of $\mathcal{H}$ is initial in $\mathcal{H}$, and ν projected to the plant variable set is an initial valuation of the plant.

7.1.4. Dealing with Urgency

The hybrid automaton $\mathcal{H}$ resulting from a refinement contains *urgent locations* and *urgent transitions*. However, available tools for the reachability analysis of hybrid

automata, like SpaceEx and Flow*, do not support urgency. Though a prototype implementation of PHAVer [MF14] supports urgent transitions, it is designed for a restricted class of models with polyhedral derivatives. To solve this problem, we make adaptations to the reachability analysis algorithm and apply some model transformations as follows.

Firstly, we adapt the reachability analysis algorithm such that no time successors are computed in urgent locations.

Secondly, for the urgent transition in the PLC synchronizer model (see Figure 4.3), we remove its urgency and set the time horizon Δ in the reachability analysis to δ, i. e. we restrict the time evolution in location *cycle* to δ.

Thirdly, for each remaining urgent transitions $(\hat{l}, \alpha, g, r, \hat{l}')$, we remove its urgency and replace the invariant $\mathit{Inv}(\hat{l})$ of the source location by the condition $\mathit{Inv}(\hat{l}) \wedge cl(\neg g \vee \neg \mathit{Inv}(\hat{l}')[r(V)/V])$, where $cl(\cdot)$ denotes the closure of a set and the expression $\mathit{Inv}(\hat{l}')[r(V)/V]$ (substituting all variables by their reset expressions) is the weakest precondition of satisfying $\mathit{Inv}(\hat{l}')$ after applying the reset function. Though this transformation is over-approximative, due to the strengthening of the invariants we introduce additional behavior in the source location only as long as the dynamics moves along the boundary between the invariants of the source and the target locations. An example is shown in Figure 7.1(c).

7.1.5. Dealing with Zeno Paths

The transformation to resolve non-convex invariants allows paths with infinitely many mode switches in zero time. This is called *Zeno* behavior which should be avoided since both the running time and the over-approximation might increase drastically.

One possibility to avoid these Zeno behaviors is to force a minimal time elapse ε in each cycle of a location set introduced for the encoding of a non-convex invariant. To do so, we can introduce a fresh clock t_z and modify at least one transition $e = (l, \alpha, g, r, l')$ in each cycle by an annotated variant (l, α, g', r', l'), where g' is g strengthened by the requirement $t_z \geq \varepsilon$, and r' is r extended with the assignment $t_z := 0$. Additionally, we add the differential equation $\dot{t_z} = 1$ to all locations on the cycle. The result of this transformation is shown in Figure 7.1(d). Note that the above transformation eliminates Zeno paths, but it leads to an *under-approximation* of the original behavior.

Another possibility avoiding the introduction of a new variable is to modify the reachability analysis algorithm such that the first flowpipe segment in the source location of such transitions e computes time successors for ϵ and from this first segment no jump successors are computed along e. If the model is safe, we also complete the reachability analysis for those, previously neglected jump successors, in order to re-establish the over-approximation.

7.1.6. CEGAR Iterations

For the initial abstraction and after each refinement we start a reachability analysis on the model at the current level of abstraction. The refinement is iterated until 1) the reachability analysis terminates without reaching an unsafe state, i. e. the model is correct, or 2) a fully refined path from an initial to an unsafe state is found. In the case of 2), the unsafe behavior might result from the over-approximative computation, thus the analysis returns that the model is *possibly* unsafe.

7.2. Integrating CEGAR into the Reachability Analysis

In this chapter we explain how the proposed CEGAR approach can be integrated into flowpipe-computation-based reachability analysis for hybrid systems. After some general information we give implementation details that have been important for the integration in the analysis tool SpaceEx.

7.2.1. Adapting the Reachability Analysis Algorithm

Restarting the complete model analysis in each refinement iteration leads to a re-computation of the whole search tree, including those parts that are not affected by the refinement step. To prevent such restarts, we do the model refinement on-the-fly during the reachability analysis and backtrack in the search tree only as far as needed to remove affected parts.

For this computation we need some additional bookkeeping: During the generation of the search tree, we label all time successor nodes $(l, \mathcal{V})$ with the set $V \subseteq V_{plant}$ of all plant variables, for which arbitrary behavior was assumed in the flowpipe computation, resulting in a node $(l, \mathcal{V}, V)$. In the initial tree all time successor nodes are labeled with the whole set of plant variables. Discrete successors are labeled with the empty set.

We start with the fully abstract model, i. e. with the composition $\mathcal{H}$ of the synchronizer and the SFC models, extended by the variables of the plant. Note that initially we do not add any information about the plant dynamics, i. e. we allow arbitrary behavior for the plant.

We apply a reachability analysis to this initial model. If it is proven to be safe, we are done. Otherwise, we identify a path in the search tree that leads to an unsafe state set and extend the *active* function to *active*′ as previously described.

However, instead of re-starting the analysis on the explicit model induced by the extended *active*′ function, we proceed as follows:

- *Backtracking:* When a refinable counterexample is found, we delete all nodes $(l, \mathcal{V}, V)$ (and all subtrees below them) in the search tree for which

  $$\mathit{active}'(l, x) \setminus \mathit{active}(l, x) \neq \emptyset$$

  for some $x \in V$. We mark the parents of deleted nodes as not completed.

Algorithm 3: SuccessorComputation

```
   input :
   /* search tree of the reachability analysis                    */
1    SearchTree tree;
   /* non-completed node in search tree                           */
2    Node n_0 = (l_0, V_0, ∅);
   /* variables with arbitrary dynamics in l_0                    */
3    VariableSet V ⊆ V_cont;
   /* dynamics in l_0                                             */
4    ODESet C;
   /* maximal time that can be spent in l_0                       */
5    TimeHorizon Δ = mτ;
   /* flowpipe computation                                        */
6  if l_0 is not urgent then
7      for i = 1 to m do
8          V_i := flow(V_{i-1}, C, τ, Inv(l_0));
9          if V_i = ∅ then
10             m := i − 1;
11         else
12             n_i :=tree.addChild(n_{i-1}, (l_0, V_i, V));
   /* jump successor computation                                  */
13 for i = 1 to m do
14     foreach transition e = (l_0, α, g, r, l) do
15         V := jump(V_i, g, r, Inv(l));
16         if V ≠ ∅ then
17             tree.addChild(n_i, (l, V, ∅));
18 for i = 0 to m do
19     mark n_i completed;
```

- *Model Refinement:* After the backtracking, we refine the automaton model that is used for the analysis on-the-fly, by replacing the location(s) with arbitrary behavior in newly refined variables x by the locations that result from the refinement. After this modification, we apply the reachability analysis algorithm on the parents of refined nodes to update the search tree.

- *Reachability Computation:* According to a heuristics, we iteratively apply Algorithm 3 to the set of non-completed nodes in the search tree until we detect an unsafe set (in which case a new refinement iteration starts) or all nodes are completed (in which case the model is safe).

In the following, we explain how Algorithm 3 generates the successors of a tree node $(l_0, \mathcal{V}_0, V_0)$.

First the algorithm computes the time successors if the location is not urgent (lines 6-12): The set of states $\mathit{flow}(\mathcal{V}_{i-1}, C, \tau, \mathit{Inv}(l_0))$ reachable from the node n_{i-1} under dynamics C within time τ is computed for all flowpipe segments within the time horizon and added as a child of n_{i-1}, with the set $V \subseteq V_{\mathit{cont}}$ of the variables with arbitrary dynamics in l_0 attached to it. Note that though we use a fixed step size τ it could also be adjusted dynamically.

Next, the successor nodes of each flowpipe segment n_i that are reachable via a discrete jump are computed (lines 13-17). For each transition $e = (l_0, \alpha, g, r, l)$, the valuation set $\mathcal{V}$ is computed that is reachable via transition e when starting in $\mathcal{V}_i$ (line 15). The successor $(l, \mathcal{V}, \emptyset)$ is inserted into the search tree as a child of n_i; it is labeled with the empty set of variables since no arbitrary behavior was involved (line 17).

Finally, since all possible successors of all n_i are added to the search tree, they are marked as completed (lines 18-19). Optionally, we can also mark all new jump successor nodes whose valuation sets are included in the valuation set of another node with the same location component as completed. Since the inclusion check is expensive, it is not done for each new node, but in a heuristic manner.

In most implementations, several additional techniques (e. g. clustering and aggregation) are used to speed up computations, which we do not discuss here as they are not relevant for understanding our approach.

7.2.2. Implementation

The prototype implementation of this CEGAR methodology is available under [8]. For the implementation we must be able to generate a (presumed) *counterexample*, if an unsafe state is reachable. Moreover, our modeling approach uses *urgent locations* in which time cannot elapse and *urgent transitions* whose enabledness forces to leave the current location. Although FLOW* provides presumed counterexamples, we decided to integrate our method into SPACEEX. The reason is that SPACEEX provides different algorithms for the reachability analysis. Important for us is the PHAVER scenario [MF14] that supports urgent transitions and non-convex invariants for a simpler class of hybrid automata. Furthermore, in [BDF+13] an extension of SPACEEX for hybrid automata is presented where the search is guided by a cost function. This enables a more flexible way of searching the state space compared to a breadth- or depth-first-search.

7.2.2.1. Some SpaceEx Implementation Details

The SPACEEX tool computes the set of reachable states in so-called *iterations*. In each iteration, a state set is chosen, for which both the time elapse and the jump successors are computed. The *waiting list* of states that are reachable but have not yet been analyzed, is initialized with the set of initial states. At the beginning of an iteration, the front element w of the waiting list is picked for exploration. First the time elapse successors T of w are computed and stored in the set of reachable states

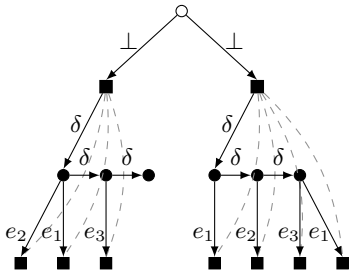


Figure 7.2.: The search tree (empty circle: distinguished root; filled rectangle: jump successor node; filled circle: time successor node; dashed connections: shortcut to the next jump successors).

(*passed list*). Afterwards, the jump successors J are computed for each $s \in T$. These states are added to the waiting list, as they are non-completed.

In SpaceEx, the *passed* and the *waiting list* are implemented as *FIFO* queues, i. e. elements are added at the end of the list and taken from the front.

When either the waiting list is empty (i. e. a fixed-point has been reached) or the specified number of iterations is completed, the analysis stops. The reachable states are the union of the state sets in the passed and the waiting list. If *bad states* are specified, the intersection of the reachable and the bad states is computed.

7.2.2.2. Search Tree

An important modification we had to make in SpaceEx is the way of storing reachable state sets discovered during the analysis. Whereas SpaceEx stores those sets in a queue, our algorithm relies on a search tree. Thus we added a corresponding tree data structure. We distinguish between *jump successor* and *time successor* nodes which we represent graphically in Figure 7.2 by filled rectangles and filled circles, respectively. The set of initial states are the children of the distinguished root node which is represented by an empty circle. Each node can have several jump and at most one time successor nodes as children. For a faster access, each jump successor node stores a set of pointers to the next jump successors in its subtree (dashed arrows in Figure 7.2).

To indicate whether all successors of a node have been computed, we introduce a *completed* flag. In each iteration, we determine a non-completed tree node. If its location is non-urgent, we compute its time successors and the jump successors of all time successors. Afterwards, the chosen node and all computed time successors are marked as completed. For urgent locations only the jump successors are computed. We use *BFS* to find the next non-completed tree node. The iterative search stops if either all tree nodes are completed or the state set of a node intersects with a given set of bad states. The latter is followed by a refinement and the deletion of a part of the search tree (note that the parents of deleted nodes are marked as non-completed). After this backtracking we start the next iteration with a new BFS from

the root node. If the BFS gives us a node which already has some children then this node was previously completed and some of its children have been deleted during backtracking. In this case we check for each successor whether it is already included as a child of the node before adding it to the tree.

Refinement relies on counterexample paths, i. e. on paths from an initial to a bad state. To support more information about counterexample paths, we annotate the nodes in the search tree as follows. Each jump successor node contains a reference to the transition that led to it, and each time successor node stores the time horizon that corresponds to its time interval in the flowpipe computation.

7.2.2.3. Urgent Locations

Our implementation supports urgent locations, for which no time successors are computed.

7.2.2.4. Bad States

In SpaceEx, a set of bad (*forbidden*) states can be specified by the user. After termination of the reachability analysis algorithm, the intersection of the reachable states with the forbidden ones are computed and returned as output information.

In our implementation, we stop the reachability computation once a reachable bad state has been found. Therefore, we perform the intersection check for each node directly after it has been added to the tree. This allows us to perform a refinement as soon as a reachable bad state is detected.

7.2.2.5. Refinement

When a counterexample is detected, a heuristics chooses a (set of) location(s) and corresponding conditional ODEs for the refinement. We extend the set of active ODEs and refine the hybrid automaton model on-the-fly. Afterwards, the analysis automatically uses the new automaton model and the backtracked search tree to continue the reachability analysis.

7.2.2.6. Backtracking

When the model refinement is completed, we delete all nodes (and their subtrees) whose location was refined. The parents of deleted nodes are marked as non-completed. This triggers that their successors will be re-computed. Since we use a BFS search for non-completed nodes, first the successors of such nodes are computed before other nodes are considered.

7.2.2.7. Refinement Heuristics

We implemented a command line interface that allows us to choose the set of locations and corresponding conditional ODEs for the refinement manually whenever a

counterexample has been detected. This provides us with the most flexibility since any strategy can be applied. Several heuristics could be investigated and automated strategies for promising heuristics could be implemented.

7.2.2.8. Analysis Output

In case a counterexample path is detected that is fully refined, we abort the analysis and output the counterexample path. It can be used to identify the part of the model that leads to a violation of the specified property. Otherwise, the analysis is continued until either a fixed-point is found or the number of maximal allowed iterations was computed.

7.3. Outlook

In this chapter we presented a second CEGAR-based methodology for the reachability analysis of SFC-controlled chemical plants. We integrated a CEGAR loop into an existing tool for the reachability analysis of hybrid automata. We obtain relatively small system models by abstracting away parts of the dynamic behavior. As the experimental results presented in Chapter 9 will show, this approach is applicable to small benchmarks. However, even the model of a small benchmark consists of 20 dimensions and more than thousand transitions in the fully refined model. As expected, the analysis for such a model is quite time consuming.

8. Advanced Verification Techniques for PLC-Controlled Plants

The focus of the previous chapters has been the development of *CEGAR* based analysis methods for SFC controlled plants. However, even for small examples, the analysis is relatively time consuming. Thus, we examined our models to find the issues that make the analysis costly.

An intrinsic characteristic of our model is the great number of discrete variables. Unfortunately, in state-of-the-art tools, there is no support for discrete variables but they are modeled as continuous ones that increase the dimension. However, separating the discrete variables from the continuous ones for the reachability analysis has a strong impact on the running time for our models.

A second issue is the extensive use of urgency that we use to model the PLC cycle, e. g. to synchronize the SFC controllers on reading the input from the plant and writing the output. Although it is possible to model urgent locations and urgent jumps, we would benefit from direct tool support for urgency.

We decided to use the HyPro reachability analysis tool for hybrid systems to analyze the impact of these two enhancements on our models. HyPro provides a library of implementations for a broad variety of different state set representations and analysis techniques. HyPro is designed to be modular and to provide an easy-to-use library of state set representations for fast prototypical implementations of reachability analysis algorithms. Though of course efficiency plays an important role, the main objective of HyPro is not efficiency but clear modularity in the variety of representations. We want to thank the developers of the tool, first of all Stefan Schupp, for implementing support for both discrete variables and urgency in HyPro, which was not yet released at the time of writing.

Outline In this chapter, we show two enhancements for the analysis of plant control. In Section 8.1, the benefit of separating discrete variables from continuous ones are explained. The second enhancement is the direct support of urgent locations and jumps which is targeted in Section 8.2. Note that neither the support for discrete variables nor the introduction of urgency changes the expressiveness of the automaton models. In Section 8.3, we present an enhanced modeling approach that reduces the size of our models. Moreover, Zeno behavior is reduced to the part of the model where the plant dynamics is evaluated. An outlook of this chapter is given in Section 8.4.

8.1. Discrete Variables

The values of discrete variables do not change during time elapse but only during discrete jumps, whereas continuous variables can be affected by both time evolution and discrete jumps. The support of discrete variables in our models enables us to handle the discrete variables differently during the reachability analysis.

We separate our set of variables $V := V_d \cup V_c$ into a set of discrete variables V_d and a set of continuous variables V_c for the analysis and perform all operations separately for the sets of discrete and continuous variables. Since the computation effort of most operations grows exponential in the number of dimensions, we gain performance by splitting each state set into two separate state sets with smaller dimensions, one for the discrete and one for the continuous variables.

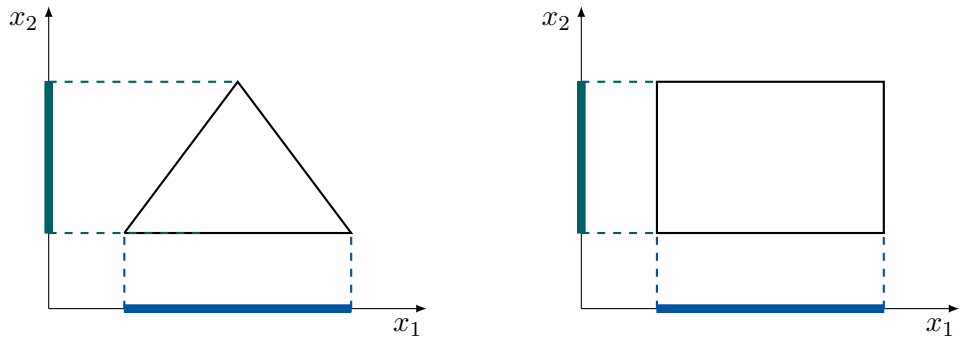
When taking a time transition, the set of continuous variables evolves according to the specified dynamics whereas the set of discrete variables remains untouched. Thus, the dimension of the state set for the flowpipe construction can be reduced to the number of continuous variables. Due to the high number of discrete variables in our models, this saves a noticeable amount of running time. Another advantage of neglecting the set V_d of discrete variables during time evolution is a precision gain for the discrete variables. Since only the set V_c changes during the flowpipe computation, the over-approximation of the set of states that are reachable during time elapse is reduced to the set of continuous variables.

Invariant intersections as well as guard intersections and resets are computed separately for the sets V_d of discrete and V_c of continuous variables. Note that in the models of our applications there are no mixed expressions referring to both discrete and continuous variables. The only mixed terms model sensoring and assign the value of an expression over continuous variables to discrete variables. Thus, all operations can safely be performed for both sets of variables separately without loosing precision.

8.2. Urgency

Urgency enables us to leave a location immediately if a given condition is satisfied by the current system state. To model urgency, it is necessary to introduce clocks since we have to restrict the time evolution. Urgent locations have to be left immediately via a discrete jump, i. e. we need a clock and additional invariant constraints to guarantee that time cannot elapse. However, the first flowpipe segment is always computed by state-of-the-art tools which causes unnecessary over-approximation. For urgent transitions, we have to ensure that time cannot elapse further in the current location if the guard of an urgent transition evaluates to true in the current state, which can again be modeled by additional invariant constraints.

Direct tool support for urgency however would bring a performance and precision gain. Support for urgent locations can be implemented by skipping the flowpipe construction completely. Thereby the over-approximation that occurs, if the first



(a) Projection for a triangle.

(b) Projection for a box.

Figure 8.1.: Projection.

flowpipe segment is computed, does not apply and we gain both performance and precision.

For urgent transitions the analysis algorithm could be informed whether the guard of an urgent transition is enabled for some states $\Sigma' \subseteq \Sigma$ in a given state set Σ since the time evolution has to be stopped and only discrete jumps can be taken for the states from Σ'. If $\Sigma' = \Sigma$, we do not need to compute further flowpipe segments. However, if $\Sigma' \subset \Sigma$ then flowpipe computation needs to be continued for $\Sigma \setminus \Sigma'$. Unfortunately, set difference cannot be computed efficiently in an over-approximative manner by available techniques.

In our models, urgent transitions are time triggered, i. e. we want to leave a location at the end of the PLC cycle which is at time δ. Therefore, we introduced a new clock to our model that measures the cycle time. However, this clock increases the dimension of our model and is affected by over-approximation during flowpipe computations. Thus, we decided to make further changes in the analysis tool HyPro for this special case of urgency by separating the set of clocks that are solely used for urgent transitions from the remaining continuous variables. Thereby we can reduce the dimension of our models. As there are no expressions in the models that refer to both variable sets, we can handle the clocks separately during the reachability computations. However, in contrast to the separation of discrete variables, this separation comes with the disadvantage of additional over-approximations.

The reason for this over-approximation is that by separating the clocks from the continuous variables we loose information on the relation between those two variable sets. In general, if we apply projection to a state set P resulting in a lower-dimensional state set P', we loose the information which extensions of states in P' are included in P. Figure 8.1 illustrates this phenomenon: Consider the triangle-shaped state set in Figure 8.1(a). If we separate the variables x_1 and x_2 from each other and only consider their projections, we loose information on the relation between the two variables and cannot reconstruct the original shape anymore. All we know is that the cross product of the two projected sets contains the original set.

However, if the global state set is the cross product of the projections, it is possible to reconstruct the original set precisely (see for example the rectangular set in Figure 8.1(b)). Otherwise, if necessary we can reduce the over-approximation error by reducing the step size of the flowpipe computation. Another solution to avoid over-approximation is to store the connection between the two sets by introducing a fresh clock that is reset on all jumps and that is included in both sets; now we can build the cross product of two state sets for the separated variable sets and project out the fresh clock dimension to achieve the state set for the united variable set. Note that this problem does not occur for separate sets of discrete and continuous variables since the discrete variables do not change their values over time. Thus, the two sets of variables are independent from each other and therefore the cross product of two state sets for the discrete respectively continuous variables is the precise global state set.

Although the over-approximation due to separating the clocks from the continuous variables and using the projections seems to be an issue, in our experiments, the advantage of the reduction of the dimension outbalanced the drawbacks of over-approximation.

8.3. Automata Models

We build the parallel composition of the controller automata and the automaton that models the PLC cycle. The plant dynamics is given by the parallel composition of the automata for the physical quantities. Instead of building the parallel composition of these two automata, we can reduce the model size by alternating the execution of the controllers at cycle start and the execution of the plant dynamics for the duration δ of a cycle. During time evolution the discrete variable values remain unchanged. To be able to restore the controller states after the cycle time, we store the current locations of the controllers in some fresh variables. This way, after the physical quantities evolved for time δ, we can jump back to the stored location in the controller automaton with a write transition. This optimization is illustrated in Figure 8.2. A further example for the thermostat is shown in Figure 8.3; see the next chapter for the description of the controller. Note that this modeling approach eliminates Zeno behavior due to mode switches in the plant dynamics during controller execution. Moreover, since time does not evolve in the urgent locations corresponding to controller execution, we do not need to specify any dynamics.

8.4. Outlook

In this chapter, we introduced three enhancements for the analysis of our models. A first characteristic of our models is the huge number of discrete variables. The analysis running time can be reduced by separating the sets of discrete and continuous variables and by handling these sets differently during analysis. Urgency is a second

characteristic of our models. Although urgent locations and transitions can be modeled even if there is no tool support for them, the running time can be reduced and the accuracy of the resulting set of reachable space can be improved if the analysis supports urgency. Last, we presented an improved modeling approach that reduces the model size by alternating the execution of the controllers and the evaluation of the plant dynamics. In the experimental results, we present the evaluation of the advanced verification techniques (Section 9.4).

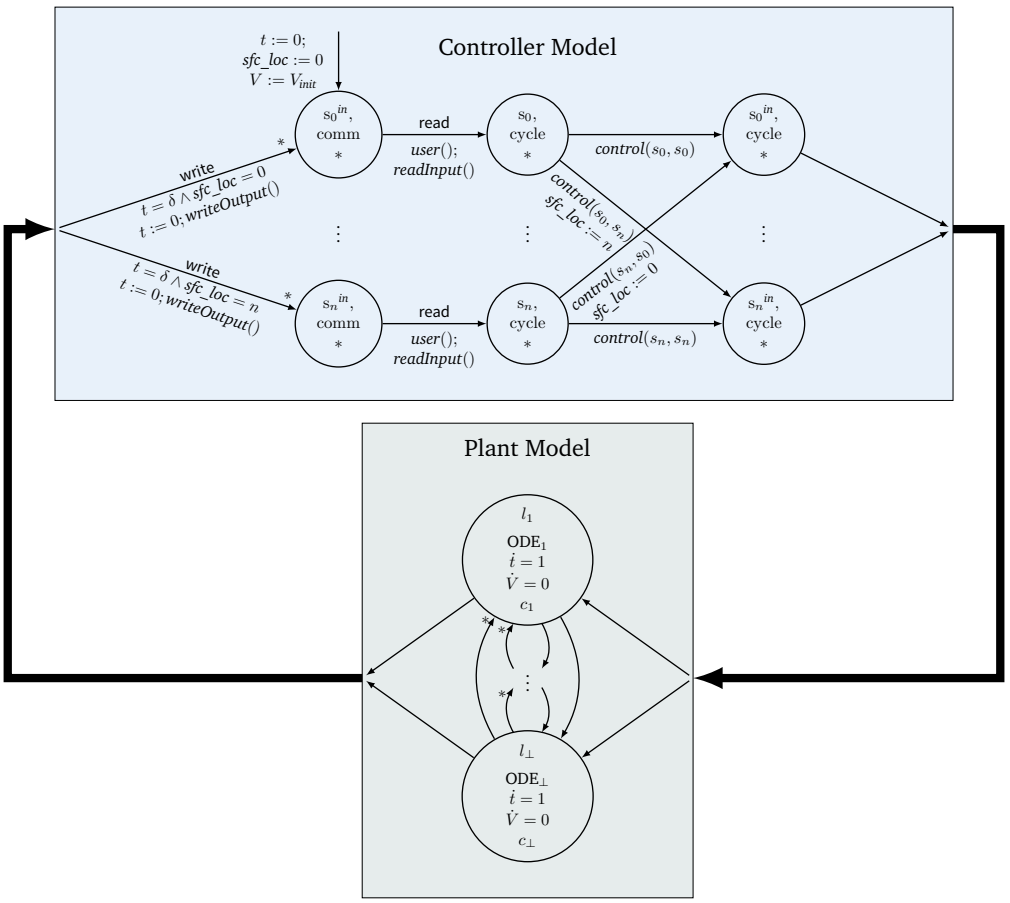


Figure 8.2.: The model that alternates the execution of controllers and the plant dynamics. The method *control*(s_i, s_j) computes the guard and the reset assignments if the control moves from s_i to s_j. Each outgoing transition of the controller model is merged with each incoming transition of the plant model and vice versa.

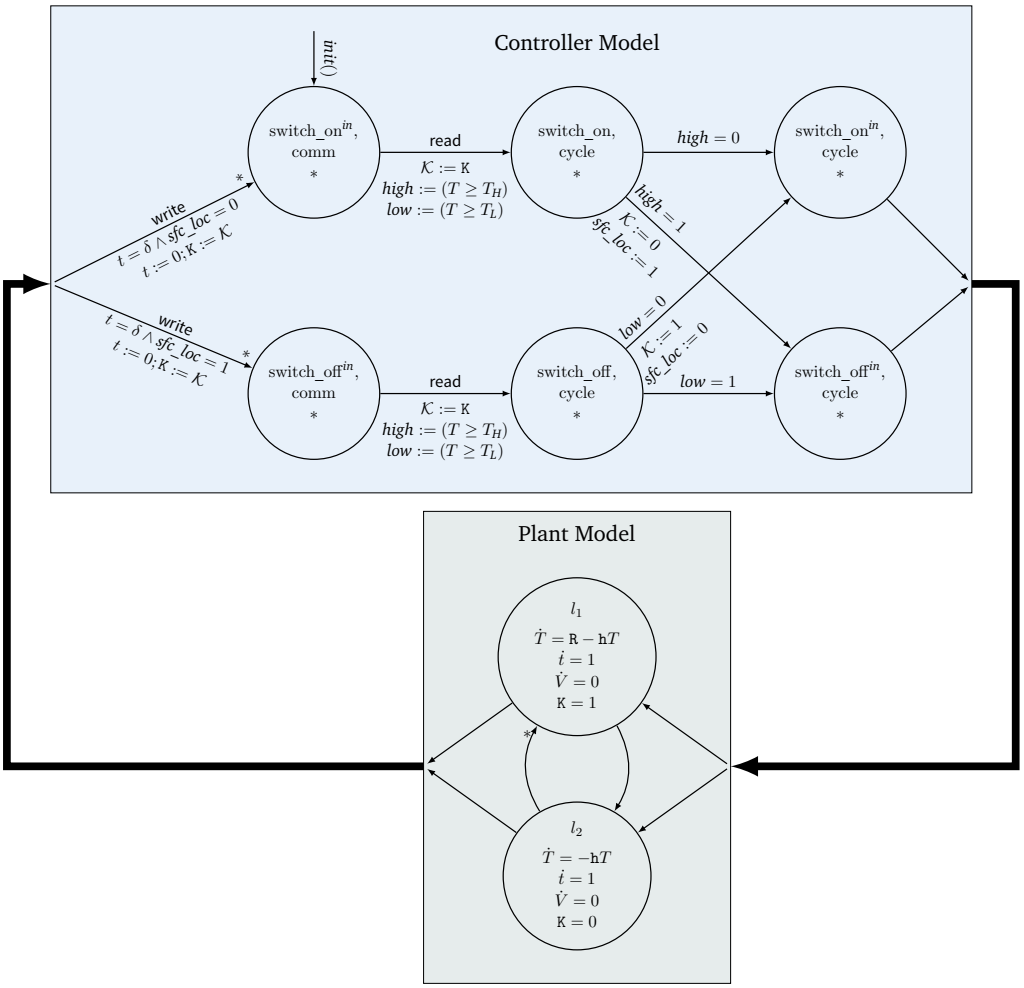


Figure 8.3.: Alternating model for the thermostat example that is introduced in Section 9.1. We assume the following initialization: $init() = \mathtt{K} = 1; \mathcal{K} = 1; \mathtt{low} = 1; low = 1; \mathtt{high} = 0; high = 0; T = 20; t = 0; sfc_loc = 0$.

9. Experimental Results

We created prototype implementations of the proposed analysis methods for chemical plant control (see Chapters 6, 7, and 8) and tested them using different benchmarks from the area of plant control. They all consist of simple plants with one or two SFC controllers: The first example is a thermostat controller that keeps the room temperature within a given interval, the second example is a leaking water tank that can be filled via an unlimited water supply, and the last example is the closed two tank system from Section 3.2.

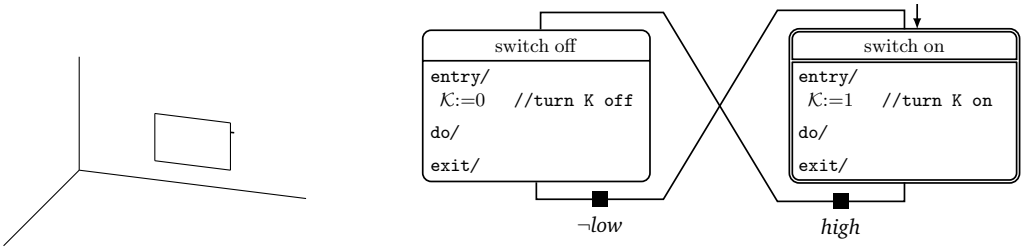
Outline In this chapter, we present our experimental results for the proposed analysis methods for chemical plant control. We start with an introduction of the benchmarks in Section 9.1. Afterwards, we present experimental results for the first CEGAR-based analysis approach from Chapter 6 in Section 9.2, the results for the second CEGAR approach from Chapter 7 in Section 9.3, and the results for the advanced verification techniques for PLC-controlled plants that have been implemented in the analysis tool HYPRO (see Chapter 8) in Section 9.4. We conclude this chapter in Section 9.5.

9.1. Benchmarks

For each benchmark, we model the plant dynamics and each controller in separate hybrid automata. Another model is created for the synchronization of the cyclic controller execution on a PLC and the processing of user input. We build the parallel composition of the separate models for the plant dynamic, the controllers, and the PLC cycle. In the following, we introduce the three benchmarks.

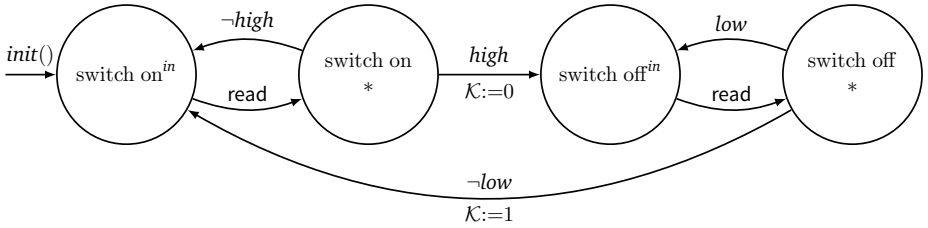
9.1.1. Thermostat

In this benchmark, we consider a room with a heater as shown in Figure 9.1(a). Note that this benchmark is different from the thermostat benchmark from Chapter 5. The heater's thermostat measures the temperature T and switches the heater on and off to keep the room temperature above a lower limit T_{min} and below an upper limit T_{max} with $T_{min} < T_{max}$. Initially, the temperature is $T = T_I$ and the heater is on ($\mathtt{K} = 1$). When the temperature rises above the temperature T_H, the heater is switched off and when the temperature drops below T_L, the heater is switched on again. We assume $T_{min} < T_L < T_H < T_{max}$. The SFC controller of the thermostat is shown in Figure 9.1(b) and the corresponding automaton model in Figure 9.1(c).



(a) Heater with thermostat.

(b) SFC controller.



(c) Automaton model for the thermostat controller with $\mathit{init}() = \mathtt{K} = 1; \mathcal{K} = 1; \mathtt{low} = 1; \mathit{low} = 1; \mathtt{high} = 0; \mathit{high} = 0; T = 20; t = 0$.

Figure 9.1.: The thermostat benchmark.

The plant variables are given by the variable T for the physical quantity that models the room temperature, the actuator variable K that models the state of the heater (on: $\mathtt{K} = 1$, off: $\mathtt{K} = 0$), and by the sensor variables low and high that detect low and high temperatures, respectively ($T_L < T$: $\mathtt{low} = 0$, $\mathtt{high} = 0$; $T_L \leq T < T_H$: $\mathtt{low} = 1$, $\mathtt{high} = 0$; $T \geq T_H$: $\mathtt{low} = 1$, $\mathtt{high} = 1$). The SFC controller has input variables $\mathcal{K}, \mathit{low}$, and high in which the current states of the heater and the sensors are stored at the beginning of each PLC cycle. The output variable of the SFC is $\mathcal{K}$ to which the computation result is written during the cycle in form of a request to switch the heater on ($\mathcal{K} = 1$) or off ($\mathcal{K} = 0$). An overview over the plant variables ($V_{act}, V_{sen}, V_{cont}$) and the controller variables (V_{in}, V_{loc}, V_{out}) is given below:

$$\begin{aligned} V_{act} &= (\mathtt{K}) & V_{in} &= (\mathcal{K}, \mathit{low}, \mathit{high}) \\ V_{sen} &= (\mathtt{low}, \mathtt{high}) & V_{loc} &= () \\ V_{cont} &= (T) & V_{out} &= (\mathcal{K})\,. \end{aligned}$$

When the heater is switched on, the temperature rises according to the differential equation $\dot{T} = \mathtt{h} - \mathtt{R}T$, where h is a heating constant and R is a room constant. When the heater is switched off, the change in temperature is given by $\dot{T} = -\mathtt{R}T$. The resulting conditional ODE system is:

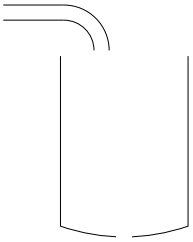
$$\begin{aligned} (c_1^T, \mathrm{ODE}_1^T) &= (\mathtt{K} = 1, \quad \dot{T} = \mathtt{h} - \mathtt{R}T) \\ (c_2^T, \mathrm{ODE}_2^T) &= (\mathtt{K} = 0, \quad \dot{T} = -\mathtt{R}T)\,. \end{aligned}$$

We want to verify that the room temperature always stays between T_{min} and T_{max} for the following constants and initial set:

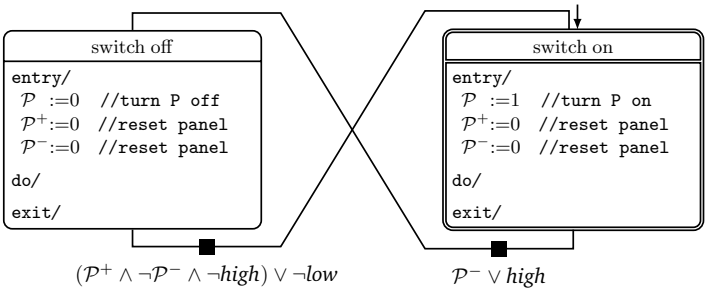
Constants: $T_I = 20°C$, $T_L = 18°C$, $T_H = 23°C$, $T_{min} = 16°C$, $T_{max} = 24°C$, $\mathtt{h} = 3$, $\mathtt{R} = 0.1$
Initially: $T = T_I$, $\mathtt{K} = \mathcal{K} = 1$, $\mathtt{low} = \mathit{low} = 1$, $\mathtt{high} = \mathit{high} = 0$
Unsafe Set: $\varphi := T > T_{max} \vee T < T_{min}$

9.1.2. Leaking Water Tank

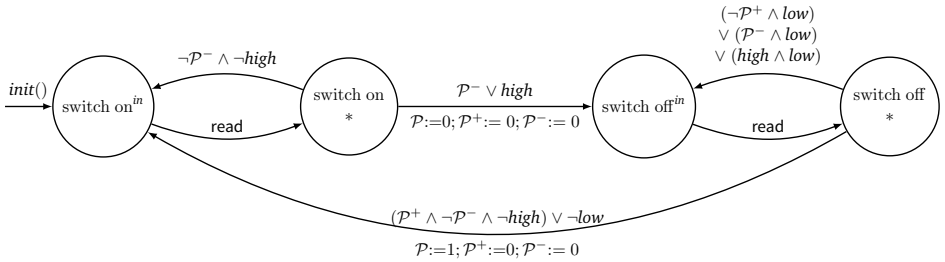
The second benchmark is a tank that can be filled via an unlimited water supply (see Figure 9.2(a)). If the pump P is switched on, water flows into the tank. The tank is constantly leaking through an outlet in the bottom. The tank is equipped with two sensors high and low to detect a high or low water level, respectively. Via a user input panel, the request to switch the pump on or off can be given. We model a user that toggles between switching the pump off and on in each PLC cycle, e. g., if the pump is currently working, the user requests to switch the pump off and if the pump is switched off, the user requests to switch it on again. A fault detection prevents the tank from flooding or from running dry, i. e. the pump is switched off or on, respectively, if the water level reaches the position H or drops below the position L



(a) Leaking tank with water supply.



(b) SFC controller.



(c) Automaton model for the leaking tank controller with $\mathit{init}() = \mathtt{P} = 1; \mathcal{P} = 1; \mathtt{P}^+ = 0; \mathcal{P}^+ = 0; \mathtt{P}^- = 1; \mathcal{P}^- = 1; \mathtt{low} = 1; \mathit{low} = 1; \mathtt{high} = 0; \mathit{high} = 0; h = 10; t := 0$.

Figure 9.2.: The leaking tank benchmark.

of the sensor `high` or `low`, respectively. The SFC controller for the leaking tank is shown in Figure 9.2(b) and the corresponding automaton model in Figure 9.2(c).

The plant variables are given by the physical quantity h for the water level of the tank, by the actuator variable $\mathtt{P}$ that models the state of the pump (on: $\mathtt{P} = 1$, off: $\mathtt{P} = 0$), and by the variables $\mathtt{P}^+, \mathtt{P}^-$ that model the user input. The user can request to switch the pump on ($\mathtt{P}^+ = 1$) or off ($\mathtt{P}^- = 1$). The plant variables `low` and `high` model the sensors that detect low ($h < L$), normal ($L \leq h < H$) or high ($h \geq H$) water levels.

The controller has five input variables $\mathcal{P}, \mathcal{P}^+, \mathcal{P}^-, \mathit{low}$, and high which indicate the pump state (on: $\mathcal{P} = 1$, off: $\mathcal{P} = 0$), the user requests to switch the pump on ($\mathcal{P}^+ = 1$) or off ($\mathcal{P}^- = 1$), and the sensor values at the beginning of the PLC cycle. During the PLC cycle, output is written to the variables $\mathcal{P}, \mathcal{P}^+$, and $\mathcal{P}^-$ in form of a request to switch the pump on or off or in form of a reset of the user input. An overview of the plant variables ($V_{act}, V_{sen}, V_{cont}$) and the controller variables (V_{in}, V_{loc}, V_{out}) is given below:

$$\begin{aligned} V_{act} &= (\mathtt{P}, \mathtt{P}^+, \mathtt{P}^-) & V_{in} &= (\mathcal{P}, \mathcal{P}^+, \mathcal{P}^-, \mathit{low}, \mathit{high}) \\ V_{sen} &= (\mathtt{low}, \mathtt{high}) & V_{loc} &= () \\ V_{cont} &= (h) & V_{out} &= (\mathcal{P}, \mathcal{P}^+, \mathcal{P}^-)\,. \end{aligned}$$

When the pump is switched on and the tank is not full ($h \leq F$), the water level rises according to the differential equation $\dot{h} = k_1 - k_2$, where k_1 is the amount the water level rises per time unit due to the water inflow, and $k_2 < k_1$ is the leaking constant that gives the decrease of water level per time unit due to water constantly leaking from the tank. When the pump is switched off ($\mathtt{P} = 0$) and the tank is not empty ($h \geq 0$), the water level change is given by $\dot{h} = -k_2$. The water level is constant if the tank is either flooding with the pump being on or empty. The resulting conditional ODE system is:

$$\begin{aligned} (c_1^h, \mathrm{ODE}_1^h) &= (\mathtt{P} = 1 \wedge h \leq F, \quad \dot{h} = k_1 - k_2) \\ (c_2^h, \mathrm{ODE}_2^h) &= ((\mathtt{P} = 1 \wedge h \geq F) \vee (h \leq 0), \quad \dot{h} = 0 \quad) \\ (c_3^h, \mathrm{ODE}_3^h) &= (\mathtt{P} = 0 \wedge h \geq 0, \qquad\qquad \dot{h} = \quad -k_2)\,. \end{aligned}$$

We want to analyze whether the tank can flood or run dry, i. e. if $h \leq 0$ or $h \geq F$ for the following constants and initial set:

Constants: $h_I = 10$, $L = 6$, $H = 12$, $F = 20$, $k_1 = 3.5$, $k_2 = 1.5$
Initially: $h = h_I$, $\mathtt{P} = \mathcal{P} = 1$, $\mathtt{P}^+ = \mathcal{P}^+ = 0$, $\mathtt{P}^- = \mathcal{P}^- = 1$, $\mathtt{low} = \mathit{low} = 1$, $\mathtt{high} = \mathit{high} = 0$
Unsafe Set: $\varphi := h \leq 0 \ \vee \ h \geq F$.

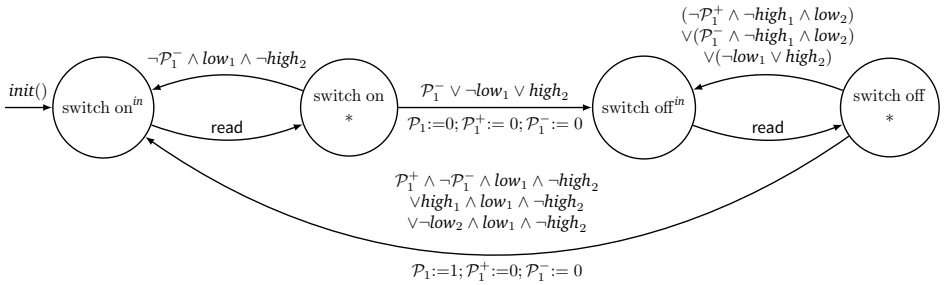


Figure 9.3.: Automaton model for the controller of pump $\mathtt{P}_1$ of the two tank example with $\mathit{init}() = \mathtt{P}_i = 1; \mathcal{P}_i = 1; \mathtt{P}_i^+ = 0; \mathcal{P}_i^+ = 0; \mathtt{P}_i^- = 1; \mathcal{P}_i^- = 1; \mathtt{low}_i = 1; \mathit{low}_i = 1; \mathtt{high}_i = 0; \mathit{high}_i = 0; h_i = 10; t = 0$.

9.1.3. Two Tank System

We recall the two tank example from Section 3.2 where two cylindrical tanks are connected by pipes in a closed system (see Figure 3.1 on page 15). The automaton model for the SFC controller in Figure 3.2 on page 17 is recalled in Figure 9.3.

As described in Section 3.2, we use the following variable encodings, where $\mathtt{P}_i$ denotes the state of pump i, $\mathtt{P}_i^+, \mathtt{P}_i^-$ the user request to switch pump i on or off, respectively, and $\mathtt{low}_i, \mathtt{high}_i$ model the sensors that detect low or high water levels in tank i. The water level of tank i is modeled by the variable h_i. The corresponding variables for the state of the pump, the user requests, and the sensors in the SFC controller are given by $\mathcal{P}_i, \mathcal{P}_i^+, \mathcal{P}_i^-, \mathit{low}_i$, and high_i.

$$
\begin{aligned}
V_{act} &= (V_{act}^1, V_{act}^2) & V_{in} &= (V_{in}^1, V_{in}^2) \\
V_{sen} &= (V_{sen}^1, V_{sen}^2) & V_{loc} &= () \\
V_{cont} &= (V_{cont}^1, V_{cont}^2) & V_{out} &= (V_{out}^1, V_{out}^2)
\end{aligned}
$$

with

$$
\begin{aligned}
V_{act}^i &= (\mathtt{P}_i, \mathtt{P}_i^+, \mathtt{P}_i^-) & V_{in}^i &= (\mathcal{P}_i, \mathcal{P}_i^+, \mathcal{P}_i^-, \mathit{low}_i, \mathit{high}_i) \\
V_{sen}^i &= (\mathtt{low}_i, \mathtt{high}_i) & V_{out}^i &= (\mathcal{P}_i, \mathcal{P}_i^+, \mathcal{P}_i^-)\,. \\
V_{cont}^i &= (h_i)
\end{aligned}
$$

The conditional ODEs for the two tank system have been introduced in Example 4.2 in Section 4.2. However, some of the conditions are not linear convex and must be transformed into a set of linear convex conditions as described previously.

The transformed conditional ODE system for tank 1 is presented below.

$$
\begin{aligned}
(c_{1,1}^{h_1}, \mathrm{ODE}_1^{h_1}) &= (\neg \mathtt{P}_1 \wedge \neg \mathtt{P}_2, \quad \dot{h}_1 = 0) \\
(c_{1,2}^{h_1}, \mathrm{ODE}_1^{h_1}) &= (\neg \mathtt{P}_1 \wedge h_1 = F_1, \quad \dot{h}_1 = 0) \\
(c_{1,3}^{h_1}, \mathrm{ODE}_1^{h_1}) &= (\neg \mathtt{P}_1 \wedge h_2 = 0, \quad \dot{h}_1 = 0) \\
(c_{1,4}^{h_1}, \mathrm{ODE}_1^{h_1}) &= (h_2 = F_2 \wedge \neg \mathtt{P}_2, \quad \dot{h}_1 = 0) \\
(c_{1,5}^{h_1}, \mathrm{ODE}_1^{h_1}) &= (h_2 = F_2 \wedge h_1 = F_1, \quad \dot{h}_1 = 0) \\
(c_{1,6}^{h_1}, \mathrm{ODE}_1^{h_1}) &= (h_2 = F_2 \wedge h_2 = 0, \quad \dot{h}_1 = 0) \\
(c_{1,7}^{h_1}, \mathrm{ODE}_1^{h_1}) &= (h_1 = 0 \wedge \neg \mathtt{P}_2, \quad \dot{h}_1 = 0) \\
(c_{1,8}^{h_1}, \mathrm{ODE}_1^{h_1}) &= (h_1 = 0 \wedge h_1 = F_1, \quad \dot{h}_1 = 0) \\
(c_{1,9}^{h_1}, \mathrm{ODE}_1^{h_1}) &= (h_1 = 0 \wedge h_2 = 0, \quad \dot{h}_1 = 0) \\
(c_{2,1}^{h_1}, \mathrm{ODE}_2^{h_1}) &= (\neg \mathtt{P}_1 \wedge \varphi_{2\to1}, \quad \dot{h}_1 = k_2) \\
(c_{2,2}^{h_1}, \mathrm{ODE}_2^{h_1}) &= (h_1 = 0 \wedge \varphi_{2\to1}, \quad \dot{h}_1 = k_2) \\
(c_{2,3}^{h_1}, \mathrm{ODE}_2^{h_1}) &= (h_2 = F_2 \wedge \varphi_{2\to1}, \quad \dot{h}_1 = k_2) \\
(c_{3,1}^{h_1}, \mathrm{ODE}_3^{h_1}) &= (\varphi_{1\to2} \wedge \neg \mathtt{P}_2, \quad \dot{h}_1 = -k_1) \\
(c_{3,2}^{h_1}, \mathrm{ODE}_3^{h_1}) &= (\varphi_{1\to2} \wedge h_1 = F_1, \quad \dot{h}_1 = -k_1) \\
(c_{3,3}^{h_1}, \mathrm{ODE}_3^{h_1}) &= (\varphi_{1\to2} \wedge h_2 = 0, \quad \dot{h}_1 = -k_1) \\
(c_{4,1}^{h_1}, \mathrm{ODE}_4^{h_1}) &= (\varphi_{1\to2} \wedge \varphi_{2\to1}, \quad \dot{h}_1 = k_2 - k_1)
\end{aligned}
$$

Since the conditions $c_{1,6}^{h_1}$ and $c_{1,8}^{h_1}$ cannot be fulfilled for $F_i > 0$, we can remove them and obtain a conditional ODE system with fourteen entries. The conditional ODE system for h_2 is analogous but the derivatives have inverted signs.

We want to analyze whether the tanks can run dry or flood, i. e. if $h_i \leq 0$ or $h_i \geq F_i$ for the following two sets of constants and initial valuations. The first set is used for the benchmarks in Section 9.2 and in Section 9.3 where the two CEGAR approaches are evaluated. The second set of configuration and initial values is used for the evaluation of the advanced analysis methods in Section 9.4. We use a cycle time of δ and a clock t to plot the results against the time.

Table 9.1.: A possible counterexample trace for the first configuration of the two tank system. The variable valuations at the beginning of each PLC cycle are given.

Cycle	h_1	h_2	$\mathtt{P}_1$	$\mathtt{P}_2$	$\mathtt{full}_1$	$\mathtt{high}_1$	$\mathtt{low}_1$	$\mathtt{empty}_1$	$\mathtt{full}_2$	$\mathtt{high}_2$	$\mathtt{low}_2$	$\mathtt{empty}_2$
1	11	11	1	1	0	0	1	0	0	0	1	0
2	**12**	10	1	1	0	**1**	1	0	0	0	1	0
3	**13**	9	1	**0**	0	**1**	1	0	0	0	1	0
4	10	12	1	0	0	0	1	0	0	0	1	0
5	7	15	1	0	0	0	1	0	0	0	1	0
6	4	18	1	0	0	0	1	0	0	0	1	0
7	**1**	21	1	0	0	0	**0**	0	0	0	1	0
8	**0**	24	**0**	**1**	0	0	**0**	**1**	0	0	1	0

Set 1

Constants: $k_1 = 3$, $k_2 = 4$, $L_1 = 2, L_2 = 8.5$, $H_1 = 11.5, H_2 = 28.5$, $F_1 = 20$, $F_2 = 35$, $\delta = 1$

Initially: $h_1 = 11$, $h_2 = 11$, $\mathtt{P}_1 = \mathtt{P}_2 = \mathcal{P}_1 = \mathcal{P}_2 = 1$, $\mathtt{P}_1^+ = \mathtt{P}_2^- = \mathcal{P}_1^+ = \mathcal{P}_2^- = 0$, $\mathtt{low}_1 = \mathtt{low}_2 = \mathit{low}_1 = \mathit{low}_2 = 1$, $\mathtt{high}_1 = \mathtt{high}_2 = \mathit{high}_1 = \mathit{high}_2 = 0$, $t = 0$

Unsafe Set: $\varphi := h_1 \leq 0 \vee h_1 \geq F_1 \vee h_2 \leq 0 \ \vee \ h_2 \geq F_2$

Set 2

Constants: $k_1 = 4$, $k_2 = 3$, $L_1 = 2$, $L_2 = 2$, $H_1 = 15$, $H_2 = 15$, $F_1 = 20$, $F_2 = 20$, $\delta = 1$

Initially: $h_1 = 10$, $h_2 = 10$, $\mathtt{P}_1 = \mathtt{P}_2 = \mathcal{P}_1 = \mathcal{P}_2 = 1$, $\mathtt{P}_1^+ = \mathtt{P}_2^- = \mathcal{P}_1^+ = \mathcal{P}_2^- = 0$, $\mathtt{low}_1 = \mathtt{low}_2 = \mathit{low}_1 = \mathit{low}_2 = 1$, $\mathtt{high}_1 = \mathtt{high}_2 = \mathit{high}_1 = \mathit{high}_2 = 0$, $t = 0$

Unsafe Set: $\varphi := h_1 \leq 0 \vee h_1 \geq F_1 \vee h_2 \leq 0 \ \vee \ h_2 \geq F_2$

Note that the first configuration does not yield a safe system model. A possible counterexample trace is shown in Table 9.1. The model can be corrected by lifting the position of the lower sensor low_1 to a level L_1', where even if the water level of U_1 falls below the position L_1', $\mathtt{P}_1$ can be switched off before the tank becomes empty. For a sensor position $L_1' > 2\delta k_1$ the tank model is safe.

9.2. CEGAR-based Verification I: Bounded Model Checking

In this section, we illustrate how the first proposed methodology for CEGAR in plant control works using the two tank system from Section 3 and Subsection 9.1.3 as benchmark. We want to prove that unsafe states are reachable for Set 1 of constants and initial valuations. For this approach we have to adapt the set φ of unsafe states. It is usually expressed in the physical domain by a predicate. However, for this approach, we introduced the logical sensors $\mathtt{empty}_i$ and $\mathtt{full}_i$ to indicate an empty or a full tank. Thus, we define the set of unsafe states by the equivalent predicate $\varphi_d := \mathtt{empty}_1 \vee \mathtt{full}_1 \vee \mathtt{empty}_2 \vee \mathtt{full}_2$.

The BMC-based CEGAR approach checks PLC executions of increasing length for safety. For the example execution in Table 9.1 we see that an unsafe state is reachable at the end of cycle 7. However, because we read the sensor levels at the beginning of the cycles, a counterexample will be found at the beginning of cycle 8, i. e. in the 8th BMC iteration. The model sizes of the hybrid automata generated for the reachability analysis are shown in Table 9.2.

Table 9.2.: Components of the hybrid automata generated during the BMC iterations.

Cycles	$\|Loc\|$	$\|Edge\|$	$\|V\|$
1	17	240	5
2	33	736	5
3	49	1232	5
4	65	1728	5
5	81	2224	5
6	97	2720	5
7	113	3216	5
8	129	3712	5

Currently, the analysis takes several hours to compute the counterexample for this small benchmark. The reason is that the BMC search has to check up to 2^{88} possible combinations for the sensor values in V_{act} and the logical sensors $\texttt{empty}_i$ and $\texttt{full}_i$ until a counterexample is detected in PLC cycle eight. During this search, 1526 discrete counterexamples have been detected by the BMC, i. e. this is the number of hybrid reachability analysis runs that have to be performed. Currently, we spend most of the time with the generation of the hybrid automata models. However, there is a lot of potential for optimizing. We believe, that with the enhancements that we proposed in Chapter 6, the running times can be strongly improved.

9.3. CEGAR-based Verification II: SpaceEx Integration

All benchmark that are presented in this section have been performed on an Athlon II X4 640 machine with 4GB RAM running Ubuntu 14.04 LTS, 64-bit. We configured SpaceEx to compute flowpipes for a time-horizon of $\delta + \varepsilon$. Since a time-horizon of δ produces the warning that the result is incomplete because the time horizon was reached without exhausting all states, we add $\varepsilon = 0.2$ to the time horizon δ. Each flowpipe segment is computed for a sampling-time of 0.01.

The analysis is started with a hybrid model where arbitrary dynamics is assumed for h_1 and h_2. As the water level of both tanks can change arbitrarily, a first counterexample is detected in the first PLC cycle, where both pumps are switched on. We refine the abstraction in location $(on_1^{in}, on_2^{in}, cycle)$ by adding the conditional ODEs $(c_{4,1}^{h_1}, \text{ODE}_1^{h_1})$ and $(c_{4,1}^{h_2}, \text{ODE}_1^{h_2})$, where the water levels of both tanks are assumed to be between L_i and H_i and both pumps are switched on ($\texttt{P}_i$).

Table 9.3.: Model sizes for the second CEGAR approach.

Model	$\lvert Loc \rvert$	$\lvert Edge \rvert$	$\lvert V \rvert$
Abstraction	20	128	20
Refinement 1	26	224	20
Refinement 2	32	338	20
Refinement 3	51	1174	20
Refinement 4	55	1242	20
Fully refined	80	1808	20

After the refinement, we find another counterexample: In the second PLC cycle, the water level of the first tank reaches the $high_1$ sensor level, thus the SFC decides to switch pump $\mathtt{P}_2$ off to prevent the first tank from flooding and changes the location. However, the pump is switched off at the beginning of the next cycle, such that currently, there is no change in the dynamic behavior of the plant. As a consequence, we refine the abstraction in location $(on_1^{in}, off_2^{in}, cycle)$ by adding the same conditional ODEs $(c_{4,1}^{h_1}, \text{ODE}_1^{h_1})$ and $(c_{4,1}^{h_2}, \text{ODE}_1^{h_2})$ as before.

At the beginning of the third PLC cycle the pump $\mathtt{P}_2$ is switched off, such that the conditions $c_{4,1}^{h_i}$ are no longer fulfilled and the analysis runs into a location with arbitrary behavior of the water levels. We refine $(on_1^{in}, off_2^{in}, cycle)$ again, but this time with the conditional ODEs $(c_{2,1}^{h_1}, \text{ODE}_1^{h_1})$ and $(c_{2,1}^{h_2}, \text{ODE}_1^{h_2})$, i. e. the water level of U_1 is reduced by three units per time unit while the water level of the second tank is increased by three units per time unit.

Again, the analysis is repeated and finally reaches PLC cycle seven with $h_1 = 1$ and $h_2 = 21$. Thus $h_1 < L_1$ and the SFC decides to switch $\mathtt{P}_1$ off and $\mathtt{P}_2$ on to avoid the first tank from running dry. We enter a new location with arbitrary behavior and have to refine it. Thus, location $(off_1^{in}, on_2^{in}, cycle)$ is refined by $(c_{2,1}^{h_1}, \text{ODE}_1^{h_1})$ and $(c_{2,1}^{h_2}, \text{ODE}_1^{h_2})$. Within the PLC cycle time δ, the water levels can reach $h_1 = 0$ and $h_2 = 24$ such that the conditions $c_{2,1}^{h_1}$ and $c_{2,1}^{h_2}$ evaluate to false in the system state. Thus an unsafe state is reached via a fully refined counterexample and the analysis is aborted.

The model sizes of the initial abstraction, the four refined models, and the fully refined model for this benchmark are shown in Table 9.3. Note that the violation of φ could be proven on a much smaller abstract model compared to the fully concretized model. The time for the analysis including the refinement of the automaton model and the search tree was 69.0 seconds. Additionally, the time for the four manual refinement selections was 9.0 seconds. In that time, 237 iterations have been computed, where in each iteration both a flowpipe and all jumps successors that are reachable from states within this flowpipe are computed. Thus, in average more than three iterations have been computed per second. The first refinement was done in the fourth iteration when the first non-urgent location was analyzed. The other refinements were done in iterations 13, 31, 206, and 236.

Table 9.4.: Running times and iterations for different abstractions of the model with a complete restart after each refinement step.

Model	Running time [sec]	Iterations
Abstraction	1.2	4
Refinement 1	1.5	10
Refinement 2	5.6	27
Refinement 3	57.4	195
Refinement 4	60.7	225
Sum	**126.4**	**461**

This two tank benchmark seems to be very small, however, due to the state-space-explosion problem of the parallel composition, the fully concretized automaton model (with the invariants made convex) is unexpectedly large. In the above experiments (as illustrated in Table 9.3) we applied some automated simplification techniques (removing locations whose unreachability can be determined by preprocessing, and removing transitions that can never be enabled). *Without* these simplifications, the fully concretized model for this rather small example would have 1568 locations and 122500 transitions. Many of these reductions were possible mainly due to the special structure of the problem, however, there might be other seemingly simple problems that cannot be simplified to this high extent and lead to hard problems when considering the composed model. These facts indicate that the reachability analysis of PLC-controlled plants can be tackled only in combination with reduction, like the abstraction techniques that we proposed in this thesis.

We also analyzed the performance gain of the tightly integrated CEGAR method. We created the initial fully abstract model and all four refinements in the above case study explicitly, and analyzed each of them in isolation using our adapted SpaceEx version. As before, the analysis gets aborted immediately if the reachability of an unsafe state has been detected, however, now the analysis of concretized models cannot use the results (i.e., cannot backtrack and extend the search tree) of previous analysis steps. The overall running time for all 5 models was 126.4 seconds (see Table 9.4) which nearly doubles the running time of our tightly integrated approach (69.0 seconds).

We did not perform a comparison with an unmodified SpaceEx version for two reasons. Firstly, our modifications cause the tool to stop if the reachability of an unsafe state is detected. However, the unmodified tool performs a complete fixed-point analysis before checking the reachability of unsafe states. As unsafe states are reachable in our models, the full fixed-point computation needs (much) more effort, therefore the comparison would be unfair. Secondly, our models contain urgent transitions which are not supported by SpaceEx.

9.4. Advanced Verification Techniques for PLC-Controlled Plants

Finally, we ran some benchmarks using the advanced verification techniques for PLC-controlled plants that have been integrated in the analysis tool HyPro. Table 9.5 shows the model sizes for the modeling approach from Chapter 4 and the new modeling approach from Section 8.3 where the controller and the plant are separated. The new modeling approach can reduce the number of locations and edges by a noticeable amount, i. e. the number of locations and edges are reduced by up to 89% and 93%, respectively. In the following we use the advanced modeling approach in the experiments.

We created the following variants of all three presented benchmarks from Section 9.1 using the new modeling approach from Section 8.3: The model without enhancements (*orig.*), the model with discrete variables (*disc.*), the model with urgent locations and transitions (*urg.*), and the model with discrete variables and urgent locations and transitions (*urg.,disc.*).

Table 9.5.: Model sizes of the benchmarks. For models with urgency, the continuous dimension is reduced by one since we do not need a clock for the PLC cycle time.

Benchmark	Model	Dimensions		$\lvert Loc \rvert$	$\lvert Edge \rvert$
		continuous	discrete		
Thermostat	original	3	4	12	28
	advanced	3	5	8	18
Leaking Tank	original	3	8	24	104
	advanced	3	9	11	34
Two Tank System	original	4	16	336	4856
	advanced	4	17	38	340

To analyze the influence of discrete variables and urgency on the performance of the analysis tool HyPro we analyzed all four models for each benchmark. For a comparison, we analyzed the model without enhancements also with the tool SpaceEx. For the analysis we used different step sizes, `chull` aggregation, no clustering, and box representations in HyPro and support functions for box templates in SpaceEx. All analyses have been limited by a global time horizon. We performed the benchmarks on a machine with an i7(4790K) and 16GB RAM running Ubuntu 15.10, 64-bit.

The results in Table 9.6 show that both urgency and the support for discrete variables have a noticeable effect on the running times. For the thermostat example the running times for model *orig.* can be reduced by up to 95% for model *disc.* and by roughly 99% for the models *urg.* and *urg.,disc.*. Thus, both enhancements can help to reduce the running time. However, the impact of the support of discrete variables seems to be bigger than the impact of urgency on the running time.

Table 9.6.: Running time comparison for thermostat, the leaking tank and the two tank system benchmarks using different PLC cycle times δ. We used the analysis tools HYPRO and SPACEEX with `chull` aggregation but without clustering. While HYPRO computes box representations using exact numbers, SPACEEX uses support functions with double arithmetic. We denote timeouts (running times > 1800 sec) by TO.

Benchmark	Time Horizon	Stepsize	Running time [sec] HYPRO				SPACEEX
			orig.	*urg.*	*disc.*	*urg.,disc.*	*orig.*
	2	0.010	55.403	2.648	0.047	0.013	0.120
	2	0.005	52.844	2.687	0.072	0.023	0.171
Thermostat	5	0.010	121.374	6.020	0.109	0.026	0.381
$\delta = 0.5$	5	0.005	120.435	5.824	0.153	0.074	0.425
	10	0.010	254.855	12.259	0.284	0.059	1.128
	10	0.005	229.733	10.840	0.304	0.132	0.858
	8	0.010	TO	TO	0.152	0.053	0.390
	8	0.005	TO	TO	0.257	0.092	0.623
Leaking Tank	20	0.010	TO	TO	0.684	0.233	1.598
$\delta = 2$	20	0.005	TO	TO	1.232	0.433	2.606
	40	0.010	TO	TO	4.117	1.395	3.780
	40	0.005	TO	TO	7.156	2.526	6.073
	4	0.010	TO	TO	3.343	1.769	0.792
Two Tank System	4	0.005	TO	TO	4.697	2.384	1.051
$\delta = 1$	10	0.010	TO	TO	290.241	113.342	1.875
	10	0.005	TO	TO	398.068	160.318	2.501
Benchmarks using a special HYPRO version that merges the splitted flowpipes							
	4	0.010	TO	TO	0.424	0.200	0.792
Two Tank System	4	0.005	TO	TO	0.614	0.287	1.051
$\delta = 1$	10	0.010	TO	TO	0.932	0.415	1.875
	10	0.005	TO	TO	1.377	0.635	2.501

When we compare the running times of SPACEEX and HYPRO on the model *orig.*, our benchmarks showed that SPACEEX is superior to HYPRO on the thermostat example by up to 0.2%. This can be explained by the fact that SPACEEX uses double precision floating-point numbers for the computation whereas HYPRO uses exact numbers. Moreover, SPACEEX uses support functions for the state set representations whereas HYPRO computes boxes. At least for higher dimensions the computation of support functions should be faster than the computation of boxes. However, on the model *urg.,disc.* with both enhancements HYPRO is 5 to 19 times faster than SPACEEX on the model *orig.* without enhancements.

Another observation is, that whereas on the thermostat and the leaking tank benchmark HYPRO on *urg.,disc.* is faster than SPACEEX on *orig.*, this changes for the two tank system, where the running times of HYPRO on *urg.,disc.* are much higher than SPACEEX on *orig.*. The reason is the more complex model with two controllers

in combination with the fact that HyPro cannot compute fixed-points whereas the SpaceEx tool can. In each PLC cycle, both controllers perform the SFC computations, but we do not restrict the order in which the controllers are executed. Thus in each cycle, the analysis can branch. Independent of the execution order, both branches reach the same state after both controllers have performed their actions. For the fixed-point analysis, SpaceEx checks for each computed state, if it has already been computed during the reachability analysis. If a state is revisited, the two branches are merged. Since HyPro does not support a fixed-point analysis, we have exponential branching in the analysis which clearly explains the high running times.

We extended the analysis tool HyPro such that flowpipes with identical initial segments are merged. We rerun the two tank benchmark with the new HyPro version that joins the branching flowpipe computations. The results are shown in the lower part of Table 9.6. As expected, the running times of HyPro are much faster than before and the running times of HyPro on *disc.* and *urg.,disc.* are faster than the running times of SpaceEx on the *orig.* model.

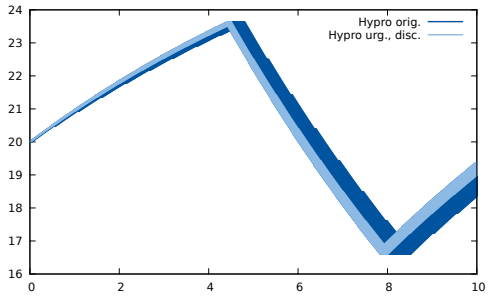
Figure 9.4 shows the reachable states of the thermostat model that have been computed for different parameter configurations using HyPro and SpaceEx. Whereas the support of discrete variables has no effect on the computed set of reachable states, the accuracy of the result can be enhanced by the use of urgency (see Figures 9.4(a) and 9.4(b)). Below, comparison between SpaceEx and HyPro are shown. Although there is not much difference, the over-approximation computed by SpaceEx on the model with no enhancements is bigger than the over-approximation computed by HyPro(see Figures 9.4(c) and 9.4(d)). As a result, SpaceEx needs to compute a second branch where the temperature falls which results purely from over-approximative computations. In the last row, the improved accuracy of HyPro on the enhanced model *urg.,disc.* is compared to the reachable states computed by SpaceEx on the model *orig.* without enhancements (see Figures 9.4(e) and 9.4(f)).

Finally, the accuracy gain due to the advanced analysis techniques that have been implemented in HyPro are shown in Figure 9.5 and in Figure 9.6 for the leaking tank and the two tank benchmarks.

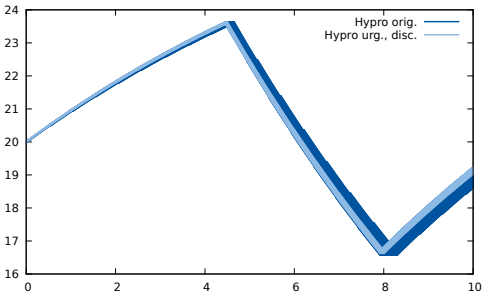
9.5. Outlook

We presented the experimental results for our CEGAR-based analysis approaches and for the advanced analysis techniques that have been implemented in the analysis tool HyPro. We used three different benchmarks that consist of one or two controllers and the controlled plants. Our BMC-based approach runs for several hours on the two tank system. It is likely that the running time can be reduced if the proposed enhancements from Section 6.3 were integrated in our implementation.

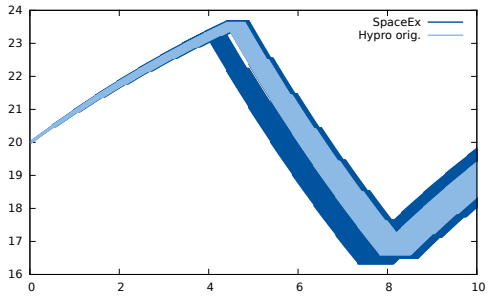
The second CEGAR approach yields better running times. We built the models for each iteration of the CEGAR loop and performed a reachability analysis for each model separately. We could show that the integrated refinement approach nearly halves the running time of the separately performed analyses.



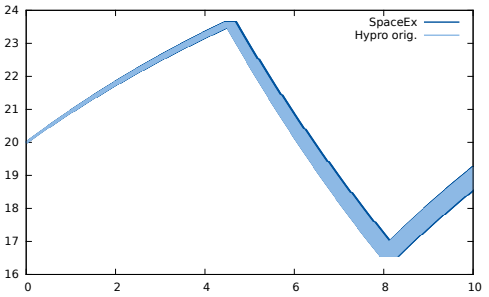
(a) Step size 0.010, HYPRO *orig.* vs. HYPRO *urg.,disc.*



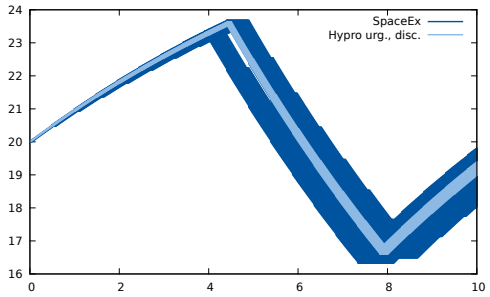
(b) Step size 0.005, HYPRO *orig.* vs. HYPRO *urg.,disc.*



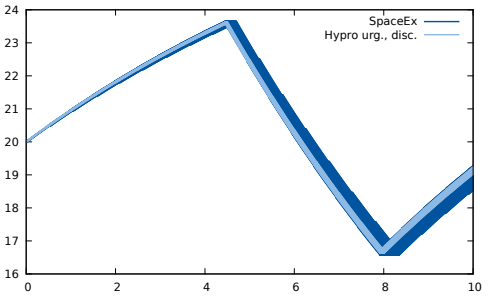
(c) Step size 0.010, SPACEEX *orig.* vs. HYPRO *orig.*



(d) Step size 0.010, SPACEEX *orig.* vs. HYPRO *orig.*



(e) Step size 0.010, SPACEEX *orig.* vs. HYPRO *urg.,disc.*



(f) Step size 0.010, SPACEEX *orig.* vs. HYPRO *urg.,disc.*

Figure 9.4.: SPACEEX and HYPRO results for different parameter sets on the thermostat benchmark.

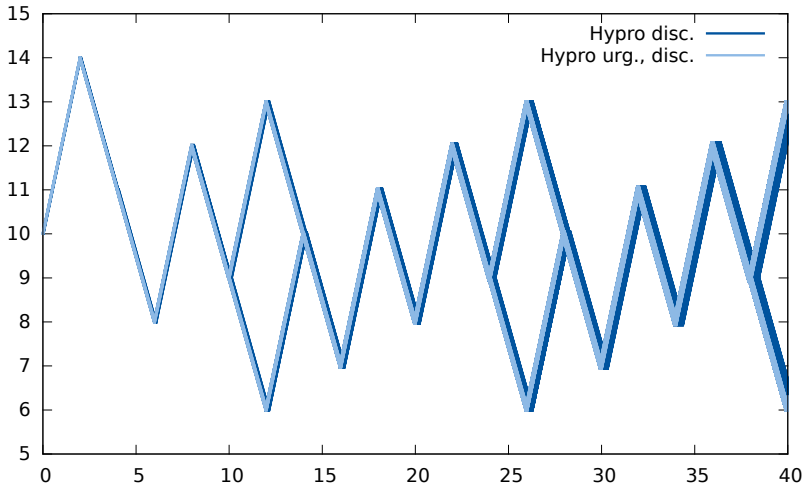


Figure 9.5.: HYPRO results for the leaking tank example with step size 0.005, a global time horizon of 40 and $\delta = 2$.

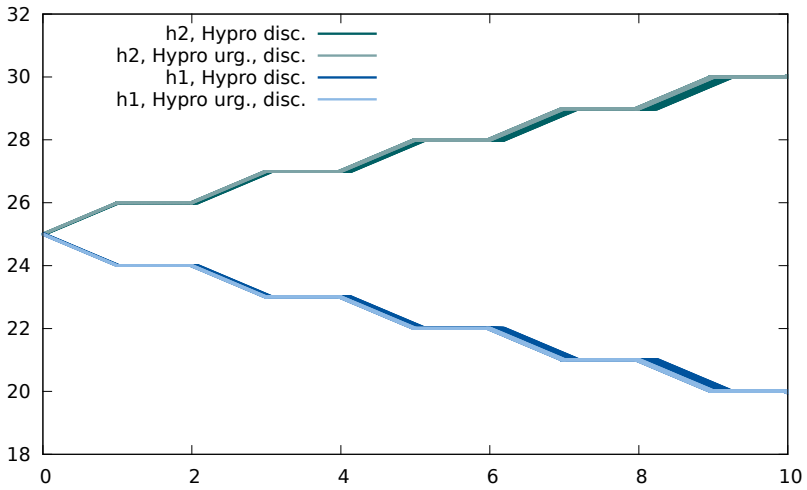


Figure 9.6.: HYPRO results for the two tank system with step size 0.005, a global time horizon of 10 and $\delta = 1$.

Last, we evaluated the impact of the advanced analysis techniques on our models and could show a noticeable reduction of running time and also more accurate state set computations. For these benchmark we used a different parameter configuration for the two tank system benchmark such that the resulting model has been no Zeno behavior. We could observe that models with Zeno behavior produce much longer running times than models without Zeno behavior.

As a conclusion, we we expect that integrating the advanced analysis techniques in our second CEGAR approach will produce better results. Moreover, the running time comparison between HYPRO and SPACEEX indicates that if we develop support for inexact arithmetic computations and complete the implementation of support functions in HYPRO then there is hope to be able to analyze models of practically relevant sizes using the enhancements presented in this thesis.

10. Summary

In this part of the thesis we presented different approaches for the analysis of automated control in chemical plants. First, we showed how we can build hybrid automata models for SFC controllers, for the synchronization of the cyclic controller execution on a PLC, and for the dynamic behavior of a plant. The parallel composition of those automata is the final model for our analysis.

Since the models become quite large, even for relatively small benchmarks, we developed CEGAR-based verification techniques in which we build abstractions with smaller model sizes. If a counterexample is detected for an abstraction, a refinement heuristics is applied to obtain a refined model. Since state-of-the-art tools for flowpipe-computation-based reachability analysis for hybrid systems did not provide counterexamples, we presented different approaches to find presumable counterexamples. It is possible to use model augmentation or to parse information from the output of existing tools to detect counterexamples. For these approaches, no modification of the tools is needed, however, model augmentations affects the model size and both methods have to wait until a complete reachability analysis has been computed before the counterexamples are obtained. A more efficient and accurate way to compute counterexamples is to extend an analysis tool such that during the analysis additional information is stored from which counterexamples can be extracted. Unfortunately, a counterexample can be spurious, which means that it might only occur due to the over-approximative computation of the reachable states and the corresponding path is not possible in the system. Thus, we showed how a possibly spurious counterexample can be validated using simulation. The validating part might be improved by implementing backward refinement of the time intervals on the timed paths, by using a rigorous simulation technique for hybrid automata, and by the development of advanced heuristics for the selection of initial points that are used in the simulation. Moreover, the use of several tools might help to minimize the over-estimation in a presumable counterexample, since we can always take the best result among all tools. Preliminary results have suggested that the function calculus of the tool ARIADNE can be additionally used to validate counterexamples.

Being able to extract counterexamples, we started to develop CEGAR-based verification techniques. Therefore, we needed to create abstractions and a method for the counterexample-guided refinement. We started with a first approach where we extract the discrete behavior of the model and abstract away the continuous dynamics. This allows for the use of efficient analysis tools for discrete models that usually compute counterexamples if unsafe states are reachable. Afterwards, a reachability analysis is performed on the concrete hybrid system which models the dynamic plant behavior. However, this analysis is restricted to the discrete counterexample

which is given in form of a control sequence. Although currently, our prototypical implementation is not competitive concerning the running time, we presented several ideas to improve this methodology. We believe that implementing them will drastically enhance the running time of this approach.

Instead of pursuing the first CEGAR-based methodology any further, we developed a second approach that is purely based on reachability analysis for hybrid automata. We integrated a CEGAR loop into an existing tool for reachability analysis of hybrid automata. We obtain relatively small system models by abstracting away parts of the dynamic behavior. In each refinement iteration, additional dynamic behavior for one or more physical quantities in the plant is added to the model. The experimental results show that this approach is applicable to small benchmarks. However, even the model of a small benchmark consists of 20 dimensions and more than thousand transitions in the fully refined model. As expected, the analysis for such a model is quite time consuming.

To improve the reachability analysis for our models from the area of controlled plants, we added support for discrete variables to remove the dimensionality of our models and direct support for urgency which yields both a reduced running time and a better accuracy. Since we implemented these advanced verification techniques in the analysis tool HYPRO whereas the second CEGAR-based approach has been integrated into the analysis tool SPACEEX, we cannot show the benefits for the CEGAR approach directly. However, combining these approaches will clearly result in reduced running times which enables the verification of bigger benchmarks.

Overall, we can conclude that the verification of controlled plants remains challenging since we have to deal with large system models that need advanced verification techniques for an efficient analysis.

PART II

Synthesis of Control Strategies for Hybrid Vehicles

11. Introduction

In recent years, the focus on fuel consumption and emission of vehicles has increased in society, politics, and the automotive industry. As a result, pure electric vehicles have been placed on the market. However, for these vehicles, there is a trade-off between the possible driving range and the weight of the battery. *Hybrid electric vehicles (HEVs)* are equipped with an *internal combustion engine (ICE)* and an *electrical motor (EM)* and thus benefit from the advantages of both propulsion systems. The ICE allows a wider driving range and is efficient for high torque values. The EM produces no pollutant emission and is efficient for low torque and speed. Thus, the EM can support the combustion engine for a better efficiency.

Here, we consider *parallel hybrid electric vehicles (PHEVs)*, where the ICE and the EM are coupled on the same axis in parallel. Thus, the engines produce the requested torque for driving either separately or in combination. Moreover, the ICE can be used to recharge the battery by generating more torque than requested. Alternatively, electric energy can be recuperated while braking, using the EM as a generator.

A *control strategy* has to distribute the torque requested by the driver between the engines within their specified physical limits. This problem is known as the *energy management problem* for HEVs [GS13]. In the past, heuristic control strategies (e. g., rule-based control strategies [PR07]) as well as control strategies based on optimal control (e. g., equivalent consumption minimization strategies [PDG$^+$02, GS13]) have been proposed. If an estimation on the future driving conditions is available, predictive optimal-control-based strategies can be applied, that use e. g., dynamic programming [BZV$^+$10] or evolutionary programming [PIGV01]. An overview of different kinds of control strategies is given in [PB14].

An online control strategy has to provide a torque split within a limited computation time even if it runs on a control unit of a vehicle with limited processing power. For the DFG research project OASys (Online Algorithms for Optimal Control of Hybrid Propulsion Systems), several real-time capable control strategies have been implemented, i. e. strategies that provide a new torque distribution every 0.02 seconds. Sascha Geulen, Martina Joševski, Michael Tegethoff, and Melanie Winkler developed several non-predictive equivalent consumption minimization strategy (ECMS), and Martina Joševski contributed two predictive control strategies based on dynamic programming. My main contribution to this chapter is a control strategy based on *genetic algorithms (GAs)* [Gol89]. It considers multiple objectives: The overall goal is to minimize the fuel consumption. Secondly, our strategy maximizes the available electric energy while keeping the state of charge of the battery near a reference value to ensure a long battery lifetime. Finally, drivability aspects are considered

during the optimization to increase the driving comfort (e. g., to reduce the noise emission of the internal combustion engine).

Genetic algorithms have already been applied in the context of HEVs. On the one hand, the sizing of powertrain components has been optimized [FQ06]; on the other hand, the parameters of control strategies have been tuned using genetic algorithms [PB14]. In [SAMR10], a GA-based optimization for the scheduling of the electrical generator has been presented for a series hybrid solar vehicle. However, control strategies that are directly built on genetic algorithms are rare: In [WUEK08], a control strategy based on genetic algorithms has been introduced where the optimization is done over the complete driving cycle. Usually, even if the final destination is known in advance, it is difficult to determine a precise prediction for the route to the final destination. Furthermore, due to the computational overhead of an optimization over the whole driving cycle, such a strategy is not real-time capable and can only be used as a reference strategy. In contrast to that, we restrict the optimization to a limited prediction horizon and thus get a real-time capable control strategy.

We implemented a genetic algorithm library called GENEIAL [9] and used it to build a GA-based control strategy. Besides some standard genetic operators that are provided by GENEIAL, the library can easily be extended by customized genetic operators. For our control strategy, we use this feature to create smoothing variants of standard genetic operators to obtain better drivability. The GA-based control strategy optimizes the control by starting a genetic algorithm run for each time step.

We compared our GA-based control strategy with other control strategies that have been implemented for the OASys project. Moreover, we evaluated our control strategy and derived a real-time capable configuration for the genetic algorithm yielding good optimization results. Our control strategy uses a fitness function that computes a weighted sum of a set of objectives. Therefore, we evaluate the influence of various weights on different driving cycles. We show the trade-off between drivability and fuel consumption minimization. Drivability can be obtained by an additional objective in the fitness function or by a search space restriction. We examine the convergence of the genetic algorithm and the influence of the population size and the number of iterations on both the result and the running time for a set of configurations. Finally, we embedded all control strategies into two learning-based control strategies that have been developed by Janosch Fuchs, Sascha Geulen, Michael Tegethoff, and Melanie Winkler and analyzed their performance.

In the following sections, all control strategies that were used to evaluate our GA-based control strategy are presented, although I did not contribute to the development of those strategies, in order to be able to interpret the experimental results at the end of this chapter.

Outline Part II of this thesis is organized as follows. In Chapter 12 the vehicle model, the energy management problem, and the notation used in the context of genetic algorithms are introduced. The `MATLAB/Simulink` simulation model was not my contribution but has been developed by Martina Joševski and Frank-Josef

Heßeler. The C++ vehicle model originated from Ulrich Loup's work and was later maintained and extended by me. We present some basic control strategies in Chapter 13. Most of them have been implemented by Sascha Geulen, Martina Joševski, Michael Tegethoff, and Melanie Winkler. My main contribution to the OASys project has been the development of a GA-based strategy which is described in Subsection 13.2.3. Finally, the learning-based control strategies are introduced in Chapter 14. They have been implemented by Sascha Geulen, Melanie Winkler, Janosch Fuchs, and Michael Tegethoff. Experimental results are presented in Chapter 15. We conclude this part in Chapter 16. A list of the acronyms, variables, and constants used in this part is given in Glossary 16.

12. Preliminaries

To understand the following chapters, some basic knowledge of parallel hybrid electric vehicles (PHEVs), control strategies, and genetic algorithms that we use in our control strategy will be needed.

Outline In Section 12.1, our PHEV model is presented. Afterwards, we introduce the energy management problem (Section 12.2) that a control strategy has to solve in order to distribute the requested torque over the available engines. The content of these first two introductory sections originates from [6, 7] and has been co-written by Martina Joševski. Genetic algorithms that can be used to solve the energy management problem are explained in Section 12.3.

12.1. Vehicle Model

We use a simplified model of a first generation *Toyota Prius*, but with a *parallel hybrid vehicle powertrain* (Figure 12.1), to analyze the performance of our control strategies. This parallel hybrid electric vehicle has an internal combustion engine (ICE) and an electrical motor (EM). Both engines are directly coupled to the same axis which is mounted to a manual transmission gearbox. The gearbox is connected to the wheels using a differential. So, the engines and the gearbox move with the same angular velocity $\omega_{ice} = \omega_{em} = \omega_{req}$. We denote the current gear by h and model gear shifts as discrete changes, i. e. the clutch is always engaged if the vehicle moves. The gear ratio that corresponds to gear h is denoted by r_h and can be used to convert the angular velocity at the gearbox into the angular velocity at the wheels $\omega_{wh} = \omega_{req}/r_h$. The correlation between the speed of the vehicle and the angular velocity at the wheels is given by $v = r_{wh}\omega_{wh}$, where r_{wh} is the wheel radius. If an acceleration/deceleration a is requested by the driver, the necessary torque T_{wh} at the wheels is given by the longitudinal dynamics of the vehicle

$$T_{wh} = r_{wh}\left(\frac{1}{2}\rho C_d A v^2 + (m + m_r)a + mgf_r\cos(\theta) + mg\sin(\theta)\right) ,$$

where ρ is the density of air, C_d the air drag resistance, A the vehicle frontal area, m the mass of the vehicle, m_r the equivalent mass of the rotating parts of the vehicle, g the acceleration of gravity, f_r the rolling resistance, and θ the road slope.

The torque at the wheels results from the torques T_{ice}, T_{em}, and T_{br} of the ICE, EM, and the brakes (BR), respectively and is given by

$$T_{wh} = \eta_{gb} r_h (T_{ice} + T_{em}) - T_{br} ,$$

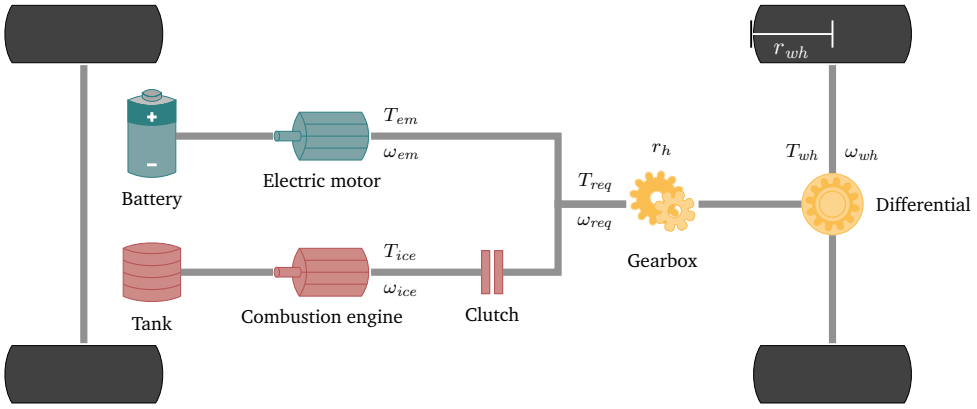


Figure 12.1.: The powertrain configuration of a parallel hybrid electric vehicle.

where η_{gb} is the mechanical transmission efficiency which we assume to be constant. We ignore the dynamics of the internal combustion engine and the electrical motor and assume immediate torque responses. The torque T_{req} that has to be generated by the engines can be obtained by

$$T_{req} = T_{ice} + T_{em} = \frac{T_{wh} + T_{br}}{\eta_{gb} r_h} .$$

The instantaneous fuel consumption $\dot{m}_f$ is given by a function depending on the current angular velocity and the torque produced by the internal combustion engine. We approximate this function by interpolation on a discrete map.

The *battery state of charge (SoC)* can be computed from the input power P_{em} of the electrical motor, the maximum cell capacity $Q_{batt,max}$, the battery current I, and the open circuit voltage U_{oc}. The change of the SoC is given by

$$\dot{SoC} = -\frac{I}{Q_{batt,max}} = -\frac{P_{em}}{U_{oc} Q_{batt,max}} .$$

The input power P_{em} depends on the current angular velocity and torque of the electrical motor. It is obtained by interpolation on a discrete map. We assume that the battery is given by an equivalent circuit where the battery cells are connected in series and we neglect internal losses. To guarantee a long battery lifetime a control strategy has to keep the battery state of charge near a reference value SoC_{ref}.

For a smooth and safe operation of the hybrid powertrain, the following limits are given for the angular velocities and torques of the engines as well as for the battery state of charge:

$$\begin{aligned}
\omega_{ice,min} &\leq \omega_{ice} \leq \omega_{ice,max} \\
\omega_{em,min} &\leq \omega_{em} \leq \omega_{em,max} \\
0 &\leq T_{ice} \leq T_{ice,max}(\omega_{ice}) \\
T_{em,min}(\omega_{em}) &\leq T_{em} \leq T_{em,max}(\omega_{em}) \\
SoC_{min} &\leq SoC \leq SoC_{max}
\end{aligned}$$

The bounds $T_{ice,max}$, $T_{em,min}$ and $T_{em,max}$ are functions of the respective angular velocities and are represented by static maps. Note that T_{ice} and T_{br} are always non-negative. However, T_{em} can have a negative value, in which case the battery is charged. Another way to charge the battery is recuperation of braking energy by opening the clutch at the ICE and using the EM as a generator.

12.2. Energy Management

In PHEVs, both the torque of the internal combustion engine T_{ice} and the torque of the electrical motor T_{em} sum up to the torque at the gearbox T_{req}. At each time step $t \in \{0, \ldots, T\}$ a control strategy computes for a given input a split $u(t) \in \mathbb{Q}_{>0}$ that specifies the amount of torque that is produced by the internal combustion engine. The remaining torque is generated by the electrical motor.

$$T_{ice}(t) = u(t) \cdot T_{req}(t)$$
$$T_{em}(t) = (1 - u(t)) \cdot T_{req}(t)$$

In the following we use T^u_{ice} and T^u_{em} when we explicitly refer to the control function u under which the torques have been determined.

As input a control strategy gets a *driving cycle* which consists of a speed function $v(t)$, a road slope function $\theta(t)$, and a gear function $h(t)$. For a given driving cycle, the *quality* of a split function $u(t)$ is evaluated using an evaluation function $J(u(t), SoC(t), t)$. Common aspects that are considered are the fuel consumption, the battery state of charge, and the drivability of the split function. The aim in the *discretized energy management problem* is to find a cost-minimal split function $u^*(t)$ with value

$$J^* \;=\; \min_{u(\cdot)} \sum_{t=0}^{T} J(u(t), SoC(t), t) \;\;,$$

using the vehicle dynamics given in Section 12.1 and an initial battery state of charge $SoC(0) = SoC_0$. We denote by J^u value of the evaluation function J for a control u. $J^*_{[t,t+T_p]}$ is the solution for the optimal control restricted to a given prediction horizon of length T_p starting at time step t.

In general, the split of the optimal solution $u^*(t)$ for the energy management problem may fluctuate arbitrarily. Due to the noise emission of the ICE for varying torques, this results in bad drivability and reduces the driving comfort. To prevent this, on the one hand, an additional constraint can be added to the optimization that keeps the difference of consecutive splits in a predefined range. On the other hand, an evaluation function can be implemented that prefers consecutive splits with a small deviation.

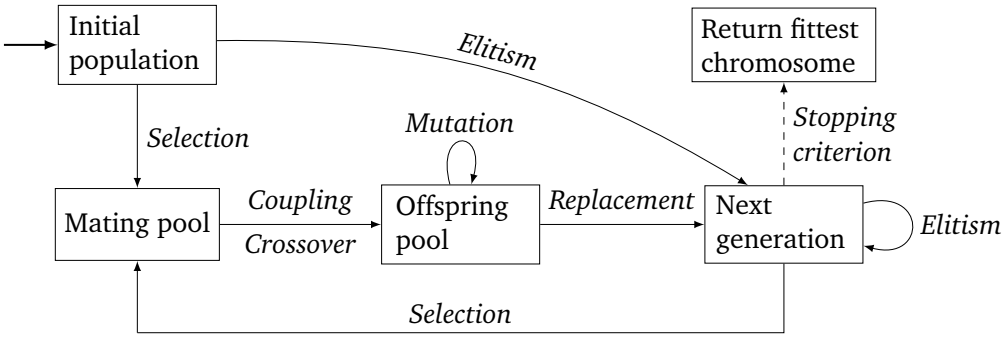


Figure 12.2.: Schematic work flow of a genetic algorithm.

12.3. Genetic Algorithms (GAs)

Genetic algorithms (GA) [Gol89, HH04] belong to the class of stochastic local search algorithms [RN10]. Instead of performing a systematical search, genetic algorithms evolve a set of individual search paths. In general, this class of algorithms does not guarantee optimal solutions. However, for problems with a huge search space, they are fast in computing suitable solutions. In the area of the energy management problem for PHEVs, where a split has to be computed within a limited time frame, local search algorithms are a reasonable choice since they can terminate the computation any time and provide the best solution they have computed so far.

Genetic algorithms are inspired from evolution by natural selection. Figure 12.2 gives a schematic overview of a genetic algorithm. In each iteration (*generation*) of the optimization, a genetic algorithm uses a set of solutions (*population*). Initially, the set of solutions (*initial population*) is arbitrary, often randomly generated. Each solution is called a *chromosome* and can represent a single value (*gene*) or a sequence of values (*genes*). For each chromosome in the population, a value (*fitness*) is computed using an evaluation function (*fitness function*). The solutions with the highest values (*fittest chromosomes*) are selected for generating new solutions by adding them to the *mating pool*. A chromosome c_i is selected with a probability $f_i \;/\; \sum_{j=1}^{K} f_j$ (*fitness-proportional selection*), where f_i is the fitness of chromosome c_i and K is the population size.

New solutions are generated by *crossover* and *mutation*. For crossover, at least two chromosomes from the mating pool are selected (*coupled*) and their solutions are combined. These new chromosomes (*children*) are put to the *offspring pool*. The size of the offspring pool is $\gamma \cdot K$, where γ is called *crossover rate*.

Note that children can be "far away" from the original solutions (*parents*) in the search space. However, in the course of evolution, the solutions converge and the effect of the crossover becomes smaller (*decreasing diversity* in the population). Further randomness is added to the search by *mutation*, where the solutions of the offspring pool are randomly changed with a probability μ (*mutation rate*).

The next generation is built from the chromosomes of the current generation, the offspring pool, and a new set of randomly generated solutions. Usually, the fittest chromosomes, the *elite*, survive in the next generation. Note that this *elitism* guarantees that the best solutions so far are used in the next generation and thus the fitness of the best chromosome cannot decrease. The other chromosomes of the current generation are *replaced* by chromosomes of the offspring pool either randomly or depending on their fitness. If the size of the offspring pool is not sufficient to replace the chromosomes, additional randomly generated chromosomes are added to the next generation.

When the *stopping criterion* has been reached, the optimization terminates and the genetic algorithm outputs the best solution, i. e. the fittest chromosome of the last generation. Different operators and methods for selection, crossover, mutation, elitism and replacement have been proposed in the literature [Gol89].

12.3.1. GeneiAL

We implemented an extensible genetic algorithms library called GENEIAL, which is published under the MIT license [9]. Being the building block of our control strategy, the library was designed to encapsulate the work flow of a genetic algorithm. GENEIAL provides a framework to maximize the fitness value of chromosomes in a population over the course of generations using a custom fitness function for a user-defined optimization problem. The library offers the user high flexibility and extensibility while simultaneously featuring good scalability for the core functionality, e. g., through multi-threading support for evaluating the fitness function. For this purpose, GENEIAL supports user-defined chromosome types. Specifically, it is possible to optimize sequences of genes, i. e. the result is a sequence of optimized values.

The library facilitates the user to specify how initial chromosomes are generated: By default the initial population is filled with chromosomes that have random values within a specified interval. However, for chromosomes with a gene sequence, it is also possible to specify the maximal difference between two consecutive genes. Instead of the generation of random initial populations, GENEIAL also supports the partial or full reuse of a previous population from a prior optimization run.

Additionally, GENEIAL allows the implementation of customized genetic operators. Apart from commonly known genetic operators (e. g., roulette-wheel selection, N-point crossover, and uniform mutation [Gol89]), we implemented problem-specific genetic operations for crossover and mutation of chromosomes with a gene sequence (smoothed crossover, smoothed mutation).

The optimization evolves until a stopping criterion is fulfilled. GENEIAL provides the following basic stopping criteria: The *fitness value criterion* is triggered when the best fitness value of the current generation is greater than or equal to a specified value. Alternatively, the computation can be terminated when a maximal number G_{max} of generations has been produced. The *fixed-point criterion* compares the best fitness value of the current generation with the best fitness value in previous gener-

ations within a specified window size. The computation is stopped when the difference between the fitness values is smaller than a given threshold value. GENEIAL allows to create advanced user-defined stopping criteria as well as combining (via AND, OR, XOR, NOT) multiple stopping criteria.

A set of diagnostic tools enables analyzing the genetic algorithm's behavior, i. e. to measure its running time and to analyze the behavior of single genetic operators. Additionally, GENEIAL can be easily extended by user-specified observers with customized pre- and post-processing logic, e. g., before a new generation is evolved.

13. Basic Control Strategies

In this chapter, we present different control strategies that have been developed for the DFG research project OASys. Control strategies solve the energy management problem to distribute the requested torque over the available engines. We differentiate between *non-predictive* control strategies that have no knowledge about future driving conditions, and *predictive* control strategies that have information over the driving conditions for a limited time frame which is called prediction horizon. All control strategies that are presented in this chapter are real-time capable. A real-time capable control strategy provides a new torque distribution every 0.02 seconds. However, the torque split used to compute this torque distribution can remain fixed for up to one second. Most of the control strategies have been developed by Sascha Geulen, Martina Joševski, Michael Tegethoff, and Melanie Winkler. My contribution is the genetic algorithm (GA)-based control strategy that is presented in Section 13.2.3. Although I did not implement most of the control strategies, I present them in my thesis since we evaluated our strategy against the other control strategies that have been developed for the OASys project.

Outline For the OASys project, both non-predictive and predictive control strategies have been developed. The non-predictive control strategies are presented in Section 13.1. Two naive control strategies that use either the combustion engine or the electrical motor and different equivalent consumption minimization strategies (ECMSs) have been implemented by Sascha Geulen, Martina Joševski, Michael Tegethoff, and Melanie Winkler. The text from Subsection 13.1.2 was published in [6] and has been written by Sascha Geulen and Martina Joševski. In Section 13.2, the predictive control strategies that have been developed for the project, are introduced. Martina Joševski implemented two control strategies that use dynamic programming to solve the energy management problem. She contributed the text for Subsection 13.2.1 and Subsection 13.2.2 that was published in [6]. I contributed the control strategy that uses a GA for optimization, which is presented in Subsection 13.2.3. Section 13.3 concludes this chapter.

13.1. Non-predictive Control Strategies

The control strategies that are presented in this section get the current driving conditions as input but do not have any information about the future.

13.1.1. ICE and EM Control Strategies

Two simple control strategies that can be used as a reference are the strategies that drive purely with the ICE or the EM, respectively, whenever this is feasible. The control strategy ICE uses the EM only for low angular velocities, i. e. for $\omega_{ice} < \omega_{ice,min}$. Analogously, the strategy EM uses the electrical motor whenever this is feasible according to the limits for the angular velocities and torques of the engines as well as for the battery state of charge (cf. Section 12.1). In particular, if $SoC < SoC_{min}$, EM uses the internal combustion engine.

13.1.2. Equivalent Consumption Minimization Strategy (ECMS)

Non-predictive control strategies operate independent of the route, i. e. they do not have any knowledge on the future driving. Such a strategy can either be an all-purpose strategy, or a specialized strategy, e. g. for (extra-)urban driving. Whereas for autonomous strategies, all-purpose control strategies seem to be best-suited, specialized strategies can be effectively combined in learning-based control strategies. Sascha Geulen, Martina Joševski, Michael Tegethoff, and Melanie Winkler implemented different types of ECMS as non-predictive control strategies.

In ECMS, the cost of using the energy stored in the battery is weighted against the cost of using fuel with a time-dependent equivalence factor $s(\tau)$:

$$J^* = \min_{u(\cdot)} \sum_{\tau=t_0}^{t_f} \dot{m}^u_{f,equ}(\tau) \quad \text{with}$$
$$\dot{m}^u_{f,equ}(\tau) = \dot{m}_f(\omega_{ice}(\tau), T^u_{ice}(\tau)) + \frac{s(\tau)}{H_l} P_{em}(\omega_{em}(\tau), T^u_{em}(\tau))$$

under the vehicle dynamics from Section 12.1, where H_l is the lower heating value of the fuel. The equivalent cost function $\dot{m}_{f,equ}$ is the Hamiltonian of the Pontryagin's Minimum Principle (PMP) [GS13]. Thereby, the optimization problem is shifted to the evaluation of the cycle-dependent co-state variable or equivalence factor $s(t)$. Various methods are available for the online estimation of $s(t)$. Here, we apply two known approaches for the equivalence factor computation and denote the corresponding ECMS controllers as *adaptive ECMS (A-ECMS)* and *telemetry ECMS (T-ECMS)*. These two approaches have been developed by Martina Joševski.

13.1.3. Adaptive ECMS (A-ECMS)

The battery has an optimal lifetime if its battery state of charge is kept close to a given reference value SoC_{ref}. In the adaptive ECMS the equivalence factor is determined based on the instantaneous deviations of the battery state of charge from its reference value SoC_{ref}. The adaptation is performed at each time step according to a PI-controller by

$$s(t) = s_0 + \kappa_p(SoC_{ref} - SoC(t)) + \kappa_i \sum_{\tau=t_0}^{t} (SoC_{ref} - SoC(\tau)) \ ,$$

where s_0 is an initial guess, κ_p is the proportional feedback gain, and κ_i is the integral feedback gain.

13.1.4. Telemetry ECMS (T-ECMS)

The value of the optimal equivalence factor depends on the sign of the deviation of the battery state of charge from SoC_{ref} at the end of the driving cycle. Since the final sign of the electric energy is not known during real-time operation, T-ECMS defines a probability function

$$p(t) = \frac{E_e^+(t)}{E_e^+(t) - E_e^-(t)} \ ,$$

where $E_e^+(t)$ and $E_e^-(t)$ denote maximum positive and negative electric energy which may occur at the end of the driving cycle. The equivalence factor $s(t)$ varies in time between values in the interval $[s_{chg}, s_{dis}]$, according to the probability function $p(t)$ using the equation

$$s(t) = p(t)s_{dis} + (1 - p(t))s_{chg} \ .$$

Thereby, the parameters s_{dis} and s_{chg} are the equivalence factors which would correspond to a positive electric energy use and a negative electric energy use at the end of a current mission. Further details on the parameters $E_e^+(t)$, $E_e^-(t)$, s_{chg}, and s_{dis} are presented in [SBG04].

13.2. Predictive Control Strategies

When the entire driving cycle of length T is known in advance, the highest level of information is available and the optimal control sequence can be determined using optimization algorithms. However, in real-world driving situations this is not viable. Instead of performing the optimization over the entire driving cycle, it will be limited to a prediction horizon of length $T_p < T$ time steps. In general, a precise prediction of the future driving conditions is not available. But e. g., a navigation system can provide an estimation of the velocity, road slope, and chosen gear and thus the torque demand for a certain time interval.

At each time step t, the optimization algorithms used here calculate the optimal control sequence for the time interval $[t, t + T_p]$. Only the first value of the resulting control sequence is applied to the powertrain and the optimization algorithm is repeated at each time step with updated hybrid electric vehicle (HEV) state and driving profile values.

Martina Joševski implemented two predictive control strategies based on dynamic programming and I contributed a genetic algorithm based control strategy.

13.2.1. Receding Dynamic Programming (RDP)

In receding dynamic programming the sum over the weighted sum of the fuel mass flow and the deviation of the battery state of charge from SoC_{ref} is minimized for the interval $[t, t+T_p]$:

$$J^*_{[t,t+T_p]} = \min_{u(\cdot)} \sum_{\tau=t}^{t+T_p} w_Q \cdot \dot{m}_f(\omega_{ice}(\tau), T^u_{ice}(\tau)) + w_R \cdot (SoC_{ref} - SoC(\tau))$$

such that the equations from Section 12.1 hold, where w_Q and w_R represent the corresponding weights for the fuel consumption and energy buffer usage. The finite horizon cost is minimized using dynamic programming (DP).

13.2.2. Receding Dynamic Programming with Cost-to-Go Approximation (ADP)

The finite horizon cost is extended by an approximate cost-to-go which represents the cost from the end of the finite prediction horizon until the end of the driving cycle. Thereby, as introduced in [BZV+10], the relation between the Hamilton-Jacobi-Bellman (HJB) equation and the Pontryagin's Minimum Principle is used to determine the cost-to-go term. This term can be approximated by a linear function of the battery state of charge deviation from its reference value. The global fuel minimization problem is converted to a finite horizon optimal control problem with an approximated cost-to-go and the following performance index is minimized:

$$J^*_{[t,t+T_p]} = \min_{u(\cdot)} \sum_{\tau=t}^{t+T_p} \dot{m}_f(\omega_{ice}(\tau), T^u_{ice}(\tau)) + \underbrace{\lambda(SoC(t), t)(SoC_{ref} - SoC(t+T_p))}_{\text{Cost-to-go term}}$$

such that the equations from Section 12.1 hold, where the expression which approximates λ linearly depends on the SoC:

$$\lambda(SoC(t), t) \approx \lambda_0 + \kappa_0(SoC(t) - SoC_{ref})$$

with λ_0 and κ_0 being tuning parameters. Additionally, the parameter λ can be related to the ECMS equivalence factor as follows [BZV+10]:

$$\lambda(SoC(t), t) = \frac{Q_{batt,max} U_{oc}}{H_l} s(t).$$

In the scope of this work an optimal equivalence factor $s(t)$ for a respective driving cycle is used to compute λ.

13.2.3. Genetic Algorithm (GA)

Our GA-based control strategy is a predictive control strategy for computing a near-optimal control. In contrast to control strategies based on DP, our strategy stores a

feasible solution in each optimization step. Thus, it can provide a solution even if its stopping criterion is not yet fulfilled.

In each time step, the strategy gets predictions on the speed of the vehicle, the slope of the road, and the gear used by the driver for all time steps within the prediction horizon, i. e. for the current time step and the following T_p time steps.

It optimizes a split sequence of length $T_p + 1$ and returns only the first value of this sequence as the split for the current time step.

13.2.3.1. Chromosomes

The chromosomes represent split sequences with a split for each time step within the prediction horizon. We use split values $u(i)$ from the set $\{0, 0.01, \ldots, 1\}$ for $i \in \{t, \ldots, t + T_p\}$.

$u(t)$	$u(t+1)$	$u(t+2)$	$\cdots$	$u(t+T_p)$

13.2.3.2. Population

The population is a set $\mathcal{C} = \{c_1, \ldots, c_K\}$ of K chromosomes, where K is fixed for all generations. The initial population is generated uniformly at random, i. e. each chromosome is a sequence of randomly generated splits $c_k = u^k(t) \circ u^k(t+1) \circ \ldots \circ u^k(t+T_p)$ for $c_k \in \mathcal{C}$. Alternatively, we can generate the initial population for a time step $t+1$ by reusing the optimized population of time step t. For this, the split sequence of each chromosome is shifted by one and a random split value is appended for time step $t+T_p+1$, i. e. $c'_k = u^k(t+1) \circ u^k(t+2) \circ \ldots \circ u^k(t+T_p+1)$, where $u^k(t+T_p+1)$ is randomly generated.

13.2.3.3. Fitness Function

The fitness function is used to evaluate the chromosomes. In the area of HEVs, multiple objectives have to be considered: the fuel consumption, the battery state of charge (*SoC*) at the end of the prediction horizon, the difference between *SoC* and SoC_{ref}, and the difference between consecutive splits. We implemented the following fitness functions that compute values within the interval $[0, 1]$ for a prediction horizon of length $T_p + 1$. Each of these fitness functions computes a high fitness value for good chromosomes and a low value for bad ones.

- *Fuel consumption minimization* e_f*:* We estimate a lower bound ($\dot{m}_{f,min}$) and an upper bound ($\dot{m}_{f,max}$) for the instantaneous fuel consumption within the prediction horizon $[t, t+T_p]$ and set

$$e_f := 1 - \left(\frac{1}{T_p+1} \cdot \sum_{\tau=t}^{t+T_p} \dot{m}_f(\omega_{ice}(\tau), T^u_{ice}(\tau)) - \dot{m}_{f,min} \right) / \left(\dot{m}_{f,max} - \dot{m}_{f,min}\right) \quad .$$

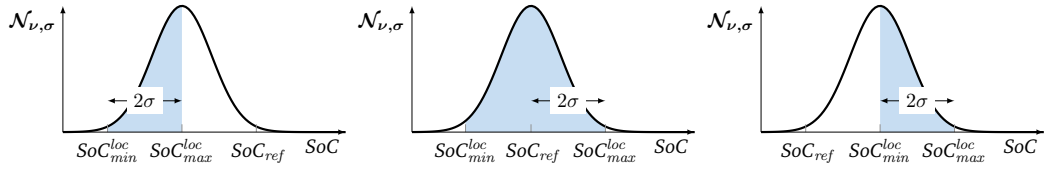


Figure 13.1.: Shifted and scaled normal distribution depending on SoC^{loc}_{min}, SoC^{loc}_{max}, and SoC_{ref}. The area of the interval $[SoC^{loc}_{min}, SoC^{loc}_{max}]$ is filled blue. Left: $SoC^{loc}_{max} < SoC_{ref}$, center: $SoC^{loc}_{min} \leq SoC_{ref} \leq SoC^{loc}_{max}$, right: $SoC^{loc}_{min} > SoC_{ref}$.

Note that this equation scales the average fuel consumption to the interval $[0, 1]$ and that GENEIAL maximizes the fitness function, whereas the fuel consumption has to be minimized.

- *SoC level maximization* e_{sl}*:* We estimate a lower bound (SoC^{loc}_{min}) and an upper bound (SoC^{loc}_{max}) for the battery state of charge at time step $t + T_p$ and set

$$e_{sl} := \left(SoC(t + T_p) \; - \; SoC^{loc}_{min}\right) \; / \; \left(SoC^{loc}_{max} \; - \; SoC^{loc}_{min}\right) \quad .$$

- *SoC deviation minimization* e_{sd}*:* The idea behind this evaluation function is to keep the battery state of charge near SoC_{ref} using a normal distribution function (Figure 13.1). Let $\mathcal{N}_{\nu,\sigma}$ be the normal distribution function with expected value ν and standard deviation σ. We set

$$e_{sd} := \mathcal{N}_{\nu,\sigma}(SoC(t + T_p)) \; / \; \mathcal{N}_{\nu,\sigma}(\nu) \quad ,$$

where ν and σ depend on the estimates of SoC^{loc}_{min}, SoC^{loc}_{max}, and on SoC_{ref}. With $SoC_{diff} := \max\left(|SoC^{loc}_{min} - SoC_{ref}|, |SoC^{loc}_{max} - SoC_{ref}|\right)$, we use

$$(\nu, \sigma) := \begin{cases} (SoC^{loc}_{max}, \; (SoC^{loc}_{max} - SoC^{loc}_{min}) \; / \; 2) & \text{for } SoC^{loc}_{max} < SoC_{ref} \\ (SoC_{ref}, \quad SoC_{diff} \; / \; 2) & \text{for } SoC^{loc}_{min} \leq SoC_{ref} \leq SoC^{loc}_{max} \\ (SoC^{loc}_{min}, \; (SoC^{loc}_{max} - SoC^{loc}_{min}) \; / \; 2) & \text{for } SoC^{loc}_{min} > SoC_{ref} \end{cases} .$$

Thereby, we shift the normal distribution function such that the battery state of charge $SoC \in [SoC^{loc}_{min}, SoC^{loc}_{max}]$ that minimizes $|SoC - SoC_{ref}|$ is mapped to the highest value. Furthermore, the normal distribution function is scaled such that the battery state of charge $SoC \in [SoC^{loc}_{min}, SoC^{loc}_{max}]$ that maximizes $|SoC - SoC_{ref}|$ is mapped to a value near 0.

- *Split difference minimization* e_{ud}*:* Let $[u_{min}, u_{max}]$ be the allowed split range. With $u(-1) = 0$, we set

$$e_{ud} := 1 \; - \; \left(\frac{1}{T_p + 1} \; \cdot \; \sum_{\tau=t}^{t+T_p} (u(\tau) - u(\tau - 1))^2 \right) \; / \; (u_{max} - u_{min})^2 \quad .$$

- *Split difference transgression* e_{ut}*:* Let $\Delta_{u,max}$ be the maximal allowed difference between consecutive splits, $[u_{min}, u_{max}]$ the allowed split range, and $trans_{max} := \Delta_{u,max} - (u_{max} - u_{min})$ the maximal possible transgression of $\Delta_{u,max}$. We set

$$e_{ut} := 1 - \left(\frac{1}{T_p + 1} \cdot \sum_{\tau=t}^{t+T_p} i_\tau \cdot (\Delta_{u,max} - |u(\tau) - u(\tau-1)|)^2 \right) / \, trans_{max}^2 \ ,$$

$$\text{where } i_\tau := \begin{cases} 0 & \text{if } |u(\tau) - u(\tau-1)| < \Delta_{u,max} \\ 1 & \text{else} \end{cases} .$$

The fitness value $f(c_k(t))$ of a chromosome c_k at time step t is a weighted sum of these evaluation functions:

$$f(c_k(t)) = w_f e_f(c_k(t)) + w_{sl} e_{sl}(c_k(t)) + w_{sd} e_{sd}(c_k(t)) + w_{ud} e_{ud}(c_k(t)) + w_{ut} e_{ut}(c_k(t))$$

In each time step, a genetic algorithm runs searches for a suitable solution of the energy management problem, i. e., it searches the chromosome with the highest fitness value.

$$J_{[t,t+T_p]}^{c_k(t)} = \max_{c_k(t)} f(c_k(t)) = \min_{c_k(t)} (1 - f(c_k(t)))$$

Note that the fitness weights are positive numbers that sum up to one. So, the fitness value is in the interval $[0, 1]$. Thus, the maximization problem that is solved by the genetic algorithm can easily be transformed in a minimization problem.

The tuple $(w_f, w_{sl}, w_{sd}, w_{ud}, w_{ut})$ of evaluation weights is called a *fitness configuration* of a GA-based control strategy.

13.2.3.4. Genetic Operators

The genetic operators (selection, coupling, crossover, mutation, elitism, replacement) are used to build a new generation. For selection, we chose *roulette wheel selection* [Gol89] that selects chromosomes based on the idea of spinning a roulette wheel. The space that a chromosome occupies on the wheel corresponds to its probability, i. e., we use a fitness-proportional selection.

The coupling of the chromosomes in the mating pool is done randomly. This means that two chromosomes are chosen uniformly at random for mating.

For crossover, we use different strategies that generate two children for each chromosome couple. N*-point crossover* swaps between the parents' genes at N randomly selected points (Figure 13.2). *Smoothed crossover* is a variant of N-point crossover, where the split differences between neighboring genes of the offspring are kept within $\pm\Delta_{u,max}$ for better drivability.

The offspring is mutated using *uniform mutation*, i. e. a gene is selected uniformly at random for mutation. A set of k randomly selected genes of a chosen chromosome are mutated. This means that the split is replaced by a random value. We also

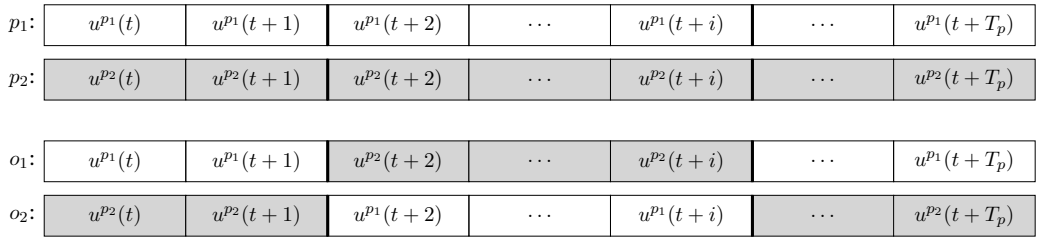


Figure 13.2.: 2-point crossover for the points $(t+1, t+i)$.

implemented *smoothed mutation*, a variant of uniform mutation. After a uniform mutation, the ε-neighborhood of each of the k mutated genes is increased/decreased in the same direction than the mutated gene for better drivability. Thereby, we assure that the split differences of consecutive genes are kept within $\pm\Delta_{u,max}$.

For the new generation, an elite of the chromosomes with the highest fitness is selected and these chromosomes are added to the next generation. Furthermore, we replace the worst chromosomes by new offspring from the offspring pool. If the number of offspring is smaller than the number of chromosomes that should be replaced, we add randomly generated chromosomes to the next generation.

As stopping criterion, the algorithm terminates after G_{max} generations in order to ensure a bounded computation time. The first torque split of the chromosome with the best fitness value $J^*_{[t,t+T_p]} = \max_{c_i(t)} J^{c_k(t)}_{[t,t+T_p]}$ in the population of the current generation is returned as the optimization result.

The GA-configuration $C_{GA} = (K, \gamma, \mu, G_{max})$ of a GA-based control strategy is a tuple that contains the population size, the crossover rate, the mutation rate, and the maximal number of generations. In our implementation, the number of elite chromosomes is given by $(1-\gamma)K$.

13.3. Outlook

In this chapter, the basic control strategies that have been developed for the OASys project have been presented. Although I only contributed the GA-based control strategy, I explained the other control strategies as well, since we used them for the evaluation of our control strategy. The results are shown in Section 15.3. As expected, the predictive control strategies yield better results than the non-predictive control strategies. The main focus of the OASys project has been the implementation of learning-based control strategies that are explained in the following chapter. They use a set of basic control strategies as experts. The basic control strategies that have been presented in this chapter have been integrated in the learning-based control strategies. The experimental results are shown in Section 15.4.

14. Learning-based Control Strategies

In this chapter, two learning-based control strategies that have been developed by Sascha Geulen, Melanie Winkler, Janosch Fuchs, and Michael Tegethoff are introduced. These strategies use two different *online learning* algorithms that have access to a set $\mathcal{E}$ of M strategies. Online learning algorithms belong to the class of online algorithms. An online algorithm computes in each time step an output only based on the input given to the algorithm in the current time step and in previous time steps. So, in general, online algorithms do not require any prediction about the inputs given to the algorithms in the future.

The two learning-based control strategies considered in this thesis are based on *no-regret* online learning algorithms. An online learning algorithm has the no-regret property if it guarantees for each input sequence that the computed output sequence yields a cost that is at most barely larger than the cost of the best strategy in $\mathcal{E}$ for this input sequence chosen from the set $\mathcal{E}$ in hindsight. For a general introduction to online algorithms and no-regret online learning algorithms we refer to [BEY98, BM07]. More details on the development of learning-based control strategies for PHEVs can be found in the PhD thesis of Melanie Winkler [Win13].

In the context of the energy management problem of an HEV, the set $\mathcal{E}$ consists of M basic control strategies. So, the no-regret property ensures that in the long term, the computed torque splits yield a fuel consumption that is at most barely larger than the fuel consumption of the best basic control strategy in $\mathcal{E}$.

Outline In this chapter, the structure of learning-based control strategies are presented in Section 14.1. Afterwards, two learning-based control strategies are presented in Section 14.2 and Section 14.3. The text of these three sections originates from our joint publication [6] and has mainly been written by Sascha Geulen. This chapter is concluded in Section 14.4.

14.1. Structure of the Learning-based Control Strategies

The first learning-based control strategy is called LBCS-SD. LBCS-SD is based on the online learning algorithm *Shrinking Dartboard (SD)*. The second learning-based control strategy is LBCS-WF. This control strategy is based on the online learning algorithm *Weighted Fractional (WF)*. Both SD and WF were introduced in [GVW10]. Furthermore, it was proven in [GVW10] that both SD and WF have the no-regret property. For this reason, the learning-based control strategies presented here are in the long term as good as the best control strategy in $\mathcal{E}$ even without knowledge of the future driving conditions.

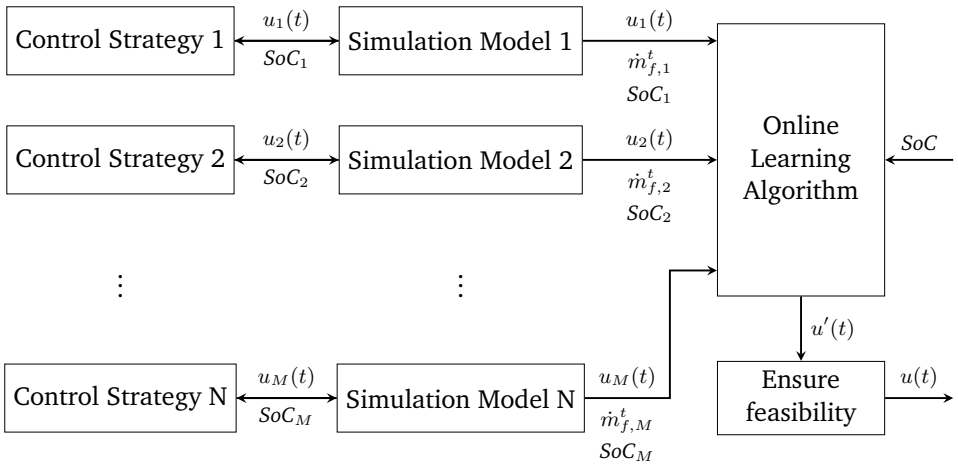


Figure 14.1.: Learning-based control strategies.

Both learning-based control strategies were evaluated with different sets of control strategies. These sets $\mathcal{E}$ were composed of control strategies based on the concepts presented in the previous sections. Some of these control strategies are predictive strategies. Furthermore, some were tuned for specific driving conditions (e. g., city or highway driving).

Fig. 14.1 provides an overview of the learning-based control strategies. The control strategy LBCS-SD uses the online learning algorithm SD in the block *Online Learning Algorithm*, whereas LBCS-WF uses WF. In each time step t, both learning-based control strategies LBCS-SD and LBCS-WF first compute the torque splits $u_i(t)$ of all control strategies $i \in \mathcal{E}$ each using a simplified simulation model of the HEV. Then, both learning-based control strategies pass these torque splits to the online learning algorithm. The online learning algorithm uses the M torque splits to compute a torque split $u'(t)$. The details of the computation are explained in the next sections. Afterwards, both learning-based control strategies check whether the computed $u'(t)$ satisfies the constraints for the angular velocities, the engine torques, and the battery state of charge from Section 12.1.

If not, $u'(t)$ is adjusted in order to ensure feasibility. The adjusted (if necessary) torque split is denoted by $u(t)$. Finally, LBCS-SD and LBCS-WF use $u(t)$ to drive the HEV in the considered time step.

By this procedure, the online learning algorithm evaluates each control strategy based only on the fuel consumption it would have had if it would have been used to drive the HEV. Furthermore, since both SD and WF are no-regret online learning algorithms, the learning-based control strategies guarantee that their fuel consumptions are in the same order as the fuel consumption of the best control strategy in $\mathcal{E}$. This guarantee even holds without knowledge of the driving conditions in advance (as long as the driving cycle is long enough).

14.2. Shrinking Dartboard

Shrinking Dartboard (SD) is a randomized online learning algorithm [GVW10]. In each time step t, SD (Algorithm 4) first reads the torque splits $u_i(t)$ of all control strategies in $\mathcal{E}$ (Lines 2 and 8). Initially, SD chooses a control strategy e^1 in $\mathcal{E}$ uniformly at random (Line 3). The torque split of SD in time step t is the torque split $u_{e^t}(t)$ of the chosen control strategy e^t in time step t (Lines 4 and 14). SD adds to $u_{e^t}(t)$ the term $\varepsilon(SoC_{e^t} - SoC)$. Thereby, the SoC of SD converges to the SoC of the chosen control strategy. The factor ε is a constant that is set initially to some suitable value. For each control strategy $i \in \mathcal{E}$, SD maintains a weight w_i^t. Initially, all weights are set to 1 (Line 1). After each time step, the weights of all control strategies are updated by $w_i^t = w_i^{t-1}(1-\eta)^{\dot{m}_{f,i}^{t-1}}$ (Line 6). So, the new weight of each control strategy depends on its weight in the previous time step, its fuel consumption $\dot{m}_{f,i}^{t-1}$ in the previous time step and the learning rate η of SD, which is a constant that is set initially to some suitable value. SD uses the weights to compute a probability distribution $Q^t = (q_1^t, \ldots, q_M^t)$ with $q_i^t = w_i^t / (\sum_{j=1}^{M} w_j^t)$ (Line 7). With probability $(1-\eta)^{\dot{m}_{f,e^{t-1}}^{t-1}}$, SD uses in the next time step the same control strategy (Line 10). Otherwise, it chooses at random a control strategy e^t using the probability distribution Q^t (Line 12).

Algorithm 4: (`Shrinking Dartboard (SD)`)

1 $w_i^1 = 1, q_i^1 = \frac{1}{M}$, for all $i \in \mathcal{E}$
2 read $u_i(1)$, for all $i \in \mathcal{E}$
3 choose a control strategy e^1 at random according to $Q^1 = (q_1^1, \ldots, q_M^1)$
4 use torque split $u'(1) = u_{e^1}(1)$
5 **for** $t = 2, \ldots, T$ **do**
6 $\quad w_i^t = w_i^{t-1}(1-\eta)^{\dot{m}_{f,i}^{t-1}}$, for all $i \in \mathcal{E}$
7 $\quad q_i^t = \frac{w_i^t}{\sum_{j=1}^{M} w_j^t}$, for all $i \in \mathcal{E}$
8 $\quad$ read $u_i(t)$, for all $i \in \mathcal{E}$
9 $\quad$ **with probability** $(1-\eta)^{\dot{m}_{f,e^{t-1}}^{t-1}}$ **do**
10 $\quad\quad$ set $e^t = e^{t-1}$ (no control strategy change)
11 $\quad$ **else**
12 $\quad\quad$ choose a control strategy e^t at random according to $Q^t = (q_1^t, \ldots, q_M^t)$
13 $\quad$ use torque split $u'(t) = u_{e^t}(t) + \varepsilon(SoC_{e^t} - SoC)$

14.3. Weighted Fractional

Weighted Fractional (WF) is a deterministic online learning algorithm [GVW10]. In each time step t, WF (Algorithm 5) first reads the torque splits $u_i(t)$ of all control strategies in $\mathcal{E}$ (Lines 2 and 7). The torque split of WF is the weighted average over

all torque splits $u_i(t)$ (Lines 3 and 8). WF adds to the weighted average the term $\varepsilon(\sum_{i=1}^{M} q_i^t SoC_i - SoC)$. Thereby, the SoC of WF converges to the SoC of the weighted average over all control strategies. The factor ε is again a constant that is set initially to some suitable value. By w_i^t the weight of control strategy $i \in \mathcal{E}$ in time step t is denoted and by q_i^t its contribution to the weighted average. After each time step, the weights and contributions are updated. The weight of control strategy $i \in \mathcal{E}$ in time step t is $w_i^t = w_i^{t-1}(1-\eta)^{\dot{m}_{f,i}^{t-1}}$ (Line 5) and its contribution is the normalized weight over all control strategies, i. e., $q_i^t = w_i^t / (\sum_{j=1}^{M} w_j^t)$ (Line 6). So, the contribution of each control strategy in the weighted average depends on its weight in the previous time step, its fuel consumption $\dot{m}_{f,i}^{t-1}$ in the previous time step and the learning rate η of WF, which is again a constant that is initially set to some suitable value.

Algorithm 5: (`Weighted Fractional (WF)`)

1 $w_i^1 = 1, q_i^1 = \frac{1}{M}$, for all $i \in \mathcal{E}$
2 read $u_i(1)$, for all $i \in \mathcal{E}$
3 use torque split $u'(1) = \sum_{i=1}^{M} q_i^t u_i(1)$
4 **for** $t = 2, \ldots, T$ **do**
5 $\quad w_i^t = w_i^{t-1}(1-\eta)^{\dot{m}_{f,i}^{t-1}}$, for all $i \in \mathcal{E}$
6 $\quad q_i^t = \frac{w_i^t}{\sum_{j=1}^{M} w_j^t}$, for all $i \in \mathcal{E}$
7 $\quad$ read $u_i(t)$, for all $i \in \mathcal{E}$
8 $\quad$ use torque split $u'(t) = \sum_{i=1}^{M} q_i^t u_i(t) + \varepsilon(\sum_{i=1}^{M} q_i^t SoC_i - SoC)$

14.4. Outlook

In this chapter, two learning-based control strategies have been presented. We integrated the basic control strategies from Chapter 13 as experts into the learning-based control strategies. Whereas SD follows the control of the currently chosen expert in each timestep, WF derives a new control from the computed control values of all experts. In the next section we will evaluate these control strategies on different driving cycles.

15. Experimental Results

In this section, the evaluation of the presented control strategies on three different driving cycles is given. We evaluated the presented control strategies both as standalone strategies but also integrated in two learning-based control strategies.

Outline This chapter starts with an introduction of the benchmark settings that have been used for the evaluation of the control strategies. In Section 15.1, we introduce three driving cycles and give some details of our vehicle model. The evaluation of the basic control strategies as standalone control strategies is presented in Section 15.2. The evaluation has been done collaboratively by Sascha Geulen, Martina Joševski, and me. The text has also been written collaboratively and was published in [7, 6]. The special focus in this thesis is our GA-based control strategy. In Section 15.3, we present the analysis of different parameter configurations for the genetic algorithm to determine real-time capable parameters with good optimization results. The text of this section has been published in [7] and has been written mainly by me but with some support from Sascha Geulen and Benedikt Wolters. Finally, Sascha Geulen and Janosch Fuchs integrated the basic control strategies from Chapter 13 in the learning-based control strategies from Chapter 14 with little help on our `C++` car model from Benedikt Wolters and me. Together, we performed and evaluated some benchmarks. The results are shown in Section 15.4 and have been written collaboratively for [6]. The chapter is concluded in Section 15.5.

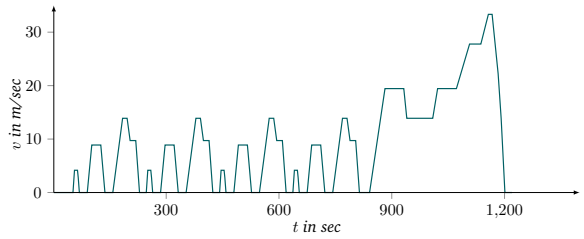
15.1. The Benchmark Settings

Together with Martina Joševski and Sascha Geulen, we chose the following commonly used driving cycles as benchmarks: the *NEDC New European Driving Cycle* (Figure 15.1(a)) is the standard driving cycle used in Europe and is composed of a part of urban and a part of extra-uban driving. The distance of the NEDC is $11.023km$, the total duration is $1220s$, and the average speed is $62.6km/h$. The city program FTP-75, and the highway program HWFET (Figure 15.1(c) and Figure 15.1(b), respectively) of the *EPA Federal Test Procedure (FTP)* are the standard driving cycles used in the US. The distance of FTP-75 is $17.77km$ which is traveled in $1887s$ at an average speed of $34.1km/h$. HWFET lasts $765s$ at an average speed of $77.7km/h$ and a total distance of $16.46km$.

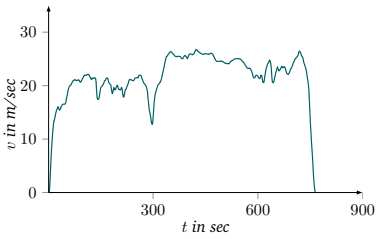
We simulated the benchmarks on the `MATLAB/Simulink` simulation model developed by Martina Joševski and Frank-Josef Heßeler which uses the vehicle dynamics from Section 12.1. In Table 15.1, the settings for the gearbox, the vehicle, the battery, the EM, the ICE, and the constants to compute the energy losses are given.

Table 15.1.: Settings for the gearbox, the vehicle, the battery, the EM, the ICE, and the constants to compute the the energy losses.

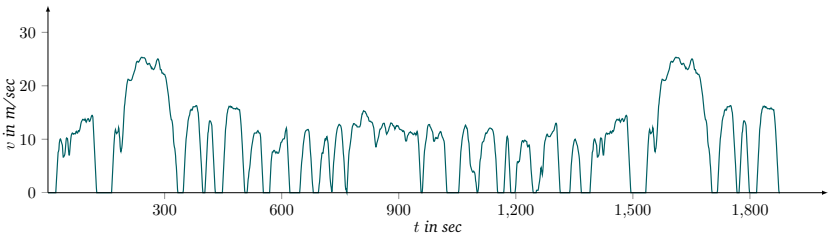
r_i	gear ratio of gear i for $i \in \{1, \ldots, 5\}$ with $\{r_1, \ldots, r_5\} = \{15.17, 8.34, 5.38, 3.94, 2.75\}$		-
η_{gb}	mechanical transmission efficiency	0.98	-
SoC_{min}	minimal battery SoC	0.50	-
SoC_{max}	maximal battery SoC	0.70	-
SoC_{ref}	reference battery SoC	0.60	-
$P_{em,min}$	minimal input power of the EM	0.00	kW
$P_{em,max}$	maximal input power of the EM	280.17	kW
$Q_{batt,max}$	maximal cell capacity of the battery	6.00	Ah
U_{oc}	open circuit voltage of the battery	200.00	V
r_{wh}	wheel radius	0.22	m
A	frontal area of the vehicle	1.80	m^2
m	mass of the vehicle	1440.00	kg
m_r	equivalent mass of the rotating parts of the vehicle	0.05	kg
$\omega_{em,min}$	minimal angular velocity of the EM	0.00	rad/s
$\omega_{em,max}$	maximal angular velocity of the EM	628.32	rad/s
$T_{em,min}$	minimal EM torque	-305.00	Nm
$T_{em,max}$	maximal EM torque	305.00	Nm
$\omega_{ice,min}$	minimal angular velocity of the ICE	104.72	rad/s
$\omega_{ice,max}$	maximal angular velocity of the ICE	418.88	rad/s
$T_{ice,min}$	minimal ICE torque	8.55	Nm
$T_{ice,max}$	maximal ICE torque	101.98	Nm
ρ	density of air	1.18	kg/m^3
C_d	air drag resistance for a Toyota Prius	0.25	-
g	acceleration of gravity	9.81	m/s^2
f_r	rolling resistance	0.00	-



(a) The NEDC is composed of a city and a highway part.



(b) The HWFET is the highway program of the EPA FTP.



(c) The FTP-75 is the city program of the EPA FTP.

Figure 15.1.: The benchmark driving cycles.

In the following, real-time capable control strategies are considered. For real-time capability the current torque T_{req} is distributed between the engines every 0.02 seconds based on a given torque split. However, we allow that the torque split computed by a control strategy can stay fixed for one second. Note that also the learning-based control strategies provide a new distribution every 0.02 seconds based on the incoming torque splits of the basic control strategies.

All control strategies use $SoC_{ref} = 0.6$ and keep the state of charge of the battery in the interval $[0.5, 0.7]$.

In our benchmarks all predictive control strategies use a prediction horizon of length $T_p = 20$ seconds. For this length, a navigation system can provide a sufficiently good approximation on the future driving conditions.

To model the imprecise information given by a navigation system, variants of the three driving cycles with velocity-dependent noise for the prediction are used. Both learning-based control strategies are evaluated with different sets of control strategies that are composed of basic control strategies based on the concepts presented in Section 13.

All control strategies implemented for the OASys project have been integrated into the `MATLAB/Simulink` simulation model. While some of the control strategies have been directly implemented in `MATLAB/Simulink` (the ICE strategy, the EM strategy, some of the equivalent consumption minimization strategies from Section 13.1.2, the predictive control strategies with dynamic programming from Sections 13.2.1 and 13.2.2), others have been developed in `C++` and later been integrated into the simulation model (some of the equivalent consumption minimization strategies from

Section 13.1.2, the predictive control strategy that uses evolutionary programming from Section 13.2.3, and the learning-based strategies from Sections 14.2 and 14.3).

For the following benchmarks, if nothing else is said, a GA-based control strategy was used with 2-point crossover, uniform mutation, and the real-time capable GA-configuration $C_{GA} = (50, 0.75, 0.1, 100)$. Some preliminary simulations showed that for $\eta = 0.5$ and $\varepsilon = 10.0$ the learning-based strategies have the lowest fuel consumption.

15.2. Comparison of the Control Strategies

In this section, an overview of the performance of the basic control strategies that are presented in this thesis is given. In the following, the results of the GA-based control strategy and the learning-based control strategies are discussed in more detail.

Table 15.2 gives the fuel consumption in g and the battery state of charge (*SoC*) at the end of the driving cycle for a set of common control strategies.

Table 15.2.: Fuel consumption in gram and *SoC* at the end of the driving cycle for various control strategies. GA_{best} uses the GA-configuration $C_{GA} = (50, 0.75, 0.1, 100)$ and the fitness configuration $(0.5, 0.0, 0.5, 0.0, 0.0)$ (NEDC, HWFET) or $(0.5, 0.5, 0.0, 0.0, 0.0)$ (FTP-75). The lowest and highest fuel consumption values are typed in black and blue bold print, respectively.

	NEDC		FTP-75		HWFET	
Strategy	Fuel cons.	*SoC*	Fuel cons.	*SoC*	Fuel cons.	*SoC*
ICE	**428.101**	0.7000	**727.606**	0.7000	**379.841**	0.7000
EM	392.686	0.6246	593.519	0.5336	360.903	0.5805
T-ECMS	391.641	0.6246	592.514	0.5335	358.858	0.5805
A-ECMS	390.239	0.6246	592.168	0.5442	356.356	0.5806
ADP	388.726	0.6245	589.266	0.5336	352.513	0.5805
RDP	387.081	0.6246	591.783	0.5614	350.621	0.5807
GA_{best}	**380.123**	0.6263	**581.001**	0.5398	**347.776**	0.5789

Table 15.2 shows that the highest fuel consumption is obtained by the strategy ICE followed by the EM strategy.

However, the control strategies based on optimal control (RDP, ADP, A-ECMS, T-ECMS, GA) that optimize when to use the electrical motor achieve better results. Note that both ECMS does not need a priori knowledge of the driving cycle. The best results are achieved by the GA-based control strategy. The fuel consumption of this strategy is up to 1.8% lower than the results for the other control strategies on all driving cycles.

15.3. Evaluation of the GA-based Control Strategy

In this section, an analysis of our GA-based control strategy with different parameter settings is presented. While Benedikt Wolters performed most of the benchmarks under my supervision, Sascha Geulen joined for the evaluation and the writing [7].

15.3.1. Fitness Evaluation and Drivability

In this section, the results of our GA-based control strategy for different fitness weights is presented. First, we searched for a configuration of fitness weights with optimal results for various driving cycles. In a first set of experiments (Table 15.3), we chose the real-time capable GA-configuration $C_{GA} = (50, 0.75, 0.1, 100)$ and used *fitness configurations* $(w_f, w_{sl}, w_{sd}, w_{ud}, w_{ut})$ that neglect the drivability ($w_{ud} = w_{ut} = 0$).

Table 15.3.: Fuel consumption in gram and *SoC* at the end of the driving cycle for different fitness weights and the real-time capable GA-configuration $C_{GA} = (50, 0.75, 0.1, 100)$. The lowest and highest fuel consumption values are typed in black and blue bold print, respectively.

Fitness weights					NEDC		FTP-75		HWFET	
w_f	w_{sl}	w_{sd}	w_{ud}	w_{ut}	Fuel cons.	*SoC*	Fuel cons.	*SoC*	Fuel cons.	socmath
0.0	0.0	1.0	0.0	0.0	397.632	0.7000	598.475	0.6325	362.076	0.6768
0.0	0.2	0.8	0.0	0.0	397.722	0.7000	598.507	0.6327	362.080	0.6770
0.0	0.4	0.6	0.0	0.0	397.848	0.7000	599.146	0.6348	362.171	0.6777
0.0	0.5	0.5	0.0	0.0	397.913	0.7000	602.638	0.6503	362.287	0.6779
0.0	0.6	0.4	0.0	0.0	423.424	0.7000	711.675	0.7000	373.724	0.7000
0.0	0.8	0.2	0.0	0.0	**424.079**	0.7000	713.216	0.7000	374.075	0.7000
0.0	1.0	0.0	0.0	0.0	424.075	0.7000	**713.247**	0.7000	374.041	0.7000
0.2	0.0	0.8	0.0	0.0	397.058	0.7000	597.775	0.6316	361.880	0.6760
0.2	0.2	0.6	0.0	0.0	397.550	0.7000	597.730	0.6323	361.929	0.6770
0.2	0.4	0.4	0.0	0.0	397.585	0.7000	597.956	0.6330	361.980	0.6775
0.2	0.6	0.2	0.0	0.0	423.832	0.7000	708.784	0.7000	374.160	0.7000
0.2	0.8	0.0	0.0	0.0	424.009	0.7000	711.952	0.7000	374.158	0.7000
0.4	0.0	0.6	0.0	0.0	391.080	0.6820	597.439	0.6307	354.816	0.6410
0.4	0.2	0.4	0.0	0.0	393.618	0.6943	597.439	0.6313	358.780	0.6608
0.4	0.4	0.2	0.0	0.0	397.530	0.7000	597.226	0.6321	361.718	0.6772
0.4	0.6	0.0	0.0	0.0	423.632	0.7000	705.355	0.7000	**374.402**	0.7000
0.5	0.0	0.5	0.0	0.0	**380.123**	0.6263	595.223	0.6205	**347.776**	0.5789
0.5	0.5	0.0	0.0	0.0	382.325	0.6260	**581.001**	0.5398	349.560	0.5789
0.6	0.0	0.4	0.0	0.0	386.894	0.6260	584.393	0.5331	353.313	0.5789
0.6	0.2	0.2	0.0	0.0	386.462	0.6260	583.841	0.5330	351.940	0.5789
0.6	0.4	0.0	0.0	0.0	384.365	0.6260	582.048	0.5330	350.275	0.5789
0.8	0.0	0.2	0.0	0.0	386.914	0.6260	584.897	0.5330	354.080	0.5789
0.8	0.2	0.0	0.0	0.0	384.917	0.6260	582.834	0.5330	352.463	0.5789
1.0	0.0	0.0	0.0	0.0	384.806	0.6260	583.413	0.5330	353.265	0.5789

The quality of the optimization (fuel consumption) depends on the fitness configuration. For bad configurations, the GA-based strategy performs only slightly better

than the ICE strategy; for good configurations, the fuel consumption is within the range of the best control strategies. Good results are achieved for $w_f \in [0.5, 1.0]$. The fitness configuration $(0.5, 0.5, 0.0, 0.0, 0.0)$ yields good results on all driving cycles.

In a second approach (Table 15.4), we considered the drivability evaluation in the fitness function (i. e. $w_{ud} > 0$ or $w_{ut} > 0$). First, we neglected the battery state of charge in the optimization, i. e. we set $w_{sl} = w_{sd} = 0$ (upper part of Table 15.4). However, we considered also the optimization of the fuel consumption, the battery state of charge, and the drivability simultaneously (lower part of Table 15.4).

We use $\sum|.| := \sum_{t=0}^{T} |u(t) - u(t-1)|$ with $u(-1) := 0$ to measure the drivability of the computed split sequences. The best drivability (low $\sum|.|$) is achieved for high w_{ud} values. However, we observe that additional drivability constraints in the fitness evaluation impose a fuel consumption that is only up to 1.7% higher (fitness configuration $(0.0, 0.0, 0.0, 1.0, 0.0)$ on HWFET) than for the fitness configuration $(1.0, 0.0, 0.0, 0.0, 0.0)$ that neglects drivability.

Table 15.4.: Fuel consumption in gram, *SoC* at the end of the driving cycle, and the sum of the split differences of consecutive time steps ($\sum|.|$) for different fitness weights and the GA-configuration $C_{GA} = (50, 0.75, 0.1, 100)$. The lowest and highest values for $\sum|.|$ are typed in black and blue bold print, respectively.

Fitness weights					NEDC			FTP-75			HWFET		
w_f	w_{sl}	w_{sd}	w_{ud}	w_{ut}	Fuel cons.	*SoC*	$\sum\|.\|$	Fuel cons.	*SoC*	$\sum\|.\|$	Fuel cons.	*SoC*	$\sum\|.\|$
0.0	0.0	0.0	0.0	1.0	386.822	0.6259	111.23	589.103	0.5585	149.91	356.827	0.5789	92.64
0.0	0.0	0.0	0.2	0.8	389.248	0.6259	53.93	587.981	0.5366	73.09	359.213	0.5789	45.51
0.0	0.0	0.0	0.4	0.6	389.291	0.6259	52.31	588.118	0.5369	72.50	359.264	0.5789	44.06
0.0	0.0	0.0	0.6	0.4	389.298	0.6259	51.55	588.075	0.5365	71.04	359.266	0.5789	43.54
0.0	0.0	0.0	0.8	0.2	389.317	0.6259	51.31	587.937	0.5358	70.17	359.288	0.5789	43.16
0.0	0.0	0.0	1.0	0.0	389.325	0.6259	51.13	587.916	0.5357	**69.70**	359.271	0.5790	**42.98**
0.2	0.0	0.0	0.0	0.8	387.969	0.6260	71.42	586.399	0.5330	121.31	357.231	0.5789	91.44
0.2	0.0	0.0	0.2	0.6	389.050	0.6259	54.38	587.686	0.5330	83.00	358.585	0.5789	71.68
0.2	0.0	0.0	0.4	0.4	389.223	0.6260	50.53	587.848	0.5330	77.53	358.785	0.5789	63.59
0.2	0.0	0.0	0.6	0.2	389.300	0.6259	50.66	587.853	0.5330	76.23	358.950	0.5789	59.55
0.2	0.0	0.0	0.8	0.0	389.361	0.6260	**48.85**	587.958	0.5330	74.11	359.032	0.5789	57.23
0.4	0.0	0.0	0.0	0.6	387.847	0.6260	69.67	586.394	0.5330	122.72	357.121	0.5789	96.17
0.4	0.0	0.0	0.2	0.4	388.772	0.6260	57.94	587.560	0.5330	90.06	358.265	0.5789	79.69
0.4	0.0	0.0	0.4	0.2	388.998	0.6259	55.09	587.719	0.5330	84.64	358.511	0.5789	72.74
0.4	0.0	0.0	0.6	0.0	389.134	0.6260	52.98	587.746	0.5330	83.94	358.600	0.5789	70.47
0.6	0.0	0.0	0.0	0.4	387.719	0.6260	72.51	586.156	0.5330	126.73	356.879	0.5789	102.95
0.6	0.0	0.0	0.2	0.2	388.545	0.6260	59.99	587.340	0.5330	97.64	357.843	0.5789	91.72
0.6	0.0	0.0	0.4	0.0	388.717	0.6260	58.88	587.438	0.5330	96.18	358.001	0.5789	88.45
0.8	0.0	0.0	0.0	0.2	387.146	0.6259	74.71	585.949	0.5330	136.32	356.095	0.5789	116.63
0.8	0.0	0.0	0.2	0.0	387.960	0.6260	65.64	586.937	0.5330	109.79	357.015	0.5789	107.37
1.0	0.0	0.0	0.0	0.0	384.845	0.6260	**116.64**	583.357	0.5330	**213.93**	353.186	0.5789	**168.29**
0.05	0.05	0.00	0.90	0.00	389.268	0.6259	53.50	587.579	0.5332	73.97	359.145	0.5789	52.66
0.05	0.00	0.05	0.90	0.00	389.228	0.6259	54.26	587.588	0.5338	76.29	359.025	0.5789	52.11
0.05	0.00	0.05	0.80	0.10	389.207	0.6259	52.37	587.481	0.5333	77.73	358.992	0.5789	53.30

In Table 15.5, we restricted the search space to drivable split sequences, i. e., two consecutive splits differ by at most $\Delta_{u,max}$. Small $\Delta_{u,max}$ values improve drivability whereas the fuel consumption is only slightly worse than the results in Table 15.4.

Table 15.5.: Fuel consumption in gram, *SoC* and ($\sum |.|$) at the end of the driving cycle. The fitness configuration $(1, 0, 0, 0, 0)$, the GA-configuration $C_{GA} = (50, 0.75, 0.1, 100)$, and the *smoothing variants* of N-*point crossover* and *uniform mutation* have been used. The lowest and highest values for $\sum |.|$ are typed in black and blue bold print, respectively.

	NEDC			FTP-75			HWFET		
$\Delta_{u,max}$	Fuel cons.	*SoC*	$\sum \lvert.\rvert$	Fuel cons.	*SoC*	$\sum \lvert.\rvert$	Fuel cons.	*SoC*	$\sum \lvert.\rvert$
0.1	388.471	0.6259	**35.19**	588.044	0.5330	**64.68**	358.312	0.5792	**74.11**
0.2	386.700	0.6259	46.06	586.411	0.5330	90.35	352.408	0.5791	99.52
0.3	386.155	0.6259	50.45	585.729	0.5330	109.33	351.386	0.5791	107.31
0.4	386.030	0.6259	55.59	585.263	0.5330	125.73	351.748	0.5791	116.15
0.5	385.980	0.6260	63.01	584.897	0.5330	138.25	352.386	0.5791	127.76
0.6	385.753	0.6259	67.13	584.682	0.5330	153.52	352.851	0.5791	138.66
0.7	385.520	0.6259	76.05	584.370	0.5330	168.72	353.286	0.5791	148.26
0.8	385.385	0.6260	84.87	584.272	0.5330	184.91	353.571	0.5791	161.16
0.9	385.289	0.6260	93.57	584.088	0.5330	199.62	353.777	0.5791	169.61
1.0	385.400	0.6260	**109.26**	584.038	0.5330	**210.10**	353.985	0.5791	**177.49**

15.3.2. Real-time Capability

We examined the convergence behavior of our strategy for population sizes $K \in \{30, 50, 100, 200, 500, 1000\}$. We used the fitness configuration $(1.0, 0.0, 0.0, 0.0, 0.0)$, $\gamma = 0.75$, $\mu = 0.1$, an elite of size $(1 - \gamma)K$, and we terminated the optimization after 100 generations.

Figure 15.2 shows the increase of the fitness values over 100 generations of a genetic algorithm run for different population sizes. The population size influences the maximal fitness value of the initial population and the convergence rate, which influences the quality of the optimization result. However, as a compromise between the quality of the optimization result and the real-time capability of our control strategy, we use a population size of $K = 50$.

15.4. Evaluation of the Learning-based Control Strategies

In this section, the results of the examination of the learning-based control strategies is presented. These benchmarks have been performed mainly by Sascha Geulen with some help from Benedikt Wolters and me. Sascha Geulen as the developer of the learning-based control strategies did the most writing parts, however, I also contributed [6].

Both learning-based control strategies are evaluated with a set of basic control strategies from Section 13. Some of these strategies are tuned for specific driving conditions (e. g. city or highway driving). The results of the learning-based control strategies are compared against the results of the basic control strategies that are used by the learning-based strategies.

Table 15.6 gives the results for the learning-based control strategies LBCS-SD and LBCS-WF, respectively on the following set of basic control strategies: {ICE,

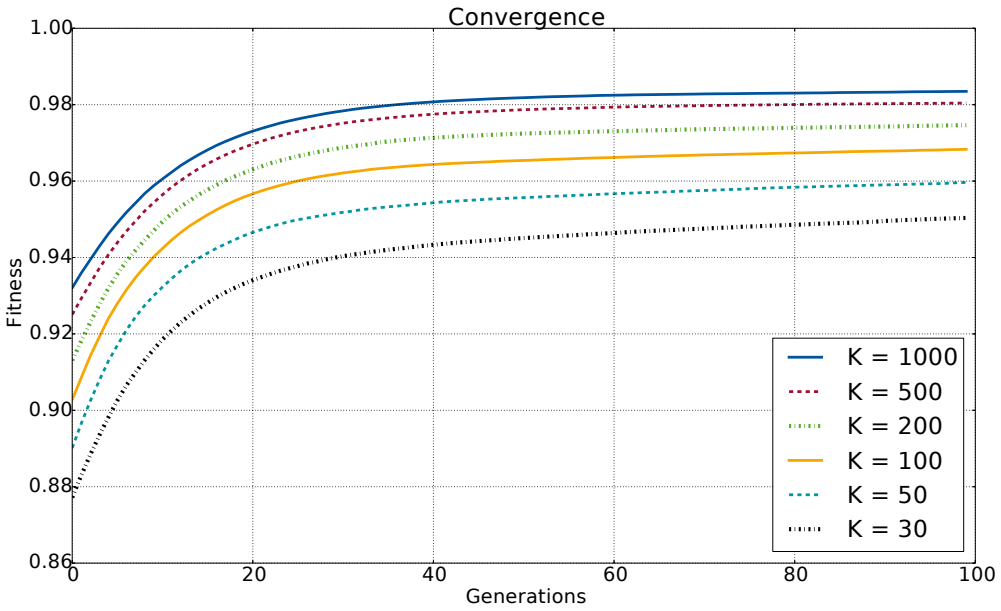


Figure 15.2.: Fitness values of the best chromosome in the population over the generations of a genetic algorithm run with $\gamma = 0.75$ and $\mu = 0.1$ for different population sizes.

A-ECMS, T-ECMS, RDP, ADP, GA1, GA2, GA3}. The three instances GA1, GA2, and GA3 of the GA-based strategy use noised variants of the NEDC, FTP-75, and HWFET driving cycle for prediction. Note that the computation time of the learning-based strategies increases linearly in the number of basic control strategies, however, these computations can be done in parallel. Note that in contrast to Table 15.2, where we list the total fuel consumption, Table 15.6 lists the average consumption.

To model imprecise information given by a navigation system, we use variants of the three driving cycles with velocity-dependent noise for prediction. In the following, we denote by GA_{best} the GA-based strategy with the lowest fuel consumption.

Our results show that the fuel consumption of LBCS-SD is lower than the fuel consumption of the best basic control strategy (GA) on the NEDC and the FTP-75 driving cycle. On HWFET, the GA-based control strategy achieves a slightly better result than LBCS-SD. Nevertheless, the fuel consumption of LBCS-SD is comparable to the fuel consumption of the best GA-based strategy, i. e. LBCS-SD learns the best prediction for the GA-based strategy. LBCS-SD reduces the fuel consumption by up to 0.3% compared to the best basic control strategy (on NEDC) and by up to 28% compared to the combustion engine only strategy (on HWFET).

Figure 15.3(a), 15.3(b), and 15.3(c) show the fuel consumption in $l/100$km for the three driving cycles. After a short learning phase in the beginning of the driving cycle, the fuel consumption of both learning-based control strategies converge to the fuel consumption of the best basic control strategy.

Table 15.6.: Experimental results: Fuel consumption $[l/100\text{km}]$.

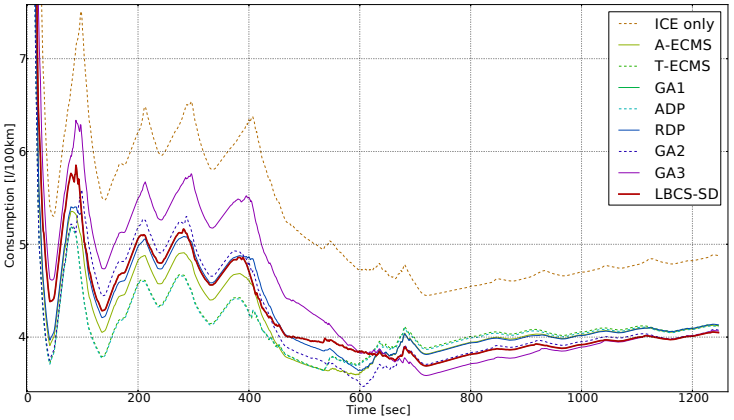
Control strategy	NEDC	FTP-75	HWFET
ICE	4.878	3.213	4.858
A-ECMS	4.121	2.829	3.631
T-ECMS	4.126	2.882	3.519
RDP	4.132	2.841	3.629
ADP	4.111	2.852	3.503
GA_{best}	4.063	2.825	**3.395**
LBCS-SD	**4.049**	**2.824**	3.495
LBCS-WF	4.079	2.826	3.620

15.5. Outlook

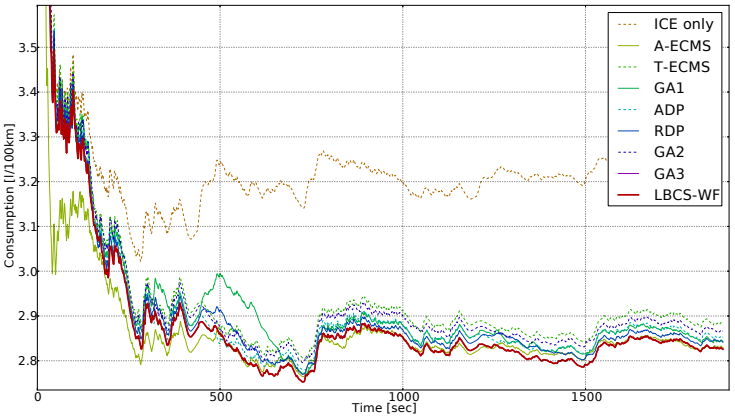
In this chapter, we presented the evaluation of the control strategies that have been implemented for the OASys project. Our optimization aimed at reducing the fuel consumption. As expected, the control strategies with prediction horizon achieve better optimization results than the control strategies without prediction. Different configurations of our GA-based control strategy achieved the best results on all three benchmark driving cycles. A more thorough evaluation of the GA-based control strategy revealed fitness weights and GA-configurations that yield good optimization results and real-time capable computation results. We integrated the basic control strategies into the learning-based control strategies LBCS-SD and LBCS-WF and compared the results of the learning-based control strategies with the results of the basic control strategies. LBCS-SD achieved better results than LBCS-WF on all three benchmarks and the fuel consumption of LBCS-SD was comparable to the fuel consumption of the best basic control strategy.

Although we can adjust the parameters of the genetic algorithm to get a real-time capable control strategy, the heuristic search for the best solution candidate is time consuming and the processing units in vehicles are limited. Thus, in ongoing work which is not part of this thesis we create GA-based control strategies with reduced running times. Therefore, we discretize the state space of our optimization problem and use the genetic algorithm based control strategy to compute the control for a set of grid points offline. These points can either be evenly distributed or with an adaptive step size such that the resolution of more likely areas in the state space is higher. With the help of this grid file we want to build two control strategies that behave similar to the GA-based control strategy but have a reduced running time.

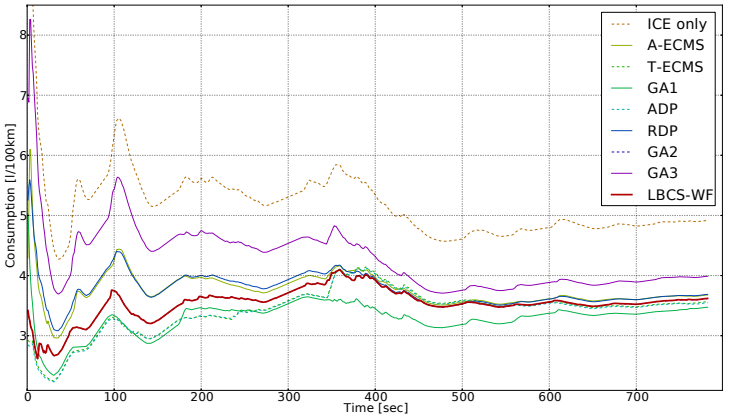
The first of those control strategies uses the grid to interpolate the control for an arbitrary point in the state space from the control of the neighboring precomputed grid points. A second control strategy uses a neural network that has been trained offline with the grid file. We expect these control strategies to approximate our GA-based control strategies with much faster running times.



(a) LBCS-SD on NEDC.



(b) LBCS-WF on FTP-75.



(c) LBCS-WF on HWFET.

Figure 15.3.: Fuel consumption in $[l/100\text{km}]$ of basic control strategies and LBCS-SD or LBCS-WF, respectively on different driving cycles.

16. Summary

In this second part of the thesis, we introduced a set of control strategies that can be either used as standalone strategies or integrated as experts in a learning-based control strategy. Our contribution has been the GA-based control strategy, whereas the other strategies have been developed by our project partners.

We compared the presented control strategies on the driving cycles NEDC, FTP-75, and HWFET. The GA-based control strategy showed promising results, since the fuel consumption of our strategy is comparable to the results of commonly used control strategies. On the driving cycle NEDC, the fuel consumption of the GA-based control strategy has been about 1.8% lower than the fuel consumption of the best considered reference strategy.

However, the results of the GA-based control strategy highly depend of a good parameter setting. Thus, we did a more thorough evaluation of different parameter sets to determine suitable settings. During our experiments, we obtained good results for a crossover rate of of $\gamma = 0.75$, a mutation rate of $\mu = 0.1$, and $(1 - \gamma)K$ elite chromosomes. We analyzed different population sizes and found out that for a size $K = 50$ and an optimization over a maximal number of 100 generations, the GA-based control strategy is still real-time capable and yields sufficiently good results. An evaluation of the fitness function revealed that an even influence of the fuel consumption and either the current battery level (SoC) or the SoC deviation achieves the best results. At the cost of loosing optimality, additional constraints can be added to the control strategy to compute a smooth split sequence which improves the driving comfort. However, our results show that adding these constraints has little effect on the fuel consumption at the end of a driving cycle such that the advantages outweigh the drawbacks. The real-time capability of the GA-based control strategy can be further improved. Currently, a split sequence is computed for a prediction horizon of twenty seconds, but only the first split of this sequence is used by our strategy. Instead of starting the optimization anew at each time point, it would be possible to use the first x computed torque splits from the sequence. The saved computation time could e. g. be used to achieve better optimization results with a higher population size.

In a last experiment, we integrated all presented basic control strategies as experts into the learning-based control strategies LBCS-SD and LBCS-WF. The learning-based control strategies show promising results for the energy management problem of hybrid electric vehicles. Whereas basic control strategies that used a prediction horizon yield better results than control strategies without knowledge of future driving conditions, the learning-based control strategies do not need a prediction horizon. However in our setting, they benefit from control strategies that have knowl-

edge of the future. LBCS-SD has a lower fuel consumption on the NEDC and FTP-75 driving cycles than the best basic control strategy used by the learning-based control strategies. The learning-based control strategies reduced the fuel consumption by up to 0.3%.

Conclusion

Whereas synthesis of new products using advanced techniques from the area of computer science seems to be manageable, the application of formal analysis is still challenging. Working with system models that exhibit discrete and continuous behavior in engineering means that such system are not always specified by mathematical formulas but often given in form of lookup tables, e. g. the engine characteristics of a vehicle. Such data is hard to formalize, and even if a model can be built, formal analysis is still hardly applicable to large systems with complex behavior.

We successfully implemented a GA-based control strategy for HEVs and integrated it into a learning-based control strategy. Although, the running times and memory usage might be an issue when making the step from simulation to real cars, tweaking the GA-configuration might solve the problem. In our current work the GA-based control strategy is used to compute control laws offline. By either learning this control with neural networks or by interpolating a control from the precomputed data, the behavior of the GA-based control strategy can be approximated while the running time is much faster.

The focus of this work has been the formal analysis of hybrid models of SFC-controlled plants. In the last decades, many improvements have been made for the reachability analysis of hybrid systems such that a broad variety of different tools exists. Since those tools originate from academia, the main focus is to develop novel techniques for an improved analysis. However, even for small applications, models can become quite large which results in high running times. The running times could be reduced by highly specialized algorithms for the analysis that exploit the characteristics of a model.

For our CEGAR-based analysis of SFC-controlled plants, we needed a tool that generates counterexamples and that either terminates or interrupts the analysis as soon as a counterexample is detected. Moreover, urgency and a separate handling of discrete and continuous variables during the analysis improved the running times of our benchmarks.

Our work indicates that a library for the formal analysis of hybrid automata is needed, allowing to create specialized analysis algorithms. Unfortunately, such a library does not exist and modifying existing tools currently is the only viable option. However, expert knowledge is needed and it is hard to maintain the changes for new versions of the analysis tool. Living in a world where technology plays an important role, verification of software and hardware systems should play an essential role and should be integrated into the development process. The availability of customizable analysis approaches for hybrid systems would be a first step to make formal analysis for hybrid systems more usable for different application scenarios.

Literature

[ÁBKS05] Erika Ábrahám, Bernd Becker, Felix Klaedtke, and Martin Steffen. Optimizing bounded model checking for linear hybrid systems. In *Proc. of VMCAI'05*, volume 3385 of *LNCS*, pages 396–412. Springer, 2005.

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[AD14] Matthias Althoff and John M. Dolan. Online verification of automated road vehicles using reachability analysis. *Transaction on Robotics*, 30(4):903–918, 2014.

[ADG07] Eugene Asarin, Thao Dang, and Antoine Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43(7):451–476, 2007.

[ADI02] Rajeev Alur, Thao Dang, and Franjo Ivancic. Reachability analysis of hybrid systems via predicate abstraction. In *Proc. of HSCC'02*, volume 2289 of *LNCS*, pages 35–48. Springer, 2002.

[ADI03] Rajeev Alur, Thao Dang, and Franjo Ivancic. Counter-example guided predicate abstraction of hybrid systems. In *Proc. of TACAS'13*, volume 2619 of *LNCS*, pages 208–223. Springer, 2003.

[ADM02] Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In *Proc. of CAV'02*, volume 2404 of *LNCS*, pages 746–770. Springer, 2002.

[ASB08] Matthias Althoff, Olaf Stursberg, and Martin Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *Proc. of CDC'08*, pages 4042–4048. IEEE, 2008.

[Bau04] Nanette Bauer. *Formale Analyse von Sequential Function Charts*. PhD thesis, Universität Dortmund, 2004.

[BCC+06] Andrea Balluchi, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Tiziano Villa, and Alberto L. Sangiovanni-Vincentelli. Ariadne: A framework for reachability analysis of hybrid automata. In *Proc. of MTNS'06*, 2006. http://citeseerx.ist.psu.edu/viewdoc/summary;

jsessionid=38D52AAE5CF9A16A50BBE90316CFAFFC?doi=10.1.1.103.6516.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Proc. of TACAS'99*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.

[BCM13] Olivier Bouissou, Alexandre Chapoutot, and Samuel Mimram. Computing flowpipe of nonlinear hybrid systems with numerical methods. *CoRR*, abs/1306.2305, 2013.

[BCMP98] Luciano Baresi, Stefania Carmeli, Antonello Monti, and Mauro Pezzè. PLC programming languages: A formal approach. In *Proc. of Automation '98*. ANIPLA, 1998.

[BDF+13] Sergiy Bogomolov, Alexandre Donzé, Goran Frehse, Radu Grosu, Taylor T. Johnson, Hamed Ladan, Andreas Podelski, and Martin Wehrle. Abstraction-based guided search for hybrid systems. In *Proc. of SPIN'13*, volume 7976 of *LNCS*, pages 117–134. Springer, 2013.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BFG+14] Sergiy Bogomolov, Goran Frehse, Marius Greitschus, Radu Grosu, Corina Pasareanu, Andreas Podelski, and Thomas Strump. Assume-guarantee abstraction refinement meets hybrid systems. In *Proc. of HVC'14*, volume 8855 of *LNCS*, pages 116–131. Springer, 2014.

[BHLE04] Nanette Bauer, Ralf Huuck, Ben Lukoschus, and Sebastian Engell. A unifying semantics for sequential function charts. In *SoftSpez Final Report, volume 3147 of LNCS*, volume 3147 of *LNCS*, pages 400–418. Springer, 2004.

[Bie09] Armin Biere. Bounded model checking. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 457–481. IOS Press, 2009.

[BM07] Avrim Blum and Yishay Mansour. Learning, regret minimization, and equilibria. In *Algorithmic Game Theory*, chapter 4, pages 79–101. Cambridge University Press, 2007.

[BMP99] Olivier Bournez, Oded Maler, and Amir Pnueli. Orthogonal polyhedra: Representation and computation. In *Proc. of HSCC'99*, volume 1569 of *LNCS*, pages 46–60. Springer, 1999.

[BZV+10] Hoseinali Borhan, Chen Zhang, Ardalan Vahidi, Anthony M. Phillips, Ming L. Kuang, and Stefano Di Cairano. Nonlinear model predictive control for power-split hybrid electric vehicles. In *Proc. of CDC'10*, pages 4890–4895. IEEE, 2010.

[CÁ11] Xin Chen and Erika Ábrahám. Choice of directions for the approximation of reachable sets for hybrid systems. In *Proc. of EUROCAST'11*, volume 6927 of *LNCS*, pages 535–542. Springer, 2011.

[CÁS12] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Proc. of RTSS'12*, pages 183–192. IEEE, 2012.

[CÁS13] Xin Chen, Erika Ábrahám, and Sriram. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of CAV'13*, volume 8044 of *LNCS*, pages 258–263. Springer, 2013.

[CBGV12] Pieter Collins, Davide Bresolin, Luca Geretti, and Tiziano Villa. Computing the evolution of hybrid systems using rigorous function calculus. In *Proc. of ADHS'12*, pages 284–290. IFAC, 2012.

[CBRZ01] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[CFH+03a] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Foundations of Computer Science*, 14(4):583–604, 2003.

[CFH+03b] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Olaf Stursberg, and Michael Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *Proc. of TACAS'03*, volume 2619 of *LNCS*, pages 192–207. Springer, 2003.

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proc. of CAV'00*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

[CK98] Alongkrit Chutinan and Bruce H. Krogh. Computing polyhedral approximations to flow pipes for dynamic systems. In *Proc. of CDC'98*, volume 2, pages 2089–2094. IEEE, 1998.

[CK03] Edmund M. Clarke and Daniel Kroening. Hardware verification using ANSI-C programs as a reference. In *Proc. of ASP-DAC'03*, pages 308–311. IEEE, 2003.

[DKL07] Henning Dierks, Sebastian Kupferschmid, and Kim G. Larsen. Automatic abstraction refinement for timed automata. In *Proc. of FORMATS'07*, volume 4763 of *LNCS*, pages 114–129. Springer, 2007.

[DM11] Parasara S. Duggirala and Sayan Mitra. Abstraction refinement for stability. In *Proc. of ICCPS'11*, pages 22–31. IEEE, 2011.

[DMVP15] Parasara S. Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. C2E2: A verification tool for Stateflow models. In *Proc. of TACAS'15*, volume 9035 of *LNCS*, pages 68–82. Springer, 2015.

[Dri12] Kai Driessen. Counterexample-guided abstraction refinement for hybrid SFC verification. Bachelor thesis, RWTH Aachen University, 2012.

[Dri14] Kai Driessen. Modular verification for PLC controlled hybrid systems. Master thesis, RWTH Aachen University, 2014.

[Egg14] Andreas Eggers. *Direct Handling of Ordinary Differential Equations in Constraint-solving-based Analysis of Hybrid Systems*. PhD thesis, Universität Oldenburg, Germany, 2014.

[ELS05] Sebastian Engell, Sven Lohmann, and Olaf Stursberg. Verification of embedded supervisory controllers considering hybrid plant dynamics. *Software Engineering and Knowledge Engineering*, 15(2):307–312, 2005.

[FCJK05] Ansgar Fehnker, Edmund M. Clarke, Sumit K. Jha, and Bruce H. Krogh. Refining abstractions of hybrid systems using counterexample fragments. In *Proc. of HSCC'05*, volume 3414 of *LNCS*, pages 242–257. Springer, 2005.

[FH06] Martin Fränzle and Christian Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2006.

[FI04] Ansgar Fehnker and Franjo Ivancic. Benchmarks for hybrid systems verification. In *Proc. of HSCC'04*, volume 2993 of *LNCS*, pages 326–341. Springer, 2004.

[FL00] Georg Frey and Lothar Litz. Formal methods in PLC programming. In *Proc. of SMC'00*, volume 4, pages 2431–2436. IEEE, 2000.

[FLD+11] 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 test. In *Proc. of CAV'11*, volume 6806 of *LNCS*, pages 379–395. Springer, 2011.

[FQ06] Li-Cun Fang and Shi-Yin Qin. Concurrent optimization for parameters of powertrain and control system of hybrid electric vehicle based on multi-objective genetic algorithms. In *Proc. of SICE-ICASE'06*, pages 2424–2429. IEEE, 2006.

[Fre08] Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. *Software Tools for Technology Transfer*, 10(3):263–279, 2008.

[Gir05] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Proc. of HSCC'05*, volume 3414 of *LNCS*, pages 291–305. Springer, 2005.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, first edition, 1989.

[GP07b] Antoine Girard and George J. Pappas. Approximation metrics for discrete and continuous systems. *Transactions on Automatic Control*, 52(5):782–798, 2007.

[GPB05] Nicolo Giorgetti, George J. Pappas, and Alberto Bemporad. Bounded model checking of hybrid dynamical systems. In *Proc. of CDC'05*, pages 672–677. IEEE, 2005.

[GS13] Lino Guzzella and Antonio Sciarretta. *Vehicle Propulsion Systems: Introduction to Modeling and Optimization*. Springer, third edition, 2013.

[GVW10] Sascha Geulen, Berthold Vöcking, and Melanie Winkler. Regret minimization for online buffering problems using the weighted majority algorithm. In *Proc. of COLT'10*, pages 132–143, 2010. http://colt2010.haifa.il.ibm.com/papers/COLT2010proceedings.pdf#page=140.

[Hae16] Rebecca Haehn. Learning control strategies for hybrid vehicles using neural networks. Bachelor thesis, RWTH Aachen University, 2016.

[Hap13] Kim M. Haps. Datatypes and tools for the analysis of hybrid systems. Bachelor thesis, RWTH Aachen University, 2013.

[HH04] Randy L. Haupt and Sue E. Haupt. *Practical Genetic Algorithms*. Wiley-Interscience, second edition, 2004.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1-2):110–122, 1997.

[HKD98] George Hassapis, Isabella Kotini, and Zoe Doulgeri. Validation of a SFC software specification by using hybrid automata. In *Proc. of INCOM'98*, pages 65–70. Pergamon, 1998.

[HKJM13] Martin Herceg, Michal Kvasnica, Colin N. Jones, and Manfred Morari. Multi-Parametric Toolbox 3.0. In *Proc. of ECC'13*, pages 502–510, 2013. http://control.ee.ethz.ch/~mpt.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Computer and System Sciences*, 57(1):94–124, 1998.

[HMR14] Willem Hagemann, Eike Möhlmann, and Astrid Rakow. Verifying a PI controller using SoapBox and Stabhyli: Experiences on establishing properties for a steering controller. In *Proc. of ARCH'14*, volume 34 of *EPiC Series in Computer Science*. EasyChair, 2014.

[HyC] HyCreate: A tool for overapproximating reachability of hybrid automata. http://stanleybak.com/projects/hycreate/hycreate.html.

[HyR] HyReachhome page. https://embedded.rwth-aachen.de/doku.php?id=en:tools:hyreach.

[Imm15] Fabian Immler. Tool presentation: Isabelle/HOL for reachability analysis of continuous systems. In *Proc. of ARCH14-15*, volume 34 of *EPiC Series in Computer Science*, pages 180–187. EasyChair, 2015.

[Int03] Int. Electrotechnical Commission. *Programmable Controllers, Part 3: Programming Languages, 61131-3*, 2003.

[Ion12] Alin Ionascu. Modeling and controler synthesis of hybrid propulsion systems using artificial intelligence. Master thesis, RWTH Aachen University, 2012.

[JKWC07] Sumit K. Jha, Bruce H. Krogh, James E. Weimer, and Edmund M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In *Proc. of HSCC'07*, LNCS, pages 287–300. Springer, 2007.

[KGCC15] Soonho Kong, Sicun Gao, Wei Chen, and Edmund M. Clarke. dReach: δ-reachability analysis for hybrid systems. In *Proc. of TACAS'15*, volume 9035 of *LNCS*, pages 200–205. Springer, 2015.

[Küh98] Wolfgang Kühn. Zonotope dynamics in numerical quality control. In *Mathematical Visualization: Algorithms, Applications and Numerics*, pages 125–134. Springer, 1998.

[KV02] Alexander B. Kurzhanski and Pravin Varaiya. On ellipsoidal techniques for reachability analysis. *Optimization Methods and Software*, 17(2):177–237, 2002.

[KV06] Alex A. Kurzhanskiy and Pravin Varaiya. Ellipsoidal toolbox. Technical report, EECS, UC Berkeley, 2006.

[Le 09] Colas Le Guernic. *Reachability Analysis of Hybrid Systems with Linear Continuous Dynamics*. PhD thesis, Université Joseph Fourier, 2009.

[LG09] Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In *Proc. of CAV'09*, volume 5643 of *LNCS*, pages 540–554. Springer, 2009.

[Luk05] Ben Lukoschus. *Compositional Verification of Industrial Control Systems - Methods and Case Studies*. PhD thesis, Christian-Albrechts-Universität zu Kiel, 2005.

[MF14] Stefano Minopoli and Goran Frehse. Non-convex invariants and urgency conditions on linear hybrid automata. In *Proc. of FORMATS'14*, volume 8711 of *LNCS*, pages 176–190. Springer, 2014.

[MT00] Ian Mitchell and Claire J. Tomlin. Level set methods for computation in hybrid systems. In *Proc. of HSCC'00*, volume 1790 of *LNCS*, pages 310–323. Springer, 2000.

[Ost13] Thomas Osterland. Memory- and time-related action qualifiers in HSFCs. Bachelor thesis, RWTH Aachen University, 2013.

[PB14] Aishwarya Panday and Hari O. Bansal. A review of optimal energy management strategies for hybrid electric vehicle. *Vehicular Technology*, 2014:1–19, 2014. Article ID 160510.

[PDG+02] Gino Paganelli, Sebastien Delprat, Thierry-Marie Guerra, Janette Rimaux, and Jean-Jacques Santin. Equivalent consumption minimization strategy for parallel hybrid powertrains. In *Proc. of VTC'02*, pages 2076–2081. IEEE, 2002.

[PDMV13] Pavithra Prabhakar, Parasara S. Duggirala, Sayan Mitra, and Mahesh Viswanathan. Hybrid-automata-based CEGAR for rectangular hybrid systems. In *Proc. of VMCAI'13*, volume 7737 of *LNCS*, pages 48–67. Springer, 2013.

[PIGV01] Antonio Piccolo, Lucio Ippolito, Vincenzo Galdi, and Alfredo Vaccaro. Optimisation of energy flow management in hybrid electric vehicles via genetic algorithms. In *Proc. of ASME'01*, pages 434–439. IEEE, 2001.

[PQ08] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems. In *Proc. of IJCAR'08*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008.

[PR07] Pierluigi Pisu and Giorgio Rizzoni. A comparative study of supervisory control strategies for hybrid electric vehicles. *Transactions on Control Systems Technology*, 15(3):506–518, 2007.

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.

[RPV16] Nima Roohi, Pavithra Prabhakar, and Mahesh Viswanathan. Hybridization based CEGAR for hybrid automata with affine dynamics. In *Proc. of TACAS'16*, volume 9636 of *LNCS*, pages 752–769. Springer, 2016.

[RS05] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *Proc. of HSCC'05*, volume 3414 of *LNCS*, pages 573–589. Springer, 2005.

[SAMR10] Marco Sorrentino, Ivan Arsie, Raffaele Di Martino, and Gianfranco Rizzo. On the use of genetic algorithm to optimize the on-board energy management of a hybrid solar vehicle. *Oil & Gas Science and Technology*, 65(1):133–143, 2010.

[SBG04] Antonio Sciarretta, Michael Back, and Lino Guzzella. Optimal control of parallel hybrid electric vehicles. *Transactions on Control System Technology*, 12(3):352–363, 2004.

[Seg07] Marc Segelken. Abstraction and counterexample-guided construction of ω-automata for model checking of step-discrete linear hybrid models. In *Proc. of CAV'07*, volume 4590 of *LNCS*, pages 433–448. Springer, 2007.

[SK03] Olaf Stursberg and Bruce H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In *Proc. of HSCC'03*, volume 2623 of *LNCS*, pages 482–497. Springer, 2003.

[SKB13] Karsten Scheibler, Stefan Kupferschmid, and Bernd Becker. Recent improvements in the SMT solver iSAT. In *Proc. of MBMV'13*, pages 231–241, 2013. http://www.avacs.org/fileadmin/Publikationen/Open/scheibler.mbmv2013.pdf.

[Spa] State space explorer: Spaceex. http://spaceex.imag.fr/.

[Str04] Ofer Strichman. Accelerating bounded model checking of safety properties. *Formal Methods in System Design*, 24(1):5–24, 2004.

[TD13] Romain Testylier and Thao Dang. NLTOOLBOX: A library for reachability computation of nonlinear dynamical systems. In *Proc. of ATVA'13*, volume 8172 of *LNCS*, pages 469–473. Springer, 2013.

[Wes16] Patricia Wessel. Grid-based control strategy for hybrid vehicles. Bachelor thesis, RWTH Aachen University, 2016.

[Win13] Melanie Winkler. *Algorithms for Online Buffering Problems and Applications to the Power Control of a Hybrid Electric Vehicle*. PhD thesis, RWTH Aachen University, 2013.

[WUEK08] Randombage S. Wimalendra, Lanka Udawatta, Eran Edirisinghe, and Suranga Karunarathna. Determination of maximum possible fuel economy of HEV for known drive cycle: Genetic algorithm based approach. In *Proc. of ICIAFS'08*, pages 289–294. IEEE, 2008.

Notations

Part I – Analysis of Automated Control in Chemical Plants

a	An action $a \in \mathcal{A}$ of an SFC.
act	The finite set of variables for the actuators of the plant in the BMC approach.
active	A function that maps a location and a variable to the set of active conditional ordinary differential equations (ODEs).
Act	The set of activities of a hybrid automaton.
$\mathcal{A}$	The set of actions of an SFC.
$\mathcal{A}_a$	The set $\mathcal{A}_a \subseteq \mathcal{A}$ of active actions of an SFC. $\mathcal{A}_a^0$ denotes the initial set of active actions.
b	An action block $b = (q, a) \in B$ of an SFC is a tuple of an action qualifier $q \in \{\texttt{entry}, \texttt{do}, \texttt{exit}\}$ and an action $a \in \mathcal{A}$.
B	$B \subseteq \{\texttt{entry}, \texttt{do}, \texttt{exit}\} \times \mathcal{A}$ is the set of action blocks of an SFC.
$\mathcal{B}$	A function $\mathcal{B} : \mathcal{S} \to 2^{B_\mathcal{A}}$ that assigns a set of action blocks to each step of an SFC.
$\mathbb{B}$	The set $\{0, 1\}$ of Boolean values representing *true* and *false*, respectively.
c	The condition of a conditional ODE.
cl	A function that computes the closure of a given set.
$\mathcal{C}$	An SFC $\mathcal{C} = (V_\mathcal{C}, \mathcal{S}, \mathcal{A}, s_0, \mathit{Trans}, \mathcal{B}, \sqsubset, \prec, \mathit{Hist})$.
$\overline{\mathcal{C}}$	The set of the toplevel SFC $\mathcal{C}$ and all its nested sequential function charts (SFCs) at all depths.
d	A transition $r = (S, g, S') \in \mathit{Trans}$ of an SFC that leads from a set S of source steps to a set S' of target steps, if the guard g is fulfilled by the current state.
D	The union of the data type domains for all variables in a set of variables.
e	An edge $e = (l, \alpha, g, r, l') \in \mathit{Edge}$ of a hybrid automaton.
empty	The SFC variable for the sensor that detects an empty tank in the tank benchmarks.

`empty`	The plant variable for the sensor that detects an empty tank in the tank benchmarks.
`enabled`	A function `enabled` : $(Loc \times \mathcal{V}^{V_\mathcal{H}}) \to 2^{Edge}$ that returns the outgoing edges of the current location whose guards evaluate to *true* under the current valuation.
Edge	The set $Edge \subseteq Loc \times Lab \times 2^{\mathcal{V}^{V_\mathcal{H}}} \times (\mathcal{V}^{V_\mathcal{H}} \to 2^{\mathcal{V}^{V_\mathcal{H}}}) \times Loc$ of edges (jumps) in a hybrid automaton.
$\mathcal{E}$	The set $\mathcal{E} \subseteq Trans$ of enabled transitions of an SFC.
f	An activity is a function $f : \mathbb{R}_{\geq 0} \to \mathcal{V}^{V_\mathcal{H}}$ which describes the evolution of the values of continuous variables $V_\mathcal{H}$ over time.
full	The SFC variable for the sensor that detects a full tank in the tank benchmarks.
`full`	The plant variable for the sensor that detects a full tank in the tank benchmarks.
F	Constant for the position of the full sensor in the tank benchmarks.
g	A guard $g \subseteq 2^{\mathcal{V}^{V_\mathcal{H}}}$ of an edge in a hybrid automaton with variable set $V_\mathcal{H}$.
h	The variable that models the waterlevel of a tank in the tank benchmarks.
high	The SFC variable for the sensor that detects a high waterlevel in the tank benchmarks.
`h`	The heating constant in the thermostat benchmark.
`high`	The plant variable for the sensor that detects a high waterlevel in the tank benchmarks.
H	Constant for the position of the high sensor in the tank benchmarks.
Hist	The history flag of an SFC.
$\mathcal{H}$	A hybrid automaton $\mathcal{H} = (Loc, Lab, V_\mathcal{H}, Edge, Act, Inv, Init, Urg)$.
I	A quantifier-free arithmetic formula over the variables $V_\mathcal{C}$ that defines the set of initial states of a PLC program for bounded model checking.
Init	The set $Init \subseteq Loc \times \mathcal{V}^{V_\mathcal{H}}$ of initial states of a hybrid automaton.
Inv	A function $Inv : Loc \to 2^{\mathcal{V}^{V_\mathcal{H}}}$ that assigns an invariant to each location of a hybrid automaton.
k	Constant for the waterlevel increase/decrease of a tank per time unit in the tank benchmarks.

$\mathtt{K}$	The plant variable for the heater from the thermostat benchmark.
$\mathcal{K}$	The SFC variable for the heater from the thermostat benchmark.
l	A location $l \in \mathit{Loc}$ of a hybrid automaton.
l_0	An initial location $l_0 \in \mathit{Loc}$ of a hybrid automaton.
low	The SFC variable for the sensor that detects a low waterlevel in the tank benchmarks.
`low`	The plant variable for the sensor that detects a low waterlevel in the tank benchmarks.
L	Constant for the position of the low sensor in the tank benchmarks.
Lab	The finite set of labels of a hybrid automaton.
Loc	The set of locations of a hybrid automaton.
max_{tr}	An upper bound for the counter *tr* that counts the taken transitions along a path in a hybrid automaton.
$\mathbb{N}$	The set of natural numbers including 0. The set of positive natural numbers is denoted by $\mathbb{N}_{>0}$.
o	The differential equation of a conditional ODE.
Prog	$\mathit{Prog} = (I, \mathit{Tr}, \mathit{Prop})$ encodes the semantics of a PLC program that is used for bounded model checking.
Prop	A quantifier-free arithmetic formula over the variables $V_{\mathcal{C}}$ that defines the set of safe states of a PLC program for bounded model checking.
`P`	The plant variable for a pump from the tank benchmarks.
$\mathtt{P}^+$	The plant variable for the request to switch a pump on in the tank benchmarks.
$\mathtt{P}^-$	The plant variable for the request to switch a pump off in the tank benchmarks.
$\mathcal{P}$	The SFC variable for a pump from the tank benchmarks.
$\mathcal{P}^+$	The SFC variable for the request to switch a pump on in the tank benchmarks.
$\mathcal{P}^-$	The SFC variable for the request to switch a pump off in the tank benchmarks.
q	An action qualifier $q \in \{\mathtt{entry}, \mathtt{do}, \mathtt{exit}\}$ of an SFC.
r	The reset function $r : \mathcal{V}^{V_{\mathcal{H}}} \to \mathcal{V}^{V_{\mathcal{H}}}$ of an edge in a hybrid automaton with variable set $V_{\mathcal{H}}$.

R	The room constant in the thermostat benchmark.
$\mathcal{R}$	A set of hybrid automaton states that are reachable within a given time.
$\mathbb{R}$	The set of real numbers. The set of positive real numbers excluding 0 is denoted by $\mathbb{R}_{>0}$ and the set of non-negative real numbers is $\mathbb{R}_{\geq 0}$.
s	A step of an SFC.
s_0	The initial step of an SFC.
sen	The finite set of variables for the sensors of the plant in the BMC approach.
sort	A function that sorts a given set of SFC actions according to a given action order $\sqsubset$.
source	A function source : $2^{Trans} \rightarrow 2^{\mathcal{S}}$ that assigns the set of corresponding source steps to a given set of transitions of an SFC.
S	A subset of steps in $\mathcal{S}$.
SFC_a	The set $SFC_a \subseteq \overline{\mathcal{C}}$ of active SFCs.
$\mathcal{S}$	The set of all steps of an SFC.
$\mathcal{S}_a$	The set $\mathcal{S}_a \subseteq \mathcal{S}$ of active steps of an SFC. $\mathcal{S}_a^0$ denotes the initial set of active steps.
$\mathcal{S}_r$	The set $\mathcal{S}_r \subseteq \mathcal{S}$ of ready steps of an SFC. $\mathcal{S}_r^0$ denotes the initial set of ready steps.
t	A variable that denotes a point in time.
timer	A clock that measures the time that is spent in the current location.
tr	A counter for the number of taken transitions for each path in a hybrid automaton.
tstamp	A variable that stores the last time at which a transition i has been taken for each path in a hybrid automaton.
target	A function target : $2^{Trans} \rightarrow 2^{\mathcal{S}}$ that assigns the set of corresponding target steps to a given set of transitions of an SFC.
T	The room temperature in the thermostat benchmark.
T_{max}	The upper bound for the allowed temperature in the thermostat benchmark.
T_{min}	The lower bound for the allowed temperature in the thermostat benchmark.
T_H	Constant for the temperature where the heater is switched off in the thermostat benchmark.
T_I	Constant for the initial room temperature in the thermostat benchmark.
T_L	Constant for the temperature where the heater is switched on in the thermostat benchmark.

Tr	A quantifier-free arithmetic formula over the variables $V_{\mathcal{C}} \cup V_{\mathcal{C}}'$ that defines the transition relation of a PLC program for bounded model checking.
Trans	$\mathit{Trans} \subseteq (2^{\mathcal{S}} \setminus \{\emptyset\}) \times \Phi(V_{\mathcal{C}}) \times (2^{\mathcal{S}} \setminus \{\emptyset\})$ is a finite set of guarded transitions of an SFC.
$\mathit{Trans}^{\overline{\mathcal{C}}}$	The union of the transitions in all SFCs in $\overline{\mathcal{C}}$.
$\mathcal{T}$	The set $\mathcal{T} \subseteq \mathit{Trans}$ of taken transitions of an SFC.
U	Variable that denotes a tank in the tank benchmarks.
Urg	A function $\mathit{Urg} : (\mathit{Loc} \cup \mathit{Edge}) \to \mathbb{B}$ that marks the urgent locations and transitions.
V	A finite set of variables.
V_{act}	The finite set of variables for the actuators of a plant.
V_c	A finite set of continuous variables.
V_{cont}	A finite set of continuous variables for the physical quantities of a plant.
V_d	A finite set of discrete variables.
V_{in}	The finite set of input variables of an SFC.
V_{loc}	The finite set of local variables of an SFC.
V_{plant}	A finite set $V_{plant} = V_{act} \cup V_{sen}$ of plant variables consists of a set of actuator variables and a set of sensor variables.
V_{sen}	The finite set of variables for the sensors of a plant.
V_{out}	The finite set of output variables of an SFC.
$V_{\mathcal{C}}$	The finite set of variables of the SFC $\mathcal{C}$ with $V_{\mathcal{C}} := V_{in} \cup V_{loc} \cup V_{out}$.
$V_{\mathcal{H}}$	The finite variable set of the hybrid automaton $\mathcal{H}$.
V_1	The finite set of variables of a naive hybrid automaton.
V_2	The finite set of variables of an advanced hybrid automaton.
`V`	The plant variable for a valve in the tank benchmarks.
$\mathcal{V}$	The set of valuations for a given variable set.
$\mathcal{V}^{V_{\mathcal{C}}}$	The set of valuations for the variable set $V_{\mathcal{C}}$. A valuation is a function $\nu : V_{\mathcal{C}} \to \mathbb{R} \in \mathcal{V}^{V_{\mathcal{C}}}$ assigns a value to each variable $x_i \in V_{\mathcal{C}}$.
$\mathcal{V}^{V_{\mathcal{H}}}$	The set of valuations for the variable set $V_{\mathcal{H}}$. A valuation is a function $\nu : V_{\mathcal{H}} \to \mathbb{R} \in \mathcal{V}^{V_{\mathcal{H}}}$ assigns a value to each variable $x_i \in V_{\mathcal{H}}$.
$\mathbb{Z}$	The set of integers.
α	A label $\alpha \in \mathit{Lab}$ of a hybrid automaton.
δ	The cycle time of a PLC cycle.
Δ	The constant for the maximal time elapse in a flowpipe computation.

ε	The constant for the minimal time elapse to avoid Zeno behavior in a hybrid automaton.
ζ	A convex set.
τ	Constant for the step size of a timestep during the flowpipe construction.
μ	A variable valuation $\mu : \bigcup_{i=0}^{k} V_{\mathcal{C}}^{i} \to \mathbb{R}$ for which the BMC formula φ_k^{BMC} evaluates to *true*; μ is also called a solution of φ_k^{BMC} or a discrete counterexample.
ν	A valuation ν of a variable set V is a function $\nu : V \to \mathbb{R}$ that assigns a value to each variable $x \in V$.
ρ	A configuration $\rho = (\nu, \mathcal{S}_r, \mathcal{S}_a, \mathcal{A}_a)$ of an SFC.
σ	A state $\sigma = (l, \nu) \in \mathit{Loc} \times \mathcal{V}$ is a location-valuation pair.
Σ	A state set $\Sigma \subseteq \mathit{Loc} \times \mathcal{V}$ contains pairs of a location and a valuation.
$\Sigma^{V_\mathcal{H}}$	The state set $\Sigma^{V_\mathcal{H}} \subseteq \mathit{Loc} \times \mathcal{V}^{V_\mathcal{H}}$ of a hybrid automaton with variable set $V_\mathcal{H}$.
φ	A real-arithmetic formula.
φ^{BMC}	The arithmetic formula $\varphi_k^{BMC} = I_0 \wedge \bigwedge_{i=0}^{k-1} \mathit{Tr}_{i,i+1} \wedge \neg \mathit{Prop}_k$ over the variables $\bigcup_{i=0}^{k} V_{\mathcal{C}}^{i}$ that states that the encoded paths of length k reach an unsafe state. This formula is used for a k-bounded safety analysis.
Φ	A set of predicates.
ψ	An explanation ψ for a spurious counterexample is an arithmetic formula that excludes the sequence of inputs and outputs produced by the unconstrained discrete analysis.
Ψ	An explanation set Ψ is a conjunction of explanations ψ for spurious counterexamples.
$\to_\mathcal{H}$	A state transformation $\to_\mathcal{H}$: $\mathit{Loc} \times \mathcal{V}^{V_\mathcal{H}} \to \mathit{Loc} \times \mathcal{V}^{V_\mathcal{H}}$ of a hybrid automaton with variable set $V_\mathcal{H}$. In a hybrid automaton $\mathcal{H}$, a jump using edge $e \in \mathit{Edge}$ is denoted by $\to_\mathcal{H}^e$ and a time step of length t by $\to_\mathcal{H}^t$.
$\sqsubset$	The action order of an SFC.
$\prec$	The transition order of an SFC.

Part II – Synthesis of Control Strategies for Hybrid Vehicles

a	The acceleration of a vehicle in m/s^2.
A	The frontal area of a vehicle in m^2 ($1.80m^2$ in our vehicle model).
c	A chromosome of a population.
C_d	The air drag resistance (0.25 in our vehicle model).
C_{GA}	A configuration $C_{GA} = (K, \gamma, \mu, G_{max})$ of a genetic algorithm.
e	A control strategy $e \in \mathcal{E}$.
e_f	The evaluation function for the fuel consumption used by our GA-based control strategy.
e_{sd}	The evaluation function for the SoC deviation from SoC_{ref} used by our GA-based control strategy.
e_{sl}	The evaluation function for the SoC level used by our GA-based control strategy.
e_{ud}	The evaluation function for the split differences used by our GA-based control strategy.
e_{ut}	The evaluation function for the split transgression of $\Delta_{u,max}$ used by our GA-based control strategy.
$E_e^+(t)$	The maximal amount of positive electric energy at the end of a driving cycle.
$E_e^-(t)$	The maximal amount of negative electric energy at the end of a driving cycle.
$\mathcal{E}$	The set of basic control strategies used in a learning-based control strategy.
f	The fitness value of a chromosome of a genetic algorithm.
f_r	The rolling resistance ($0.011 - 0.015$ for vehicle wheels on asphalt, 0.000 in our vehicle model).
g	The constant of gravity $g = 9.81m/s^2$.
G_{max}	The maximal number of iterations of a genetic algorithm.
h	The current gear $\in 0, \ldots, 5$ of a vehicle; 0 encodes a standing car where no gear is chosen.
H_l	The lower heating value of fuel ($42.70 - 44.20MJ/kg$ for gasoline, $42.60MJ/kg$ in our vehicle model).
I	The battery current in A.
J	The energy management evaluation function.

J^*	The optimal solution of the energy management evaluation function.
$J^*_{[t,t+T_p]}$	The optimal solution of the energy management evaluation function over the given prediction horizon.
$J^{c_k(t)}_{[t,t+T_p]}$	The solution of the energy management evaluation function for the chromosome k at times step t over the given prediction horizon.
J^u	The solution of the energy management evaluation function for a control function u.
K	The population size, i. e. the number of chromosomes.
m	The mass of a vehicle in kg ($1440.00kg$ in our vehicle model).
m_r	The mass of the rotating parts of the vehicle in kg ($0.05kg$ in our vehicle model).
$\dot{m}_f$	The instantaneous fuel consumption in g/s, given by a function depending on ω_{ice} and T_{ice}. $[\dot{m}_{f,min}, \dot{m}_{f,max}]$ denote the lower and the upper bound for the instantaneous fuel consumption within a given time horizon.
$\dot{m}_{f,equ}$	The equivalent cost function.
$\dot{m}^u_{f,equ}$	The equivalent cost function for the control u.
M	$M = \|\mathcal{E}\|$ is the number of basic control strategies used in a learning-based control strategy.
N	The number of crossover points used by a genetic algorithm.
$\mathcal{N}_{\nu,\sigma}$	The normal distribution function with mean ν and standard deviation σ.
P_{em}	The input power P_{em} in kW of the EM is given by a function depending on ω_{em} and T_{em}; the values range from $P_{em,min}$ to $P_{em,max}$($[0.00, 280.17]$ in our vehicle model).
q	The probability that a control strategy from $\mathcal{E}$ is chosen by the learning algorithm.
Q	The probability distribution of a learning algorithm for the set of control strategies $\mathcal{E}$.
$Q_{batt,max}$	The maximal cell capacity of the battery in Ah ($6Ah$ in our vehicle model).
r	The gear ratio $\in \{15.17, 8.34, 5.38, 3.94, 2.75\}$ of the current gear $\in \{1, 2, 3, 4, 5\}$.
r_{wh}	The radius of the wheels ($0.22m$ in our vehicle model).

s	The equivalence factor of ECMS.
s_0	The initial value of the ECMS equivalence factor.
s_{chg}	The equivalence factor that corresponds to a negative electric energy use.
s_{dis}	The equivalence factor that corresponds to a positive electric energy use.
SoC	The battery state of charge $\in [SoC_{min}, SoC_{max}]$, with initial value SoC_0 and reference value SoC_{ref}. In our vehicle model, we have $SoC_{min} = 0.1, SoC_{max} = 0.99, SoC_0 = 0.6$, and $SoC_{ref} = 0.6$..
SoC_{diff}	The maximal allowed difference between the SoC values of consecutive steps in the simulation.
SoC^{loc}_{max}	The maximal value of the SoC at the end of the prediction horizon.
SoC^{loc}_{min}	The minimal value of the SoC at the end of the prediction horizon.
t	The current time step in a simulation.
$trans_{max}$	The maximal possible transgression of $\Delta_{u,max}$ for the difference of consecutive torque splits.
T	The duration of the driving cycle.
T_{br}	The torque at the brakes in Nm.
T_{em}	The torque $T_{em} \in [T_{em,min}, T_{em,max}]$ produced by the EM in Nm ($[-305.00, 305.00]$ in our vehicle model).
T^u_{em}	The torque produced by the EM that has been determined using the control function $u(t)$.
T_{ice}	The torque $T_{ice} \in [T_{ice,min}, T_{ice,max}]$ produced by the ICE in Nm ($[8.55, 101.98]$ in our vehicle model).
T^u_{ice}	The torque produced by the ICE that has been determined using the control function $u(t)$.
T_p	The length of the prediction horizon.
T_{req}	The torque $T_{req} = T_{ice} + T_{em} = \frac{T_{wh}+T_{br}}{\eta_{gb} r_h}$ requested by the driver in Nm.
T_{wh}	The torque $T_{wh} = \eta_{gb} r_h T_{req} - T_{br}$ at the wheels in Nm.
u	The torque split $u \in [u_{min}, u_{max}]$.
U_{oc}	The open circuit voltage of the battery in V ($200V$ in our vehicle).
v	The velocity of the vehicle in m/s.
w	The weight of a control strategy from the set $\mathcal{E}$.
w_Q	The weight for the fuel consumption in the cost function of DP.

w_R	The weight for the SoC deviation from SoC_{ref} in the const function of DP.
w_f	The weight for the evaluation of the fuel consumption used by our GA-based control strategy.
w_{sd}	The weight for the evaluation of the SoC deviation from SoC_{ref} used by our GA-based control strategy.
w_{sl}	The weight for the evaluation of the SoC level used by our GA-based control strategy.
w_{ud}	The weight for the evaluation of the split differences used by our GA-based control strategy.
w_{ut}	The weight for the evaluation of the split transgression of $\Delta_{u,max}$ used by our GA-based control strategy.
γ	The crossover rate of a genetic algorithm.
$\Delta_{u,max}$	The maximal difference between consecutive torque splits.
ε	The factor for the SoC convergence term of a learning algorithm.
η	The learning rate of a learning algorithm.
η_{gb}	The transmission efficiency of the gearbox (0.98 in our vehicle model).
θ	The road slope.
κ_0	The tuning parameter for the cost-to-go term in ADP.
κ_i	The integral feedback gain of a PI-controller.
κ_p	The proportional feedback gain of a PI-controller.
λ	The factor for the cost-to-go term in ADP.
λ_0	The tuning parameter for the cost-to-go term in ADP.
μ	The mutation rate of a genetic algorithms.
ν	The mean value used by the normal distribution function.
ρ	The density of air is given by $\rho = 1.18kg/m^3$.
σ	The standard deviation used by the normal distribution function.
ω_{em}	The angular velocity in rad/s at the EM $\in [\omega_{em,min}, \omega_{em,max}]$.
ω_{ice}	The angular velocity in rad/s at the ICE $\in [\omega_{ice,min}, \omega_{ice,max}]$.
ω_{req}	The angular velocity in rad/s $\omega_{req} = \omega_{ice} = \omega_{em}$ requested by the driver.
ω_{wh}	The angular velocity in rad/s $\omega_{wh} = \omega_{req}/r_h$ at the wheels.

Glossary

A-ECMS	Adaptive ECMS
ADP	RDP with cost-to-go approximation
AlgoSyn	DFG Research Training Group “Algorithmic Synthesis of Reactive and Discrete-Continuous Systems”
BFS	Breadth-first search
BMC	Bounded model checking
BR	Brakes
CEGAR	Counterexample-guided abstraction refinement
CIS	Candidate initial state
DFG	Deutsche Forschungsgemeinschaft (engl. German Research Foundation)
DFS	Depth-first search
DP	Dynamic programming
ECMS	Equivalent consumption minimization strategy
EM	Electrical motor
EPA	United States Environmental Protection Agency
FIFO	First-in, first-out
FTP	Federal Test Procedure
FTP-75	City program of the EPA FTP
GA	Genetic algorithm
GeneiAL	Genetic Algorithm Library
HA	Hybrid automaton
HEV	Hybrid electric vehicle
HJB	Hamilton-Jacobi-Bellman
HWFET	Highway program of the EPA FTP

ICE	Internal combustion engine
IEC	International Electrotechnical Commission
LBCS-SD	Learning-based control strategy that uses SD
LBCS-WF	Learning-based control strategy that uses WF
MIT	Massachusetts Institute of Technology
NEDC	New European Driving Cycle
OASys	DFG research project “Online Algorithms for Optimal Control of Hybrid Propulsion Systems”
ODE	Ordinary differential equation
PHEV	Parallel hybrid electric vehicle
PLC	Programmable logic controller
PMP	Pontryagin’s Minimum Principle
RDP	Receding dynamic programming
SAT	Propositional satisfiability
SD	Shrinking dartboard online learning algorithm
SFC	Sequential function chart
SMT	Satisfiability-modulo-theories
SoC	Battery state of charge
T-ECMS	Telemetry ECMS
WF	Weighted fractional online learning algorithm