Rolf Drechsler
Ulrich Kühne *Editors*

Formal Modeling and Verification of Cyber-Physical Systems

1st International Summer School on Methods and Tools for the Design of Digital Systems, Bremen, Germany, September 2015

Springer Vieweg

Formal Modeling and Verification of Cyber-Physical Systems

Rolf Drechsler • Ulrich Kühne (Eds.)

Formal Modeling and Verification of Cyber-Physical Systems

1st International Summer School on Methods and Tools for the Design of Digital Systems, Bremen, Germany, September 2015

Springer Vieweg

Editors
Rolf Drechsler
University of Bremen / DFKI
Bremen, Germany

Ulrich Kühne
University of Bremen
Bremen, Germany

ISBN 978-3-658-09993-0 ISBN 978-3-658-09994-7 (eBook)
DOI 10.1007/978-3-658-09994-7

Library of Congress Control Number: 2015940758

Springer Vieweg

Printed on acid-free paper

Springer Vieweg is a brand of Springer Fachmedien Wiesbaden
Springer Fachmedien Wiesbaden is part of Springer Science+Business Media
(www.springer.com)

Preface

Today, embedded systems are ubiquitous in our everyday life, in cell phones and washing machines, but also in cars and even medical equipment. With this escape from their former habitat within desktop computers, these systems are increasingly interacting with their environment: Sensors measure physical phenomena such as temperature, acceleration, or magnetic fields, while actors manipulate the outside world, like in robots or electronically controlled combustion engines. The combination of an electronic system with a physical process is called a *cyber-physical system* (CPS).

With this paradigm, many new challenges need to be faced during modeling, design, implementation, verification, and test. For the design of hardware and software of CPS, new approaches need to be developed, taking into account non-functional requirements like energy efficiency and reliability even in harsh environments. Real-time aspects often play an important role. Furthermore, if a system is interacting with its physical environment, it becomes difficult to prove the functional correctness of the system. The combination of discrete and continuous behavior and the treatment of noisy sensor data are challenging problems.

Considering all the above aspects, CPS is an interdisciplicary topic, touching research areas such as computer science, robotics, electrical and mechanical enginieering, physics and more. In the *First International Summer School on Methods and Tools for the Design of Digital Systems*, we want to bring together experts from different fields of research and application. The goal of this summer school is to introduce PhD students to this exciting topic, and to offer them deeper insights into formal modeling and verification techniques. Regarding the application areas, the courses focus on robotics and space systems, as well as the railway domain and microfluidic biochips.

This summer school is organized by the *Graduate School System Design (SyDe)*. SyDe has been founded in 2012, funded by the German Excellence Initiative. It is a cooperation of the University of Bremen, the German Research Center for Artificial Intelligence (DFKI), and the German Aerospace Center (DLR). The graduate school provides a structured program for the formation of young researchers, bringing together PhD students from different working groups and research institutes that cover a wide range of scientific topics in hardware and system design, robotics and space systems. The summer school of 2015 is the third one organized by SyDe, and the first one open to international PhD students.

Overview. This book is divided into two parts. The first part contains the lecture notes, while the second part features selected short articles written by participating PhD students. The participants had been invited to submit a short report on their thesis projects. All submissions have undergone a review process.

The first three chapters discuss real-time and hybrid aspects, starting with *Verification of Embedded Real-time Systems*. There, Paula Herber and Sabine Glesner present a formalism for modeling real-time systems. In particular, they give semantics to SystemC models in terms of Timed Automata, and show how safety and timing properties can be verified.

In the second chapter, Frédéric Mallet introduces *MARTE/CCSL for Modeling Cyber-Physical Systems*. MARTE is a real-time extension of the UML. While the UML already provides a great variety of models and diagrams, important aspects of CPS are not captured by the standard. This chapter discusses how CPS can be modeled using appropriate extensions.

Goran Frehse gives *An Introduction to Hybrid Automata, Numerical Simulation and Reachability Analysis*. Hybrid automata are a popular formalism for the modeling of CPS, since they combine discrete and continuous aspects. Depending on the continuous dynamics of a modeled system, reachability computation can become very costly, and abstraction techniques are crucial in order to create scalable verification tools.

The following three chapters treat different aspects of verification and test. Here, it is discussed how classical methods from hardware and software verification can be applied on CPS, starting with *Model Checking and Model-Based Testing in the Railway Domain*. In this chapter, Anne Haxthausen and Jan Peleska show the integration of verification and test in an industrial context.

Modeling Unknown Values in Test and Verification is discussed in the following chapter by Bernd Becker et al. Unknown values occur frequently at the interfaces of components when following a component-based design style, or when systems are exposed to an uncertain environment, as is often the case for embedded systems and CPS. This chapter how formal techniques can be enhanced to treat such uncertainties without sacrificing exactness.

Complementary to static verification techniques, run-time monitoring is an important technique to ensure the proper functionality of complex systems. In the chapter *Specification of Parametric Monitors – Quantified Event Automata versus Rule Systems*, Klaus Havelund and Giles Reger give an overview on two parametric runtime verification systems. The formalisms are presented by means of an extensive suite of application examples, including the monitoring of a planetary rover.

The lecture notes are concluded with three chapters on practical applications of CPS, in the domains of biological micro-laboratories and robotics. Krishnendu Chakrabarty et al. present their recent *Advances in Design Automation Techniques for Digital-Microfluidic Biochips*. These systems combine electronics with biology, enabling the fast and cheap treatment of biological samples on a chip. This chapter presents the design of biochips, and how domain specific constraints can be treated.

The interaction between humans and robots is discussed in the chapter *Intuitive Interaction with Robots – Technical Approaches and Challenges* by Elsa Kirchner et al. Here, different research areas in the field of human-robot interaction are presented.

All chapters of the lecture notes contain extensive references to related work for further reading.

Acknowledgements. We would like to thank all speakers, authors and co-authors for their great contributions for the summer school and for this book. Many thanks go to the reviewers and to all participants of the summer school. We also thank the University of Bremen and our partner institutes DFKI and DLR for making this event possible. We are grateful for the financial support from our sponsoring partners *Verified Systems International* and *Concept Engineering*.

We hope you will enjoy reading this book.

Bremen, April 2015

Rolf Drechsler
Spokesman of SyDe

Ulrich Kühne
Scientific Coordinator of SyDe

Organization

The SyDe Summer School 2015 is organized by the Graduate School System Design, a joint project of the University of Bremen, the German Research Center for Artificial Intelligence (DFKI), and the German Aerospace Center (DLR). SyDe is funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

Executive Commitee

Ingrid Bode	University of Bremen
Rolf Drechsler	University of Bremen / DFKI GmbH, Germany *(Spokesman of SyDe)*
Ulrich Kühne	University of Bremen

Invited Speakers

Bernd Becker	Albert-Ludwigs University, Freiburg, Germany
Krishnendu Chakrabarty	Duke University, North Carolina, USA
Goran Frehse	Verimag, Grenoble, France
Sabine Glesner	Technical University of Berlin, Germany
Sami Haddadin	University of Hannover, Germany
Klaus Havelund	NASA Jet Propulsion Lab, Pasadena, USA
Anne Haxthausen	Technical University of Denmark
Elsa Kirchner	DFKI, Bremen, Germany
Frédéric Mallet	Sophia Antipolis, Nice, France
Jens Dalsgaard Nielsen	Aalborg University, Denmark

Reviewers

G. Aydos	D. Große	D. Schüthe
M. Diependbeck	C. Hilken	J. Seiter
S. Eggersglüß	O. Keszösce	M. Soeken
G. Fey	U. Kühne	N. Thole
S. Frehse	H. Le	R. Wille
U. Frese	J. Peters	
M. Goldhoorn	E. Schönborn	

Sponsoring Institutions

Verified Systems International GmbH, Bremen, Germany
Concept Engineering GmbH, Freiburg, Germany

Table of Contents

Lecture Notes

Verification of Embedded Real-time Systems 1
P. Herber, S. Glesner

MARTE/CCSL for Modeling Cyber-Physical Systems 26
F. Mallet

An Introduction to Hybrid Automata, Numerical Simulation and Reachability Analysis 50
G. Frehse

Model Checking and Model-Based Testing in the Railway Domain 82
A.E. Haxthausen, J. Peleska

Modeling Unknown Values in Test and Verification 122
B. Becker, M. Sauer, C. Scholl, R. Wimmer

Specification of Parametric Monitors – Quantified Event Automata versus Rule Systems 151
K. Havelund, G. Reger

Advances in Design Automation Techniques for Digital-Microfluidic Biochips 190
M. Ibrahim, Z. Li, K. Chakrabarty

Intuitive Interaction with Robots – Technical Approaches and Challenges 224
E.A. Kirchner, J. de Gea Fernandez, P. Kampmann, M. Schröer, J.H. Metzen, F. Kirchner

Physical Safety in Robotics 249
S. Haddadin

Student Short Articles

In-circuit Error Detection with Software-based Error Correction – An Alternative to TMR 272
G. Aydos

Behavior Driven Development for Tests and Verification 275
M. Diepenbeck

Semantic Object Recognition Based on Qualitative Probabilistic Spatial Relations 278
M. Goldhoorn

Constraint-based Handling of Component Networks 281
M. Goldhoorn

Model-Based Testing Against Complex SysML Models 284
C. Hilken

Integrated Model-based Testing and Model Checking with the Benefits of Equivalence Partition Testing 287
F. Hübner

An SMT-based Approach to analyze Non-Linear Relations of Parameters for Hybrid Systems 290
X. Li

Analyzing and Simulating Time Descriptions from UML/MARTE CCSL . 293
J. Peters

Design and Synthesis of Reversible Circuits using Hardware Description Languages 296
E. Schönborn

Dynamic Rebound Control and Human Robot Interaction of a Ball Playing Robot 299
D. Schüthe

Development of Consistent Formal Models 302
J. Seiter

Formal Verification of Robustness 305
N. Thole

Pose and Posture Estimation using Inertial Sensor Data 308
F. Wenk

Reconfigurable Hardware-Based Acceleration for Machine Learning and Signal Processing 311
H. Woehrle

Author Index 315

Verification of Embedded Real-time Systems

Paula Herber[1] and Sabine Glesner[2]

[1] University of Potsdam
August-Bebel-Str. 89, 14482 Potsdam, Germany
paula.herber@uni-potsdam.de
[2] Technische Universität Berlin
Ernst-Reuter-Platz 7, 10587 Berlin, Germany
sabine.glesner@tu-berlin.de

Abstract. Real-time systems are systems where the correctness does not only depend on their correct functioning but also on meeting real-time constraints. Such systems are often deployed in safety-critical applications, for example in airplanes, trains, or automotive systems. There, a failure may result in enormous costs or even in human injuries or loss of lifes. As a consequence, systematic verification and validation of real-time systems is a crucial issue.
The main application area for real-time systems are embedded applications, where the system controls technical processes that also evolve in real-time. Such systems are usually composed of deeply integrated hardware and software components, and they are developed under severe resource limitations and high quality requirements. In connection with the steadily increasing demands on multi-functioning and flexibility, analog control components are more and more replaced by complex digital HW/SW systems.
A major challenge is to develop automated quality assurance techniques that can be used for the verification and validation of complex embedded real-time systems that consist of both hardware and software. In this chapter, we give an overview over our research contributions to this topic. In particular, we present our framework for the verification of safety and timing properties of digital embedded real-time systems, which are modeled in SystemC, using timed automata and the Uppaal model checker.

1 Introduction

Embedded systems are often subject to real-time constraints. As an example, consider the control system for the airbag of a car. If the airbag inflates too late in a crash situation, the life of the occupant is endangered. Thus, it must be ensured that an airbag always inflates within a given deadline, e.g. 30 milliseconds. This is an example of a so-called hard real-time constraint, where missing the deadline is regarded as a total system failure. Embedded systems may also be subject to firm real-time constraints, where results are not usable after the deadline, or soft real-time constraints, where the usefulness of a result degrades after the deadline. Embedded systems are inherently real-time dependent because they interact with a technical process and the process evolves in real-time.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_1,

As demonstrated with the airbag example, to ensure that an embedded system meets real-time constraints is often a vital task in the system design process. This is of particular importance in safety-critical systems, where a failure results in high cost or may even endanger human lifes. For such systems, it is not sufficient to validate the system behavior by simulating and testing for some input scenarios. Only methods that consider all possible input scenarios and all possible system executions on a precise system description can provide guarantees about the system behavior. The prerequisite for such methods is an unambigious, formal description of the system under verification and the desired (real-time) requirements. The process of verifying that a formal model fulfills a given set of requirements for all possible input scenarios is called formal verification.

To formally verify real-time dependent behavior, various formalisms have been proposed. Typically, real-time is integrated into existing formalisms by introducing timers or clocks and by assigning lower and upper timing bounds to transitions. Examples for this are time petri nets [6], process algebras like timed communicating sequential processes (Timed CSP) [51], or timed automata [1]. For all of these formalisms, there exists mature tool support to verify whether a system meets given real-time requirements, e.g., the TIme petri Net Analyzer (TINA) [7] for time petri nets, the refinement checker FDR3 [16] for Timed CSP, or the model checker Uppaal for timed automata [4].

Still, one of the major challenges for the application of real-time verification in system level design processes is that it can only be applied to formal models, but formal models are often not available in practical development processes. Instead, typical design flows for embedded systems make use of *system level design languages* such as SystemC [31] to model the system behavior on abstract and intermediate levels of abstraction. SystemC supports design space exploration and performance evaluation efficiently throughout the whole design process even for large and complex HW/SW systems. It supports an integer-valued time model with arbitrary time resolution and thus can be used for modeling and simulation of embedded real-time systems. SystemC allows the description of both hardware and software, and the designs are executable on different levels of abstraction. As a consequence, co-simulation, i. e., the simultaneous execution of hardware and software, can be used to validate a system throughout the whole design process. For quality assurance of safety-critical systems, however, simulation is necessary but not sufficient. This has three reasons: First, simulation is incomplete. It can neither be applied to all possible input scenarios, nor can it be assured that all possible executions are covered in the case of non-deterministic systems. Second, although HW/SW co-designs are developed in a refinement process where an abstract design is incrementally refined to the final implementation, it is very difficult to ensure consistency between different abstraction levels. Third, simulation alone is not sufficient for a systematic and comprehensive quality assurance approach because the degree of automation is limited. In particular, the evaluation of simulation results is typically manually performed by the designer.

In our research of the past years, we have developed a framework for formally founded and comprehensive quality assurance of embedded real-time systems

that are modeled in SystemC. Our framework for the **Veri**fication of **S**ystemC designs using **T**imed **A**utomata [26] (VeriSTA) allows the verification and validation of digital HW/SW co-designs throughout the whole design process. It provides verification and validation techniques that explicitly consider real-time dependent behavior and thus, it enables us to ensure that a given system meets real-time requirements. The whole framework is automatically applicable.

In the VeriSTA framework, we assume that the HW/SW co-design process starts with an abstract design that is incrementally refined down to the final implementation. To enable model checking of safety, liveness, and real-time requirements of abstract designs, we have developed an automated transformation from SystemC into the formal language of Uppaal timed automata [24, 28, 47, 48]. To enable automated real-time testing in later development stages, we have (1) proposed an evolutionary algorithm for the automatic generation of timed input traces that achieve a high transition coverage [23] and (2) we have presented an approach for the automated generation of conformance tests based on the *relativized timed input/output conformance* (rtioco) relation proposed by [39], which can be used to validate that refined models conform to the (verified) abstract model [25, 27].

The rest of this chapter is organized as follows: In the next section, we discuss related work. Then, in Sec. 3, we give some preliminaries that are necessary to understand the remainder of this chapter. In Sec. 4, we present our VeriSTA framework and in Sec. 5 our formal semantics for SystemC. In Sec. 6, we present our approach for model checking and conformance testing, and in Sec. 7, we describe our evolutionary algorithm for the generation of timed test traces with a high transition coverage. In Sec. 8, we present experimental results and we conclude in Sec. 9.

2 Related Work

In this section, we give an overview over related work from the following areas: formal verification of SystemC designs, real-time conformance testing, and automated test generation for SystemC.

2.1 Formal Verification of SystemC Designs

There exist several approaches to provide a formal semantics for SystemC. Some of them only cope with a synchronous subset of SystemC, which means that they do not support asynchronous communication between processes, dynamic sensitivity, or thread processes. For example, in [18, 19], Große et al. present an approach for formal verification of a synthesizable subset of SystemC. They use Binary Decision Diagrams (BDDs) and bounded model checking but their approach is restricted to static sensitivity and does not cope with timing.

Several other approaches rely on the transformation of SystemC designs into some sort of state machine, as done in [22, 21, 55]. Habibi and Tahar [22, 21] transform SystemC models into equivalent state machines. However, they only

support untimed models and they do not maintain the structure of the SystemC design. Traulsen et al. [55] map (possibly timed) SystemC designs to PROMELA, but only handle SystemC designs on an abstract level, do not model the SystemC scheduler, and do not support primitive channels. In [38, 8], Kroening et al. propose a semantics for SystemC that is based on a labeled Kripke structure and automatically partition the design into a hardware and a software part to increase efficiency of verification. However, they abstract from hardware and do not consider timing or inter-process communication via sockets and channels.

Other approaches use process algebras, petri nets or a C representation for the verification of SystemC designs. The formal language SystemC$^{\mathbb{FL}}$ [42] presented by Man et al. is based on process algebras. It considers only simple communications, and no dynamic sensitivity or channels. Garavel et al. [15] translate SystemC/TLM into the process algebra LOTOS and import C Code into the LOTOS model using the verification toolbox CADP. However, the transformation has to be done manually and they only support untimed SystemC designs. Karlsson et al. [35] use a petri net based representation to verify SystemC designs. Due to the fact that modeling interactions between subnets introduces additional subnets, this approach produces a huge overhead. Behjati and Razavi [3, 49] transform a SystemC design into an equivalent REBECA (reactive object language) model and use the REBECA model checker Modere [32] for verification. The transformation is done automatically, but it only supports untimed models.

One of the few approaches for the verification of SystemC designs that also supports the TLM 2.0 standard is the work of Cimatti et al. [11–13]. They generate three different verification models from a given SystemC design and use software model checking techniques. The computational behavior is verified by mapping SystemC processes to sequential C and using an abstract scheduler. For the process interaction and communication, a model containing an explicit scheduler and symbolic processes is used. The third model contains both an explicit scheduler and explicitly modeled threads and the design is mapped to a finite state machine representation. While this approach is capable of handling the most important SystemC and TLM constructs, it does neither support complex data types like structs or arrays nor can it handle any memory-related constructs or operations. Furthermore, by using state-of-the-art software model checkers, they do not make use of any symbolic representation of time, which impedes the verification of real-time systems modeled in SystemC.

In [40], Le et al. present and approach to transform SystemC/TLM designs into the low level intermediate verification language IVL. They use the metaSMT framework for formal verification. They support assertion-based as well as property-based verification. The approach can handle the TLM 2.0 standard, integer based datatypes, arrays and pointers to primitive data types, but no complex data types like structs. Moreover, dynamic memory management is not supported. Again, no symbolic representation of time is used, which makes this approach inappropriate for the verification of real-time systems that are modeled in SystemC.

2.2 Conformance Testing for Real-time Systems

There exist several approaches to generate conformance tests for real-time systems from timed automata models and in particular to generate such tests from Uppaal. However, most of them consider only a restricted subclass of the timed automata model, or they do not allow static (*offline*) test generation. In [45, 37], techniques for the automatic generation of real-time black-box conformance tests for non-deterministic systems are presented, but only for a determinizable subclass of timed automata. The CoVer tool [30, 29] allows coverage-driven conformance test generation for Uppaal models. The search for a test case covering a specific transition is formulated as a reachability problem and the Uppaal model checker is used to generate a witness for that. The witness can then be used as a test case. However, the coverage-driven generation of test cases using a model checker is very expensive in terms of computation effort and in terms of memory consumption. For our case study, the Uppaal model checker always ran out of memory when we tried to compute the reachability of transitions in our larger case studies. As a consequence, the use of the Uppaal model checker for the generation of tests with a certain transition coverage is not applicable to samples of industrial size. Furthermore, the approach is restricted to deterministic timed automata models. In contrast to that, the TRON (*Testing Real-time systems ONline*) tool [39, 29] can also be applied to non-deterministic specifications. However, it uses an *online* test generation algorithm and thus cannot be used to generate repeatable test cases and it may be difficult to apply it to real-time systems if the testing system doesn't react in a timely fashion. For conformance testing of embedded real-time systems, it is vital to generate conformance tests *offline* and to cope with non-deterministic specifications.

2.3 Automatic Test Generation for SystemC

There also already exist several approaches to systematically generate test cases for SystemC designs. For example, in [52], a tool that generates test bench templates for functional verification of SystemC designs is presented. There, the SystemC Verification Library [53] is used to produce constraint-based random-tests. Standard code coverage criteria are used to determine when the test generation is finished. Nevertheless, all coverage metrics must be implemented manually and are only used as exit conditions and not for directed test case generation. In [20], Groe et al. present an approach to identify untested parts of a given SystemC design by using code coverage techniques. In [33], an automatic test vector generation based on code coverage analysis is presented. For that, they use standard statement, branch and path code coverage criteria in combination with a coverage flow graph construction. However, the code has to be instrumented manually and the subset of admitted designs is very restricted. Another work that deals with validation of SystemC design was presented in [43] and in [46]. The authors present an approach for the generation of directed tests using state-space exploration. The designer has to specify actions and accept states in an abstract state machine model of the given SystemC design to be capable

to set the end of a test case. Afterwards, a directed test generation is undertaken by the SpecExplorer [56], an explicit-state model checker. However, it still requires a manual specification of both the design and a state that should be reached by a test case. No coverage criteria are used. In [9], an approach to select input sequences using static analysis on a given SystemC design is presented. However, the authors do not provide any means for describing the reachable states of interest. In [10], an approach for the generation of directed tests on *register transfer level* (RTL) from *transaction level modeling* (TLM) specifications is presented. The main disadvantage of the presented approach is that it is restricted to a small subset of SystemC in pure TLM. In [36], an approach for automatic test generation for SystemC design based on manually specified use cases is proposed. The authors make no use of existing abstract SystemC designs, and they do not consider coverage criteria.

3 Preliminaries

In this section, we present the preliminaries that are necessary to understand the remainder of this chapter by introducing SystemC and Uppaal timed automata.

3.1 SystemC

SystemC [31] is a system level design language and a framework for HW/SW cosimulation. It allows modeling and execution of both hardware and software on various levels of abstraction. The design flow usually starts with approximately timed transaction-level models that are refined to time-accurate models of hardware and software components. SystemC is implemented as a C++ class library, which provides the language elements and an event-driven simulation kernel. A SystemC design is a set of communicating processes, triggered by events and interacting through channels. Modules and ports are used to represent structural information. SystemC also introduces an integer-valued time model with arbitrary time resolution. The execution of a SystemC design is controlled by the SystemC scheduler. It controls the simulation time and the execution of processes, handles event notifications and updates. Like typical hardware description languages, SystemC supports the notion of delta-cycles, which impose a partial order on parallel processes. This means that the execution is split into an *evaluate* and an *update* phase. In the first phase, concurrent processes are evaluated, i. e., their method body is executed. This may include read and write accesses to so-called *primitive channels*, which store changes in temporary variables and do not update their channel state until the update phase. This ensures that, although the processes are serialized, they all work on the same channel states (i. e., input data). A delta-cycle lasts an infinitesimal amount of time and an arbitrary, finite number of delta-cycles may be executed at one point in simulation time. Note that the order in which processes are executed within a delta-cycle is not specified in [31], i. e., it is inherently *non-deterministic*. The same holds for the order of the updates of primitive channels. The simulation semantics of a SystemC design can be summarized as follows:

1. Initialization: each process is executed once,
2. Evaluation: all processes ready to run are executed in arbitrary order,
3. Update: primitive channels are updated,
4. if there are delta-delay notifications, the corresponding processes are triggered and steps 2 and 3 are repeated,
5. if there are timed notifications, simulation time is advanced to the earliest pending timed notification and steps 2 – 4 are repeated,
6. if there are no timed notifications remaining, simulation is finished.

For a more comprehensive description of the SystemC simulation semantics, we refer to [17, 44, 50]. Overall, SystemC allows for the integrated development of digital hardware and software components, and it supports synchronous and asynchronous parts of a design. It is established as a premier choice for the evaluation of design alternatives and high level simulation of integrated HW/SW systems. Furthermore, a subset of SystemC can be automatically synthesized to hardware.

3.2 Uppaal Timed Automata

Timed automata [1] are finite-state automata extended with real-valued clock variables. The clocks are initialized with zero and then run synchronously with the same speed. Clock constraints are used to model time-dependent behavior. A timed automaton consists of a set of locations connected by directed edges. *Invariants* are assigned to locations and yield conditions, under which one may stay in the corresponding state. The *invariants* must not be violated, i. e., the location must be left before its invariant is invalidated. *Guards* are assigned to edges and yield conditions under which the automaton may change from one location to another. Edges can also be labeled with actions, for example a clock reset. *Networks of timed automata* are used to model concurrent processes, which are executed with an interleaving semantics and synchronize on channels. Formally, the semantics of timed automata and networks of timed automata are given by [5] as follows:

Definition 1 (Operational Semantics of a Timed Automaton). *A timed automaton (TA) is a tuple* (L, l_0, C, A, E, I)*, where*

- L *is a set of locations,*
- $l_0 \in L$ *is the initial location,*
- C *is a set of clock variables,*
- A *is a set of actions,*
- $E \subseteq L \times A \times B(C) \times 2^C \times L$ *is a set of edges, where* $B(C)$ *denotes a set of clock constraints*
- $I : L \to B(C)$ *assigns invariants to locations.*

We write $l \overset{a,g,r}{\to} l'$ *for* $(l, a, g, r, l') \in E$*. The semantics of a TA is defined as a transition system* $(S, s_0, \to)$*, where* $S \subseteq L \times \mathbb{R}_{\geq 0}^{|C|}$ *is a set of states,* $s_0 = (l_0, u_0)$

the initial state, and $\rightarrow \subseteq S \times (\mathbb{R}_{\geq 0} \cup A) \times S$ *the transition relation. A clock valuation is a function* $u : C \rightarrow \mathbb{R}_{\geq 0}$ *that maps a non-negative real value to each clock,* $u \in I$ *denotes that a clock valuation satisfies an invariant, and* $u' = [r \mapsto 0]u$ *denotes a clock valuation where all clocks from the clock set* r *are reset to zero. A semantic step of a timed automaton can either be a time step (1) or a discrete transition (2) along an edge in the graphical representation:*

(1) $(l, u) \xrightarrow{d} (l, u + d)$ *iff* $\forall d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(l)$
(2) $(l, u) \xrightarrow{a} (l', u')$ *iff* $l \overset{a,g,r}{\rightarrow} l'$ *such that* $u \in g \wedge u' = [r \mapsto 0]u \wedge u' \in I(l')$

Definition 2 (Semantics of a Network of Timed Automata). *A network of timed automata (NTA) consists of* n *timed automata* $\mathcal{A}_i = (L_i, l_{0,i}, C, A, E_i, I_i)$. *The semantics of NTA is defined by a transition system* $(S, s_0, \rightarrow)$. *Each state* $s \in S$ *is a tuple* $(\bar{l}, u)$, *where* $\bar{l}$ *is a location vector and* u *is a clock valuation.* $S = (L_1 \times \ldots \times L_n) \times \mathbb{R}_{\geq 0}^{|C|}$ *denotes the set of states,* $s_0 = (\bar{l_0}, u_0)$ *the initial state, and* $\rightarrow \subseteq S \times S$ *the transition relation. Furthermore,* τ *denotes an internal action,* $c!, c?$ *sending resp. receiving an event on channel* c, *and* g *denotes a clock guard.* $I(\bar{l})$ *denotes the conjunction of all invariants* $I_i(l_i)$. *A semantic step can be either a time step (1), an independent step of a single automaton (2), or a synchronization between two automata (3):*

(1) $(\bar{l}, u) \rightarrow (\bar{l}, u + d)$ *iff* $\forall d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(\bar{l})$
(2) $(\bar{l}, u) \rightarrow (\bar{l}[l_i'/l_i], u')$ *iff* $l_i \overset{\tau g r}{\rightarrow} l_i'$ *such that* $u \in g \wedge u' = [r \mapsto 0]u \wedge u' \in I(\bar{l}[l_i'/l_i])$
(3) $(\bar{l}, u) \rightarrow (\bar{l}[l_j'/l_j, l_i'/l_i], u')$ *iff* $l_i \overset{c?g_i,r_i}{\longrightarrow} l_i' \wedge l_j \overset{c!g_j,r_j}{\longrightarrow} l_j'$ *such that* $u \in (g_i \wedge g_j) \wedge u' = [r_i \cup r_j \mapsto 0]u \wedge u' \in I(\bar{l'})$

Uppaal [5, 4] is a tool suite for modeling, simulation, animation and verification of *networks of timed automata*. The Uppaal modeling language extends timed automata by bounded integer variables, binary and broadcast channels, and urgent and committed locations. A small example Uppaal timed automaton (UTA) is shown in Fig. 1. The initial location is denoted by ◎. The label `request?` denotes receiving on the channel `request` while `ack!` denotes sending on channel `ack`. The clock variable `x` is first set to zero and then used in two clock conditions: the *invariant* `x <= maxtime` denotes that the corresponding location must be left before `x` becomes greater than maxtime, and the *guard* `x >= mintime` enables the corresponding edge if `x` is greater or equal `mintime`. The symbol Ⓤ depicts an urgent location and the symbol Ⓒ a committed location. Urgent and committed locations are used to model locations where no time may pass. Leaving a committed location has priority over leaving non-committed locations.

While the semantic state space of UTA is infinite due to the real-valued clock variables, the *symbolic semantics* [5] abstracts from certain points in time and uses clock zones instead. A clock zone is a set of constraints on clock variables such that for all values within the zone, the UTA behaves equivalently. For example, in Fig. 1, the clock values $x \in [\texttt{mintime}, \texttt{maxtime}]$ can be merged into

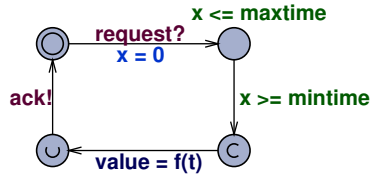


Fig. 1. Example UPPAAL Timed Automaton (UTA)

one clock zone. A symbolic state is a tuple $(\bar{l}, D)$, where $\bar{l}$ is a set of locations and D is a clock zone. We can define *symbolic timed traces* as follows:

Definition 3 (Symbolic Timed Traces). *A* symbolic timed trace *is a possibly infinite sequence of timed actions a_i, where the time is specified as a* clock zone *D_i:*

$$TTr = (D_1, a_1)(D_2, a_2)...(D_i, a_i)...$$

In UTA, an action may be composed of an event e and a manipulation of the global data space v. As a consequence, when the system state changes, we can observe an event e, a modified data space v, or both. As we are only interested in observable behavior, we partition the events and the data space into three disjoint sets of input (Ev_{in}/V_{in}), output (Ev_{out}/V_{out}), and internal events/variables (Ev_{int}/V_{int}). A *timed input or output trace* can be defined as follows:

Definition 4 (Timed input and output traces). *A timed input/output trace o of a state s is a (possibly infinite) sequence of observations, where each observation is a tuple (e, D, v) consisting of an event $e \in Ev_{in/out}$, a clock zone D in which the event occurs, and a vector $v \in V_{in/out}$ containing the values of data variables that are externally visible as inputs/outputs at this time. We use $o^k.e$, $o^k.D$, resp. $o^k.v$ to access the event, clock zone, and data vector of the kth element in a timed input or output trace. The trace is written as:*

$$o(s) = (e_0, D_0, v_0)(e_1, D_1, v_1)...(e_i, D_i, v_i)...$$

4 VeriSTA

Our framework for the automated **Veri**fication of **S**ystemC designs using **T**imed **A**utomata (VeriSTA) is based on a combination of model checking and conformance testing. The overall structure of the framework is shown in Fig. 2. We assume that SystemC designs are developed in a refinement process that starts with an abstract design, which is stepwise refined down to the final implementation. One of these manual refinement steps is shown on the left of Fig. 2. We use model checking to verify the abstract design. Then, we generate conformance tests to check whether refined designs conform to the verified abstract design. We have presented an approach for model checking of safety, timing, and

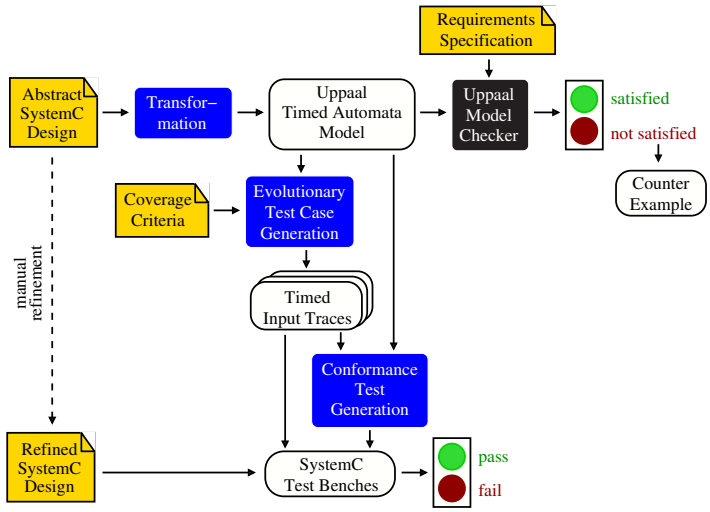


Fig. 2. VeriSTA Framework

memory-related properties of SystemC/TLM designs in [24, 28, 47, 48, 26]. The general idea is to map the informally defined semantics of SystemC to the formally well-defined semantics of UPPAAL timed automata. Using this mapping, it is, under some restrictions, possible to transform a given SystemC design into a semantically equivalent UPPAAL model. We have developed the **S**ystemC to **T**imed **A**utomata **T**ransformation **E**ngine (STATE) to perform this transformation automatically. The tool is licenced under GPL and freely available at `http://www.pes.tu-berlin.de/state_project`. After the transformation, the UPPAAL model checker can be used to verify safety, timing, and memory-related properties. The model checking flow is shown in the upper part of Fig. 2. We have demonstrated the applicability of the model checking approach with several case studies, amongst others with an Anti-Slip Regulation and Anti-lock Braking System (ASR/ABS), and an industrial TLM implementation of the AMBA advanced high-performance bus (AHB). The experiments show the applicability and performance, but also the limitations of the model checking approach.

The absolute guarantees of safety, liveness, and timing properties for all possible input scenarios come at the price of a high computational effort and high memory consumption. As a consequence, model checking can only be applied to relatively small or abstract designs. To obtain a continuous quality assurance throughout the whole design process, we have extended our framework with an approach to generate conformance tests from abstract designs. The conformance tests can automatically be generated for given timed input traces by computing all possible timed output traces using a symbolic execution, as we have shown in [25, 27]. These traces are then used to generate test benches that evaluate the conformance of a given low-level design fully automatically. In [23, 26], we presented techniques for the coverage-driven generation of timed input traces

for SystemC designs. In [26], we generated timed input traces with a high port coverage using the UPPAAL model checker. In [23], we presented an evolutionary test generation algorithm that achieves high transition coverage. With that, our framework facilitates a fully-automatic verification and validation flow that supports the whole co-design process for digital embedded real-time systems.

5 Formal Semantics for SystemC

As a formal foundation for our VeriSTA framework, we have developed a formal semantics for SystemC by defining a transformation from SystemC into UPPAAL timed automata [24, 28, 47, 48]. The transformation preserves the (informally defined) behavioral semantics and the structure of a given SystemC design and can be applied fully automatically. It can handle all relevant SystemC language elements, including process execution, interactions between processes, dynamic sensitivity and timing behavior. It supports the full TLM 2.0 standard and features a formal memory model that enables the verification of memory-related properties. It requires a few restrictions on the system under verification. First, we do not handle dynamic process creation. This should hardly narrow the applicability of the approach, as dynamic process creation is rarely used in safety-critical embedded systems. Second, UPPAAL supports only bounded integer variables, arrays and structs. This is a more severe restriction, but is leveraged by the fact that most data types used in SystemC designs can be converted to bounded integers. Third, due to the structure of our formal memory model, we do not support any type casts, and finally, we do not support recursive functions, function pointers, or direct memory accesses. These restrictions can again be considered as minor restrictions.

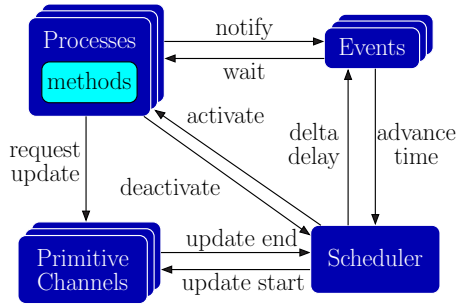


Fig. 3. Representation of SystemC designs in UPPAAL

A SystemC model consists of a set of modules, which contain methods and processes. While methods contain sequential code, processes are executed concurrently and their execution is triggered by events. Fig. 3 shows how we represent SystemC designs in UPPAAL. Each method is mapped to a single timed

automata template. Process automata are used to encapsulate these methods and care for the interactions with event objects, the scheduler, and primitive channels. The interactions are modeled using UPPAAL channels. For example, the processes notify events using `notify`, and the events trigger the processes over a `wait` channel if they are notified. To formalize the execution semantics of SystemC, we have developed UTA models of the SystemC scheduler, processes, events and other SystemC constructs such as primitive channels. As the semantics of the SystemC elements is only informally defined in [31], we *define* their formal semantics with our UTA models. Furthermore, for the transformation of a given SystemC design, these models can be instantiated arbitrarily often. With that, we achieve a compositional transformation, i. e., we transform each module separately and compose the system in a final instantiation and binding phase. As a consequence, the transformation scales well even for large SystemC designs.

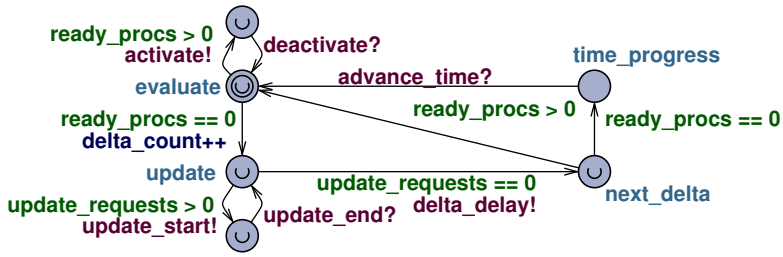


Fig. 4. The UTA model of the Scheduler

As an example for our formalization using UTA, consider the model of the SystemC scheduler shown in Fig. 4. The scheduler executes SystemC designs in an event-discrete simulation. It is a cooperative scheduler, and it uses delta cycles to impose a partial order on concurrent processes. Each delta-cycle consists of an **evaluate** and an **update** phase (both shown with labeled locations in Fig. 4). In the **evaluate** phase, each process that is ready for execution is triggered by sending an **activate** signal. If no process is ready for execution, the **update** phase is started. There, primitive channels are updated by sending an **update_start** signal to the corresponding update methods. If a delta-cycle is completed, the scheduler informs all events by sending **delta_delay**. This causes delta-delayed notifications to take place, and may cause new processes to become ready for execution. Then, the scheduler starts a new delta-cycle. Otherwise, if no more processes are ready for execution at the current time, the scheduler changes to the location **time_progress** and waits for the **advance_time** signal sent by the next pending timed notification. With that, the semantics of the SystemC scheduler is completely modeled in UTA and thus formally defined. Note that in the evaluate and update phases, all processes and updates are triggered through the same binary channel. This ensures that the order of processes

(and update methods) is chosen non-deterministically. This corresponds to the definition of the SystemC semantics in [31] and it has the advantage that we can also detect errors that are generally non-detectable with a simulator that always chooses a deterministic execution order.

6 Model Checking and Conformance Testing

Our transformation from SystemC into UPPAAL timed automata enables the application of the UPPAAL model checker to verify that the abstract design or specification satisfies a given set of requirements expressed in temporal logics.[3] Furthermore, we use the UPPAAL model to generate conformance tests [25]. The aim of conformance testing is to check for a given set of input traces whether the implementation conforms to the specification. To achieve this, we statically compute all possible timed output traces for a given timed input trace by performing a breadth-first search on the symbolic state space of the abstract UTA model. As a formal foundation for this computation, we have defined a complete definition of the symbolic semantics of UTA in [27], which extends the semantics defined by [5] with data variables and binary and broadcast channels. The symbolic execution provides all possible timed output traces as an *acceptance graph*, as shown in Fig. 5. In the acceptance graph, all completely computed timed output traces are joined into a **pass** node. Output traces for which the computation was interrupted due to an internal limit on the maximal number of computation step are joined into an **inconclusive** node. Implicitly, all traces that are not contained in the acceptance graph correspond to a **fail**. In other words, the acceptance graph only accepts traces that are possible in the abstract model. From that, we can generate SystemC test benches that evaluate the conformance of refined designs to the abstract model fully automatically. As a formal foundation

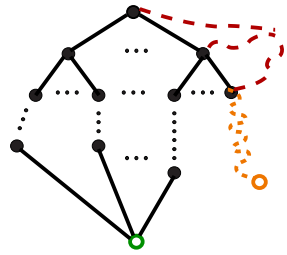


Fig. 5. Acceptance Graph

for that, we use the *relativized timed input/output conformance relation (rtioco)*

[3] UPPAAL supports a subset of CTL.

defined in [39], and slightly modify it such that it allows an explicit refinement of symbolic traces. The advantage of this is that we can explicitly express that the implementation may refine the timing behavior of the specification. To this end, we have defined the following refinement relation on timed traces:

Definition 5 (Refinement relation on timed traces). *A timed output trace o_I that can be observed on a system I refines a timed output trace o_S that can be observed on a system S if they contain the same events and variable values, and if the clock zone of each observation on I is a subset of the corresponding observation on S. We use the index set K over the elements of o_I and o_S.*

$$o_I \leq o_S \quad \textit{iff} \quad \forall k \in K : o_I^k.e = o_S^k.e \wedge o_I^k.D \subseteq o_S^k.D \wedge o_I^k.v = o_S^k.v$$

Note that the number of elements in o_I and o_S may be finite or infinite. If the number of elements is finite for one of the timed output traces, $o_I \leq o_S$ if in addition $length(o_I) = length(o_S)$.

The set of timed output traces that can be observed on a system $\mathcal{S}$ under environmental constraints $\mathcal{E}$ for a given input trace σ are denoted by $\mathrm{TTr}_o((\mathcal{S}, \mathcal{E}), \sigma)$. The set of timed input traces that are provided by an environment are denoted by $\mathrm{TTr}_i(\mathcal{E})$. With that, we can define a refinement relation on sets of timed output traces:

Definition 6 (Refinement relation on sets of timed traces). *We define the refinement relation $\sqsubseteq$ on sets of timed output traces with respect to a given environment $\mathcal{E}$ such that for each output trace of the implementation o_I an output trace of the specification o_S with $o_I \leq o_S$ must exist:*

$$\begin{gathered} TTr_o((I, \mathcal{E}), \sigma) \sqsubseteq TTr_o((S, \mathcal{E}), \sigma) \quad \textit{iff} \\ \forall o_I \in TTr_o((I, \mathcal{E}), \sigma) : (\exists o_S \in TTr_o((S, \mathcal{E}), \sigma) : o_I \leq o_S) \end{gathered}$$

Based on the definition of timed traces and refinement on sets of timed output traces, we can use the *relativized timed input/output conformance (rtioco)* defined in [39], where we replace the refinement operator with our version:

Definition 7 (Relativized timed input/output conformance (rtioco)). *I* conforms *to S under the environmental constraints $\mathcal{E}$ if for all timed input traces $\sigma \in TTr_i(\mathcal{E})$ the set of timed output traces of I is a refinement of the set of timed output traces of S for the same input trace.*

$$I \textit{ rtioco } S \quad \textit{iff} \quad \forall \sigma \in TTr_i(\mathcal{E}) : TTr_o((I, \mathcal{E}), \sigma) \sqsubseteq TTr_o((S, \mathcal{E}), \sigma)$$

The *rtioco* relation is derived from the *input/output conformance (ioco)* relation of Tretmans and de Vries [57] by taking time and environment constraints into account. Under the assumption of weak input enabledness, i. e., if any input is accepted in any state, the *rtioco* coincides with timed trace inclusion [39]. The definition ensures that the implementation may not produce outputs that are unexpected by the specification and that it must produce outputs whenever it is expected by the specification. With respect to a CTL property that is satisfied on the abstract specification, this means that the implementation also respects this property for all given input traces, as long as the CTL property is formulated with respect to observable output behavior. The internal behavior of both models may differ. Note that our conformance evaluation approach is applicable on multiple levels of abstraction, as long as the corresponding adapters, which translate between the abstraction levels, are provided. As a consequence, it can be used for quality assurance throughout the whole design flow. As we generate the test benches for automated conformance evaluation *offline*, we can reuse them in each development step. With that, we ensure the consistency between designs on different abstraction levels with a comparatively low computational effort. The whole conformance evaluation approach is automatically applicable. To reduce the computational effort and memory consumption of the test generation process itself, we have developed a set of optimizations that make use of the specifics of the SystemC semantics to drastically reduce the number of semantic states which have to be kept in memory during state-space exploration.

7 Evolutionary Generation of Timed Test Traces

With our conformance testing approach, we can automatically evaluate whether an embedded real-time system implementation conforms to its specification for a given timed input trace. To systematically derive timed input traces that cover as much of the system behavior as possible is another difficult challenge for embedded real-time systems.

We have presented an evolutionary algorithm for the generation of timed input traces from Uppaal timed automata models in [23]. The general idea behind evolutionary testing approaches is to transform test goals into search problems.

Evolutionary testing is usually realized in the following way (see Fig. 6). In a first step, a set of either random or manual test data is created. In the context of the evolutionary search algorithms, this set is called a population. Each test datum in this set represents an individual.

In a second step, each individual is executed on the system under test (or software or model) and the execution leads to some observable behavior. This is evaluated to a fitness with regard to the test goal. If at least one individual meets the test goal at this point, the search has found its solution (stopping condition, step 3). If not, depending on the used algorithm, the fitness of each individual is used to either select it for removal from the population or for transformation (typically, this means selection, mutation and recombination of

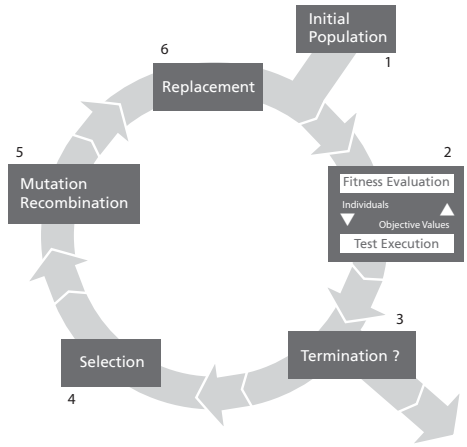


Fig. 6. The Evolutionary Test Cycle

individuals, step 4 and 5). Individuals that have been transformed are then joined with the old population. Depending on the concrete algorithm, the ratio of old and transformed individuals range from complete replacement of the old population to a mix of some individuals from the new and some of the old population (step 6). Regardless of the exact operators used, after these steps there is a set of new individuals that constitute the population of the next population. This new population is again executed and evaluated for its fitness. Every population is supposed to reveal at least equal or better fitness values for its individuals on average. This way of constant improvement with regard to better fitness values aims at finding at least one individual with properties that lead to an execution that triggers the stopping condition, i. e., fulfills the test goal.

In order to apply evolutionary algorithms to test data search problems, it is usually required to define the representation of the test data as individuals, the operators to apply for the evolution of the individuals (selection, mutation, recombination), a stopping condition for the evolutionary cycle and most importantly the fitness function that represents the test goal in a numerical way. If coverage is the test goal, as in our application, the process has to be split into several optimization runs, each trying to cover one element.

In our work [23], we have presented an approach to encode timed traces as individuals of an evolutionary algorithm. To cope with the requirement of long traces, we encode timed traces as a sequence of blocks, where each block defines a sequence of inputs that are sent with a fixed distance of time between each input. By introducing blocks, we make the timed traces easier to handle and reduce the search space for the test generation problem, but still keep a high degree of flexibility by varying the number, size, and contents of the blocks. Furthermore, we adopt a set of operators to create, to recombine, and to mutate these timed traces, which benefit from the block structure. Moreover, we present a fitness

function to evaluate the quality of an individual with respect to a given coverage goal. To this end, we adopt the fitness function of [58, 41, 34], which combines the concepts of approach-level and branching distance. The approach-level requires a graph like structure where distances between two nodes (locations or transitions) are computable. Applied to timed automata and transition coverage this means that we compute the approach level of a given location with respect to a transition we want to cover by determining the minimal path between them. The branching distance is an indicator of the likeliness to enter the transition that would bring us closer to our test goal and is computed using the guard of that transition [54]. For example, if we have a guard $a < b$, the distance is 0 if $a - b < 0$ and $a - b$ otherwise. In our case, where we use timed automata for the test generation, the guard may contain clock variables and variables that are heavily time-dependent. As a consequence, traces with a better timing behavior receive a better fitness. Thus, our fitness function implicitly takes timing into account.

The overall fitness is a combination of approach-level and branching distance and is simply computed as follows:

$$f = \mathrm{AL} + \mathrm{norm}(\mathrm{BD})$$

We have used an empirical value based on the experiments to normalize BD values in the range of 0..1. BD is normalized in order to let the approach level overrule the branching distance.

8 Experimental Results

To evaluate the applicability and the error detecting capability of our approach, we have implemented the whole VeriSTA framework, including the directed test generation process. The framework is implemented in Java, and thus it is portable and easily extendible. As a frontend for SystemC designs, we have used the Karlsruhe SystemC Parser KaSCPar [14]. Furthermore, we have used the Uppaal model checker [4] for the verification of safety and timing properties. The overall framework allows a fully-automated verification of SystemC designs, without any user-interaction. The only necessary inputs are one or more SystemC designs that should be verified, a requirements specification written in Uppaal (i. e., in a subset of CTL) and at least one SystemC test bench to provide the structural information of a test bench instantiation for the automatic test bench generation.

For our experiments, we have used two case studies: (1) a TLM implementation of the AMBA advanced high-performance bus (AHB) provided by Carbon Design Systems and (2) an Anti-Slip Regulation and Anti-lock Braking System (ASR/ABS).

The Advanced Microcontroller Bus Architecture (AMBA) Bus is an on-a-chip bus introduced by ARM Ltd[4] in 1996. The AMBA advanced high performance

[4] http://www.arm.com

bus (AHB) protocol was introduced in 1999 and features burst transfers, split transactions and a bus width of up to 128 bits. The AMBA AHB is one of the most popular on-chip bus architectures in IP-based embedded SoCs and it is used in many multimedia applications.

The AMBA AHB is a synchronous clocked bus. The timing and arbitration of the AMBA AHB are described in [2]. An AMBA AHB transfer starts with a bus request initiated by a bus master. The arbiter collects all bus requests and sends a grant signal to one master. The granted bus master then drives the address and control signals. These signals provide information on the address, direction and width of the transfer, as well as an indication if the transfer forms part of a burst. AMBA AHB uses separate read and write buses to move data from slave to the master and the other way around. Every transfer consists of an address and control cycle and one or more cycles for the data. The TLM 2.0 implementation of the AMBA AHB provided by Carbon Design Systems implements this by multiple clocked non-blocking transports for each transfer. The design implements an arbiter and a decoder as specified in [2]. The slave components receive transactions and read or write from/to memory, respectively.

The original model consists of about 1500 LOC. To meet the assumptions of our approach, we performed the following modifications: (1) we changed the sockets to TLM standard sockets, (2) we replaced the generic payload type with a specific one, (3) we replaced operators for dynamic memory management (e.g., *new, delete*) by static memory allocation and (4) we only transfer constant data through the bus. The latter modification drastically simplifies the verification problem. However, our focus is on verifying the correct concurrent behavior, synchronization, timing, and memory safety which do not depend on the data that is transfered over the bus. The modified model consists of about 1600 LOC.

The ASR/ABS system monitors the speed at each wheel and regulates the brake pressure in order to prevent wheel lockup or loss of traction and to improve the driver's control over the car. It consists of dedicated wheel speed sensors, a hydraulic modulator to control the brake pressure, an electronic control unit that runs the control algorithms, and a control area network (CAN) bus. To measure the wheel speed, the number of incoming wheel signals (ticks) within a certain amount of time are used to compute the speed at each wheel. The results are sent to an electronic control unit (ECU) via a CAN bus. On the ECU, the control algorithms for brake pressure control and anti slip regulation are executed. A simple real-time operating system (RTOS) is used to schedule the tasks and to provide an interrupt layer for interactions with the CAN bus.

The ASR/ABS was developed by using a typical HW/SW co-design flow following the TLM approach. We started with an abstract design where processes communicate over FIFO channels and where timing is only coarsely estimated. This abstract model allows the validation and verification of the control algorithm without having to cope with communication details. Then, we refined the design by using a high-speed CAN bus for communication, an interrupt layer and a simple scheduling algorithm on the electronic control unit. While the abstract design consists of approximately 500 LOC and contains 4 modules and

	1M1S	1M2S	2M1S	2M2S
(1) deadlock freedom	6:17	12:06	37:28	84:25
(2) only one master	-	-	24:07	54:32
(3.a) bus granted to M1	3:56	7:47	24:10	54:06
(3.b) bus granted to M2	-	-	24:10	54:02
(4) timing	11:37	22:24	69:15	152:04
(5.a) all pointers always valid	4:36	9:16	27:57	64:07
(5.b) no null pointer accesses	6:36	13:19	39:12	87:29

Table 1. Model Checking of the AMBA AHB (Verification times in min:sec)

18 processes that communicate over 12 channels, the refined design consists of over 2500 LOC and contains 8 modules and 26 processes that communicate over 23 channels. Overall, the ABS is well-suited to assess the performance and error detecting capability of our testing approach because we can use the abstract design to generate timed input traces and then automatically evaluate the error detecting capability on the refined design.

To assess the performance and the error detecting capability of our VeriSTA framework, we have verified the correctness of our case studies using the Uppaal model checker. For the AMBA AHB system, which makes heavy use of pointers, we have verified safety, liveness, timing, and memory-related properties. For the ASR/ABS system, we have verified safety and timing properties of the abstract model. Then, we have generated timed test inputs with complete transition coverage. Finally, we have executed the generated timed test inputs on our refined ASR/ABS system. We have used our conformance test approach to evaluate the simulation results. All experiments were run on a machine with an Intel Pentium 3.4 GHz CPU and 4 GB main memory.

8.1 Results from the AMBA advanced high-performance bus

For the AMBA AHB system, we have checked the following properties: (1) deadlock freedom, (2) only one master can write to the bus at a time (mutual exclusion), (3.a) if master M1 requests the bus, it will eventually get it granted, (3.b) if master M2 requests the bus, it will eventually get it granted, (4) bus transfers are finished within a given time limit, (5.a) all pointers in the design are always either null, or they point to a valid part of the memory array corresponding to their type, (5.b) the design never tries to access memory via a null pointer. To evaluate the scalability of our approach we have used different design sizes (from 1 master and 1 slave, 1M1S, to 2 master and 2 slaves, 2M2S). The results of the verification are shown in Table 1.

All properties have been proven to be satisfied at the end of the verification phase, and within reasonable time. During the verification, we have detected a bug in the original design which led to a deadlock situation. When a transaction is split into several separate transfers, a counter variable is used to store the number of successful transfers before the split occurs. This variable was not

Property	counter-examples		verification	
no deadlock	–	–	722.54 s	✓ *(maybe)*
ABS reacts within time limit	2.56 s	↯	555.56 s	✓ *(maybe)*
ASR reacts within time limit	3.51 s	↯	844.15 s	✓ *(maybe)*

Table 2. Model Checking of the ASR/ABS system (Verification times in seconds)

reset in the original design. As a consequence, all split transactions besides the first one failed. This is a typical example which is both difficult to detect and to correct with simulation. With our approach, the generation of a counter example took only a few minutes. Due to the structure preservation of our transformation and the graphical visualization in Uppaal, it was easy to understand the cause of the problem.

8.2 Results from the ASR/ABS system

For the ASR/ABS system, we checked the following properties: (1) deadlock freedom, (2) if the deceleration exceeds a given limit, the brake pressure is reduced within a given time limit, (3) if the acceleration exceeds a given limit, the brake pressure is increased within a given time limit. The experimental results are shown in Table 2. In the beginning, only the first property turned out to be satisfied. With little debugging effort, supported by the counter-examples produced by the model checker, we found out where the problem arose from. We made an error during the conversion of ticks into speed. This error was not detected by our previously used test cases because we unintentionally solely used test cases where changes in the tick speed happened only at full seconds. Note again that the generation of counter-examples is very fast. After the removal of defects, the state space turned out to be too large to be completely explored with our 4 GB main memory. This is mainly due to the large data ranges used in the ASR/ABS example, which cannot symbolically be captured by the Uppaal model checker. However, with bit state hashing enabled, we were able to verify that the properties are *maybe* satisfied. The result of model checking with bit state hashing is an under-approximation of the state space, i. e., the state space is only partially explored. However, it is still very well-suited for debugging purposes. Furthermore, given that a large hash table is available, it can be expected that the verification results have a high probability to be reliable if the model checker does not find a counter-example.

The effort of the timed test input generation together with the conformance test generation process was approximately 2 days. Full coverage is very difficult to achieve for the ASR/ABS example. This is due to the fact that a very specific and comparably large trace is necessary to trigger the anti-lock braking system. This trace must be chosen from an infinite set of possible input traces. In particular, the timing of the input signals plays a crucial role. However, the generation of a test bench only has to be done once in the whole development flow.

	random test generation	evolutionary test generation
missing assignment	54.48 %	60.43 %
wrong assignment	56.36 %	63.85 %
missing condition	71.42 %	93.75 %
wrong condition	66.66 %	91.86 %
communication delay	73.58 %	99.05 %
communication loss	68.18 %	86.36 %
communication error	64.86 %	97.29 %

Table 3. Error Detecting Capability

To assess the error detecting capability of our testing approach, we have injected over 1000 defects from seven fault classes into the (presumably) correct refined ASR/ABS system and checked whether these defects are detected by our automatically generated test cases. For the defect injection, we have used the following fault classes and injected each defect once at each possible occurence in the code.

- ***Missing assignment***: An assignment is removed.
- ***Wrong assignment***: An assignment is manipulated by adding or subtracting some value.
- ***Missing condition***: A condition is completely removed.
- ***Wrong condition***: A condition is manipulated.For example, reversed logical operators are used, e. g., `<` is replaced by `>`, `==` by `!=`, `&&` by `||`.
- ***Communication delay***: The communication is delayed by an arbitrary amount of time.
- ***Communication loss***: Some data is lost during the communication.
- ***Communication error***: Wrong data is communicated.

With the above fault classes and the corresponding defect injection, we obtained 1224 faulty designs for the ASR/ABS design. Table 3 summarizes the result. To evaluate our evolutionary algorithm for the generation of timed input traces, we compared it to timed input traces that were generated by random. The resulting error detection rates are shown in Table 3. In all cases, we achieved better results than random test generation. Furthermore, the timed output traces have been evaluated fully-automatically using our conformance testing approach. The experiments demonstrate that our approach is very well-suited to verify and test embedded real-time systems.

9 Conclusion

The verification of embedded real-time systems requires to have some formal model of the system behavior that explicitly considers real-time dependant behavior. Timed automata provide a formalism that enables an elegant and well-analyzable description of such systems. In particular, it enables fully-automatic

verification of safety, liveness, and timing properties expressed in temporal logics. However, timed automata - as well as other formal models for the precise description of embedded real-time systems - are rarely used in practical development processes.

Our framework for the HW/SW co-**Veri**fication of **S**ystemC designs using **T**imed **A**utomata (VeriSTA) enables an automatic transformation of SystemC designs into Uppaal timed automata. With that, we provide a formal semantics for SystemC designs that explicitly considers time-dependent behavior. The resulting Uppaal model forms the basis for formal verification and test automation. To achieve a quality assurance process that assists the design flow of embedded real-time systems efficiently and continuously from an abstract design down to the final implementation, our VeriSTA framework combines the power of model checking and testing. The general idea is to formally verify abstract designs via model checking and to generate conformance tests consisting of timed input and output traces for all subsequent refinements of the abstract design. This combination is especially well-suited for HW/SW co-verification of embedded real-time systems, as it can be applied to both the hardware and the software parts of a given design on different levels of abstraction. Furthermore, both the model checking and the conformance testing are automatically applicable. As a formal basis, we have defined a transformation from SystemC to Uppaal timed automata. The tranformation preserves both the informally defined behavior and the structure of SystemC designs. Furthermore, it is automatically applicable, introduces a negligible overhead and produces compact and comparably small models. The automatic generation of Uppaal models from SystemC designs enables the use of the Uppaal model checker and tool suite. Thus, we can prove that the design meets a given (real-time) requirements specification fully automatically. In addition, we use the formal model for automated conformance test generation and for the generation of timed test input traces that achieve a given coverage goal.

Our experimental results show that safety and timing properties of complex abstract designs can be verified with reasonable effort. Furthermore, our approach for the generation of timed input traces together with the corrresponding expected timed outputs allows the detection of significantly more faults than randomized test case generation for our case study of an Anti-Slip Regulation and Anti-lock Braking System (ASR/ABS).

Acknowledgements

We thank the students who contributed to the implementation of our tools and case studies, in particular Joachim Fellmuth, Florian Friedemann, Verena Klös, Timm Liebrenz, Tobias Pfeffer, Marcel Pockrandt, and Daniela Rose.

References

1. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)

2. ARM Ltd.: AMBA3 AHB-Lite Protocol Specification (2006)
3. Behjati, R., Sabouri, H., Razavi, N., Sirjani, M.: An effective approach for model checking systemc designs. In: Application of Concurrency to System Design (ACSD). pp. 56–61 (2008)
4. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on Uppaal. In: Formal Methods for the Design of Real-Time Systems. pp. 200–236. LNCS 3185, Springer (2004)
5. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Lecture Notes on Concurrency and Petri Nets. pp. 87–124. LNCS 3098, Springer (2004)
6. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time petri nets. IEEE Transactions on Software Engineering 17(3) (1991)
7. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool tina construction of abstract state spaces for petri nets and time petri nets. International Journal of Production Research 42(14), 2741–2756 (2004)
8. Blanc, N., Kroening, D., Sharygina, N.: Scoot: A Tool for the Analysis of SystemC Models. In: International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS). LNCS, vol. 4963, pp. 467–470. Springer (2008)
9. Bruschi, F., Ferrandi, F., Sciuto, D.: A framework for the functional verification of SystemC models. Int. Journal on Parallel Programming 33(6), 667–695 (2005)
10. Chen, M., Mishra, P., Kalita, D.: Towards RTL test generation from SystemC TLM specifications. In: IEEE International High Level Design Validation and Test Workshop. pp. 91–96. IEEE Computer Society (2007)
11. Cimatti, A., Micheli, A., Narasamdya, I., Roveri, M.: Verifying SystemC: A software model checking approach. In: Formal Methods in Computer-Aided Design (FMCAD), 2010. pp. 51 –59. IEEE (2010)
12. Cimatti, A., Griggio, A., Micheli, A., Narasamdya, I., Roveri, M.: Kratos - A Software Model Checker for SystemC. In: Computer Aided Verification, LNCS, vol. 6806, pp. 310–316. Springer (2011)
13. Cimatti, A., Narasamdya, I., Roveri, M.: Software model checking systemc. IEEE Trans. on CAD of Integrated Circuits and Systems 32(5), 774–787 (2013)
14. FZI Research Center for Information Technology: KaSCPar - Karlsruhe SystemC Parser Suite
15. Garavel, H., Helmstetter, C., Ponsini, O., Serwe, W.: Verification of an industrial SystemC/TLM model using LOTOS and CADP. In: Formal Methods and Models for Co-Design (MEMOCODE). pp. 46–55. IEEE (2009)
16. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.: Fdr3 a modern refinement checker for csp. In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 8413, pp. 187–201. Springer (2014)
17. Groetker, T.: System Design with SystemC. Kluwer Academic Publishers (2002)
18. Große, D., Drechsler, R.: Checkers for SystemC designs. In: Formal Methods and Models for Codesign. pp. 171–178. IEEE Computer Society (2004)
19. Große, D., Kühne, U., Drechsler, R.: HW/SW Co-Verification of Embedded Systems using Bounded Model Checking. In: Great Lakes Symposium on VLSI. pp. 43–48. ACM Press (2006)
20. Große, D., Peraza, H., Klingauf, W., Drechsler, R.: Embedded Systems Specification and Design Languages, chap. Measuring the Quality of a SystemC Testbench by Using Code Coverage Techniques, pp. 73–86. Springer (2008)
21. Habibi, A., Moinudeen, H., Tahar, S.: Generating Finite State Machines from SystemC. In: Design, Automation and Test in Europe. pp. 76–81. IEEE (2006)
22. Habibi, A., Tahar, S.: An Approach for the Verification of SystemC Designs Using AsmL. In: Automated Technology for Verification and Analysis. pp. 69–83. LNCS 3707, Springer (2005)

23. Hänsel, J., Rose, D., Herber, P., Glesner, S.: An evolutionary algorithm for the generation of timed test traces for embedded real-time systems. In: Intl. Conf. on Software Testing, Verification and Validation (ICST). pp. 170–179. IEEE (2011)
24. Herber, P., Fellmuth, J., Glesner, S.: Model Checking SystemC Designs Using Timed Automata. In: International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). pp. 131–136. ACM press (2008)
25. Herber, P., Friedemann, F., Glesner, S.: Combining Model Checking and Testing in a Continuous HW/SW Co-Verification Process. In: Tests and Proofs. LNCS, vol. 5668. Springer (2009)
26. Herber, P., Glesner, S.: A HW/SW Co-Verification Framework for SystemC. ACM Transactions on Embedded Computing Systems (2013)
27. Herber, P., Pockrandt, M., Glesner, S.: Automated Conformance Evaluation of SystemC Designs using Timed Automata. In: IEEE European Test Symposium (2010)
28. Herber, P., Pockrandt, M., Glesner, S.: Transforming SystemC Transaction Level Models into UPPAAL Timed Automata. In: Formal Methods and Models for Codesign (MEMOCODE). pp. 161 – 170. IEEE Computer Society (2011)
29. Hessel, A., Larsen, K.G., Mikucionis, M., Nielsen, B., Pettersson, P., Skou, A.: Formal Methods and Testing, chap. Testing Real-Time Systems Using UPPAAL. Springer (2008)
30. Hessel, A., Larsen, K.G., Nielsen, B., Petterson, P., Skou, A.: Time-optimal test cases for real-time systems. In: Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES). pp. 114–130. LNCS 2931, Springer (2003)
31. IEEE Standards Association: IEEE Std. 1666–2011, Open SystemC Language Reference Manual (2011)
32. Jaghoori, M.M., Movaghar, A., Sirjani, M.: Modere: The Model-checking Engine of Rebeca. In: ACM Symposium on Applied Computing. pp. 1810–1815. SAC '06, ACM (2006)
33. Junior, A.D., Cecilio da Silva, D.J.: Code-coverage Based Test Vector Generation for SystemC Designs. In: IEEE Computer Society Annual Symposium on VLSI. pp. 198–206. IEEE (2007)
34. Kalaji, A.S., Hierons, R.M., Swift, S.: Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM). In: Intl. Conf. on Software Testing Verification and Validation. pp. 230–239. IEEE (2009)
35. Karlsson, D., Eles, P., Peng, Z.: Formal verification of SystemC Designs using a Petri-Net based Representation. In: Design, Automation and Test in Europe (DATE). pp. 1228–1233. IEEE Press (2006)
36. Kirchsteiger, C.M., Trummer, C., Steger, C., Weiss, R., Pistauer, M.: Distributed Embedded Systems: Design, Middleware and Resources, chap. Specification-based Verification of Embedded Systems by Automated Test Case Generation, pp. 35–44. Springer (2008)
37. Krichen, M., Tripakis, S.: Real-time testing with timed automata testers and coverage criteria. In: Joint conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS-FTRTFT). pp. 134–151. LNCS 3253, Springer (2004)
38. Kroening, D., Sharygina, N.: Formal Verification of SystemC by Automatic Hardware/Software Partitioning. In: MEMOCODE. pp. 101–110. IEEE (2005)
39. Larsen, K.G., Mikucionis, M., Nielsen, B.: Formal Approaches to Software Testing, chap. Online Testing of Real-time Systems Using Uppaal, pp. 79–94. Springer (2005)

40. Le, H.M., Große, D., Herdt, V., Drechsler, R.: Verifying SystemC using an intermediate verification language and symbolic simulation. In: DAC. p. 116 (2013)
41. Lefticaru, R., Ipate, F.: Automatic State-based Test Generation using Genetic Algorithms. In: International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. pp. 188–195. IEEE Computer Society (2007)
42. Man, K.L.: An Overview of SystemCFL. In: Research in Microelectronics and Electronics. vol. 1, pp. 145– 148 (2005)
43. Mathaikutty, D.A., Ahuja, S., Dingankar, A., Shukla, S.: Model-driven test generation for system level validation. In: IEEE International High Level Design Validation and Test Workshop. pp. 83–90. IEEE Computer Society (2007)
44. Müller, W., Ruf, J., Rosenstiel, W.: SystemC: Methodologies and Applications, chap. An ASM based SystemC Simulation Semantics, pp. 97–126. Kluwer Academic Publishers (2003)
45. Nielsen, B., Skou, A.: Automated test generation from timed automata. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 343–357. LNCS 2031, Springer (2001)
46. Patel, H.D., Shukla, S.K.: Model-driven validation of SystemC designs. EURASIP Journal on Embedded Systems 2008(3), 1–14 (2008)
47. Pockrandt, M., Herber, P., Glesner, S.: Model Checking a SystemC/TLM Design of the AMBA AHB Protocol. In: IEEE/ACM Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia). pp. 66 – 75. IEEE Computer Society (2011)
48. Pockrandt, M., Herber, P., Klös, V., Glesner, S.: Model Checking Memory-Related Properties of Hardware/Software Co-Designs. In: Embedded Systems: Design, Analysis and Verification. Proceedings of the International Embedded Systems Symposium (IESS). Springer (2013)
49. Razavi, N., Behjati, R., Sabouri, H., Khamespanah, E., Shali, A., Sirjani, M.: Sysfier: Actor-based formal verification of SystemC. ACM Trans. Embedded Comput. Syst. 10(2), 19 (2010)
50. Ruf, J., Hoffmann, D.W., Gerlach, J., Kropf, T., Rosenstiel, W., Müller, W.: The Simulation Semantics of SystemC. In: Design, Automation and Test in Europe. pp. 64–70. IEEE Press (2001)
51. Schneider, S.: Concurrent and Real Time Systems: The CSP Approach. John Wiley & Sons, Inc. (1999)
52. da Silva, K.R.G., Melcher, E.U.K., Araujo, G., Pimenta, V.A.: An automatic testbench generation tool for a SystemC functional verification methodology. In: Symposium on Integrated circuits and system design. pp. 66–70. ACM Press (2004)
53. SystemC Verification Working Group - SCV: SystemC Verification Standard (2006)
54. Tracey, N.J.: A Search-based Automated Test-Data Generation Framework for Safety-Critical Software. Ph.D. thesis, University of York (2000)
55. Traulsen, C., Cornet, J., Moy, M., Maraninchi:, F.: A SystemC/TLM semantics in Promela and its possible applications. In: 14th Workshop on Model Checking Software (SPIN '07). pp. 204–222. LNCS 4595, Springer (2007)
56. Veanes, M., Campbell, C., Grieskamp, W., Schulte, W., Tillmann, N., Nachmanson, L.: Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer. In: Formal Methods and Testing, LNCS, vol. 4949, pp. 39–76. Springer (2008)
57. de Vries, R.G., Tretmans, J.: On-the-fly conformance testing using SPIN. Software Tools for Technology Transfer 2(4), 382–393 (2000)
58. Wegener, J., Buhr, K., Pohlheim, H.: Automatic Test Data Generation For Structural Testing Of Embedded Software Systems By Evolutionary Testing. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1233–1240. Morgan Kaufmann Publishers Inc. (2002)

MARTE/CCSL for Modeling Cyber-Physical Systems

Frédéric Mallet[1,2,3]

[1] Univ. Nice Sophia Antipolis, I3S UMR 7271 CNRS, 06900 Sophia Antipolis, France
[2] INRIA Sophia Antipolis Méditerranée, 06900 Sophia Antipolis, France
[3] East China Normal University, Software Engineering Institute, Shanghai, China
Frederic.Mallet@unice.fr

Abstract. Cyber Physical Systems (CPS) combine digital computational systems with surrounding physical processes. Computations are meant to control and monitor the physical environment, which in turn affects the computations. The intrinsic heterogeneity of CPS demands the integration of diverse models to cover the different aspects of systems. The UML proposes a great variety of models and is very commonly used in industry even though it does not prescribe a particular way of using those models together. The MARTE profile proposes a set of extensions to UML in a bid to allow for the modeling of real-time and embedded systems (RTES). Yet CPS are a wider class of systems than mere RTES. Hence a legitimate question arises as whether MARTE can be used for CPS as well. This paper illustrates some possible uses of MARTE to model CPS and uses logical clocks as a way to bring together the different models.

Keywords: Heterogeneous models, Logical clocks, Fuel management system

1 Introduction

Cyber-Physical Systems combine digital computational systems with surrounding physical processes. Computations are meant to control and monitor the physical environment, which in turn affects the computations. The intrinsic heterogeneity of CPS demands the integration of diverse models to cover the different aspects of systems. The Unified Modeling Language (UML) proposes a great variety of models and is very commonly used in industry even though it does not prescribe a particular way of using those models together. The profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE [25]) proposes a set of extensions to UML in a bid to allow for the modeling of real-time and embedded systems. Yet CPS are a wider class of systems than mere RTES. Hence a legitimate question arises as whether MARTE can be used for CPS as well. This paper illustrates some possible uses of MARTE to model CPS. It deliberately leaves aside the traditional aspects well covered in the literature on traditional discrete embedded systems. The central concepts put forward is the notion of

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_2,

logical clock introduced by Lamport and used ever since in many applications. The logical clocks have been brought into the UML world through MARTE time subprofile. They are used in this chapter as handles attached to various modeling elements (whether structural or behavioral, related to hardware or software). The Clock Constraint Specification Language (CCSL), initially defined in an annex of MARTE, is used to constrain/animate these handles (and the associated modeling elements) to specify the expected behavior along with some expected interactions and synchronizations amongst the diagrams.

Section 2 starts by enumerating the main, generally accepted, characteristics of cyber-physical systems. Then Section 3 introduces the UML profile for MARTE by insisting on aspects previously highlighted as important for modeling cyber-physical systems. In particular, after a general overview it elaborates on the time and allocation subprofiles before introducing CCSL. Section 4 illustrates each aspect highlighted using a common example of fuel management system borrowed from previous works on CPS.

2 Main Characteristics of CPS

The main characteristics of Cyber Physical Systems and main design challenges have been identified some years ago [17, 18]. We focus here only on proposing a modeling framework based on standard notations to capture some most important aspects. We present our view focusing on what is most relevant to understand our approach. The following characteristics are further discussed in this section and illustrated in Section 4.

CPS are:

- heterogeneous, in the sense that they combine various models of computations relying on both discrete and continuous time abstractions;
- platform-aware and resource-constrained, and thus the software depends on various non-functional properties imposed by the platform;
- time-sensitive and often safety-critical;
- widely distributed with heterogeneous interconnects.

CPS are first and foremost complex systems and as such designing them requires several models, usually hierarchical, to fully capture the different aspects and views, whether structural or behavioral. Structural models include a description of the components or blocks of the systems and of the communication media involved. Behavioral models include hierarchical state machines and data-flow streaming diagrams. Expected or faulty interactions with the surrounding environment can be captured as a set of use cases or requirements that correspond to positive or negative scenarios. Such models are usually called **heterogeneous** in the sense that they combine different models, each of which may follow a different *model of computation*.

CPS also have the main characteristics of embedded systems, which are usually **platform-aware**. Contrary to standard software engineering, embedded system design depends a lot on the execution platform on which the system

should execute, be it a system-on-a-chip (SoC), with multiple computing resources and a complex memory hierarchy, or a wide scale distributed system, with potentially all the variety of interconnects and communication media. This awareness of the platform makes it important to account for how and when the available resources are accessed or 'consumed', considering together both the spatial and temporal dimensions. The spatial dimension is not only about how much resource is available but also about where the resources are physically located in the system relative to each other. How much resource is available is indeed easy since it is usually given by the technology used and the targeted selling price. However, how the resources are used makes all the difference between two a priori equivalent products. The spatial dimension encompasses the interconnect topology, *i.e.,* physical parallelism available, but also and more importantly where the data and programs are allocated. Indeed, the distance between the data memory and the computing resource that executes the program largely impact the fetching time that may potentially largely exceed the computing time. Then, this spatial distribution requires to perform the temporal scheduling of both the execution of programs and the routing of data from memory to computing resources, forth and back. This leads to *logical concurrency* coming both from the physical parallelism and the inherent data and control dependencies of the application. CPS are therefore **resource-constrained** and **time-sensitive** systems. Even though the resources (memory size, computing power) are not necessarily as scarce as they used to be, nevertheless finding the right tradeoff between the resource usage, the computation speed and the cost makes it a multi-criteria optimization problem difficult to solve. The cost is not only measured in terms of money, but this includes all kinds of additional extra functional properties (like power, energy, thermal dissipation), also called **non-functional properties**.

More than being mere embedded systems, CPS are usually made of multiple interconnected embedded subsystems, some of which are computing devices and some other being physical devices. This requires some abilities to describe **heterogeneous interconnects**, while simpler embedded systems usually only rely on homogeneous communication structures. Being made of several computing devices also constitute a big step since it requires to model the whole system as a closed model, with software, devices but also with the environment and the expected **continuous interactions** with this environment. In standard software development, the environment is by definition outside the system to be developed. Close loop systems have been modeled for several years with tools and techniques to find approximate solutions to differential equations are well established. However, the integration with discrete models still causes problems in many tools and each of them proposes ad-hoc solutions. A seamless integration with UML models is still to be proposed.

Finally, cyber-physical systems are often big and often interacts directly with users that are not even aware of the computer. The size is an aggravating factor since a single system concerns potentially millions of people (smart cities, intelligent transportation systems...). It means some CPS are **safety-critical**,

just like embedded systems but at a larger scale. It then increases the demand to have sound models along with verification tools. Sometimes, they also require certification tools to be accredited and allowed to be use in public environments. However, we do not address at all the certification issue here.

To summarize, CPS demand an integration between continuous models, classical state-based or data-flow models, hardware descriptions, non-functional constraints. UML offers a tool-neutral non-proprietary solution that already contains most of the required notations. However, those notations need to be tailored to capture specific aspects of CPS (time, non-functional properties, continuous models). Both MARTE and SysML offer some extensions dedicated to these goals, and we discuss here some examples of useful features of either MARTE or SysML to model CPS. These notations also need to come with adequate, not tool-specific, **explicit semantics** if we are to address safety-critical issues. In this paper, the semantics is given using the Clock Constraint Specification Language [21], a formal specification language defined as a companion of MARTE in an annex.

3 The UML Profile for MARTE

This section gives a basic introduction to the MARTE profile. It gives a general overview and focus on some specific aspects that are further developed in the following sections, like the time and allocation subprofile. It also gives an introduction to the Clock Constraint Specification Language defined in an annex of MARTE since CCSL is used in this chapter to describe the interactions between the models. This is done by specifying some causal relationships between the models and the way they synchronize. This section does not cover the whole MARTE specification and a comprehensive review of MARTE along with some complementary information on how to use it for modeling cyber physical systems can be found in a dedicated book [28].

3.1 Overview

UML and its extensions. The Unified Modeling Language [26] is a general-purpose modeling language specified by the Object Management Group (OMG). It proposes graphical notations to represent all aspects of a system from the early requirements to the deployment of software components, including design and analysis phases, structural and behavioral aspects. As a general-purpose language, it does not focus on a specific domain and maintains a weak, informal semantics to widen its application field. However, when targeting a specific application domain and especially when building trustworthy software components or for critical systems where life may be at stake, it becomes necessary to extend the UML and attach a formal semantics to its model elements. The simplest and most efficient extension mechanism provided by the UML is through the definition of profiles. A UML profile adapts the UML to a specific domain by adding new concepts, modifying existing ones and defining a new visual representation for others. Each modification is done through the definition of annotations (called

stereotypes) that introduce domain-specific terminology and provide additional semantics. However, the semantics of stereotypes must be compatible with the original semantics (if any) of the modified or extended concepts, *i.e.,* the base metaclass.

The UML profile for MARTE [25] extends the UML with concepts related to the domain of real-time and embedded systems. It supersedes the UML profile for Schedulability, Performance and Time (SPT [24]) that was extending the UML 1.x and that had limited capabilities. UML 2.0 has introduced a simple (or even simplistic) model of time and has proposed several new extensions that made SPT unusable. Therefore MARTE has been defined to be compatible with UML Simple Time model and now supersedes SPT as the official OMG specification. SysML [10] is another extension dedicated to systems engineering. We use some notations coming from SysML and we introduce those notations when required. The task forces of MARTE and SysML have synchronized their effort to allow for a joint use of both profiles.

The remainder of this subsection gives an overview of the MARTE, which is made up of three parts: *Foundations*, *Design* and *Analysis*.

MARTE Foundations. The foundation part of MARTE is itself divided into five chapters: CoreElements, NonFunctionalProperties (NFP), Time, Generic Resource Modeling (GRM) and Allocation.

CoreElements define configurations and modes, which are key parameters for analysis.

In real-time systems, preserving the non-functional (or extra-functional) properties (power consumption, area, financial cost, time budget...) is often as important as preserving the functional ones. The UML proposes no mechanism at all to deal with non-functional properties and relies on mere string for that purpose. The NFP subprofile offers mechanisms to describe the quantitative as well as the qualitative aspects of properties and to attach a unit and a dimension to quantities. It defines a set of predefined quantities, units and dimensions and supports customization. NFP comes with a companion language called Value Specification Language (VSL) that defines the concrete syntax to be used in expressions of non-functional properties. VSL also recommends syntax for user-defined properties.

Time is often considered as an extra-functional property that comes as a mere annotation after the design. These annotations are fed into analysis tools that check the conformity without any actual impact on the functional model: *e.g.,* whether a deadline is met, whether the end-to-end latency is within the expected range. Sometimes though, time can also be of a functional nature and has a direct impact on what is done and not only when it is done. All these aspects are addressed in the time chapter of MARTE. The next section elaborates on the time subprofile.

GRM chapter provides annotations to capture the available resources on which the applicative part shall be deployed.

The allocation chapter gives a SysML-compatible way to make this deployment. In MARTE, we use the wording allocation since the UML deployment usually implies (in people's minds) a physical distribution of a software artifact onto a physical node. Allocation in MARTE goes further. It encompasses the physical distribution of software onto hardware, but also of tasks onto operating system processes, and, more importantly, it covers the temporal distribution (or scheduling) of operating parts that need to share a common resource (*e.g.,* several tasks executing on a single core processor, distributed computations communicating through an interconnect). This subprofile is further described in subsection 3.3.

MARTE for design. The design part has four chapters: High Level application modeling, Generic component modeling, Software Resource Modeling, and Hardware Resource Modeling.

The first chapter describes real-time units and active objects. Active objects depart from passive ones by their ability to send spontaneous messages or signals, and react to event occurrences. Normal objects, the passive ones, can only answer to the messages they receive or react on event occurrences. The three other chapters provide a support to describe resources used and in particular execution platforms on which applications may run. A generic description of resources is provided, including stereotypes to describe communication media, storages and computing resources. Then this generic model is refined to describe software and hardware resources along with their non-functional properties.

MARTE for analysis. The analysis part also has a chapter that defines generic elements to perform model-driven analysis on real-time and embedded systems.

This generic chapter is specialized to address schedulability analysis and performance analysis.

The chapter on schedulability analysis is not specific to a given technique and addresses various formalisms like the classic and generalized Rate Monotonic Analysis (RMA), holistic techniques, or extended timed automata. This chapter provides all the keywords usually required for such analyzes.

Finally, the chapter on performance analysis, even if somewhat independent of a specific analysis technique, emphasizes on concepts supported by the queueing theory and its extensions.

MARTE usages and extensions. MARTE extends the UML for real-time and embedded systems but should be refined by more specific profiles to address specific domains (avionics, automotive, silicon) or specific analysis techniques (simulation, schedulability, static analysis). Because MARTE targets different domains and/or different analysis techniques, the time model of MARTE is rich and combines physical and logical clocks. This is further explained in the next subsection.

We have briefly reviewed here the whole specification of MARTE. However, MARTE is not expected to be used as a whole on a single specification, as his

usage chapter states. It is expected to be the base of several complementary methodologies that cover different aspects of a system. This chapter covers one aspect and proposes a partial usage to capture important features. It is not intended to offer a comprehensive cover of all aspects.

3.2 Time in MARTE

This subsection only gives a very brief introduction to the time model of MARTE. See [3, 2, 20] for more explanations and examples.

Time in SPT is a metric time with implicit reference to physical time. As a successor of SPT, MARTE supports this model of time. UML 2, issued after SPT, has introduced a model of time called *SimpleTime*. This model also makes implicit references to physical time, but is too simple for use in real-time applications, and was initially devised to be extended in dedicated profiles.

MARTE goes beyond SPT and UML 2. It adopts a more general time model suitable for system design. In MARTE, Time can be physical, and considered as continuous or discretized, but it can also be logical, and related to user-defined clocks. Time may even be multiform, allowing different times to progress in a non-uniform fashion, and possibly independently to any (direct) reference to physical time. In MARTE, time is represented by a collection of *Clocks*. The use of word *Clock* comes from vocabulary used in the synchronous languages. They may be understood as a specific kind of events on which constraints (temporal, hence the name, but also logical ones) can be applied. Each clock specifies a totally ordered set of instants, *i.e.,*, a sequence of event occurrences. There may be dependence relationships between the various occurrences of different events. Thus this model, called the MARTE time structure, is akin to the Tagged Systems [16]. To cover continuous and discrete times, the set of instants associated with a clock can either be dense or discrete.

Figure 1 shows the main stereotypes introduced by MARTE Time subprofile.

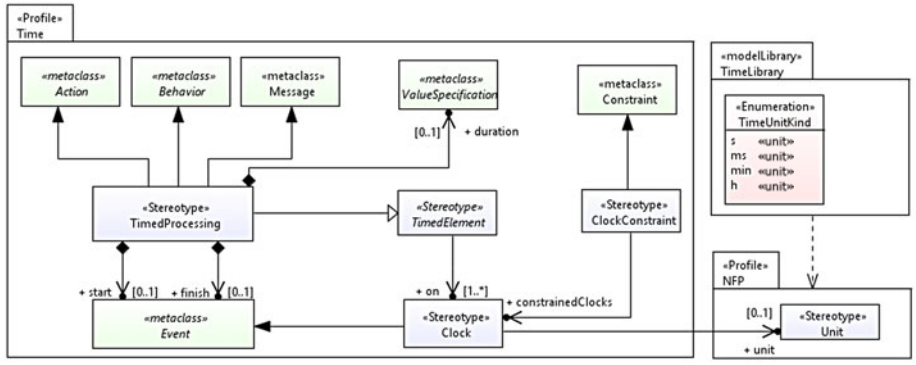


Fig. 1. Excerpt of MARTE Time subprofile

Stereotype Clock is one foundational stereotype that extends UML metaclass Event. A Clock carries specific information such as its actual unit, and values of quantitative (resolution, offset...) or qualitative (time standard) properties, if relevant.

TimedElement is another stereotype introduced in MARTE. A timed element is an abstract stereotype that associates at least one clock with a modeling element. TimedProcessing is a specialization of TimedElement, which extends the UML metaclasses Action, Behavior and Message. It defines a start and a finish event for a given action/behavior/message. These events (which are usually clocks) specify when the action starts or when it finishes. TimedProcessing also specifies the duration of an action. Duration is also measured on a given logical or physical clock. In a MARTE model of a system, stereotype TimedElement or one of its specializations is applied to model elements which have an influence on the specification of the temporal behavior of this system. The expected behavior of such TimedElements is controlled by a set of ClockConstraints. Those constraints specify dependencies between the various occurrences of events. CCSL, which is further described in the following subsection, can be used to specify those constraints formally.

The MARTE Time subprofile also provides a model library named TimeLibrary. This model library defines the enumeration TimeUnitKind which is the standard type of time units for chronometric clocks. This enumeration contains units like s (second), its submultiples, and other related units (*e.g.,* minute, hour). The library also predefines a clock called IdealClock, which is a dense chronometric clock with the second as time unit. This clock is assumed to be an ideal clock, perfectly reflecting the evolutions of physical time. It should be imported in user's models with references to physical time concepts (*e.g.,* frequency, physical duration).

3.3 Allocation in MARTE

Since embedded systems are platform-aware, one need a way to map the elements of the application onto the execution platform. This aspect is specifically addressed by the allocation subprofile of MARTE, which is further described in this subsection.

The wording *Allocation* has been retained to distinguish this notion from UML Deployment diagrams. Deployments are reserved to deploy artifacts (*e.g.,* source code, documents, executable, database table) onto deployment targets (*e.g.,* processor, server, database system). The MARTE allocation is much more general than that. For instance, it is meant to represent the allocation of a program onto a system thread, or of a process onto a processor core. More generally, it is used to represent the association of an element (action, message, algorithm) that consumes a resource onto the consumed resource (processing unit, communication media, memory). Wordings 'mapping' or 'map' have also been discarded since they very often refer to a function and then map one input from a domain to one single output in the co-domain. The allocation process,

however, is an n-to-m association. Take for instance, a bunch of tasks that need to be scheduled on several cores.

Note that the wording *execution platform* has been preferred to 'architecture' or 'hardware'. Indeed, architecture is a way to describe the structure of a system, while an execution platform contains both structural and behavioral parts. On the other hand, the execution platform is not necessarily a piece of hardware. It can be a piece of software, a virtual machine, a middleware, an operating system or a mixed platform that combines software and hardware intellectual properties (IPs).

Finally, it is also important to note that this notion of allocation is common between MARTE and SysML in a bid to ease the combination of the two profiles. In particular, for CPS both profiles must be used jointly [28].

Figure 2 shows the two main stereotypes of the subprofile, Allocate and Allocated. Allocate represents the allocation itself, while Allocated may be used on both sides to mark either the element that is allocated or the resource onto which an element is allocated. The property nature is meant to distinguish two kinds

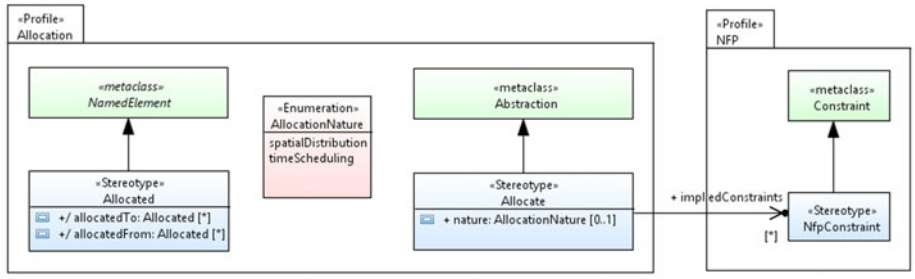


Fig. 2. Exerpt of MARTE Allocation subprofile

of possible allocations: *spatial* and *temporal*. Typically, when messages are allocated onto a buffer or a memory, this is a spatial allocation. Indeed, the message will consume/use some cells of the memory. However, when two tasks are allocated onto a processing unit, this is a temporal allocation (scheduling); It means the two tasks must be scheduled to avoid resource conflicts. When a program is allocated onto a processor, it can be seen both as spatial and temporal; Spatial because the program consumes disk and memory resources, temporal because while this program executes, another one cannot execute simultaneously. The allocation usually implies constraints that describe precisely the impact (or cost) of the allocation on the non-functional properties. This is why there is an association to a specific MARTE stereotype called NfpConstraint, *i.e.,* to capture the constraints implied by the allocation in terms of memory consumption, power consumption, execution time, for instance.

Figure 3 shows an example of allocation borrowed from [23]. The upper part depicts an algorithm as an UML activity diagram. Two input values are captured

as $in1$ and $in2$. Those values are processed respectively by actions $step1$ and $step2$ producing two intermediate values. Those two intermediate values are used by $step3$ to process the final result. The bottom part shows a possible execution platform as a composite structure diagram. It contains two tasks (or threads, or processes) that communicate through a shared memory. Each task is scheduled at a different frequency.

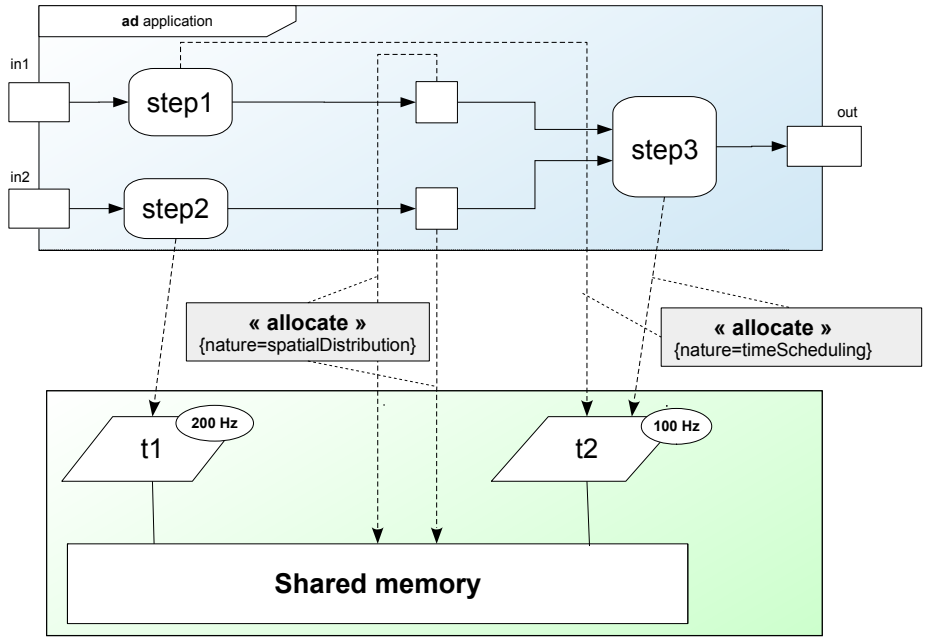


Fig. 3. Example of Allocation: spatial allocation vs. temporal scheduling

MARTE allocation is used in two different ways. The first way corresponds to a temporal scheduling operation. Indeed, the three steps from the application part are scheduled on the tasks of the execution platform. In this example, $step1$ and $step3$ are allocated to task $t2$, while $step1$ is allocated to task $t1$. The second kind of allocation is more of a spatial nature. It allocates the intermediate values to the shared memory. Even though there are different in nature, both kinds of allocations imply some constraints (temporal and non-functional) that should be checked and captured as $NfpConstraint$s. The spatial distribution actually implies that one must check whether there is enough space available in the shared memory for the two communications to overlap. The temporal scheduling implies that the two actions $step1$ and $step2$ must not execute simultaneously. Both kinds of constraints can be captured and analyzed with CCSL [23].

3.4 The Clock Constraint Specification Language

The Clock Constraint Specification Language [21] is a declarative language to build specifications of systems by accumulation of constraints that progressively refine what can be expected from the system under consideration. The specification can be used and analyzed with our tool TimeSquare [8]. CCSL mainly targets embedded systems and was then designed to capture constraints imposed by the applicative part, the execution platform or also external requirements from the users, like non-functional properties. Constraints from the application and the execution platform are bound together through allocation constraints also expressed in CCSL. The central concept in CCSL are the logical clocks, which have been successfully used for their multiform nature by synchronous languages to build circuits and control-oriented systems, to design avionic systems with data-flow descriptions or design polychronous control systems [4]. They have also been used outside the synchronous community to capture partial orderings between components in distributed systems [15]. We promote their use here for capturing the concurrency inherent to the application, the parallelism offered by the execution platform and synchronization constraints induced by the allocation.

Definition 1 (Logical discrete clock). *A* logical discrete clock *c is defined as an infinite sequence of ticks: $(c_n)_{n=1}^{\infty}$.*

Logical clocks are used to represent noticeable events of the system, *e.g.,* starting/finishing the execution of an agent, writing/reading a data from a place/memory, acquiring/releasing a resource; Their ticks are the successive (totally ordered) occurrences of the events.

In CCSL, the expected behavior of the system is described by a specification that constrains the way the clocks can tick. Basically, a CCSL specification prevents clocks from ticking when some conditions hold.

Definition 2 (CCSL specification). *A* CCSL *specification is a tuple $Spec = \langle C, Cons \rangle$, where C is a finite set of clocks and $Cons$ is a finite set of constraints.*

A CCSL specification denotes a set of schedules. If empty, there is no solution, the specification is invalid. If there are many possible schedules, it leaves some freedom to make some choices depending on additional criteria. For instance, some may want to run everything as soon as possible (ASAP), others may want to optimize the usage of resources (processors/memory/bandwidth).

Definition 3 (Schedule). *A* schedule *σ over a set of clocks C is a sequence of sets representing the ticking clocks. $\sigma : \mathbb{N} \to 2^C$.*

Given a clock c, a *step* $s \in \mathbb{N}$ and a schedule σ, $c \in \sigma(s)$ means that clock c ticks at step s for this particular schedule.

Definition 4 (Satisfaction). *A schedule σ satisfies a specification ($\sigma \models Spec$) if it satisfies all of its constraints ($\forall cons \in Cons, \sigma \models cons$).*

Note that there are usually an infinite number of schedules that satisfy a specification, we only consider the ones that do not have empty steps: $\forall n \in \mathbb{N} \setminus \{0\}, \sigma(n) \neq \emptyset$.

The schedules are synchronous but the specification is called polychronous since the constraints are independent of each others and do not know when the next step is going to happen or whether it has actually happened. In Esterel [5], there is an instruction called *pause* that waits until the next step. This is not directly possible in CCSL without explicitly building the union of all the clocks to know when something is going to happen next.

New CCSL constraints can be defined from kernel ones (see [1]) in dedicated libraries. Before presenting newly-defined constraints, we introduce here some of the kernel constraints needed. Some constraints are stateless, *i.e.,* the constraint imposed on a schedule is identical at all steps; others are stateful, *i.e.,* they depend on what has happened in previous steps.

The first basic stateless CCSL constraint is Subclocking, which prevents a (sub)clock from ticking when a (master) clock cannot tick.

Definition 5 (Subclocking). *Let a, b be two logical clocks. A schedule σ satisfies the subclocking constraint on a and b ($a \boxed{\subseteq} b$) if the following condition holds:*
$\sigma \models a \boxed{\subseteq} b \iff \big(\forall n \in \mathbb{N}, b \notin \sigma(n) \implies a \notin \sigma(n)\big)$.

Contrary to full synchronous systems, we never assume the existence of a global master clock from which all the other clocks should derive. When $a \boxed{\subseteq} b$ and $b \boxed{\subseteq} a$, the two clocks a and b are said to be synchronous, this is denoted as $a \boxed{=} b$.

One example of simple stateless CCSL constraint is Union.

Definition 6 (Union). *Let a, b be two logical clocks. A schedule σ satisfies the union constraint on a and b if the following condition holds: $\sigma \models u \triangleq a \boxed{+} b \iff \big(\forall n \in \mathbb{N}, u \in \sigma(n) \iff (a \in \sigma(n) \vee b \in \sigma(n))\big)$*

Note that Union is commutative and associative, we sometimes use an n-ary extension of this binary definition.

For stateful constraints, we use the history of clocks for a specific schedule.

Definition 7 (History). *Given a schedule σ, the* history *over a set of clocks C is a function $H_\sigma : C \times \mathbb{N} \to \mathbb{N}$ defined inductively as follows for all clocks $c \in C$:*
$H_\sigma(c, 0) = 0$
$\forall n \in \mathbb{N}, c \notin \sigma(n) \implies H_\sigma(c, n+1) = H_\sigma(c, n)$
$\forall n \in \mathbb{N}, c \in \sigma(n) \implies H_\sigma(c, n+1) = H_\sigma(c, n) + 1$

A simple example of a primitive stateful CCSL clock constraint is Causality. When an event causes another one, the effect cannot occur if the cause has not. In CCSL, causality can be instantaneous.

Definition 8 (Causality). *Let a, b be two logical clocks. A schedule σ satisfies the causality constraint on a and b if the following condition holds: $\sigma \models a \boxed{\preccurlyeq} b \iff \big(\forall n \in \mathbb{N}, H_\sigma(a, n) \geq H_\sigma(b, n)\big)$*

A small extension of Causality includes a notion of temporality and is called Precedence.

Definition 9 (Precedence). *Let a, b be two logical clocks and $\delta \in \mathbb{Z}$. A schedule σ satisfies the precedence constraint on a and b if the following condition holds:* $\sigma \models a \boxed{\delta \prec} b \iff \big(\forall n \in \mathbb{N}, H_\sigma(a,n) - H_\sigma(b,n) = -\delta \implies b \notin \sigma(n)\big)$

The primitive CCSL precedence is defined as: $a \boxed{\prec} b \equiv a \boxed{0 \prec} b$. A bounded version of precedence is defined as $a \boxed{\prec_N} b \equiv a \boxed{\prec} b \wedge a \boxed{N \prec} b$. In this paper, we use a particular case called alternation and defined as $a \boxed{\sim} b \equiv a \boxed{\prec_1} b$.

4 Illustration

This section illustrates the joint use of UML, MARTE, SysML and CCSL to model some aspects of CPS and more importantly to unify them.We start by discussing the fuel management system of an aircraft [12, 9]. This example illustrates one possible way to take into accounts some non-functional properties of CPS (see subsection 4.1). Then subsection 4.2 puts together two (heterogeneous) models and combine discrete control (*e.g.,* state machines) along with continuous aspects (*e.g.,* SysML parametrics).

4.1 Non-Functional Properties

To model the non-functional properties, we use the NFP subprofile of MARTE. To illustrate this use, we consider here, as an example, an Aircraft Fuel Management System (AFMS) [12]. Several fuel tanks are spread in the aircraft to feed the two engines. This is done for redundancy purpose; in case there is a problem with one tank then another one can be used. It also serves for balancing the weight during the flight, some fuel is moved from one tank to another depending on various requirements that vary depending on the current flight phase (mode). Weight constraints are not identical during takeoff, landing or while in cruise mode. Some valves are used to transfer fuel from one tank to another, sometimes relying on gravity, sometimes using pumps. All the transfers are processed by a computer. Such a system is clearly safety-critical since fuel is of course of major importance in an aircraft. It can also be considered as a cyber-physical systems since the whole system has to mix physical models of fuel flows along with digital aspects to deal with flight modes and various states of the different components.

Figure 4 shows the types introduced in a UML class diagrams to build the model of the aircraft fuel management system. The fuel available in a tank is measured by its weight and we use the type NFP_Weight defined in a dedicated MARTE library called MeasurementUnits. We also use enumeration WeightUnitKind that defines, amongst others, unit kg and dimension Mass (M). This process is similar to the one used previously to define the time units (s, ms. . .) for dimension Time (T).

Transfers are measured in kg/s. Such a unit is not predefined in MARTE. We then define ourselves a new dimension FlowRate (M/T^{-1}), a new unit kg/s and a

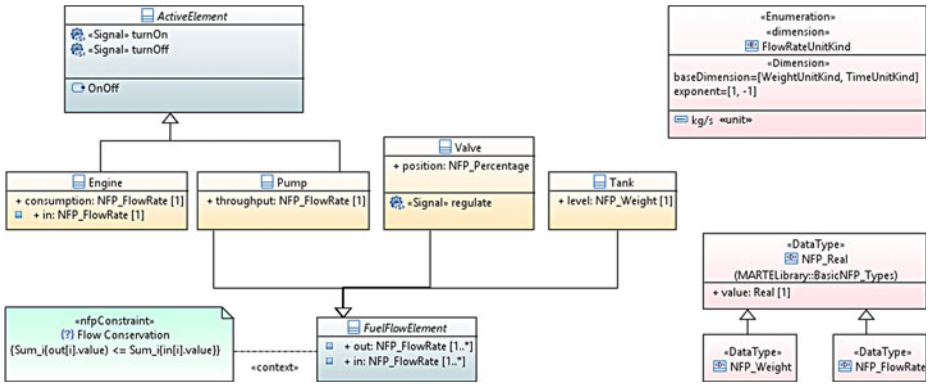


Fig. 4. NFP Types for Aircraft Fuel Management System

new type NFP_FlowRate using MARTE stereotypes defined in the NFP subprofile. The mechanism used here is further described in [28].

Once basic data types are defined we define two kinds of abstract elements. ActiveElements are elements that can be turned on and off; Engine and Pump are two examples of such kinds of elements. Such elements are commanded by two discrete signals turnOn and turnOff. FuelFlowElements are mechanical elements that connect several input flow rates to several output flow rates. For instance, a valve, when opened, transfers a part of its input flow rate coming from a tank or a pump to an output flow rate ending up either in another valve or directly into a tank. The valve is regulated by a signal that determines the position of its throttle. This position is given as a percentage[4], 0 % meaning closed while 100 % means fully opened. As shown in the figure some elements can be of both kinds. For instance, a pump can be turned on and off but also connects one input flow from a tank to an output flow, either to another tank or to a valve.

Some constraints may be specified on those elements. For instance, a fuel flow element must preserve flows between inputs and outputs, *i.e.,* they cannot be more fuel going out that the amount flowing in. This is captured by a specific NFPConstraint (see Eq. 1):

$$\sum_i out[i].value \leq \sum_i in[i].value \tag{1}$$

4.2 Heterogeneous modeling with explicit interactions

Cyber Physical systems imply using several models to combine various aspects of the system. The previous subsection shows a static view of the types involved. This subsection focuses on behavioral aspects and elaborates on the example AFMS. Two kinds of information, very different in nature, must be combined here. On the one hand, there is the flow dynamics and on the other end, there

[4] NFP_Percentage is also defined in a standard library of MARTE

is the discrete control of the valves, pumps and engines. For the former we rely on SysML parametrics, while we use UML state machines and logical clocks for the latter.

SysML has introduced a specific diagram, called *Parametrics* to describe *acausal* relations. Causal here must be understood as functional. It is a relation established between some inputs to produce an input (or even multiple outputs). Acausal, however, is a way to build relations where there is no input or output but a set of property values that follow given laws. This is not specific to continuous phenomenon but is clearly useful in that case. Considering the example of the Valve in the AFSM, one can capture an acausal relation between the output flow, input flow and the position of the throttle. Figure 5 shows the corresponding SysML parametrics. A SysML constraint block, here called FlowRegulation captures a constraint between three real numbers (see Eq. 2).

$$exhaust = intake \times k \tag{2}$$

This (simple) law is applied to property values that represent the input flow rate, the output flow rate and the throttle position. One could consider that *intake* and k are inputs from which output *exhaust* is deduced. However, capturing this as an acausal relation one can deduce k that must be applied considering the actual *intake* flow and the expected *exhaust* flow.

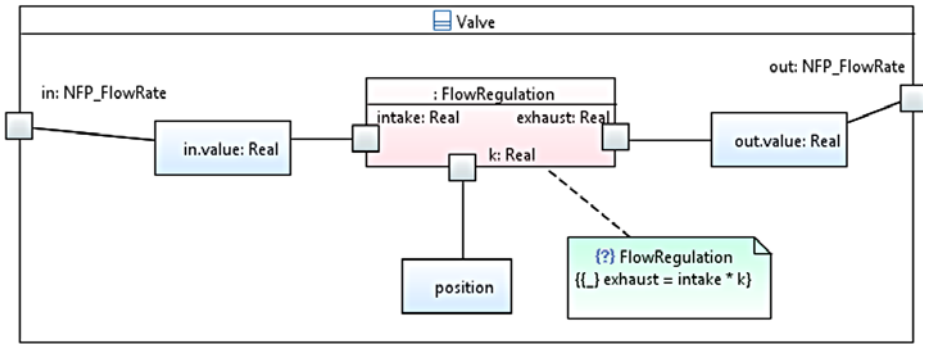


Fig. 5. Valve regulation: SysML Parametrics

Moving now to the pump, the regulation system is similar except that the pump is either *on* or off. The actual state of the pump can easily be captured as a UML state machine (see the bottom of Fig. 6). The expected behavior of this state machine can be captured using CCSL constraints. This requires to attach logical clocks to the *relevant* modeling elements. Relevant means the ones that play a role in the dynamics that is to be specified. In that example, this is done by associating clocks to the events involved in the triggering of the transitions, *i.e.*, *turnOn* and $turnOff$. Indeed, in UML, each transition has a trigger; A trigger is associated with an event; Events become clocks by using the stereotype defined

in MARTE time profile (see Section 3.2). The clock constraint then can carry the specification shown in Eq. 3, which specifies that the state machine alternates from state *On* to state *Off*, forth and back.

$$\begin{aligned} & turnOn \boxed{\sim} turnOff \\ \texttt{reads}\ & turnOn\ \textbf{alternatesWith}\ turnOff \end{aligned} \tag{3}$$

The actual flow rate of pumps depends on its throughput and on its input flow rate. Similarly to valves, the relation between the flows can be captured using SysML parametrics. The state machine and the SysML parametrics are combined together using additional CCSL logical clocks and constraints as shown in Figure 6. This can be done for instance using the allocation (see Section 3.3). Each allocation implies a set of constraints, in this case, it implies a ClockConstraint carrying a CCSL specification. In our example, the constraint may be the

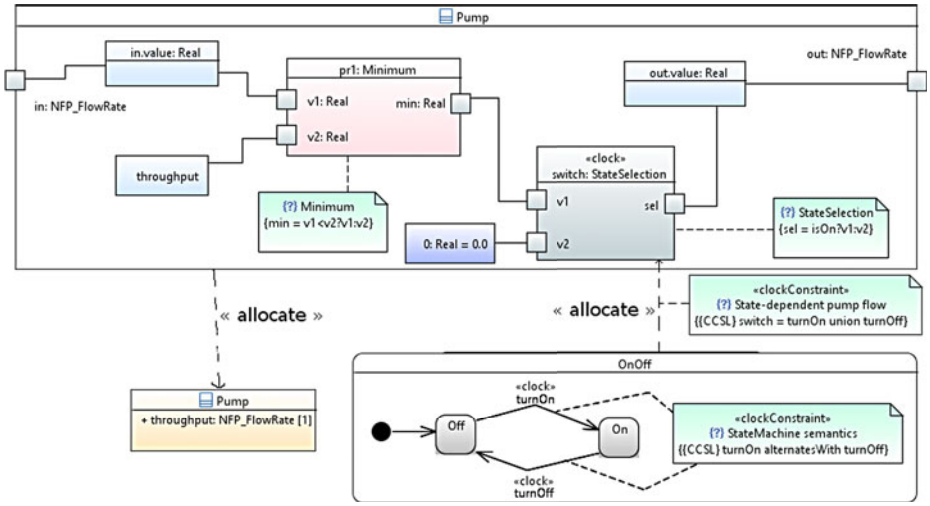


Fig. 6. Pump regulation: UML/MARTE State Machine with SysML parametrics

one given in Eq. 4.

$$\begin{aligned} & switch \boxed{=} turnOn \boxed{+} turnOff \\ \texttt{reads}\ & switch = turnOn\ \textbf{union}\ turnOff \end{aligned} \tag{4}$$

Generally speaking logical clocks should be attached to various model elements in different models and a clock constraint then specifies the relationships between those clocks. Within a single model this is useful to make explicit the expected semantics of the model. A CCSL specification is used in that case to specify what should happen and when it should happen. This is done by describing the causal relationships and the possible synchronizations between the

events of the system. In this example, we did it manually, but for classical models and assuming a 'standard' semantic interpretation, this can be done in a automatic way [7]. Whether automatic or manual, it is important to have an explicit semantics that can be used as a *golden reference* to conduct further verifications. Such a golden reference can also be used to validate a potential exogeneous transformation to a formal language.

More examples of such a use of CCSL to make a semantic adaptation between heterogeneous models can be found in other works [6, 14] although it is done in a manner independent of UML and MARTE.

4.3 Heterogeneous interconnects

As underlined in Section 2, CPS very often have a complex heterogeneous interconnect. The kind of interconnect may vary a lot depending on the systems. UML alone is already very rich to describe structural architectural aspects. Composite structure diagrams are one solution to describe the different parts of a system. SysML however distinguishes two kinds: the block diagrams and the internal block diagrams. While block diagrams are very close to UML class diagrams, the SysML internal block diagrams are closer to UML composite structure diagrams. Further distinctions are described in other works [28].

In our example, the architecture is very important since it defines the number of devices and the interrelations. In the previous subsection we have described the behavior of valves and pumps using parametrics and state machines. However, without the topology we do not know how many parametrics and how many state machines should be instantiated nor which one communicates with which other ones.

Figure 7 shows a UML composite structure model that captures the interconnection of the devices in the AFMS. The two engines (LE and RE) and fed from the left feed tank (LFT) and the right feed tank (RFT) through pump P1/P2 (respectively P3/P4) and valve N. Valve N is a special kind of valve that is commanded manually. There two other tanks in each wing to store extra fuel: the middle tank (xMT) and the outer tank (xOT). There is also one trim tank in the tail (TT). Some transfers are done through pumps and valves (like from LFT to LE), some others do not require any pump and rather takes advantage of the wing inclination for simple gravity transfers. This is the case for transfers from FT to MT and MT to OT. In the other direction however, pumps should be used. For instance, transfers from MT to FT are done through pump TP1, valves B and C. There are several redundancies in fuel paths in case of a component failure. This model is mainly a pure UML model since UML provides native mechanisms to associate a behavior (for instance a state machine) to a class, and therefore to each instance of such a class. However, the semantic models in CCSL must be replicated for each instance. With adequate tools, like the GeMoC studio[5], this instanciation can be done automatically [7].

[5] http://www.gemoc.org - The GeMoC initiative

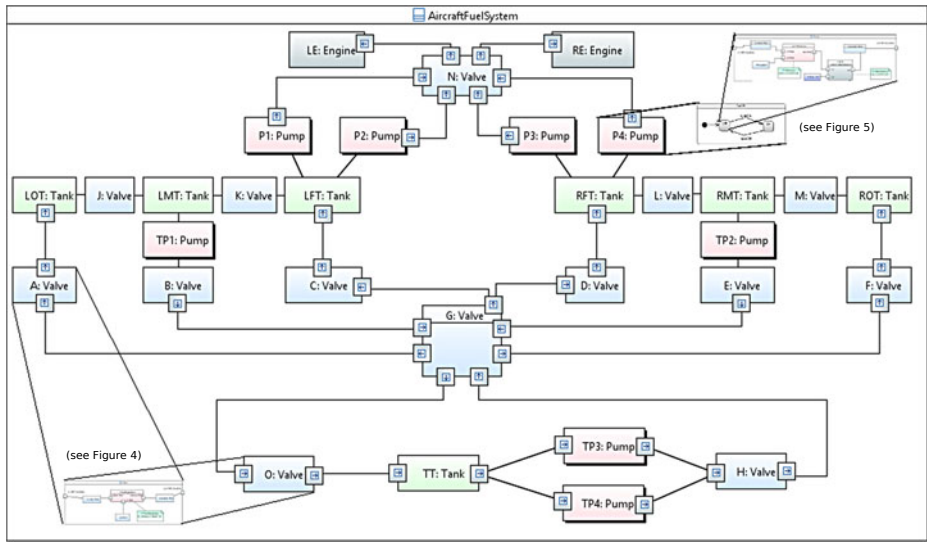


Fig. 7. Aircraft Fuel System: UML/MARTE composite structure

In this example, the platform is mainly a physical platform. MARTE does not provide any particular support for such descriptions except for providing an annotation mechanism to attach non-functional properties to elements. For instance here, the flow rates are identified. When the platform is made of digital elements (processors, memory, buses, networks) then MARTE proposes a set of stereotypes in the hardware resource modeling (HRM) and software resource modeling (SRM) subprofiles.

However, note that the arrows on the ports are not natively provided by UML. Indeed, composite structures are meant to represent static types and interconnections, not flows of information and even less flows of fuel. MARTE provides a stereotype called FlowPort to *enhance* composite structures with this capability of describing flows. Stereotype FlowPort comes with a property called direction whose values can be *in*, *out* or *inout*. This annotation is mainly used to perform simple type-checking operations. SysML also supports the same notion of FlowPort as MARTE.

In the interconnect or the execution platform is more regular, like for Network-On-Chips or multi-core processors with a regular topology like a mesh, MARTE provides a dedicated annex called Repetitive Structure Modeling (RSM). Examples of use of MARTE and CCSL together with RSM can be found elsewhere [11].

4.4 Hybrid models

In SysML constraint blocks, which are used as basic blocks for parametrics, the relations are built directly using natural language. One can then directly use traditional mathematical functions to construct the required expressions, as we

did in the previous subsection. When the functions are linear (as in the AFSM example), there is no need to rely on hybrid automata. Level crossing can be computed in a straightforward way. The logical clocks capture the event occurrences and identify the points in time where the functions must be evaluated.

In the AFSM example, however, most constraint blocks were relative to fuel flow rate. So to get the actual fuel level in tanks, one needs to integrate the flow rates over time. For instance, one can consider the simple abstraction in Eq. 5 to compute the fuel level in a tank.

$$level = \int \left(\sum_i in_i.value - \sum_j out_j.value\right) \mathrm{d}t \tag{5}$$

As the flow rate is expressed as a constant value, the integration can be done using purely discrete models.

If we consider now a temperature controller, which attempts to maintain the engine temperature between a range of values, we need to mix discrete and continuous time. Figure 8 shows such a combination. The temperature follows a different laws and the variation (derivative over time) is linear with respect to the current temperature. When the temperature reaches the maximal threshold

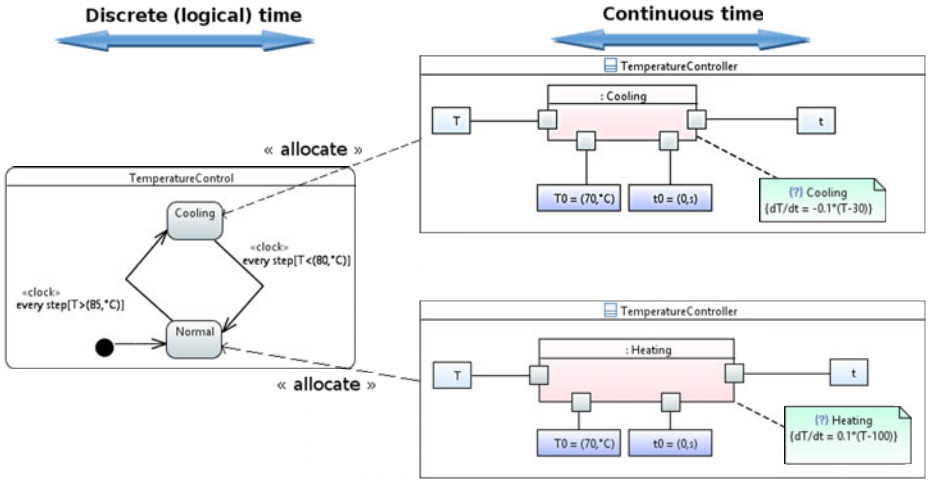


Fig. 8. Temperature controller: mixing discrete and continuous time

(*e.g.,* 85 °C), the system is overheating and the cooling system is turned on. In the cooling mode, the temperature follows Eq. 7 as a basic abstraction. Then, when the temperature goes down and reaches the minimal threshold (*e.g.,* 80 °C),

the cooling system is turned down and the temperature raises again.

$$\texttt{Normal mode:}\quad \frac{\mathrm{d}T}{\mathrm{d}t} = 0.1 \times (T - 100) \tag{6}$$

$$\texttt{Cooling mode:}\quad \frac{\mathrm{d}T}{\mathrm{d}t} = -0.1 \times (T - 30) \tag{7}$$

Note that a logical clock called step is used on the state machine. This logical step is used to drive the integration of the continuous part, using Scilab[6] and its fix-step simulation engine. The CCSL specification is fed into TimeSquare[7] [8], our analysis tool dedicated to CCSL, CCSL computes possible solutions to the constraint system. Based on these solutions it decides which clock ticks (*i.e.,* which event can occur). When the event step ticks, this drives the SciLab simulation engine which integrates the temperature controller equations depending on the current state. The resulting simulation is shown in Figure 9. This integration of

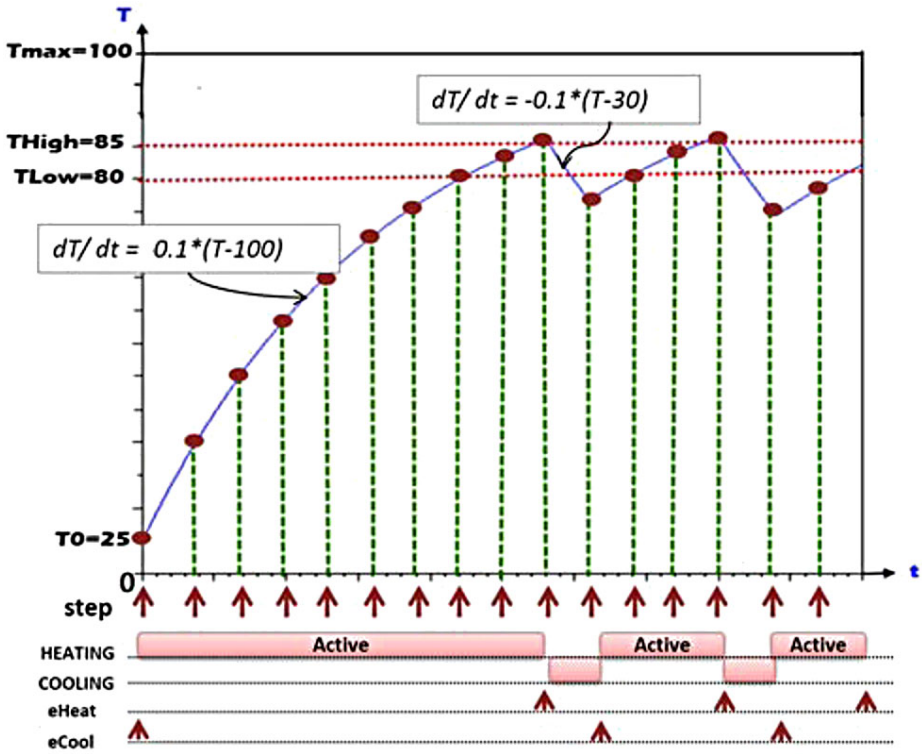


Fig. 9. Mixed simulation with TimeSquare and Scilab.

MARTE/CCSL models with SciLab is described in more details in a report [13].

[6] http://www.scilab.org
[7] http://timesquare.inria.fr

In this chapter, only the discrete aspects of MARTE clocks have been discussed. However, in MARTE clocks can be either a discrete set of ordered instants (*i.e.,* a sequence of event occurrences) or a dense set of instants. The latter characterization allows for the use of MARTE clocks in a hybrid settings. CCSL, however, as defined in Section 3.4 can only describe discrete properties. An extension of CCSL for dense clocks has initially been defined in [1] and refined in [19].

4.5 Safety critical

Because CPS are often safety-critical we need to have a precise and sound semantics for the models. The UML is described informally and has many semantic variation points. This *approximative* interpretation is very convenient for us to adapt and restrict the models to our specific needs without risking violating the usual/general understanding of UML models. In this chapter, we mainly rely on CCSL and its logical clocks to describe the events of the model. This has the main advantage to make explicit the expected semantics within the model while clocks have become first class citizens of UML MARTE models.

Apart from heterogeneous simulations as illustrated in the previous subsection, there have been a set of verification techniques and tools developed and tailored to the analysis of CCSL specifications. This subsection briefly reviews the different families of analysis techniques available for CCSL.

The first idea that comes into mind is to use CCSL as a specification of the expected behavior of the discrete control part of the system and to generate the controller based on this specification. This idea was exploited in [29] using Signal as an intermediate language. CCSL specifications were transformed into a Signal program that is then used to synthesize the controller. A similar idea is used in [27] in order to transform CCSL into a SystemC specification.

Another possible usage of CCSL is to use it to verify an existing implementation. In such an approach, a CCSL specification is transformed into an observing automata [20] used to observe some code and raise alarms whenever the specification is violated. This observer-based approach was attempted for VHDL and for the Esterel language. The main difference between those two choices is that because Esterel Studio comes with a model-checker then the observer can be used also to make an exhaustive verification of the Esterel implementation model.

Finally, if CCSL expresses the expected behavior, one solution is to analyze the CCSL specification to establish its intrinsic correctness properties [23]. An important characteristics of CCSL specifications is whether there are deadlocks or not. In CCSL, since we have polychronous specifications and the parts of a specification may be completely uncorrelated, the deadlocks may be only partial, which make them more difficult to detect. Indeed, some clocks may stop ticking as a specification feature, some others may stop because there is a contradiction between two constraints and no satisfying solution can be found. This particular aspect was studied and a general criteria was given to detect so-called bad paths in CCSL specifications. Another very important aspect is safety of CCSL specifications [22], more precisely usage of bounded memory. Because the

specification is polychronous, some parts are unrelated. If two parts execute at completely different rates, and in particular if one parts goes ultimately infinitely faster than another one, then the communication buffers between the two parts are unbounded.

5 Conclusion

In this chapter, we have presented a view for using a subset of the UML profile for MARTE to model some important aspects of cyber-physical systems. We started by emphasizing the generally accepted important characteristics of CPS. Then we have briefly introduced some parts of the MARTE specification that are important to capture some of these aspects and that are not often described in the literature. Finally the use of those constructs is illustrated mainly through a classical example of cyber-physical systems taken from the literature, *i.e.,* the fuel management system of an aircraft. While MARTE, and UML in general, provide only a language (as a set of modeling elements), we propose a way to unify and integrate the different modeling elements and diagrams through the use of logical clocks. Those logical clocks have become first-class citizens in MARTE time subprofile and are handled in our examples through the Clock Constraint Specification Language. CCSL clocks are used to describe discrete phenomena and clearly for CPS it has become very important to integrate continuous phenomena as well. MARTE clocks can be both discrete and dense so it opens a wide path to more work on defining an adequate specification language able to combine both discrete and continuous properties.

References

1. André, C.: Syntax and semantics of the Clock Constraint Specification Language (CCSL). Research Report 6925, INRIA (May 2009), http://hal.inria.fr/inria-00384077/
2. André, C., DeAntoni, J., Mallet, F., de Simone, R.: The Time Model of Logical Clocks available in the OMG MARTE profile, chap. 7, pp. 201–227. Springer Science+Business Media, LLC 2010 (July 2010), http://hal.inria.fr/inria-00495664
3. André, C., Mallet, F., de Simone, R.: Modeling time(s). In: 10th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS '07). pp. 559–573. No. 4735 in LNCS, ACM-IEEE, Springer, Nashville, TN, USA (September 2007)
4. Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P., de Simone, R.: The synchronous languages 12 years later. Proceedings of the IEEE 91(1), 64–83 (Jan 2003)
5. Berry, G., Cosserat, L.: The ESTEREL synchronous programming language and its mathematical semantics. In: Seminar on Concurrency. Lecture Notes in Computer Science, vol. 197, pp. 389–448. Springer (1984), http://dx.doi.org/10.1007/3-540-15670-4_19
6. Boulanger, F., Dogui, A., Hardebolle, C., Jacquet, C., Marcadet, D., Prodan, I.: Semantic adaptation using CCSL clock constraints. ECEASST 50 (2011), http://journal.ub.tu-berlin.de/eceasst/article/view/731

7. Combemale, B., DeAntoni, J., Larsen, M.V., Mallet, F., Barais, O., Baudry, B., France, R.B.: Reifying concurrency for executable metamodeling. In: 6th Int. Conf. on Software Language Engineering - SLE 2013. Lecture Notes in Computer Science, vol. 8225, pp. 365–384. Springer (October 2013)
8. Deantoni, J., Mallet, F.: Timesquare: Treat your models with logical time. In: Furia, C.A., Nanz, S. (eds.) TOOLS (50). Lecture Notes in Computer Science, vol. 7304, pp. 34–41. Springer (2012)
9. Derler, P., Lee, E.A., Sangiovanni-Vincentelli, A.L.: Modeling cyber-physical systems. Proceedings of the IEEE 100(1), 13–28 (2012), http://dx.doi.org/10.1109/JPROC.2011.2160929
10. Friedenthal, S., Moore, A., Steiner, R.: A Practical Guide to SysML: The Systems Modeling Language. MK/OMG (2014)
11. Glitia, C., DeAntoni, J., Mallet, F., Millo, J., Boulet, P., Gamatié, A.: Progressive and explicit refinement of scheduling for multidimensional data-flow applications using UML MARTE. Design Autom. for Emb. Sys. 16(2), 137–169 (2012), http://dx.doi.org/10.1007/s10617-012-9093-y
12. Jimenez, J.F., Giron-Sierra, J.M., Insaurralde, C., Seminario, M.: A simulation of aircraft fuel management system. Simulation Modelling Practice and Theory 15(5), 544–564 (2007), http://www.sciencedirect.com/science/article/pii/S1569190X07000160
13. Khecharem, A., Gomez, C., DeAntoni, J., Mallet, F., de Simone, R.: Execution of heterogeneous models for thermal analysis with a multi-view approach. In: Forum on specification and Design Languages (FDL'14) (2014)
14. Koch, T., Holtmann, J., DeAntoni, J.: Generating EAST-ADL event chains from scenario-based requirements specifications. In: Software Architecture - 8th European Conference, ECSA 2014, Vienna, Austria, August 25-29, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8627, pp. 146–153. Springer (2014)
15. Lamport, L.: The parallel execution of do loops. Communications of ACM 17(2), 83–93 (1974)
16. Lee, E.A., Sangiovanni-Vincentelli, A.L.: A framework for comparing models of computation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17(12), 1217–1229 (December 1998)
17. Lee, E.A.: Cyber physical systems: Design challenges. In: 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2008). pp. 363–369. IEEE Computer Society (May 2008), http://dx.doi.org/10.1109/ISORC.2008.25
18. Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems - A Cyber-Physical Systems Approach. LeeSeshia.org (2014), iSBN 978-0-557-70857-4
19. Liu, J., Liu, Z., He, J., Mallet, F., Ding, Z.: Hybrid marte statecharts. Frontiers of Computer Science 7(1), 95–108 (2013)
20. Mallet, F.: Logical Time @ Work for the Modeling and Analysis of Embedded Systems. LAMBERT Academic Publishing (January 2011), iSBN: 978-3-8433-9388-1.
21. Mallet, F., André, C., de Simone, R.: CCSL: specifying clock constraints with UML/Marte. Innovations in Systems and Software Engineering 4(3), 309–314 (2008)
22. Mallet, F., Millo, J.V., de Simone, R.: Safe CCSL specifications and marked graphs. In: 11th ACM/IEEE Int. Conf. on Formal Methods and Models for Codesign. pp. 157–166. IEEE (2013)
23. Mallet, F., de Simone, R.: Correctness issues on MARTE/CCSL constraints. Science of Computer Programming (2015), DOI: 10.1016/j.scico.2015.03.001

24. OMG: UML Profile for Schedulability, Performance, and Time Specification, v1.1. Object Management Group (January 2005), formal/05-01-02
25. OMG: UML Profile for MARTE, v1.1. Object Management Group (June 2011), formal/2011-06-02
26. OMG: UML Superstructure, v2.4.1. Object Management Group (May 2012), formal/12-05-07
27. Peters, J., Wille, R., Drechsler, R.: Generating SystemC implementations for clock constraints specified in UML/MARTE CCSL. In: 2014 19th International Conference on Engineering of Complex Computer Systems. pp. 116–125. IEEE (2014), http://dx.doi.org/10.1109/ICECCS.2014.24
28. Selic, B., Gerard, S.: Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE. Elsevier (2013)
29. Yu, H., Talpin, J., Besnard, L., Gautier, T., Marchand, H., Guernic, P.L.: Polychronous controller synthesis from MARTE CCSL timing specifications. In: 9th IEEE/ACM International Conference on Formal Methods and Models for Codesign, MEMOCODE 2011. pp. 21–30. IEEE (2011), http://dx.doi.org/10.1109/MEMCOD.2011.5970507

An Introduction to Hybrid Automata, Numerical Simulation and Reachability Analysis

Goran Frehse

Verimag, Université Joseph Fourier - Grenoble 1,
2 avenue de Vignate, Centre Equation,
38610 Giéres, France,
frehse@imag.fr

Abstract. Hybrid automata combine finite state models with continuous variables that are governed by differential equations. Hybrid automata are used to model systems in a wide range of domains such as automotive control, robotics, electronic circuits, systems biology, and health care. Numerical simulation approximates the evolution of the variables with a sequence of points in discretized time. This highly scalable technique is widely used in engineering and design, but it is difficult to simulate all representative behaviors of a system. To ensure that no critical behaviors are missed, reachability analysis aims at accurately and quickly computing a cover of the states of the system that are reachable from a given set of initial states. Reachability can be used to formally show safety and bounded liveness properties. This chapter outlines the major concepts and discusses advantages and shortcomings of the different techniques.

1 Introduction

Hybrid automata are a modeling formalism that combines discrete states with continuously evolving, real-valued variables. The discrete states and the possible transitions from one state to another are described with a finite state-transition system. A change in discrete state can update the continuous variables and modify the set of differential equations that describes how variables evolve with time. Hybrid automata are non-deterministic, which means that different futures may be available from any given state. Rates of change or variable updates can be described by providing bounds instead of fixed numbers. Incomplete knowledge about initial conditions, perturbations, parameters, etc. can easily be captured this way. Hybrid automata capture a rich variety of behaviors, and are used in a wide range of domains such as automotive control, robotics, electronic circuits, systems biology, and health care. The hybrid automaton model is well-suited for formal analysis, in the sense that sets of behaviors are readily described by mathematical equations.

In the next section, we present the hybrid automaton formalism with an example and give a formal definition of the model and its semantics. Behaviors of hybrid automata are computed in practice using numerical simulation, which

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_3,

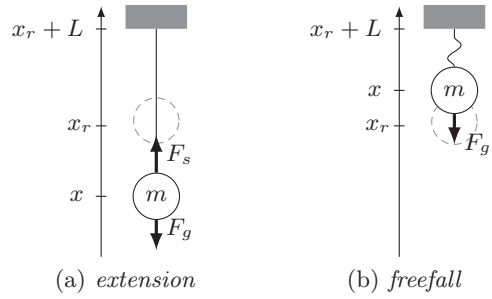


Fig. 1. A ball suspended from a ceiling by an elastic string

is presented in Sect. 3. We discuss major aspects such as approximation errors, stability, stiffness and Zeno behavior. Simulation techniques have recently been extended to provide formal guarantees for certain classes of systems. Reachability techniques are discussed in Sect. 4. Different algorithms and data structures are suitable for reachability, depending mainly on the type of dynamics: piecewise constant, affine, and nonlinear. We present an overview of the main techniques, with particular attention to scalability.

Related work is indicated throughout the text, but given the rich literature on the topic this introduction is far from exhaustive. For further reading, see [43, 4, 50].

2 Hybrid Automata

We introduce the hybrid automaton formalism with the following example. The technical details are fleshed out in the sections that follow. Consider a ball that is suspended from a ceiling by a string, as shown in Fig. 1. We will construct a simple model that only takes into account the vertical movement of the ball.

Equations of motion. Let x be the variable representing the position of the ball measured in the upwards direction, and let x_r be the position of the ball when the string is at its natural length L. The ball has mass m and is at all times subject to a gravitational force

$$F_g = -mg.$$

The string is elastic, so when the string is extended beyond its natural length L, i.e., $x \leq x_r$, it acts as a damped spring pulling the ball upwards with a force

$$F_s = -k(x - x_r) - d\dot{x},$$

where k is the spring constant and d is a damping factor that models the loss of energy through deformation of the string. When $x \geq x_r$, the string is slack and only the gravitational force is acting on the ball.

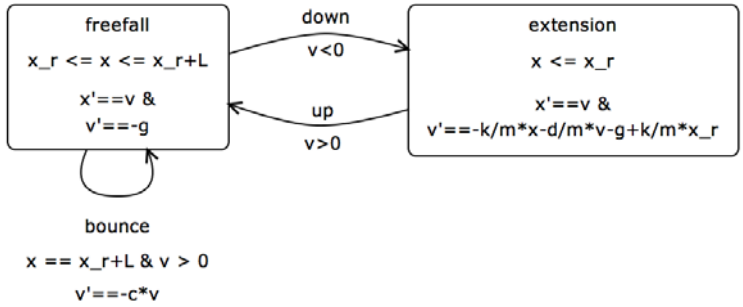


Fig. 2. A hybrid automaton model of a ball on string, constructed in the tool SpaceEx. In the flow equations, x' denotes the derivative $\dot{x}$

The case distinction for the string force leads to two *discrete states* of the system, which we shall call *freefall* and *extension*. Each state is associated with a different set of differential equations and active for different values of x. The system is in *freefall* when $x \geq x_r$ and we have

$$m\ddot{x} = F_g = -mg.$$

The system is in *extension* when $x \leq x_r$, with

$$m\ddot{x} = F_g + F_s = -mg + kx_r - kx - d\dot{x}.$$

A *discrete event*, also called *transition*, takes place when the ball hits the ceiling, i.e., when $x = x_r + L$. We assume that the collision is elastic, so the velocity of the ball changes its sign and is diminished by a factor $c \in [0, 1]$ that reflects the loss of energy during the collision. Denoting the value of x after the collision with x', we get

$$\dot{x}' = -c\dot{x}.$$

Hybrid automaton model. Figure 2 shows a hybrid automaton that models the system. Each discrete state, also called *location*, is labelled with the differential equations that govern the variables. Typically, this is a *first-order ordinary differential equation system* (ODE) of the form

$$\dot{x} = f(x).$$

We refer to the ODE as the *dynamics* of the system. To bring the laws of motion to ODE form we introduce an auxiliary variable to replace $\ddot{x}$. Let v be the velocity of the ball, i.e., $\dot{x} = v$. Then $\ddot{x} = \dot{v}$ and we can substitute $\ddot{x}$ with $\dot{v}$. Solving the equations so that all derivatives are on the left hand side leads to the dynamics shown in Fig. 2.

In addition to the ODE, the *locations* are also labelled with a staying condition, called *invariant*. It characterizes any non-differential conditions that must

hold whenever the system is in the location, such as boundary conditions or algebraic equations. In location *extension*, the invariant is $x \leq x_r$ since this is when the string extends beyond its natural length. In location *freefall*, the invariant is given by two constraints: Since the ball needs to be above where it extends the string, we have $x \geq x_r$. Since the ball cannot go further up than the ceiling, which is located at $x = x_r + L$, we have $x \leq x_r + L$.

When the ball does hit the ceiling with positive velocity, it bounces off and its velocity changes sign. This is modeled by a transition with label *bounce*. It goes from location *freefall* to *freefall*, since the ball remains subject to the same differential and boundary conditions. The transition is labeled with a *guard condition*

$$x = x_r + L \wedge v > 0,$$

which must be satisfied for a state to be able to take the jump. The target state after the jump is expressed by the *assignment*

$$v' = -cv.$$

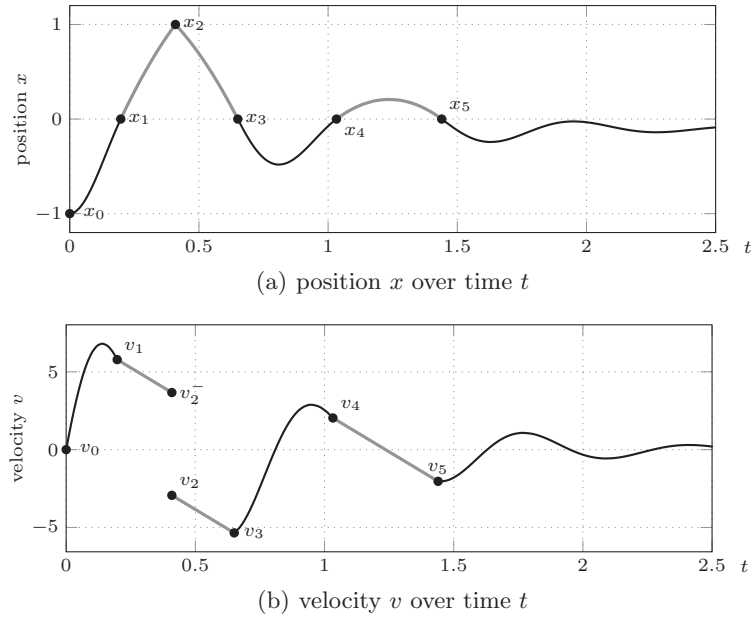
Variables not mentioned in the assignment are considered unchanged.

Transitions *up* and *down* model the switching between *freefall* and *extension*. A guard condition on the velocity of the ball is included to avoid unnecessary switches when the ball is on the switching border $x = x_r$ and $v = 0$. It also helps verification tools that are based on overapproximations, since it excludes unnecessary switches.

Run semantics. The behavior of a hybrid automaton is described by the evolution of its continuous variables and its location, i.e., its discrete state. Note that in general there need not be any correspondence between the values of the variables, called the *continuous state*, and the discrete state. In our example, the ball can be in position $x = x_r$ in location *freefall* and in location *extension*, independently of the value of v. The combination of the continuous state and the discrete state is called the *state* of the hybrid automaton. The state, by definition, is what determines the set of possible futures of the system. The evolution of the state over time is called a *run* of the system. In the following, we choose the parameter values $L = 1$, $x_r = 0$, $k = 100$, $d = 4$, $c = 0.8$, $m = 1$, and $g = 10$.

Figure 3 shows the evolution of the variables x, v over time, starting from $x_0 = -1, v_0 = 0$, and location *extension*. We follow this evolution location by location:

1. The ball is pulled upwards by the string until $x_1 = x_r$, at which point the automaton can no longer spend time in location *extension* since otherwise the invariant $x \leq x_r$ would be violated. Since v is positive, the transition *up* to location *freefall* is enabled, and taking it is the only possible future in which time can progress.
2. From (x_1, v_1), the ball continues its upward motion in a parabola until it reaches the ceiling at $x_2 = x_r + L$, where the invariant $x \leq x_r + L$ ensures that the only possible future is to take the transition *bounce*. The transition instantaneously changes the velocity from v_2^- to v_2, see Fig. 3(b).



(a) position x over time t

(b) velocity v over time t

Fig. 3. A run of the ball on string model consists of a sequence of trajectories. Trajectories in location *extension* are shown in black, trajectories in *freefall* in gray

3. From (x_2, v_2), the ball falls towards the ground. When $x_3 = x_r$, the transition *down*, which is enabled since $v < 0$, leads to location *extension*.
4. The ball is pulled upwards until it reaches $x = x_r$, where the transition *up* leads to location *freefall*.
5. In location *freefall* at (x_4, v_4), the ball follows a parabola until $x_5 = x_r$, where the transition *down* takes it back to location *extension*.
6. The trajectory from (x_5, v_5) remains in the invariant for all time, and none of the transitions are enabled. The ball converges towards its equilibrium point.

Each of the steps of the above sequence corresponds to one location and a differentiable function of time representing the evolution of the continuous variables. Going from one step to the next is associated with the label of corresponding transition. Such a sequence is called the *run* of a hybrid automaton, and the set of runs defines the semantics (the set of behaviors) of the system.

2.1 Preliminaries

Hybrid automata describe the evolution of a set of real-valued variables over time. We now introduce the notation for describing sets of values for these variables.

Variables. Let $X = \{x_1, \ldots, x_n\}$ be a finite set of identifiers we call *variables*. Attributing a real value to each variable we get a *valuation* over X, written as $\mathbf{x} \in \mathbb{R}^X$ or $\mathbf{x} : X \to \mathbb{R}$. We will use the primed variables $X' = \{x'_1, \ldots, x'_n\}$ to denote successor values and the dotted variables $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ to denote the derivatives of the variables with respect to time. Given a set of variables $Y \subseteq X$, the *projection* $\mathbf{y} = \mathbf{x} \downarrow_Y$ is a valuation over Y that maps each variable in Y to the same value that it has in x. We may simply use a vector $\mathbf{x} \in \mathbb{R}^n$ if it is clear from the context which index of the vector corresponds to which variable. We denote the i-th element of a vector $\mathbf{x}$ as $\mathbf{x}_i$ or $\mathbf{x}(i)$ if the latter is ambiguous. In the following, we use $\mathbb{R}^n$ instead of $\mathbb{R}^X$ except when the correspondance between indices and variables is not obvious, e.g., when valuations over different sets of variables are involved.

Predicates. A *predicate* over X is an expression that, given a valuation $\mathbf{x}$ over X, can be evaluated to either true or false. A *linear constraint* is a predicate

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq b,$$

where $a_1, \ldots a_n$ and b are real-valued constants, and whose sign may be strict ($<$) or nonstrict ($\leq$). A linear constraint is written in vector notation as

$$\mathbf{a}^{\mathsf{T}}\mathbf{x} \leq b,$$

with coefficient vector $\mathbf{a} \in \mathbb{R}^n$ and inhomogeneous coefficient $b \in \mathbb{R}$. A predicate over X defines a continuous set, which is the subset of $\mathbb{R}^X$ on which the predicate evaluates to true.

Polyhedra. A conjunction of finitely many linear constraints defines an $\mathcal{H}$-*polyhedron*, or polyhedron in *constraint form*,

$$\mathcal{P} = \Big\{ \mathbf{x} \;\Big|\; \bigwedge\nolimits_{i=1}^{m} \mathbf{a}_i^{\mathsf{T}}\mathbf{x} \bowtie_i b_i \Big\}, \text{ with } \bowtie_i \in \{<, \leq\},$$

with *facet normals* $\mathbf{a}_i \in \mathbb{R}^n$ and *inhomogeneous coefficients* $b_i \in \mathbb{R}$. In vector-matrix notation, an $\mathcal{H}$-*polyhedron* can be written as

$$\mathcal{P} = \Big\{ \mathbf{x} \;\Big|\; A\mathbf{x} \bowtie \mathbf{b} \Big\}, \text{ with } A = \begin{pmatrix} \mathbf{a}_1^{\mathsf{T}} \\ \vdots \\ \mathbf{a}_m^{\mathsf{T}} \end{pmatrix}, \bowtie = \begin{pmatrix} \bowtie_1 \\ \vdots \\ \bowtie_m \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}.$$

An $\mathcal{H}$-polyhedron is a *closed* set if it can be defined using only nonstrict constraints. A bounded polyhedron is called a *polytope*. Note that the constraints defining $\mathcal{P}$ are not necessarily unique. A closed polyhedron $\mathcal{P}$ can be represented in *generator form* by a pair (V, R), where $V \subseteq \mathbb{R}^n$ is a finite set of *vertices*, and $R \subseteq \mathbb{R}^n$ is a finite set of *rays*. They define the $\mathcal{V}$-*polyhedron*

$$P = \Big\{ \sum_{\mathbf{v}_i \in V} \lambda_i \cdot \mathbf{v}_i + \sum_{\mathbf{r}_j \in R} \mu_j \cdot \mathbf{r}_j \;\Big|\; \lambda_i \geq 0, \mu_j \geq 0, \sum_i \lambda_i = 1 \Big\},$$

which consists of the convex hull of the vertices, extended towards infinity along the directions of the rays. The generator representation can be extended with *closure points* to deal with non-closed polyhedra [10]. An $\mathcal{H}$-polyhedron can be converted to a $\mathcal{V}$-polyhedron and vice versa, but this may increase the complexity exponentially.

2.2 Definition and Semantics

We now give a formal definition of a hybrid automaton and its run semantics.

Definition 1 (Hybrid automaton). *[5, 35] A* hybrid automaton

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

consists of

- *a finite set of* locations $\mathsf{Loc} = \{\ell_1, \ldots, \ell_m\}$ *represents the discrete states,*
- *a finite set of* synchronization labels Lab*, also called its* alphabet*, which can be used to coordinate state changes between several automata,*
- *a finite set of edges* $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Lab} \times \mathsf{Loc}$*, also called* transitions*, which determines which discrete state changes are possible using which label,*
- *a finite set of* variables $X = \{x_1, \ldots, x_n\}$*, partitioned into uncontrolled variables U and controlled variables Y; a* state *of H consists of a location ℓ and a value for each of the variables, and is denoted by $s = (\ell, \mathbf{x})$;*
- *a set of states* Inv *called* invariant *or* staying condition*; it restricts for each location the values that x can possibly take and so determines how long the system can remain in the location;*
- *a set of* initial *states* $\mathsf{Init} \subseteq \mathsf{Inv}$*; every behavior of H must start in one of the initial states;*
- *a* flow relation Flow*, where* $\mathsf{Flow}(\ell) \subseteq \mathbb{R}^{\dot{X}} \times \mathbb{R}^X$ *gives for each state $(\ell, \mathbf{x})$ the set of possible derivatives $\dot{\mathbf{x}}$, e.g., using a differential equation such as*

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

  Given a location ℓ, a trajectory *of duration $\delta \geq 0$ is a continuously differentiable function $\xi : [0, \delta] \to \mathbb{R}^X$ such that for all $t \in [0, \delta]$, $(\dot{\xi}(t), \xi(t)) \in \mathsf{Flow}(\ell)$. The trajectory* satisfies the invariant *if for all $t \in [0, \delta]$, $\xi(t) \in \mathsf{Inv}(\ell)$.*
- *a* jump relation Jump*, where* $\mathsf{Jump}(e) \subseteq \mathbb{R}^X \times \mathbb{R}^{X'}$ *defines for each transition $e \in \mathsf{Edg}$ the set of possible successors $\mathbf{x}'$ of $\mathbf{x}$; jump relations are typically given by a* guard *set $\mathcal{G} \subseteq \mathbb{R}^X$ and an assignment (or reset) $\mathbf{x}' = r(\mathbf{x})$ as*

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

We define the behavior of a hybrid automaton with a *run*: starting from one of the initial states, the state evolves according to the differential equations whilst time passes, and according to the jump relations when taking an (instantaneous) transition. Special events, which we call *uncontrolled assignments*, model an environment that can make arbitrary changes to the uncontrolled variables.

Definition 2 (Run semantics). *A* run *of H is a sequence*

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

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

1. *The first state is an initial state of the automaton, i.e., $(\ell_0, \mathbf{x}_0) \in \mathsf{Init}$.*
2. Trajectories: *In location ℓ_i, ξ_i is a trajectory of duration δ_i that satisfies the invariant.*
3. Jumps: *If $\alpha_i \in \mathsf{Lab}$, there exists a transition $(\ell_i, \alpha_i, \ell_{i+1}) \in \mathsf{Edg}$ with jump relation $\mathsf{Jump}(e)$ such that $(\xi_i(\delta_i), \mathbf{x}_{i+1}) \in \mathsf{Jump}(e)$ and $\mathbf{x}_{i+1} \in \mathsf{Inv}(\ell_{i+1})$.*
4. Uncontrolled assignments: *If $\alpha_i = \tau$, then $\ell_i = \ell_{i+1}$ and $\xi_i(\delta_i) \downarrow_Y = \mathbf{x}_{i+1} \downarrow_Y$. This represents arbitrary assignments that the environment might perform on the uncontrolled variables $U = X \setminus Y$.*

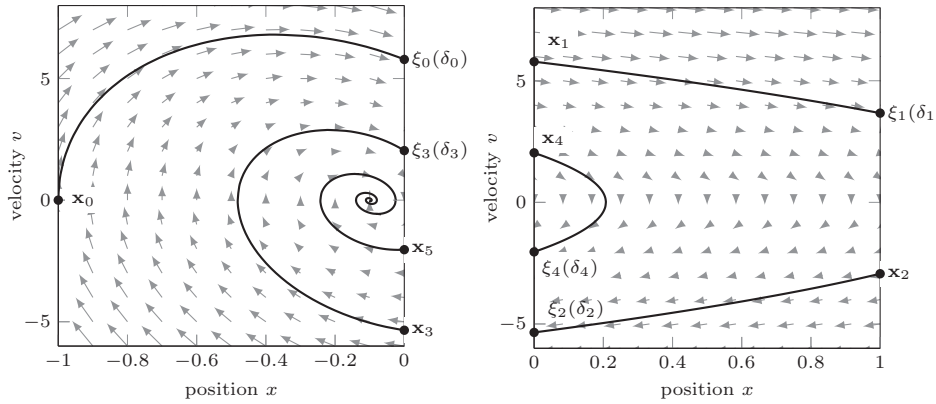
A state $(\ell, \mathbf{x})$ is reachable *if there exists a run with $(\ell_i, \mathbf{x}_i) = (\ell, \mathbf{x})$ for some i.*

Note that the strict alternation of trajectories and jumps in Def. 2 is of no particular importance. Two consecutive jumps can be represented by inserting a trajectory with duration zero (which always exists), and two consecutive trajectories can be represented by inserting an uncontrolled assignment jump that does not modify the variables.

Example 1 (Ball/String). The Ball on String example is modeled by the hybrid automaton shown in Fig. 2. All elements except the labels and the initial states are visible in the figure. Since we do not expect x or v to be modified by the environment, we consider both variables to be controlled (as is usually the case for variables whose derivative is given). The set of synchronization labels is $\mathsf{Lab} = \{\mathit{bounce}, \mathit{up}, \mathit{down}\}$. For now, we assume $\mathsf{Init} = \{\mathit{extension}\} \times \{x = -1, v = 0\}$.

In location *extension*, the dynamics are those of a damped oscillator. The ODE system is linear in the variables x and v, so its solution is a combination of exponential, sine and cosine functions of time. In location *freefall*, the dynamics are also linear, but of a particularly simple kind. The derivative of v is constant, $\dot{v} = -g$, so that v evolves in a straight line and x in a parabola. Figure 4 shows the trajectories of the same run as in Fig. 3, but in the state space (also called phase space), which allows one to graph over an infinite time horizon.

May and Must semantics. In Def. 2, transitions may be taken when they are enabled, but there is no obligation to do so – the system may remain in a location as long as the invariant is satisfied. These so-called *may* semantics allow one to include nondeterminism about when a transition will be taken, e.g., when it is not clear how fast a discrete controller will react to a stimulus. In the Ball/String example, this could be used if the length of the string (position of the ceiling) is not exactly known. In contrast, *must* or *ASAP* semantics dictate that the transition is taken as soon as possible. These semantics are used by simulators such as Simulink [45], Dymola [13], MapleSim [44], etc., since they require deterministic models. Some verification tools, like HyTech [32] and PHAVer [23], allow one to include both types of transitions.



(a) location *extension* (b) location *freefall*

Fig. 4. A sample run of the ball/string example, with trajectories $\xi_0, \ldots, \xi_5$. The initial state $\mathbf{x}_0$ corresponds to the variable values $x = -1, v = 0$ in location *extension*. The arrows indicate the direction and magnitude of the derivative

3 Numerical Simulation

The approximate computation of a run of a hybrid automaton is called *numerical simulation*. This technique is widely applied in industrial practice and has the distinct advantage that precise and highly scalable algorithms are available. In this section, we briefly introduce the basic principles and discuss the major difficulties. Numerical simulation is related to formal verification in several ways:

- The reachability methods of Sect. 4 are built on the same principles, and similar difficulties arise, such as numerics, stability, stiffness and Zeno.
- Set-based extensions of numerical simulation algorithms are available in both approximate [12] and conservative forms [11].
- Verification-by-simulation techniques extend simulation runs to their neighborhoods such that coverage can be formally guaranteed.
- Numerical simulation is typically the validation technique used for *constructing* models. Knowing its strengths and limitations may help to avoid modeling errors and oversights.
- In *testing*, a large number of simulation runs are computed to sample as much of the state space as possible. Various guiding schemes choose these runs intelligently, aiming to achieve coverage similar (but not equal) to formal methods.

Computing a simulation run for a given maximum number of jumps and a given time horizon consists of the following steps:

1. Choose a single state from the set of initial states.
2. *Continuous step:* Compute a trajectory by solving the ODE of the location, stopping when the invariant is violated or the time horizon has been reached.

3. *Discrete step:* Detect the transitions that are enabled along the trajectory. If available, choose one of the transitions, and a time point when it is enabled. Compute one of the successor states of the jump.
4. If the maximum number of jumps or the time horizon has been reached, stop. Otherwise, continue with step 2.

In the following section, we present an overview on solving ODEs, which is the main ingredient for the continuous step. In Sect. 3.2, we discuss how to detect state-based events, such as violating the invariant or entering a guard, which is the main ingredient for the discrete step. In actual implementations, both techniques are intertwined to generate a sequence of states on the run with as little computational overhead as possible.

3.1 Solving ODEs

We consider an *ordinary differential equation* (ODE) of the form

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

where $\dot{\mathbf{x}} = d\mathbf{x}/dt$ represents the rate of change of $\mathbf{x}$ with respect to time $t \in \mathbb{R}^{\geq 0}$. Solving the ODE for a given initial state $\mathbf{x}_0$ and time horizon T means to find a function $\xi(t)$ such that $\xi(0) = \mathbf{x}_0$ and $\dot{\xi}(t) = f(\xi(t))$ for all $t \in [0, T]$. This is referred to as an *initial value problem*. We are interested in solving the ODE numerically, which means computing a sequence of states $\mathbf{x}_0, \ldots, \mathbf{x}_N$ that approximates $\xi(t)$ at time points $t_0, \ldots, t_N$. The choice of time points is either fixed with a given time step h, i.e.,

$$t_{i+1} = t_i + h,$$

or h is adapted on the fly in order to achieve a given error bound. Standard ODE solvers do not guarantee actual error bounds at the computed points, since such bounds are frequently overly conservative in practice. Instead, it is guaranteed that, at least for certain classes of problems, the approximation error vanishes as $h \to 0$. While in engineering practice the linear interpolation between these points is typically considered a good approximation of $\xi(t)$, there are no a-priori bounds on the distance between the linear interpolation and the actual solution. We now provide a brief overview over the main methods for solving ODEs, a readable introduction can be found, e.g., in [14]. In the following, we consider ODEs of a single variable x. The extension to a vector $\mathbf{x}$ is straightforward.

Euler's Method The simplest integration method is called *Euler's method*. It computes the sequence

$$x_{i+1} = x_i + f(x_i)h. \tag{1}$$

An estimation for the *local error*, i.e., the error made at each step, can be obtained by comparing the sequence to a Taylor series expansion around x_i,

$$x_{i+1} = x_i + \dot{x}_i h + \frac{\ddot{x}_i}{2!}h^2 + \ldots + \frac{x_i^{(n-1)}}{n!}h^n + \mathcal{O}(h^{n+1}), \tag{2}$$

where $\mathcal{O}(h^{n+1})$ specifies that the truncation error is proportional to h^{n+1} if h is chosen small enough. Substituting $\dot{x}_i = f(x_i)$, we can see that the first two terms of the Taylor expansion are identical to Euler's sequence. The local (one-step) approximation error is therefore $\varepsilon_a = \mathcal{O}(h^2)$.

The *global error* is the sum of the local errors at each step. Note that local errors may cancel each other out. The estimation of the global error is more complex than for the local error, but it can be shown for Euler's method that it is $\mathcal{O}(h)$. It is therefore called *first-order* method. In principle, this means that any desired accuracy can be achieved by choosing h small enough. However, we must also take the numerical roundoff error into account, which is $\mathcal{O}(1/h)$. If the time steps are too small, the roundoff error will surpass the approximation error and the accuracy will decrease. This limitation is common to all ODE solvers, and motivates the search for integration methods with a smaller error for the *same number* of function evaluations.

Example 2 (Ball/String). Figure 5 shows approximations of a trajectory in location *extension*, starting from $x = -1, v = 0$. Euler's method was applied for time steps $h = 0.05, 0.025, 0.0125, 0.00625$. For $h = 0.05$, the Euler approximation diverges towards infinity. With decreasing time steps, the approximation converges towards the exact solution, shown in black. For $h = 0.00625$, the global error at the end of the trajectory amounts to 175% of the exact value. The state space view in Fig. 5(b) superimposes the trajectory approximation with a quiver plot of the derivative. At each x_i in the sequence, Euler's method applies the local derivative $f(x_i)$ for h time units, i.e., it follows $f(x_i)$ along a straight line.

Stability and Implicit Methods If the time step is too large, the global error of Euler's method may go quickly to infinity. Consider the linear ODE

$$\dot{x} = ax, \tag{3}$$

which converges to zero for $a < 0$. Euler's method computes the sequence

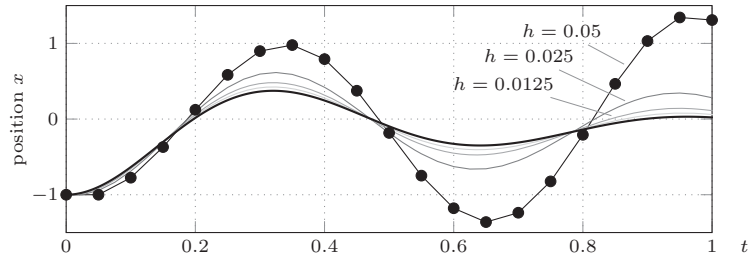
$$x_{i+1} = x_i + f(x_i)h = x_i + ax_ih = (1 + ah)x_i.$$

This sequence converges to zero iff $|1 + ah| < 1$. If $h > -2/a$, then $|x_i| \to \infty$ as $i \to \infty$. Because Euler's method converges only under certain conditions, it is called *conditionally stable*. An *unconditionally stable* method is the *backwards Euler* method, which computes the sequence

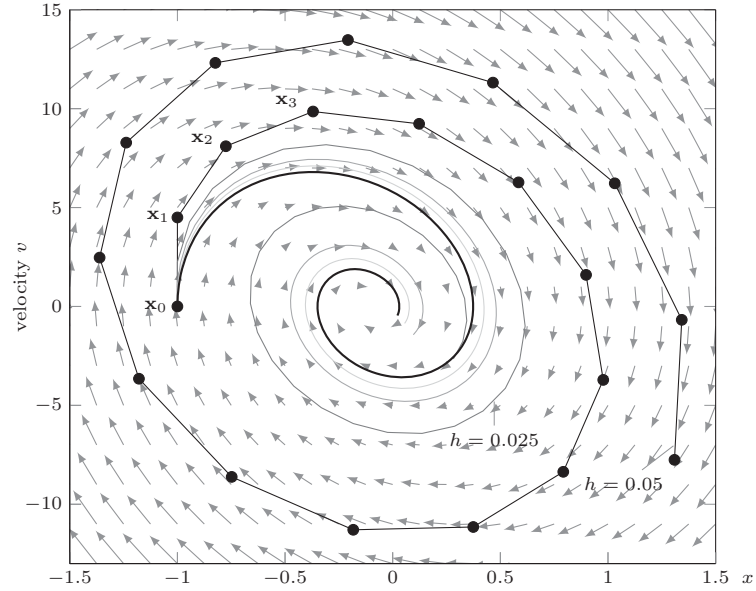
$$x_{i+1} = x_i + f(x_{i+1})h. \tag{4}$$

The backwards Euler method is an *implicit method*, since the unknown value x_{i+1} figures on both sides of the equation. It must be computed iteratively, e.g., using root-finding techniques. Like other implicit methods, the backwards Euler method thus requires more function evaluations than explicit methods. For the linear ODE (3), the backwards Euler sequence can be rearranged to

$$x_{i+1} = \frac{1}{1 - ah}x_i.$$



(a) position x in the time domain



(b) state space, with arrows indicating the derivative

Fig. 5. A trajectory of the ball/string example, approximated with Euler's method for varying time steps. The exact solution is shown in solid black

Since $\frac{1}{1-ah} < 1$ for all $a < 0$ and $h > 0$, the backwards Euler method converges whenever the ODE does, independently of the step size. It is therefore called *unconditionally stable*.

Runge-Kutta Methods Runge-Kutta (RK) methods are a family of higher-order integration methods that use intermediate evaluations of $f(x)$ to improve the precision. Explicit Runge-Kutta methods compute the sequence

$$x_{i+1} = x_i + \phi(x_i, h)h, \tag{5}$$

where the *increment function* $\phi(x_i, h)$ can be interpreted as a representative slope over the time interval. The increment function is given as

$$\phi(x_i, h) = a_1 k_1 + a_2 k_2 + \cdots + a_n k_n, \tag{6}$$

where the a_i are constants and the k_i are obtained by evaluating the ODE at intermediate states,

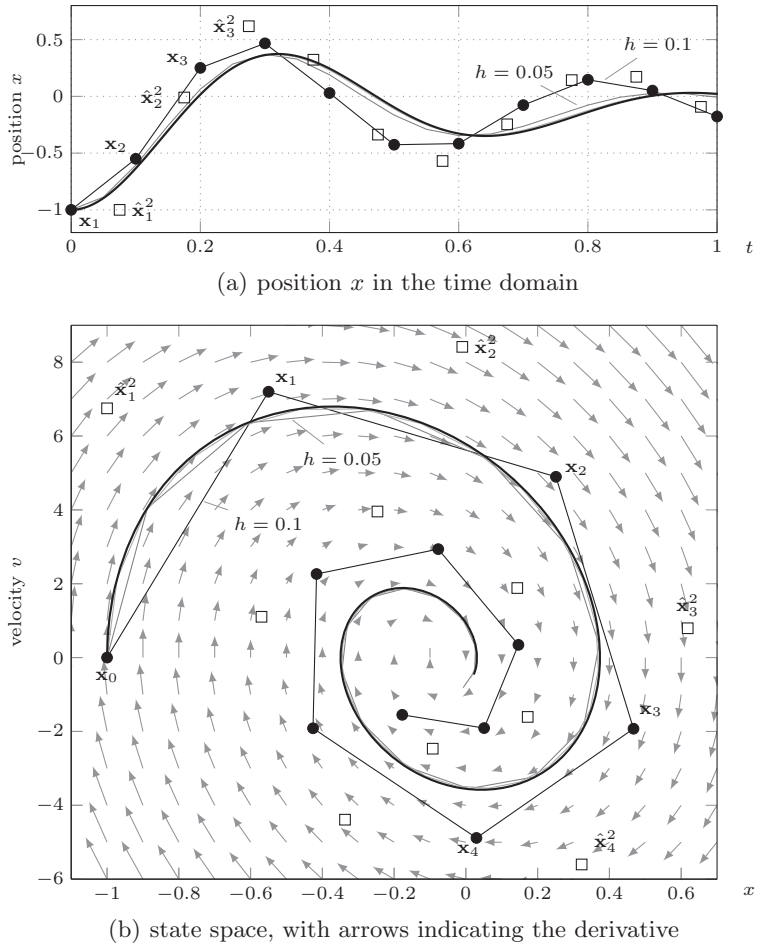
$$\begin{aligned}
&k_1 = f(\hat{x}_i^1), \;\; \hat{x}_i^1 = x_i, \\
&k_2 = f(\hat{x}_i^2), \;\; \hat{x}_i^2 = x_i + q_{11} k_1 h, \\
&k_3 = f(\hat{x}_i^3), \;\; \hat{x}_i^3 = x_i + q_{21} k_1 h + q_{21} k_2 h, \\
&\quad \vdots \qquad\qquad \vdots \\
&k_n = f(\hat{x}_i^n), \;\; \hat{x}_i^n = x_i + q_{(n-1)1} k_1 h + q_{(n-1)2} k_2 h + \cdots + q_{(n-1)(n-1)} k_{n-1} h,
\end{aligned} \tag{7}$$

where the q_{ij} are constants. Note that k_1 is used to derive the intermediate state $\hat{x}_i^2$ that leads to k_2, etc. The parameters a_i,q_{ij} are derived by equating the terms in (5) to those of a Taylor series expansion, so the method has zero error if the solution is a nth order polynomial. There are more parameters than terms, so the remaining parameters can be chosen to optimize other properties such as the truncation error. Euler's method is a RK method with $n = 1$ and $a_1 = 1$. *Ralston's method* is given by $n = 2$, $a_1 = 1/3$, $a_2 = 2/3$, and $q_{11} = 3/4$, and is the second-order RK method with the smallest truncation error. Runge-Kutta methods for $n = 2, \ldots, 5$ are used in practice, with truncation error $\mathcal{O}(h^{n+1})$ and global error $\mathcal{O}(h^n)$.

Example 3 (Ball/String). Figure 6 shows approximations of the trajectory from Ex. 2, using Ralston's method for $h = 0.1, 0.05, 0.025, 0.0125$. The number of evaluations for $h = 0.1$ with Ralston's method is the same as with Euler's method at $h = 0.05$. At $h = 0.025$, the global error at the end of the trajectory is 33% of the exact value, while at $h = 0.0125$ it is 7%, which is consistent with a global error of $\mathcal{O}(h^2)$. Figure 6(b) shows the approximation with a quiver plot of the derivative and illustrates that the $\mathbf{x}_{i+1}$ are closer to the real solution than the $\hat{\mathbf{x}}_i^2$ from which they are derived.

Error estimation and adaptive time steps Relatively precise error estimates can be obtained by comparing the result of two sequences with different levels of precision. One such approach is halving time steps, i.e., taking the difference between the result for time steps $h/2$ and h. Another is to take the difference between a $(n-1)$th order and a nth order solver. *Runge-Kutta-Fehlberg* (RKF) methods combine $(n-1)$th order and nth order RK methods such that the intermediate results from one sequence are used in the other, so it requires no more evaluations than an nth order RK method on its own. Popular ODE solvers are RKF 2(3) and RKF 4(5), also known as `ode23` and `ode45`.

The estimated error can be used to adapt the time steps. Let ε_a be the current estimate of the truncation error, and ϵ_d be the desired error. The time step can be adapted, e.g., using $h \leftarrow h\left|\epsilon_d/\varepsilon_a\right|^\alpha$, where $\alpha = 0.25$ if $\varepsilon_a < \epsilon_d$, and $\alpha = 0.2$ otherwise [48].



(a) position x in the time domain

(b) state space, with arrows indicating the derivative

Fig. 6. A trajectory of the ball/string example, approximated with Ralston's method for different time steps h. Intermediate states $\hat{\mathbf{x}}_i^2$ are shown as squares for $h = 0.1$, the exact solution is shown in solid black

Stiff Systems A system of ODEs is called *stiff* if it involves rapidly changing components together with slowly changing ones, typically with time constants differing by a factor of 1000 or more. On the scale of the slow time constant, the rapid changes thus seem to take place nearly instantaneously, and have little effect once they have died down. Nonetheless, solvers are forced to take tiny time steps throughout the entire time horizon due to stability problems caused by the fast time constants: the approximation error in each step moves the rapid component away from its equilibrium, which is followed by a rapid move back to the equilibrium that must be taken into account by taking small steps. Special

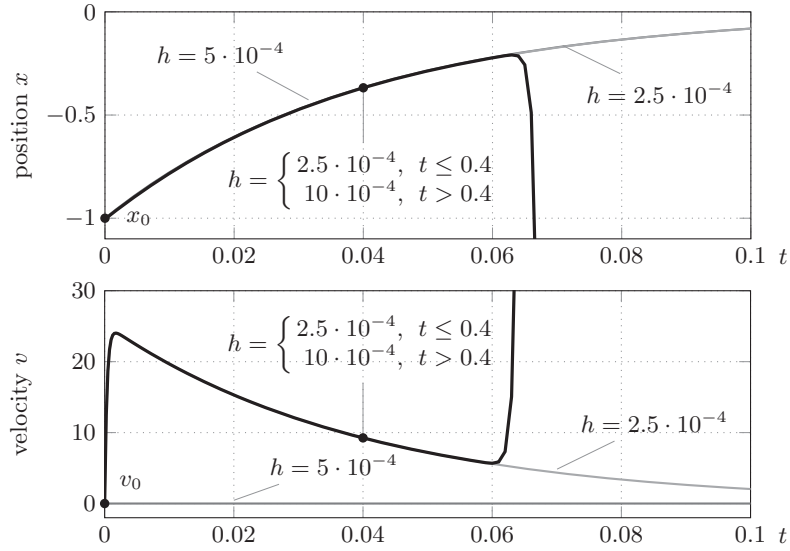


Fig. 7. When applied to stiff systems, Runge Kutta methods can be very sensitive to the time step and require a small time step throughout. Increasing the time step h even during slow phases can make the approximation diverge

solvers are available for stiff ODEs, using implicit methods to achieve stability at larger time steps.

Example 4 (Ball/String). The Ball/String system is stiff for sufficiently small values of the mass m. Figure 7 shows several approximations of the velocity trajectory from Ex. 2, obtained using Ralston's method for $m = 1/1000$. To obtain a stable result, the time step h must be about a factor 100 smaller than in Ex. 3. For $h = 0.5 \cdot 10^{-4}$, the solution is qualitatively false: v remains close to zero (x is approximated curiously well). For $h = 0.25 \cdot 10^{-4}$, the solution is approximated well, which indicates how sensitive the error is to the time step. The bold line depicts the trajectory obtained by using a time step $h = 0.25 \cdot 10^{-4}$ up to $t = 0.04$, and then switching to $h = 10 \cdot 10^{-4}$. Twenty steps after the switch, the sequence suddenly diverges to infinity.

3.2 Computing Trajectories and Jumps

Using an appropriate ODE solver from the previous section, we can approximate points on a trajectory in a given location up to arbitrary precision. However, we need to detect when the trajectory leaves the invariant, since this poses a hard limit on how long the system can stay in the location. This is related to the problem of detecting jumps, i.e., finding out when any of the outgoing transitions are enabled. Both types of events can be detected as a *zero crossing* of suitable functions that are zero on the border of the invariant and guard sets. Typically,

a vector of such functions is passed to the ODE solver, which stops whenever one of the functions changes sign from integration step to another. The solver then uses a root-finding algorithm to approximate the exact time of the crossing and returns the time and corresponding state at the crossing.

Shortcomings. Several difficulties arise in the above procedure, for a detailed discussion see [51]:

- Missed events: It is hard to ensure that no roots are missed. This means that violations of the invariants or states in the guard could go undetected.
- Increased computational cost: Using the ODE solver over a strictly increasing sequence of time points allows it to reuse certain intermediate states. This advantage is lost through the back-and-forth of the root-finding algorithm.
- Relaxed constraints: Since the ODE can only be solved approximately, the crossing state may lie slightly outside the guard or invariant. Relaxing the constraints to account for numerical errors may generate spurious behavior.

Zeno Behavior The switching times in a hybrid system may get closer and closer together, to the point that the sequence of switching times converges. This means that an infinite amount of events take place in a finite amount of time. Such behavior is called *Zeno*, and poses a particular problem for numerical simulation, since the simulator seems to get "stuck" as switching times converge.

Example 5 (Ball/String). Consider the Ball/String system turned upside down by reversing the gravitational acceleration to $g = -10$. In this form it becomes a variation of the well-known *bouncing ball*, a standard example for Zeno behavior. Figure 8 shows a run, starting from $x = -1, v = 0$, in which the switching times converge at around $t = 5.5$.

3.3 Accounting for Nondeterminism

The biggest challenge for numerical simulation is nondeterminism, such as

- sets of states: a state must be selected from the set of initial states and from the successor states in the jump relation;
- jumps times: the jump time of a transition with may semantics must be chosen from an interval of time;
- dynamics: a *differential inclusion*, such as $\dot{x} \in [-1, 1]$, leaves a choice for the derivative at each time step;
- discrete successors: if several guards are enabled simultaneously, one must choose between transitions;

A numerical simulation needs to pick one value from the set of possible choices in order to compute the next state. The set of possible runs from a single initial state grows exponentially with each choice. This is particularly detrimental if the dynamics are nondeterministic, where the choice is made at every time step.

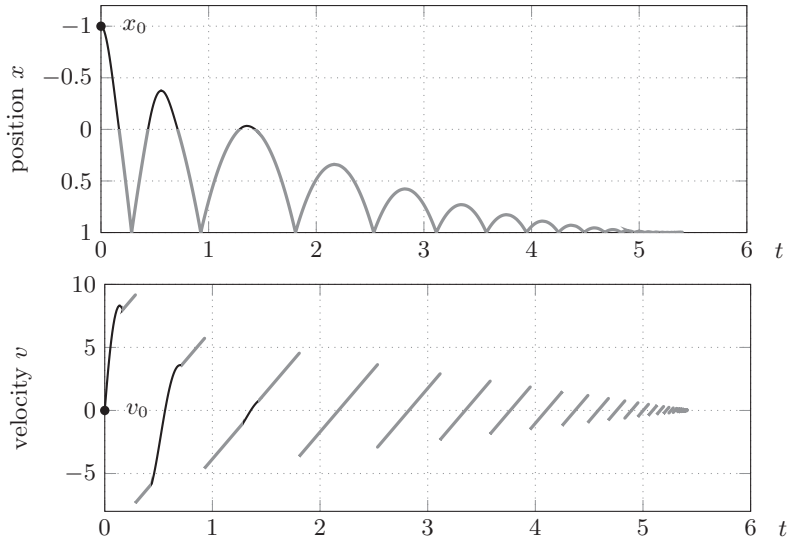


Fig. 8. Zeno behavior in the Ball/String example for $g = -10$. Trajectories in location *extension* are shown in black, trajectories in *freefall* in gray

Typical simulation environments such as Simulink [45], Modelica [46], or Ptolemy [21] use purely deterministic models with must semantics (ref. Sect 2.2), which make jump times deterministic. One way to deal with the complexity resulting from nondeterminism is to associate probabilities to the choices. This leads to *stochastic hybrid systems*, for which techniques such as Monte Carlo simulation can produce a sampling of the possible runs that is associated with probabilities [15].

3.4 Verification by simulation

Verification by simulation techniques aim at formally proving bounded-horizon properties by computing a finite number of trajectories [22, 37]. A so-called bisimulation metric is established to identify a neighborhood around each state such that the states inside remain sufficiently close together for all time. Under suitable assumptions, this makes it possible to identify a finite subset of initial states whose trajectories are sufficient to show certain properties of the system [28]. Similar techniques can be applied for parameter synthesis [20].

4 Reachability Analysis

Reachability analysis extends the concept of numerical simulation from numbers to sets. By computing with sets of states, nondeterminism in the model can be fully taken into account, and the analysis can be exhaustive, even up to an infinite time horizon. Furthermore, successor computations can be conservative in

the sense that the computed sets are sure to cover all solutions. Just like numerical simulation, reachability has to resort to approximations if the dynamics of the system are complex. On the downside, the cost of set-based computations generally increases sharply with the number of continuous variables, so scalability is critical. Scalable approximations are available for certain types of dynamics, as discussed later in this section, but this performance comes at a price in accuracy. The trade-off between runtime and accuracy remains a central problem in reachability analysis.

Decidability. The problem whether a given state is reachable from the initial states is generally undecidable for hybrid automata, which means that no algorithm exists that eventually terminates with the right answer [35]. The main subclass for which the problem is decidable are *timed automata*, where all derivatives have the value 1, guards are given by constraints with only one variable each (this may be extended to include the difference between two variables), and jumps either leave a variable unchanged or reset it to a constant value.

4.1 Reachability Algorithm

The standard method to compute the reachable states is to iterate the following *one-step successor* operators for discrete and continuous transitions. Given a set of states S, let $\mathsf{Post}_C(S)$ be the set of states reachable by letting time elapse from any state in S,

$$\mathsf{Post}_C(S) = \{(\ell, \xi(\delta)) \mid \exists (\ell, x) \in S : (\ell, \mathbf{x}) \xrightarrow{\delta, \xi} (\ell, \xi(\delta))\}.$$

Let $\mathsf{Post}_D(S)$ be the set of states resulting from a jump from any state in S,

$$\mathsf{Post}_D(S) = \{(\ell', \mathbf{x}') \mid \exists (\ell', \mathbf{x}') \in S, \exists \alpha \in \mathsf{Lab} \cup \{\tau\} : (\ell, \mathbf{x}) \xrightarrow{\alpha} (\ell', \mathbf{x}')\}.$$

Starting from the initial states, $\mathsf{Post}_C(S)$ and $\mathsf{Post}_D(S)$ are computed in alternation and all states that are obtained are recorded, as in the following sequence:

$$R_0 = \mathsf{Post}_C(\mathsf{Init}), \tag{8}$$

$$R_{i+1} = R_i \cup \mathsf{Post}_C(\mathsf{Post}_D(R_i)). \tag{9}$$

If the sequence reaches a fixed-point, i.e., when $R_{i+1} = R_i$, then R_i is the set of reachable states. Note that simply computing the sequence and testing for a fixed-point may not lead to termination, even for systems where reachability is decidable. E.g., a system with an (unbounded) counter would enter a new state at each iteration such that the fixed-point is never reached.

In tools such as HyTech [33], PHAVer [23] and SpaceEx [24], the sequence (8) is computed using *symbolic states* $s = (\ell, \mathcal{P})$, where $\ell \in \mathsf{Loc}$ and $\mathcal{P}$ is a continuous set, e.g., a polyhedron. Computing the timed successors Post_C of a symbolic state $s = (\ell, \mathcal{P})$ produces a new symbolic state $s' = (\ell, \mathcal{P}')$. Computing the jump successors Post_D of $s = (\ell, \mathcal{P})$ involves iterating over all outgoing

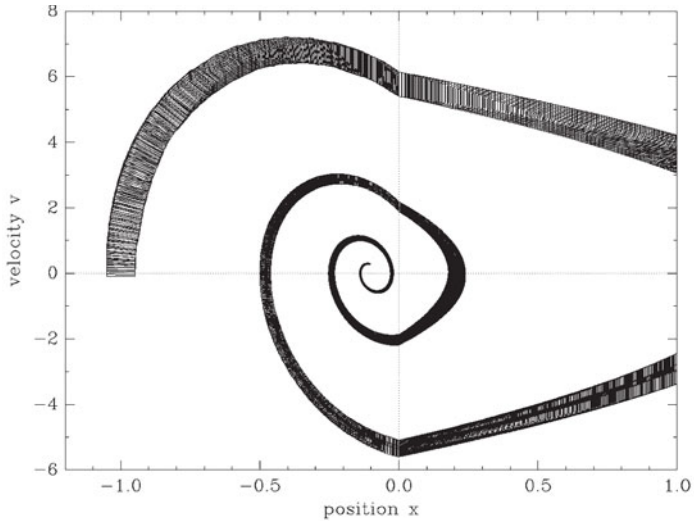


Fig. 9. Reachable states of the ball/string example, computed using SpaceEx

transitions of ℓ, and produces a set of symbolic states $\{s'_1, \ldots, s'_N\}$, each in one of the target locations. A *waiting list* contains the symbolic states whose successors still need to be explored, and a *passed list* contains all symbolic states computed so far. The fixed-point computation proceeds as follows:

1. Initialization: Compute the continuous successors of the initial states and put them on the waiting list.
2. Pop a symbolic state s from the waiting list and compute its one-step successors $\{s'_1, \ldots, s'_N\} = \mathsf{Post}_C(\mathsf{Post}_D(s))$.
3. Containment checking: Discard the s'_i that have previously been encountered, i.e., those contained in any symbolic state on the passed list. Add the remaining symbolic states to the passed and waiting list.
4. If the waiting list is empty, terminate and return the passed list as the reachable states. Otherwise, continue with step 2.

Different approaches are taken for computing the one-step successors, depending on the type of dynamics. In the following sections, we present the major methods.

Example 6. Figure 9 shows the reachable states of the ball/string example, starting from an initial set of $-1.05 \leq x \leq -0.95$, $-0.1 \leq v \leq 0.1$ in location *extension*. Initializing the waiting list with the continuous successors of the intial states, the fixed point is reached on the 6th iteration. Each symbolic state corresponds to a segment of the run from Ex. 1, and contains all of the corresponding trajectories, from any of the initial states.

4.2 Piecewise Constant Dynamics

Hybrid automata with piecewise constant dynamics (PCDA), also called *linear hybrid automata* (LHA), have

- initial states and invariants given by conjunctions of linear constraints,
- flows given by conjunctions of linear constraints over the derivatives $\dot{X}$, and
- jumps given by linear constraints over $X \cup X'$, where X' denote the variables after the jump.

The one-step successors of PCDA can be computed exactly, which is not the case for the more complex dynamics discussed in later sections. For simplicity we will assume that flow constraints are closed and bounded. Examples for flow constraints of a PCDA include differential inclusions such as $\dot{x} \in [1, 2]$, and conservation laws such as $\dot{x} + \dot{y} = 0$. The jump constraints of a PCDA can generate complex behavior, and even chaos [16]. For example, PCDA can model *discrete-time affine systems*, a widely used class of control systems, with jump constraints of the form $\mathbf{x}' = A\mathbf{x} + \mathbf{b}$.

Continuous successors. In the following, we discuss how to compute the states reachable by time elapse in a given location ℓ. Since ℓ is clear from the context we call $\mathbf{x}$ a (continuous) state. By definition, a trajectory can be an arbitrarily curved function as long as it is differentiable and satisfies the constraints of flow and invariant. For PCDA, it suffices to consider straight-line trajectories:

Lemma 1. *[36] In any given location of a PCDA, there is a trajectory $\xi(t)$ from $\mathbf{x} = \xi(0)$ to $\mathbf{x}' = \xi(\delta)$ for some $\delta > 0$ iff $\eta(t) = \mathbf{x} + \mathbf{q}t$ with $\mathbf{q} = (\mathbf{x}' - \mathbf{x})/\delta$ is a trajectory from $\mathbf{x}$ to $\mathbf{x}'$.*

Consider polyhedra $\mathcal{P}$ and $\mathcal{Q}$. The states on straight line trajectories starting in $\mathcal{P}$ with constant derivative $\dot{x} = \mathbf{q}$ for any $\mathbf{q} \in \mathcal{Q}$ are the *time successors* [6].

$$\mathcal{P} \nearrow \mathcal{Q} = \{\mathbf{x}' \mid \mathbf{x} \in \mathcal{P}, \mathbf{q} \in \mathcal{Q}, t \in \mathbb{R}^{\geq 0}, \mathbf{x}' = \mathbf{x} + \mathbf{q}t\}. \tag{10}$$

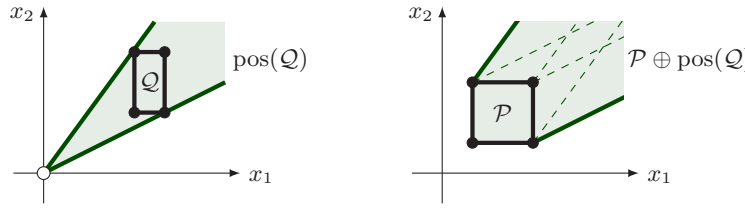
We now transform the right-hand term of (10) into a linear constraint. Let $\mathcal{P}$ and $\mathcal{Q}$ be polyhedra given in vector-matrix form as $\mathcal{P} = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$, $\mathcal{Q} = \{\mathbf{q} \mid \bar{A}\mathbf{q} \leq \bar{\mathbf{b}}\}$. Eliminating $\mathbf{q} = \frac{\mathbf{x}' - \mathbf{x}}{t}$ for $t > 0$ and multiplying with t yields

$$\mathcal{P} \nearrow \mathcal{Q} = \left\{ \mathbf{x}' \;\middle|\; A\mathbf{x} \leq \mathbf{b} \;\wedge\; \bar{A}(\mathbf{x}' - \mathbf{x}) \leq \bar{\mathbf{b}} \cdot t \;\wedge\; t \geq 0 \right\}. \tag{11}$$

The above set is a polyhedron that can be computed by quantifier elimination over $X \cup \{t\}$ using, e.g., Fourier-Motzkin elimination. The time successors can also be obtained using geometrical operations, as illustrated in Fig. 10. The *cone* of $\mathcal{Q}$ is the polyhedron $\text{pos}(\mathcal{Q}) = \{\mathbf{q} \cdot t \mid \mathbf{q} \in \mathcal{Q}, t \geq 0\}$. The *Minkowski sum* is defined as $\mathcal{P} \oplus \mathcal{Q} = \{\mathbf{p} + \mathbf{q} \mid \mathbf{p} \in \mathcal{P}, \mathbf{q} \in \mathcal{Q}\}$. The time successors are [31]

$$\mathcal{P} \nearrow \mathcal{Q} = \mathcal{P} \oplus \text{pos}(\mathcal{Q}). \tag{12}$$

If $\mathcal{P}$ and $\mathcal{Q}$ are closed with generator representation (V, R) and $(\bar{V}, \bar{R})$, respectively, then a generator representation of $\mathcal{P} \nearrow \mathcal{Q}$ is $(V, R \cup \bar{V} \cup \bar{R})$. It remains to ensure that the time successors satisfy the invariant $\mathsf{Inv}(\ell)$, which leads to the following continuous successor operator for PCDA.



(a) $\mathcal{Q}$ and its cone $\mathrm{pos}(\mathcal{Q})$ (b) $\mathcal{P}$ and $\mathcal{P} \nearrow \mathcal{Q} = \mathcal{P} \oplus \mathrm{pos}(\mathcal{Q})$

Fig. 10. The time successors $\mathcal{P} \nearrow \mathcal{Q}$, obtained using geometric operations on $\mathcal{P}$ and $\mathcal{Q}$

Lemma 2. *[6] The* continuous successors *of a polyhedron $\mathcal{P}$ in location ℓ are*

$$\mathsf{post}_\ell(P) = \big(P \nearrow \mathsf{Flow}(\ell)\big) \cap \mathsf{Inv}(\ell).$$

The computation of the time successors is of exponential complexity. In the form of (11), it requires quantifier elimination, while in the form of (12) it requires switching representations, since intersection computed on $\mathcal{H}$-polyhedra and Minkowski sum on $\mathcal{V}$-polyhedra.

Discrete successors. The *discrete successors* of a polyhedron $\mathcal{P}$ for an edge $e = (\ell, \alpha, k)$ is the polyhedron:

$$\mathsf{post}_e(P)\big\{\mathbf{x}' \;\big|\; \exists \mathbf{x} \in \mathcal{P} : (\mathbf{x}, \mathbf{x}') \in \mathsf{Jump}(\epsilon) \wedge \mathbf{x}' \in \mathsf{Inv}(k)\big\}.$$

This set is defined using existential quantification, and computing it may require costly quantifier elimination. Frequently occurring special cases can be computed more efficiently. Consider $\mathsf{Jump}(e)$ given by a guard $\mathbf{x} \in \mathcal{G}$ and an assignment $\mathbf{x}' = C\mathbf{x} + \mathbf{d}$, with a constant matrix C and a vector $\mathbf{d}$ of appropriate dimensions. The discrete successors are

$$\mathsf{post}_e(P) = \big(C(\mathcal{P} \cap \mathcal{G}) \oplus \{\mathbf{d}\}\big) \cap \mathsf{Inv}(k). \tag{13}$$

If C is invertible and $\mathcal{P}, \mathcal{G}$ are $\mathcal{H}$-polyhedra, the computation is straightforward since intersection corresponds to concatenation of constraints, and for any polyhedron $\mathcal{Q} = \{\mathbf{x} \mid A\mathbf{x} \le \mathbf{b}\}$,

$$C\mathcal{Q} \oplus \{\mathbf{d}\} = \{\mathbf{x} \mid AC^{-1}\mathbf{x} \le b + C^{-1}\mathbf{d}\}.$$

4.3 Piecewise Affine Dynamics

Hybrid automata with piecewise affine dynamics (PWA) have

- initial states and invariants given by conjunctions of linear constraints,
- flows given by affine ODEs, and
- jumps given by a guard set and linear assignments.

We divide the continuous variables into *state variables* $X = \{x_1, \ldots, x_n\}$, whose derivative is explicitly defined, and *input variables* $U = \{u_1, \ldots, u_m\}$, whose derivative is unconstrained. The input variables can be used to model nondeterminism such as open inputs to the system, approximation errors, disturbances, etc. In each location of a PWA, the continuous dynamics are given by *affine ODEs* of the form

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}, \qquad \mathbf{u} \in \mathcal{U}, \tag{14}$$

where A and B are matrices of appropriate dimension and the *input set* $\mathcal{U}$ is compact and convex. Note that $\mathcal{U}$ may be specified in the invariant. To simplify notation in this section, we assume that constants are modeled with $\mathcal{U}$, e.g., $\dot{\mathbf{x}} = A\mathbf{x}+\mathbf{b}$ with $B = I$ and $\mathcal{U} = \{b\}$. Some differential inclusions can be brought to the form of (14) by introducing auxiliary variables. The jump constraints of an edge e are defined by a *guard* set $\mathcal{G}$ and an assignment of the form

$$\mathbf{x}' = C\mathbf{x} + D\mathbf{u}, \tag{15}$$

where $\mathbf{x}'$ denotes the value of $\mathbf{x}$ after the jump, u is defined as above and C and D are matrices of appropriate dimension.

Continuous successors. We start with the basic construction, which ignores the invariant. The evolution of the input variables is described by an *input signal* $\zeta : \mathbb{R}^{\geq 0} \to \mathcal{U}$ that attributes to each point in time a value of the input $\mathbf{u}$. The input signal does not need to be continuous. A trajectory $\xi(t)$ from a state $\mathbf{x}_0$ is the solution of the differential equation (14) for initial condition $\xi(0) = \mathbf{x}_0$ and a given input signal ζ. It has the form

$$\xi_{\mathbf{x}_0,\zeta}(t) = e^{At}\mathbf{x}_0 + \int_0^t e^{A(t-s)}B\zeta(s)ds. \tag{16}$$

It consists of the superposition of the solution of the *autonomous* system, obtained for $\zeta(t) = 0$, and the *input integral* obtained for $x_0 = 0$. Let $\mathcal{X}_t$ be the states reachable in time t from any state in $\mathcal{X}_0$ and let $\mathcal{Y}_t$ be the states reachable from $\mathcal{X}_0 = \{0\}$, then (16) can be written as

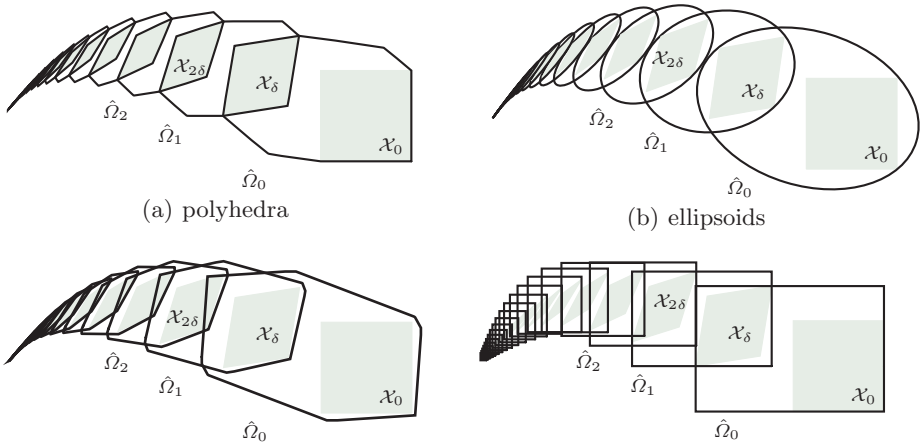
$$\mathcal{X}_t = e^{At}\mathcal{X}_0 \oplus \mathcal{Y}_t. \tag{17}$$

The goal is to compute a finite sequence of sets $\Omega_0, \Omega_1, \ldots$ such that

$$\bigcup_{0 \leq t \leq T} \mathcal{X}_t \subseteq \Omega_0 \cup \Omega_1 \cup \ldots. \tag{18}$$

We present the construction of the sequence Ω_k for a fixed *time step* $\delta > 0$ such that Ω_k covers $\mathcal{X}_t$ for $t \in [k\delta, (k+1)\delta]$, as illustrated in Fig. 11. The so-called *semi-group* property of reachability says that, starting from $\mathcal{X}_s$, for any $s \geq 0$, and then waiting r time units leads to the same states as starting from $\mathcal{X}_0$ and waiting $r + s$ time units. Applying this to (17), we obtain that for any $r, s \geq 0$,

$$\mathcal{X}_{r+s} = e^{Ar}\mathcal{X}_s \oplus \mathcal{Y}_r. \tag{19}$$



(a) polyhedra (b) ellipsoids

(c) zonotopes (d) support functions (box directions)

Fig. 11. A sequence of sets $\Omega_0, \Omega_1, \ldots$ that covers $\mathcal{X}_t$ over a finite time horizon T. The choice of set representation for Ω_k has a substantial impact on accuracy and computational complexity

Substituting $r \leftarrow \delta$, $s \leftarrow k\delta$, we get a time discretization

$$\mathcal{X}_{(k+1)\delta} = e^{A\delta}\mathcal{X}_{k\delta} \oplus \mathcal{Y}_\delta.$$

It follows that if we have initial approximations Ω_0 and Ψ_δ such that

$$\bigcup_{0 \le t \le \delta} \mathcal{X}_t \subseteq \Omega_0, \qquad \mathcal{Y}_\delta \subseteq \Psi_\delta, \tag{20}$$

then the sequence

$$\Omega_{k+1} = e^{A\delta}\Omega_k \oplus \Psi_\delta. \tag{21}$$

satisfies (18). Note that Ω_0 covers the reachable set over an interval of time $[0, \delta]$, while Ψ_δ covers the values of the input integral at a single time instant δ.

Computing initial approximations Ω_0 and Ψ_δ . The set Ω_0 needs to cover $\mathcal{X}_t$ from $t = 0$ to $t = \delta$. A good starting point for such a cover is the convex hull of $\mathcal{X}_0$ and $\mathcal{X}_\delta$. One approach, shown in Fig. 12(a), is to compute the convex hull in constraint representation, and push the facets out far enough to be conservative [29]. The required values can be computed from a Taylor approximation of (16) [8], or by solving an optimization problem [18]. Note that the cost of computing the exact constraints of the convex hull can be exponential in the number of variables, which limits the scalability of this approach.

A scalable way to obtain Ω_0 is to bloat $\mathcal{X}_0$ and $\mathcal{X}_\delta$ enough to compensate for the curvature of trajectories, as illustrated in Fig. 12(b). The approach from [26] uses uniform bloating and whose approximation error is asymptotically linear in the time step δ as $\delta \to 0$. This is asymptotically optimal for any approximation

(a) pushing facets (b) bloating

Fig. 12. An approximation Ω_0 that covers $\mathcal{X}_t$ for $t \in [0, \delta]$ can be obtained from the convex hull of $\mathcal{X}_0$ and $\mathcal{X}_\delta$ and compensating for the curvature of trajectories

containing the convex hull of $\mathcal{X}_0$ and $\mathcal{X}_\delta$ [41]. The bloating factor is derived from a Taylor approximation of (16), whose remainder is bounded using norms. To formalize the above statements, we use the following notation. Let $\|\cdot\|$ be a vector norm and let $\|A\|$ be its induced matrix norm.[1] Let $\mu(\mathcal{X}) = \max_{x \in \mathcal{X}} \|x\|$ and let $\mathcal{B}$ be the unit *ball* of the norm, i.e., the largest set $\mathcal{B}$ such that $\mu(\mathcal{B}) = 1$. For a scalar c, let $c\mathcal{X} = \{cx \mid x \in \mathcal{X}\}$.

Lemma 3. *[26] Given a set of initial states $\mathcal{X}_0$ and affine dynamics (14), let*

$$\begin{aligned} \alpha_\delta &= \mu(\mathcal{X}_0) \cdot (e^{\|A\|\delta} - 1 - \|A\|\delta), \\ \beta_\delta &= \tfrac{1}{\|A\|}\mu(B\mathcal{U}) \cdot (e^{\|A\|\delta} - 1), \\ \Omega_0 &= \text{chull}(\mathcal{X}_0 \cup e^{A\delta}\mathcal{X}_0) \oplus (\alpha_\delta + \beta_\delta)\mathcal{B}, \\ \Psi_\delta &= \beta_\delta \mathcal{B}. \end{aligned}$$

Then $\bigcup_{0 \le t \le \delta} \mathcal{X}_t \subseteq \Omega_0$ and $\mathcal{Y}_\delta \subseteq \Psi_\delta$.

Approximations and the wrapping effect. The sequence in (21) can be problematic to compute since the complexity of Ω_k may increase sharply with k. To avoid this increase in complexity, we approximate each Ω_k by a simplified set. Let Appr be an *approximation function* such that for any set $\mathcal{P}$, $\mathcal{P} \subseteq \text{Appr}(\mathcal{P})$. The sequence (21) then becomes

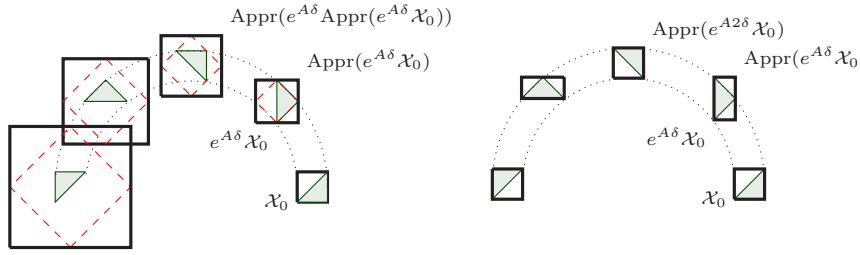
$$\hat{\Omega}_{k+1} = \text{Appr}(e^{A\delta}\hat{\Omega}_k \oplus \Psi_\delta). \tag{22}$$

The recursive application of the approximation function can lead to an exponential increase in the approximation error. This phenomenon is known in numerical analysis as the *wrapping effect* [38] and is illustrated in Fig. 13.

For affine dynamics, the wrapping effect can be avoided by combining two techniques [27]. First, the alternation of the map $e^{Ak\delta}$ with the Minkowski sum in (21) is avoided by splitting it into two sequences

$$\begin{aligned} \hat{\Psi}_{k+1} &= \text{Appr}(e^{Ak\delta}\Psi_\delta) \oplus \hat{\Psi}_k, \qquad \text{with } \hat{\Psi}_0 = \{0\}, \\ \hat{\Omega}_k &= \text{Appr}(e^{Ak\delta}\Omega_0) \oplus \hat{\Psi}_k. \end{aligned} \tag{23}$$

[1] For example, the *infinity norm* $\|x\|_\infty = \max\{|x_1|, \ldots, |x_n|\}$ induces the matrix norm $\|A\| = \max_{1 \le i \le n} \sum_{j=1}^{m} |a_{ij}|$, where A is of dimension $n \times m$. Its ball $\mathcal{B}_\infty$ is a cube of side length 2.



(a) with wrapping effect (b) using a wrapping-free algorithm

Fig. 13. An example for the wrapping effect, with $e^{A\delta}$ performing a rotation of 45 degrees around the origin. The exact solution is $e^{Ak\delta}\mathcal{X}_0$ (shaded). Mapping (dashed) and then applying the approximation operator (thick) at each step leads to the wrapping effect. For visual clarity, $\mathcal{X}_0$ is used here instead of Ω_0

Second, the approximation operator is chosen such that

$$\mathrm{Appr}(\mathcal{P} \oplus \mathcal{Q}) = \mathrm{Appr}(\mathcal{P}) \oplus \mathrm{Appr}(\mathcal{Q}),$$

which is the case, e.g., for the interval hull (bounding box). Under this assumption it holds that $\hat{\Omega}_k = \mathrm{Appr}(\Omega_k)$, which means the resulting approximation is free of the wrapping effect.

Invariants. A simple but frequently sufficient heuristic to account for the invariant is to stop computing the sequence Ω_k as soon as Ω_k lies completely outside of the invariant. The computed $\Omega_0, \ldots, \Omega_{k-1}$ are then intersected with $\mathsf{Inv}(\ell)$, which produces an overapproximation of the exact solution. A more precise solution can be obtained by intersecting at each step with the set of states reachable from the invariant itself [30].

Discrete successors. Consider an edge $e = (\ell, \sigma, k)$ of a PWA, with guard set $\mathcal{G}$ and assignment

$$\mathbf{x}' = C\mathbf{x} + D\mathbf{u}.$$

Recall that $u \in \mathcal{U}$, where $\mathcal{U}$ is compact, convex and given by constraints in $Inv(\ell)$. The discrete successors of a set $\mathcal{P}$ is

$$\mathsf{post}_e(\mathcal{P}) = \big(C(\mathcal{P} \cap \mathcal{G}) \oplus D\mathcal{U}\big) \cap \mathsf{Inv}(k).$$

Set Representations Whether the presented successor operators $\mathsf{post}_\ell(\mathcal{P})$ and $\mathsf{post}_e(\mathcal{P})$ are efficient to compute, depends on the type of set used for $\mathcal{P}$ and how it is represented. We summarize some of the set representations proposed in literature. Scalable implementations and approximations need to be available for the operators in the algorithm. Using the initial approximation from Lemma 3

and the recurrence equation (23), the operators are linear map, Minkowski sum, convex hull and intersection.

Polyhedra. Figure 11(a) shows a reach set approximation computed using polyhedra. The class of polyhedra is closed under all required operations, i.e., linear map, Minkowski sum, convex hull, and intersection. However, not all of them scale well. As mentioned in Sect. 4.2, intersection is computed on $\mathcal{H}$-polyhedra and Minkowski sum on $\mathcal{V}$-polyhedra, and the result can be of exponential complexity in both forms. A polyhedral approximation for the non-scalable operations can be efficiently computed by a-priori fixing the facet normals of the result, which leads to so-called *template polyhedra*. The accuracy of the approximation can be increased by including additional directions, leading to a scalable approach [9].

Ellipsoids. A scalable reachability algorithm for affine dynamics is obtained for ellipsoids [39], see Fig. 11(b). An *ellipsoid* $\mathcal{E}(\mathbf{c}, Q) \subseteq \mathbb{R}^n$ is represented by a center $\mathbf{c} \in \mathbb{R}^n$ and a positive definite[2] matrix $Q \in \mathbb{R}^{n \times n}$,

$$\mathcal{E}(\mathbf{c}, Q) = \big\{ \mathbf{x} \;\big|\; (\mathbf{x} - \mathbf{c})^{\mathsf{T}} Q^{-1} (\mathbf{x} - \mathbf{c}) \leq 1 \big\}.$$

Deterministic affine transforms can be computed efficiently for ellipsoids. However, ellipsoids are not closed under Minkowski sum, convex hull, nor intersection. One therefore suffers from the wrapping effect unless $B\mathcal{U}$ is a singleton. Efficient approximations are available for Minkowski sum, convex hull, and special cases of intersection, but the computation of discrete successors can be problematic in terms of accuracy. For an implementation, see [40].

Zonotopes. Zonotopes are a subclass of central-symmetric polytopes that has been used successfully for reachability analysis [26, 3], see Fig. 11(c). A *zonotope* $\mathcal{P} \subseteq \mathbb{R}^n$ is defined by a center $\mathbf{c} \in \mathbb{R}^n$ and generators $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbb{R}^n$ as

$$\mathcal{P} = \big\{ \mathbf{c} + \sum\nolimits_{i=1}^{k} \alpha_i \mathbf{v}_i \;\big|\; \alpha_i \in [-1, 1] \big\}.$$

Affine transformations and Minkowski sum can be computed efficiently for zonotopes. Since zonotopes are closed under Minkowski sum, it is straightforward to devise an approximation operator Appr that distributes over Minkowski sum and use the wrapping-free sequence (23). Zonotopes are neither closed under convex hull, nor under intersection. But efficient approximations exists, and the accuracy of approximating the convex hull in the above reachabililty algorithm can be improved by taking smaller time steps. However, the lack of accuracy in intersections can make the computation of discrete successors with zonotopes problematic. In special cases it can be advantageous to use an approach called *continuization* to avoid the intersection operation, see [2].

Support functions. The support function of a convex set represents the set exactly. Support functions lead to very scalable algorithms since linear map,

[2] A matrix Q is positive definite iff it is symmetric and $\mathbf{x}^{\mathsf{T}} Q \mathbf{x} > 0$ for all $\mathbf{x} \neq 0$.

(a) support function in direction d (b) outer approximation

Fig. 14. Evaluating the support function in a set of directions gives a polyhedral outer approximations that can be computed very efficiently

Minkowski sum, and convex hull correspond to simple operations on vectors and scalars [42, 30].

The *support function* $\varrho_{\mathcal{P}} : \mathbb{R}^n \to \mathbb{R}$ of a nonempty, closed, bounded, and convex set $\mathcal{P}$ is $\varrho_{\mathcal{P}}(\mathbf{d}) = \max\{\mathbf{d}^{\mathsf{T}}\mathbf{x} \mid \mathbf{x} \in \mathcal{P}\}$. It attributes to every *direction* $d \in \mathbb{R}^n$ the position of the tangent halfspace in that direction, see Fig. 14(a). The values of the support function over a set of directions $\mathcal{D} \subseteq \mathbb{R}^n$ define an *outer approximation*

$$\lceil \mathcal{P} \rceil_{\mathcal{D}} = \bigcap_{\mathbf{d} \in \mathcal{D}} \{\mathbf{d}^{\mathsf{T}} x \leq \varrho_{\mathcal{P}}(\mathbf{d})\}.$$

If $\mathcal{D} = \mathbb{R}^n$ or $\mathcal{D}$ is the ball of a norm, then $\lceil \mathcal{P} \rceil_{\mathcal{D}} = \mathcal{P}$. If $\mathcal{D}$ is a finite set of directions, the outer approximation is a polyhedron, see Fig. 14(b). If $\mathcal{D}$ consists of the positive and negative axis directions, the result is an interval hull (bounding box), see Fig. 11(d). If the goal is to compute an outer approximation of a given accuracy, one does not escape the curse of dimensionality: an outer approximation with an error of ε in n dimensions requires $\mathcal{O}(1/\varepsilon^{n-1})$ directions. However, even a small number of directions can lead to reachability results with an acceptable approximation error [24].

Computing the support function of the sequence (23) for a given direction can be done very efficiently even without the approximation operator Appr [30]. Linear map, Minkowski sum, and convex hull are easily computed with support functions. The intersection operation is more complex, and can be formulated as an optimization problem [30]. Switching to $\mathcal{H}$-polyhedra (the outer approximation) before intersection operations can avoid scalability problems, but leads to an overapproximation error [24].

Clustering The accuracy of the approximation in Lemma 3 depends on the size of the time step. This property, common to all approaches cited in Sect. 4.3, points to a potential bottleneck: To achieve a desired accuracy, one may end up with a large number of sets to cover the required time horizon. In the next iteration of the fixed point computation, each one of these sets may become the initial set of yet another sequence, easily leading to an exponential increase in the number of sets. If only very few of these sets intersect with the guard

sets, the discrete successor computation acts as a filter that might just keep the number of sets manageable. But this is not the case in general; note that these sets necessarily overlap. To prevent an explosion in the number of sets, a common approach is to cluster together all sets that intersect with the same guard [30]. The clustering operation, e.g., taking the convex hull, can itself be costly and adds to the approximation error in a way that is not easy to quantify. An approach to obtain a suboptimal number of clusters for a given error bound is presented in [25].

4.4 Nonlinear Dynamics

We give only a very brief overview of techniques that deal with nonlinear dynamics

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

where f is usually assumed to be globally Lipschitz continuous.

Linearization. One way to deal with nonlinear dynamics is to approximate them with affine dynamics $\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{u}, \mathbf{u} \in \mathcal{U}$ and then use reachability algorithms for affine dynamics. First, the states are confined to a bounded domain $\mathcal{S}$. This could be the invariant in a location, or $\mathcal{S}$ can be derived from suitable bounds around a given set of initial states. Then, a suitable matrix A and vector $\mathbf{b}$ are chosen. For example, linearizing $f(x)$ around a point $\mathbf{x}_0 \in \mathcal{S}$ gives matrix elements $a_{ij} = \frac{\partial f_i}{\partial x_j}\big|_{\mathbf{x}=\mathbf{x}_0}$ and $\mathbf{b} = f(\mathbf{x}_0) - A\mathbf{x}_0$. Finally, one derives a set $\mathcal{U}_\epsilon$ that bounds the error such that for all $x \in \mathcal{S}$, $f(x) - (Ax + b) \in \mathcal{U}_\epsilon$. Such bounds can be obtained using, e.g., interval arithmetic or optimization techniques. The states reachable using the affine dynamics $\dot{x} = Ax + u,\ u \in \mathcal{U}_\epsilon \oplus \{b\}$ cover those of the original nonlinear dynamics.

The accuracy of the linearization depends on the size of the domain $\mathcal{S}$ and can be increased by partitioning $\mathcal{S}$ into smaller parts. This process is known as *phase portrait approximation* [34]. It can be of use even when dealing with purely continuous dynamical systems, and is also referred to as *hybridization* [7].

Polynomial Approximations. If the dynamics are polynomial, bringing them to Bernstein form allows one to compute conservative approximations of successors sets in polynomial form [19, 47]. Another approach is to use *Taylor models*, which are polynomial approximations of a functions that are derived from a higher-order Taylor expansion and an interval bound on the remainder. The resulting ODE can be solved by iterative approximations using the Picard operator. The reachable states are approximated by sets that are polyhedra [49] or polynomial images of intervals [17]. A similar approach uses polynomial images of zonotopes, which are themselves images of intervals [1]. Since polynomial images of intervals are generally not closed under intersection, the accuracy may be diminished when computing discrete successors.

5 Conclusions

Systems with mixed continuous-discrete dynamics can exhibit complex behaviors that are difficult to analyze and predict, even for small systems with only a handful of variables. If safety or performance is critical, one would like to verify that the systems behaves according to the specification. Hybrid automata provide a rigorous mathematical formalism for describing and verifying such systems. Certain types of specifications can also be described in this form, as *monitor automata* that are run in parallel with the rest of the system, with an error location that is reachable if the specification is violated.

The most basic way to analyze the behavior of a hybrid automaton is to pick an initial state and numerically compute an approximation of one of its runs. By computing a sufficiently large number of such runs, sampling a variety of initial states and other sources of nondeterminism, one hopes to get a fairly good idea about the system. But this is not exhaustive and critical behavior may be missed. Set-based reachability, in the form presented in this chapter, is an extension of numerical simulation that establishes a conservative cover of all possible runs. If a safety specification is satisfied by the cover, one can be sure that none of the runs violates the specification. If the cover violates the specification, one either needs to tune the analysis parameters in order to obtain a closer approximation, try to confirm the violation with numerical simulation, or resort to alternative techniques.

The biggest drawback of set-based reachability is the computational cost, which depends on the number of variables in the system and the complexity of the dynamics. Scalable algorithms are known for systems with piecewise affine dynamics, but the trade-off between approximation accuracy and computational cost remains a challenge. Recent progress for systems with nonlinear dynamics has also lead to encouraging results for more and more complex systems. As reachability techniques mature, it remains to figure out how to best integrate them in the design and engineering process, connect them to existing models, and establish suitable specifications.

References

1. M. Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Hybrid systems: computation and control (HSCC'13)*, pages 173–182. ACM, 2013.
2. M. Althoff and B. H. Krogh. Avoiding geometric intersection operations in reachability analysis of hybrid systems. In *Hybrid Systems: Computation and Control (HSCC'12)*, pages 45–54. ACM, 2012.
3. M. Althoff, B. H. Krogh, and O. Stursberg. Analyzing reachability of linear dynamic systems with parametric uncertainties. In A. Rauh and E. Auer, editors, *Modeling, Design, and Simulation of Systems with Uncertainties*. Springer, 2011.
4. R. Alur. Formal verification of hybrid systems. In S. Chakraborty, A. Jerraya, S. K. Baruah, and S. Fischmeister, editors, *EMSOFT*, pages 273–278. ACM, 2011.

5. R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
6. R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, LNCS 736, pages 209–229. Springer, 1993.
7. E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Inf.*, 43(7):451–476, 2007.
8. E. Asarin, T. Dang, O. Maler, and O. Bournez. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *LNCS*, pages 20–31. Springer, 2000.
9. E. Asarin, T. Dang, O. Maler, and R. Testylier. Using redundant constraints for refinement. In *Automated Technology for Verification and Analysis*, pages 37–51. Springer, 2010.
10. R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.
11. O. Bouissou, A. Chapoutot, S. Mimram, and B. Strazzulla. Set-based simulation for design and verification of simulink models. In *Embedded Real Time Software and Systems (ERTS'14)*, 2014.
12. O. Bouissou, S. Mimram, and A. Chapoutot. Hyson: Set-based simulation of hybrid systems. In *RSP*, pages 79–85. IEEE, October 2012.
13. D. Brück, H. Elmqvist, S. E. Mattsson, and H. Olsson. Dymola for multi-engineering modeling and simulation. In *Proceedings of Modelica*, 2002.
14. R. P. Canale and S. C. Chapra. Numerical methods for engineers. *Mc Graw Hill, New York*, 1998.
15. C. G. Cassandras and J. Lygeros. *Stochastic hybrid systems*. CRC Press, 2006.
16. C. Chase, J. Serrano, and P. J. Ramadge. Periodicity and chaos from switched flow systems: contrasting examples of discretely controlled continuous systems. *Automatic Control, IEEE Transactions on*, 38(1):70–83, 1993.
17. X. Chen, E. Ábrahám, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *RTSS*, pages 183–192. IEEE Computer Society, 2012.
18. A. Chutinan and B. H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In F. W. Vaandrager and J. H. van Schuppen, editors, *HSCC*, volume 1569 of *LNCS*, pages 76–90. Springer, 1999.
19. T. Dang and R. Testylier. Reachability analysis for polynomial dynamical systems using the bernstein expansion. *Reliable Computing*, 17(2):128–152, 2012.
20. A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*, pages 167–170. Springer, 2010.
21. J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity-the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.
22. G. E. Fainekos, A. Girard, and G. J. Pappas. Temporal logic verification using simulation. In *Formal Modeling and Analysis of Timed Systems*, pages 171–186. Springer, 2006.
23. G. Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *STTT*, 10(3):263–279, 2008.
24. G. Frehse, C. L. Guernic, A. Donzé, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In G. Gopalakrishnan and S. Qadeer, editors, *CAV*, LNCS. Springer, 2011.

25. G. Frehse, R. Kateja, and C. Le Guernic. Flowpipe approximation and clustering in space-time. In *Hybrid systems: computation and control (HSCC'13)*, pages 203–212. ACM, 2013.
26. A. Girard. Reachability of uncertain linear systems using zonotopes. In M. Morari and L. Thiele, editors, *HSCC*, volume 3414 of *LNCS*, pages 291–305. Springer, 2005.
27. A. Girard, C. L. Guernic, and O. Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In J. P. Hespanha and A. Tiwari, editors, *HSCC*, volume 3927 of *LNCS*, pages 257–271. Springer, 2006.
28. A. Girard and G. Zheng. Verification of safety and liveness properties of metric transition systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(S2):54, 2012.
29. M. R. Greenstreet. Verifying safety properties of differential equations. In *Computer Aided Verification*, pages 277–287. Springer, 1996.
30. C. L. Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *LNCS*, pages 540–554. Springer, 2009.
31. N. Halbwachs, Y.-E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *International Static Analysis Symposium, SAS'94*, Namur (Belgium), September 1994.
32. T. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, pages 110–122, 1997.
33. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. In O. Grumberg, editor, *CAV*, volume 1254 of *LNCS*, pages 460–463. Springer, 1997.
34. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.
35. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.
36. P.-H. Ho. *Automatic Analysis of Hybrid Systems*. PhD thesis, Cornell University, Aug. 1995. Technical Report CSD-TR95-1536.
37. A. A. Julius, G. E. Fainekos, M. Anand, I. Lee, and G. J. Pappas. Robust test generation and coverage for hybrid systems. In *Hybrid Systems: Computation and Control*, pages 329–342. Springer, 2007.
38. W. Kühn. Rigorously computed orbits of dynamical systems without the wrapping effect. *Computing*, 61(1):47–67, 1998.
39. A. B. Kurzhanski and P. Varaiya. *Dynamics and Control of Trajectory Tubes*. Springer, 2014.
40. A. A. Kurzhanskiy and P. Varaiya. Ellipsoidal toolbox (et). In *Decision and Control, 2006 45th IEEE Conference on*, pages 1498–1503. IEEE, 2006.
41. C. Le Guernic. *Reachability analysis of hybrid systems with linear continuous dynamics*. PhD thesis, Université Grenoble 1 - Joseph Fourier, 2009.
42. A. V. Lotov, V. A. Bushenkov, and G. K. Kamenev. *Interactive Decision Maps*, volume 89 of *Applied Optimization*. Kluwer, 2004.
43. O. Maler. Algorithmic verification of continuous and hybrid systems. In *Int. Workshop on Verification of Infinite-State System (Infinity)*, 2013.
44. MapleSoft. Maplesim 7: Advanced system-level modeling. `http://www.maplesoft.com/products/maplesim`, 2015.
45. MathWorks. Mathworks simulink: Simulation et model-based design, Mar. 2014. `www.mathworks.fr/products/simulink`.

46. S. E. Mattsson, H. Elmqvist, and M. Otter. Physical system modeling with modelica. *Control Engineering Practice*, 6(4):501–510, 1998.
47. P. Prabhakar and M. Viswanathan. A dynamic algorithm for approximate flow computations. In E. Frazzoli and R. Grosu, editors, *HSCC*, pages 133–142. ACM, 2011.
48. W. H. Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge University Press, 2007.
49. S. Sankaranarayanan, T. Dang, and F. Ivančić. Symbolic model checking of hybrid systems using template polyhedra. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 188–202. Springer, 2008.
50. P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.
51. F. Zhang, M. Yeddanapudi, and P. Mosterman. Zero-crossing location and detection algorithms for hybrid system simulation. In *IFAC World Congress*, pages 7967–7972, 2008.

Model Checking and Model-based Testing in the Railway Domain

Anne E. Haxthausen[1] and Jan Peleska[2]

[1] DTU Compute
Technical University of Denmark
E-mail: aeha@dtu.dk
[2] Department of Mathematics and Computer Science,
University of Bremen
E-mail: jp@informatik.uni-bremen.de

Abstract. This chapter describes some approaches and emerging trends for verification and model-based testing of railway control systems. We describe state-of-the-art methods and associated tools for verifying interlocking systems and their configuration data, using bounded model checking and k-induction. Using real-world models of novel Danish interlocking systems, it is exemplified how this method scales up and is suitable for industrial application. For verification of the integrated HW/SW system performing the interlocking control tasks, a model-based hardware-in-the-loop testing approach is presented. The trade-off between complete test strategies capable of uncovering every error in implementations of a given fault domain on the one hand, and on the other hand the unmanageable load of test cases typically created by these strategies is discussed. Pragmatic approaches resulting in manageable test suites with good test strength are explained. Interlocking systems represent just one class of many others, where concrete system instances are created from generic representations, using configuration data for determining the behaviour of the instances. We explain how the systematic transition from generic to concrete instances in the development path is complemented by associated transitions in the verification and testing paths.

1 Introduction

Background. Railway control systems represent an example of cyber-physical systems, as they are computational entities controlling physical objects like trains, points, and signals, using a distributed communication topology. Their task is to ensure safe train movements through railway networks. In Europe, railway control systems must adhere to the CENELEC standards: the standard EN50128 [13], for example, applies to the development of software in railway control systems. For software of the highest criticality, it requires the application of formal specification and design models and formalised, justified verification and validation (V&V) activities to be performed. The objective of such formalisations is to ensure that potential safety breaches caused by invalid configuration

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_4,

data or erroneous control algorithms can be identified in a systematic way. If formal methods application can also be "mechanised" by means of suitable tools, this contributes to the efficiency of V&V in a considerable way, provided that the tools are qualified, so that one can rely on their verification results.

An overview of trends in formal methods applications to railway signalling can be found in [6, 14]. As of today, three major formal verification approaches are applied to railway control systems: *theorem proving*, *global model checking*, and *bounded model checking*. Theorem proving has been successfully applied to industrial cases, see, e.g. [3] and [16], but has the disadvantage of not being fully automated. Global model checking has the advantage of being fully automated, but has the disadvantage that it may lead to state space explosions, cf., e.g. [15], where a systematic study of applicability bounds of the symbolic model-checker NuSMV and the explicit model checker SPIN showed that these popular model checkers could only verify railway interlocking systems for small railway networks. Bounded model checking (BMC) only investigates model properties in the neighbourhood of a given set of model states, exploring this neighbourhood for a limited number of transition steps, using SAT-based or SMT-based constraint solving techniques to find witnesses for given properties [4]. This avoids explicit and symbolic (BDD) model representations: global model behaviour is specified in propositional form by the transition relation, and concrete behaviours are calculated by means of solving the unrolled transition relation in conjunction with a proposition encoding the verification objective or its negation. In theory, BMC can also prove global model properties, if the transition relation is unrolled for a sufficient number of steps [5]. In practise, however, this will also frequently lead to exhaustion of memory resources or to unacceptable execution times, because the complexity of SAT or SMT solving tasks may become exponential in the number of unrolling steps.

Testing is of course an integrated standard activity of the V&V cycle for railway control systems. There is, however, currently an increasing interest in *model-based testing (MBT)*, because the possibility to derive test cases, test data, test procedures, and traceability information from test models in an automated way [25, 7] offers substantial improvements of both testing efficiency and quality [23]. MBT depends on formal methods in a crucial way: (1) precise model semantics is needed in order to calculate concrete test data suitable to check a given test objective, and (2) the justification of test strategies (typically their *test strength*, i.e. their capabilities to uncover errors in the system under tests (SUT)) strongly relies on mathematical modelling and theorem proving.

Objectives. In this contribution, a formal methods approach to modelling and automated V&V of modern interlocking systems in the framework of the *European Train Control System (ETCS)* is described. Both system designs and V&V methods exploit the generic "production line" character of state-of-the-art interlocking systems: interface types and control algorithms are typically designed in a re-usable way, and this generic system is instantiated with concrete data describing the railway network to be controlled and the interlocking tables defining

the concrete train routes through the network. On the V&V side, generic proof obligations and test objectives can be identified once and for all, and these are automatically transformed into concrete verification conditions and test cases during the instantiation process.

We use BMC to verify that the re-usable design instantiated with the concrete configuration data results in a system whose behaviour fulfils the concrete safety and user requirements. Due to the complexity considerations described above, we apply BMC only to verify whether a property violation can occur within k transition steps from some initial state. However, for the verification of safety properties it is needed to prove that they hold in *all* reachable states. Therefore we combine BMC with inductive reasoning; this technique is called *k-induction*. It allows to verify *global* model properties, because BMC is just used to prove the induction step, for which an investigation of k transition steps in the vicinity of a safe starting point suffices.

With the globally verified concrete system model at hand, the testing process can use this model as a *test oracle*, that is, as a reference for checking the actual system behaviour observed during tests. Moreover, the model can be used to identify the "relevant" test cases needed to check the conformance of system behaviour with the model. We first prove the existence of *complete test suites* that are capable of detecting every deviation of system behaviour from the required behaviour specified in the model, provided that the deviations lie in a certain class of possible behaviours, called the *fault domain*. Complete test suites, however, are usually too big to be performed with acceptable effort. It is therefore shown how global test objectives can be decomposed into separate functional tests, and how equivalence class testing strategies can be applied to reduce test suites to manageable sizes.

Our approach is illustrated by means of a case study that has been derived from novel interlocking system designs currently implemented for Denmark's novel ETCS high-speed railway network.

V&V-Workflow. The V&V-approach sketched above induces the workflow depicted in Fig. 1, which is described in a more refined manner in the sections below. Interlocking systems and the behaviour of trains moving through railway networks can be modelled in a generic way, so that each concrete behavioural model can be instantiated using a model generator that inputs the generic model and the concrete configuration data consisting of network and route descriptions. The concrete model is then to be verified with respect to the safety of the specified routes and the route control algorithm. To this end, all safety-related verification conditions are created for the concrete model according to a generic "recipe". The verification conditions are safety properties expressible as temporal logic formulas. Their validity is globally verified by a well-known inductive proof concept, where the base case and the induction step can be discharged in a mechanised way by means of bounded model checking.

At this stage, we have a formal behavioural model of the interlocking system and its railway network, whose safety properties have been formally established.

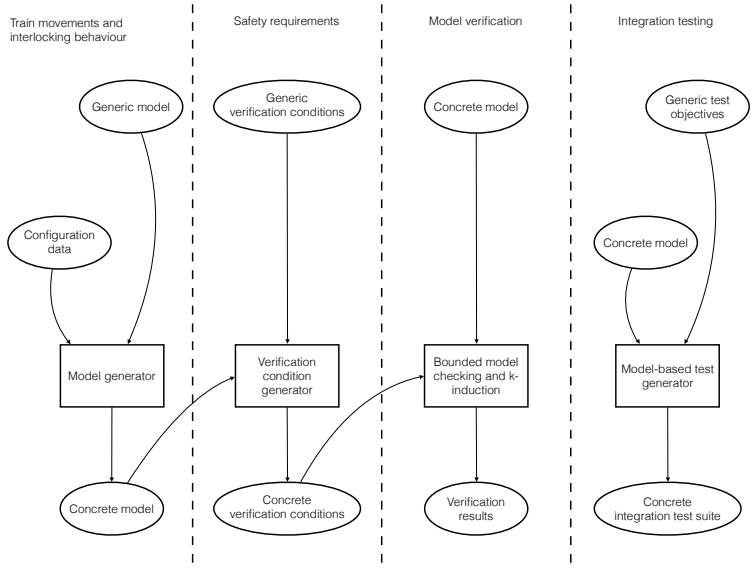


Fig. 1: Workflow associated with the V&V approach presented here.

As a consequence, this model is an ideal reference for both software development and model-based testing of the integrated HW/SW system. In this chapter, we focus on the latter activity. Using the concrete model and, again, a generic recipe for test case generation, integration tests suites with well-defined test strength can be generated.

Section overview. In Section 2 we describe some technical preliminaries; these are needed to introduce the k-induction technique for formal verification of user and safety requirements for railway control systems. In Section 3 we show how this method has been applied to a case study: the future Danish ERTMS/ETCS level 2 based interlocking systems. In Section 4 we describe how to perform model-based testing in the interlocking system domain and discuss suitable testing strategies and heuristics. Finally, a conclusion is drawn in Section 5.

2 Formal Verification

In this section we present a fully automated, formal, model-based method [19] for verifying product lines of railway control systems. In Section 3 we show how this method has been applied to a new product line of the future Danish ERTMS/ETCS level 2 based interlocking systems. The method has also been (partially) applied to a German tramway control system, see [20] and to an existing Danish relay interlocking system [18, 17].

2.1 Verification by Bounded Model Checking and k-Induction

As indicated in Section 1, we apply BMC in combination with k-induction for verifying global safety properties and user requirements of railway interlocking system instances. The idea of this method is to apply bounded model checking to perform a *k-induction* proof. Before we show the method (see [30, 10] for further details), we first explain the mathematical background.

Mathematical Preliminaries. As *behavioural models* of systems, we will use Kripke structures. A *Kripke structure* M is a five tuple (S, s_0, R, L, AP) with finite state space S, initial state $s_0 \in S$, a total transition relation $R \subseteq S \times S$, and labelling function $L : S \to 2^{AP}$, where AP is a set of atomic propositions and 2^{AP} is the power set of AP. The labelling function L maps a state s to the set $L(s)$ of atomic propositions that hold in s.

For the formalising of *desired, global properties* of a model M, we will use core formulas ϕ in propositional logic over the set of atomic propositions AP of M. The set of these core formulas (propositions) is the least set satisfying the following rules: (1) If $\phi \in AP$, then ϕ is a formula and called an *atomic proposition*. (2) If ϕ, ψ are formulas, then $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, and $\phi \Rightarrow \psi$ are also formulas. The *satisfaction relation* $\models$ between states $s \in S$ and formula ϕ is the least relation satisfying:

- $s \models \phi$, if $\phi \in L(s)$
- $s \models \neg\phi$, if it is not the case that $s \models \phi$
- $s \models \phi \wedge \psi$, if $s \models \phi$ and $s \models \psi$
- $s \models \phi \vee \psi$, if $s \models \phi$ or $s \models \psi$
- $s \models \phi \Rightarrow \psi$, if $s \models \psi$ whenever $s \models \phi$

When $s \models \phi$ holds, ϕ is said to *hold* in s.

Two states s_i and s_{i+1} are said to be *consecutive* in M, if there is a transition from s_i to s_{i+1}, i.e. $R(s_i, s_{i+1})$ holds. A state s is said to be *reachable* from another state s' in M, if there is a finite path of consecutive states $s', \ldots, s$ in M, starting in s' and ending in s. The *reachable states of* M is the set of states reachable from the initial state s_0 of M. We write $M \models \mathbf{G}\ \phi$ (*"M satisfies globally ϕ"*) to denote that formula ϕ holds in all states of M that are reachable from the initial state s_0.

Typical models have states $s \in S$ assigning specific values to *model variables*. Assume given a finite set V of variable symbols and for each $v \in V$ an associated value domain D_v. Then, formally speaking, the state space is the set of all valuation functions $s : V \to \bigcup_{v \in V} D_v$, such that $s(v) \in D_v$ for all $v \in V$. If S is such a set of variable valuation functions, its initial state can be expressed by the proposition

$$\mathcal{I}(s_0) \equiv \bigwedge_{v \in V} v = s_0(v)$$

The transition relation $R \subseteq S \times S$ can be expressed in propositional form as a proposition Φ over free variables from V and $V' = \{v' \mid v \in V\}$, such that

$$R = \{(s, s') \in S \times S \mid \Phi(s, s')\}$$

where $\Phi(s, s')$ is the proposition Φ with every occurrence of $v \in V$ replaced by its value $s(v)$ valid in state s, and every occurrence of $v' \in V'$ replaced by its s'-value $s'(v)$.

Using the proposition representation of initial state and transition relation, any finite path $s_0, \ldots, s_{k-1}, s_i \in S$ of length k through the model, starting at the initial model state s_0, is identified by a solution of the formula

$$\text{path}(s_0, k) \equiv \mathcal{I}(s_0) \wedge \bigwedge_{i=1}^{k-1} \Phi(s_{i-1}, s_i) \tag{1}$$

Every finite path $s_n, \ldots, s_{n+k}, s_i \in S$ of length $k+1$, starting in an *arbitrary* model state s_n (whether reachable or not), is characterised by the formula

$$\text{path}(k+1) \equiv \bigwedge_{i=n+1}^{n+k} \Phi(s_{i-1}, s_i) \tag{2}$$

The Method. For the problems we are considering in this paper, our goal is to verify $M \models \mathbf{G}\ \phi$, where M is a Kripke model of a railway control system and ϕ expresses safety properties or user requirements that should hold in all reachable states of M. To prove the validity of $M \models \mathbf{G}\ \phi$, the following *k-induction principle* for $k > 0$ can be applied.

Base Case. Prove the following for any path of k consecutive states $s_0, \ldots, s_{k-1}$ in M, starting from the initial state s_0:

ϕ holds for all the states of $s_0, \ldots, s_{k-1}$.

Induction Step. Prove the following for any path of $k+1$ consecutive states $s_n, \ldots, s_{n+k}$ in M of which the first s_n is an arbitrary state (reachable or not from the initial state s_0):

If ϕ holds in the first k states $s_n, \ldots, s_{n+k-1}$, then ϕ must also hold in the $(k+1^{st})$ state s_{n+k}.

Given the model's transition relation R in propositional form Φ and a proposition $\mathcal{I}$ characterising the initial state, the base case can be verified with BMC, by showing that the following formula has no witness, that is, no solution.

$$\text{base\_case\_violation} \equiv \text{path}(s_0, k) \wedge \neg\big(\bigwedge_{i=0}^{k-1} \phi(s_i)\big)$$

(with $\text{path}(s_0, k)$ defined by Equation (1)). If the model checker finds a solution for base_case_violation, this proves the existence of a finite path $s_0, s_1, \ldots, s_{k-1}$ through the model, where ϕ is violated in at least one of its states. If no such witness can be found, this proves the base case. The induction step is proved by using BMC to show that no witness exists for the formula

$$\text{induction\_step\_violation} \equiv \text{path}(k+1) \wedge \big(\bigwedge_{i=0}^{k-1} \phi(s_{n+i})\big) \wedge \neg\phi(s_{n+k})$$

(with path$(k+1)$ defined by Equation (2)). Here the first state $s_n \in S$ is unrestricted, and the solver tries to find a solution of $(k+1)$ consecutive states starting from s_n, such that ϕ holds in $s_n, \ldots, s_{n+k-1}$, but is violated in s_{n+k}. If no such witness can be found, this proves the induction step. Using this induction scheme, there are the following possible cases:

- The base case is violated. Then one can conclude that $\mathbf{G}\phi$ does not hold.
- Both the base case and the induction step are verified (no violations are found). Then one can conclude that $\mathbf{G}\phi$ holds.
- The base case is verified, but the induction step is violated. From this one can't immediately conclude that $\mathbf{G}\phi$ does not hold, as the induction step may fail for sequences starting at an unreachable state, i.e. for sequences that do not correspond to real executions of the model. Only if the first state s_n of the violation trace is reachable one can conclude that $\mathbf{G}\phi$ does not hold.

Hence, due to the third case, the method may give rise to false negatives. As a remedy against this problem the desired property ϕ should be strengthened with an auxiliary property ψ that is false for those unreachable states, s_n, for which the induction step would otherwise fail. One should now use the induction principle to simultaneously prove $\phi \wedge \psi$ instead of only proving ϕ. The auxiliary property ψ is called a *strengthening invariant*.

An alternative to invariant strengthening is to increase k one by one, until reaching a k, where either the base case fails or both the base case and the induction step hold. In order to ensure that this method terminates (and hence becomes complete), one has to add an extra condition

$$\bigwedge_{n \leq i < j \leq n+k} (s_i \neq s_j)$$

to the induction_step_violation formula, cf. e.g. [10]. This extra condition expresses that all the states $s_n, \ldots, s_{n+k}$ are distinct such that only acyclic execution sequences are considered. According to [10], any k-induction proof can be reduced to a 1-induction proof by means of invariant strengthening. In this paper we use the latter method, as our experience showed that already for small values of k, state explosion happens.

2.2 Formal Modelling

In this section, we describe our approach for creating formal models.

Modelling Product Lines. A characteristic feature of railway control systems is the need for producing an individual system with each installation: the requirements for each concrete control system typically depend on individual parameters such as – in case of interlocking systems – the railway network to be controlled and the train routes allowed through that network. Therefore it is a

common practise to design the software as a collection of generic units that can be re-used for many systems by instantiating these units with the individual configuration/application data required for each concrete system. It is evident that formal models of such systems should reflect this generic product line character. Therefore, in our method we will use *generic models that can be instantiated with configuration data*.

Domain-Specific Approach. In recent years, *domain-specific, generative methods* [11] for software development have gained wide interest. One of the main objectives addressed by these methods is the possibility for a given domain to re-use various artifacts when developing software. The re-use of models for a product line of systems can be enabled by developing re-configurable systems as suggested above. Domain-specific methods typically use domain-specific languages and generators for the construction of re-configurable artefact's. An *application generator* is a tool that takes a specification of an application as input and returns an application as output. It yields this application by instantiating the generic part of the application with configuration data that it derives from the specification. The specifications are formulated in a domain-specific language (DSL). In contrast to general-purpose specification and programming languages, a *domain-specific language* is dedicated to a specific application domain by using the terminology of that domain. Hence, it can be used by domain experts who are not specialists in the field of information technology. Typically the 'applications' are software source code represented in a high-level programming language, but they can also be formal models or formal requirement specifications.

Inspired by that, our method suggests to provide a domain-specific language and a generator that from specifications in this language can produce formal models by instantiating a generic model with configuration data. Likewise, there should be a generator that produces formal specifications of required safety properties.

2.3 Method Summary

Our methodological approach results in a domain-specific development and V&V framework for railway control systems comprising the following elements (see Fig. 2).

1. A domain-specific language (DSL).
2. A collection of development tools, including
   (a) a DSL specification editor and well-formedness checker for configuration data,
   (b) a bounded model checker that can perform k-induction,
   (c) a generator that takes a well-formed generic DSL model and a well-formed DSL specification of configuration data as input and produces a behavioural model M of the control system and its physical environment as a Kripke structure with initial condition and transition relation in propositional form,

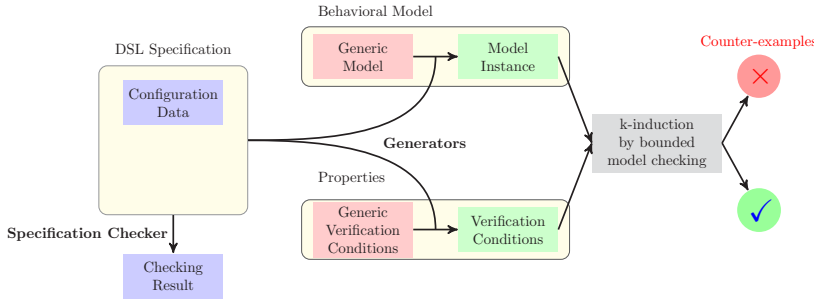


Fig. 2: Toolchain.

(d) a generator that takes a well-formed DSL specification of configuration data and generic verification conditions as input and produces the required verification conditions (safety conditions ϕ and strengthening conditions ψ expressed in the property language of the model checker).

For a given product line, initially the generic model (which is common for all control systems of the product line) is created and checked to be well-formed. Then for each control system to be verified, users should first apply the specification editor to specify the configuration data in the domain-specific language and check the resulting description D by means of the specification checker. Next, the generators should be applied to the generic model and the specification of configuration data to produce (1) a model M of the control system and its physical environment and (2) the required state invariants ϕ and ψ. Finally, the bounded model checker is applied to automatically check the validity of $M \models \mathbf{G}(\phi \wedge \psi)$ by means of k-induction.

3 Interlocking System Verification – a Case Study

In this section we illustrate the verification method described in Section 2.3 by providing a framework for constructing and verifying both generic software and concrete configuration data for a new family of ERTMS/ETCS level 2 based interlocking systems [12]. These systems are going to be deployed in Denmark over the next years until 2021 as part of the Danish Signalling Programme[3]. The work presented in this section is based on the publications [32, 33].

3.1 The Novel Danish Interlocking Systems

Complete ETCS signalling systems consist of a multitude of components, such as interlocking systems, radio block centres, track elements (e.g. points, balises, axle counters), and on-board equipment (e.g. the *European Vital Computer (EVC)* performing automated train protection according to the ETCS protocols). An

[3] `http://www.bane.dk/signalprogrammet`

interlocking system has the task to control the points in a railway network and to set safe train routes through this network according to traffic control requests. In the old signalling systems, interlocking systems also controlled signals placed along the tracks. In the new signalling systems, there are no signals installed along the tracks, but only *marker boards* to show the start and end of routes. The interlocking systems now have a *virtual signal* associated with each marker board. A virtual signal can be OPEN or CLOSED, allowing or disallowing trains to pass the associated marker board. Based on the state of the virtual signals, *movement authorities* (permissions to proceed) are sent via radio block centres to the on-board units in the trains.

3.2 The Domain-specific Language for Interlocking Systems

In our DSL, a description D of a concrete interlocking system configuration consists of a *network diagram* N and an *interlocking table* T. The configuration (N, T) is then complemented by the generic algorithms controlling the allocation of train routes through the network.

Network Diagrams. A network diagram N outlines the geographical arrangement of the tracks and track-side equipment. Fig. 3 shows an example of a network diagram for a typical smaller station. From the diagram it can be seen that the station has six linear sections (`b10,t10,t12,t14,t20,b14`), two points (`t11,t13`), and eight marker boards (`mb10,...,mb21`). A linear section is a section with up to two neighbours: one at the *up* end, and one at the *down* end.[4] The linear section `t12` in Fig. 3, for example, has `t13` and `t11` as neighbours at its up end and down end, respectively. A point can have up to three neighbours: one at the *stem*, one at the *plus* end, and one at the *minus* end. Point `t11` in Fig. 3, for example, has `t10`, `t12`, and `t20` as neighbours at its stem, plus, and minus ends, respectively. Linear sections and points are collectively called *(train detection) sections*, as they are each provided with train detection equipment which the interlocking system uses to determine whether the section is occupied by a train or not. A point can be switched between two positions: PLUS and MINUS. When it is in the PLUS/MINUS position, the *stem* is connected to the *plus*/*minus* end. Along each linear section, up to two marker boards (one for each direction) can be installed. A marker board can only be seen in one direction and is used as reference location (for start and end of routes) for trains going in that direction. For instance, `mb13` in Fig. 3 is installed along section `t12`, and it is intended for travel direction up.

Interlocking Tables. An interlocking table T specifies the routes in the given network diagram N and the conditions (to be used by the interlocking system)

[4] In Denmark, *up* and *down* denote the directions in which the distance from a reference location is *increasing* and *decreasing*, respectively. The reference location is the same for both up and down, e.g. an end of a line.

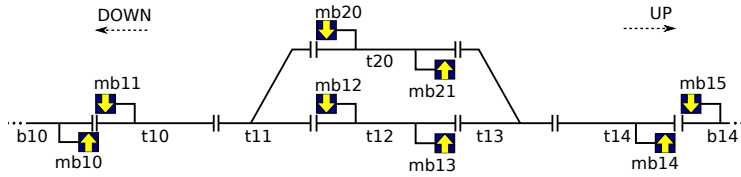


Fig. 3: An example railway network layout.

for setting these routes. In railway signalling terminology, *setting* a route denotes the process of allocating the resources – i.e. linear sections, points, and virtual signals – for the route, and then locking the route exclusively for only one train.

Interlocking tables (see Table 1) have one row for each route r. Each row has the following fields. Field **id** contains the route identification r, field **source** states its source marker board $\mathbf{src}(r)$, field **dest** states its destination marker board $\mathbf{dst}(r)$, and field **path** specifies the sequence $\mathbf{path}(r)$ of track sections associated with r. Moreover, field **overlaps** specifies the sections $\mathbf{overlaps}(r)$ which are located beyond the destination marker board and must be vacant in order to avoid collisions in case a train should fail to stop. Field **points** describes the set $\mathbf{points}(r)$ of points associated with the route. This includes points in the path and overlap, and points used for flank and front protection. For each point p associated with r, its required position $\mathbf{req}(r, p)$ to be enforced when allocating r is specified. The table field **signals** specifies the set $\mathbf{signals}(r)$ of virtual signals that must be CLOSED for flank or front protection of the route. The field **conflicts** describes the set $\mathbf{conflicts}(r)$ of routes *conflicting* with r: if two routes require the same point to be in different positions, or if the routes overlap such that concurrent use could lead to train collisions, they are considered to be conflicting. The set **Route** contains all route identifications in the interlocking table T. Set **Signal** denotes all marker boards, **Point** all points, and **LSection** all linear sections occurring in T.

Table 1: Excerpt of the interlocking table for the network layout in Fig. 3.

id	**source**	**dest**	**points**	**signals**	**path**	**conflicts**
1a	mb10	mb13	t11:p;t13:m	mb11;mb12;mb20	t10;t11;t12	1b;2a;2b;3;4;5a;5b;6b;7
..	...	...	...	...	...	...
7	mb20	mb11	t11:m	mb10;mb12	t11;t10	1a;1b;2a;2b;3;5b;6a

The overlap column is omitted as it is empty for all of the routes.
'p' means PLUS, 'm' means MINUS.

Table 1 shows an excerpt of an interlocking table for the network shown in Fig. 3. As can be seen, one of the routes has id `1a`, goes from `mb10` to `mb13` via three sections `t10`, `t11` and `t12`, and has no overlap. It requires point `t11` (in its path) to be in PLUS position and point `t13` (outside its path) to be in MINUS position (as a protecting point). The route has `mb11`, `mb12` and `mb20` as

protecting signals, and it is in conflict with routes `1b`, `2a`, `2b`, `3`, `4`, `5a`, `5b`, `6b`, and `7`.

3.3 Framework Implementation

Following the method in Section 2.3, we have implemented a framework of tools centered around the domain-specific language. Fig. 4 illustrates the architecture of the implemented framework.

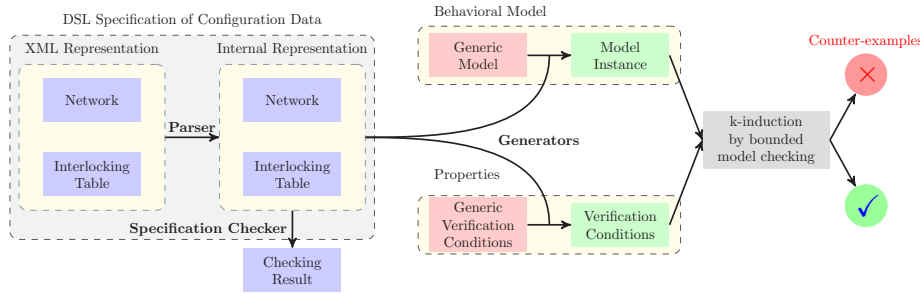


Fig. 4: Implemented framework.

The domain-specific language for configuration data has been given an XML representation as well as a graphical representation. A graphical editor has been developed and from this the XML representation can be exported. A parser from the XML representation into an internal DSL representation has been implemented as well. The generator tools have been implemented to take as input an internal DSL representation and return an internal representation of a Kripke model M and an internal representation of state invariants ϕ and ψ, respectively. As model checker, we use the bounded model checker of the RT-Tester model-based testing and bounded model checking tool, described in more detail in [25]. This model checker takes as input M[5], ϕ, and ψ, and performs k-induction using the SONOLAR SMT solver described in [28].

3.4 Generated Models

Given a domain-specific description D consisting of a network layout N and an interlocking table T, the model generator returns a representation of a Kripke Structure $M = (S, s_0, R, L, AP)$ modelling the behaviour of the interlocking system and its operational environment. Below we will outline the state space S, the initial state s_0, and the transition relation R of M.

[5] Precisely speaking, it takes the variable symbols $v \in V$, their domains D_v, as well as the initial condition $\mathcal{I}$ and the transition relation $\mathcal{R}$ in propositional form as input.

State Space. As described above, every state is a valuation function $s : V \to \bigcup_{v \in V} D_v$, such that $s(v) \in D_v$ for all variable symbols $v \in V$. Every variable domain D_v is a finite subset of $\mathbb{N}_0$. Below, we describe how the state space is constructed from the set of sections (points and linear sections), the set of marker boards in the network description N, and the set of routes in the interlocking table T:

Point Positions. For each point p in N there are two variables:

- p.POS – representing the actual position of p
- p.CMD – representing the position of p, as requested by the interlocking

The value of p.POS can be one of the following: PLUS(0), MINUS(1), or INTERMEDIATE(2), the latter representing the transition phases PLUS $\leftrightarrow$ MINUS. The value of p.CMD can only be PLUS or MINUS.

Virtual Signal Aspects. For each marker board s in N, there is an associated virtual signal s (with the same name as the marker board) for which there are two variables:

- s.ACT – representing the actual aspect of s, as "seen" by the train
- s.CMD – representing the aspect of s, as requested by the interlocking

The values of these variables can be one of the following: OPEN(0) or CLOSED(1). The actual aspect of a virtual signal is an abstraction of the information transmitted to the train: Only if the virtual signal is OPEN, the train is allowed to pass the associated marker board. The values of the two variables may differ due to the delay in the communication between the interlocking system and the trains.

Section Occupancy Status. For each linear section l in N, there are two variables, each representing an occupancy status:

- l.D2U representing the occupancy status of l in direction up
- l.U2D representing the occupancy status of l in direction down

and for each point p in N, there are three variables:

- p.S2PM representing the occupancy status of p in direction stem to plus or minus
- p.P2S representing the occupancy status p in direction plus to stem
- p.M2S representing the occupancy status p in direction minus to stem

Each of these variables is represented by an integer FRO encoded by 3 bits F, R, and, O. Seven out of eight possible bit combinations are used, as specified in Table 2. We distinguish between six ways a section can be occupied by trains in the given direction: four ways a single train can occupy the section and two ways it can be occupied by two trains travelling in the same direction. If, for a given direction, a section has occupancy status 111 or 011 (i.e. a single train is

Table 2: Representation of occupancy statuses.

FRO	represented occupancy status
000	vacant
101	occupied by the front, but not by the rear of a train
011	occupied by the rear, but not by the front of a train
111	occupied by both the front and by the rear of a train
001	occupied by a train, but neither by its front nor by its rear
010	occupied by two trains: a rear-end collision
110	occupied by two trains: a rear-end collision

occupying the section and its rear has left the previous section) and a second train going in the same direction is entering the section from the previous section, a rear-end collision, represented by occupancy statuses 010 and 110, respectively, will happen. Head-on collisions are represented by non-zero values of more than one occupancy status variable associated with a linear section or point. If, for example, l.D2U $\in \{101, 111\}$ and at the same time l.U2D $\in \{101, 111\}$, this represents a head-on collision situation in section l.

Using the bit vector representation has the advantage that some transition rules and safety properties can be expressed efficiently using arithmetic bit-wise operations on the occupancy statuses, as shown in Sections 3.5 and 3.6.

Section Modes. For each section e, there is a variable e.MODE representing the mode of e: FREE(0), EXLCK(1) (the section is *exclusively* locked for a route), or IN-USE(2) (the element is occupied by a train).

Route Modes. For each route r, a variable r.MODE is used to represent the current mode of that route. A route can be in one of the following modes: FREE(0), MARKED(1), ALLOCATING(2), LOCKED(3), or IN-USE(4). The use of these modes will be explained further below, in connection with Fig. 5.

Initial State. The initial state s_0 is the (safe) state in which all points are in PLUS position and requested to be so, all virtual signals are CLOSED and requested to be so, all detection sections are vacant and FREE, and all routes are FREE. With the state representation introduced above, this means that s_0 is the state in which all variables are evaluated to 0. In propositional form, this

is expressed by

$$\begin{aligned}\mathcal{I} \equiv & \bigwedge_{s \in \mathbf{Signal}} s.\mathrm{ACT} = s.\mathrm{CMD} = 0 \wedge \\ & \bigwedge_{p \in \mathbf{Point}} p.\mathrm{POS} = p.\mathrm{CMD} = p.\mathrm{S2PM} = p.\mathrm{P2S} = p.\mathrm{M2S} = p.\mathrm{MODE} = 0 \wedge \\ & \bigwedge_{l \in \mathbf{LSection}} l.\mathrm{D2U} = l.\mathrm{U2D} = l.\mathrm{MODE} = 0 \wedge \\ & \bigwedge_{r \in \mathbf{Route}} r.\mathrm{MODE} = 0\end{aligned}$$

Transition Relation. The transition relation $R \subseteq S \times S$ is represented in propositional form by $\Phi(s, s')$ as explained above. Below we explain how Φ is constructed.

The general principle is as follows: First propositions (also called transition rules) for basic transitions are defined. They take the form *guard* $\wedge$ *effect*, where *guard* is a proposition over the variables in V and *effect* is a proposition over free variables from $V \cup V'$. Then these propositions are combined. If two propositions ϕ and ψ are describing transitions that have the same priority, their combined behaviour is described nondeterministically by the proposition $\phi \vee \psi$. If the transitions of ϕ have higher priority than those described by ψ, their combined behaviour (for which we use the notation $\phi\ [>]\ \psi$) is described by the proposition $\phi \vee (\neg g_\phi \wedge \psi)$, where g_ϕ is the disjunction of the guards of the transition rules in ϕ. Hence, in this combination, transitions in ψ can't happen, when some transitions in ϕ are enabled.

To follow the principle explained above, we first divide the transitions in the model into four classes, such that the transitions in each class have the same priority, and for each class we define a proposition describing the combined behaviour of the transitions in that class. The classes and their propositions are as follows:

1. Φ_d for the combination of all route dispatching transitions (as done by the interlocking system).
2. Φ_i for the combination of all other route control transitions (as done by the interlocking system).
3. $\Phi_{elements}$ for the combination of all transitions switching the actual position of points (as done by point machines) and all transitions transmitting virtual signals to trains so the virtual signals change their actual aspects (done by radio communication over a radio block centre).
4. Φ_{trains} for the combination of all transitions changing the section occupancy status for train detection sections (due to train movements).

With these considerations, we define

$$\Phi = \Phi_d \vee (\Phi_i\ [>]\ (\Phi_{elements}\ [>]\ \Phi_{trains}))$$

Route dispatching as expressed by Φ_d can happen at any time in a nondeterministic way. The internal transitions Φ_i of the route controller happen much faster than the transitions involving state changes in track elements. Therefore Φ_i has higher priority than the transitions involving track elements. The latter are again faster than train movements along the routes, so $\Phi_{elements}$ has higher priority than Φ_{trains}. This prioritisation allows us to abstract from physical time, while excluding unrealistic transition sequences that could never happen due to physical constraints.

In the following paragraphs, we outline the transition rules for the transitions in the four classes above.

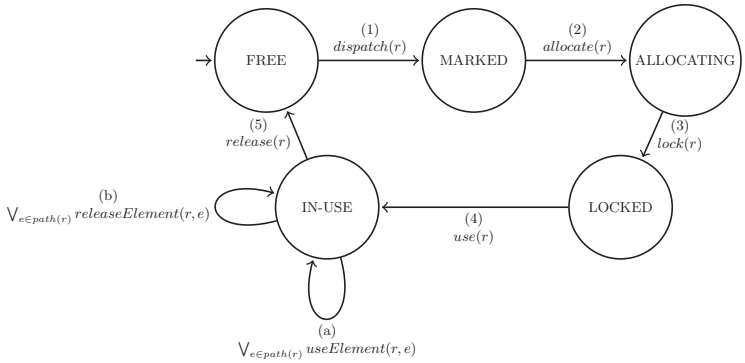


Fig. 5: Life-cycle of the route controller for route r, showing how the value of r.MODE is changed.

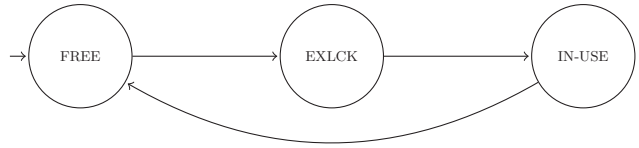


Fig. 6: Life-cycle of the mode e.MODE of a section e.

Transition Rules for the Interlocking System. The interlocking system has a *route controller* for each route r in the interlocking table. Fig. 5 shows the "life-cycle" of a route controller for a route r. It goes through five modes represented by r.MODE as follows.

(0) Initially the route mode is FREE.

(1) *dispatch(r)*: If the route controller is in mode FREE and receives a request to set the route, the route is dispatched, i.e. its mode is set to MARKED.
(2) *allocate(r)*: In the MARKED mode, the route controller continuously checks the status of different track-side elements to detect conditions for starting the allocation of the route: it checks whether (1) the sections of the path and the overlap of the route are vacant, (2) none of the conflicting routes r' are in mode ALLOCATING or in mode LOCKED, and (3) each section in the path of the route is in mode FREE, each section in the overlap of the route is in mode FREE or in mode EXLCK, and each protecting point is in mode FREE or in mode EXLCK and locked in the required position. This condition check is performed as an atomic action, so that route controllers of conflicting routes cannot come to the simultaneous conclusion that their route can be allocated. This is reflected in the transition relation by a disjunctive structure over all routes. When these conditions are met, the route controller commands the protecting signals of the route to close and it commands points used by the route to be switched to their required positions as stated in the interlocking table. It changes the mode to EXLCK for each section e in r's path, and the route mode of r is set to ALLOCATING.
(3) *lock(r)*: In the ALLOCATING mode, the route controller constantly monitors the status of the track-side elements. When the signals and points have changed their actual states as commanded in the previous step, the source signal of the route is commanded to OPEN, allowing a train to enter the route, and the route mode is set to LOCKED.
(4) *use(r)*: In the LOCKED mode, when the actual aspect of the source signal of the route has changed to OPEN and a train enters the route, the source signal of the route is commanded to be CLOSED preventing other trains from entering, and the route mode is set to IN-USE.
(5) *release(r)*: In the IN-USE mode, the sections of the route are sequentially being used (transition (a)) and released again (transition (b)) as explained further below. The route is *released*, i.e. the route mode is set to FREE, when the train has finished using the route, i.e. the train has passed the destination marker board, or the train has come to standstill in front of this.

While the route controller is executing, each of the sections e of the route goes through a life cycle with three modes as shown in Fig 6. Initially the mode is FREE. In transition (2) in Fig. 5 the mode of e is changed to EXLCK. When the route controller is in mode IN-USE and the train enters section e, the mode of e is changed to IN-USE (by transition (a)), and when the train leaves section e, the section is released (by transition (b)), so the mode of e is changed to FREE. Hence, the sections of the route are sequentially used and sequentially released as the train moves along the route.

For each of the transitions ((1) – (5), (a) and (b)) explained above there is a proposition (see the labels on the arrows in Fig. 5) specifying the transition. Transition (1), for example, has the proposition $dispatch(r)$ specified by

$$r.\mathrm{MODE} = \mathrm{FREE} \wedge r.\mathrm{MODE}' = \mathrm{MARKED}$$

It states that a route can be dispatched arbitrarily when its mode is FREE. This is an over approximation (on the safe side): in the real world, a route can only be dispatched when requested by an approaching train.

Transition Rules for Points. For each point p there are two transition rules for switching the actual position of the point:

$$p.\text{POS} \neq p.\text{CMD} \ \wedge \ p.\text{POS} \neq \text{INTERMEDIATE} \wedge \ p.\text{POS}' = \text{INTERMEDIATE}$$

and

$$p.\text{POS} = \text{INTERMEDIATE} \wedge p.\text{POS}' = p.\text{CMD}$$

The first states that when the actual position of the point p differs from the requested position and is not the *intermediate* position, the point is switched to the *intermediate* position, and the second states that when it is in *intermediate* position, it switches to the requested position.

Transition Rules for Virtual Signals. For each virtual signal s there is one transition rule Φ_s for changing the actual aspect (as seen by the train):

$$s.\text{ACT} \neq s.\text{CMD} \wedge s.\text{ACT}' = s.\text{CMD}$$

It states that when the actual aspect $s.\text{ACT}$ of a signal s differs from the commanded aspect $s.\text{CMD}$, the actual aspect of the signal is set to the commanded aspect.

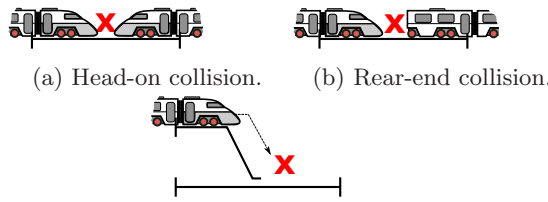
Transition Rules for Train Movements. Trains do not occur as explicit state components of our model. Instead, train *movements* are implicitly modelled via changes in the occupancy status of train detection sections, inspired by the "rubber-band" model described in [1]. This implicit model is advantageous when compared to the explicit one, because it covers an arbitrary number of trains of arbitrary length. In the implicit model of train movements, train length – in terms of numbers of sections that a train occupies – may vary as trains move. This variation reflects the actual view of interlocking systems of the train length: although trains have fixed geometric length, their length – in terms of the number of sections that they occupy – as seen by the interlocking systems is not fixed.

It is out of the scope to show all the transition rules for changing the occupancy status of track sections. Here we will just mention that there are rules expressing conditions and effects of the following kinds of events: the front or the rear of a train enters a boarder section, leaves a boarder section, or leaves one section and enters the next section in the travel direction of the train. The transition rules reflect the following assumptions about train behaviour:

- Trains enter/leave only the network at/from boarder sections.
- Trains have a driving direction which is not changed while using a route.

- Trains follow the tracks (without jumping) and move in their given travel direction.
- Trains do not pass boarder marks for which the actual aspect of the associated virtual signal is CLOSED.
- Trains do not split.

3.5 Generated Safety Conditions



(a) Head-on collision. (b) Rear-end collision.

(c) Derailment.

Fig. 7: Cases of hazardous situations.

Given a domain-specific description D, the verification conditions generator returns a state invariant condition ϕ which is a proposition over the state variables given in Section 3.4.

The proposition expresses conditions for a state to be safe, i.e. it expresses that there are no hazardous situations that can lead to train collisions and train derailments. Fig. 7 shows three types of hazardous situations:

(a) *Head-on collision*: the case where two trains are approaching each other on the same track section,
(b) *Rear-end collision*: the case where two trains are following each other on the same track section, and
(c) *Derailment*: the case where a train enters a point which is not in right position for the train movement or the point switches while the train is driving through the point.

The state invariant ϕ is of the form

$$\phi = \neg(\bigvee_{l:Linear} Hazard_l \vee \bigvee_{p:Point} Hazard_p) \tag{3}$$

where $Hazard_l$ and $Hazard_p$ are propositions specifying conditions for hazards to occur on a linear section l and a point p, respectively. These propositions are disjunctions of sub-propositions expressing hazards of the types stated above:

$$Hazard_l = HeadOnCollision_l \vee RearCollision_l \tag{4}$$

$$Hazard_p = HeadOnCollision_p \vee RearCollision_p \vee Derailment_p \tag{5}$$

The sub-propositions are expressed in terms of the track occupancy variables. For instance:

$$HeadOnCollision_l = l.\text{D2U} \cdot l.\text{U2D} > 0 \tag{6}$$

As $l.\text{D2U} \cdot l.\text{U2D} > 0$ iff $l.\text{D2U} > 0$ and $l.\text{U2D} > 0$, the formula expresses that the section is occupied in both down-to-up ($l.\text{D2U} > 0$) and up-to-down ($l.\text{U2D} > 0$) directions.

3.6 Invariant Strengthening

Given a domain-specific description D, the verification conditions generator also returns a state invariant condition (proposition) ψ for invariant strengthening as explained in Section 2.1. ψ is the conjunction of many propositions expressing properties that must hold in reachable states. ψ has been chosen such that it is false for unreachable states for which the k-induction with k=1 fails. An example of such strengthening properties is given in the following.

Train Integrity. Some states of the variables expressing the train occupancy status of the track sections are not feasible as they correspond to situations that are not physically possible. An example of an infeasible state is one in which the variables express that a section e is occupied in one direction by a train without the front being on the section, but the next section in that travel direction is unoccupied.

The train integrity conditions can be formalised as a conjunction of formulas over the track occupancy variables. For each travel direction (*up* and *down*), there is a formula for each section e that has a next section in the given travel direction. The pattern of such a formula depends on the other sections the current section is connected to in the given travel direction. For instance, for travel direction *up* and a linear section l that has a linear section l' as neighbour in travel direction *up*, the formula will take the following form:

$$(l.\text{D2U} \;\&\; 0b101) = 0b001 \text{ iff } (l'.D2U \;\&\; 0b011) = 0b001 \tag{7}$$

where & is the *bit-wise and* operator. This formula expresses that section l is occupied by a train in direction *up* (the O bit of l.D2U is 1) without the front being on the section (the F bit of l.D2U is 0), if and only if section l' is occupied by a train in direction *up* (the O bit of $l'.D2U$ is 1) without a rear being on the section (the R bit of $l'.D2U$ is 0). Formula (7) shows the expressiveness of our state encoding, allowing properties to be efficiently represented in compact formulas.

3.7 Verification Experiments

We have used the tool-chain to verify the safety properties $\mathbf{G}\phi$ for model instances of a number of railway networks, ranging from a trivial toy network to a large station (Køge) extracted from the early deployment line of the new Danish

Table 3: Verification results for different cases of networks.

Case	Linears	Points	Signals	Routes	States	Time	Memory
Branch	6	1	6	4	10^{32}	2	63
Junction	8	2	8	10	10^{55}	9	137
Station	6	2	8	12	10^{53}	11	128
Gadstrup-Havdrup	21	5	24	33	10^{156}	146	626
Køge	57	23	60	73	10^{429}	3868	4457

Branch, Junction and Station are invented networks (of which Station is the example of the typical smaller station shown in Fig. 3), while Gadstrup-Havdrup and Køge are stations extracted from the early deployment line in the Danish Signalling Programme. Time is given in seconds and memory in MB.

signalling systems. In all cases it was sufficient to make k-induction with $k = 1$, when using the invariant strengthening explained in Section 3.6. An example where k-induction was applied with $k > 1$ can be found in [26].

Table 3 shows the results of these verification experiments. For each case, the table gives the size of a network in terms of the number of linear sections, points, signals, and routes in the configuration, and the approximate number of generated states in the corresponding model instance. The two last columns show the approximate accumulated verification time and memory usage. All experiments have been performed on Intel(R) Core(TM) i7-3520M CPU @ 2.90GHz, 8GB RAM, Ubuntu 14.04 LTS, Linux 3.14.1-031401-generic x86_64 kernel.

We also injected errors into models. Counter examples for these were normally found in relatively short time. This appears to be a general trend when dealing with interlocking systems [22]. In a few cases, it took long time to find counter examples. Such examples usually represent very subtle errors in the model or the configuration data, which may be easily overlooked by inspection.

4 Model-based Testing

4.1 Model-based Testing Terminology

Test cases, oracles and procedures. The dynamic testing activities for a given test campaign are called *test suites*, and these are structured into sequences of *test cases*. A test case is a specification fragment serving a *test objective*, that is, a goal to examine a specific aspect of SUT behaviour. Test cases are associated with

- specification of inputs to the SUT that are suitable for investigating the test objective under consideration,
- specification of the expected behaviour of the SUT in response to these inputs, and
- references to the SUT requirements that are (partially) tested by these inputs.

When testing reactive systems, inputs usually consist of *input traces*, that is, finite sequences of timed input vectors, each vector stimulating the SUT at a certain point in time of the test execution. The execution of one or more test cases is performed by *test procedures*. The relations between requirements, test cases, test procedures and results are called *traceability data*.

Model-based testing (MBT) is a variant of dynamic testing, where the SUT is executed, and its behaviour is checked against the expected behaviour specified by a *test model*. These checkers are usually called *test oracles*. Models are associated with a behavioural semantics from which the set of *computations*, that is, infinite paths of consecutive model states that can be visited during a model execution, can be calculated. Using links to syntactic model elements or to property specifications identifying sets of computations, requirements are associated with these computation sets. Links to syntactic elements can also be represented by property specifications. A suitable logic for specifying properties is, for example, *Linear Temporal Logic (LTL)* [9], because the models of LTL formulas are computations.

Since requirements R are linked to sets of model computations, and the latter are characterised by properties ϕ, it is possible to derive *requirements-based test cases* from the model in an automated way: a *symbolic test case* for R is a property ψ satisfying $\psi \Rightarrow \phi$. This means, that the property ψ characterising the symbolic test case specifies a subset of the computations representing requirement R in the model. A finite computation prefix satisfying ψ is then a *concrete test case*; restricting the computation to its input vectors results in the input data to be used when executing this case.

Alternatively, properties can be used to specify symbolic *structural test cases*, whose witnesses cover specific syntactic model portions, such as

- interfaces,
- state machine transitions, or
- operations.

Test data calculation. Calculating witnesses from symbolic test cases can be performed in analogy to the BMC technique introduced in Section 2: formula

$$tc \equiv \text{path}(s_0, k+1) \wedge G(s_0, \ldots, s_k) \tag{8}$$

with $\text{path}(s_0, k)$ defined by Equation (1) specifies all paths of length k of the model that start in s_0, perform k transitions and end in some state s_k, such that the witness $s_0, \ldots, s_k$ also fulfils proposition $G(s_0, \ldots, s_k)$ which encodes the test objective associated with the test case under consideration. In the BMC verification described in Section 2, witnesses for propositions similar to tc represented model errors indicating violations of safety or user requirements. When generating test cases tc, however, witnesses denote concrete test cases, from which the input traces can be extracted, to be sent to the SUT during test execution.

Formula G encodes the symbolic test case in propositional form. When using LTL formulas ϕ to represent symbolic test cases, the *finite BMC encoding* of LTL formulas described in [5] can be used to transform ϕ into propositional form G.

Automation. One of the most attractive features of the MBT approach is that it can be fully automated, once a test model is available. (1) Exploiting the model references to requirements, test cases related to requirements can be automatically identified and refined, using strategies like equivalence class partitioning and boundary value testing. Concrete test data can be calculated for abstract test cases, using the BMC witness generation principle, supported by SMT solvers. (3) Test oracles can be automatically created, since the model already specifies the expected behaviour of the SUT. (4) Test procedures executing concrete test data against the SUT and checking its reactions by means of test oracles can be automatically generated and executed. (5) Traceability data can be automatically compiled by exploiting the known relationships between requirements, model elements, test cases, procedures and results. For the purposes discussed in this chapter, the RT-Tester model-based testing tool has been applied for both BMC and testing. Its automation capabilities are discussed in [25], and special test case generation strategies implemented in RT-Tester are described in [21]. Examples of industrial-size models that can be efficiently handled by the tool have been presented in [23, 27].

Development model versus test model. Model-based testing can be applied in two different ways. If – for example by verification via model checking – the correctness of the reference model is ensured, then code *and* tests can be developed from this model. The objective of testing is then only to verify the consistency between the hardware/software integration and the model. Note that this includes the verification of the object code and of the additions performed by linkers and loaders. This scenario is depicted in Fig. 8.

If, however, there are doubts about the completeness and correctness of the development model, it is advisable to have a second, redundant, model developed by the V&V team, and this is a proper test model. The associated scenario is shown in Fig. 9. The test execution now verifies the consistency of the SUT behaviour with the *test* model. This allows for detecting errors in the development model as well. On the other hand, the effort of this V&V approach is higher, since a separate test model has to be developed. Moreover, debugging the causes of failed test executions has to take into account the potential errors in the test model, in addition to errors in development model and code.

For more details about the topics introduced in this section see [25].

4.2 Overall Test Objectives for the Railway Application

For testing our railway control system application, the model verification discussed in the sections above suggests a one-model MBT approach as described in Fig. 8: it can be assumed that the model of the railway network, the specified routes and their associated tables, as well as the control algorithm applied by the interlocking system is complete and correct. The main test objective is to verify the consistency of the integrated HW/SW system with the model portion specifying the required behaviour of the interlocking system. This model portion

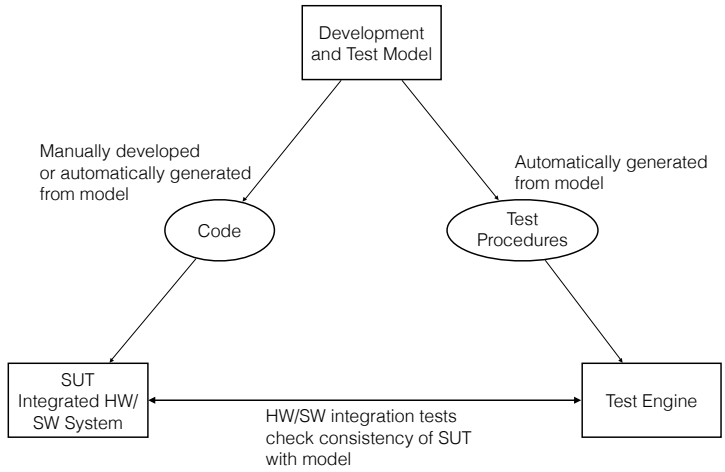


Fig. 8: MBT with joint development and test model.

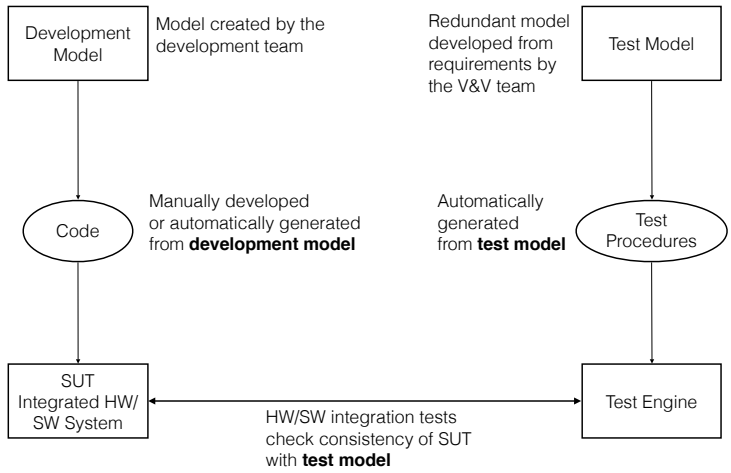


Fig. 9: MBT with separate development and test models.

is represented by the route controller introduced above. The overall test objective is to verify that the integrated HW/SW-system realising the route controller behaves in all situations – that is, in all possible combinations of the network status and for all possible combinations of simultaneous route requests sent to the controller – as its formal sub-model specified above.

4.3 Semantic Domain: I/O State Transition Systems

The V&V activities performed by means of bounded model checking described above used Kripke Structures for modelling the behavioural semantics of the interlocking system (more precisely, its route controller component) and its operational environment, the railway network. This is a *closed* system in the sense that for every interface, both communication partners (readers and writers) were part of the system model. As a consequence, no explicit notion of input and output was needed: the complete system just performed state transitions according to the underlying Kripke Structure's transition relation. For hardware-in-the-loop testing, however, the SUT – that is, the route controller – is stimulated by a test engine simulating the SUT's operational environment and at the same time observing the SUT reactions to the stimulations. To this end, we use a very general semantics that is related to the Kripke Structure semantics used for model checking, but streamlined for testing purposes.

A *state transition system (STS)* is a triple $\mathcal{S} = (S, s_0, R)$ with state space S, initial state $s_0 \in S$ and total transition relation $R \subseteq S \times S$. Obviously, every Kripke Structure is an STS, but with the additional availability of a labelling function mapping states to sets of atomic propositions. Next, we restrict the STS under consideration to those possessing a notion of variable valuations, input, and output: For test purposes, we need to stimulate certain variables of the SUT input interface, and observe SUT reactions at its output interface. At the same time, the SUT may process some internal state variables that cannot be observed during black-box testing.

An *input/output state transition system (IOSTS)* is an STS $\mathcal{S} = (S, s_0, R)$ where states $s \in S$ are valuation functions $s : V \to D$ as introduced in Section 2 for Kripke structures, such that in addition, the variable space V can be partitioned into disjoint sets $V = I \cup M \cup O$ called input variables, (internal) model variables, and output variables, respectively.

In the exposition below, variable symbols are enumerated with the naming conventions $I = \{x_1, \ldots, x_t\}$, $M = \{m_1, \ldots, m_p\}$, $O = \{y_1, \ldots, y_q\}$. The cross product of the input domains is abbreviated by $D_I = D_{x_1} \times \cdots \times D_{x_t}$; D_M and D_O are defined analogously. Application of a state valuation function s to the input vector is written by $s(\boldsymbol{x}) = (s(x_1), \ldots, s(x_t))$, expressions $s(\boldsymbol{m})$ and $s(\boldsymbol{y})$ are defined in the same way. By $s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\}$, $\boldsymbol{c} \in D_I$ we denote the state s' which coincides with s on all variables from $M \cup O$, but maps the input vector to valuation $s'(\boldsymbol{x}) = \boldsymbol{c}$. For $(s_1, s_2) \in R$ we also use the shorter expression $R(s_1, s_2)$. Restricting a state s to variable symbols from a set $U \subseteq V$ is denoted by $s|_U$. This function has domain U and coincides with s on this domain.

IOSTSs have the most general semantic model for SUTs where variables with non-trivial data types have to be considered. A "trivial data type" would be one whose elements correspond to a finite enumeration.[6] IOSTSs can be extended to Kripke Structures in a natural way: define a set of atomic propositions AP as a subset of

$$\mathcal{A}(V) = \{p \mid p \text{ is an atomic proposition with free variables in } V\}$$

For example, $v = d$ with $v \in V$ and $d \in D_v$ is an element of $\mathcal{A}(V)$; for variables $v, v_1, v_2 \in V$ with numerical domains also $v < d, v \leq d, v > d, v \geq d, v_1 < v_2$, and atomic propositions involving arithmetic expressions, such as $v_1^2 + 3 \cdot v_2 \leq v$ are elements of $\mathcal{A}(V)$. The labelling function is then specified in a natural way by

$$\forall s \in S : L(s) = \{p \in AP \mid s \models p\}$$

where $s \models p$ (*"s is a model of p"*) means that proposition p, after replacing every free variable occurrence v by its valuation $s(v)$, evaluates to `true`.

Since we are dealing with safety-critical interlocking system controllers, we only consider *deterministic* IOSTSs. This means that for every pair of traces $s_0.s_1 \ldots s_n$, $s_0.s'_1 \ldots s'_n$ with equal length,

$$(s_0|_I).(s_1|_I) \ldots (s_n|_I) = (s_0|_I).(s'_1|_I) \ldots (s'_n|_I) \Rightarrow$$
$$(s_0|_O).(s_1|_O) \ldots (s_n|_O) = (s_0|_O).(s'_1|_O) \ldots (s'_n|_O)$$

This formula expresses the fact that identical input sequences lead to identical output sequences. Two states of $\mathcal{S}$ are called *I/O-equivalent* if applying the same sequence of inputs to both states results in the same sequence of outputs from both states. Two IOSTSs are I/O-equivalent, if their initial states are.

A test case for an IOSTS can be expressed by means of an input sequence $\iota \in D_I^*$ and an output sequence $\rho \in D_O^*$ of the same length as ι specifying the outputs expected when applying ι to the initial state. If the resulting observable state sequence $(s_0|_{I \cup O}).(s_1|_{I \cup O}) \ldots (s_n|_{I \cup O})$ fulfils some LTL formula ψ with free variables in $I \cup O$, then this test case traces back to every requirement specified by some formula ϕ over free variables in $I \cup O$, such that $\psi \Rightarrow \phi$.

Observe that the Kripke structure M modelling the interlocking system in Section 3 provided a nondeterministic safe over-approximation of the interlocking system to be realised: for example, route dispatching was interleaved with interlocking control activities and network transitions in a nondeterministic way. The objective of the IOSTS-based test suite is to show that the SUT behaviour conforms to a deterministic refinement of M.

4.4 Complete Testing Strategies

Dijkstra's famous statement that tests can only prove the presence of bugs but not their absence has to be clarified. It is indeed possible to *prove* the absence of

[6] If all SUT data types were trivial, the SUT semantics could be represented even simpler by means of Mealy or Moore automata: these operate on finite input and output alphabets, and possess finite internal state [2].

bugs in the SUT by testing, as long as certain hypotheses of the true behaviour of the SUT can be assumed to be valid. In the field of model-based testing, a well-adopted approach to capture such hypotheses is to introduce *fault models* $\mathcal{F} = (\mathcal{S}, \sqsubseteq, \mathcal{D})$. Each $\mathcal{F}$ consists of a *reference model* $\mathcal{S}$, a *conformance relation* $\sqsubseteq$, and a *fault domain* $\mathcal{D}$ [29]. $\mathcal{S}$ specifies the expected behaviour of the SUT, $\sqsubseteq$ specifies a relation between models of the same semantic domain as $\mathcal{S}$, and $\mathcal{D}$ consists of a (usually infinite) set of models $\mathcal{S}'$ from this domain, that may conform to the reference model ($\mathcal{S}' \sqsubseteq \mathcal{S}$) or not. A test suite is called *complete with respect to* $\mathcal{F}$, if and only if all tests of the suite will pass for every $\mathcal{S}' \in \mathcal{D}$ conforming to $\mathcal{S}$, and at least one test will fail when executed against a non-conforming member of $\mathcal{D}$.

Dijkstra's statement is still valid in the sense that for black box testing, no test suite can be universally complete for every possible implementation of a reference model, because it cannot be determined whether the SUT has additional internal states that were not covered by the test suite under consideration. An implementation created by a malicious agent could contain a counter variable c, so that the implementation behaviour conforms to the reference model, as long as the number of processing cycles recorded in c is less then some maximum which is unknown to the tester. After c has reached this maximum, erroneous behaviour is shown. Such a malicious error seed inside an application is called a *time bomb*.

We will now consider failure models $\mathcal{F} = (\mathcal{S}, \sqsubseteq, \mathcal{D})$ over IOSTSs. The reference model $\mathcal{S}$ is a deterministic IOSTS with finite state space ($|S| \leq n \in \mathbb{N}$), variable space $V = I \cup M \cup O$ as described above, and as conformance relation $\sqsubseteq$ we choose I/O-equivalence. Since the latter is an equivalence relation and therefore symmetric, we will use the symbol $\sim$ for specifying equivalent states or IOSTSs from now on. As fault domains we use

$$\mathcal{D} = \mathcal{D}(m) = \{\mathcal{S}' \mid \mathcal{S}' \text{ is a deterministic IOSTS over variables from } V, \text{ and } |S'| \leq m\}, \quad m \geq |S|$$

The following theorem establishes a well-known fact about the existence of finite complete test suites, adapted to the context of IOSTSs. Other variants of this theorem play an important role for deriving complete finite test suites in the semantic domain of timed automata [31], and for showing that bounded model checking can *globally* prove the absence of safety violations in the model, if the BMC instance (8) is unrolled for a sufficient number k of steps [5].

Theorem 1. *Given fault model* $\mathcal{F} = (\mathcal{S}, \sqsubseteq, \mathcal{D}(m)), m \geq n = |S|$ *as specified above, define the following test suite.*

1. *If the initial outputs* $(s_0|_O), (s'_0|_O)$ *of reference model* $\mathcal{S}$ *and SUT* $\mathcal{S}'$*, respectively, differ already, set the test to FAILED.*
2. *Otherwise perform a breadth-first search over the state space of* $\mathcal{S}$ *to a depth (that is, up to a length of input sequences) of* $m \times n$*. (Observe that this leads to* $\mathcal{S}$*-states being visited more than once.)*

3. *For every state s reached during the search by means of input sequence $\iota \in D_I^*$, define test cases*

$$\iota.\boldsymbol{c}$$

for each possible input valuation $\boldsymbol{c} \in D_I$. (Observe that when doing breadth-first search to a depth of $m \times n$, the maximal length of ι is $m \times n - 1$.)
4. *For every test case $\iota.\boldsymbol{c}$, use the outputs produced by $\mathcal{S}$ when applying this input sequence to the initial state s_0 as test oracle.*

Then the resulting test suite is complete for $\mathcal{F}$.

Proof. Consider the product IOSTS constructed from $\mathcal{S}$ and an arbitrary member $\mathcal{S}' \in \mathcal{D}(m)$. Apply an input sequence $\tau.\boldsymbol{c} \in D_I^*$ with length $\#\tau \geq m \times n$ to the initial state (s_0, s'_0). Since the product IOSTS has at most $m \times n$ reachable states, applying τ to (s_0, s'_0) will lead to re-visiting a state pair $(s_1, s'_1) \in S \times S'$ that was already reached by a test case with a shorter input sequence $\iota, \#\iota < \#\tau$, and from there all inputs $\boldsymbol{c}$ had already been exercised and compared to the expected behaviour modelled by $\mathcal{S}$. Therefore it is unnecessary to exercise test case $\tau.\boldsymbol{c}$. □

This result is quite interesting from the theoretical point of view. In practise, however, its application is often infeasible, due to the large size of the state spaces involved, or due to the impossibility to identify a reliable estimate for $m \geq n$: given m and n, the size of the test suite is $|D_I|^{n \times m}$. Let us apply this size formula to the route controller for illustration purposes. For each route, the controller executes a life cycle state machine as shown in Fig. 5. Ignoring the additional states caused by sequential release, such a state machine has 5 states. With r routes to manage, the internal controller state space has therefore size 5^r. Let us focus on the normal behaviour case, so that feed backs from track elements are always well-determined by the state of the routes. Then the input vectors can be restricted to each possible combination of route allocation requests (1 = route is requested, 0 = route is not requested), so $|D_I| = 2^r$. This results in a total controller state space size of $n = 2^r \times 5^r > 2^{3r}$. Assuming that $m = n$, the product IOSTS built from reference specification and implementation has state space size $m \times n > 2^{6r}$. Let us now assume that $r = 10$, then this results in more than $(2^{10})^{(2^{60})} = 2^{(10 \times 2^{60})}$ test cases. This number of test cases is far beyond any practical feasibility. As a consequence, it is necessary to elaborate a plausible heuristics for selecting a very small subset of these test cases which is still trustworthy as a HW/SW integration test.

4.5 Test Requirements Enforced by Standards

When complete testing strategies cannot be applied to an SUT, the standards for safety-critical systems – in our case the EN50128:2011 [13] – set up rules how to decide whether a test strategy is acceptable for the SUT. The infeasibility to perform a complete strategy on system level is redeemed by performing tests on different levels of abstraction (see [13, Figure 4]).

- All software requirements have to be tested during the *software validation phase*.
- All aspects of software architecture, design, and interfaces have to be tested during the *software integration phase*.
- All components (functions, methods, ...) have to be tested during the *component testing phase*.

All test cases have to be traced back to the functional, structural, or non-functional requirements they (partially) verify [13, Table A.5, 5.]. The software code covered by these tests has to be identified. Different testing techniques have to be applied:

- *Functional black-box tests* are used to test against functional requirements on all levels identified above.
- *Structural tests* have to be performed in order to cover both software architecture and local control structures in software units.
- *Performance tests* are performed to demonstrate that the available resources (CPU power, memory, etc.) are adequate to realise the functional requirements.
- *Avalanche tests* are performed to investigate the stability of the SUT in overload situations.
- All interfaces have to be exercised.
- *Boundary value tests* have to be performed on all interfaces.
- *Equivalence classes* have to be identified for all systems whose state domains are too large to be enumerated. It has to be justified that the classes are sufficiently fine-grained.
- *Input partition testing* based on these equivalence classes has to be applied for all input (sub-) vectors whose value domains cannot be enumerated.

Furthermore, it is recommended to assess the resulting *test suite strength*, that is, its capability to uncover errors in the SUT, has to be assessed. This can be achieved, for example, by *error seeding*: the SUT software is *mutated* by introducing errors into the software code. It is then checked whether the test strength suffices to uncover these mutations [13, D.21].

4.6 Generic Domain-specific Test Strategy

As we have seen in Section 2.2, interlocking control systems are *generic* in the sense that a control algorithm performing route allocation is designed once and for all. Then, following the product line paradigm, concrete interlocking system control components – in our case, the route controller – are instantiated or configured by associating the generic algorithm with concrete network descriptions N and interlocking tables T, as has been illustrated in the case study described in Section 3.

In Fig. 10 the interface and internal state variables of the route controller are shown. The controller inputs route requests from trains, the actual signal aspects, point positions, and the vacancy status of linear track sections and

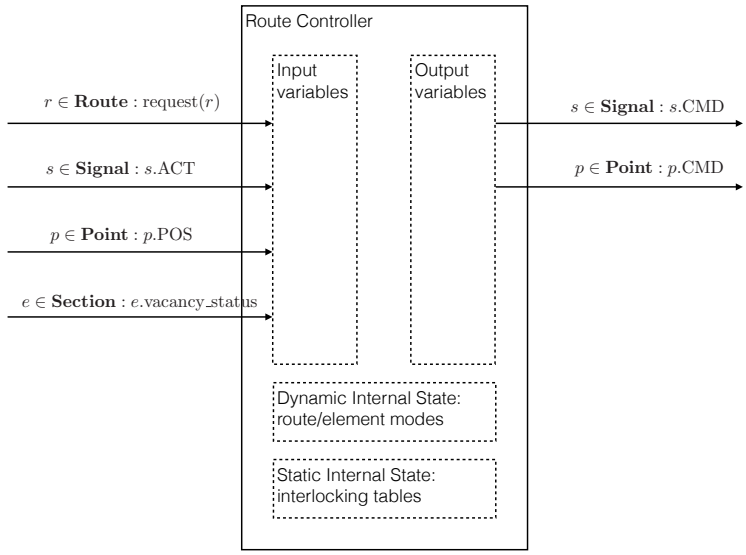


Fig. 10: Route controller interface and internal state.

points.[7] The controller writes to the signal interfaces in order to switch signal aspects, and it writes to the input interfaces of points, in order to change their position as required by the routes to be allocated.

It is typical for systems built according to the generic product line paradigm, that requirements also exist already on the generic level and are instantiated together with the concrete system configuration. A detailed analysis of test objectives on the generic level has been presented in [24, Section 5.2]. The generic user requirements can be summarised as follows.

UR-01 All routes specified in the interlocking table can be allocated.
UR-02 Every subset of non-conflicting routes can be allocated in parallel.

The generic safety requirements are

SR-01 It can never be the case that linear track sections or points that are part of conflicting routes r_1, r_2 are simultaneously allocated for both r_1 and r_2.[8]

[7] The interface e.vacancy_status is just an abstraction of the occupancy status introduced in Section 3: e.vacancy_status = VACANT if and only if the sum of all occupancy statuses associated with e is zero. Otherwise it is OCCUPIED.

[8] Observe that for the case where sequential release is disallowed, this requirement simplifies to *"It can never be the case that two conflicting routes are simultaneously allocated"*. In presence of sequential release, however, it may be the case that routes r_1, r_2 share, for example, a single point t_{ij}, and that t_{ij} has already been passed by a train riding along route r_1, so that all elements from the start of r_1 up to, and including t_{ij} have been released. In that case, r_2 can also be allocated concurrently to r_1, and t_{ij} is in state *"allocated for r_2"*.

SR-02 Whenever a route is locked, all route elements including overlaps are vacant, and all points and signals are in the states required for this route according to the interlocking table: the points are in the position specified for the route in the table, the route's entry signal is in state OPEN, and all other signals associated with the route are CLOSED.
SR-03 Whenever a track element which is part of an allocated route is sequentially released, there is no train on this route which is still to pass this element.
SR-04 Whenever a route is marked as free, all of its elements have been sequentially released.[9]

The above safety requirements are completed by specifying robust behaviour (*safety robustness requirements (SRR)*):

SRR-01 Whenever a signal fails to assume the requested aspect CLOSED, the system transits into safe state, that is, all operative signals are switched to CLOSED.
SRR-02 Whenever a point that has been locked before fails to keep up the requested position, the system transits into safe state.

Since we have already shown by model checking that the interlocking tables specify safe behaviour of trains and track elements for the network under consideration, the safety requirements SR-01,...,SR-04 and SRR-01, SRR-02 imply the validity of the safety requirements

SR-05 Whenever a route is locked or already in use, its flank protection is ensured.
SR-06 Head-on collisions can never occur.
SR-07 Rear-end collisions can never occur.
SR-08 Derailment due to erroneous point positions can never occur.

The final generic safety requirement

SR-09 Derailment due to overspeeding can never occur.

identified in [24, Section 5.2] does not have to be handled in the context analysed in this article, because we are dealing with safe control decisions of the interlocking system, while SR-09 is a safety requirement delegated to the trains' on-board controllers responsible for *automated train protection (ATP)*. Indeed, the *European Vital Computer* – this is the name of the on-board controller of ETCS systems – supervises the conformance of train velocity with the restrictions imposed in any system state along each route. For this so-called *ceiling speed supervision*, it is even possible to design and execute a complete test suite; this has been elaborated in [8, 7].

Typically, the generic requirements are formally specified as formulas $\mathbf{G}\phi$, where ϕ is a conjunction over certain configuration elements. Formalising, for

[9] In this case it follows from SR-03, SR-04 that no train resides on the route anymore.

example, requirement SR-02 introduced above, leads to

$$
\begin{aligned}
\text{SR-02} &\equiv \mathbf{G}\phi_{\text{SR-02}} \\
\phi_{\text{SR-02}} &\equiv \bigwedge_{r \in \mathbf{Route}} (r.\text{MODE} = \text{LOCKED} \Rightarrow \\
&\quad ((\bigwedge_{\substack{e \in \mathbf{path}(r) \cup \\ \mathbf{overlaps}(r)}} \text{vacant}(e)) \wedge \\
&\quad (\bigwedge_{p \in \mathbf{points}(r)} p.\text{POS} = p.\text{CMD} \wedge p.\text{CMD} = \mathbf{req}(r,p)) \wedge \\
&\quad (\bigwedge_{\substack{mb \in \\ \mathbf{signals}(r) - \mathbf{src}(r)}} s.\text{ACT} = s.\text{CMD} \wedge s.\text{CMD} = \text{CLOSED}) \wedge \\
&\quad (\mathbf{src}(r).\text{CMD} = \text{OPEN})))
\end{aligned}
$$

"Re-translating" this back to natural language, this means *"In every reachable state, whenever some route r is locked, all associated track elements (including overlaps) are vacant. All points associated with this route are in the position requested by the command issued from the interlocking system to the point, and this command is the same as the required point state* $\mathbf{req}(r,p)$ *for p if allocated in this route. Moreover, the entry signal* $\mathbf{src}(r)$ *for this route has been commanded to show aspect OPEN, while every other signal s referenced for this route has aspect CLOSED."*.

When instantiating to a concrete system, these formulas lead to concrete test cases, by specialising to the routes and track elements occurring in the system configuration. The symbolic test cases for testing requirement SR-02, for example, when instantiating a concrete system with routes $\mathbf{Route} = \{r_1, \ldots, r_n\}$ have symbolic input specifications

$$
\begin{aligned}
\text{TC-02-i} &\equiv \mathbf{F}\psi_{\text{TC-02-i}},\ i = 1, \ldots, n \\
\psi_{\text{TC-02-i}} &\equiv r_i.\text{MODE} = \text{LOCKED}
\end{aligned}
$$

which means *"Generate a sequence of inputs to the route controller, so that finally route* r_i *is put into mode* LOCKED *by the controller"*. If such a state has been reached during test execution, the expected results of test case TC-02-i are checked as specified in SR-02 by verifying the validity of proposition

$$
\begin{aligned}
\text{TC-EXP-02-i} &\equiv \psi_{\text{TC-EXP-02-i}} \\
\psi_{\text{TC-EXP-02-i}} &\equiv (\bigwedge_{\substack{e \in \mathbf{path}(r_i) \cup \\ \mathbf{overlaps}(r_i)}} \text{vacant}(e)) \wedge \\
&\quad \Big(\bigwedge_{p \in \mathbf{points}(r_i)} p.\text{POS} = p.\text{CMD} \wedge p.\text{CMD} = \mathbf{req}(r_i,p))\Big) \wedge \\
&\quad (\bigwedge_{\substack{s \in \\ \mathbf{signals}(r_i) - \mathbf{src}(r_i)}} s.\text{ACT} = s.\text{CMD} \wedge s.\text{CMD} = \text{CLOSED}) \wedge \\
&\quad (\mathbf{src}(r_i).\text{CMD} = \text{OPEN})
\end{aligned}
$$

While the n symbolic test cases specified above are obviously sufficient to show that requirement SR-02 has been implemented in the concrete SUT, there are very many computations fulfilling $\mathbf{F}\psi_{\text{TC-02-i}}$. As a consequence, it has to be decided how many different witnesses – i.e. concrete test cases – of $\mathbf{F}\psi_{\text{TC-02-i}}$ have to be generated in order to say that the requirement has been tested in sufficient depth. A well-founded answer to this questions leads to the investigation of boundary values and equivalence classes.

4.7 Functional Decomposition and Equivalence Classes

One of the most widely used heuristics to reduce test suite size is based on functional decomposition. Functions that justifiably do not affect each other can be tested separately. In the railway control application considered here this decomposition can be performed along certain route allocation states: for example, locking a completely allocated route does not affect the route's (sequential) release, because these two activities never happen at the same time for the same route. We therefore illustrate the design of equivalence class testing strategies by means of the locking function. Its basic behaviour is sketched in Fig. 11.

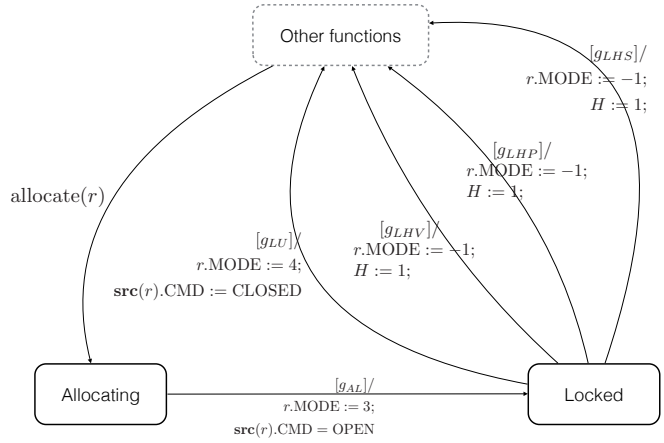


Fig. 11: State transitions related to the route locking function.

The locking functionality is activated after the allocation of a given route r has been completed. This completion condition g_{AL} ("guard condition for

transition from allocation state to locked state") is specified by proposition

$$
\begin{aligned}
g_{AL} &\equiv \text{s\_act} \wedge \text{p\_req} \wedge \text{e\_vac} \wedge \text{e\_lck} \\
\text{s\_act} &\equiv \bigwedge_{s \in \mathbf{signals}(r)} s.\text{ACT} = \text{CLOSED} \\
\text{p\_req} &\equiv \bigwedge_{p \in \mathbf{points}(r)} p.\text{POS} = \mathbf{req}(r,p) \\
\text{e\_vac} &\equiv \bigwedge_{\substack{e \in \\ \mathbf{path}(r) \cup \\ \mathbf{overlaps}(r)}} \text{vacant}(e) \\
\text{e\_lck} &\equiv \bigwedge_{e \in \mathbf{path}(r)} e.\text{MODE} = \text{EXLOCK}
\end{aligned}
$$

This means that the locked state is entered when (1) all signals associated with route r according to the interlocking table show the CLOSED aspect, (2) all points are in the position required for r according to this table, (3) all track elements along the route, including its overlap, are vacant, and (4) all elements associated with r are in locked mode. The transition into the locked state is accompanied by setting the internal route controller state to LOCKED(3) and by commanding the OPEN aspect for the route's entry signal $\mathbf{src}(r)$.

While the locking function is active, one normal behaviour and three exceptional behaviour transitions are specified: (a) If the train which allocated r enters the route, the locking function terminates successfully, and the route controller transits into the IN-USE state for this route. This is expressed formally by guard condition

$$
\begin{aligned}
g_{LU} &\equiv (\neg v_f) \wedge (\neg s_f) \wedge \text{s\_act}_1 \wedge \text{p\_req} \wedge \text{e\_vac}_1 \wedge \text{e\_lck} \\
v_f &\equiv \text{vacant}(\mathbf{first}(r)) \\
s_f &\equiv (\mathbf{src}(r).\text{ACT} = \text{CLOSED}) \\
\text{s\_act}_1 &\equiv \bigwedge_{\substack{s \in \\ \mathbf{signals}(r) - \\ \{\mathbf{src}(r)\}}} s.\text{ACT} = \text{CLOSED} \\
\text{e\_vac}_1 &\equiv \bigwedge_{\substack{e \in \\ (\mathbf{path}(r) - \{\mathbf{first}(r)\}) \cup \\ \mathbf{overlaps}(r)}} \text{vacant}(e)
\end{aligned}
$$

Exiting the LOCKED state for r is then accompanied by switching the entry signal back to CLOSED.

Each of the three exceptional behaviour transitions leaving the lock state sets the hazard indication $H := 1$ which leads to switching all signal aspects to CLOSED, while the route controller assumes the safe mode denoted here by output $H := -1$. The identified hazards are (a) detection of a non-vacant element which is part of the route (guard condition g_{LHV}), (b) detection of an illegal point position (guard g_{LHP}), and (c) detection of an illegal signal aspect

(guard g_{LHS}). These guards are defined by

$$g_{LHV} \equiv (\neg \text{e\_vac}_1) \wedge \text{s\_act}_1 \wedge \text{p\_req} \wedge \text{e\_lck}$$
$$g_{LHP} \equiv (\neg \text{p\_req}) \wedge \text{e\_vac}_1 \wedge \text{s\_act}_1 \wedge \text{e\_lck}$$
$$g_{LHS} \equiv (\neg \text{s\_act}_1) \wedge \text{e\_vac}_1 \wedge \text{p\_req} \wedge \text{e\_lck}$$

The above state machine model showing control states, transitions, guard conditions, and associated actions can be used to derive a propositional representation of the transition relation specifying the behaviour of the function under consideration. The transition relation can be used to identify *I/O-equivalence classes*: this equivalence relation partitions the state space in such a way that for every pair of states residing in the same class, the functional behaviour will be the same when applying the same sequence of inputs from each state. Having identified the I/O-equivalence classes, the associated *input equivalence classes* can be derived. Any pair of an I/O-equivalence class C and an input equivalence class X is logically connected in the following way.

> There exists an I/O-equivalence class $C' = C'(C, X)$, such that for all $s \in C, \boldsymbol{c} \in X$, the input $\boldsymbol{c}$ applied to state s results in the same output and leads to a transition into C'.

The details of this construction have been elaborated in [21]. There it has been shown that for given fault domains depending on the number of I/O-equivalence classes and on the granularity of input equivalence classes in the SUT, complete finite test suites in the sense explained above can be constructed. While these suites are significantly smaller than the ones resulting from Theorem 1, they would still be too large to be executed for the complete route controller in practise. Restricting, however, the strategy application to single functions, such as the locking functionality discussed here, meaningful and technically feasible test suites can be generated. The generation procedure according to [21] has been implemented in the model-based testing tool RT-Tester [8, 7].

Examples for I/O-equivalence classes of the locking function for route r are specified by the following propositions.

$$C_1 \equiv r.\text{MODE} = \text{ALLOCATING} \wedge s_f \wedge (\neg \text{s\_act}_1) \wedge \text{p\_req} \wedge \text{e\_vac}_1 \wedge v_f \wedge \text{e\_lck}$$
$$C_2 \equiv r.\text{MODE} = \text{LOCKED} \wedge \text{s\_act}_1 \wedge \text{p\_req} \wedge \text{e\_vac}_1 \wedge v_f \wedge \text{e\_lck}$$
$$\ldots\ldots\ldots$$

Class C_1 comprises all route controller states where route r is currently being allocated, the whole route is vacant and all points are set into the required position, but there are still some signals associated with the route that have to change their aspect to CLOSED – this is expressed by $(\neg \text{s\_act}_1)$. Class C_2 specifies all stable controller states in locked mode, where the train has not yet entered the route (v_f still evaluates to true).

Examples for input equivalence classes are characterised by the following propositions.

$$
\begin{aligned}
X_1 &\equiv s_f \wedge (\neg \text{s\_act}_1) \wedge \text{p\_req} \wedge \text{e\_vac}_1 \wedge v_f \wedge \text{e\_lck} \\
X_2 &\equiv s_f \wedge \text{s\_act}_1 \wedge \text{p\_req} \wedge \text{e\_vac}_1 \wedge v_f \wedge \text{e\_lck} \\
X_3 &\equiv (\neg s_f) \wedge \text{s\_act}_1 \wedge \text{p\_req} \wedge \text{e\_vac}_1 \wedge v_f \wedge \text{e\_lck} \\
X_4 &\equiv (\neg s_f) \wedge \text{s\_act}_1 \wedge \text{p\_req} \wedge \text{e\_vac}_1 \wedge \neg(v_f) \wedge \text{e\_lck} \\
&\ldots\ldots\ldots
\end{aligned}
$$

The effect of applying input vectors from the input equivalence classes X_i to states from the I/O-equivalence classes C_j is specified in the Mealy automaton depicted in Fig. 12.

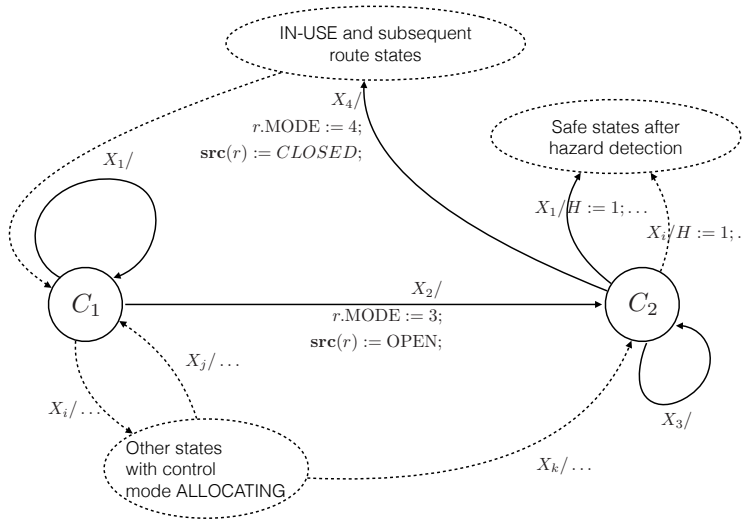


Fig. 12: Mealy machine abstraction of the route locking function.

Input equivalence class X_1 specifies all input vectors (see Fig. 10) where the entry signal for route r shows aspect CLOSED, all points are in r-position, and r is vacant, but at least one other signal associated with r is still open. Input class X_2 has fixed all inputs related to r in a way that allows to enter the LOCKED state, but note that there are numerous components of the input vector that are not determined by these requirements: all points, signals, and track sections that are unrelated to r can be set in an arbitrary manner.

When in any state of C_1, applying any input vector from X_1 leads back to a C_1 state, and no output changes occur during this transition. Applying, however, an X_2 input to a C_1 state leads to a transition to a stable C_2 state

representing the LOCKED mode. Applying X_1-inputs to any C_2-state triggers a robustness transition accompanied by setting the hazard indication: some of the signals associated with r do not remain stably in their CLOSED state. Applying an X_4-input to any C_2-state, triggers a transition into an IN-USE state, and this terminates the locking function for r.

4.8 Further Test Reduction Heuristics

Even when using equivalence class partitioning, the resulting complete test suites may be too large to be executed within acceptable time. Therefore it is interesting to investigate further heuristics explaining how to select subsets of complete test suites that have promising test strength. One of these heuristics suggests to reduce the *robustness tests*, while completely exploring the *normal behaviour tests*. A closer analysis of complete test suites shows that these contain very many robustness test steps: in each stable system state, every possible input (or a representative of its input equivalence class) should be exercised in order to show that the SUT is not affected by "unwanted" inputs in the current state. In contrast to this, normal behaviour test steps stimulate transitions leaving the current stable state or performing self-loops accompanied by new output actions. Experience shows that errors occur more frequently when transiting from a stable state than when having to ignore unwanted inputs in such a state. This suggests to reduce the number of robustness test steps in each stable state by selecting a small number of them in a random manner, but including boundary values that should still keep the system stable.

5 Conclusion

In this chapter we have described domain-specific aspects of formal modelling, bounded model checking, and model-based testing in the railway domain. It has been shown how the generic nature of interlocking systems gives rise to generic proof obligations expressing both user and safety requirements. Moreover, test strategies can be expressed already on the generic level. Instantiation of concrete systems is performed by configuring the generic software with concrete data structures specifying the railway network and the interlocking tables describing the routes through this network, to be controlled by the interlocking system.

It has been shown how the system instantiation process is accompanied by instantiation of proof obligations and test cases. The safety of control algorithms, when operating on the concrete configuration data, can be globally verified using a combination of bounded model checking and k-induction. As a consequence, the behavioural model induced by the configuration data can be regarded as being complete and correct, so that any implementation conforming to the model will also be completely safe and fulfil all user requirements. It has been motivated that finite complete system integration test strategies exist; these are capable of detecting every deviation of the integrated HW/SW system from the expected behaviour modelled by the concrete system configuration. In practice, however,

the resulting number of test cases would be too large to be executed within acceptable time. Therefore heuristics are applied to further reduce the size of the test suites.

Acknowledgments The first author has been supported by the RobustRailS project[10] funded by the Danish Council for Strategic Research. A major objective of RobustRailS is to support the ongoing development, verification, and validation of the novel Danish high-speed train network from an scientific angle. The second author has been partially supported by the openETCS project[11] funded by the European ITEA2 organisation. Both projects stimulated the work presented in this chapter. The case study described in Section 3 has been conducted in collaboration with Linh Hong Vu, who is a PhD student of the authors.

References

1. M. Aanæs and H. P. Thai. Modelling and Verification of Relay Interlocking Systems. Master's thesis, Technical University of Denmark, DTU Informatics, E-mail: reception@imm.dtu.dk, 2012.
2. István Babcsányi. Equivalence of Mealy and Moore Automata. *Acta Cybernetica*, 14:541–552, 2000.
3. Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. Météor: A successful application of b in a large project. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 369–387, Berlin Heidelberg, 1999. Springer.
4. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In Rance Cleaveland, editor, *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.
5. Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5), November 2006. arXiv: cs/0611029.
6. Dines Bjørner. New Results and Current Trends in Formal Techniques for the Development of Software for Transportation Systems. In *Proceedings of the Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'2003), Budapest/Hungary*. L'Harmattan Hongrie, May 15-16 2003.
7. Cécile Braunstein, Anne E. Haxthausen, Wen ling Huang, Felix Hübner, Jan Peleska, Uwe Schulze, and Linh Hong Vu. Complete model-based equivalence class testing for the ETCS ceiling speed monitor. In S. Merz and J. Pang, editors, *Proceedings of the ICFEM 2014*, volume 8829 of *Lecture Notes in Computer Science*, pages 380–395. Springer Berlin Heidelberg, November 2014.

[10] http://www.robustrails.man.dtu.dk
[11] http://openetcs.org

8. Cécile Braunstein, Wen-ling Huang, Jan Peleska, Uwe Schulze, Felix Hübner, Anne E. Haxthausen, and Linh Hong Vu. A SysML test model and test suite for the ETCS ceiling speed monitor. Technical report, Embedded Systems Testing Benchmarks Site, 2014-04-30. Available under `http://www.mbt-benchmarks.org`.
9. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
10. Leonardo De Moura, Harald Rueß, and Maria Sorea. Bounded Model Checking and Induction: From Refutation to Verification. In *Computer Aided Verification*, pages 14–26. Springer, 2003.
11. Ulrich W. Eisenecker and Krzysztof Czarnecki. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.
12. ERTMS. Annex A for ETCS Baseline 3 and GSM-R Baseline 0, April 2012.
13. CENELEC European Committee for Electrotechnical Standardization. *EN 50128:2011 – Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems*. 2011.
14. Alessandro Fantechi. Twenty-Five Years of Formal Methods and Railways: What Next? In Steve Counsell and Manuel Núñez, editors, *Software Engineering and Formal Methods*, volume 8368 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 2014.
15. Alessio Ferrari, Gianluca Magnani, Daniele Grasso, and Alessandro Fantechi. Model Checking Interlocking Control Tables. In Eckehard Schnieder and Géza Tarnai, editors, *FORMS/FORMAT 2010 – Formal Methods for Automation and Safety in Railway and Automotive Systems*, pages 107–115. Springer, 2010.
16. A. E. Haxthausen and J. Peleska. Formal Development and Verification of a Distributed Railway Control System. *IEEE Transaction on Software Engineering*, 26(8):687–701, 2000.
17. Anne E. Haxthausen. Automated Generation of Formal Safety Conditions from Railway Interlocking Tables. *International Journal on Software Tools for Technology Transfer (STTT), Special Issue on Formal Methods for Railway Control Systems*, 16(6):713–726, 2014.
18. Anne E. Haxthausen, Marie Le Bliguet, and Andreas A. Kjær. Modelling and Verification of Relay Interlocking Systems. In Christine Choppy and Oleg Sokolsky, editors, *15th Monterey Workshop: Foundations of Computer Software, Future Trends and Techniques for Development*, number 6028 in Lecture Notes in Computer Science, pages 141–153. Springer, 2010. Invited paper.
19. Anne E. Haxthausen and Jan Peleska. Efficient Development and Verification of Safe Railway Control Software. In *Railways: Types, Design and Safety Issues*, pages 127–148. Nova Science Publishers, Inc., 2013.
20. Anne E. Haxthausen, Jan Peleska, and Sebastian Kinder. A Formal Approach for the Construction and Verification of Railway Control Systems. In *Formal Aspects of Computing*, volume 23, pages 191–219. Springer, 2011.
21. Wen-ling Huang and Jan Peleska. Complete model-based equivalence class testing. *International Journal on Software Tools for Technology Transfer*, pages 1–19, 2014.
22. Phillip James and Markus Roggenbach. Automatically Verifying Railway Interlockings Using SAT-based Model Checking. In *Electronic Communications of the EASST*, volume 35. EASST, 2011.
23. Helge Löding and Jan Peleska. Timed moore automata: test data generation and model checking. In *Proc. 3rd International Conference on Software Testing, Verification and Validation (ICST'10)*. IEEE Computer Society, 2010.

24. Kirsten Mewes. *Domain-specific Modelling of Railway Control Systems with Integrated Verification and Validation*. PhD thesis, University of Bremen, 2010. http://www.dr.hut-verlag.de/978-3-86853-359-0.html.
25. Jan Peleska. Industrial-Strength Model-Based Testing - State of the Art and Current Challenges. In Alexander K. Petrenko and Holger Schlingloff, editors, Proceedings 8th Workshop on Model-Based Testing, Rome, Italy, volume 111 of *Electronic Proceedings in Theoretical Computer Science*, pages 3–28. Open Publishing Association, 2013.
26. Jan Peleska, Daniel Große, Anne E. Haxthausen, and Rolf Drechsler. Automated verification for train control systems. In E. Schnieder and G. Tarnai, editors, *Formal Methods for Automation and Safety in Railway and Automotive Systems, Braunschweig, Germany, December, 2004*, pages 252–265. Technical University of Braunschweig, ISBN 3-9803363-8-7, 2004.
27. Jan Peleska, Artur Honisch, Florian Lapschies, Helge Löding, Hermann Schmid, Peer Smuda, Elena Vorobev, and Cornelia Zahlten. A real-world benchmark model for testing concurrent real-time systems in the automotive domain. In Burkhart Wolff and Fatiha Zaidi, editors, *Testing Software and Systems. Proceedings of the 23rd IFIP WG 6.1 International Conference, ICTSS 2011*, volume 7019 of *LNCS*, pages 146–161, Heidelberg Dordrecht London New York, November 2011. IFIP WG 6.1, Springer.
28. Jan Peleska, Elena Vorobev, and Florian Lapschies. Automated test case generation with SMT-solving and abstract interpretation. In Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *Nasa Formal Methods, Third International Symposium, NFM 2011*, volume 6617 of *LNCS*, pages 298–312, Pasadena, CA, USA, April 2011. Springer.
29. A. Petrenko, N. Yevtushenko, and G. v. Bochmann. Fault models for testing in context. In Reinhard Gotzhein and Jan Bredereke, editors, *Formal Description Techniques IX – Theory, application and tools*, pages 163–177. Chapman&Hall, 1996.
30. Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a SAT-solver. In Jr. Hunt, Warren A. and Steven D. Johnson, editors, *Formal Methods in Computer-Aided Design*, volume 1954 of *Lecture Notes in Computer Science*, pages 127–144. Springer Berlin Heidelberg, 2000.
31. J.G. Springintveld, F.W. Vaandrager, and P.R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, March 2001.
32. Linh Hong Vu, Anne E. Haxthausen, and Jan Peleska. A Domain-Specific Language for Railway Interlocking Systems. In Eckehard Schnieder and Géza Tarnai, editors, *FORMS/FORMAT 2014 - 10th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems*, pages 200–209. Institute for Traffic Safety and Automation Engineering, Technische Universität Braunschweig, 2014.
33. Linh Hong Vu, Anne E. Haxthausen, and Jan Peleska. Formal Modeling and Verification of Interlocking Systems Featuring Sequential Release. In *Formal Techniques for Safety-Critical Systems*, volume 476 of *Communications in Computer and Information Science*. Springer International Publishing Switzerland, 2015.

Modeling Unknown Values in Test and Verification

Bernd Becker, Matthias Sauer, Christoph Scholl, and Ralf Wimmer

Albert-Ludwigs-Universität Freiburg, Germany
{becker | sauerm | scholl | wimmer}@informatik.uni-freiburg.de

Abstract. With increasing complexities and a component-based design style the handling of unknown values (e. g., at the interface of components) becomes more and more important in electronic design automation (EDA) and production processes. Tools are required that allow an accurate modeling of unknowns in combination with algorithms balancing exactness of representation and efficiency of calculation. In the following, state-of-the-art approaches are described that enable an efficient and successful handling of unknown values using formal techniques in the areas of Test and Verification.

1 Introduction

Unknown (X) values increasingly emerge in different phases of the design and production process, and have to be handled by corresponding electronic design automation (EDA) tools. Examples include unspecified inputs or black boxes in the design, uncontrolled sequential elements, clock domain crossings or A/D boundaries. In all of these cases, the logic value of a signal is not defined and hence, only partial information on the circuit is available.

In the following, we describe current state-of-the-art approaches that enable (in principle) an exact handling of such unknown values using formal techniques in two fundamental areas of the design process, i. e., Test and Verification. In addition, efficient methods to trade off quality and computation times of the analysis are reported.

1.1 Unknown Values in Circuit Test

Logic simulation, fault simulation and test pattern generation are fundamental techniques in electronic design automation with applications, e. g., in validation, test and also product quality estimation.

Automatic test pattern generation (ATPG) algorithms for stuck-at faults either compute a pattern that detects a given fault or prove its untestability. They are typically based on structural methods such as the D-algorithm [1], PODEM [2] or FAN algorithm [3], or on Boolean satisfiability (SAT) reasoning [4–7].

However, depending on the circuit and test method, a very high fraction of signals may have X-values (see e. g., [8], [9]) that have to be taken into account

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_5,

during the test pattern generation process. Such X-sources include non-finalized parts in early design steps. But also during operation and test application, X-values may be caused by uncontrolled sequential elements, at clock domain crossings, or A/D boundaries. Additional X-sources are introduced by specific test methods such as faster-than-at-speed testing [10] or the consideration of complex fault models (e. g., open fault models [11]).

Different extensions of the basic two-valued Boolean circuit logics have been proposed to model signal states in the circuit in presence of X-values (e. g., [12–14]). However, all of them lead to pessimism of forward implication in test generation as reconverging X-values that depend on each other cannot be modeled accurately (cf. Section 3.1). To improve accuracy, restricted symbolic simulation [15] extends the number of symbols to distinguish *different* X-states and their inversion. This allows to reduce the pessimism [16], unless multiple X-states from different X-sources converge at a gate.

In general, the limited number of symbols does not allow to reflect all correlations between X-valued signals and at reconvergencies, where X-canceling may occur, the accurate output value cannot be computed any more. Test generation algorithms based on n-valued logic cannot prove the untestability of faults in the support of X-valued signals and may not be able to find a detecting pattern for all testable faults.

Therefore, design techniques that remove the impact of X-values on the circuit have been proposed and are specifically employed in the context of so-called build-in-self-test (BIST) schemes. X-canceling [17] or X-masking [18] allows to increase the number of detected faults at the cost of additional hardware structures. The overestimation of X-values in classical algorithms leads to unnecessary effort invested in such X-avoidance techniques.

The accurate computation of signal states in a circuit in presence of X-values can be achieved by formal reasoning for register-transfer and gate level simulation [19–21]. The methods used rely on symbolic computation by Boolean satisfiability (SAT), quantified Boolean formula (QBF) reasoning, or binary decision diagrams (BDDs).

More accurate or even fully accurate fault simulation can be performed even for large circuits by a combination of heuristics and SAT reasoning and allows a significant increase of fault coverage [22–24]. In principle, both logic and fault simulation in presence of X-values are NP-complete problems. And also deterministic test pattern generation for stuck-at faults in presence of X-values is at least an NP-hard problem [24].

In contrast to propositional formulae used for SAT, quantified Boolean formulae [25], where variables are existentially or universally quantified, allow a succinct representation for all possible X-values. The recent advances in the performance of QBF solvers, for example conflict driven learning [26], resolution and expansion based algorithms [27], or preprocessing [28] enable exact reasoning about fault testability in presence of Xs even for larger circuits. Doing this, an efficient stuck-at fault test generation algorithm able to prove testability or untestability

of faults in presence of X-values can be realized as outlined in greater detail in Section 3.

1.2 Unknown Values in Verification

Unknown values in circuit verification can occur, for instance, when a circuit is only partially available. Partially available means that for some of the circuit's components only their interface is known, i. e., the signals entering and leaving the components, but neither their internal structure nor the computed function. These missing parts are called *black boxes*. The actual values at their outputs are unknown. Verification has to take this into account.

There are different reasons for considering such partial (or incomplete) circuits: Errors in a circuit design should be detected as early as possible; the later errors are corrected the higher are the incurred costs. Therefore it is desirable to apply verification techniques already in an early stage of the design process when not all parts of a circuit have been implemented yet.

A further reason for considering incomplete circuits is that some modules like multipliers are notoriously hard to verify: If the property to be checked is expected to be independent of such a module, the module can be removed from the circuit, and instead it is checked whether the property under consideration holds for all possible replacements of the missing part. If this is the case, then the property also holds for the complete circuit. Otherwise either the remaining circuit is faulty or the removed module and the property interact in some unexpected way.

Considering incomplete circuits can also be beneficial for error diagnosis during debugging. Assume that an error is contained in one of the circuit's modules, but it is not known in which one. If, after removing one module, verification yields that there is an implementation of the removed part such that the considered property holds, then it is likely that the error is contained in the removed module.

If error diagnosis and error rectification are performed late in the design cycle when already a lot of efforts have been made to perform logic synthesis or even place & route steps for the complete design, then the question will be whether the design can be rectified by changing *locally* confined black boxes only, without introducing new connections to global signals leading to enormous costs for re-synthesis. A similar situation occurs in case of Engineering Change Order (ECO, small changes of specification late in the design cycle) where only locally confined parts (black boxes) should be replaced in order to satisfy the changed specification without sacrificing too much of the design efforts. In this case it is particularly important to preserve the interface of the black boxes.

The synthesis of digital controllers [29, 30] that ensure certain properties of the system at hand can also be considered as a black-box verification problem: The controller to be synthesized is the black box, and one asks whether there is an implementation such that the given property holds.

Depending on the application there are two different problem classes that are of interest: On the one hand, *realizability* asks whether there is an implementation of the black boxes such that the given property holds. On the other hand,

validity asks whether the property holds for all possible implementations. Since validity and realizability are dual properties—a property φ is valid iff $\neg\varphi$ is not realizable—we concentrate in the following on realizability problems.

The problem whether an incomplete combinational circuit can be completed such that it becomes equivalent to a given specification (*partial equivalence checking*, PEC) was first considered in [31] where several *approximate* and *exact* methods to solve the PEC problem have been presented. If an approximate algorithm reports that there is no implementation for the black boxes such that the specification holds, the desired specification is indeed not realizable. However, if such an algorithm is not able to prove non-realizability, this can be due to the approximate nature of the method, and the desired functionality may nevertheless be *not* realizable. The algorithms in [31] are based on solving SAT or QBF formulations of PEC. The SAT formulations are efficient to solve, but also rather inaccurate due to a coarse approximation. Their accuracy is improved in several steps, leading to a QBF formulation that can solve PEC for a single black box exactly. In [31] additionally an exact characterization of realizability of PEC for multiple black boxes has been proposed (based on the decomposability of a certain Boolean relation). However, no feasible algorithmic method for solving the problem has been given.

Nevertheless, [31] was the first paper to consider an exact solution of the PEC problem taking into account that the *interfaces* of the black boxes in the incomplete circuit have to be preserved. Apart from the approach in [32, 33], the diagnosis and rectification problem respecting local interfaces has not been addressed in the literature so far. In [34, 35], e. g., rectifications are computed, but they are allowed to depend on arbitrary signals in the circuit. (Moreover, in contrast to [35], [34] uses a SAT formulation to compute rectifications for a given set of counterexamples only, without considering correctness for *all* possible inputs.) The approach of [32, 33] solves the PEC problem exactly, but it is restricted to problem instances of moderate sizes, since the black boxes are replaced by function tables using an exponential number of Boolean variables. A more efficient complete approach, based on solving dependency quantified Boolean formulas (DQBFs) was presented in [36].

We have extended the application of realizability checking to sequential circuits which are specified by a set of properties (safety properties or more general properties formulated in Computation Tree Logic (CTL [37]). Here the question is whether an incomplete sequential design may be extended by black box implementations such that a set of given properties is satisfied. Also the problem of deciding validity is considered. We developed various approaches for solving the realizability problem either in an approximate or an exact manner. In the following we discuss some representative approaches: In [38], we provided a series of approximate methods with different precision and costs for deciding the realizability of CTL properties using symbolic methods. The approximations are based on different methods to model the effect of the unknowns at the black box outputs to the overall circuit. Moreover, [38] presents an exact method for deciding realizability for incomplete circuits with several black boxes under

the assumption that the black boxes may contain only a bounded amount of memory. This exact method is based on introducing an exponential number of new variables and is therefore only suitable for small problem instances. In [39] similar approximation methods are applied in the context of realizability checking of safety properties based on bounded model checking techniques (BMC—here a sequential circuit is "unrolled" for a number of time frames). This approximate approach leads to SAT or QBF problems. Here, the precision of modeling is not given by the user, but it is adapted automatically based on the difficulty of the problem. The approach is guided by proofs that non-realizability can not be shown using the weaker methods, independently from the number of BMC unrollings, i. e., independently from the length of a counterexample which does not depend on the implementation of the black boxes. [39] has been enhanced later on by [40] which provides proofs based on inductive arguments that non-realizability can not be shown even by our most exact QBF based methods (also independently from the number of BMC unrollings). The approach of [40] provides an exact decision procedure for realizability in the case that the design contains exactly one black box which is allowed to read all input signals (which means that it has "complete information").

In Section 4 we sketch some of the state-of-the-art techniques to solve the realizability problem of incomplete circuits.

1.3 Minimization/Maximization in Test and Verification

We finish this introductory remarks by mentioning an interesting application of unknowns to optimize the quality of patterns. More details can be found in the papers referenced.

Modeling of unknown values can be used to generalize results by forcing a target property to hold while, at the same time, requiring a maximal number of unknown values. Such a solution is helpful as only a minimal set of information needed to guarantee the property is computed and hence the solution is generalized.

A well-known instance of such an optimization problem in the test domain is the problem of finding a test pattern for a given fault requiring only a minimal set of inputs to be defined. In the verification domain, a likewise problem is finding a generalized trace that leads to an (unwanted) error state.

Both problems can be solved (optimally) using maximization techniques such as [41, 42]. They work on top of the encoding techniques presented in this book chapter by requiring a certain primary property (e. g., the detection of a fault) to hold, while at the same time maximizing secondary objectives such as the number of inputs set to X.

2 Basics

In this section, we provide an overview on the underlying formal methods considered in the chapter as well as the handling and encoding of Boolean circuits.

2.1 Boolean Satisfiability and Extensions

The following two subsections provide a brief overview on the satisfiability problem (SAT) and on quantified Boolean formula (QBF). The interested reader is referred to [25] for more details.

Deciding the satisfiability of a propositional Boolean formula (SAT) is an NP-complete problem [43]. The formula is typically provided in conjunctive normal form (CNF). A CNF is a conjunction of clauses, and a clause is a disjunction of literals, e. g., $(a \vee \neg b)$ with the Boolean variables a and b.

Many SAT-related formalisms have been introduced in recent decades. A prominent extension to the Boolean satisfiability problem is the *Maximum Satisfiability* problem (MaxSAT), an optimization problem, which is used e.g. for the applications referenced in Subsection 1.3. Intuitively, in a MaxSAT problem we try to satisfy *as many clauses as possible* in φ. In this context the clauses are also called *soft clauses*. There are several natural extensions of MaxSAT like *Weighted MaxSAT* and *Partial MaxSAT*. In the former extension the clauses are labeled with non-negative weights and the goal is to maximize the sum of the weights of the satisfied clauses. In the latter extension there are additional so-called *hard clauses*, which *must* be satisfied, whereas the soft clauses are treated as in MaxSAT. Likewise SAT, one obtains a model which indicates the MaxSAT objective: the number of soft clauses (or the sum of the clause weights) which are satisfied simultaneously.

2.2 Quantified Boolean Formulas

A quantified Boolean formula (QBF) is a propositional formula in which the variables are quantified or bounded by existential ($\exists$) or universal ($\forall$) quantifiers. A QBF can be transformed into the prenex normal form (PCNF) $\psi = \mathcal{Q}_1 X_1 \mathcal{Q}_2 X_2 \ldots \mathcal{Q}_n X_n \varphi$, with $\mathcal{Q}_i \in \{\exists, \forall\}$ and X_i disjoint sets of Boolean variables. In a PCNF all quantifiers are grouped together in a so-called prefix and precede a quantifier-free propositional formula in CNF, called the matrix φ. We define the quantifier level by the number of quantifier alternations (i. e., from $\exists$ to $\forall$ or vise versa), reading the prefix from left to right. Without loss of generality, we assume that level 0 is always existential.

As an example, a QBF in PCNF ψ with three quantifier levels is satisfied if and only if: there *exists* an assignment for all variables on quantifier level 0 such that for *every* assignment for all variables on quantifier level 1, an assignment for all variables on quantifier level 2 *exists*, such that the matrix is satisfied.

Modern QBF solvers are also able to provide a model for free (unbounded) variables of the QBF. Semantically these free variables are similar to variables quantified at level 0. To increase readability, we write in the following that we extract the model for the variables on level 0 instead of using the terminology of free variables.

The complexity of QBF satisfiability is determined by the number of quantifier alternations between existential and universal quantifiers and vice versa in the prenex form. The general problem of QBF satisfiability is a PSPACE-complete problem [44].

2.3 Dependency Quantified Boolean Formulas

Dependency quantified Boolean formulas (DQBF) are a generalization of QBF. In QBF, each existential variable depends on all universal variables on lower quantification levels. DQBF relaxes this restriction and allows existential variables to depend on arbitrary sets of universal variables.

This section mainly follows the descriptions in [36, 45].

Let φ be a Boolean formula over the Boolean variables $x_1, \ldots, x_n, y_1, \ldots, y_m$, and $D_1, \ldots, D_m \subseteq \{x_1, \ldots, x_n\}$ sets of Boolean variables. A *dependency-quantified Boolean formula (DQBF)* ψ has the form:

$$\psi := \forall x_1 \forall x_2 \ldots \forall x_n \exists y_1(D_1) \exists y_2(D_2) \ldots \exists y_m(D_m) : \varphi.$$

The sets D_i are called dependency sets of y_i and the formula φ is ψ's matrix.

We denote $V^{\exists} = \{y_1, \ldots, y_m\}$ as the set of existential variables and $V^{\forall} = \{x_1, \ldots, x_n\}$ the set of universal variables. If $y_i \in V_{\psi}^{\exists}$ is an existential variable with dependency set D_i, a *Skolem function* for y_i is a function $s_{y_i,D_i} : \mathcal{A}_{D_i} \to \{0,1\}$. In this case, $\varphi[s_{y_i,D_i}/y_i]$ denotes the expression resulting from φ by replacing each occurrence of y_i by a Boolean expression for the Skolem function s_{y_i,D_i}.

For a variable $x \in D_i$ we denote by $s_{y_i,D_i|x=0}$ the Skolem function $s_{y_i,D_i\setminus\{x\}} : \mathcal{A}_{D_i\setminus\{x\}} \to \{0,1\}$ which results from s_{y_i,D_i} by setting the variable x constantly to 0. Accordingly for $s_{y_i,D_i|x=1}$.

Let $\psi := \forall x_1 \forall x_2 \ldots \forall x_n \exists y_1(D_1) \exists y_2(D_2) \ \ldots \exists y_m(D_m) : \varphi$ be a DQBF. ψ is *satisfied* (written $\vDash \psi$) if and only if there are Skolem functions s_{y_i,D_i} for $i = 1, \ldots, m$ such that $\varphi[s_{y_i,D_i}/y_i \ \forall y_i \in V^{\exists}]$ is a tautology.

First solver implementations for DQBF are already available. We refer the reader to, e. g., [45, 46] for more information on solving DQBFs.

Every QBF can be understood as a DQBF: the QBF $\Psi := \forall X_1 \exists Y_1 \ldots \forall X_n \exists Y_n : \varphi$, where $X_i \subseteq \{x_1, \ldots, x_n\}$ and $Y_i \subseteq \{y_1, \ldots, y_n\}$ are disjoint sets of variables, is equivalent to the DQBF

$$\psi := \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \varphi$$

where $D_{y_j} = \bigcup_{\ell=1}^{k} X_\ell$ if Y_k is the unique set with $y_j \in Y_k$.

2.4 From Circuits to Formulas

By using a *Tseitin encoding* [47], a SAT instance (as CNF representation) Φ_C of a circuit C can be generated, whose size is linear in the circuit size. A Tseitin encoding of a circuit defines a Boolean variable for each line. These variables are used to represent the function of each gate based on its inputs using a two-valued logic (01-logic).

For instance, an AND gate with the inputs a and b and the output g is characterized by $g \leftrightarrow (a \wedge b)$. The corresponding encoding Φ_g for this gate g would be

$$\Phi_g := \big\{\{a, \neg g\}, \{b, \neg g\}, \{\neg a, \neg b, g\}\big\}.$$

In extension to the two-valued Tseitin encoding of a circuit, the three-valued $01X$-encoding based on [14] is often used to represent unknown (X) values.

The $01X$-logic consists of three values $\{0, 1, X\}$, which are encoded using two Boolean variables as follows: $0 = (1, 0)$, $1 = (0, 1)$, $X = (0, 0)$. The combination $(1, 1)$ is not allowed.

The $01X$-encoding for the same AND gate g, Φ_g, would be

$$\Phi_g := \big\{\{\neg a_1, g_1\}, \{\neg b_1, g_1\}, \{a_1, b_1, \neg g_1\}, \{a_2, \neg g_2\}, \{b_2, \neg g_2\}, \{\neg a_2, \neg b_2, g_2\}\big\}.$$

In comparison to a standard Tseitin encoding, the support for the X-symbol leads to larger SAT instances and hence usually harder instances but at the same time allows reasoning about unknown values. A drawback of this formulation is its pessimism that may incorrectly predict unspecified values on path reconvergencies.

3 Unknown Values in Circuit Test

As already mentioned an unknown (X) value models an unknown *binary* state of a signal. This excludes *undefined* values that are not binary resulting for example from undefined voltage levels. Signals at which unknown values originate are called *X-sources*. Of course, depending on the circuit structure these unknown values may imply further unknown values within the circuit, and an efficient and effective determination of signals with unknown values turns out to be of major interest with respect to test algorithms.

Logic and fault simulation as well as solving the ATPG problem are essential techniques in electronic design automation. The accuracy and therefore effectiveness of standard algorithms is compromised by unknown or X-values. As demonstrated in the following, using standard three-valued 01X logic this results in a pessimistic overestimation of X-valued signals in the circuit and a pessimistic underestimation of fault coverage.

3.1 Standard X-Logic Simulation

Standard X-logic simulation algorithms are based on n-valued logic systems with a limited number of symbols to denote the signal states in the simulation (i. e., three-valued 01X logic). Not all X-states, and the correlations between them, are represented accurately. The result may either underestimate the number of X-values as in the case of logic simulation using Verilog models [48], or pessimistically overestimate their number.

Example 1. Fig. 1 shows a circuit with three gates and three inputs. The simulation result of pattern $(a, b, c) = (1, \mathrm{X}, 1)$ with a standard 3-valued logic simulator is annotated to the circuit lines. The signals d, e, and f are evaluated to the unknown value X by the simulator. However, exhaustive simulations assuming $b = 0$ and $b = 1$ would show that the output f has the logic value 1 in both cases as the signals d and e always have opposite logical values. Hence, three-valued simulation overestimates the number of signals with an unknown value.

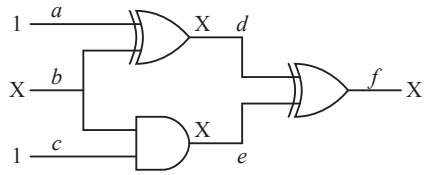


Fig. 1. Pessimistic simulation result with a 3-valued logic simulator.

For fault simulation like the parallel pattern single fault (PPSFP) or the concurrent algorithm [49], [50], [51], [52] this leads to the fact that they either pessimistically underestimate the number of detected faults or vice versa, the number of potentially detected faults is overestimated and count a fraction of potentially detected faults as detected [53]. Both inaccuracies impact product quality and may increase test overhead and cost.

3.2 Accurate Logic Simulation

Removing the pessimism of classical n-valued fault simulation requires to solve the problem of distinguishing reconverging X-values (as present in Fig. 1) that depend on each other.

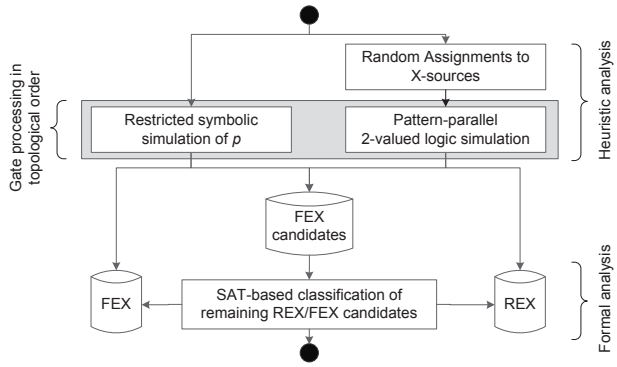


Fig. 2. Exact fault free simulation for a pattern p [24].

In [24] an accurate logic simulation algorithm based on solving a sequence of SAT-instances is proposed.

It consists of two consecutive steps as depicted in Fig. 2. In the first *heuristic analysis* step a restricted symbolic simulator and a 2-valued logic simulator are used as heuristics to classify a high number of REXs (Real X), FEXs (False X) and FEX candidates at low computational cost. In the second *formal analysis* step, the set of FEX candidates is formally analyzed. For the formal proof whether

a FEX candidate is a REX or not, the state-of-the-art incremental SAT solver antom [54] is utilized. The details of the steps are described in the folowing.

Heuristic Analysis In the heuristic analysis the pattern p is simulated using restricted symbolic simulation (RSS, [15]) and 2-valued pattern-parallel simulation of randomized assignments to the X-sources to classify as many signals as REX, FEX and FEX candidates as possible. The gates of the circuit are processed in topological order and for each gate, RSS and 2-valued simulation are performed. The identified FEX candidates are later classified using SAT reasoning.

In RSS, for each X-value at the X-sources a unique symbol X_i is introduced in addition to the two symbols for logic-0 and logic-1. Hence, X-values from different X-sources are distinguishable. Furthermore, each X-symbol can be negated. This allows the correct evaluation of simple local reconvergences of X-valued signals and increases accuracy compared to 3-valued simulators. For the example in Fig. 1, RSS correctly computes the output value at f as logic-1, since the symbol X_b introduced at X-source b is correctly tracked at d as $\neg\mathrm{X}_b$ and at signal e as X_b. Hence, the reconvergence is exactly evaluated to logic-1. Thus, RSS identifies a subset of $\mathrm{FEX}^G(p)$. In the proposed algorithm, the resulting value of RSS of signal s and pattern p is stored in $v^G(p, s)$.

A subset of $\mathrm{REX}^G(p)$ is efficiently found by a 2-valued pattern-parallel logic simulation. 64 random patterns are generated by assigning randomized values to the X-sources. The signal values are computed in one simulation run. One 64-bit integer $v = [v^0, \ldots, v^{63}]$ is used to represent the values of each signal. For input i, v_i is derived from the simulated pattern p and set to $v_i = [0, \ldots, 0]$ or $v_i = [1, \ldots, 1]$ if i is logic-0 or logic-1, respectively. At X-source q, a randomized 64-bit integer is generated and assigned to $v_q = [v_q^0, \ldots, v_q^{63}], v_q^k \in \{0, 1\}, 0 \leq k \leq 63$. v_q is used for the evaluation of the direct fanout of q.

After finishing both simulations, each signal is classified as logic-0, logic-1 or REX, FEX or FEX candidate. If RSS derived a logic value, the signal does not need to be considered in the subsequent steps. If an unknown value is calculated for s, the values of $v_s = [v_s^0, \ldots, v_s^{63}]$ of the pattern-parallel simulation are taken into account. If at least one pair of values $v_s^k, v_s^l (0 \leq k, l \leq 63)$ has complementary values, the signal s belongs to $\mathrm{REX}^G(p)$. If all v_s^k bit are equal, s is marked as FEX candidate. The classification of these signals is done with an incremental SAT-solver as explained in the next section.

Classification of Remaining FEX Candidates The FEX candidates are exactly classified by use of an incremental SAT solver. Input to the SAT solver is a Boolean formula in conjunctive normal form (CNF) which maps the classification of a signal to a Boolean satisfiability problem.

For each FEX candidate s it is already known that all 64 random assignments to the X-sources force s to value $v_s^k (0 \leq k \leq 63)$ of either logic-0 or logic-1. Signal s is a FEX, if and only if it can be proven that s cannot have the complementary value $\neg v_s^k$ for any assignment to the X-sources. Thus, the Boolean formula is constructed such that it is satisfiable, if and only if s can be driven to

$\neg v_s^k$. If the formula is satisfiable, s depends on the X-sources and is classified as REX. Otherwise s is independent of the X-sources and classified as FEX.

The FEX candidates are evaluated starting from the X-sources in topological order. To increase efficiency, the SAT instance is extended incrementally for each FEX candidate exploiting the result from the simulation step as well as learnt knowledge from analysis of previous FEX candidates.

To check whether s can be driven to $\neg v_s^k$, the characteristic equations of the gates in the adjustment cone, resp. transitive fanin, of s are translated into CNF and added to the SAT instance using the Tseitin transformation (c.f. 2.4. The size of the resulting SAT instance is reduced by only considering the gates which have been classified as REX or FEX candidate for pattern p.

This SAT instance is extended by a temporary unit clause with only one literal (called assumption) for FEX candidate s which constrains the value of s in the search process of the SAT solver. If the value of s in the pattern parallel simulation was $v_s = [0, \ldots, 0]$, the assumption $\{s\}$ is added to constrain the SAT search to assignments to the X-sources which imply s to logic-1. If the instance is satisfiable, s belongs to the set REX. Otherwise s is a FEX with value logic-0 and $v^G(p, s)$ is updated. In the latter case, the unit clause $\{\neg s\}$ is added permanently to the SAT instance to reduce runtime for subsequent calculations of the SAT solver. Correspondingly, if the value of s in the pattern parallel simulation was $v_s = [1, \ldots, 1]$, the assumption $\{\neg s\}$ is added.

For the classification of the next FEX candidate s' in topological order, the CNF instance is extended incrementally to include the adjustment cone of s', i. e.,, only the clauses for gates which are not yet Tseitin transformed are added.

During exact simulation, the algorithm maintains a lookup table derived from the result of the RSS step. The table contains the information if a symbol for an X-state assigned to signals during RSS is a logic-0, a logic-1 or a REX. Before analyzing a FEX candidate s using the SAT technique, a fast lookup is performed to check whether the corresponding symbol X_s has already been computed. If the classification for X_s is already known, s is set to the corresponding state. Otherwise, s is classified as described above. This effectively restricts the use of the SAT solver to signals at which REX values converge.

3.3 Accurate Fault Simulation

We distinguish definite detection (DD) and potential detection (PD) of a fault. A fault f is definitely detected (DD) if an observable output o exists where the fault effect is visible independent of the logic value assignment to the X-sources. Let the functions $v^G(p, s)$ and $v^f(p, s)$ return the logic value of signal s under a pattern p in the fault free and faulty case in presence of unknown values.

The definite detection of a stuck-at-ϕ fault f ($\phi \in \{0, 1\}$) at line l under a pattern p is given as

$$\mathrm{DD}^f(p) := \exists o \in O : v^G(p, o), v^f(p, o) \in \{0, 1\} \wedge v^G(p, o) \neq v^f(p, o), \qquad (1)$$

where O is the set of output signals of the circuit. If f is not definitely detected, f is potentially detected (PD) if the fault is activated and an observable output o

exists where the fault effect can be deterministically measured for at least one logic value assignment to the X-sources:

$$\mathrm{PD}^f(p) := \neg\mathrm{DD}^f(p) \wedge v^G(p,l) = \neg\phi \wedge \\ \exists o \in O : v^G(p,o) \in \{0,1\} \wedge o \in \mathrm{REX}^f(p). \quad (2)$$

Note that 3-valued fault simulation underapproximates the number of definitely detected faults since three-valued simulation overestimates the number of signals with X-values. Consequently, the number of potentially detected faults provides an overapproximation.

The exact simulation classifies a set of target faults as definitely detected (DD), potentially detected (PD) or undetected for a test set in presence of unknowns. An overview of the fault simulation of a pattern p is given in Fig. 3. 3-valued fault simulation is used to mark as many target faults as possible as DD. For the remaining faults, an exact analysis is conducted.

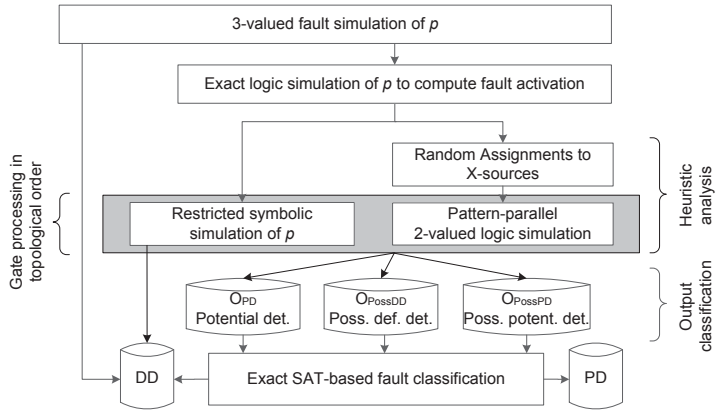


Fig. 3. Exact fault simulation for a pattern p and classification as definitely detected (DD) or potentially detected (PD) [24].

The exact analysis starts with the exact logic simulation of the fault free circuit for pattern p to compute the set of activated faults. These faults are then analyzed serially. For the fault simulation of an activated fault f, f is injected into the circuit model. The algorithm then proceeds in two phases similar to the fault free approach: A heuristic simulation and an exact calculation step. During the simulation step the behavior of the faulty circuit is simulated in event-driven manner by RSS and 2-valued pattern-parallel logic simulation which evaluates random assignments to the X-sources. If the results of the simulations allow the fault classification as DD or undetected, a further analysis is not required. Otherwise, the SAT solver is invoked for analysis of the outputs of the faulty circuit. Internal signals in the faulty circuit do not need to be considered since the values at observable outputs are sufficient to reason about fault detection.

3.4 Accurate Test Pattern Generation (X-ATPG)

The ATPG framework from [55] is able to prove the testability of stuck-at faults in presence of X-values. Fig. 4 shows the complete flow which combines accurate fault simulation (c.f. Section 3.3), incremental SAT-based test generation with a classical three-valued encoding and accurate QBF-based reasoning to efficiently analyze the faults.

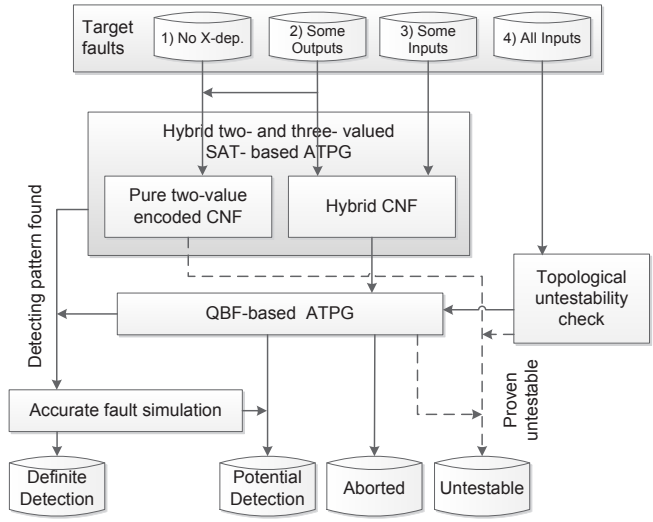


Fig. 4. Overview of the ATPG flow.

Using a topological analysis, the faults under analysis are partitioned into four groups w. r. t. their relation to the X-sources in the circuit (cf. Fig. 4):

1. No structural dependence on the X-sources: Neither the justification cone of the fault, nor its propagation cone depend on X-sources.
2. A subset of the outputs in the propagation cone depends on X-sources. The justification cone and at least one output in the propagation cone do not depend on X-sources.
3. A subset of the inputs in the justification cone of the fault depends on X-sources. At least one input in its justification cone is a controllable input.
4. The justification cone is driven exclusively by X-sources.

Afterwards the faults of each group are processed using the most suitable algorithms to keep the runtime as low as possible – while guaranteeing an accurate classification. First, all faults without X-dependency are processed by the hybrid SAT-based algorithms based on a pure two-valued signal encoding. In case a constructed formula is satisfiable, a test pattern is extracted and accurately simulated to implement fault dropping and to mark faults as potentially detected (cf. Section 3.2). Otherwise, the fault is untestable.

All faults for which some outputs or some inputs depend on X-sources are subsequently processed by the SAT-based ATPG using a hybrid two- and three-valued encoding. In case a constructed formula is satisfiable, a test pattern is extracted and simulated. Otherwise, the SAT-based approach only allows to prove the untestability, if the fault site itself does not depend on X-sources and fault activation is not possible. For all other faults which may still be detectable, a QBF is constructed and analyzed using a QBF solver for the final classification. Faults for which all inputs depend on X-sources and which have not been classified as untestable by a topological untestability check are also analyzed using the QBF-based approach.

Finally, each fault classified as untestable is analyzed again for potential detection by the QBF solver (cf. Section 3.3).

QBF-based Detection of Stuck-at Faults The construction of the QBF is split into the generation of the matrix and the quantification of the variables.

Construction of the Matrix The matrix of the QBF in CNF is constructed similar to a classical two-valued SAT-based ATPG instance. The state of each signal is modeled by a single binary variable. X-values are not explicitly specified in the matrix but modeled by universal variable quantification.

To construct the matrix for a fault f, all necessary gates for the fault-free circuit representation C^G and the propagation cone C_P^f of the fault f in the faulty circuit are modeled as formulae in CNF. Additionally, D-chains are added to encode propagation paths from the fault site to the outputs and to guide the search for a test pattern. For the D-chains, d-variables are added for each signal in the propagation cone of the fault. If the signal s has complementary values in C^G and C_P^f, d_s evaluates to 1.

Finally, a single clause $\mathcal{D} := \bigvee_{o \in \mathrm{O}} d_o$ is added to ensure that at least one d-literal of a circuit output is logically 1. This leads to the following propositional formula in CNF:

$$\mathrm{CUT} = C^G \wedge C_P^f \wedge (\text{D-chain clauses}) \wedge \mathcal{D}.$$

Variable Quantification All variables used in the matrix need to be properly quantified to guarantee a valid test in case the formula is satisfiable – or otherwise to serve as a proof that a test pattern does not exist. It is important to respect the scope of quantification, i. e., the sequence of quantifier alternations.

For fault detection, we search for *one* test pattern that satisfies the matrix for *all* possible assignments to the X-sources. Thus, the variables representing the circuit inputs are existentially quantified on level 0 and precede the universally quantified variables representing the X-sources on level 1.

The internal signals S and the d-variables used for the D-chains are subsequently existentially quantified at level 2. This results in the following QBF:

$$\underbrace{\exists I}_{\substack{\text{Controllable}\\ \text{inputs}}} \overbrace{\forall X}^{\text{X-sources}} \underbrace{\exists S \exists D}_{\substack{\text{Int. signals,}\\ \text{D-chain variables}}} \text{CUT}.$$

This QBF is satisfiable if and only if there exists an input assignment which excites an observable difference at at least one (not necessarily the same) output for each possible assignment to the X-sources.

Enforcing Definite Detection at Circuit Outputs: To establish definite detection according to Equation (1), the solution space is constrained by limiting the detecting outputs to a single fixed one. That is, for all possible assignments to the X-sources, the fault effect must be observable at one particular output.

This constraint is implemented by additional variables o_i for the outputs in the propagation cone which only evaluate to 1 if the fault effect is observable at output i for all assignments to the X-sources. The clause $(o_1 \vee o_2 \vee \ldots \vee o_n)$ enforces that at least one of the variables o_i evaluates to 1 and thus, the fault is always observable at at least one output. To guarantee that the observable output is fixed for all possible X-values, the variables in $O = \{o_i \mid 1 \leq i \leq n\}$ are existentially quantified at quantifier level 0 preceding the universal quantification of the X-sources on level 1. The relation between o_i and the D-chains are established by adding one implication per output $(o_i \rightarrow d_i)$ to the matrix:

$$\exists \boldsymbol{O} \exists I \forall X \exists S \exists D \left(\text{CUT} \wedge \bigvee_i o_i \wedge \bigwedge_i (o_i \rightarrow d_i) \right).$$

This enforces a fixed detecting output over all assignments to X-sources. However, the observable difference, i. e., the signal values in the fault-free and faulty circuit at that output is still allowed to be one of the four possibilities $(0/1), (1/0), (x_i, \neg x_i), (\neg x_i, x_i)$. The latter two cases correspond to situations where an output always shows complementary states in the fault-free and faulty circuit for all assignments to the X-sources, but the value in the fault-free and faulty circuit are not stable for all assignments to X-sources. In these cases, it is not possible to distinguish between a fault-free and a faulty circuit during testing.

Enforcing Known Binary Values at Circuit Outputs: A known binary value at the observing output in the fault-free circuit is enforced by adding two variables v_i^0, v_i^1 per output to represent its stable value in the fault-free case when it detects the fault. This automatically constrains the faulty case as well. If v_i^0 (v_i^1) is true, output i has the stable value 0 (1) in the fault-free circuit. The two implications $(v_i^0 \rightarrow \neg s_i)$ and $(v_i^1 \rightarrow s_i)$ for output i establish that relation, assuming that $s_i \in S$ is the signal variable representing the value of output i in the fault-free circuit. In the formula $\phi_{\text{Stable output}}$, the implication $(o_i \rightarrow (v_i^0 \vee v_i^1))$ ensures that output i has a stable value if o_i is asserted:

$$\phi_{\text{Stable output}} := \bigwedge_i \left((o_i \rightarrow (v_i^0 \vee v_i^1)) \wedge (v_i^0 \rightarrow \neg s_i) \wedge (v_i^1 \rightarrow s_i) \right).$$

With the existential quantification of the variables $v_i^0, v_i^1 \in V$ on level 0 we obtain the following QBF:

$$\begin{aligned}\mathrm{DD} := {} & \exists O \exists \boldsymbol{V} \exists I \forall X \exists S \exists D \\ & \Big(\mathrm{CUT} \wedge \bigvee_i o_i \wedge \bigwedge_i (o_i \rightarrow d_i) \wedge \phi_{\text{Stable output}}\Big)\end{aligned}$$

This QBF is satisfiable if and only if a fault is testable according to the definite detection condition of Section 3.3. If the formula is not satisfiable, it is proven that no test pattern exists for definite detection.

Taken together, we depicted a complete ATPG flow able to prove testability or untestability of stuck-at faults in presence of unknown values. The algorithm combines incremental 2- and 3-valued SAT-based test pattern generation, accurate fault simulation in presence of unknown values, and QBF-based test generation.

4 Unknown Values in Verification

In the following we turn to the formal verification digital circuits in the presence of unkowns. We first define partial circuits and validity and realizability of a property regarding a partial circuit. Then we present approaches how partial circuits can be analyzed.

4.1 Incomplete Circuits

An *incomplete (or partial) circuit* is a combinational or sequential circuit containing so-called *black boxes* (BBs). A black box is a module of a circuit whose interface is known but not its internal structure. A partial sequential circuit is sketched in Fig. 5. The circuit contains m black boxes $\mathrm{BB}_1, \ldots, \mathrm{BB}_m$, shown as black rectangles. Their input signals are denoted by $\boldsymbol{I}_1, \ldots, \boldsymbol{I}_m$ and their output signals by $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_m$. The primary inputs of the circuit are $\boldsymbol{x} = (x_0, \ldots, x_n)$, the current state is given by the signals $\boldsymbol{s} = (s_0, \ldots, s_r)$. The input cones of the black boxes compute the functions $\boldsymbol{I}_i = \boldsymbol{F}_i(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{i-1})$. We thereby assume that there are no cyclic dependencies between the black boxes and that they are topologically ordered, i. e., BB_i only depends on the values computed by $\mathrm{BB}_1, \ldots, \mathrm{BB}_{i-1}$. To simplify notation, we assume w. l. o. g. that no black box output is directly connected to a black box input, i. e., $\boldsymbol{Z}_i$ is disjoint from $\boldsymbol{I}_j$ for all i, j. If this is not the case, we insert a buffer between the corresponding black boxes, which does not modify the functionality of the circuit. Finally the output $\boldsymbol{y}$ and the next state $\boldsymbol{s}'$ of the circuit are given by the Boolean functions $(\boldsymbol{y}, \boldsymbol{s}') = \boldsymbol{R}(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_m)$.

We assume that the contents of the black boxes are combinational circuits. If we allow the black boxes to contain an arbitrary amount of memory, the interesting decision problems (see below) become undecidable [56]. The case of black boxes with a bounded amount of memory can be reduced to the case

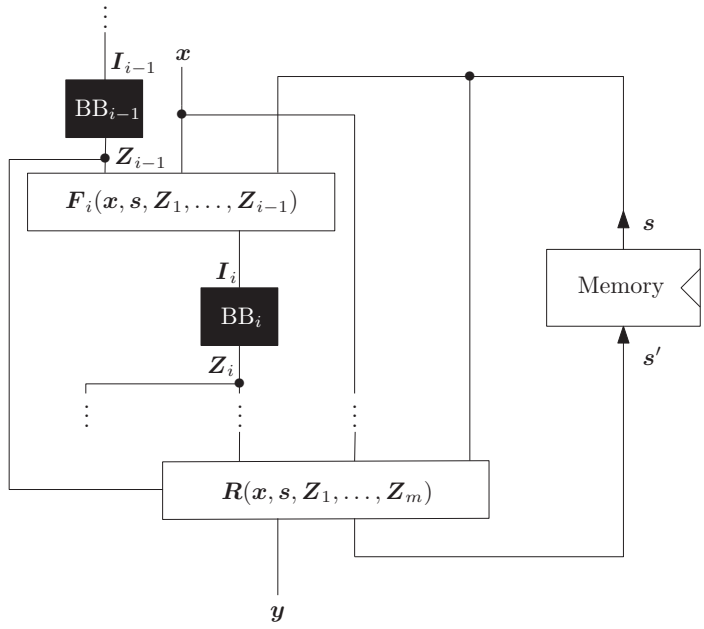


Fig. 5. Notations for an incomplete sequential circuit

of combinational black boxes by adding the memory of the black boxes to the surrounding circuit such that these memory cells are read and written only by the corresponding black box.

For a given property φ two questions regarding a partial circuit are of interest: On the one hand, *realizability* asks whether there is an implementation of the black boxes such that the complete circuit satisfies φ. On the other hand, *validity* asks whether φ is satisfied for all possible implementations. Since validity of φ is given iff $\neg\varphi$ is not realizable, we restrict ourselves in the following to realizability problems.

For partial combinational circuits we assume that the property φ is given as a circuit. The resulting realizability problem is known as the *partial equivalence checking problem* (PEC) [31]. We combine the property circuit and the partial circuit into a single miter circuit: corresponding inputs are connected, corresponding output are combined via an XOR gate; in case of several outputs, the outputs of the XOR gates are combined via an OR gate. The resulting circuit has the property that its single output is 1 for an assignment of the primary inputs and an implementation of the black boxes iff the specification φ and the implementation compute the same output values. Realizability means then: Are there implementations of the black boxes such that the output of the miter circuit is constantly 1?

For partial sequential circuits we consider invariant properties: Given a Boolean formula $\mathrm{inv}(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{y})$, which describes the states of the circuit that satisfy

the invariant, are there implementations of the black boxes such that inv($\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{y}$) is satisfied in each step of the circuit? For more general classes of properties like arbitrary CTL properties, we refer the reader to [57, 38, 58].

4.2 Incomplete Combinational Circuits

We will first show how the PEC problem can be solved for combinational circuits. We extend the methods known from the previous sections, starting with SAT-based symbolic 01X simulation, which were already used in the previous sections. Finally, we extend these methods to DQBF-based formulations which constitute a complete decision method for PEC.

SAT-based Approximations SAT-based methods use symbolic $\{0, 1, X\}$ simulation: The outputs of the black boxes carry unknown values and are therefore assigned the value X, while the primary inputs are forced to be either 0 or 1. Realizability is refuted if an input pattern can be found which leads to value 0 at the primary output of the miter circuit. Realizability is proven if all input patterns lead to output value 1.[1] However, a third case is possible, namely that the unknown value X propagates to the primary output for some input patterns. In this case, no statement can be made regarding realizability.

To decide whether there exists an input pattern that refutes realizability, a SAT-formulation can be used. To encode the three-valued logic we use the encoding introduced in Section 2.4 and force the output y of the miter circuit to be zero by adding appropriate unit clauses. The result is a Boolean formula in CNF whose satisfiability proves that the design is not realizable.

While this method is efficient in practice, it has the drawback that it constitutes a rather coarse approximation: If the unknown value X propagates to the output, no statement about the realizability can be made.

QBF-based Approximations The quality of the approximation can be improved by universal quantification over the possible input values: For all possible values at the inputs, there have to be values of the black box outputs such that the desired property is satisfied (i. e., the output of the miter circuit is 1). This yields QBF formulations for deciding PEC. We first show how to derive the matrix of the QBF formula and then define an appropriate quantifier prefix.

In contrast to circuit test applications where unknown values typically appear at the primary or secondary inputs of the circuit, we have to take into account here that black boxes are not necessarily directly connected to the primary inputs, but to internal signals. In this case not all possible combinations of values may arrive at the inputs of the black boxes. Since we use universal quantification for the black box inputs we have to ensure that the matrix of our formula is satisfied if the value of the black box inputs $\boldsymbol{I}_i$ deviates from the values obtained as a

[1] Note that in this case also validity holds.

function $\boldsymbol{F}_i(\boldsymbol{x}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{i-1})$. This leads to the following formula:

$$\varphi := \big(\boldsymbol{I}_1 \not\equiv \boldsymbol{F}_1(\boldsymbol{x})\big) \vee \cdots \vee \big(\boldsymbol{I}_m \not\equiv \boldsymbol{F}_m(\boldsymbol{x}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{m-1})\big) \vee R(\boldsymbol{x}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_m).$$

By applying Tseitin transformation [47], which introduces auxiliary variables $\boldsymbol{H} = (h_1, \ldots, h_p)$ for the internal signals of the circuit, one can obtain a CNF φ' that is satisfiability equivalent to φ and whose size is linear in the size of φ. The variables in $\boldsymbol{H}$ are existentially quantified in the quantifier prefix.

As we will see later when we consider complete decision procedures for PEC, QBF is—like the SAT-based method described above—only an approximation in case that the design contains more than one black box. However, we can give both under- and over-approximations: If an over-approximating QBF is unsatisfied, we can conclude the unrealizability of the PEC. If an under-approximating QBF is satisfied, this implies the realizability of the PEC. The other outcomes do not allow a statement regarding the realizability of the PEC. Over- and under-approximations only differ in their quantifier prefix.

For a QBF prefix $\mathcal{Q}_1 V_1\, \mathcal{Q}_2 V_2\, \ldots\, \mathcal{Q}_k V_k$ with variables $V = V_1 \cup \cdots \cup V_k$ and quantifiers $\mathcal{Q}_i \in \{\exists, \forall\}$ such that $Q_i \neq Q_{i+1}$ for all $i = 1, \ldots, k-1$, we say that a variable $u \in V$ is in the *scope* of variable $w \in V$ if V_i, V_j are the unique sets with $u \in V_i$, $w \in V_j$ and $j < i$ holds. We write $w \prec u$ if u is in the scope of w. We extend this to vectors $\boldsymbol{U}, \boldsymbol{W}$ of variables such that $\boldsymbol{U} \prec \boldsymbol{W}$ iff $u \prec w$ for all $u \in \boldsymbol{U}$ and $w \in \boldsymbol{W}$.

For an over-approximating quantifier prefix we have to take care that each black box output is (at least) in the scope of the (primary and black box) inputs which are directly or indirectly read by the black box. For an under-approximating prefix, each black box output is allowed to be at most in the scope of these variables.

Formally spoken, the requirement on the quantifier order can be translated as follows: Each over-approximating QBF prefix has to satisfy the constraint

$$\forall i = 1, \ldots, m : \boldsymbol{I}_i \subseteq \{v \in V \mid v \prec \boldsymbol{Z}_i\}, \tag{3}$$

while for each under-approximation we have

$$\forall i = 1, \ldots, m : \boldsymbol{I}_i \supseteq \{v \in V \mid v \prec \boldsymbol{Z}_i\}. \tag{4}$$

Since the Tseitin variables $\boldsymbol{H}$ are implied by the gate's inputs, they are added as the right-most existential quantifier block.

Example 2. An over-approximation is given by $\forall \boldsymbol{I}_1 \ldots \forall \boldsymbol{I}_m \exists \boldsymbol{Z}_1 \ldots \exists \boldsymbol{Z}_m \forall \boldsymbol{x} \exists \boldsymbol{H} : \varphi'$, an under-approximation by $\exists \boldsymbol{Z}_1 \ldots \exists \boldsymbol{Z}_m \forall \boldsymbol{x} \forall \boldsymbol{I}_1 \ldots \forall \boldsymbol{I}_m \exists \boldsymbol{H} : \varphi'$.

Over- and under-approximating prefixes are not unique, and the choice of the prefix can influence the truth value of the formula. It is therefore desirable to make the approximation as strong as possible, i. e., having the black box outputs in the scope of as few (many) universal variables as possible. An approximation is exact if it is both an over- and an under-approximation, or equivalently, if (3) and (4) are satisfied with equality. In general, such an exact QBF formulation does not need to exist if the circuit contains more than one black box.

Example 3. Consider an incomplete circuit with a single black box, i. e., $m = 1$. Then

$$\forall \boldsymbol{I}_1 \exists \boldsymbol{Z}_1 \forall \boldsymbol{x} \exists \boldsymbol{H} : \varphi'$$

is an exact QBF formulation, i. e., it is satisfiable if and only if the PEC is realizable.

Complete Methods The QBF method provides more accuracy than the 01X-approximation. However, in case the design contains more than one black box, QBF is still not powerful enough to express realizability exactly, because there is typically no QBF prefix that expresses the dependencies of the black box outputs from the corresponding inputs exactly.

Example 4. Consider a design with two black boxes BB_1 and BB_2. The input of BB_1 is x_1, its output y_1, the input of BB_2 is x_2 and its output y_2. There are three admissible QBF prefixes: $\forall x_1 \forall x_2 \exists y_1 \exists y_1$, $\forall x_1 \exists y_1 \forall x_2 \exists y_2$, and $\forall x_2 \exists y_2 \forall x_1 \exists y_1$. None of these is exact: in the first, both black box outputs depend on both inputs, in the second prefix, BB_2 depends on both inputs, and in the third this holds for BB_1.

In order to be able to express the dependencies correctly, one has to resort to DQBF ([36], see also Section 2.3). It is able to express arbitrary dependencies and therefore constitutes a complete decision method for arbitrary incomplete combinational circuits.

We specify the quantifier prefix of the DQBF; the matrix is the same as for the QBF approximations. The primary inputs $\boldsymbol{x}$ and the black box inputs $\boldsymbol{I}_1, \ldots, \boldsymbol{I}_m$ are again universally quantified, all other variables are existentially quantified. The dependency set of black box output $z_{i,j}$ contains exactly the inputs $\boldsymbol{I}_i$ of BB_i. Hence, the resulting DQBF is:

$$\psi := \forall \boldsymbol{x} \forall \boldsymbol{I}_1 \ldots \forall \boldsymbol{I}_m \exists \boldsymbol{Z}_1(\boldsymbol{I}_1) \ldots \exists \boldsymbol{Z}_m(\boldsymbol{I}_m) \exists \boldsymbol{H}(\boldsymbol{x}, \boldsymbol{I}_1, \ldots, \boldsymbol{I}_m) : \varphi'.$$

The formula ψ is satisfied if and only if we can replace all $\boldsymbol{Z}_i$ with Skolem functions $\boldsymbol{f}_{\boldsymbol{Z}_i, \boldsymbol{I}_i}$ such that φ' becomes a tautology. The Skolem functions $\boldsymbol{f}_{\boldsymbol{Z}_i, \boldsymbol{I}_i}$ exist if and only if there are implementations for the black boxes BB_i of the PEC, such that the specification is realized. Therefore the Skolem functions constitute implementations of the black boxes that satisfy the specification.

4.3 Incomplete Sequential Circuits

We have extended the application of realizability checking to sequential circuits which are specified by a set of properties (safety properties or more general properties formulated in Computation Tree Logic (CTL [37]). In [38], e. g., we provided a series of approximate methods with different precision and cost for deciding the realizability of CTL properties using symbolic methods. The approximations were based on different methods to model the effect of the unknowns at the black box outputs to the overall circuit and also an exact

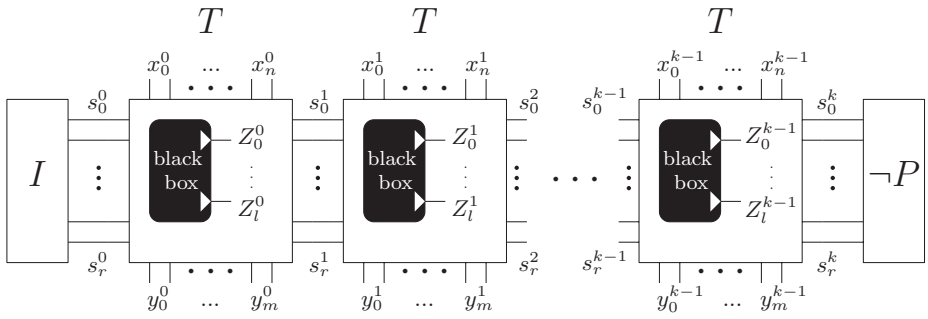


Fig. 6. Encoding of the BMC Problem for Incomplete Designs [40].

method was presented for deciding realizability for incomplete circuits with several black boxes under the assumption that the black boxes may contain only a bounded amount of memory. In this overview however, we restrict our attention to realizability checking of safety properties based on bounded model checking techniques (BMC) [39, 40].

BMC for Incomplete Designs BMC for incomplete designs aims to refute the realizability of a property, that is, it tells the designer, no matter how the unknown parts of the system will be implemented, the property will always fail. To put it in other words, the error is already in the implemented system. If this is the case, then we call the property P *unrealizable*. Here we restrict the properties to *invariants*. In a first formulation we make use of QBF modeling where the variables representing the black box outputs are universally quantified. We allow black box replacements to have arbitrary sequential behavior, that is, the black box can produce different output values for the same input values at different time steps. To encode the BMC problem of incomplete designs we are naming the variables as shown in Fig. 6. We use an upper index to specify the time instance of a variable. s_i^j denotes the i-th state bit in the j-th unfolding (let $\boldsymbol{s}^j = s_0^j, \ldots, s_r^j$). The same holds for the primary inputs $\boldsymbol{x}^j = x_0^j, \ldots, x_n^j$, the primary outputs $\boldsymbol{y}^j = y_0^j, \ldots, y_m^j$, and the black box outputs $\boldsymbol{Z}^j = Z_0^j, \ldots, Z_l^j$. The next state variables $\boldsymbol{s}^{j+1}$ depend on the current state, the primary inputs and the black box outputs. The whole circuit is transformed according to [47] using additional auxiliary variables $\boldsymbol{H}^j$ for each unfolding depth j. The predicate describing the initial states is given by $I(\boldsymbol{s}^0)$. Since we assume a single initial state in this paper, the initial state $I(\boldsymbol{s}^0)$ is encoded by unit clauses, setting the respective state bits to their initial value. The transition relation of time frame i is given by $T(\boldsymbol{s}^{i-1}, \boldsymbol{x}^{i-1}, \boldsymbol{Z}^{i-1}, \boldsymbol{s}^i)$. The invariant $P(\boldsymbol{s}^k)$ is a Boolean expression over the state variables[2] of the k-th unfolding. Using this information, the quantifier prefix (and the matrix) for the unrealizability problem results in the

[2] In general the property can also check the primary outputs, but for sake of convenience, we omit details here.

QBF formula (5). For the sake of simplicity we include the variables representing the primary outputs of unfolding depth j into $\boldsymbol{H}^j$.

$$
\begin{array}{rcl}
BMC(k) & := & \exists\ \boldsymbol{s}^0\ \boldsymbol{x}^0 \qquad \forall\ \boldsymbol{Z}^0 \quad \exists\ \boldsymbol{H}^0 \\
& & \exists\ \boldsymbol{s}^1\ \boldsymbol{x}^1 \qquad \forall\ \boldsymbol{Z}^1 \quad \exists\ \boldsymbol{H}^1 \\
& & \qquad\qquad\ \vdots \\
& & \exists\ \boldsymbol{s}^{k-1}\ \boldsymbol{x}^{k-1} \forall\ \boldsymbol{Z}^{k-1} \exists\ \boldsymbol{H}^{k-1} \\
& & \exists\ \boldsymbol{s}^k \\
\\
& & I(\boldsymbol{s}^0) \wedge \bigwedge_{i=1}^{k} T(\boldsymbol{s}^{i-1}, \boldsymbol{x}^{i-1}, \boldsymbol{Z}^{i-1}, \boldsymbol{s}^i) \wedge \neg P(\boldsymbol{s}^k)
\end{array}
\tag{5}
$$

The semantics following from the prefix corresponds to the following question:

> Does there exist a state $\boldsymbol{s}^0 = s_0^0, \ldots, s_r^0$ and an input vector $\boldsymbol{x}^0 = x_0^0, \ldots, x_n^0$ at depth 0 such that for all possible values of the black box outputs $\boldsymbol{Z}^0 = Z_0^0, \ldots, Z_m^0$ there exists an assignment to all auxiliary variables $\boldsymbol{H}^0$ (resulting in a next state $\boldsymbol{s}^1 = s_0^1, \ldots, s_r^1$) and an input vector $\boldsymbol{x}^1 = x_0^1, \ldots, x_n^1$ at depth 1, etc. such that the property is violated?

The BMC procedure iteratively unfolds the incomplete circuit for $k = 0, \ldots, K$ until a predefined maximal unfolding depth K is reached. If a QBF solver finds $BMC(k)$ satisfiable, the unrealizability of the property P has been proven. In that case the resulting system can reach a "bad state" after k steps, no matter how the black box is implemented.

We can prove that, whenever $BMC(k)$ is *unsatisfiable*, there is an implementation of the black box which is able to avoid error paths of length k *as long as the black box is allowed to read all primary inputs*. However, if the black box in the design at hand is not directly connected to all primary inputs, (i. e., if the black box does not have "complete information"), such an implementation does not need to exist. Thus, for black boxes having "incomplete information" the property may be unrealizable although BMC with QBF modeling is not able to prove this. In this case, DQBF is necessary to express the actual dependencies of the black boxes on their inputs.

In the following we give an example illustrating the approach:

Example 5. Consider the incomplete circuit shown in Fig. 7. The state bits s_0 and s_1 depend on the current state, the primary input x, and the black box outputs Z_0 and Z_1, respectively, and are computed by the transition functions $s_0' = x + Z_0$ and $s_1' = x \cdot Z_1 + s_0 \cdot \neg Z_1$. Let the invariant property $P = \neg(s_0 \wedge s_1)$ state that s_0 and s_1 must never be 1 at the same time. Let the initial state of the system be defined as $s_0^0 = s_1^0 = 0$. After checking for an initial violation of the property, the BMC procedure unfolds the system once, and tries to find an assignment to x^0 such that for all possible assignments to Z_0^0 and Z_1^0 the state $(1, 1)$ can be reached. Indeed, $x^0 = 1$ implies $s_0 = 1$ for all assignments to the black box outputs, however, for $Z_1^0 = 0$ $s_1 = 1$ can not be obtained (neither by

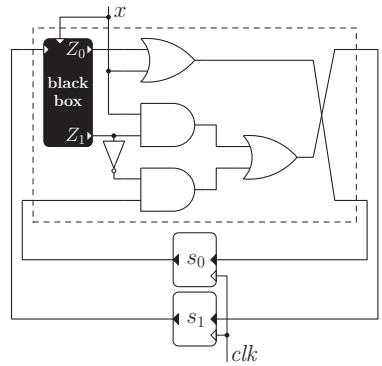


Fig. 7. Example Incomplete Design [40].

setting $x^0 = 0$ nor $x^0 = 1$). Thus, $BMC(1)$ is unsatisfiable and BMC continues by adding a second copy of the transition relation to the problem. If $x^0 = 1$, the current state bit s_0^1 at the second unfolding evaluates to 1 as well. Furthermore, if x^1 is set to 1, the next state bits s_0^2 and s_1^2 evaluate to 1 for all values of Z_0^1 and Z_1^1. Hence, when applying the input pattern $x^0 = x^1 = 1$, a state violating P can be reached after two steps for all actions of the black box and thus, $BMC(2)$ is satisfiable and P is unrealizable.

SAT-based Approximations and 01X-Hardness As already shown in previous sections, QBF can be approximated using 01X-logic. In the previous example, e. g., it is not necessary to use QBF encoding for Z_0. When applying $Z_0 = X$, unrealizability still can be proven by applying $x^0 = x^1 = 1$.

Since the problem instances using 01X-modeling are typically easier to solve, we are using the following verification flow:

Given an incomplete design and an invariant, we start the BMC process with a pure 01X-modeling, that is we extend Boolean logic by a third value 'X' which then is applied to all black box outputs. Using the two-valued encoding proposed by Jain (cf. Section 2.4), the BMC unfoldings still yield SAT problems which can be solved by a state-of-the-art SAT solver. However, 01X-modeling may be too coarse to prove unrealizability leading to unsatisfiable BMC instances for every unfolding depth (we call such verification problems *01X-hard*). In [39] we presented a method based on Craig interpolation to classify 01X-hard problems on-the-fly along the BMC process, thus preventing the solver running into unsatisfiable instances forever. Additionally, the computed Craig interpolants provide information about the origin of the 01X-hardness and a subset of the black box outputs which have to be modeled more precisely using QBF is heuristically determined. Now a QBF-based BMC tool processes the information gathered from the Craig interpolants and uses one universally quantified variable for each black box output which needs a more precise modeling. Using a combined 01X/QBF-modeling (or a pure QBF-modeling) the BMC unfoldings yield QBF

formulas. In that way, the precision of modeling is not given by the user, but it is adapted automatically based on the difficulty of the problem.

QBF-based Approximations and QBF-hardness However, even when using the more precise QBF modeling technique to model the unknown behavior of the black box, no result is guaranteed. At this point an extension given in [40] is introduced into the workflow. Similar to 01X-hardness for 01X-modeled incomplete designs, a QBF modeled BMC problem can now be classified as *QBF-hard*, if BMC would continuously run into unsatisfiable unfoldings.

As already discussed above, under certain conditions (black boxes having "incomplete information") the BMC procedure using a QBF formulation is not able to prove unrealizability even if the property is indeed unrealizable. In this sense the QBF formulation is a sound but incomplete approximation (just as 01X-modeling which is also an approximation, but is strictly coarser). If unrealizability can not be proven due to the approximative nature of the method or if the property is really realizable, then the BMC procedure described above would produce unsatisfiable QBF formulas for all unfoldings and would never return a result.

The idea of proving QBF-hardness is as follows: The QBF-based BMC procedure classifies a property as unrealizable, iff there exist input sequences of some length k such that independently from the black box actions the property will be violated after k steps. Conversely, the QBF-based BMC procedure is not able to prove unrealizability with an unfolding of length k or smaller, if for each input valuation in each time frame there is an action of the black box such that the property is fulfilled after k steps, and additionally all states on these paths also fulfill the property. Furthermore, if we can prove for this scenario that after at most k steps every state has already been visited before, we can be sure that the QBF-based BMC procedure will *never* produce a satisfiable instance, since for every input pattern it is possible to determine at least one realization of the black box leading to a state which does not violate the property, independently from the length of the unfolding.

This concept is illustrated in Fig. 8. Let $\boldsymbol{s}^0 = \boldsymbol{\sigma}_1^0$ be the initial state which fulfills P. Next, the graph branches for all possible assignments $\boldsymbol{\xi}_1^0, \ldots, \boldsymbol{\xi}_m^0$ to the primary inputs $\boldsymbol{x}^0$. For each of these values $\boldsymbol{\xi}_i^0$ there exists an action of the black box outputs $\boldsymbol{Z}^0 = \boldsymbol{\zeta}_i^0$ leading to next states $\boldsymbol{s}^1 = \boldsymbol{\sigma}_i^1$ which all fulfill P. Once a state is equivalent to a state which was visited before (which is indicated by a dashed backward arrow in Fig. 8 stating that $\boldsymbol{\sigma}_1^1 = \boldsymbol{\sigma}_1^0$, $\boldsymbol{\sigma}_1^2 = \boldsymbol{\sigma}_2^1$, $\boldsymbol{\sigma}_m^2 = \boldsymbol{\sigma}_1^0$, respectively), this branch does not need to be further explored. If at some depth all so far explored states point back to already visited states, then the black box outputs are set in a way that the system remains in "good states" forever, i. e., we are in the situation sketched above and we can be sure that the QBF-based BMC procedure will never produce a satisfiable instance, independently from the length of the unfolding. Thus, determining whether a graph fulfilling the aforementioned properties exists answers the question of whether a design is QBF-hard.

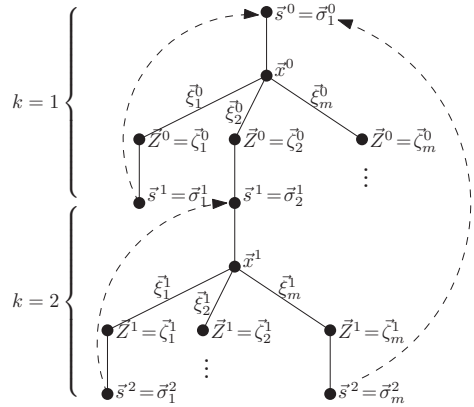


Fig. 8. QBF-Hardness Graph [40].

In [40] it has been shown that the existence of a QBF-hardness graph can be checked using a series of QBF formulas. Once the QBF-hardness of the design under verification is proven, two options are considered. In case of a combined 01X/QBF-modeling an abstraction refinement procedure will identify more black box outputs for QBF-modeling (in an extreme case yielding a pure QBF-modeled problem) and repeat the QBF-based BMC procedure. If all black box outputs are already QBF-modeled, the design is passed to a BMC tool supporting DQBF with Henkin quantifiers providing the next level of accuracy of black box modeling. Fig. 9 illustrates the complete verification flow.

5 Conclusion

In this paper, we have seen that unknown values appear at several points in the design process of a digital circuit. We have presented different methods for modeling such unknown values: (1) 01X-logic, which is efficient, but pessimistic and over-estimates the set of signals which carry an unknown value, (2) quantified Boolean formulas, which constitute an accurate formalism for modeling many problems arising in the test of digital circuits, and (3) dependency-quantified Boolean formulas, which are accurate if partial knowledge has to be taken into account, e. g., when black boxes in a circuit design have only access to a subset of the circuit's signals.

References

1. Roth, J.P.: Diagnosis of automata failures: A calculus and a method. IBM J. Res. Dev. **10**(4) (1966) 278–291
2. Goel, P.: An implicit enumeration algorithm to generate tests for combinational logic circuits. In: Proc. Fault Tolerant Computing Symposium. (1980) 145–151

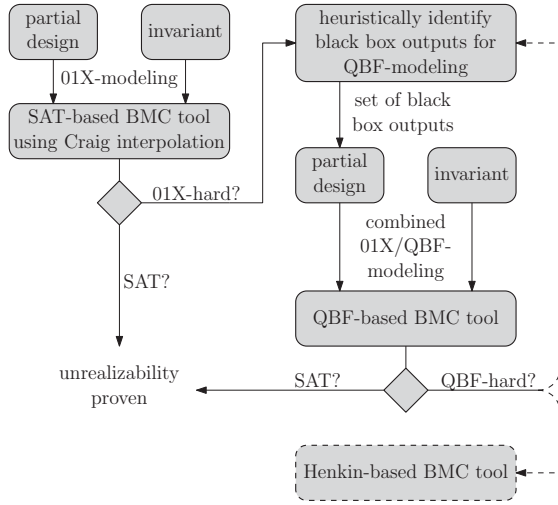


Fig. 9. Workflow.

3. Fujiwara, H., Shimono, T.: On the acceleration of test generation algorithms. IEEE Trans. on Computers **C-32**(12) (1983) 1137 –1144
4. Larrabee, T.: Test pattern generation using Boolean satisfiability. IEEE Trans. on Computer-Aided Design **11**(1) (1992) 4–15
5. Stephan, P., Brayton, R., Sangiovanni-Vincentelli, A.: Combinational test generation using satisfiability. IEEE Trans. on CAD of Integrated Circuits and Systems **15**(9) (1996) 1167–1176
6. Czutro, A., Polian, I., Lewis, M., Engelke, P., Reddy, S.M., Becker, B.: Thread-parallel integrated test pattern generator utilizing satisfiability analysis. International Journal of Parallel Programming **38**(3-4) (2010) 185–202
7. Eggersglüß, S., Drechsler, R.: Atpg based on Boolean satisfiability. In: High Quality Test Pattern Generation and Boolean Satisfiability. Springer (2012) 59–70
8. Wohl, P., Waicukauski, J., Neuveux, F.: Increasing scan compression by using X-chains. In: Int'l Test Conference (ITC). (2008) 1–10
9. Ramdas, A., Sinanoglu, O.: Toggle-masking scheme for X-filtering. In: European Test Symposium (ETS). (2012) 1–6
10. Ahmed, N., Tehranipoor, M.: A novel faster-than-at-speed transition-delay test method considering IR-drop effects. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **28**(10) (2009) 1573–1582
11. Hillebrecht, S., Polian, I., Engelke, P., Becker, B., Keim, M., Cheng, W.T.: Extraction, simulation and test generation for interconnect open defects based on enhanced aggressor-victim model. In: Int'l Test Conference (ITC). (2008) 1–10
12. Muth, P.: A nine-valued circuit model for test generation. IEEE Trans. on Computers **C-25**(6) (1976) 630–636
13. Flores, P., Neto, H., Marques Silva, J.: An exact solution to the minimum size test pattern problem. In: IEEE Int'l Conf. on Computer Design (ICCD). (1998) 510–515

14. Jain, A., Boppana, V., Mukherjee, R., Jain, J., Fujita, M., Hsiao, M.: Testing, verification, and diagnosis in the presence of unknowns. In: IEEE VLSI Test Symposium (VTS). (2000) 263–268
15. Carter, J., Rosen, B., Smith, G., Pitchumani, V.: Restricted symbolic evaluation is fast and useful. In: IEEE/ACM Int'l Conf. on Computer Aided Design (ICCAD). (1989) 38 –41
16. Kundu, S., Nair, I., Huisman, L., Iyengar, V.: Symbolic implication in test generation. In: Proc. Conference on European Design Automation. (1991) 492–496
17. Touba, N.: X-canceling MISR : An X-tolerant methodology for compacting output responses with unknowns using a MISR. In: Int'l Test Conference (ITC). (2007) 1–10
18. Tang, Y., Wunderlich, H., Engelke, P., Polian, I., Becker, B., Schloffel, J., Hapke, F., Wittke, M.: X-masking during logic BIST and its impact on defect coverage. IEEE Trans. on Very Large Scale Integration (VLSI) Systems **14**(2) (2006) 193–202
19. Elm, M., Kochte, M.A., Wunderlich, H.J.: On determining the real output Xs by SAT-based reasoning. In: IEEE Asian Test Symposium (ATS). (2010) 39–44
20. Chou, H.Z., Chang, K.H., Kuo, S.Y.: Accurately handle don't-care conditions in high-level designs and application for reducing initialized registers. IEEE Trans. on Computer-Aided Design **29**(4) (2010) 646–651
21. Wilson, C., Dill, D., Bryant, R.: Symbolic simulation with approximate values. In Hunt, W., Johnson, S., eds.: Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD). Vol. 1954 of LNCS. Springer (2000) 507–522
22. Kochte, M.A., Elm, M., Wunderlich, H.J.: Accurate X-propagation for test applications by SAT-based reasoning. IEEE Trans. on Computer-Aided Design **31**(12) (2012) 1908–1919
23. Hillebrecht, S., Kochte, M.A., Wunderlich, H.J., Becker, B.: Exact stuck-at fault classification in presence of unknowns. In: European Test Symposium (ETS). (2012) 1–6
24. Erb, D., Kochte, M.A., Sauer, M., Hillebrecht, S., Schubert, T., Wunderlich, H.J., Becker, B.: Exact logic and fault simulation in presence of unknowns. ACM Trans. on Design Automation of Electronic Systems (TODAES) **19**(3) (2014)
25. Biere, A., Heule, M., van Maaren, H., Walsh, T., eds. Frontiers in Artificial Intelligence and Applications 185. In: Handbook of Satisfiability. IOS Press (2009)
26. Zhang, L., Malik, S.: Conflict driven learning in a quantified Boolean satisfiability solver. In: IEEE/ACM Int'l Conf. on Computer Aided Design (ICCAD). (2002) 442–449
27. Biere, A.: Resolve and expand. In: Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT). Vol. 3542 of LNCS, Springer (2005) 59–70
28. Giunchiglia, E., Marin, P., Narizzano, M.: sQueezeBF: An effective preprocessor for QBFs based on equivalence reasoning. In: Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT). Vol. 6175 of LNCS. Springer (2010) 85–98
29. Bloem, R., Könighofer, R., Seidl, M.: SAT-based synthesis methods for safety specs. In: Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI). Vol. 8318 of LNCS, Springer (2014) 1–20
30. Bloem, R., Egly, U., Klampfl, P., Könighofer, R., Lonsing, F.: SAT-based methods for circuit synthesis. In: Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD), IEEE (2014) 31–34
31. Scholl, C., Becker, B.: Checking equivalence for partial implementations. In: ACM/IEEE Design Automation Conference (DAC), ACM Press (2001) 238–243

32. Jo, S., Matsumoto, T., Fujita, M.: SAT-based automatic rectification and debugging of combinational circuits with LUT insertions. In: IEEE Asian Test Symposium (ATS), Niigata, Japan, IEEE Computer Society (2012) 19–24
33. Jo, S., Gharehbaghi, A.M., Matsumoto, T., Fujita, M.: Debugging processors with advanced features by reprogramming LUTs on FPGA. In: Int'l Conf. on Field-Programmable Technology (FPT), Kyoto, Japan, IEEE (2013) 50–57
34. Smith, A., Veneris, A.G., Ali, M.F., Viglas, A.: Fault diagnosis and logic debugging using boolean satisfiability. IEEE Trans. on CAD of Integrated Circuits and Systems **24**(10) (2005) 1606–1621
35. Sülflow, A., Fey, G., Drechsler, R.: Using QBF to increase accuracy of SAT-based debugging. In: Int'l Symposium on Circuits and Systems (ISCAS), Paris, France, IEEE (2010) 641–644
36. Gitina, K., Reimer, S., Sauer, M., Wimmer, R., Scholl, C., Becker, B.: Equivalence checking of partial designs using dependency quantified Boolean formulae. In: IEEE Int'l Conf. on Computer Design (ICCD), Asheville, NC, USA, IEEE Computer Society (2013) 396–403
37. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic Verification of Finite–State Concurrent Systems Using Temporal Logic Specifications. ACM Trans. on Programming Languages and Systems **8**(2) (1986) 244–263
38. Nopper, T., Scholl, C.: Symbolic model checking for incomplete designs with flexible modeling of unknowns. IEEE Trans. on Computers **62**(6) (2013) 1234–1254
39. Miller, C., Kupferschmid, S., Lewis, M.D.T., Becker, B.: Encoding techniques, craig interpolants and bounded model checking for incomplete designs. In: Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT). Vol. 6175 of LNCS, Springer (2010) 194–208
40. Miller, C., Scholl, C., Becker, B.: Proving QBF-hardness in bounded model checking for incomplete designs. In: Int'l Workshop on Microprocessor Test and Verification (MTV), IEEE Computer Society (2013)
41. Sauer, M., Reimer, S., Polian, I., Schubert, T., Becker, B.: Provably Optimal Test Cube Generation Using Quantified Boolean Formula Solving. In: Asia and South Pacific Design Automation Conference (ASPDAC). (2013) 533–539
42. Reimer, S., Sauer, M., Schubert, T., Becker, B.: Using maxbmc for pareto-optimal circuit initialization. In: Int'l Conf. on Design, Automation & Test in Europe (DATE), IEEE (2014) 1–6
43. Cook, S.A.: The complexity of theorem-proving procedures. In: Annual ACM Symposium on Theory of Computing (STOC), ACM (1971) 151–158
44. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time (preliminary report). In: Annual ACM Symposium on Theory of Computing (STOC), New York, NY, USA, ACM (1973) 1–9
45. Gitina, K., Wimmer, R., Reimer, S., Sauer, M., Scholl, C., Becker, B.: Solving DQBF through quantifier elimination. In: Int'l Conf. on Design, Automation & Test in Europe (DATE), Grenoble, France, IEEE (2015)
46. Fröhlich, A., Kovásznai, G., Biere, A., Veith, H.: iDQ: Instantiation-based DQBF solving. In: Intl. Workshop on Pragmatics of SAT (POS), Vienna, Austria (2014)
47. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic **Part 2** (1970) 115–125
48. Turpin, M.: The dangers of living with an X (bugs hidden in your Verilog). In: Boston Synopsys Users Group Meeting. (2003) 1–34
49. Ulrich, E.G., Baker, T.: The concurrent simulation of nearly identical digital networks. In: Papers on Twenty-five years of electronic design automation. 25 years of DAC (1988) 318–323

50. Waicukauski, J., Eichelberger, E., Forlenza, D., Lindbloom, E., McCarthy, T.: Fault simulation for structured VLSI. VLSI Systems Design **6**(12) (1985) 20–32
51. Antreich, K., Schulz, M.: Accelerated fault simulation and fault grading in combinational circuits. IEEE Trans. on Computer-Aided Design **6**(5) (1987) 704 – 712
52. Lee, H., Ha, D.: An efficient, forward fault simulation algorithm based on the parallel pattern single fault propagation. In: Int'l Test Conference (ITC). (1991) 946–955
53. Rudnick, E., Patel, J., Pomeranz, I.: On potential fault detection in sequential circuits. In: Int'l Test Conference (ITC). (1996) 142–149
54. Schubert, T., Lewis, M., Becker, B.: Antom—solver description. SAT Race (2010)
55. Hillebrecht, S., Kochte, M.A., Erb, D., Wunderlich, H.J., Becker, B.: Accurate QBF-based test pattern generation in presence of unknown values. In: Int'l Conf. on Design, Automation & Test in Europe (DATE). (2013) 436–441
56. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Annual Symposium on Foundations of Computer Science, IEEE Computer Society (1990) 746–757
57. Miller, C., Nopper, T., Scholl, C.: Symbolic CTL model checking for incomplete designs by selecting property-specific subsets of local component assumptions. In: Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen" (MBMV), Universitätsbibliothek Berlin, Germany (2009) 87–96
58. Nopper, T., Scholl, C.: Approximate symbolic model checking for incomplete designs. In: Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD). Vol. 3312 of LNCS, Springer (2004) 290–305

Specification of Parametric Monitors
Quantified Event Automata versus Rule Systems

Klaus Havelund[1]* and Giles Reger[2]

[1] Jet Propulsion Laboratory, California Inst. of Technology, USA
[2] University of Manchester, UK

Abstract. Specification-based runtime verification is a technique for monitoring program executions against specifications formalized in formal logic. Such logics are usually temporal in nature, capturing the relation between events occurring at different time points. A particular challenge in runtime verification is the elegant specification and efficient monitoring of streams of events that carry data, also referred to as parametric monitoring. This paper presents two parametric runtime verification systems representing two quite different approaches to the problem. QEA (Quantified Event Automata) is a state machine approach based on trace-slicing, while LogFire is a rule-based approach based on the Rete algorithm, known from AI as being the basis for many rule systems. The presentation focuses on how easy it is to specify properties in the two approaches by specifying a collection of properties gathered during the 1st International Competition of Software for Runtime Verification (CSRV 2014), affiliated with RV 2014 in Toronto, Canada.

1 Introduction

Ensuring the correctness or security of a software system is traditionally approached in two ways, with *static analysis* and with *dynamic analysis*. By static analysis we shall broadly understand any approach that does not execute the program using a traditional execution platform, in contrast to dynamic analysis, where the program is executed. Static analysis techniques include for example abstract interpretation, theorem proving and model checking. The distinction is somewhat vague. Some techniques are difficult to classify, for example software model checkers which execute a program using a specialized virtual machine. Dynamic analysis includes testing, which is concerned with generating test inputs for the system, and applying test oracles (monitors) that can determine whether a particular run is satisfactory. However, dynamic analysis can also be applied after deployment of the software in the field, for example to profile behavior, load, etc. under realistic conditions. The concept of cyber-physical system (CPS) is receiving increased attention in the research community. A CPS is

* The research performed by this author was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_6,

a system of collaborating computational elements controlling physical entities. The correctness of such systems is extremely difficult to ensure statically. It is therefore important to ensure that at least observed executions of such systems satisfy certain properties.

Runtime verification [30, 41, 49] (RV) is a subfield of dynamic analysis focusing on *analyzing executions*, including collections thereof, either during test (test oracles), or after deployment. The field is not concerned with test case generation. Even though the field can appear rather narrow, by tradition it includes the following sub-fields. *Specification-based* monitoring, the topic of this paper, is concerned with checking a program execution against a formal specification of one or more requirements, expressed in some form of logic, for example state machines, regular expressions, temporal logic, grammars, or rule-systems. That is, given a program P, and a specification ψ, and an execution trace τ of P, an RV system will be able to determine whether τ satisfies ψ, also formalized as: $\tau \models \psi$. Logics must be expressive and monitors must be efficient. In *runtime analysis*, program executions are analyzed with specialized algorithms. Examples include algorithms for detecting concurrency problems such as deadlock potentials and data races. In *fault protection*, a monitored system will when violating a property be brought from an unsafe state to a safe state. *Specification learning* covers learning (mining) specifications from execution traces, which are then used for either system comprehension, or fed back into a verification system for further analysis, or used for monitoring future revisions of the software. *Trace visualization* of execution traces serves the purpose of comprehending what the system does. Finally, *program instrumentation* is concerned with how to instrument programs to emit events to a monitor, which then analyzes the event stream. For example aspect-oriented programming can be used for this purpose to automatically insert event emitting code at positions relevant for the properties being verified. Static analysis can be used to minimize the number of instrumentation points. RV technology can be stand-alone or incorporated into programming languages, simple assertions being a standard example.

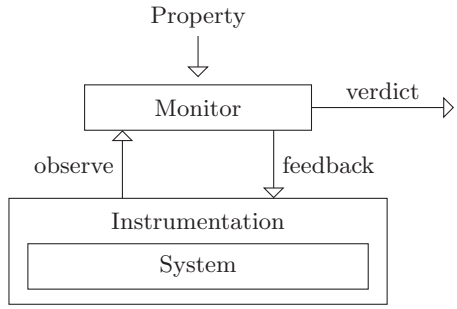


Fig. 1. A typical monitoring infrastructure.

Figure 1 shows a typical monitoring infrastructure, which supports the following activities. *Monitor creation*: a monitor is created, potentially from a formal property. *Instrumentation*: the system is instrumented to generate events for the monitor. *Execution*: The system is executed, generating events that are *observed* by the monitor. These events are either monitored online, as they are generated, or they are stored in logs, which are later analyzed by the monitor. *Responses*: the monitor produces for each consumed event a *verdict* indicating the status of the property, depending on the event sequence seen so far. In the case of online monitoring, the monitor may send *feedback* to the system, so that corrective actions can be taken by the system.

In this paper we shall focus on specification-based runtime verification, and in particular on what is referred to as *parametric monitoring*: the monitoring of event streams where events can carry data. That is, say we are monitoring a file system, then an event may have the form `read`(f) where f is a file identifier, a parameter to the `read` event. Parametric monitoring is particularly challenging as, for each incoming event, it involves the efficient lookup of information about the previous part of the execution trace that is relevant for that event's particular parameter (or set of parameters), in order to determine what the appropriate action of the monitor should be. We shall describe and apply two parametric runtime verification systems, representing two seemingly different main approaches of organizing monitors. The systems will be demonstrated on four classes of properties stemming from respectively the JAVA programming language API, banking, planetary rovers, and finally concurrency in programming languages. These properties were gathered during the 1st Intl. Competition of Software for Runtime Verification (CSRV 2014), affiliated with RV 2014 in Toronto, Canada [2], and also presented in [20]. Our focus is on specification, and we shall therefore assume that programs/systems to be monitored have been instrumented appropriately to emit the necessary events.

The first system, QEA (Quantified Event Automata) [4, 53], is a state-machine logic based on a so-called trace-slicing approach, an approach that has shown to yield extremely efficient monitors. The approach involves conceptually slicing a trace into projections, a projection for each parameter combination. Fast indexing makes this very efficient. The state of a monitor is abstractly seen as a mapping from parameter values to monitor states. Note that trace slicing is not to be confused with program slicing. The latter involves slicing a program into projections. The second system, LOGFIRE [40], is a rule-based system, inspired by rule systems as developed within the artificial intelligence community. LOGFIRE's implementation is based on the RETE algorithm, often used for implementing rule engines. A monitor consists of a set of rules, which abstractly seen operate on a (structured) set of facts, where a fact is a named record $F(v_1, \ldots, v_n)$. Here F is a name and $v_1, \ldots, v_n$ are values. A rule's left-hand side can check for the presence or absence of facts, and the right-hand side can add or delete facts. In reality, however, facts are inserted into a network, the RETE network (*rete* means "net" in Latin), making rule evaluation more efficient. QEA

is a so-called *external* DSL (Domain Specific Language), whereas LogFire is a so-called *internal* DSL, an API in the Scala programming language.

The rest of the paper is organized as follows. Section 2 gives a brief survey of specification-based runtime verification systems. Section 3 introduces the two logics qea and LogFire. Sections 4, 5, 6, and 7 contain the specification in the two logics of the Java API, banking, rover, and concurrency properties respectively. Section 8 summarizes and discusses the specification experience. Finally Section 9 concludes the paper.

2 Survey of specification-based runtime verification

Numerous runtime verification systems have been developed over the last 15 years, some of which will be mentioned here. We will only focus on specification-based systems that verify whether program executions satisfy user-provided formal specifications. As discussed in the introduction, the field is broader than this. Initial specification-based systems could only handle propositional events. These include for example Temporal Rover [26], Mac [48], and Java PathExplorer [43, 42]. Later work has studied such propositional monitoring logics from a more theoretic point of view, including notions such as 4-valued logics [15] and monitorability [29]. During the last decade there has been an increasing focus on systems for monitoring data parameterized events, so-called parametric monitoring. The first systems to handle parameterized events appeared around 2004, and include such systems as Eagle [7], Hawk [21], Jlo [58], TraceMatches [3], and Mop [18, 51]. Several systems have appeared since then. RV systems usually implement specification languages which are based on formalisms such as state machines [27, 34, 37, 18, 28, 9], regular expressions [3, 18], temporal logic [48, 26, 43, 7, 59, 21, 58, 57, 18, 9, 13, 36, 14, 22], variations over the μ-calculus [7], grammars [18], and rule-based systems [12, 40]. A few of these logics incorporate time as a built-in concept, for example the metric first-order temporal logics in the early TemporalRover [26] and in the more recent MFOTL [13]. If no special concept of time is introduced, time observations can be considered as just data, as is common in rule-based systems [12, 40].

In this paper we focus on two important approaches to parametric monitoring, namely the trace-slicing approach, represented by qea, and the rule-based approach, represented by LogFire. Two other important approaches based on trace-slicing are TraceMatches [3] and Mop [18, 51]. Slicing-based approaches are generally extremely efficient, but at the cost of lack of some expressiveness, as pointed out in [4]. qea is an attempt to augment the expressiveness of slicing-based approaches, without losing too much of the efficiency. Rule systems are expressive.

The Rete-based LogFire is inspired by the Ruler system [11, 12, 1], itself a rule-based system. Ruler, however, is not influenced by Rete. Ruler led to the study of the Rete algorithm (in LogFire), in order to determine its relevance for runtime verification. Ruler itself was inspired by MetateM [5] and Eagle [7], a linear time μ-calculus for monitoring, with past time as well

as future time operators. Although attractive, the implementation of EAGLE appeared complex, leading to the simpler implementation in RULER. Another derivative from RULER is LOGSCOPE [10, 35, 8], which is a data parameterized state machine oriented monitoring logic, implemented as a simplified rule-engine, and applied to log files at Jet Propulsion Laboratory (JPL), for testing of the Mars Curiosity rover. The QEA formalism has similarities with LOGSCOPE. Other rule-based systems include DROOLS [24], JESS [45] and CLIPS [19]. Standard rule systems usually enable processing of facts, which have a life span. In contrast, LOGFIRE additionally implements events, which are instantaneous. DROOLS supports a notion of events, which are facts with a limited life span, inspired by the concept of *Complex Event Processing* (CEP), described by David Luckham in [50]. The DROOLS project has an effort ongoing, defining functional programming extensions to DROOLS [25]. In contrast, by embedding a rule system in an object-oriented and functional language, LOGFIRE can leverage the already existing host language features.

The classification of runtime verification frameworks into slicing-based and rule-based is an idealization, and more research is needed in order to make a clean classification. Some logics, for example, handle data as constraints, including the first-order Linear Temporal Logic LTL^{FO} [14], and the first-order linear temporal logic based on SMT (Satisfiability Modulo Theories) solving described in [22]. Some systems based on Linear Temporal Logic (LTL) [52] apply rewriting of LTL formulas, inspired by [33]. These include for example [26, 43, 7, 59, 58, 9, 36].

QEA, like most of the logics mentioned above, is an *external DSL*, a standalone "small" language equipped with its own customized parser. In contrast, LOGFIRE is an *internal* SCALA *DSL*, essentially an API in SCALA. Two other rule-based internal DSLs for SCALA exist: HAMMURABI [32] and ROOSCALOO [55]. HAMMURABI, which is not RETE-based, achieves efficient evaluation of rules by evaluating these in parallel, assigning each rule to a different SCALA actor. ROOSCALOO [55] is RETE based, but is not documented in any form other than experimental code. Other internal SCALA DSLs for monitoring include TRACECONTRACT [9] and DAUT (Data automata) [38, 39], both of which are based on parameterized state machines. An embedding of LTL in HASKELL is described in [59]. MOPBOX [17] is a JAVA library for monitoring, offering a re-implementation of the efficient trace-slicing algorithms contained in MOP [18, 51], but defining the interface as an API.

3 Introduction to QEA and LogFire

This section introduces the two logics QEA (Section 3.2) and LOGFIRE (Section 3.3). The logics are illustrated using a file usage example, presented below.

3.1 The file usage example

Consider that we can monitor the following file usage events:

- `open`$(f, m, size)$: records that file f is being opened in mode m (R for read and W for write), and $size$ denotes the size of the file in bytes on opening.
- `close`(f): records closing file f.
- `read`(f): records reading a file f.
- `write`(f, b): records writing b bytes to file f.

The correct usage of the file system is captured by the following informal requirements:

- A file starts closed and cannot be opened (closed) if already open (closed).
- A file if opened must eventually be closed.
- A file can only be read or written if it is open in the corresponding mode.
- No file can exceed 16MB.

We will use this example to illustrate QEA and LogFire in the following two sections.

3.2 Introduction to QEA

Quantified event automata (QEA) [4, 53] is an automaton-based specification language for monitoring parameterized events. It is based on the idea of *parametric trace-slicing* [18], which is most prominently used by Mop. The theory behind QEA generalizes this previous notion of slicing in a number of ways that will be discussed below. The MarQ tool [54] takes QEA specifications and can monitor them either online on Java programs or offline on recorded traces.

```
qea{
  Forall(f)
  accept next(closed){
    open(f,'R',_) -> readonly
    open(f,'W',size) -> writeonly
  }
  next(readonly){
    read(f) -> readonly
    close(f) -> closed
  }
  next(writeonly){
    write(f,b) if [size+b <= 16000000 ]
               do [ size+=b ] -> writeonly
    close(f) -> closed
  }
}
```

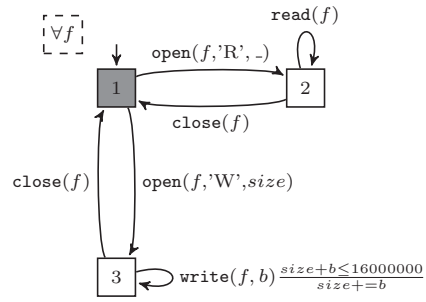


Fig. 2. A QEA model of the file usage property in both text and graphical formats.

Specification of file usage example in QEA. The QEA specification of the file usage property is given in Figure 2 in both a textual and a graphical representation. In other work we have mainly used the graphical representation as we

find it more readable. However, as it is not an input format, it is not appropriate for discussing specification approaches and will use the textual format later.

A QEA consists of some quantifications and an automaton. In this example the quantification is **Forall** (`f`) stating that f is a universally quantified variable. This quantifier list can also include **Exists** quantification and a global guard on quantified variables introduced by the **Where** keyword.

The automaton has three states relating to the three states a file can be in. In the textual format the first state defined is always taken to be the initial state. States have a kind, either **next** or **skip**, which defines what should happen when a transition cannot be taken; **next** means a next event is required, **skip** means the next event will be skipped if no transition can be taken. The states here are all **next** states i.e. from the `closed` state it is only possible to `open` a file as an event is required to make a transition.

Only the `start` is state is an **accept** state. As defined later, a trace is accepted if it has a path to an accepting state. The language also includes implicit **success** and **failure** states which are **skip** states with no outgoing transitions and where the former is an **accept** state.

In the `writeonly` state the `write` transition has a guard `size+b <= 16000000` and assignment `size+=b`, which are used to ensure that the size of the file does not exceed the limit, note how the `size` variable is given the initial size when the file is opened. The **if** keyword introduces guards and the **do** keyword introduces assignments.

Let us consider how we would monitor this property on the following trace:

$$\texttt{open}(A, \text{'R'}, 0).\texttt{open}(B, \text{'W'}, 15999900).\texttt{write}(B, 100).\texttt{read}(A).\texttt{write}(B, 100).\texttt{close}(B)$$

Here there are two files, A and B. The quantification $\forall f$ tells us that we need to *slice* the trace on these values. This gives us the following two trace slices associated with bindings of f:

$$\begin{array}{lcl} [f \mapsto A] & \mapsto & \tau_1 = \texttt{open}(A, \text{'R'}, 0).\texttt{read}(A) \\ [f \mapsto B] & \mapsto & \tau_2 = \texttt{open}(B, \text{'W'}, 15999900).\texttt{write}(B, 100).\texttt{write}(B, 100).\texttt{close}(B) \end{array}$$

The trace slice τ_1 should then be interpreted for the automaton with f replaced by A. In this case there is a path in the automaton from state `closed` to state `readonly`. But, as this state is not accepting, any trace finishing in this state is rejected. Similarly the trace slice τ_2 should be interpreted on the automaton with f replaced by B. Here we can capture the behavior through the rewriting of a configuration containing the current state and binding of free variable $size$:

$$\langle 1, [] \rangle \xrightarrow{\texttt{open}(B,\text{'W'},15999900)} \langle 3, [size \mapsto 15999900] \rangle \xrightarrow{\texttt{write}(B,100)} \langle 3, [size \mapsto 16000000] \rangle$$

After two events we reach state 3 (`writeonly`) but cannot take the `write` transition as the guard `size+b <= 16000000` does not hold. As the state is labelled **next** this leads to failure as **next** states must be able to make a move on the next event. This trace violates the property as at least one, in this case both, of the trace slices are not accepted by the instantiated automaton.

As we consider the trace slices τ_1 and τ_2 separately, we would have had the same result for any interleaving of τ_1 and τ_2. This interleaving can be restricted if two trace slices contain the same ground event, as the slices must 'synchronize' on an occurrence of this event in the full trace.

Defining QEA acceptance through trace-slicing. QEA has been formally defined elsewhere [4, 53], here we give a flavor of the structures and notion of trace acceptance. We begin with the basic definitions. An event in the alphabet $\Sigma(X, Y)$ is of the form $e(\overline{z})$ where $\overline{z} \in (X \cup Y \cup \text{Val})^*$ for values Val and disjoint variable sets X and Y. We separate the variables into those that will be quantified X and those that will remain free Y; this distinction will be clarified below. An event is ground if $\overline{z} \in \text{Val}^*$. A trace is a finite sequence of ground events. A binding is a map (partial function with finite domain) from variables to values. Bindings can be applied to events to rewrite their variables. A guard is a predicate on bindings. An assignment is a (partial) function on bindings. An event $e(\overline{z})$ *matches* a ground event $e(\overline{v})$ if there is a binding θ such that $\theta(e(\overline{z})) = e(\overline{v})$ and $\texttt{match}(e(\overline{z}), e(\overline{v}))$ is the minimal such binding with respect to the sub-map relation (if such a binding exists, undefined otherwise).

An event automaton (EA) is a (potentially non-deterministic) finite state machine with alphabet $\Sigma(X, Y)$ where transitions can be labelled with guards and assignments. Recall that states can be either **next** or **skip** as described above. An EA can be *grounded* with a binding with domain X. A grounded EA has an acceptance relation for traces (over its alphabet) defined for configurations (pairs of states and bindings). We do not give this relation here but it is the standard transition relation extended for guards, assignments and the notion of **next** and **skip** states. For an EA $\mathcal{E}$ and a grounding binding θ, the acceptance relation defines a ground language $\mathcal{L}(\theta(\mathcal{E}))$ over $\Sigma(\theta(X), \text{Val})$ as the set of all traces that can reach a configuration with an accepting state. Note that free variables (Y) are replaced by Val as they can take any value that is acceptable to the transition relation i.e. satisfies the guard and assignment structures.

For example, the automaton in Fig. 2 has alphabet $\Sigma(\{f\}, \{size, b\})$. The grounded language for binding $[f \mapsto A]$ would contain traces such as

$$\texttt{open}(A, \text{'R'}, 0).\texttt{read}(A).\texttt{close}(A).\texttt{open}(A, \text{'R'}, 0).\texttt{read}(A).\texttt{close}(A)$$
$$\texttt{open}(A, \text{'W'}, 12000).\texttt{write}(A, 100).\texttt{write}(A, 100).\texttt{close}(A)$$

as they reach an accepting state and satisfy all guards.

A QEA is a pair consisting of a quantifier list $\Lambda(X)$ and an EA over alphabet $\Sigma(X, Y)$. The quantifier list consists of universal and existential quantification over variables X, can include a global guard over X and can be negated. The domain of a quantified variable $x \in X$ is given as those values that can be bound to x when matching events from the EA's alphabet with events in the trace i.e.

$$\mathcal{D}(\tau, x) = \{\texttt{match}(e(\overline{z}), e(\overline{v}))(x) \mid e(\overline{z}) \in \Sigma(X, Y), e(\overline{v}) \in \tau\}$$

In the previous section we extract the domain $\mathcal{D}(\tau, f) = \{\mathrm{A}, \mathrm{B}\}$ as, for example,

$$\begin{array}{ll} \texttt{match}(\texttt{open}(A, \text{`}R\text{`}, 0), \texttt{open}(f, \text{`}R\text{`}, \mathit{size})) & = [f \mapsto A, \mathit{size} \mapsto 0] \\ \texttt{match}(\texttt{close}(B), \texttt{close}(f)) & = [f \mapsto B] \end{array}$$

The notion of trace acceptance is defined using *trace-slicing*. We first consider pure universal quantification. Given a QEA with $\Lambda = \forall x_1 ... \forall x_n$ and EA $\mathcal{E}$, the trace τ is accepted if for *every* binding θ such that $\theta(x_i) \in \mathcal{D}(\tau, x_i)$ the trace $\tau \downarrow_{\Sigma(\theta(X),Y)}$ is in $\mathcal{L}(\theta(\mathcal{E}))$ where the slicing (projection) operation $\downarrow_{\mathcal{A}}$ is defined as

$$\begin{array}{lll} \epsilon \downarrow_{\mathcal{A}} & = \epsilon & \\ \tau.e(\overline{v}) \downarrow_{\mathcal{A}} & = \begin{array}{l} \tau \downarrow_{\mathcal{A}} .e(\overline{v}) \\ \tau \downarrow_{\mathcal{A}} \end{array} & \begin{array}{l} \text{if } \exists e(\overline{z}) \in \mathcal{A} \text{ such that } \texttt{matches}(e(\overline{z}), e(\overline{v})) \\ \text{otherwise} \end{array} \end{array}$$

i.e. for the binding θ we *slice* the trace to give only events relevant to θ, then we check if that trace slice is accepted by the EA grounded with θ. This can be appropriately modified for existential and alternating quantification. For example, the quantifier alternation $\exists x \forall y$ says that there is a value $d \in \mathcal{D}(\tau, x)$ such that $\tau \downarrow_{\Sigma(\theta(X),Y)}$ is in $\mathcal{L}(\theta(\mathcal{E}))$ for every binding θ where $\theta(x) = d$.

Note that this slicing can place ground events in multiple slices i.e. if we had an alphabet $\{\texttt{create}(c, i), \texttt{update}(c), \texttt{use}(i)\}$ for quantified c and i, the event $\texttt{create}(A)$ would be relevant to bindings $[c \mapsto A, i \mapsto 1]$ and $[c \mapsto A, i \mapsto 2]$. Also free variables are ignored for slicing, so an event in the trace matching an event using only free variables would be relevant to all bindings.

Monitoring algorithm. Whilst the above gives a reasonable definition of trace acceptance, this is not a pragmatic method for runtime monitoring as it requires multiple passes of the trace. Instead, an incremental notion of acceptance has been introduced [4, 53] which maintains a map from bindings of quantified variables to sets of configurations. As not all information is available at the start it is necessary to track partial bindings of quantified variables, which requires careful treatment to preserve the semantics described above. The introduction of this map from bindings to configurations lends itself to forms of *indexing*, which is what has made tools that use trace-slicing, such as MOP, highly efficient. MARQ implements a symbol-based form of indexing that uses the alphabet of each (partially) instantiated EA to locate the relevant configurations.

Free versus quantified variables. One aspect of QEA that deserves clarification is the difference between *quantified* and *free* variables. Recall that an EA has two sets of variables, X and Y, and the quantifications range of X, leaving those in Y free. The QEA in Figure 2 has one quantified variable f and two free variables b and size. As we saw in the earlier examples, we use values for f to *slice* the trace and then fix this value when evaluating the trace. The variables b and size are *rebound* whenever they match a new value and can be checked in guards and updated in assignments.

```
class FileUsage extends Monitor {
  "r1" -- 'open('f, 'm, 'size) & not('Open('f, '_, '_)) |->
    insert('Open('f, 'm, 'size))
  "r2" -- 'Open('f, '_, '_) & 'open('f, '_, '_) |-> fail()
  "r3" -- 'Open('f, '_, '_) & 'close('f) |-> remove('Open)
  "r4" -- 'Open('f, 'm, '_) & 'read('f) |-> ensure('m.string == "R")
  "r5" -- 'read('f) & not('Open('f, '_, '_)) |-> fail()
  "r6" -- 'Open('f, 'm, 'size) & 'write('f, 'b) |-> {
    ensure('m.string == "W" && 'size + 'b <= 16000000)
    update('Open('f, 'm, 'size + 'b))
  }
  "r7" -- 'write('f) & not('Open('f, '_, '_)) |-> fail()

  hot('Open)
}
```

Fig. 3. An LogFire model of the file usage property.

Actual guards and assignments. Whilst the qea language theoretically supports arbitrary guards and assignments, the MarQ monitoring tool currently only supports those relating to arithmetic and sets. For arithmetic we have the standard comparison operations (i.e. $=, \leq$), arithmetic operators (i.e. $+, \times$) and update operations (i.e. +=,++). For sets we have set definition, addition and removal, and the contains (in) predicate.

3.3 Introduction to LogFire

LogFire [40] is a rule-based specification language specifically developed for monitoring streams of parameterized events. It is developed as an internal Scala DSL (an API in Scala), allowing a user to freely mix rule programming with traditional programming. It is based on the Rete algorithm [31], specifically as described in [23]. The Rete algorithm is the basis for various rule-based systems. LogFire augments it with a distinction between facts and events and implements an indexing algorithm for optimizing rule evaluation. LogFire can be used for online as well as offline monitoring, although offline monitoring (log analysis) has been the main focus. In the case of online monitoring, the tool would have to be connected to the application via some instrumentation tool, such as AspectJ [46].

Specification of file usage example in LogFire The LogFire specification of the file usage property is given in Fig. 3. LogFire only provides a textual language. One could imagine a graphical notation for rule-based systems, similar to (but yet different from) the visualization of qea models, as illustrated in Figure 2. However, this is not explored in this work. A LogFire monitor is defined as a Scala class that extends (is a sub-class of) the class `Monitor`, which defines all the LogFire primitives (constants, variables and functions), which allows one to write rules. A monitor extending this class must define zero or more rules, each of the form:

```
"name" -- 'cond1 & ... & condn |-> action
```

A rule has a name (a string), followed by a left-hand side, which is a list of conditions separated by conjunction ('`&`'); and a right-hand side following the arrow (`|->`), this is the action. The state of a monitor at any time during the evaluation of an event stream is conceptually (simplified) a set of facts of the form $F(v_1, \ldots, v_n)$, where F is a name and $v_1, \ldots, v_n$ is a list of ground values. This set is referred to as the *fact memory*. Observed events are also facts, which, however, only exist a brief moment when observed. By convention, names of observed events consist of all small letters, whereas names of internally generated facts start with a capital letter. A condition in a rule's left-hand side can check for the presence or absence of a particular fact (including events), and the action on the right-hand side of the rule can add or delete facts, produce error messages, or cause other side effects. Generally, an action can be any SCALA code. Left-hand side matching against the fact memory usually requires unification of variables occurring in conditions. In case all conditions on a rule's left-hand side match (become true), the right-hand side action is executed. This model is very well suited for processing data rich events, and is simple to understand by nature of its very operational semantics. It is interesting to note that finite-state machines can be mapped into rule systems.

The monitor in Figure 3 should be understood as follows. The monitor operates with one fact: $Open(f, m, size)$, representing the fact that file f has been opened in mode m, and currently has size $size$. There are seven rules, named `r1` ... `r7`. Rule `r1` states that upon the occurrence of an `open(f, m, size)` event, if the file has not been opened yet (there is no fact in the fact memory that matches the pattern `Open(f, _, _)`), then an `Open(f, m, size)` fact is inserted into the fact memory. Free identifiers on the left-hand side, appearing as SCALA quoted symbols, such as `'size`, are bound when matched against facts, including events. Rule `r2` states that if a file has been opened, an open event will cause a failure to be reported. Rule `r3` handles the closing of a file, causing the fact matching `Open(f, _, _)` to be removed from the fact memory. Rule `r4` ensures that a file can only be read if it has been opened in read-mode. An alternative, also allowed, formulation of this rule would have been:

```
"r4" -- 'Open('f, "W", '_) & 'read('f) |-> fail()
```

Rule `r5` states that reading a file that is not open is reported as a failure. Rule `r6` captures writing to a file, and ensures that the file has been opened in write-mode, and that the new size does not exceed the upper allowed bound. In addition, the `Open` fact is updated to record the increased size. Rule `r7` states that it is illegal to write to a non-opened file. Finally, the fact `Open` is recorded to be a so-called *hot* fact, which is equivalent to a non-acceptance state in a state machine. When monitoring terminates there should be no hot facts in the fact memory. Figure 4 shows how the fact memory evolves as the events are consumed in the previous trace used to illustrate QEA.

Implementation of LogFire In the presentation above, the configuration of LOGFIRE was described as being a set of facts, those active at any moment during monitoring. Facts can be added or deleted from this set. This explanation is valid

$$\{\} \xrightarrow{\texttt{open}(A,\text{'R'},0)} \{\texttt{Open}(\text{'A'},\text{'R'},0)\} \xrightarrow{\texttt{open}(B,\text{'W'},15999900)} \{\texttt{Open}(\text{'A'},\text{'R'},0),\texttt{Open}(\text{'B'},\text{'W'},15999900)\} \xrightarrow{\texttt{write}(B,100)} \{\texttt{Open}(\text{'A'},\text{'R'},0),\texttt{Open}(\text{'B'},\text{'W'},16000000)\} \xrightarrow{\texttt{read}(A)} \{\texttt{Open}(\text{'A'},\text{'R'},0),\texttt{Open}(\text{'B'},\text{'W'},16000000)\} \xrightarrow{\texttt{write}(B,100)} \text{error}$$

Fig. 4. Evolution (simplified) of LogFire fact memory.

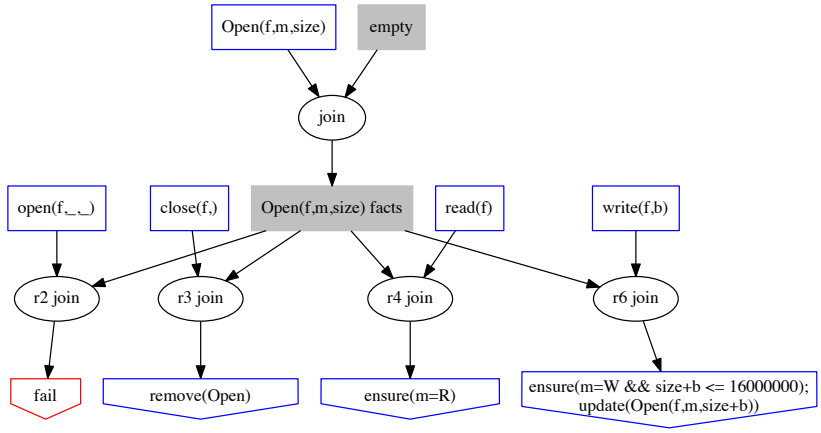


Fig. 5. Rete network for rules `r2`, `r3`, `r4`, and `r6`.

for understanding specifications. However, such a set implementation would be potentially inefficient when evaluating rules against an incoming event. Consider for example the configuration:

$$\{\texttt{Open}(\text{'A'},\text{'R'},0),\texttt{Open}(\text{'B'},\text{'W'},15999900)\}$$

and the incoming event $\texttt{write}(B,100)$. We could now (i) evaluate all the seven rules, one by one, and for each we would (ii) scan the set above to examine if it contains a relevant fact $\texttt{Open}(\text{'B'},\text{'W'},\ldots)$. To avoid this, LogFire implements the Rete algorithm, which optimizes (i). In addition, LogFire augments this algorithm with indexing into this set, which optimizes (ii). The Rete algorithm stores rules and facts as a network, which "glues" rules together that have common prefixes of left-hand sides.

A Rete network for rules `r2`, `r3`, `r4`, and `r6` is shown in Figure 5. Suppose rule `r1` fires and adds an `Open`(f,m,size) fact. This will enter the upper left so-called *alpha memory*. Newly added facts (including events) are added to

alpha memories (white boxes). The top *join node* will be activated and merge the incoming fact with previous facts, of which there are none (the initial so-called *beta memory*, the upper right grey box, is empty). The result is stored in a new beta memory. Each of these beta memories represents a prefix of the conditions in a rule. They contain all facts matching a corresponding condition with pointers back to facts in preceding beta memories, establishing a collection of chains of facts. Since `Open`(f,m,size) occurs as the first condition of rules `r2`, `r3`, `r4`, and `r6`, these now are connected to this beta memory. For example, in the case of a `write`$(B, 100)$ event, the event is added to the right-most alpha memory, whereafter rule `r6`'s join node is activated, corresponding to firing of rule `r6`. Thereby we avoid evaluating all the other rules, corresponding to the optimization of case (i) above.

Concerning optimization (ii), recall that a beta memory contains all facts matching a corresponding condition. It can for example be the set above containing the two facts: `Open`('A','R',0) and `Open`('B','W',15999900). On the observation of a `write`$(B, 100)$ event, rule `r6`'s join node will now have to search for an `Open`('B',...,...) fact in this set. With a large number of facts, this can be costly. To optimize this search, the beta memory is organized as an index from file identifiers to sets of facts:

$$A \mapsto \{\texttt{Open}(\text{'A'}, \text{'R'}, 0)\}$$
$$B \mapsto \{\texttt{Open}(\text{'B'}, \text{'W'}, 15999900)\}$$

It is the join node's responsibility to look up the relevant facts in the beta node when it is activated with an event from an alpha node. The first argument to the `write`$(B, 100)$ event tells the join node to look up B in the index. As such the implementation has some similarity with the slicing approach.

LogFire syntax It remains to briefly explain how rules are interpreted. Consider for example rule `r4`. This rule is by the Scala compiler interpreted as the following chain of method calls:

```
R("r4").--(C('Open).apply(('f, 'm, '_))).&(C('read).apply('f)).|->{
  ensure('m.string == "R")
}
```

Scala allows dots and parentheses around method arguments to be omitted in calls of methods on objects. In the above expansion these have been inserted. In addition, two so-called *implicit functions* `R` and `C` have been applied by the Scala compiler. A user-defined implicit function from type T_1 to type T_2 is applied by the compiler when a value of type T_1 occurs in a place where a value of type T_2 is expected (the function must be unique). For example, `R` is defined as follows, returning an (anonymous) object, which defines the method '`--`', which again returns an object defining the methods '`&`' and '`|->`'.

```
implicit def R(name: String) = new {
  def --(c: Condition) = new RuleDef(name, List(c))
}

class RuleDef(name: String, conditions: List[Condition]) {
```

```
  def &(c: Condition) = new RuleDef(name, c :: conditions)

  def |->(stmt: => Unit) {
    addRule(Rule(name, conditions.reverse, Action((x: Unit) => stmt)))
  }
}
```

4 Specification of Java API properties

The first domain we consider is the task of checking compliance with the JAVA library API. These properties are common examples in publications on runtime verification, and some featured multiple times in the competition. In this case they are all about collections but properties of sockets and streams have been discussed elsewhere [53]. There has also been a systematic effort to formalize the informal properties in the JAVA library documentation [47]. Properties about programming APIs, specifically object-oriented languages like JAVA, have common characteristics. Typically they are about a small set of objects, and if more than one object is targeted then the objects are typically *connected* in some way i.e. one is created from another.

4.1 HasNext

This property applies to every `java.util.Iterator` object, and requires that `hasNext`() be called before `next`() and that `hasNext`() returns true. Two events are monitored: `hasNext`(i, r) is triggered when `hasNext` is called on Iterator i with result r and `next`(i) is triggered when `next` is called on Iterator i.

QEA specification. The specification defines a `safe` and `unsafe` state; a `next` event is only allowed in the `safe` state, which is reached by `hasNext` returning true.

```
qea {
  Forall(i)
  accept skip(unsafe){
    hasnext(i,r) if [ r = true ] -> safe
    next(i) -> failure
  }
  accept skip(safe){
    next(i) -> unsafe
  }
}
```

LogFire specification. The monitor uses one fact, `Safe(i)`, to record when `hasnext` has been called returning true. It is required by `next`, as a guard, and then removed. The monitor, as subsequent LOGFIRE monitors, is named `M` in order to keep naming brief. Similarly, rule names are kept short.

```
class M extends Monitor {
  "r1" -- 'hasnext('i, true) |-> insert('Safe('i))
  "r2" -- 'Safe('i) & 'next('i) |-> remove('Safe)
  "r3" -- 'next('i) & not('Safe('i)) |-> fail()
}
```

4.2 Counting iterator

If a `java.util.Iterator` object is created from a collection of size s then we can only call `next` on that iterator at most s times. Two events are relevant: `iterator`(i,s) records the creation of iterator i from a collection of size s; and `next`(i) records call `next` on iterator i. As iterators are JAVA objects they can only be created once.

QEA specification. This QEA saves the size of the iterator and decreases this size on each `next` event whilst it is safe to do so. As the `iterate` state is a **next** state, a failure will occur if a `next` event occurs and the guard `csize > 0` cannot be satisfied.

```
qea{
  Forall(i)
  accept skip(start){ iterator(i,csize)  -> iterate }
  accept next(iterate) { next(i) if [ csize > 0 ] do [ csize-- ] -> iterate }
}
```

LogFire specification. The monitor uses one fact, `Iterate(i,csize)`, to record that there are `csize` elements left in the collection that iterator `i` is derived from. It is decreased on each observation of a `next`.

```
class M extends Monitor {
  "r1" -- 'iterator('i, 'csize) |-> 'Iterate('i, 'csize)
  "r2" -- 'Iterate('i, 'csize) & 'next('i) |-> {
    if ('csize > 0)
      update('Iterate('i, 'csize - 1))
    else
      fail()
  }
}
```

4.3 UnsafeMapIterator

If a collection is created from a `java.util.Map` object (via calls to `values` or `keySet`) and then an `java.util.Iterator` object is created from that collection, then the iterator cannot be used after the original map has been updated. Four events are relevant: `create`(m,c) records the creation of collection c from map m; `iterator`(c,i) records the creation of iterator i from collection c; `update`(m) records m being updated; and `use`(i) records the usage of iterator i. As collections and iterators are JAVA objects they can only be created once.

QEA specification. This QEA specifies the path to failure i.e. the sequence of events that reach a non-accepting state. Note that quantification is over all maps, collections and iterators. A naive monitoring algorithm would create all such bindings, even for unrelated objects - this is an inherent inefficiency in trace-slicing that must be avoided through careful implementation and extension of the theory. Note the use of **skip** states to ignore irrelevant events.

```
qea{
  Forall(m,c,i)
  accept skip(start){  create(m,c) -> hascol }
  accept skip(hascol){ iterator(c,i) -> hasit }
  accept skip(hasit){  update(m) -> updatedm }
  accept skip(updatedm){ use(i) -> failure }
}
```

LogFire specification. The monitor uses three facts: `HasCol(m,c)` to record when collection `c` has been created from a map `m`; `HasIt(m,i)` to record that iterator `i` has been created from a collection created from map `m`; and finally `UpdateM(i)` to record that the map that iterator `i` is derived from has been updated, and hence no further iteration (use) is allowed.

```
class M extends Monitor {
  "r1" -- 'create('m, 'c) |-> insert('HasCol('m, 'c))
  "r2" -- 'HasCol('m, 'c) & 'iterator('c, 'i) |-> insert('HasIt('m, 'i))
  "r3" -- 'HasIt('m, 'i) & 'update('m) |-> insert('UpdateM('i))
  "r4" -- 'UpdateM('i) & 'use('i) |-> fail()
}
```

4.4 Hashing persistence

Objects placed in a hashing structure, such as a `HashSet` or `HashMap`, should have persistent hash codes whilst in the structure for the usage to be sound. Otherwise we may have the situation where we add an object and then get the result false when checking for its presence. Three events are relevant: `add`(c,o,h), `observe`(c,o,h) and `remove`(c,o,h) respectively record the addition, observation and removal of object o on hashing collection c using hash code h.

QEA specification. We have chosen not to quantify over collections in this QEA as it is possible to specify the property by quantifying over objects only. Note that monitoring complexity depends on the number of bindings, which can be, in the worst case, exponential in the number of quantified variables. This specification works by tracking how many collections the object is in and ensuring that the hash code remains persistent whilst this is non-zero. This assumes that the `add` event only occurs when the object did not previously exist in the collection, and `remove` only occurs when it did. If these assumptions cannot be enforced (via instrumentation) then the specification would need to quantify over c.

```
qea{
  Forall(o)
  accept skip(out){
    add(c,o,h) do [ count:=1; ] -> in
  }
  accept next(in){
    add(c,o,h2) if [ h = h2 ] do [ count++ ] -> in
    observe(c,o,h2) if [ h = h2 ] -> in
    remove(c,o,h2) if [ count > 1 and h = h2 ] do [ count-- ] -> in
    remove(c,o,h2) if [ count = 1 and h = h2 ] -> out
  }
}
```

LogFire specification. The monitor uses one fact, `In(c,o,h)`, to record that collection `c` contains object `o`, which had hash code `h` when first added. This should remain the hash code as long as the object is in that collection.

```
class M extends Monitor {
  "r1" -- 'add('c, 'o, 'h) |-> insert('In('c, 'o, 'h))
  "r2" -- 'In('c, 'o, 'h1) & 'observe('c, 'o, 'h2) |-> ensure('h1.i == 'h2.i)
  "r3" -- 'In('c, 'o, 'h1) & 'add('c, 'o, 'h2) |-> ensure('h1.i == 'h2.i)
  "r4" -- 'In('c, 'o, 'h1) & 'remove('c, 'o, 'h2) |-> {
    ensure('h1.i == 'h2.i)
    remove('In)
  }
}
```

5 Specification of banking properties

The next application domain we consider is that of banking. The following properties are concerned with accounts and transfers. There is a focus on timed properties i.e. ensuring that an action occurs within a given timeframe. In both logics time is modeled as time stamps, which are just data.

5.1 Unique accounts

An account approved by the administrator may not have the same account number as any other already existing account in the system. The event `approve`(id) indicates that an account with id has been approved.

QEA specification. To specify this property in a pure trace-slicing style we would quantify over ids and record a failure if two events occur with the same quantified id.

```
qea{
  Forall(id)
  accept skip(start){ approve(id) -> once }
  accept skip(once) { approve(id) -> failure }
}
```

However, with QEAs we can also maintain a set S of previously seen account ids. This is more like the fact-based approach taken by LogFire. It does not utilize the trace-slicing mechanisms but will be more efficient for this very simple property as trace-slicing, and the associated indexing, is not required.

```
qea{
  accept next(safe){ approve(id) if[ not(id in S) ] do[ S.add(id) ] -> safe }
}
```

LogFire specification. The monitor uses one fact, `Approved(id)`, to record that account `id` has been approved.

```
class M extends Monitor {
  "r1" -- 'approve('id) |-> insert('Approved('id))
  "r2" -- 'Approved('id) & 'approve('id) |-> fail()
}
```

5.2 Greylisting

Once grey-listed, a user must perform at least three incoming transfers before being white-listed. There are three relevant events. `greyList`(*user*) and `whiteList`(*user*) indicate that *user* was grey and white-listed respectively and `transfer`(*user*) records the fact that *user* performed a transfer.

QEA specification. This specification has two states indicating the status of the user. The number of transfers are counted in the `grey` state (and zeroed on a `greyList` event) so that this count can be checked at a `whiteList` event.

```
qea{
  Forall(u)
  accept skip(white){
    greyList(u) do [ count:=0 ] -> grey
  }
  accept next(grey){
    transfer(u) do [ count++ ] -> grey
    greyList(u) do [ count:=0 ] -> grey
    whiteList(u) if [count >= 3 ] -> white
  }

}
```

LogFire specification The monitor uses one fact, `Grey(u,count)`, to record that user `u` has been grey-listed, and since then has had `count` incoming money transfers, initially 0.

```
class M extends Monitor {
  "r1" -- 'greyList('u) & not('Grey('u, '_)) |-> insert('Grey('u, 0))
  "r2" -- 'Grey('u, '_) & 'greyList('u) |-> update('Grey('u, 0))
  "r3" -- 'Grey('u, 'count) & 'transfer('u) |-> update('Grey('u, 'count + 1))
  "r4" -- 'Grey('u, 'count) & 'whiteList('u) |-> {
    if ('count < 3)
      fail()
    else
      remove('Grey)
  }
}
```

5.3 Reconciling accounts

The administrator must reconcile accounts every 1000 attempted external money transfers or an aggregate total of one million dollars in attempted external transfers. There are two events: `reconcile` is a propositional event that indicates that all accounts have been reconciled; and `transfer`(*amount*) indicates that *amount* was transferred. Invalid traces have two forms. The first is where there are more than 1000 events between `reconcile` events and the second is where the sum of amounts exceeds one million dollars.

QEA specification. Here the `safe` state indicates some safe amount has been transferred and we can only stay in this state if each transfer is safe. To ensure this a running count and total is kept.

```
qea{
  accept next(start){
    transfer(amount) do[ count:=1; total:=amount ] -> safe
  }
  accept next(safe){
    transfer(amount) if[ count < 1000 and total < 1000000 ]
                     do[ count++; total += amount ] -> safe
    reconcile do[ count:=0; total:= 0] -> safe
  }
}
```

LogFire specification The monitor uses one fact, `Sums(count,total)`, carrying two sums: `count`, which is the number of transfers since the last reconciliation, and `total`, which is the total sum of money transferred since the last reconciliation. An initial `Sums(0,0)` fact is added to the fact memory (`addFact('Sums)(0,0)`) before monitoring starts.

```
class M extends Monitor {
  "r1" -- 'Sums('_, '_) & 'reconcile() |-> update('Sums(0, 0))
  "r2" -- 'Sums('count, 'total) & 'transfer('a) |-> {
    if ('count + 1 > 1000 || 'total + 'a > 1000000)
      fail()
    else
      update('Sums('count + 1, 'total + 'a))
  }

  addFact('Sums)(0,0)
}
```

The monitor above purely uses rules to implement the property. However, we can, as in the QEA monitor, use global variables `count` and `total`. This is shown in the following version, illustrating how LogFire monitors can mix rules and programming:

```
class M extends Monitor {
  var count: Int = 0
  var total: Int = 0

  "r1" -- 'reconcile() |-> {count = 0; total = 0}
  "r2" -- 'transfer('a) |-> {
    if (count + 1 > 1000 || total + 'a > 1000000)
      fail()
    else {
      count += 1; total += 'a
    }
  }
}
```

5.4 Maximum withdrawals

The number of withdrawal operations performed within 10 minutes before a customer logs off is less than or equal to the allowed limit of 3. It is assumed that a trace records the activities of a single user. A user is not required to log off. There are two events of interest: `withdraw`(*time*) records the time that

a customer made a withdrawal and `logoff`(*time*) records the time that the customer logged off.

QEA specification. This QEA can be read as 'we do not reach failure' as it is a negated QEA accepting invalid traces. The QEA is non-deterministic, creating a new configuration per time window. A failure is detected if one of these configurations reaches the **success** state, as the specification is negated. The combination of non-determinism and negation is required as a trace is accepted if *at least one path* reaches an accepting state, therefore we use this approach if we want failure when *all paths* are required to reach an accepting state.

On each `withdraw` event a new configuration is created in the `safe` state with count 1. On each subsequent `withdraw` event (within the time window) this count is increased until it reaches 3 and the configuration is transferred to the `unsafe` state. Any `logoff` occurring within the time window for a configuration in the `unsafe` state will lead to failure. Note that, due to the use of **next** states, configurations will be removed if an event occurs that does not satisfy any transitions i.e. one that happens outside of that configuration's time window.

```
qea{
  Negated
  skip(start){
    withdraw(t1) do[ count:=1 ] -> safe
    withdraw(_) -> start
  }
  next(safe){
    withdraw(t2) if[ t2-t1 <= 10 and count < 3 ] do [ count++ ] -> safe
    withdraw(t2) if[ t2-t1 <= 10 and count = 3 ] do [ count++ ] -> unsafe
  }
  next(unsafe){
    withdraw(t2) if[ t2-t1 <= 10 ] -> unsafe
    logoff(t2)   if[ t2-t1 <= 10 ] -> success
  }
}
```

LogFire specification The monitor uses one fact, `Count(time,count)`, which is created upon each withdrawal event, using the same form of non-determinism as used in the QEA specification. It tracks the number of withdrawals since (and including) that withdrawal, should a logoff occur. It carries two pieces of data: `time`, which is the time of the withdrawal, and `count`, which is the number of withdrawals since then.

```
class M extends Monitor {
  "r1" -- 'withdraw('time) |-> insert('Count('time, 1))
  "r2" -- 'Count('time, 'count) & 'withdraw('time2) |-> {
    if ('time2 - 'time <= 10)
      update('Count('time, 'count + 1))
    else
      remove('Count)
  }
  "r3" -- 'Count('time, 'count) & 'logoff('time2) |-> {
    ensure('time2 - 'time > 10 || 'count <= 3)
    remove('Count)
  }
}
```

5.5 Transaction limit reporting

The property requires that a client's executed transactions must be reported within at most 5 days if the transferred amount exceeds a given threshold of $2,000. A transaction only occurs once. The event `trans`(c, t, a, ts) denotes that the client c performs transaction t, transferring the amount a at timestamp ts. The `report`(t, ts) event denotes that transaction t is reported at timestamp ts.

QEA specification. In this specification, making a transfer for an amount more than 2000 moves us into an `unsafe` state for a transaction. A `report` event within 5 days takes us to a `safe` state. To ensure that failures are captured early, this specification only allows `transfer` events for other clients to occur within the 5 day waiting period.

```
qea{
  Forall(t)
  accept skip(safe){ trans(t,a,ts1) if [ a > 2000 ] -> unsafe }
  skip(unsafe){
    trans(_,_,ts2) if[ ts2-ts1 > 5 ] -> failure
    report(t,ts2)  if[ ts2-ts1 <= 5 ] -> success
  }
}
```

LogFire specification The monitor uses one fact, `Unsafe(t,ts)`, representing the fact that a transaction `t` occurred at time `ts` of more than $2,000$ dollars. This fact is declared hot, meaning that monitoring should not terminate with such a fact in the fact memory – it has to eventually be reported (and within 5 days).

```
class M extends Monitor {
  "r1" -- 'transfer('t, 'a, 'ts) |-> {
    if ('a > 2000) insert('Unsafe('t, 'ts))
  }
  "r2" -- 'Unsafe('t, 'ts1) & 'report('t, 'ts2) |-> {
    ensure('ts2 - 'ts1 <= 5)
    remove('Unsafe)
  }

  hot('Unsafe)
}
```

5.6 Transaction limit authorization

The property requires that executed transactions of any customer must be authorized by some employee before they are executed if the transferred money exceeds a given threshold of $2,000. Authorization only lasts for 21 days and a transaction must be authorized at least 2 days before it is made. A transaction can only occur once. The `trans`(t,a,ts) indicates transaction t occurred at time ts for amount a. The event `auth`(t, ts), indicates the authorization of the transaction t at timestamp ts.

QEA specification. In the `unauth` state transactions can only be below 2,000. The **`auth`** event takes us to the `auth` state, where any transaction can occur as long as we are inside the authorized period.

```
qea{
  Forall(t)
  accept skip(unauth){
    trans(t,a,_) if [ a < 2000 ] -> success
    trans(t,a,_) if [ a >= 2000 ] -> failure
    auth(t,auth_ts) -> auth
  }
  accept skip(auth){
    trans(t,a,ts) if [ ts-auth_ts >= 2 and ts-auth_ts < 21 ] -> success
    trans(t,a,ts) if [ ts-auth_ts < 2  and ts-auth_ts > 21 ] -> failure
    auth(t,auth_ts) -> auth
  }
}
```

LogFire specification The monitor uses one fact, `Auth(t,ts)`, representing the fact that transaction `t` has been authorized at time `ts`. Note how authorizations are updated in case such exist.

```
class M extends Monitor {
  "r1" -- 'auth('t, 'ts) & not('Auth('t, '_)) |-> insert('Auth('t, 'ts))
  "r2" -- 'Auth('t, '_) & 'auth('t, 'ts) |-> update('Auth('t, 'ts))
  "r3" -- 'Auth('t, 'ts1) & 'trans('t, 'a, 'ts2) |-> {
    ensure('a < 2000 || ('ts2 - 'ts1 >= 2 && 'ts2 - 'ts1 < 21))
    remove('Auth)
  }
  "r4" -- 'trans('t, 'a, '_) & not('Auth('t, '_)) |-> {
    ensure('a < 2000)
  }
}
```

5.7 Report approval

The property represents an approval policy for publishing business reports within a company. The property requires that any report must be approved prior to its publication. Furthermore, the property asks that the person who publishes the report must be an accountant and the person who approves the publication must be the accountants manager at the time of the approval. Finally, the approval must happen within at most 10 days before the publication.

The event $\texttt{publish}(a, f, ts)$ denotes the publication of the report f by the accountant a at timestamp ts. The event $\texttt{approve}(m, f, ts)$ denotes the publishing approval of the report f by the manager m at timestamp ts. The event $\texttt{mgr\_S}(m, a)$ marks the time when m starts being manager of accountant a, and the event $\texttt{mgr\_F}(m, a)$ marks the corresponding finishing time. Analogously, $\texttt{acc\_S}(a)$ and $\texttt{acc\_F}(a)$ mark the starting and finishing times when a is an accountant

QEA specification. This specification keeps track of the current managers of accountant a in set M. There are then two states: `nota` and `isa` indicate that the accountant is not approved or is. Then `ts_app` tracks the most recent

time at which the report was legitimately approved so that this can be checked on a `publish` event. In QEA it is assumed that sets are empty on their first occurrence. The **defined** predicate is true if a variable has been given a value.

```
qea{
  Forall(f,a)
  accept next(nota){
    mgr_S(m,a) do [ M.add(m) ] -> nota
    mgr_F(m,a) do [ M.remove(m) ] -> nota
    approve(m,f,ts_app) if [ m in M ] -> nota
    acc_S(a) -> isa
  }
  accept next(isa){
    mgr_S(m,a) do [ M.add(m) ] -> isa
    mgr_F(m,a) do [ M.remove(m) ] -> isa
    approve(m,f,ts_app) if [ m in M ] -> isa
    acc_F(a) -> nota
    publish(a,f,ts_pub) if [ defined(ts_app) and ts_pub - ts_app < 10 ] -> isa
  }
}
```

LogFire specification The monitor uses the following facts: `Mgr(m,a)` to record that manager `m` is manager of accountant `a`; `Acc(a)` to record that `a` is an accountant; and finally `Appr(a,f,ts)` to record that accountant `a` at time `ts` has been approved by one of his/her managers to publish report `f`. Note how a `Appr(a,f,ts)` fact is generated for each accountant `a` that a manager is manager of when he/she approves a report `f`. Note also how approvals are updated in case such exist.

```
class M extends Monitor {
  "r1" -- 'mgr_S('m, 'a) |-> 'Mgr('m, 'a)
  "r2" -- 'Mgr('m, 'a) & 'mgr_F('m, 'a) |-> remove('Mgr)
  "r3" -- 'acc_S('a) |-> 'Acc('a)
  "r4" -- 'Acc('a) & 'acc_F('a) |-> remove('Acc)
  "r5" -- 'approve('m, 'f, 'ts) & 'Mgr('m, 'a) & not('Appr('a, 'f, '_)) |->
    insert('Appr('a, 'f, 'ts))
  "r6" -- 'Appr('a, 'f, '_) & 'approve('m, 'f, 'ts) & 'Mgr('m, 'a) |->
    update('Appr('a, 'f, 'ts))
  "r7" -- 'publish('a, '_, '_) & not('Acc('a)) |-> fail()
  "r8" -- 'publish('a, 'f, '_) & not('Appr('a, 'f, '_)) |-> fail()
  "r9" -- 'Appr('a, 'f, 'ts1) & 'publish('a, 'f, 'ts2) |-> ensure('ts2 - 'ts1 <= 10)
}
```

5.8 Withdrawal limit

The property is rooted in the domain of fraud detection. The property requires that the sum of withdrawals of each user in the last 28 days does not exceed the limit of $10,000. The event `withdraw`$(u, a, ts)$ denotes the withdrawal of the amount a by the user u at timestamp ts.

QEA specification. This is another specification of the form 'there does not exist a path to failure'. This time we capture the behavior required to perform a bad withdraw. On each `withdraw` event we create a new configuration that represents the start of a new 28 day period.

```
qea  {

  Negated
  Exists(u)

  skip(start){
    withdraw(u,a,_) -> start
    withdraw(u,a,ts) do[s:=a] -> withdrawn
    withdraw(u,a,_) if[a > 10000] -> success
  }
  next(withdrawn){
    withdraw(u,a,ts2) if[ ts2-ts < 28 and s+a >  10000 ] -> success
    withdraw(u,a,ts2) if[ ts2-ts < 28 and s+a <= 10000 ]
                      do[s+=a] -> withdrawn
  }
}
```

LogFire specification The monitor uses one fact, `Withdrawn(u,a,ts)`, to record that user `u` withdrew `a` dollars at time `ts` (not including amounts greater than 10,000). Such a fact is produced for each withdrawal, and is used to monitor the next 28 days from that point, accumulating the withdrawn amounts.

```
class M extends Monitor {
  "r1" -- 'withdraw('u, 'a, 'ts) |-> {
    if ('a <= 10000)
      insert('Withdrawn('u, 'a, 'ts))
    else
      fail()
  }
  "r2" -- 'Withdrawn('u, 'sum, 'ts1) & 'withdraw('u, 'a, 'ts2) |-> {
    if ('ts2 - 'ts1 <= 28) {
      val newSum = 'sum + 'a
      if (newSum <= 10000)
        update('Withdrawn('u, newSum, 'ts1))
      else
        fail()
    } else remove('Withdrawn)
  }
}
```

6 Specification of rover properties

In this set of properties we consider the operation of planetary rovers. The first two properties consider communication and the last three consider internal resource management.

6.1 Rover coordination

This property relates to the self-organization of communicating rovers and captures the situation where (at least) one rover is able to communicate with all other (known) rovers. The property states that there exists a leader (rover) who has pinged every other (known) rover and received an acknowledgement. The events `ping`(*from*,*to*) and `ack`(*to*,*from*) indicate that *from* pinged *to* and *to* acknowledged *from* respectively. The leader does not need to have pinged itself. The set of known rovers are those that ping/ack or have been pinged/acked.

QEA specification. This specification uses quantifier alternation to capture the property that there is one rover `r1` that sends `ping` and receives an `ack` from every other different rover `r2`. The **Join** declaration indicates that the domains of `r1` and `r2` should be joined i.e. if a value is considered for `r1` it should also be considered for `r2` and vice-versa.

```
qea{
  Exists(r1) Forall(r2) Where(r1!=r2) Join(r1,r2)
  skip(start) { ping(r1,r2) -> pinged  }
  skip(pinged){ ack(r2,r1)  -> success }
}
```

LogFire specification. The monitor uses the following facts: `Ping(r1,r2)` to record that rover `r1` pings rover `r2`; `Node(r)` to record that `r` is a node; `Reach(r1,r2)` to record that node `r1` has pinged node `r2`, and that `r2` has acknowledged back; `Cand(r)` to record that rover `r` is a candidate as leader, all nodes are declared as candidates; and finally `End()` records the end of the trace, at which point the final computation is performed. At that point candidates are removed if there is some node they do not reach. If no candidates are left at some point an error is reported, there is no leader.

```
class M extends Monitor {
  "r1" -- 'ping('r1, 'r2) |-> {
    insert('Node('r1))
    insert('Node('r2))
    insert('Ping('r1, 'r2))
  }
  "r2" -- 'ack('r1, 'r2) |-> {
    insert('Node('r1))
    insert('Node('r2))
  }
  "r3" -- 'Node('r) |-> {
    insert('Cand('r))
    insert('Reach('r, 'r))
  }
  "r4" -- 'Ping('r1, 'r2) & 'ack('r2, 'r1) |-> {
    insert('Reach('r1, 'r2))
    remove('Ping)
  }
  "r5" -- 'end() |-> 'End()
  "r6" -- 'End() & 'Cand('r1) & 'Node('r2) & not('Reach('r1, 'r2)) |->
    remove('Cand)
  "r7" -- 'End() & not('Cand('_)) |-> fail()
}
```

The LogFire specification is clearly more complicated than the QEA automaton. In LogFire we are limited since we cannot directly model a universal quantifier nested under an existential quantifier as in the following quantified linear temporal logic formula: $\exists leader \bullet \forall n \bullet \Diamond(ping(leader, n) \wedge \Diamond ack(n, leader))$, where $\Diamond\psi$ has the classical meaning: eventually ψ for some temporal formula ψ.

6.2 Command nesting

If a command with identifier B starts after a command with identifier A has started, then command B must succeed before command A succeeds (last issued – first to succeed). A command can only be started and succeed once. The events `com`(id) and `suc`(id) record the issuing and success of command id respectively.

QEA specification. This specification captures the property that command `c2` is nested inside command `c1`. Note that, due to the symmetry of `c1` and `c2`, for every two values there will be two instances of the QEA considering each command being nested inside the other. As the trace is considered after instantiation, the events `com(c1)` and `com(c2)` are distinct as they will be instantiated with different values. The states capture different stages in each command. Note that the events `com(c2)` and `suc(c2)` on the initial state are necessary if `c2` is not nested inside `c1`. Similarly, the `finishedEarly` state is entered if `c2` only occurs after `c1` succeeds.

```
qea{
  Forall(c1,c2)
  accept next(none){
    com(c2) -> none; suc(c2) -> none
    com(c1) -> startedOne
  }
  accept next(startedOne){
    com(c2) -> startedTwo
    suc(c2) -> finishedEarly
  }
  accept next(startedTwo){
    suc(c2) -> finishedTwo
  }
  accept next(finishedTwo){
    suc(c1) -> finished
  }
  accept next(finishedEarly){
    com(c2) -> finishedEarly; suc(c2) -> finishedEarly
  }
  accept next(finished){}
}
```

LogFire specification. The monitor uses the following facts: `Com(x)` to record that command `x` has been issued (this fact is never deleted); `Suc(x)` to record that command `x` has succeeded (this fact is never deleted); and finally `Ord(x,y)` to record that command `y` has been issued after command `x`, and therefore must succeed before command `x`.

```
class M extends Monitor {
  "r1" -- 'com('x) |-> insert('Com('x))
  "r2" -- 'Com('x) & 'com('x) |-> fail()
  "r3" -- 'Com('x) & 'suc('x) |-> insert('Suc('x))
  "r4" -- 'suc('x) & not('Com('x)) |-> fail()
  "r5" -- 'Suc('x) & 'suc('x) |-> fail()
  "r6" -- 'Com('x) & not('Suc('x)) & 'com('y) |-> 'Ord('x, 'y)
  "r7" -- 'Ord('x, 'y) & 'suc('y) |-> remove('Ord)
  "r8" -- 'Ord('x, 'y) & 'suc('x) |-> fail()
}
```

6.3 Resource lifecycle

This property represents the lifecycle of a resource with respect to a task, as managed by a planetary rover's internal resource management system - or any resource management system in general. The lifecycle goes as follows:

– A resource may be requested by the task

- A requested resource may be denied or granted to the task
- A granted resource may be rescinded or cancelled
- A resource may only be requested by a task if that task does not currently hold the resource
- A granted resource must eventually be cancelled

We use the events `request`(t, r), `deny`(t, r), `grant`(t, r), `rescind`(t, r) and `cancel` (t, r) for a task t and resource r.

QEA specification. The specification captures the three valid states for a resource, with respect to a task, and the valid transitions for each state.

```
qea{
  Forall(t,r)
  accept next(free){
    request(t,r) -> requested
  }
  accept next(requested){
    deny(t,r) -> free
    grant(t,r) -> granted
  }
  accept next(granted){
    cancel(t,r) -> free
    rescind(t,r) -> granted
  }
}
```

LogFire specification. The monitor uses the following two facts: `Req(t,r)` to record that resource `r` has been requested by task `t`; and `Grant(t,r)` to record that resource `r` has been granted to task `t`.

```
class M extends Monitor {
  "r1" -- 'request('t,'r) |-> insert('Req('t,'r))
  "r2" -- 'Req('t,'r) & 'deny('t,'r) |-> remove('Req)
  "r3" -- 'Req('t,'r) & 'grant('t,'r) |-> {
    remove('Req)
    insert('Grant('t,'r))
  }
  "r4" -- 'Grant('t,'r) & 'cancel('t,'r) |-> remove('Grant)
  "r5" -- 'deny('t,'r) & not('Req('t,'r)) |-> fail()
  "r6" -- 'grant('t,'r) & not('Req('t,'r)) |-> fail()
  "r7" -- 'request('t,'r) & 'Grant('t,'r) |-> fail()
  "r8" -- 'rescind('t,'r) & not('Grant('t,'r)) |-> fail()
  "r9" -- 'cancel('t,'r) & not('Grant('t,'r)) |-> fail()

  hot('Grant)
}
```

6.4 Resource management

Every resource should only be held by at most one task at any one time. If a resource is granted to a task it should be cancelled before being granted to another task. This is therefore a mutual exclusion property. The event `grant`(t, r) captures that task t is granted resource r, similarly `cancel`(t, r) captures that task t releases resource r.

QEA specification. This specification captures the notion of a bad grant. A bad grant occurs when a resource has been granted to a task and is then granted again to any task (including the task holding the resource). It also uses a guard to ensure that the task granted the resource is the task that cancels the resource.

```
qea{
  Forall(r)
  accept next(free){ grant(t1,r) -> granted }
  accept next(granted){
    grant(_,r) -> failure
    cancel(t2,r) if [ t1 = t2 ] -> free
  }
}
```

LogFire specification. The monitor uses one fact, `Granted(t,r)`, representing that task `t` has been granted resource `r`.

```
class M extends Monitor {
  "r1" -- 'grant('t, 'r) & not('Granted('_, 'r)) |-> insert('Granted('t, 'r))
  "r2" -- 'Granted('_, 'r) & 'grant('_, 'r) |-> fail()
  "r3" -- 'Granted('t, 'r) & 'cancel('t, 'r) |-> remove('Granted)
  "r4" -- 'cancel('t, 'r) & not('Granted('t, 'r)) |-> fail()
}
```

6.5 Resource conflict management

This property represents the management of conflicts between resources as managed by a planetary rovers internal resource management system - or any resource management system in general. It is assumed that conflicts between resources are declared at the beginning of operation. After this point resources that are in conflict with each other cannot be granted at the same time. A conflict between resources r_1 and r_2 is captured by the event `conflict`(r_1,r_2) and a conflict is symmetrical. Resources are granted and cancelled using `grant`(r) and `cancel`(r) respectively.

QEA specification. The specification quantifies over two resources and has two separate states representing each resource being granted (after being put in conflict). Note the symmetry of the `conflict` events required to capture the relationship. Elsewhere [53] this property has been specified differently with efficiency in mind; an encoding of the property that would be more efficient to monitor would replace the `r2` quantification with a set that collects all resources in conflict with `r1`. This would be more efficient as the monitoring algorithm is exponential in the number of quantified variables.

```
qea{
  Forall(r1,r2)
  accept skip(start){
    conflict(r1,r2) -> free
    conflict(r2,r1) -> free
  }
  accept skip(free){
    grant(r1) -> granted1
    grant(r2) -> granted2
```

```
  }
  accept next(granted1){
    cancel(r1) -> free
  }
  accept next(granted2){
    cancel(r2) -> free
  }
}
```

LogFire specification. The facts used by the monitor: `Conflict(r1,r2)` to record that there is a conflict between resources `r1` and `r2` – for each such conflict added its symmetric fact is also added; `Granted(r)` to record that resource `r` has been granted; and finally `Locks(r1,r2)` to record that resource `r1` has been granted, which is in conflict with `r2`, which therefore is locked from being granted also. The `Lock` predicate is needed since LogFire currently does not permit negation of conjunctions in conditions.

```
class M extends Monitor {
  "r1" -- 'conflict('r1, 'r2) |-> {
    insert('Conflict('r1, 'r2))
    insert('Conflict('r2, 'r1))
  }
  "r2" -- 'grant('r) & not('Granted('r)) & not('Locks('_, 'r)) |->
    insert('Granted('r))
  "r3" -- 'Granted('r) & 'grant('r) |-> fail()
  "r4" -- 'Locks('_, 'r) & 'grant('r) |-> fail()
  "r5" -- 'Granted('r1) & 'Conflict('r1, 'r2) |-> 'Locks('r1, 'r2)
  "r6" -- 'Granted('r) & 'cancel('r) |-> remove('Granted)
  "r7" -- 'Locks('r1,'r2) & 'cancel('r1) |-> remove('Locks)
  "r8" -- 'cancel('r) & not('Granted('r)) |-> fail()
}
```

7 Specification of concurrency properties

Finally we consider two properties related to concurrency and synchronization via locks.

7.1 Lock nesting

A thread should release a lock as many times as it acquires the lock. Additionally, locks taken within a call to a method should be released during that call. This property therefore represents a *double nesting* of method calls and lock taking. Abstractly this could be viewed as parenthesis matching for two kinds of parenthesis. The four events of interest are `begin`(t) and `end`(t), which respectively record the beginning and end of a method for thread t, and `lock`(t,l) and `unlock`(t,l), which respectively record the locking and unlocking of lock l by thread t.

QEA specification. This QEA specifies paths to failure for a thread and lock using the negated existential pattern seen earlier. The first path to the `failed` state via the `locked` state is followed when the lock is held when exiting the

method it was taken in. There also exists a set of paths that finish in the `inside` or `locked` states when either a method is not exited or a lock not unlocked. Finally, unlocking a lock that is not locked will also lead to failure. The behavior is captured using the counter `depth` to track the depth of the thread's call stack. Each **`begin`** event creates a new configuration `inside` a method call; this effectively attempts to find a failing path for each suffix of the trace starting with a **`begin`** event. The `count` counter tracks the lock depth.

```
qea{
 Negated
 Exists(t,l)
 skip(outside){
    begin(t) do [ depth:=1 ] -> inside
    begin(t) -> outside
  }
  accept skip(inside){
    begin(t) do [ depth++ ] -> inside
    end(t) if [ depth = 1 ] do [ depth:=0] -> finished
    end(t) if [ depth > 1 ] do [ depth -- ] -> inside
    lock(t,l) do [ count:=1 ] -> locked
    unlock(t,l) -> failed
  }
  skip finished {}
  accept skip (locked){
    lock(t,l) do [ count++ ] -> locked
    unlock(t,l) if [count > 1 ] do [ count  ] -> locked
    unlock(t,l) if [ count =1 ] do [ count:=0 ] -> inside
    begin(t) do [ depth++ ] -> locked
    end(t) if [ depth > 1 ] do [ depth  ] -> locked
    end(t) if [ depth = 1 ] -> failed
  }
  accept skip(failed){}
}
```

LogFire specification The monitor uses the following two facts: `Inside(t,d)` to record that thread `t` is currently at a method activation depth of `d` (it has called methods nested `d` times without returning from any of them); and `Locked(t,l,d,c)` to record that thread `t` has taken lock `l` a total of `c` times while at activation depth level `d`.

```
class M extends Monitor {
  "r1" -- 'begin('t) & not('Inside('t, '_)) |-> insert('Inside('t, 1))
  "r2" -- 'Inside('t, 'd) & 'begin('t) |-> update('Inside('t, 'd + 1))
  "r3" -- 'Inside('t, 'd) & 'end('t) |-> {
    if ('d.int > 1)
      update('Inside('t, 'd - 1))
    else
      remove('Inside)
  }

  "r4" -- 'Inside('t, 'd) & 'lock('t, 'l) & not('Locked('t, 'l, 'd, '_)) |->
    insert('Locked('t, 'l, 'd, 1))
  "r5" -- 'Inside('t, 'd) & 'Locked('t, 'l, 'd, 'c) & 'lock('t, 'l) |->
    update('Locked('t, 'l, 'd, 'c + 1))
  "r6" -- 'Inside('t, 'd) & 'Locked('t, 'l, 'd, 'c) & 'unlock('t, 'l) |-> {
    if ('c > 1)
      update('Locked('t, 'l, 'd, 'c - 1))
    else
      remove('Locked)
  }
  "r7" -- 'Inside('t, 'd) & 'unlock('t, 'l) & not('Locked('t, 'l, 'd, '_)) |->
```

```
    fail()
  "r8" -- 'Inside('t, 'd) & 'Locked('t, '_, 'd, '_) & 'end('t) |-> fail()

  hot('Locked)
}
```

7.2 Lock ordering

This property represents a conservative deadlock-avoidance strategy that prevents cycles between locks by enforcing a partial ordering on locks: a thread can only take a lock L_2 while holding a lock L_1 if L_1 precedes L_2 in the partial ordering. The property states that for every two (different) locks, if they are taken in one order in one part of the system, then they are not taken in the opposite order in another part of the system. The events `lock`(t,l) and `unlock`(t,l) respectively capture the locking and unlocking of lock l by thread t.

QEA specification. This specification quantifies over a pair of threads and a pair of (distinct) locks. If the locks are taken by the first thread in one order, then they cannot be taken in a different order by the second thread.

```
qea{
  Forall(t1,t2,l1,l2)
  Where(l1 != l2)

  accept skip(start){ lock(t1,l1) -> lock1 }
  accept skip(lock1){
    unlock(t1,l1) -> start
    lock(t1,l2) -> lock12
  }
  accept skip(lock12){ lock(t2,l2) -> lock122 }
  accept skip(lock122){
    unlock(t2,l2) -> lock12
    lock(t2,l1) -> failure
  }
}
```

LogFire specification The monitor uses the following two facts: `Locked(t,l)` to record that thread `t` has taken lock `l` and not yet released it; and `Edge(l1,l2)` to record that thread `t` at some point held lock `l1`, while nested acquiring lock `l2`.

```
class M extends Monitor {
  "r1" -- 'lock('t, 'l) |-> insert('Locked('t, 'l))
  "r2" -- 'Locked('t, 'l) & 'unlock('t, 'l) |-> remove('Locked)
  "r3" -- 'Locked('t, 'l1) & 'lock('t, 'l2) |-> insert('Edge('l1, 'l2))
  "r4" -- 'Edge('l1, 'l2) & 'Edge('l2, 'l1) |-> fail()
}
```

8 Summary and discussion

In this section we summarize and discuss our experience specifying the various properties in the two different logics. We reflect on the two logics from a linguistic perspective and make suggestions for improvements to each logic. We also discuss some issues relating to the pragmatic differences between the two methods.

8.1 Relationship to temporal logic

Two approaches to specification of property monitors have been presented, QEA which is automaton-based, and LOGFIRE, which is rule-based. From the point of view of writability and readability it is clear that both logics are low-level in the sense that specifications are somewhat verbose and can be hard to read. The standard alternative approach is some form of temporal logic or regular expressions. In many cases it is likely that these more abstract logics can make specifications easier to write and read. However, note that some properties will not benefit from the abstractions of these higher level logics. For example, the resource lifecycle property in Section 6.3 is suited to a low-level specification style and will likely have a highly convoluted specification in temporal logic.

Note that it is common folklore that temporal logic can be difficult for users to write and read. However, we do believe that many properties can more easily be stated in a combination of temporal logic and regular expressions, as for example found in the SALT language [16], itself influenced by PSL [60]. Occasionally when understanding a property we would draw a time line and plot in events on the time line, not far from the time line notation proposed in TIMEEDIT [56].

The classical way of giving semantics to temporal logic within the model checking community is to translate temporal formulas to automata [33, 44]. Similarly, it has been shown how to translate temporal formulas (LTL) to rules [6]. Temporal logics for parametric monitoring are, however, typically not translated into automata (or rules), but are rather interpreted over the structure of the formulas, which evolve as events are consumed. One reason for this discrepancy in approach within the model checking and runtime verification communities is in part due to lack of automata concepts that involve data. Note that extended state machines (state machines with variables that can be checked in transition guards and updated in transition actions) are not sufficient since variables are global, in contrast to QEA where they are local to a slice. Both QEA and LOGFIRE can be considered as target for translations from parametric temporal logic.

LTL (Linear Temporal Logic) is a more realistic candidate for such translations than CTL (Computation Tree Logic) since a single trace is linear. However, one can imagine monitoring CTL on sets of traces. A transformation of LTL into either logic (QEA or LOGFIRE) would be straightforward, assuming a version of LTL with a finite-trace semantics, as RV is an activity carried out on finite traces only.

In the case of LOGFIRE, since it really is a SCALA API, such translation can be defined as templates in SCALA, as described in [40]. This allows a mix of rule-based programing and temporal logic (in addition to traditional programming). A similar mix of state machines and temporal logic can be found in TRACECONTRACT [9] and DAUT (Data automata) [38, 39]. Similarly, temporal logic can be translated into QEA. We believe that a combination of low-level automata/rules and high-level temporal logic/regular expressions would be a convenient specification formalism. It is interesting to observe that if one allows states/facts to be anonymous (un-named) one obtains systems much related to temporal logic.

This corresponds to having transitions labelled with sequences of events, in contrast to just single events. This can be viewed as a basic abstraction mechanism.

8.2 A few notes on specification styles

Several QEA automata have been stated in *positive form*: describing only valid transitions. This is in contrast to a *negative form*, in which erroneous transitions are called out explicitly, in QEA by leading to a **failure** state and in LogFire by leading to an **error** state. As an example, consider the introductory file usage example. The QEA specification in Figure 2 expresses that a read or write operation is not allowed on a file unless it has been opened, by simply not containing such a transitions out of the `closed` state. In contrast, the LogFire specification in Figure 3 expresses this explicitly as two failure transitions (rules `r5` and `r7`). All properties in LogFire are stated in negative form since positive form formulations are not possible. In QEA one has a choice. It can be debated which of the two forms (positive or negative) that in general is more readable.

Another difference is that QEA supports negation of automata, i.e. the QEA specifies erroneous behavior as success with the understanding that this verdict should be negated. This can be hard to read as one must translate success into failure. The negation is needed for non-deterministic QEA where no paths should lead to failure, as the acceptance condition for the automaton is defined as: *there exists a path to an accepting state*. In LogFire *all paths must lead to an acceptance state*.

Neither QEA nor LogFire support the specification of time as a built-in concept. In both cases time is modeled as time stamps, which are just data like any other data. This leads to a difficulty in determining when time bounds get exhausted since it requires an event with a new time stamp. Time violations are not necessarily detected as soon as they happen but rather when the next event arrives. This is demonstrated in the transaction limit reporting property in Section 5.5.

8.3 Expressiveness and complexity

The expressiveness and complexity of formalisms presented here have not been formally studied at the time of writing. However, it seems plausible that both formalisms are Turing complete, hence equally expressive, and able to express any form of verifiable properties. We here consider LogFire without including all of Scala for writing actions. Of course, if we allow any Scala code to be executed the answer to this question is obvious.

The complexity of monitoring in general depends on the property being monitored and the trace. In the worst case, the monitor may end up storing the entire trace, and in each step search this. For this reason it is important that such search is optimized. In general, however, a monitor stores an abstraction of the so-far observed trace, as a function of the property monitored. This makes monitoring pragmatically possible.

In trace-slicing based techniques such as QEA the number of bindings used in trace-slicing is, in the worst case, exponential in the length of the trace. In practice, values are reused and (for online monitoring) garbage collection (see Sec. 8.5) reduces the set of bindings during monitoring. However, it is often possible to define a specification with fewer quantified variables at the cost of introducing additional guards and assignments. This can dramatically improve the monitoring performance.

8.4 Comments on logics

QEA. One of QEA's main features is that it allows arbitrary interleaving of quantifiers, universal as well as existential. For example, a universal quantification can be nested underneath an existential quantification, which was specifically useful in the specification of the rover coordination property in Section 6.1. Another useful feature of QEA is its support for variables that are local to a slice. LogFire also allows for declaration of variables (since LogFire is a Scala API). However, these are global to the monitor. In LogFire variables local to what corresponds to a slice are modeled as parameters to facts.

One significant drawback of the slicing-based approach of QEA is that a guard on a transition only can test on variables within one slice, and not across slices. The consequence of this restriction is not clear. In LogFire all facts are visible to all rules referring to them. A related disadvantage is that QEA must declare a finite quantifier list, making it difficult to specify either second-order properties (i.e., for all subsets) or properties over a variable number of parameters (i.e., for n rovers for variable n). Both kinds of properties can be handled by the more flexible rule-based approach. Another minor drawback is that the automaton approach in QEA considers states to be distinct. This means that if some variables need updating in several states, such update transitions are needed in all those states, hence causing a repetition of specification. This issue does not occur in LogFire.

During the specification exercise it was recognized that QEA specifications could be simplified by the introduction of pre-defined **success** and **failure** states. Other possible modifications could include the following. QEA specifications contain many state modifiers. A choice could be to introduce defaults, such that for example states by default were accept states. Similarly, one could choose a default amongst the skip/next state modifiers. In TraceContract [9] for example, by default all states are accept and skip states. As we have learned from programming, an if-then-else construct would be useful in order to avoid repeating conditions on transitions.

LogFire. LogFire allows a rule's left-hand side conditions to refer to numerous facts and negations thereof. This yields an expressive power, which in QEA can be partially emulated by introducing sets. Furthermore, LogFire supports transitive closure on facts. The Rete engine will execute rules on a set of facts until a fixed point is reached. This feature is not available in classical automata.

It can be useful for expressing for example reachability properties. This is needed for example if generalizing the lock order property in Section 7.2 to N threads, where N is unknown before monitoring.

LogFire's disadvantages include the following. Due to the fact that it is an API in Scala, user-defined names (event names and fact names) are quoted symbols. This is somewhat inconvenient, and is a consequence of names not being first-class citizens in Scala, as they are not in most programming languages. Finally, facts have to be removed explicitly, in contrast to state machines when taking a transition out of a state.

During the specification exercise it was recognized that LogFire specifications could be simplified by the introduction of **hot** facts (non-accept facts). Other possible modifications could include the following. A fact could be declared as *transient*, meaning that if it occurs in a rule that fires it is removed. LogFire currently does not allow conditions that are negations of conjunctions of facts, as is allowed in the original Rete algorithm. This would be a useful addition. Finally, also in LogFire would an if-then-else construct be useful. The Ruler system [11, 12, 1] has hot and transient facts, as well as an if-then-else construct.

8.5 Under the hood

With respect to implementations, both the slicing algorithm in qea and the augmented Rete algorithm in LogFire use indexing to access states/facts relevant for an incoming event. Future research will expose the exact relationship between the two approaches. Furthermore, there are three pragmatic issues related to monitoring that we have not discussed in depth. They are instrumentation, object identity, and garbage collection. We will briefly explain their relevance. Instrumentation techniques must be used to extract events from running programs/systems; the extracted events might be passed directly to an online monitor or recorded in log files for later processing. To deal with data values, they must have a notion of object identity, i.e., an object such as an Iterator should be consistently recorded using the same identifier. In languages such as Java it is possible to use either an object's *reference identity* (i.e., ==) or *semantic identity* (i.e., `equals`). Usually a property is written with one in mind and getting the correct verdict will depend on using the intended notion of identity. The reference identity of an object is consistent over time, whilst the semantic identify can change, i.e. the semantic identity of a collection may change as its contents changes, making semantic identity inappropriate for monitoring in some cases. Using reference identity requires storing the reference in the monitor (as normally also does semantic identity, but one can get around it, see below). Storing the references can be problematic in garbage-collected languages such as Java, where storing (in the monitor) a reference to an object can prevent the object from being garbage collected when the monitored application no longer refers to it (introducing a memory leak). A monitor can, however, appropriately clean its own data structures to prevent this, which can speed up monitoring. One way to do this in Java is to use reference identity in combination with *weak*

references. Alternatively, one can represent the monitored object by some new object with the same semantic identity, and use semantic identity on this new object thereafter.

9 Conclusion

We have presented two monitoring logics, QEA, which is automaton-based, and LOGFIRE, which is rule-based. These logics can be used for writing monitors directly, or they can be the target of translations from temporal logics. The logics are comparable. However, the distinguishing features of QEA are that it allows existential as well as universal quantification, arbitrarily mixed, as well as variables that are local to slices. The distinguishing features of LOGFIRE are its rule system, where rule conditions can refer to multiple facts and their negations, and where a repeated fixed point evaluation strategy allows for rules to compute the transitive closure, useful for expressing certain reachability properties. QEA is an external DSL whereas LOGFIRE is an internal DSL (and API in SCALA). We showed the application of the two logics to the specification of properties obtained from the 1st international runtime verification competition. Future work includes fully understanding how the two monitoring algorithms relate to each other; improvements on the notations; as well as merging these respective logics with temporal logic.

Acknowledgements We would like to thank the organizers, Ezio Bartocci, Borzoo Bonakdarpour, and Yliès Falcone, of the 1st International Competition of Software for Runtime Verification (CSRV 2014), affiliated with RV 2014 in Toronto, Canada [2]. In addition we would like to thank the participants contributing the properties to the competition.

References

1. RuleR website. http://www.cs.man.ac.uk/∼howard/LPA.html.
2. CSRV 2014. http://rv2014.imag.fr/monitoring-competition.
3. C. Allan, P. Avgustinov, A. S. Christensen, L. Hendren, S. Kuzins, O. Lhoták, O. de Moor, D. Sereni, G. Sittamplan, and J. Tibble. Adding trace matching with free variables to AspectJ. In *OOPSLA'05*. ACM Press, 2005.
4. H. Barringer, Y. Falcone, K. Havelund, G. Reger, and D. Rydeheard. Quantified Event Automata - towards expressive and efficient runtime monitors. In *18th International Symposium on Formal Methods (FM'12), Paris, France, August 27-31, 2012. Proceedings*, volume 7436 of *LNCS*. Springer, 2012.
5. H. Barringer, M. Fisher, D. M. Gabbay, G. Gough, and R. Owens. Metatem: An introduction. *Formal Asp. Comput.*, 7(5):533–549, 1995.
6. H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Program monitoring with LTL in Eagle. In *Parallel and Distributed Systems: Testing and Debugging (PADTAD'04), Santa Fee, New Mexico, USA*, volume 17 of *IEEE Computer Society*, April 2004.

7. H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In *VMCAI*, volume 2937 of *LNCS*, pages 44–57. Springer, 2004.
8. H. Barringer, A. Groce, K. Havelund, and M. Smith. Formal analysis of log files. *Journal of Aerospace Computing, Information, and Communication*, 7(11):365–390, 2010.
9. H. Barringer and K. Havelund. TraceContract: A Scala DSL for trace analysis. In *17th International Symposium on Formal Methods (FM'11), Limerick, Ireland, June 20-24, 2011. Proceedings*, volume 6664 of *LNCS*, pages 57–72. Springer, 2011.
10. H. Barringer, K. Havelund, D. Rydeheard, and A. Groce. Rule systems for runtime verification: A short tutorial. In *Proc. of the 9th Int. Workshop on Runtime Verification (RV'09)*, volume 5779 of *LNCS*, pages 1–24. Springer, 2009.
11. H. Barringer, D. Rydeheard, and K. Havelund. Rule systems for run-time monitoring: from Eagle to RuleR. In *Proc. of the 7th Int. Workshop on Runtime Verification (RV'07)*, volume 4839 of *LNCS*, pages 111–125, Vancouver, Canada, 2007. Springer.
12. H. Barringer, D. E. Rydeheard, and K. Havelund. Rule systems for run-time monitoring: from Eagle to RuleR. *J. Log. Comput.*, 20(3):675–706, 2010.
13. D. A. Basin, F. Klaedtke, and S. Müller. Policy monitoring in first-order temporal logic. In T. Touili, B. Cook, and P. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, Proceedings*, volume 6174 of *LNCS*, pages 1–18. Springer, 2010.
14. A. Bauer, J.-C. Küster, and G. Vegliach. From propositional to first-order monitoring. In *Runtime Verification - 4th Int. Conference, RV'13, Rennes, France, September 24-27, 2013. Proceedings*, volume 8174 of *LNCS*, pages 59–75. Springer, 2013.
15. A. Bauer, M. Leucker, and C. Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In *Proc. of the 7th Int. Workshop on Runtime Verification (RV'07)*, volume 4839 of *LNCS*, pages 126–138, Vancouver, Canada, 2007. Springer.
16. A. Bauer, M. Leucker, and J. Streit. SALT – structured assertion language for temporal logic. In Z. Liu and J. He, editors, *Formal Methods and Software Engineering*, volume 4260 of *Lecture Notes in Computer Science*, pages 757–775. Springer Berlin Heidelberg, 2006.
17. E. Bodden. MOPBox: A library approach to runtime verification. In *Runtime Verification - 2nd Int. Conference, RV'11, San Francisco, USA, September 27-30, 2011. Proceedings*, volume 7186 of *LNCS*, pages 365–369. Springer, 2011.
18. F. Chen and G. Roşu. Parametric trace slicing and monitoring. In *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09)*, volume 5505 of *LNCS*, pages 246–261, 2009.
19. Clips website: http://clipsrules.sourceforge.net.
20. H. C. Cruz. Optimisation techniques for runtime verification. Master's thesis, University of Manchester, 2014.
21. M. D'Amorim and K. Havelund. Event-based runtime verification of Java programs. In *Workshop on Dynamic Program Analysis (WODA'05)*, volume 30(4) of *ACM Sigsoft Software Engineering Notes*, pages 1–7, 2005.
22. N. Decker, M. Leucker, and D. Thoma. Monitoring modulo theories. In E. Ábrahám and K. Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Grenoble, France, April 7-11, 2014. Proceedings*, volume 8413 of *LNCS*, pages 341–356. Springer, 2014.
23. R. B. Doorenbos. *Production Matching for Large Learning Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1995.

24. Drools website: http://www.jboss.org/drools.
25. Drools functional programming extensions website: https://community.jboss.org/wiki/FunctionalProgrammingInDrools.
26. D. Drusinsky. The temporal rover and the ATG rover. In *SPIN Model Checking and Software Verification*, volume 1885 of *LNCS*, pages 323–330. Springer, 2000.
27. D. Drusinsky. *Modeling and Verification using UML Statecharts*. Elsevier, 2006. ISBN-13: 978-0-7506-7949-7, 400 pages.
28. Y. Falcone, J.-C. Fernandez, and L. Mounier. Runtime verification of safety-progress properties. In *Proc. of the 9th Int. Workshop on Runtime Verification (RV'09)*, volume 5779 of *LNCS*, pages 40–59. Springer, 2009.
29. Y. Falcone, J.-C. Fernandez, and L. Mounier. What can you verify and enforce at runtime? *J Software Tools for Technology Transfer*, 14(3):349–382, 2012.
30. Y. Falcone, K. Havelund, and G. Reger. A tutorial on runtime verification. In M. Broy and D. Peled, editors, *Summer School Marktoberdorf 2012 - Engineering Dependable Software Systems, to appear*. IOS Press, 2013.
31. C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.
32. M. Fusco. Hammurabi - a Scala rule engine. In *Scala Days 2011, Stanford University, California*, 2011.
33. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In P. Dembinski and M. Sredniawa, editors, *In Protocol Specification Testing and Verification (PSTV)*, volume 38, pages 3–18. Chapman & Hall, 1995.
34. J. Goubault-Larrecq and J. Olivain. A smell of ORCHIDS. In *Proc. of the 8th Int. Workshop on Runtime Verification (RV'08)*, volume 5289 of *LNCS*, pages 1–20, Budapest, Hungary, 2008. Springer.
35. A. Groce, K. Havelund, and M. H. Smith. From scripts to specifications: the evolution of a flight software testing effort. In *32nd Int. Conference on Software Engineering (ICSE'10), Cape Town, South Africa*, ACM SIG, pages 129–138, 2010.
36. S. Hallé and R. Villemaire. Runtime enforcement of web service message contracts with data. *IEEE Transactions on Services Computing*, 5(2):192–206, 2012.
37. K. Havelund. Runtime verification of C programs. In *Proc. of the 1st TestCom/-FATES conference*, volume 5047 of *LNCS*, Tokyo, Japan, 2008. Springer.
38. K. Havelund. Data automata in Scala. In M. Leucker and J. Wang, editors, *8th International Symposium on Theoretical Aspects of Software Engineering, TASE 2014, Changsha, China, September 1-3. Proceedings*. IEEE Computer Society Press, 2014.
39. K. Havelund. Monitoring with data automata. In T. Margaria and B. Steffen, editors, *6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. Track: Statistical Model Checking, Past Present and Future (organized by Kim Larsen and Axel Legay), Corfu, Greece, October 8-11. Proceedings*, volume 8803 of *LNCS*, pages 254–273. Springer, 2014.
40. K. Havelund. Rule-based runtime verification revisited. *Software Tools for Technology Transfer (STTT)*, April 2014. Published online.
41. K. Havelund and A. Goldberg. Verify your runs. In *Verified Software: Theories, Tools, Experiments, VSTTE 2005*, pages 374–383, 2008.
42. K. Havelund and G. Roşu. Efficient monitoring of safety properties. *Software Tools for Technology Transfer*, 6(2):158–173, 2004.
43. K. Havelund and G. Rosu. Monitoring programs using rewriting. In *16th ASE conference, San Diego, CA, USA*, pages 135–143, 2001.

44. G. J. Holzmann. *The Spin Model Checker – Primer and Reference Manual*. Addison-Wesley, 2004.
45. Jess website: http://www.jessrules.com/jess.
46. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In J. L. Knudsen, editor, *Proc. of the 15th European Conference on Object-Oriented Programming*, volume 2072 of *LNCS*, pages 327–353. Springer, 2001.
47. C. Lee, D. Jin, P. O. Meredith, and G. Roşu. Towards categorizing and formalizing the JDK API. Technical Report http://hdl.handle.net/2142/30006, Department of Computer Science, University of Illinois at Urbana-Champaign, March 2012.
48. I. Lee, S. Kannan, M. Kim, O. Sokolsky, and M. Viswanathan. Runtime assurance based on formal specifications. In *PDPTA*, pages 279–287. CSREA Press, 1999.
49. M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, may/june 2008.
50. D. Luckham, editor. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2002.
51. P. Meredith, D. Jin, D. Griffith, F. Chen, and G. Roşu. An overview of the MOP runtime verification framework. *Software Tools for Technology Transfer (STTT)*, 14(3):249–289, 2012.
52. A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society, 1977.
53. G. Reger. *Automata Based Monitoring and Mining of Execution Traces*. PhD thesis, University of Manchester, 2014.
54. G. Reger, H. C. Cruz, and D. Rydeheard. MARQ: monitoring at runtime with QEA. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*, 2015.
55. Rooscaloo website: http://code.google.com/p/rooscaloo.
56. M. Smith, G. Holzmann, and K. Ettessami. Events and constraints: a graphical editor for capturing logic properties of programs. In *5th Int Sym. on Requirements Engineering, Toronto, Canada*, volume 55(2), pages 14–22. 2001.
57. V. Stolz. Temporal assertions with parameterized propositions. In *Proc. of the 7th Int. Workshop on Runtime Verification (RV'07)*, volume 4839 of *LNCS*, pages 176–187, Vancouver, Canada, 2007. Springer.
58. V. Stolz and E. Bodden. Temporal assertions using AspectJ. In *Proc. of the 5th Int. Workshop on Runtime Verification (RV'05)*, volume 144(4) of *ENTCS*, pages 109–124. Elsevier, 2006.
59. V. Stolz and F. Huch. Runtime verification of concurrent Haskell programs. In *Proc. of the 4th Int. Workshop on Runtime Verification (RV'04)*, volume 113 of *ENTCS*, pages 201–216. Elsevier, 2005.
60. M. Vardi. From Church and Prior to PSL. In O. Grumberg and H. Veith, editors, *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 150–171. Springer Berlin Heidelberg, 2008.

Advances in Design Automation Techniques for Digital-Microfluidic Biochips

Mohamed Ibrahim, Zipeng Li and Krishnendu Chakrabarty

Duke University, Durham, NC 27708, USA

Abstract. Due to their emergence as an efficient platform for point-of-care clinical diagnostics, digital-microfluidic biochips (DMFBs) have received considerable attention in recent years. They combine electronics with biology, and they integrate various bioassay operations, such as sample preparation, analysis, separation, and detection. In this chapter, we first present an overview of digital-microfluidic biochips. We next describe emerging computer-aided design (CAD) tools for the automated synthesis and optimization of biochips from bioassay protocols. The chapter includes solutions for fluidic-operation scheduling, module placement, droplet routing, and pin-constrained chip design. We also show how recent advances in the integration of sensors into a DMFB can be exploited to provide cyberphysical system adaptation based on feedback-driven control.

Keywords: digital-microfluidic biochips, computer-aided design (CAD), synthesis, cyberphysical design, testing, fault diagnosis.

1 Introduction

Microfluidic biochips, also referred to as lab-on-a-chip, are revolutionizing many areas of biochemistry and biomedical sciences, such as high-throughput DNA sequencing, point-of-care clinical diagnosis, protein crystallization, enzymatic analysis, proteomic analysis, and environmental toxicity monitoring [1–3]. Most traditional microfluidic biochips are based on a flow-based platform with microchannels, microvalves, micropumps, and addressable chambers. Using these microdevices, various biochemical applications, including polymerase chain reaction (PCR), DNA purification, and protein crystallization, can be performed in an automated manner. A drawback of flow-based microfluidics is the lack of reconfigurability. Therefore, flow-based microfluidic biochips are only suitable for a narrow class of bioassays, and each chip is specific to a target application.

An alternative category of microfluidic biochips, referred to as digital microfluidic biochips (DMFBs) and based on the principle of electrowetting-on-dielectric (EWOD) [3,4], is an attractive platform to overcome above limitations. Fabricated DMFBs based on glass and PCB substrates are shown in. Fig. 1.

In this section, we provide an overview of DMFBs and the EWOD principle, analyze basic fluidic operations of the DMFB, and provide motivation for design automation (e.g., high-level synthesis, chip-level design, and fault modeling and testing, etc.) for DMFBs.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_7,

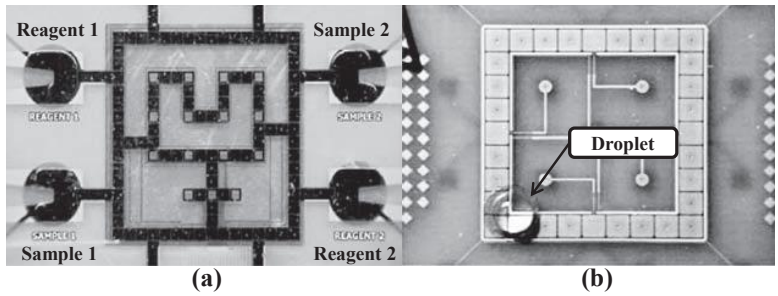


Fig. 1. Fabricated digital microfluidic biochips: (a) glass substrate [5]; (b) PCB substrate [6].

1.1 Overview of Digital Microfluidics

A DMFB is an example of a lab-on-a-chip that is capable of carrying out biochemistry on a chip. A typical DMFB consists of a bottom layer and a top layer [7]. For cyberphysical DMFBs, a photodetector (PD) for optical sensing can also be integrated into the top plate of the DMFB [8]; see Fig. 2. An optical fiber, pigtailed to the input optical waveguide, is used to launch the polarized laser beam into the integrated optical system to realize sensing. A dielectric layer is deposited on both surfaces of bottom and top plate to further increase the hydrophobicity and insulativity. Because manipulated droplets with biological samples are usually of micro-liter or nano-liter volume, in order to avoid the evaporation of reagents, a filler medium (usually silicone oil) is used between two plates. Manipulated using EWOD, droplets of samples and reagents can be transported along the electrodes, and fluidic operations such as dispensing, transporting, mixing, dilution and splitting can be carried out on the chip.

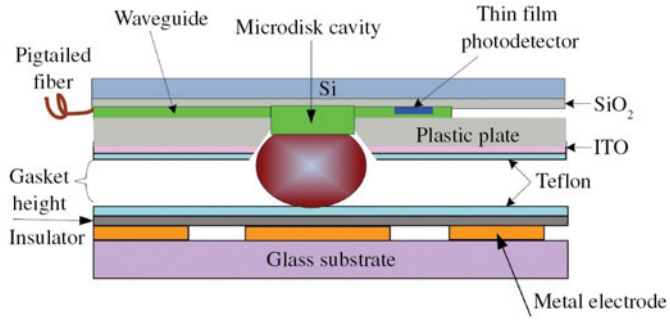


Fig. 2. A sideview of a two-layer digital microfluidic biochip with an integrated photodetector (PD) [8].

Compared with conventional benchtop procedures, a DMFB offers advantages of lower cost, portability, less power consumption, easier system integration, and less human error. Hence, DMFBs are revolutionizing many biochemical analysis procedures, including high-throughput DNA sequencing, point-of-care clinical diagnosis, and protein crystallization [7, 9]. In recent years, with the integration of different types of sensors (such as capacitive sensors [10] and optical sensors [8] in a cyberphysical platform), real-time feedback and software-based control can be realized and DMFBs have become even more attractive.

1.2 Principle of Electrowetting-on-Dielectric (EWOD)

EWOD is a physical phenomenon where an electric field can influence the wetting behavior of a conductive droplet in contact with a hydrophobic and insulated electrode. When an electric voltage is applied between the liquid droplet and the electrode, the electric charge changes the free energy on the dielectric surface. Therefore, an electric field across the insulator is generated, which lowers the interfacial tension between the liquid and the insulator surface (as illustrated in Fig. 3) according to the Lippman-Young equation, see (1):

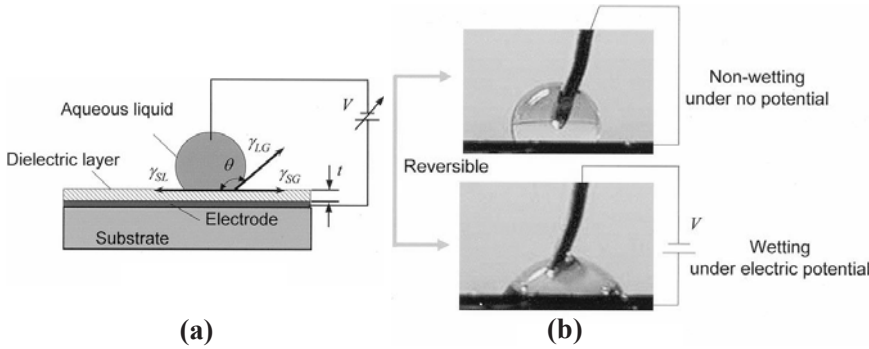


Fig. 3. Principle of electrowetting on dielectric (EWOD). (a) Schematic configuration. (b) EWOD demonstration [11].

$$\gamma_{SL}(V) = \gamma_{SL}(V = 0) - \frac{C}{2}V^2 \tag{1}$$

where γ_{SL} is the solid-liquid interfacial tension, $C(\mathrm{F}/m^2)$ is the capacitance of the dielectric layer, and V is the applied voltage. At the three-phase contact line, Young's equation describes the relationship between the contact angle and interfacial tensions:

$$\cos\theta = \frac{\gamma_{SG} - \gamma_{SL}}{\gamma_{LG}} \tag{2}$$

where γ_{SG} is the solid-gas interfacial tension, and γ_{LG} is the liquid-gas interfacial tension. Combining (1) and (2), the change in the contact angle can be described by (3):

$$\cos\theta = \cos\theta_0 - \frac{\varepsilon_0\varepsilon}{2\gamma_{LG}t}V^2 \tag{3}$$

where θ_0 is the equilibrium contact angle with no applied potential, ε_0 is the permittivity of vacuum, ε is the dielectric constant of the dielectric layer, and t is the thickness of the dielectric layer.

1.3 Fluidic Operations on the DMFB

Fluidic operations, e.g., transportation, dispensing and storing, mixing, and splitting and merging, can be achieved on a DMFB based on the principle of EWOD.

Transportation. A liquid droplet on the DMFB can be transported by asymmetrically changing the interfacial tension, i.e., by applying electric potential. Details of droplet transportation can be found in [12]. Droplet velocities of 100 mm/s at 60 V have been reported for droplets ranging in volume from 1 μL to 1 nL [3].

Dispensing and Storing. Dispensing (droplet generation) is a critical fluidic operation for a DMFB, because it can be viewed as fluidic I/O and the world-to-chip interface. Generally, droplet dispensing refers to the process of generating small-volume liquid droplets on the unit electrode for manipulation on the DMFB. Conversely, droplets can also be stored in the on-chip reservoir.

In order to dispense droplets from an on-chip reservoir, liquid needs to be first pulled out of the reservoir and then be separated from the reservoir: these steps can be seen in Fig. 4.

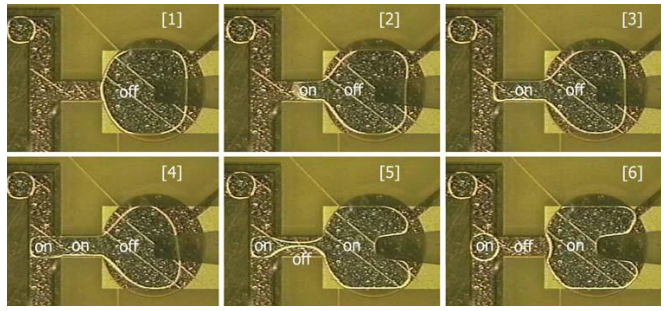


Fig. 4. Droplet dispensing from an on-chip reservoir [5].

Splitting and Merging. The splitting and merging operations use three consecutive electrodes. In splitting, the outer two electrodes are turned on, and the inner electrode is turned off. In general, the hydrophilic forces induced by the two outer electrodes stretch the droplet while the hydrophobic forces in the center pinch off the liquid into two small droplets [13]. In contrast, for the merging operation, the inner electrode is turned on, and the outer two electrodes are turned off. The steps involved in droplet splitting and merging can be seen in Fig. 5.

Mixing. The mixing operation on the DMFB can either be used for preprocessing, sample dilution, or for reactions between samples and reagents of predefined volume. If we define the consecutive on and off of one electrode as one cycle, mixing operations usually take around 1,000 cycles [3]. Creating turbulent flow at mixing regions or creating multilaminates are two effective ways to speed up the mixing operation [3].

1.4 Motivation for Automated Chip Design and Testing

As an emerging technology, the market for biochips is likely to grow rapidly, motivated by applications from the pharmaceutical and healthcare fields. Because of the trend of increasing chip size and the integration of an increasing number of on-chip devices, manual design for biochips is no longer feasible. In particular, design-automation tools can reduce the difficulty of design and help to ensure that manufactured biochips are versatile and reliable.

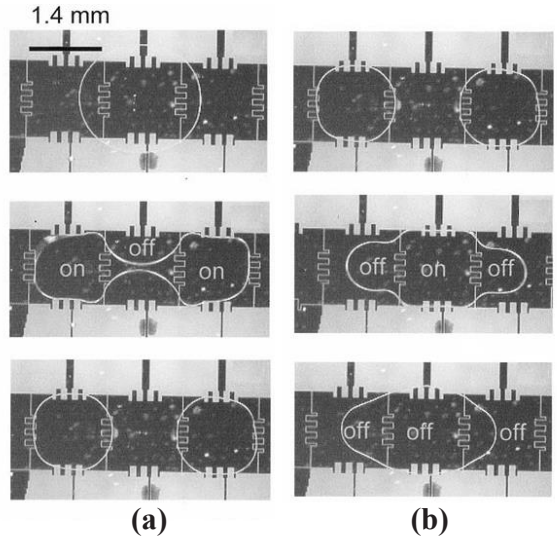


Fig. 5. Sequential images of successful (a) splitting and (b) merging of droplets at 25 V (gap size d = 70 μm, electrode is 1.4 mm×1.4 mm, volume is 0.2 μl) [11].

Therefore, there is a need to provide the same level of CAD support for the biochip designer as in the semiconductor industry. With the help of these CAD tools, biochip users, including chemists, nurses, doctors and clinicians, will adapt more easily to this new technology, and conversely, designers can be freed from cumbersome and labor-intensive work, and they can concentrate more on improving chip reliability and reducing chip cost.

An illustration of an automated DMFB design flow is shown in Fig. 6. Design automation for a DMFB includes synthesis (both architectural-level synthesis and physical-level synthesis), chip-level design (control pin assignment and wire routing), and fault modeling and testing (for structural testing and functional testing). These design steps will be covered in detail throughout this chapter.

Researchers now envision an automated design flow and system operation that will transform biochip use, in the same way as compilers and operating systems revolutionized computing in the 60s and 70s, and design automation revolutionized IC design in the 80s and 90s.

1.5 Motivation for Cyberphysical Chip Design

Biochemical experiments are inherently prone to randomness and lack of precision in their operations. Operational errors can potentially happen during droplet manipulation, but error recovery based on the repetition of experiments leads to wastage of expensive reagents and hard-to-prepare samples. One solution for this problem is to use a statistical predictive model for error analysis [15], but this approach is computationally expensive and does not leverage the reconfiguration capability of digital-microfluidic biochips.

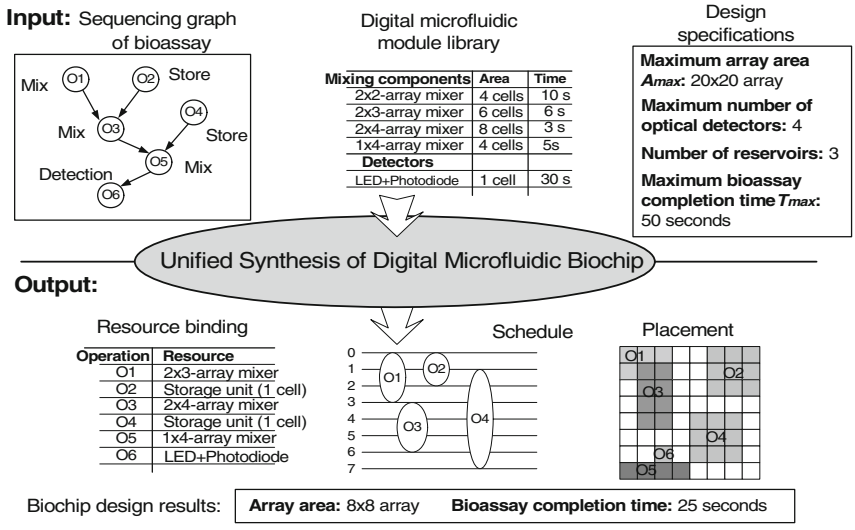


Fig. 6. Illustration of automated DMFB design [14].

With recent advances in the integration of biosensors in digital-microfluidic biochips [47, 48], it is now feasible to develop "physical-aware" control systems that can use sensor readouts at checkpoints and dynamically reconfigure the biochip to recover from an error. This form of cyberphysical integration provides higher guarantees for more robust bioassay execution within an autonomous microfluidic platform. Cyberphysical design of digital-microfluidic biochips will be covered in Section 5.

2 Architectural-Level Synthesis

The automated design and high-level synthesis of DMFBs have attracted considerable interest in recent years. DMFB synthesis consists of two parts: architectural level synthesis and physical-level synthesis. In this section, we focus on architectural-level synthesis.

2.1 Design Objectives

For architectural-level synthesis, the major goal is to schedule corresponding operations and to bind each operation to a limited set of resources (e.g., dilution and mixing). With appropriate architectural-level synthesis, maximum operation parallelism and minimum execution time can be achieved. Minimizing execution time is critical for bioassays due to the following reasons. First, biological samples and chemical reagents are sensitive to the environment, therefore long execution time may lead to degradation. Second, long execution time cannot meet the requirements of rapid time-to-result and point-of-care applications. Finally, long execution time means more electrode actuation, which may result in the degradation of DMFBs. In the design flow of DMFBs, a bioassay can be modeled as a directed, acyclic and polar sequencing graph. Each node represents a fundamental operation (e.g., mixing, dilution, dispensing and detection), and a direct edge represents the dependency relationship between two operations. Once resource binding is accomplished, the completion time for each operation can also be determined, and the corresponding sequencing graph can be constructed.

For some critical bioassays, such as clinical diagnostics, it is important to verify the correctness of on-chip fluidic operation results. Therefore, error recovery can also be embedded in the architectural-level synthesis. We will cover this topic in detail in Section 5. Here, we focus on conventional synthesis methods that do not consider error recovery.

2.2 Synthesis Algorithms

Several algorithms, such as integer linear programming (ILP)-based synthesis [16], Tabu-search-based synthesis [24] and priority scheduling algorithm [17] have been proposed to handle architecture-level synthesis without error recovery.

Su et al. proposed the first system-level method that attempted to apply classical architectural-level synthesis techniques to the design of DMFBs. In their work, architectural-level synthesis is viewed as the problem of scheduling assays and resource binding for maximum parallelism. An ILP model is formulated to obtain optimal scheduling results under resource constraints.

A clinical diagnosis procedure, namely *in-vitro* diagnosis, is used in their work. The diagnosis protocol was mapped to a sequencing graph model (see Fig. 7). The graph $G(V, E)$ has the vertex set V in correspondence with the set of operations, and the edge set E representing dependencies. Each node v_i is assigned a weight $d(v_i))$ denoting the time taken for operation v_i. It was assumed that there are m types of physiological fluids for n types of enzymatic measurements.

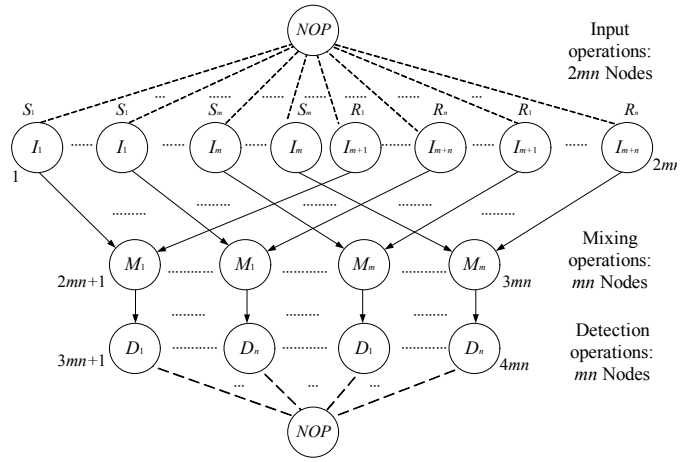


Fig. 7. Sequencing graph model of a multiplexed biomedical assay [16].

The ILP-based model is formulated to minimize the assay completion time under resource constraints. The objective function is to minimize completion time $C = \{St_i + d(v_i)\}$, where St_i is the starting time of operation v_i and $d(v_i)$ represents the duration time for v_i.

Constraints consist of dependency constraints and resource constraints. Dependency constraints are of the following type: if operation v_j depends on v_i, then constraint $St_j \geq St_i + d(v_i)$ should be satisfied. There are three resource constraints: reservoir port constraint, reconfigurable storage unit constraint and optical detector constraint. A binary variable X_{ij} is defined first to be 1 if operation v_i starts at time slot j, otherwise X_{ij} is set to be 0. Reservoir port constraint can be expressed to be $\sum_{i:v_i \in I_k} X_{ij} \leq 1$, $1 \leq k \leq m + n$ and $1 \leq j \leq T$. Reconfigurable storage unit constraint is formulated to be $Nmixer(j) + 0.25Nmemory(j) \leq Na$ for $1 \leq j \leq T$, where $Nmixer(j)$ and $Nmemory(j)$

is the number of mixer needed and the number of storage units needed at time slot j. N_d detectors are assigned to each enzymatic assay. Without loss of generality, N_d is set to 1, and the optical detector constraint is expressed as $\sum_{i:v_i \in D_1} \sum_{l=j-d(v_i)}^{j} X_{ij} \leq 1, ..., \sum_{i:v_i \in D_{n1}} \sum_{l=j-d(v_i)}^{j} X_{ij} \leq 1$ for $1 \leq j \leq T$. Experimental results show that optimal solutions can be obtained using this ILP model for small problem instances.

DMFB Synthesis (G, C, L)
1 $< A^o, B^o >$=InitialSolution(G, L)
2 $\prod^o$=CriticalPath(G, A^o, B^o)
3 $< A, B, \prod >$=TabuSearch$(G, C, L, A^o, B^o, \prod^o)$
4 $< S, P >$=ScheduleAndPlace$(G, C, A, B, \prod)$
5 **return** $\prod =< A, B, S, P >$

Fig. 8. Synthesis algorithm for DMFBs.

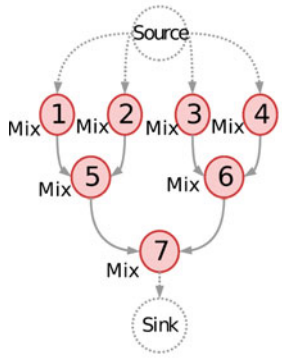
Two other algorithms, namely genetic algorithm (GA) and modified list scheduling (M-LS), were also proposed in [16]. For bigger instances, both GA and M-LS can generate good results. However, GA is more time-consuming when compared with M-LS.

Maftei et al. proposed a Tabu Search-based synthesis approach that can determine the allocation, resource binding, and scheduling of on-chip bioassays. Their work can be formulated as follows. Given a biochemical application model as a graph $G(V, E)$, a biochip in the form of an $m \times n$ array, and a module library L, it is possible to synthesize the implementation $\psi =< A, B, S, P >$ (including the allocation, binding, scheduling and placement), which can minimize the completion time δ_G. This work also considered the movement of virtual devices during their operation to reduce the application completion time.

A Tabu Search (TS) metaheuristic for architecture-level synthesis was used in [24]. The synthesis algorithm for DMFBs is shown in Fig. 8. After binding and allocation have been obtained by TS, List Scheduling (LS) heuristic was used to determine the schedule S of operations. The priority $\prod$ for each operation is also determined by TS.

Note that TS is a metaheuristic based on a neighborhood search technique that uses design transformations applied to the current solution to generate a set of neighboring solutions. A mixing stage of the polymerase chain reaction (see Fig. 9(a)) and module library (see Fig. 9(b)) are used to illustrate how TS works.

Consider a current solution as shown in Fig. 10(a). The current Tabu list shown on the right contains recently performed transformations. Starting from this solution, TS generates neighboring solutions. Three possible solutions are presented in Fig. 10(b)-(d). Based on the completion time, TS will select the movement in Fig. 10(b).

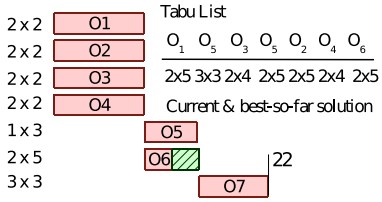


(a) PCR/M graph

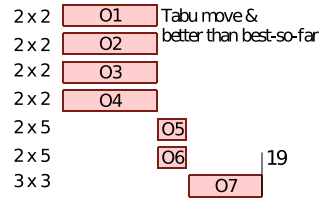
Operation	Area (cells)	Time (s)
Mixing	2×5	2
Mixing	2×4	3
Mixing	1×3	5
Mixing	3×3	7
Mixing	2×2	10
Dilution	2×5	4
Dilution	2×4	5
Dilution	1×3	7
Dilution	3×3	10
Dilution	2×2	12
Detection	1×1	30

(b) Module library

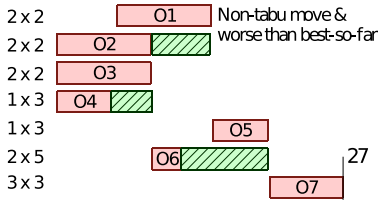
Fig. 9. A PCR sequence graph and corresponding module library for experimental evaluation in [24].



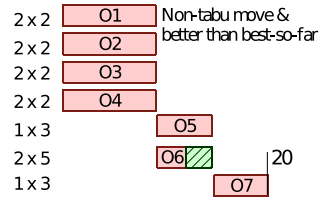
(a) Current solution



(b) Rebind O_5 to a 2×5 module



(c) Rebind O_4 to a 1×3 module



(d) Rebind O_7 to a 1×3 module

Fig. 10. Demonstration of rebind process in an example of Tabu Search neighborhood [24].

Corresponding experimental results for two real-life examples, *in-vitro* diagnostics on human physiological fluids (IVD) and the mixing stage of PCR, demonstrate that the proposed Tabu Search-based approach can obtain optimal results using much less computing time when compared with the ILP method.

Ricketts et al. proposed a hybrid priority scheduling algorithm (HPA) to perform scheduling for a resource-constrained DMFB [17]. For a system with S_m samples and R_n reagents, the genetic encoding in [16] results in a search space of $(4S_mR_n)!$ possible solutions, while HPA results in a search space of only S_mR_n candidates. This improvement is based on the realization that (1) some operations do not directly conflict with each other, and (2) operations (dispensing, mixing and detection) are performed independently, and a large number of encoded chromosomes in the original $4S_mR_n$ priority scale will not yield better solutions.

Therefore, they allocated the same priorities to all operations that are directly interdependent, which removes the possibility of hold and wait due to unfulfilled incoming edge requirements. In this way, while all the $(S_mR_n)!$ combinations are not tried, the experimental results demonstrate this is sufficient to achieve near-optimal results.

3 Physical-Level Synthesis

Unlike architectural-level synthesis, physical-level synthesis addresses the placement of resources and the routing of droplets to satisfy objectives such as area minimization or throughput maximization. It creates the final layout of the biochip, consisting of the placement of microfluidic modules such as mixers and storage units [22–24], the routes that droplets take between different modules [28–30], and other geometrical details [20]. This is the reason why this step is also known as geometry-level synthesis.

In addition, when two or more droplet routes intersect or overlap with each other, it is likely that a droplet that arrives at a later clock cycle can be contaminated by the residue left behind by another droplet that passed through at an earlier clock cycle, resulting in *cross-contamination* [21]. To avoid incorrect assay outcomes, wash operations (*i.e.,* wash-droplet routing) should also be incorporated in physical-level design.

In this section, we provide an overview of the key problems in physical-level synthesis of DMFBs. We describe emerging CAD tools for automating module placement, droplet routing, and cross-contamination avoidance in DMFBs.

3.1 Module Placement

Module placement is one of the key physical design problems for DMFBs. Based on the results obtained from architectural-level synthesis described in Section 2 (i.e., a schedule of bioassay operation, a set of microfluidic modules, and the binding of bioassay operations to modules), placement determines the locations of each module on the microfluidic array [23].

Design Objectives. The most important optimization objective in placement is the minimization of chip area. Since solutions to the placement problem can provide the designers with guidelines on chip footprint, area minimization frees up more unit cells for other fluidic functions such as sample preparation and collection [21]. During placement, some performance constraints, such as an upper limit on assay completion time and maximum allowable chip array, must be satisfied.

Placement Algorithms. DMFBs enable dynamic reconfiguration of the microfluidic array during run-time, thus they allow the placement of different modules on the same location during different time intervals. In [23], Su and Chakrabarty modeled the placement of modules as a 3-D packing problem, as shown in Fig. 11. Each microfluidic module is represented by a 3-D box, the base of which denotes the rectangular area of the module and the height denoting the time-span of its operation. Microfluidic biochip placement can now be viewed as the problem of packing these boxes to minimize the total base area while avoiding overlaps [23]. Since the microfluidic placement problem is shown to be NP-complete [23], a heuristic simulated annealing (SA) algorithm is used to solve it in a computationally efficient manner.

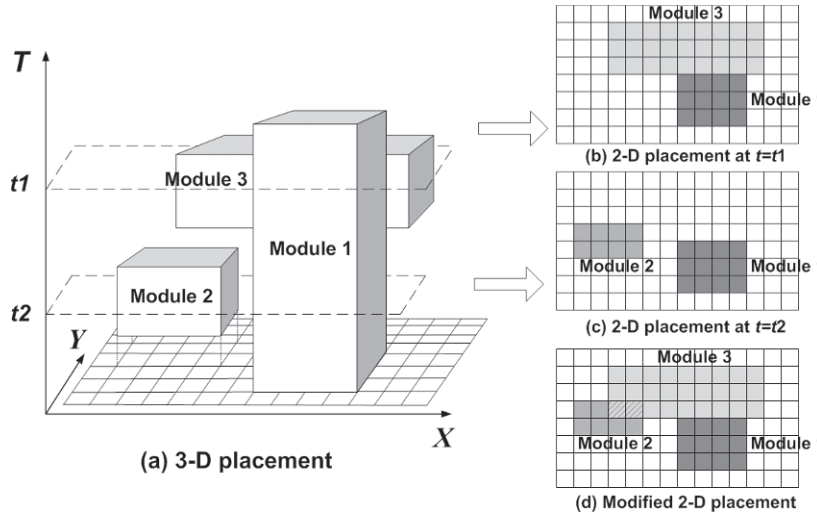


Fig. 11. Reduction from 3-D placement to a modified 2-D placement [23].

Motivated by simultaneous scheduling and placement in dynamically reconfigurable FPGAs (DRFPGAs), Yuh et al. adopt a T-tree topological representation to solve the placement problem for DMFBs [22]. A T-tree is a 3-ary tree used to represent a compact placement that is suitable for volume optimiza-

tion, and thereby is more likely to generate solutions that are within a defined 3D cube. Consequently, given a set of m tasks, let W_i, H_i, and T_i denote the width, height, and duration of each task, respectively, $1 \leq i \leq m$. Based on the three-dimentional space (X,Y,T), the T-tree represents the geometric relationship between two tasks as follows. If node n_j is the left child of node n_i, module v_j, must be placed adjacent to module v_i in the T^+ direction, i.e., $t_j = t_i + T_i$. If node n_k is the middle child of node n_i, module v_k must be placed in the Y^+ direction of module v_i, with the t-coordinate of v_k equal to that of v_i, *i.e.*, $t_k = t_i$ and $y_k \geq y_i + H_i$. If node n_l is the right child of node n_i, module v_l must be placed in the X^+ direction of module v_i, with the t- and y-coordinates equal to those of v_i, *i.e.*, $t_l = t_i$ and $y_l = y_i$. Figure 12 illustrates a compact placement with its corresponding T-tree [22]. Based on this T-tree model, a simulated annealing-based algorithm is also employed to perform placement [22].

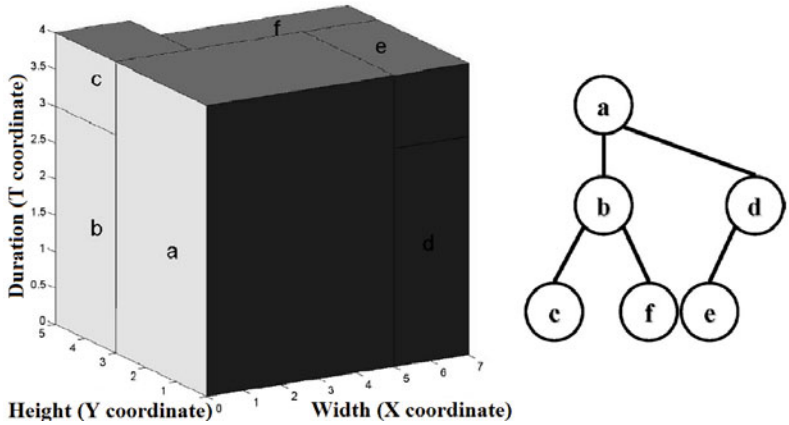


Fig. 12. A compact placement and its corresponding T-tree [22].

Minimizing chip area is not the only objective of placement algorithms. Since DMFBs are fabricated using standard microfabrication techniques [25], the array may contain several defective cells [23]. Some defects that have been reported for fabricated biochips include dielectric breakdown, shorts between adjacent electrodes, electrode degradation, particle contamination and residue, and etch variations. Therefore, reconfiguration techniques have been considered in [22,23] to bypass faulty cells in order to tolerate defects. Finally, as a way of optimizing for the completion time of a biochemical application, Maftei et al. in [24] proposed a Tabu Search metaheuristic algorithm for the synthesis of DMFBs as discussed in Section 2. This includes a placement procedure that takes advantage of dynamic reconfigurability to achieve dynamic placement of modules, even *during* their execution.

3.2 Droplet Routing

A key problem in biochip physical design is droplet routing between modules, and between modules and I/O ports (i.e., on-chip reservoirs) [20]. The dynamic reconfigurability inherent in digital microfluidics allows different droplet routes to share cells on the microfluidic array during different time intervals. In this way, the routes in DMFBs can be viewed as *virtual* routes, which make droplet routing different from the classical VLSI routing problem.

Design Objectives. The objective of any droplet routing method for DMFBs is to minimize the number of cells used for routing, while satisfying constraints imposed by fluidic properties. As resource sharing in a time-multiplexed fashion is allowed in a DMFB, droplet routing can be modeled in three dimensions, where z-axis is for time, which enables us to optimize geometric paths and temporal schedules simultaneously. Cho and Pan created a graph model to visualize the allowable movements of a droplet at (x,y,t) [26], as shown in Fig. 13(a). They also used the 3-D space to visualize the fluidic constraints that need to be satisfied to avoid unwanted mixing between droplets moving in parallel. Let d_i at (x_i^t,y_i^t) and d_j at (x_j^t,y_j^t) denote two independent droplets at time t. Then, the following constraints must be satisfied at any time t during routing:

1. Static constraint: $|x_i^t - x_j^t| > 1$ or $|y_i^t - y_j^t| > 1$.
2. Dynamic constraint: $|x_i^{t+1} - x_j^t| > 1$ or $|y_i^{t+1} - y_j^t| > 1$ or $|x_i^t - x_j^{t+1}| > 1$ or $|y_i^t - y_j^{t+1}| > 1$.

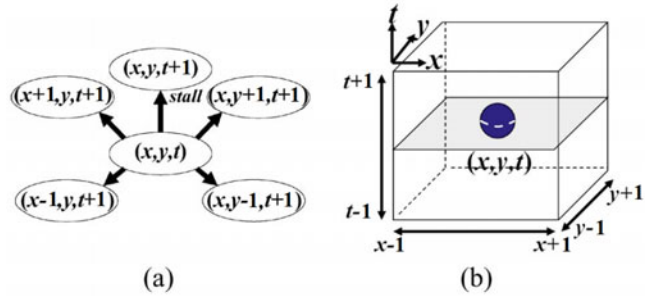


Fig. 13. Graph model and fluidic constraints for DMFB design [26]: (a) a graph for droplet routing models geometric paths as well as temporal schedules simultaneously; (b) dynamic and static fluidic constraints prevent undesirable mixing of droplets during movement.

The above constraints are visually illustrated in Fig. 13(b); there should not be any other droplets in a cube centered by any droplet [26].

Typically, the fluidic route of a droplet can be modeled either as a 2-pin net or a 3-pin net (for mixing). Each net is characterized by source port(s) and sink

port. Since the routing problem is usually linked with the placement procedure that gives the 2-D placement configurations of fluidic modules at different time intervals, the droplet routing problem is decomposed into a series of subproblems. A set of nets is considered for routing in each subproblem.

Droplet-Routing Algorithms. In [27], Böhringer used the concept of motion planning of multiple moving robots to derive the paths of moving droplets over the chip array. Paths are generated based on a prioritized A* search algorithm, where the priority of each droplet can be assigned randomly or based on specific hardware guidelines, such as droplet volatility. Note that this approach is time-consuming due to the time spent for searching through all feasible routing solutions; *i.e.,* evaluating a large number of candidate routes. Moreover, as a post-synthesis technique, routability is not always guaranteed. This was the motivation for Xu and Chakrabarty to integrate droplet routing in the synthesis flow in [28]. In their routing-aware synthesis approach, they attempt to achieve high-routability mapping of bioassay protocols to the microfluidic array. Instead of inferring accurate routability information based on post-synthesis routing, Xu and Chakrabarty adopt simple estimates of routability based on the inter-module distances; thus droplet routing can be flexibly incorporated in the previously mentioned simulated annealing synthesis. Using a real-life a protein assay, Xu and Chakrabarty show the benefits of incorporating routability information into the synthesis framework on the assay completion time, as illustrated in Fig. 14.

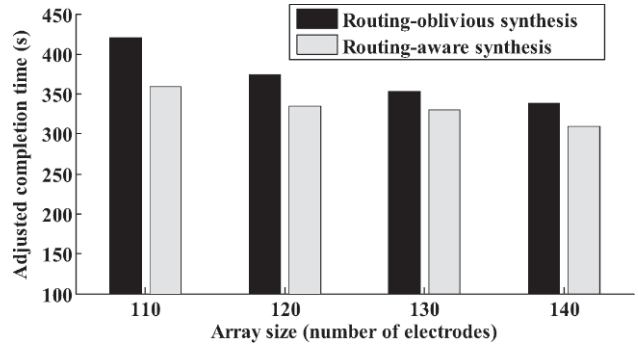


Fig. 14. Assay completion times (with droplet transportation time included) for Su and Chakrabarty [23] and for the routing-aware synthesis method in [28].

As a way of creating optimal solutions for droplet routes, Yuh et al. in [29] proposed a network-flow based routing algorithm that can concurrently route a set of non-overlapping nets. This method consists of three stages. The first stage calculates the criticality of each net based on the available array cells for each net. Global routing is considered in the second stage to identify a set of non-overlapping nets and the resulting information is used in a min-cost

max-flow (MCMF) algorithm to derive global-routing paths. In the third stage, negotiation-based detailed routing iteratively routes each net in the decreasing order of their criticality. Due to the iterative nature of the proposed algorithm, it shows better performance compared with [23, 27]. However, a significant bottleneck of the network-flow formulation is the distribution of blockages. To conservatively guarantee the fluidic constraints discussed above, a channel with at least three cells is considered in the network-flow formulation. Hence, if the width of the channel between blockages is less than three cells (even though a droplet can pass through it), the channel will not be utilized in the network-flow formulation, resulting in suboptimal solutions in terms of routability.

All the approaches mentioned above consider the blockages to be present on the grid during the entire routing time; this is usually not the case. Therefore, Keszocze et al. provide a droplet routing solution that considers temporality of blockages in [30]. In addition, to tackle the complexity of droplet routing, they exploit the power of *Satisfiability Modulo Theory* (SMT) solvers to obtain an exact routing solution. This approach has been shown to provide high-quality results, but questions are still being raised regarding its scalability.

3.3 Cross-Contamination Avoidance

Cross-contamination potentially arises as a result of unrestricted sharing of unit cells by various droplet routes. Cross-contamination occurs when the residue left behind by one droplet transfers to another droplet with undesirable consequences, such as misleading assay outcomes (false position, incorrect diagnosis, etc.). As a result, the droplet routing problem for biochips must consider the avoidance of cross-contamination during droplet transportation while satisfying both timing goals and fluidic constraints.

Cross-contamination avoidance was first introduced in [31] by Zhao and Chakrabarty. The goal of their proposal is twofold: 1) Avoiding cross contamination between different droplet routes. 2) Minimizing the time needed for droplet routing. Therefore, for each subproblem, the set of droplet routes need to be disjoint to avoid paths intersections (*i.e.*, paths sharing cells). By having a planar undirected graph $G = (V, E)$ such that each vertex $v \in V$ represents an electrode in the array and each edge $e \in E$ connects only two adjacent electrodes, the solution is obtained by applying the problem of finding disjoint paths (vertex-disjoint or edge-disjoint) for pairs of vertices in the graph G.

To minimize the time needed for routing, the Lee algorithm is adopted to obtain the shortest path between the net pins. Note that after any net has been routed, the cells occupied by its path are marked as obstacles for the unrouted nets to avoid cross-contamination. Therefore, the latter route is disjoint with respect to all the previous routes. It is obvious that the routing order within the subproblem is critical since it influences the routability of all the nets, such ordering can be optimized based on a predefined *priority* equation. Note that the routing of each net is based on an upper limit on the time that must not be exceeded. Hence, timing violations introduces the need for scheduling washing operations between successive droplet visits for cleaning.

Since cross-contamination can also happen across two subproblems, the droplet routes for the nets in the current subproblem should avoid sharing cells with the routes in the predecessor sub-problem. Therefore, after droplet routing in one subproblem, a wash operation needs to be introduced. In a wash operation, a wash droplet is routed to traverse selected cells and remove residue. The delay introduced by the wash operation is equivalent to the time needed to route the wash droplet. For a subproblem that includes a follow-up wash operation, its droplet-routes will not be treated as obstacles for the next subproblem.

Despite the novelity of Zhao and Chakrabarty's work, their method depends on interrupting the biochemical operations between successive subproblems, which introduces a significant timing overhead. Huang et al. interleaved functional droplet-routing steps with wash-droplet routing by introducing contamination aware routing methods [32]. Instead of using the shortest path in routing, they adopted a k-shortest path routing technique to minimize the contaminated spots within one subproblem. To convert the generated subproblem routing paths from sequential to concurrent ones in order to obtain the clock cycle corresponding each contaminationed spot, routing compaction by dynamic programming was used. With this information, a minimum-cost circulation (MCC) graph is adopted to simultaneously clean intra- and inter-contaminations to minimize the used cells and the execution time. This work, however, overlooked the order in which multiple cross-contamination spots are considered. Therefore, Zhao and Chakrabarty followed up in [33] with a technique that synchronizes wash-droplet routing with sample/reagent droplet-routing steps by controlling the arrival order of droplets at cross-contamination sites.

The effort spent on routing wash droplets depends on the placement topology obtained from previous automation stages. This was illustrated by Lin and Cheng in [34] through Fig. 15. Therefore, to cope with cross-contamination while reducing the effort spent on wash operations, Lin and Cheng introduced an early crossing reduction algorithm during placement. Crossing reduction was obtained using a bipartite matching formulation. More details about the algorithm can be found in [34].

4 Chip-Level Design

After fluidic-level synthesis, the control information for each electrode for performing bioassay protocols can be obtained. In chip-level design, electrode addressing and wire routing impose constraints on the electrical connectivity and signal planning. Therefore, chip-level design has been reported as the bottleneck for fabrication of DMFBs, and it is directly related to manufacturing complexity and fabrication cost [35]. In this part, we present an overview of automated chip-level design for DMFBs that addresses electrode addressing and wire routing.

4.1 Architecture of DMFBs

DMFBs typically rely on EWOD-based actuator, and electric potential is used to actuate electrodes by changing the wettability of droplets, such that droplets

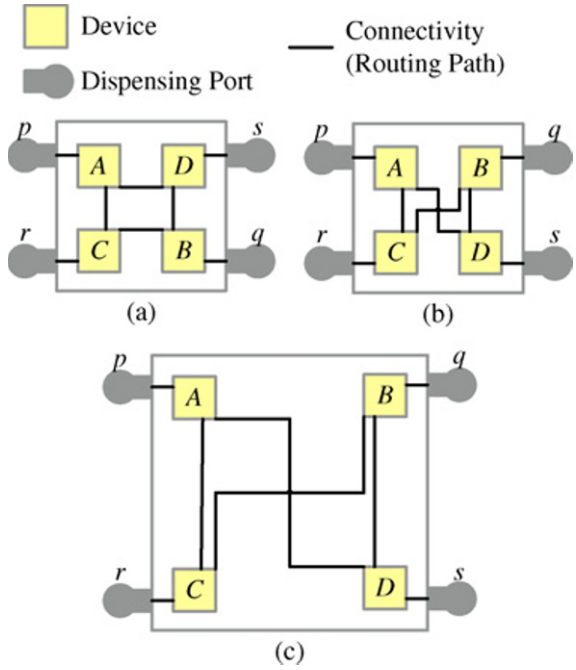


Fig. 15. Illustration of the fact that the minimum number of routing-path crossings is determined by the placement topology, and this minimum number cannot be reduced by routing or by compactness changes [34]. (a) Placement topology with zero routing-path crossing. (b) Different placement topology that implements the same design as in (a) but has at least one routing-path crossing. (c) Less compact placement result with the same placement topology as in (b) and consequently the same minimum number of routing crossings.

can be driven along the active electrodes [36]. A schematic view of a DMFB chip can be seen in Fig. 16(a), a typical DMFB contains a patterned electrode array, conduction wires, electrical pads, and a substrate [37]. A signal plan and electrical connections between pins and electrical pads can be established in a pin array (see Fig. 16(b)). As a result, research efforts on chip-level design can be grouped into two main design steps: 1) electrode addressing and 2) wire routing [38].

4.2 Electrode Addressing

Electrode addressing is a method whereby electrodes are addressed through control pins to identify input signals [38]. *Direct addressing* is an approach where each electrode is directly addressed with an independent control pin [36]. *Direct addressing* can maximize the flexibility of electrode controls. However, the num-

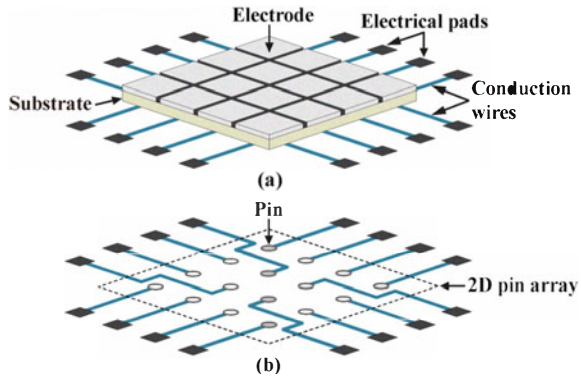


Fig. 16. (a) Schematic view of a DMFB chip. (b) Design model on a 2D pin array [38].

ber of control pins is usually limited for DMFBs, especially for DMFBs with a high-density electrode array.

Therefore, *pin-constrained* electrode addressing is used for solving this problem; a limited number of pins can be used to control a large number of electrodes in DMFBs. Electrode actuation sequences, which store the droplet control information, consist of a sequence of signal status ("1"(actuated), "0"(de-actuated), or "X"(don't-care)) of the electrode at each time step [39]. A don't-care signal "X" can be mapped to either "0" or "1". *Broadcast addressing* in [39] focuses on electrode grouping based on the compatibility of actuation sequences. If two actuation sequences are compatible with each other, these two electrodes can be connected to the same control pin to reduce the total number of pins. For example, by replacing "X" in the actuation sequence of e_4 with "1", the actuation sequences for electrodes e_4 and e_5 in Fig. 17(b) can be made to be compatible with each other, therefore, these two electrodes can be controlled by only one pin.

A *compatibility graph* is used for broadcasting addressing in [39]. As shown in Fig. 18, each vertex represents an electrode and an edge between two electrodes indicates that their actuation sequences are compatible with each other. Based on the compatibility graph, electrode addressing is mapped to the *clique partitioning* problem, which is known to be NP-hard [40].

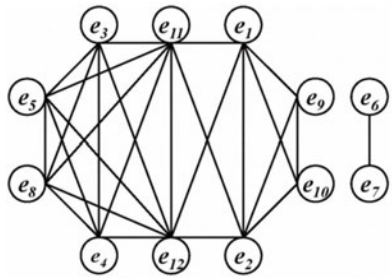
Therefore, several heuristic algorithms have been proposed for electrode addressing. A partitioning algorithm based on "droplet trace", which is extracted from the scheduling and droplet routing results, has been proposed in [41]. In combination with this partitioning algorithm, the "Connect-5 algorithm" is used for efficient pin assignment. An ILP-based algorithm was proposed in [42], which effectively minimizes the assay time and pin count. The work in [43] formulated electrode addressing and power reduction into a minimum-cost and maximum-flow (MCMF) network, and a progressive electrode-addressing scheme was proposed for reducing design complexity.

(a) (c) Pin count: 12 (d) Pin count: 5

Electrode	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
Activation Sequence	1	1	0	0	0	X	X	0	X	X	X	X
	0	0	1	1	1	0	0	1	X	X	X	X
	0	0	0	0	0	1	1	0	0	0	X	X
	X	X	0	0	0	0	0	0	1	1	0	0
	X	X	X	X	1	0	0	1	X	X	1	1

(b)

Fig. 17. (a) Electrodes for handling fluidic operations. (b) Scheduled fluidic functions in the form of actuation sequences. (c) Electrode addressing using direct-addressing scheme. (d) Electrode addressing using the broadcast-addressing scheme.



(a)

Two electrode grouping results		
Pin	Result 1	Result 2
1	e_1, e_{10}	e_1, e_2
2	e_3, e_8, e_{12}	e_3, e_4, e_8
3	e_6, e_7	e_6, e_7
4	e_2, e_9	e_9, e_{10}
5	e_4, e_5, e_{11}	e_5, e_{11}, e_{12}

(b)

Fig. 18. (a) A *compatibility graph*. (b) Two possible electrode-grouping results.

4.3 Wire Routing

After electrodes have been addressed with control pins, wire routing is required to establish the correspondence between control pins and signal pads. A conduction wire is used for the connection of inner pins to outer signal pads, which is similar to the typical escape routing problem in VLSI design [44]. However, in pin-constrained DMFBs, multiple-terminal pins sharing the same signal are required to be routed together, which results in the interdependence between electrode addressing and wire routing. This means that different electrode addressing solutions may lead to different wire-routing results. Therefore, if electrode addressing and wire routing are not considered together, the quality of the design may be adversely affected. For example, separate considerations of electrode addressing and wire routing results in an infeasible routing result in

Fig. 19(a), while the simultaneous consideration of electrode addressing and wire routing results in a feasible routing solution (see Fig. 19(b)).

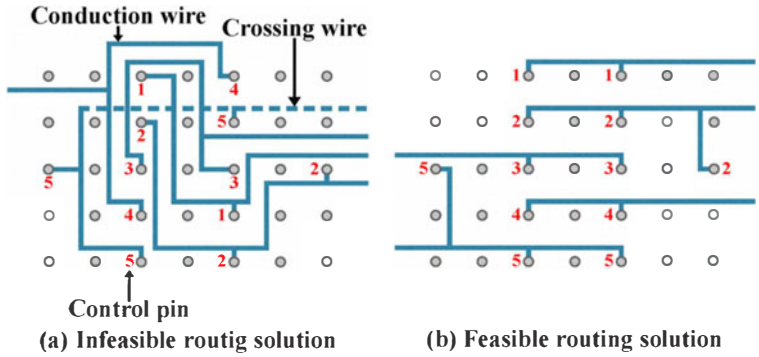


Fig. 19. Consideration of electrode addressing and routing [38]: (a) separately; (b) simultaneously.

Very little work has considered the problem of automated wire routing for DMFBs. The work in [46] proposed a pin-count-aware routing algorithm for broadcast electrode-addressing DMFBs. The routing problem can be effectively solved using two kinds of flow formulations, *maximum-flow network* and *minimum flow network*, which result in lower pin count, higher routability and shorter wirelength. The work in [45] proposed an ILP-based two-stage technique of global routing and progressive routing. In the global routing stage, the pin count and wirelength can be simultaneously minimized in a global manner. Then, progressive routing iteratively completes the addressing and routing steps using a minimum-cost maximum-flow model.

5 Cyberphysical Design for Digital Microfluidics

During bioassay exection on a DMFB, the droplets are likely to encounter some parametric changes or operational errors due to the nature of fluid-handling operations (such as unbalanced droplets resulting from split operation) [49], or due to the electric charge that gets trapped over time in the dielectric layer below the electrodes [50]. With recent advances in sensing techniques for DMFBs (e.g., integrated waveguides [47], capacitive sensors [3], and droplet monitoring using CCD cameras [48]), the design of physical-aware synthesis/control software is now feasible. By physical-aware, we refer to the fact that the software can receive information about the outcome (e.g., error-free, erroneous, and improper droplet size) of fluid-handling operations based on feedback from the sensing system [50]. This information can be used in various forms to adapt the control software to the new condition of the cyberphysical system. Figure 20 depicts the components of a cyberphysical microfluidic platform.

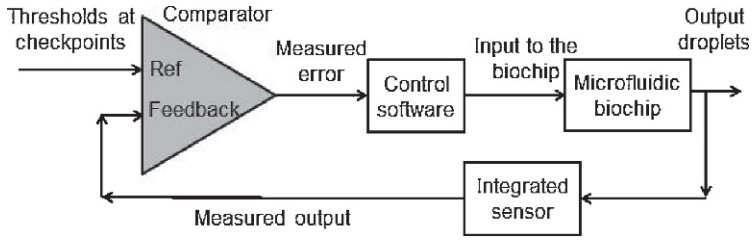


Fig. 20. Schematic of the cyberphysical digital microfluidic system [50].

In this section, we describe how the concept of cyberphysical systems can be leveraged in designing adaptive synthesis tools for DMFBs. The following cases will be analyzed: operation variability inherent from erroneous change in droplet size, uncertainties that are inherent in the completion times of fluidic operations such as mixing and thermal cycling, and erroneous execution of an operation due to electrode breakdown.

5.1 Online Synthesis With Operation Variability

Under ideal conditions, when two droplets come together for a mixing operation, the resulting droplet has a volume equal to the sum of the volumes of the input droplets. After a split operation, the resulting droplets have volumes equal to half of the larger merged droplet. However, the volume of a droplet can also vary due to parametric faults, such as non-uniform electrode coating or unequal actuation voltages [49]. Since biochemical applications have high accuracy requirements, it is imperative to address the biochemical operation variability that results in erroneous volume variations.

In [49], Alistar et al. proposed a biochemical application model that captures the sensing operations needed to detect an error, and the subgraphs that have to be executed for recovery. Figure 21(b) shows the insertion of the sensing operations O_8 and O_9 into the original protocol sequencing graph $G(V, E)$. During a sensing operation, the droplet is transported to an on-chip capacitive sensor [3] to have its volume measured. Two outcomes are possible after a sensing operation; success or failure. Alistar et al. adapted the application model to this conditon by incorporating conditional edges that connect the sensing operation to the corresponding successor operations, as shown in Fig. 21(c). The edge labeled *err* corresponds to the case when the volume sensed was erroneous, whereas the edge *not err* represents correct droplet volume. To recover from an erroneous situation by incorrect droplet volume, it is necessary to provide recovery subgraphs for each sensing operation even before starting execution, such subgraph contains all operations needed to recover from a sensed erroneous droplet volume. Figure 22 shows the recovery subgraphs for both O_8 and O_9.

Based on this application model, the goal is to determine an implementation that minimizes the completion time in case no errors occur, and at the same

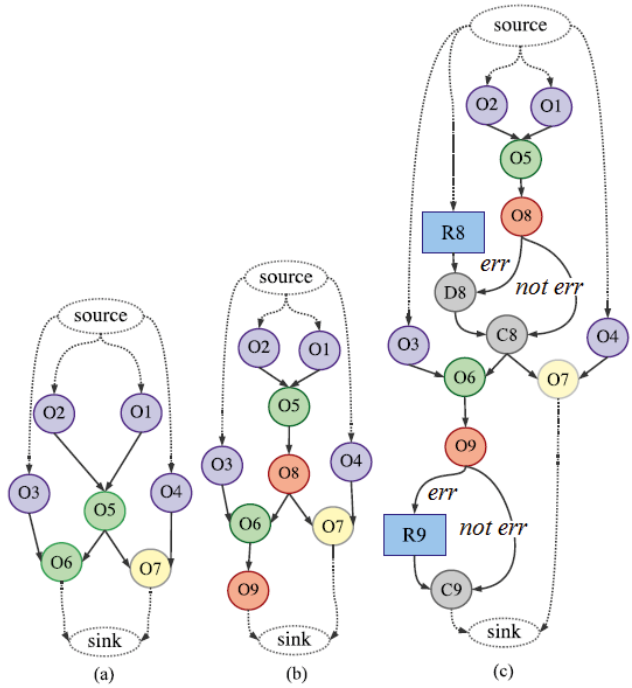


Fig. 21. Biochemical application model [49]. (a) Original sequencing graph. (b) Insertion of sensing operations. (c) A complete application graph with the consideration of sensing outcomes.

time, is fault-tolerant to operation variability. Therefore, [49] followed a strategy that has two components: (1) an offline synthesis algorithm is initially used to synthesize the application while considering the minimization of the application completion time, and (2) an online synthesis algorithm that is based on List-Scheduling (LS) is employed to minimize the recovery time. The authors used the Tabu Search (TS)-based approach for their offline synthesis, but they provided an LS-based online synthesis (ONS) approach that can better exploit the actual biochip configuration; thus protocol execution can be adapted according to sensing outcomes. The ONS is invoked whenever an error is detected, and the associated subgraph is then employed for recovery.

5.2 Synthesis with Completion-Time Uncertainties

The precision of fluidic operations is vital for the accuracy of analytical bioassays. For example, in quantitative measurement for glucose concentration in blood [51], accurate measurements cannot be obtained if the mixing time for the blood sample and the enzymatic reagent is not controlled precisely. Due to the inherent variability and randomness of biological/chemical processes, the

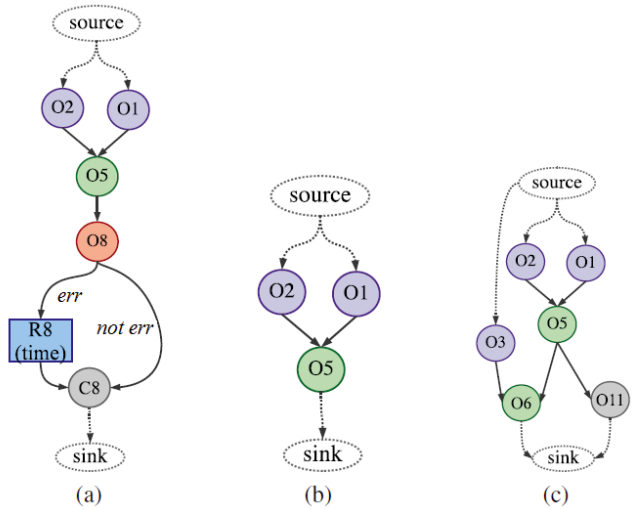


Fig. 22. Recovery subgraphs for O_8 and O_9 [49].

problem of completion-time uncertainties in fluidic operations remains even after careful characterization of a bioassay. To overcome this problem, Luo et al. exploited the advantages of cyberphysical DMFBs to facilitate the precise implementation of the essential operations such as droplet dispensing and mixing without specifying a module library [52].

The basic idea is that some fluidic operations are frequency-sensitive. For example, the rate at which a droplet is transported or dispensed is proportional to the clock frequency; thus the higher the frequency, the faster the completion. However, in order to avoid electrode degradation due to increasing switching activity [53], it is necessary to choose an appropriate clock frequency. Therefore, Luo et al. suggested running different categories of operations at different clock frequencies. This was achieved by scheduling transportation/dispensing operations and dilution/mixing operations at different time segments. The time segment to implement droplet transportation is defined as the transportation phase (T phase), and the segment to implement dilution/mixing operations is defined as the dilution/mixing phase (D/M phase) [see Fig. 23(a)]. For each phase, only transportation operations or the dilution/mixing operations are carried out on the chip.

At runtime, the biochip operates under clock frequency f_T in the T phase. Output droplets of previous steps and droplets dispensed from reservoirs are moved to the modules where the subsequent dilution/mixing operations are to be carried out. After all the droplets arrive at their destination modules, the biochip enters the D/M phase. The dilution/mixing operations that are scheduled in the same phase start together, and they are carried out under clock frequency $f_{D/M}$. When the feedback from sensors indicates all the dilution/mixing operations

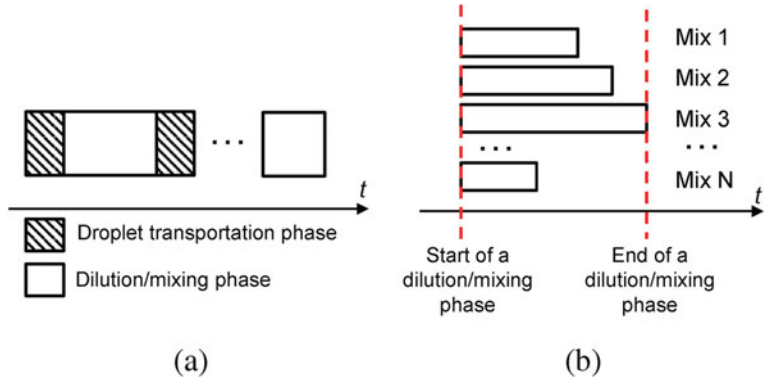


Fig. 23. (a) Droplet transportation and dilution/mixing operations are scheduled in different phases. (b) Start and end time of a D/M phase. [52].

have already been completed, the D/M phase ends and the biochip enters the next T phase, as shown in Fig. 23(b). In this way, based on sensor feedback, the biochip switches between the T phase and the D/M phase with different clock frequencies. This approach can be implemented either using a tunable frequency-divider implemented on the field-programmable gate array (FPGA) of the cyberphysical system or by controlling the output frequency of the signal generator using the control software [52]. In this case, the time spent on each dilution/mixing operation is determined based on the sensor feedback.

In their work, Luo et al. described synthesis software that can determine the schedule and module-placement for the operations on a level-by-level basis by considering the operations dependencies. This approach was introduced for sequencing graphs with a directed-tree structure, as shown in Fig. 24(a). In the example shown in Fig. 24(a), operations 1, 2, and 6 are three mixing operations. Here, 1 and 2 are the lowest-level operations and their outputs are the inputs of 6, hence, 1 and 2 must be completed before 6 starts. Based on the interdependency relationship, operations 1 and 2 are implemented in the same D/M phase, while operation 6 is schedule to be implemented in the next D/M phase, as shown in Fig. 24(b). Next, mixing operations 1 and 2 are mapped to two mixers on the biochip. Since the inputs of mixing operation 6 are the outputs of operations 1 and 2, the authors refer to the region that overlaps with the modules for 1 and 2 as the execution region of operation 6, as shown in Fig. 24(c). After the mixing operation has been completed for each mixer, the product droplet stays inside the mixer until the end of the D/M phase, *i.e.*, part of the mixer works as a storage unit. Note that there is no module-to-module transportation in this case.

For sequencing graphs that are not directed trees, the synthesis results can be derived by a means of graph partitioning, thus converting the sequence graph into a set of directed trees [52].

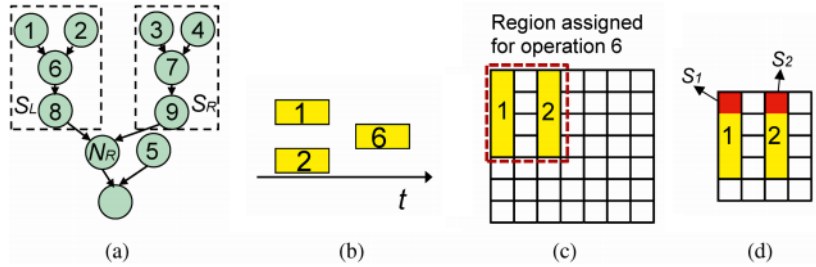


Fig. 24. (a) Sequencing graph with a tree structure [52]. (a) Droplet transportation and dilution/mixing operations are scheduled in different phases. (b) Scheduling results for mixing operations 1, 2, and 6. (c) Module-placement for 1, 2, and 6. (d) Storage units S_1 and S_2 inside the modules assigned for operations 1 and 2.

5.3 Synthesis with Error Recovery

A DMFB device is said to have a failure if its operation does not match its specified behavior. In order to detect defects using electrical methods, fault models that efficiently represent the effect of physical defects at some level of abstraction have been described in [55, 56]. These models can be used to capture the effect of physical defects that produce incorrect behavior. Faults can be caused by manufacturing imperfections, or by degradation during use as electrodes are actuated. Possible causes of defects are dielectric breakdown due to high voltages, degradation of the insulator because of long actuation periods for the electrode, short-circuited electrodes, and open in the metal connection between the electrode and the control source. These defects result in a failure in electrode activation for droplet transportation.

The correctness of bioassay outcomes can be determined by utilizing on-chip detectors. In addition, physical-aware control software can be used in a DMFB platform to implement an error-recovery method. In this subsection, we discuss some methods that consider embedding error recoverability in the synthesis of DMFBs. More analysis from the perspective of design space parameters for error recovery can be found in [56].

Resynthesis for Error Recovery. In [50], Luo et al. introduced a reliability-driven reconfiguration method that can dynamically respond to error signals from the underlying hardware. Utilizing two different sensing approaches (CCD camera-based sensor or integrated optical detector), the proposed system can adapt to the new situation by deriving new schedules, module placement, and droplet routing pathways with minimum impact on the time-to-response. Thus a dynamic resynthesis approach is considered.

To leverage the advantages of dynamic resynthesis, additional intermediate product droplets must be stored in specially designated locations of the chips to facilitate recovery in a manner that will be explained later. These droplets

are defined as *backup* droplets. Also, additional droplets of samples and reagents must be dispensed from reservoirs for error recovery.

Characterizing the response of the control system when an error occurs depends on the type of erroneous fluid-handling operations. Therefore, fluidic-handling operations are formally categorized as follows [50]. Category I: this is the set of all reversible operations that can be simply re-executed when an error is detected, such as dispensing operations. Category II: this includes the set of nonreversible operations for which immediate predecessors can provide backup droplets. Operations in this category can also be simply re-executed because their input droplets are stored on chip. Category III: this corresponds to the set of nonreversible operations for which their immediate predecessors cannot provide backup droplets. If an error occurs in an operation in this category, we not only need to re-execute the operation itself but also to *backtrace* to its predecessors. Figure 25(a) clarifies the above-mentioned classification on a typical sequencing graph. Figure 25(b) shows that some electrodes on the boundary of the biochip are reserved for storage of backup droplets. The algorithm used to determine the recovery sequence for operation opt_i is shown in Fig. 26. Note that the operator P_r is defined as $P_r : O \rightarrow \bigcup_{i=O_1,O_2,\ldots O_k} \{opt_i, opt_j | opt_j \in pred(opt_i)\ \forall\ j\}$.

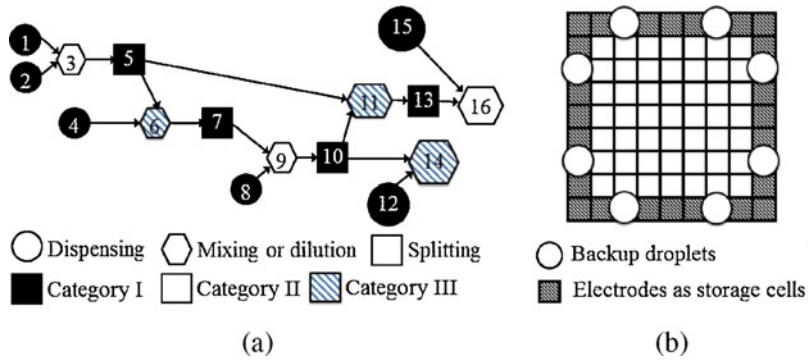


Fig. 25. (a) Example of a sequencing graph corresponding to a bioassay protocol. (b) Layout of a biochip with reserved area for error recovery [50].

Besides the initial synthesis (which is also known as offline date preparation [50]), the control software is also responsible for monitoring the execution of the bioassay and performing resynthesis if an error is detected. The resynthesis process is tightly coupled with the category to which the erroneous operation belongs as discussed above. All operations that need to be rescheduled are placed in a priority queue Q based on topological order. These operations include error recovery operations and all successors of the erroneous operation. The algorithm that is used for dynamic resynthesis of a bioassay is shown in Fig. 27.

Recovery Backtrace (opt_i)
1 Classify operations into Category I, Category II, and Category III;
2 Initialization of $R_i : R_i = opt_i$;
3 Initialization of intermediate variable $R_e : R_e = P_r(R_i)$;
4 **while** $(R_E - R_i) \cap \{$ Set of operations in Category III $\} \neq \phi$ **do**
5 Update $R_i : R_i = P_r(R_i)$;
6 Update $R_e : R_e = P_r(R_i)$;
7 **end while**
8 $R_i = R_e$;
9 R_i is the set of recovery operations for opt_i;

Fig. 26. Pseudocode for determining the recovery operations for opt_i [50].

Dynamic Resynthesis ()
1 Localize the erroneous operation according to the feedback at checkpoints;
2 Determine the operations which need to be adjusted and store them into a priority queue Q;
3 Delete all initial synthesis results for operations in Q;
4 **while** $Q \neq \phi$ **do**
5 Search available resource for operation q_0 which has the highest priority in Q;
6 Remove q_0 from Q;
7 **end while**

Fig. 27. Pseudocode for dynamic resynthesis of a bioassay [50].

Dictionary-Based Error Recovery. Even though dynamic resynthesis is able to leverage the advantages of chip integration in cyberphysical systems, its reaction time for error recovery still represents a bottleneck for several biochemical application that require highly precise time-control in each step of chemical synthesis, such as flash chemistry. To overcome this drawback, Luo et al. proposed a dictionary-based hardware-assisted error-recovery method [54].

The key idea in this method is to precompute and store recovery actuation sequences for all errors of interest that can occur during a bioassay. When an error is detected by on-chip sensors during the execution of a bioassay, the cyberphysical system can simply look up the recovery solution in the dictionary rather than performing online resynthesis using an in-the-loop computer. This dictionary-based solution therefore reduces response time and enables flash chemistry [54].

The dictionary-based error recovery approach can be implemented using the finite-state machine (FSM) shown in Fig. 28. The control signals for the biochip are determined by the current state of the FSM; the state transition of the FSM is triggered by the analog feedback indicating that an error has occurred. The error dictionary plays the role of a precomputed database that links each possible state to the corresponding outputs. Entries in the dictionary record all possible errors and the resynthesis results, which include the corresponding error

recovery operations. When an error occurs, the cyberphysical system can utilize the dictionary and load the precomputed synthesis results.

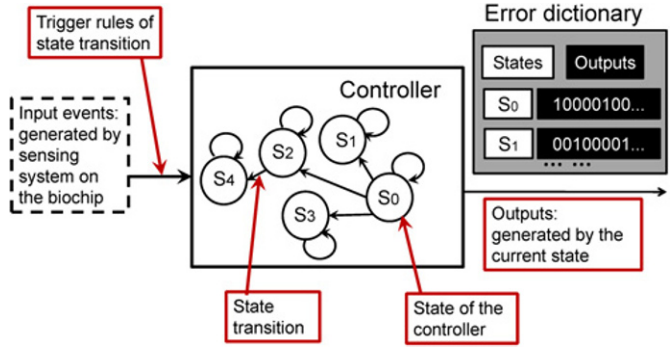


Fig. 28. The finite-state machine control system implementation of the cyberphysical system. The FSM runs on the microcontroller or the FPGA, and its current state determines control signals applied on the biochip. A state transition of the FSM is triggered by the detection of error(s) [54].

Since the error dictionary needs to store resynthesis solutions for errors of interest, the data volume can be high. Consider a typical protein dilution bioassay with 103 fluidic operations. If we limit ourselves to errors involving at most two operations, the memory required for storage of the error dictionary without compaction can be as high as 28.75 MB. However, the memory on a low-cost FPGA is typically limited. For example, according to the datasheet of several widely-used FPGAs (which cost less than $50), the capacities of their on-chip memories are less than 1 MB. External memory devices can increase the size of total memory, however, their prices can be as high as the low-cost FPGAs. Therefore, the above solutions are not compatible with the goal of low-cost biochip platforms that can be used for field deployment and point-of-care clinical diagnostics. To address this problem, Luo et al. further proposed a compaction procedure for an error dictionary, which includes resynthesis solutions for all the possible error combinations in bioassay [54]. Then, the vectors (*i.e.,* actuation matrices) stored in the dictionary are compacted in a lossless manner. The size of compacted dictionary in the above example can be reduced to only 0.96 MB. Figure 29 shows the components of the dictionary-based error-recovery system. Note that no software execution is needed for online error recovery, thus the need for a computer and the related interfaces can be eliminated.

Experimental Case Study. Recently, Hu et al. implemented a hardware-assisted error recovery platform [55]. Errors in droplet transportation on a chip array are detected using a capactive sensor. The schematic of the control system

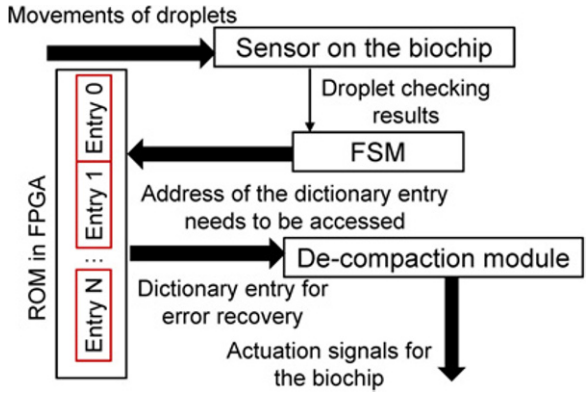


Fig. 29. Modules and their interconnections for the dictionary-based error recovery system [54].

setup is shown in Fig. 30 [55, 56]. The hardware/software interface is realized through seamless interaction between control software, an off-the-shelf microcontroller, a shift register for synchronous actuation of 32 chip electrodes, and a frequency divider implemented on an FPGA. The intermediate microcontroller provides the following functionalities: 1) communication with the desktop application via a USB link, 2) a parallel-to-serial conversion for the 32-bit electrode actuation vector provided by the desktop and injecting the serial data into the shift register, 3) metering the signal frequency received from the FPGA to interpret the capacitive sensing readout.

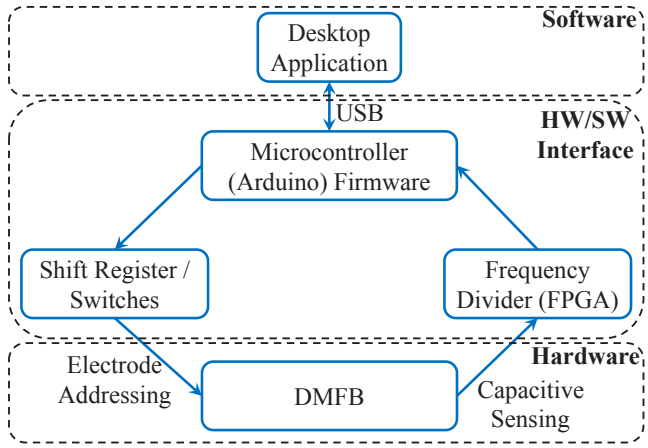


Fig. 30. Illustration of the control system loop [55].

On the hardware side, some electrodes are equipped with the sensing circuit; these electrodes are designated as *checkpoints*. For effective error recovery, a number of such checkpoints must be incorporated [55]. However, precise localization of errors is not possible with this sensor technology because the only information available to us is that a droplet is stuck within a region between two checkpoints. Therefore, to ensure reliability whenever an error is detected, region-level bypassing is employed during control-software rollback to recover from the error. Based on this scenario, the chip array is divided into several regions with a checkpoint associated with each region. When a "missing droplet" error is detected at a checkpoint, it can be inferred that the defect lies on the path between this checkpoint and the previous checkpoint on the designated droplet route, thus rollback is initiated. A droplet under test retraces its path from the current checkpoint to the previous checkpoint. Note that the rollback is deemed to have been successful when the previous checkpoint once again reports the presence of a droplet (after a known number of clock cycles). If retracing fails, i.e., the previous checkpoint does not report a droplet, we conclude that the droplet is irreversibly stuck [55]. As a result, a new droplet needs to be dispensed and the cells between the two checkpoints have to be permanently discarded for reliability.

In this setup, Hu et al. used a 32-electrode chip. The chip was divided into four regions. The positions of checkpoints are determined to ensure that each region has exactly one checkpoint. From the pre-computed activation sequences, the arrival time for a droplet at a checkpoint is known. Hence, based on the sensor readout, the control software decides whether a droplet has arrived as expected. If a droplet reaches a checkpoint as planned, the experiment will be continued and the droplet will enter the next region. If not, the reconfiguration steps will be triggered automatically, a backup route will be generated in real-time and the new electrode activation sequences will be immediately fed into the biochip.

Hu et al. carried out several experimental runs using the fabricated chip, sensing hardware, the hardware/software interface, and the control software described before [55]. Videos illustrate successful re-routing of droplets and bypassing of faults whenever an error is detected [57].

6 Conclusion

We have presented a survey of research on design automation for digital microfluidic biochips. We first provided an overview of the digital-microfluidic platform showing the principle of operation based on the electrowetting phenomenon. Advances and techniques from different research groups around the world in both architectural as well as physical-level synthesis have been illustrated. The use of current advances in biosensor technology for designing cyberphysical digital-microfluidic biochips has been highlighted. We have presented three applications to show how cyberphysical integration can be used. Finally, demonstration of a laboratory experiment for error-recovery based on the cyberphysical integration

is now a door-opener for the deployment and use of biochips in the emerging marketplace.

References

1. Schulte, T., Bardell, R. L., Weigl, B. H.: Microfluidic technologies in clinical diagnostics. Clinica Chimica Acta 321, 1–10 (2002)
2. Guiseppi-Elie, A., Brahim, S., Slaughter, G., Ward, K. R.: Design of a subcutaneous implantable biochip for monitoring of glucose and lactate. IEEE Sensors Journal 5, 345–355 (2005)
3. Fair, R. B.: Digital microfluidics: is a true lab-on-a-chip possible? Microfluidics and Nanofluidics 3, 245–281 (2007)
4. Berthier, J.: Micro-drops and digital microfluidics. William Andrew (2012)
5. Srinivasan, V., Pamula, V. K., Fair, R. B.: Droplet-based microfluidic lab-on-a-chip for glucose detection. Analytica Chimica Acta 507, 145–150 (2004)
6. Chen, X., Cui, D., Liu, C., Li, H., Chen, J.: Continuous flow microfluidic device for cell separation, cell lysis and DNA purification. Analytica chimica acta 584, 237–243 (2007)
7. Chang, Y.-H., Lee, G.-W., Huang, F.-C., Chen, Y.-Y., Lin, J.-L.: Integrated polymerase chain reaction chips utilizing digital microfluidics. Biomedical Microdevices 8, 215–225 (2006)
8. Luan, L., Evans, R. D., Jokerst, N. M., Fair, R. B.: Integrated optical sensor in a digital microfluidic platform. IEEE Sensors Journal 8, 628–635 (2008)
9. Sista, R., Hua, Z., Thwar, P., Sudarsan, A., Srinivasan V., Eckhardt, A., Pollack, M., Pamula, V.: Development of a digital microfluidic platform for point of care testing. Lab on a Chip 8, 2091–2104 (2008)
10. Hu, K., Hsu, B.-N., Madison, A., Chakrabarty, K., Fair, R. B.: Fault detection, real-time error recovery, and experimental demonstration for digital microfluidic biochips. Proc. DATE, 559–564 (2013)
11. Cho, S. K., Moon, H., Kim, C.-J.: Creating, transporting, cutting, and merging liquid droplets by electrowetting-based actuation for digital. microfluidic circuits. IEEE Journal of Microelectromechanical Systems 12, 70–80 (2003)
12. Mugele, F., Baret, J.-C.: Electrowetting: from basics to applications. Journal of Physics: Condensed Matter 17, R705 (2005)
13. Berthier, J., Clementz, Ph., Raccurt, O., Jary, D., Claustre, P., Peponnet, C., Fouillet, Y.: Computer aided design of an EWOD microdevice. Sensors and Actuators A: Physical 127, 283–294 (2006)
14. Su, F., Chakrabarty, K.: Unified high-level synthesis and module placement for defect-tolerant microfluidic biochips. Proc. DAC, 825–830 (2005)
15. Baranyi, J., Csernus, O., Beczner, J.: Error analysis in predictive modelling demonstrated on mould data. International Journal of Food Microbiology 170, 78–82 (2014)
16. Su, F., Chakrabarty, K.: Architectural-level synthesis of digital microfluidics-based biochips. Proc. ICCAD, 223–228 (2004)
17. Ricketts, A., Irick, K., Vijaykrishnan, N., Irwin, M.: Priority scheduling in digital microfluidics-based biochips. Proc. DATE, 329–334 (2006)
18. Zhao, Y., Xu, T., Chakrabarty, K.: Integrated control-path design and error recovery in the synthesis of digital microfluidic lab-on-chip. ACM J. Emerg. Technol. Comput. Syst 6, 11 (2010)

19. Zhao, Y., Xu, T., Chakrabarty, K.: Online synthesis for error recovery in digital microfluidic biochips with operation variability. Proc. DTIP, 53–58 (2012)
20. Chakrabarty, K., Fair, R.B., Zeng, J.: Design Tools for Digital Microfluidic Biochips: Toward Functional Diversification and More Than Moore. IEEE Trans. on CAD of Integrated Circuits and Systems 29, 1001–1017 (2010)
21. Ho, T.-Y., Chakrabarty, K., Fair, R.B., Pop, P.: Digital microfluidic biochips: Recent research and emerging challenges. Proc. CODES+ISSS, 335–343 (2011)
22. Yuh, P.-H., Yang, C.-L., Chang, Y.-W.: Placement of Defect-Tolerant Digital Microfluidic Biochips Using the T-tree Formulation. ACM J. Emerg. Technol. Comput. Syst 3, 13:1–32 (2007)
23. Su, F., Chakrabarty, K.: Module placement for fault-tolerant microfluidics-based biochips. ACM Trans. Design Automation of Electronic Systems 11, 682–710 (2006)
24. Maftei, E., Pop, P., Madsen, J.: Tabu Search-Based Synthesis of Dynamically Reconfigurable Digital Microfluidic Biochips. Proc. CASES, 195–203 (2009)
25. Fair, R. B. et al.: Electrowetting-based on-chip sample processing for integrated microfluidics. Proc. IEDM, 32.5.1–32.5.4 (2003).
26. Cho, M., Pan, D.Z.: A High-Performance Droplet Routing Algorithm for Digital Microfluidic Biochips. IEEE Trans. on CAD of Integrated Circuits and Systems 27, 1714–1724 (2008)
27. Böhringer, K.F.: Modeling and Controlling Parallel Tasks in Droplet-Based Microfluidic Systems. IEEE Trans. on CAD of Integrated Circuits and Systems 25, 334–344 (2006)
28. Xu, T., Chakrabarty, K.: Integrated Droplet Routing and Defect Tolerance in the Synthesis of Digital Microfluidic Biochips. ACM J. Emerg. Technol. Comput. Syst 4, 11:1–23 (2008)
29. Yuh, P.-H., Yang, C.-L., Chang, Y.-W.: BioRoute: A Network-Flow-Based Routing Algorithm for the Synthesis of Digital Microfluidic Biochips. IEEE Trans. on CAD of Integrated Circuits and Systems 27, 1928–1941 (2008)
30. Keszocze, O., Wille, R., Drechsler, R.: Exact routing for digital microfluidic biochips with temporary blockages. Proc. ICCAD, 405–410 (2014)
31. Zhao, Y., Chakrabarty, K.: Cross-Contamination Avoidance for Droplet Routing in Digital Microfluidic Biochips. Proc. DATE,1290–1295 (2009)
32. Huang, T.-W., Lin, C.-H., Ho, T.-Y.: A Contamination Aware Droplet Routing Algorithm for Digital Microfluidic Biochips. Proc. ICCAD, 151–156 (2009)
33. Zhao, Y., Chakrabarty, K.: Synchronization of Washing Operations with Droplet Routing for Cross-Contamination Avoidance in Digital Microfluidic Biochips. Proc. DAC, 635–640 (2010)
34. Lin, C.C.-Y, Chang, Y.-W.: Cross-Contamination Aware Design Methodology for Pin-Constrained Digital Microfluidic Biochips. IEEE Trans. on CAD of Integrated Circuits and Systems 30, 817–828 (2011)
35. Su, F., Chakrabarty, K., Fair, R.: Microfluidics-based biochips: technology issues, implementation platforms, and design-automation challenges. IEEE Trans. on CAD of Integrated Circuits and Systems 25, 211–223 (2006)
36. Pollack, M., Shenderov, A., Fair, R.: Electrowetting-based actuation of droplets for integrated microfluidics. Lab Chip 2, 96–101 (2002)
37. Gong, J., Kim, C.: Direct-referencing two-dimensional-array digital microfluidics using multilayer printed circuit board. Journal of Microelectromechanical Systems 17, 257–264 (2008)
38. Huang, T., Lin, Y., Chang, J., Ho, T.: Chip-level design and optimization for digital microfluidic biochips. Proc. MWSCAS, 1–4 (2011)

39. Xu, T., Chakrabarty, K.: Broadcast electrode-addressing for pin-constrained multi-functional digital microfluidic biochips. Proc. DAC, 173–178 (2008)
40. Gross, J., Yellen, J.: Graph theory and its applications. CRC press (2005)
41. Xu, T., Chakrabarty, K.: Droplet-trace-based array partitioning and a pin assignment algorithm for the automated design of digital microfluidic biochips. Proc. CODES+ISSS, 112–117 (2006)
42. Lin, C., Chang, Y.: ILP-based pin-count aware design methodology for microfluidic biochips. IEEE Trans. on CAD of Integrated Circuits and Systems 29, 1315–1327 (2010)
43. Huang, T., Su, H., Ho, T.: Progressive network-flow based power-aware broadcast addressing for pin-constrained digital microfluidic biochips. Proc. DAC, 741–746 (2011)
44. Alpert, C., Mehta, D., Sapatnekar, S.: Handbook of algorithms for physical design automation. CRC press (2008)
45. Huang, T., Ho, T.: A two-stage ILP-based droplet routing algorithm for pin-constrained digital microfluidic biochips. Proc. DAC, 201–208 (2010)
46. Huang, T., Yeh, S., Ho, T.: A network-flow based pin-count aware routing algorithm for broadcast electrode-addressing EWOD chips. Proc. ICCAD, 425–431 (2010)
47. Jokerst, N. and others: Progress in chip-scale photonic sensing. IEEE Trans. Biomed. Circuits Syst 3, 202–211 (2009)
48. Shin, Y.-J., Lee, J.-B.: Machine vision for digital microfluidics. Review of Scientific Instruments 81, 014 302:1–7 (2010)
49. Alistar, M., Pop, P., Madsen, J.: Online synthesis for error recovery in digital microfluidic biochips with operation variability. Proc. DTIP, 25–27 (2012)
50. Luo, Y., Chakrabarty, K., Ho, T.-Y.: Error Recovery in Cyberphysical Digital-Microfluidic Biochips. IEEE Trans. on CAD of Integrated Circuits and Systems 32, 59–72 (2013)
51. Hadwen, B., Broder, G., Morganti, D., Jacobs, A., Brown, C.: Programmable large area digital microfluidic array with integrated droplet sensing for bioassays. Lab Chip 12, 33053313 (2012)
52. Luo, Y., Chakrabarty, K., Ho, T.-Y.: Biochemistry Synthesis on a Cyberphysical Digital Microfluidics Platform Under Completion-Time Uncertainties in Fluidic Operations. IEEE Trans. on CAD of Integrated Circuits and Systems 33, 903–916 (2014)
53. Huang, L., Koo, B., Kim, C. J.: Evaluation of anodic Ta_2O_5 as the dielectric layer for EWOD devices. Proc. IEEE MEMS, 428–431 (2012)
54. Luo, Y., Chakrabarty, K., Ho, T.-Y.: Real-time error recovery in cyberphysical digital-microfluidic biochips using a compact dictionary. IEEE Trans. on CAD of Integrated Circuits and Systems 32, 1839–1852 (2013)
55. Hu, K., Hsu, B.-N., Madison, A., Chakrabarty, K., Fair, R.: Fault detection, real-time error recovery, and experimental demonstration for digital microfluidic biochips. Proc. DATE, 559–564 (2013)
56. Ibrahim, M., Chakrabarty, K.: Error recovery in digital microfluidics for personalized medicine. Proc. DATE, (2015) [accepted for publication]
57. http://microfluidics.ee.duke.edu/Published_Videos/2013_DATE/

Intuitive Interaction with Robots - Technical Approaches and Challenges

Elsa Andrea Kirchner[1,2], Jose de Gea Fernandez[2], Peter Kampmann[2], Martin Schröer[1], Jan Hendrik Metzen[1], and Frank Kirchner[1,2]

[1]Robotics Group, University of Bremen, Robert-Hooke Strasse 1, 28359 Bremen, Germany
[2]Robotics Innovation Center, German Research Center for Artificial Intelligence (DFKI GmbH), Robert-Hooke Strasse 1, 28359 Bremen, Germany
elsa.kirchner@dfki.de

http://robotik.dfki-bremen.de

Abstract. A challenging goal in human-robot interaction research is to build robots that are intuitive interaction partners for humans. Although some research does focus on building robots which look and behave exactly like a human, even simple toylike robots can be accepted as adequate and intuitive interaction partners. However, for complex interaction tasks, intelligent support, or cooperative behavior more advanced and "on board" solutions have to be developed, that still support natural interaction behavior between human and robot. This chapter will discuss some relevant research in the field of human-robot interaction which is fundamental for more complex but still intuitive interaction. The focus is to convey the complexity of research that is required and to point out different research areas which are relevant to achieve the goal of developing robots that can be natural interaction partners for humans.

Keywords: inherent safety, compliant actuator, embedded system, verification, FPGA, model checking, interface, gesture, speech, embedded brain reading, physiological computing, imitation learning, skill learning, transfer learning, active learning

1 Introduction

Today, personal computers can be found in almost every household. Prerequisite for their entry into everyday life was not only that they became smaller in size and affordable but that their user interfaces became more intuitive to enable easy interaction and usage. But how about robots? Having a robotic companion to clean the house or garden is a well-accepted wish of many of us. However, the idea to utilize robotic systems for tasks that involve direct and close interaction with humans does not always meet approval. An introduction of technical systems for close interaction does often go along with social and ethic discussions, similar to

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_8,

discussions about the permanent availability when using mobile communication devices. Such discussions are especially taking place with respect to the usage of robotic systems to take over tasks and roles which are socially explicitly reserved for humans, like the personal care of elderly people or the education of our children. Often, a highly debated question is whether robotic systems shall substitute for humans and human-human interaction and whether the utilization of robotic systems may lead to a social impoverishment. In these discussions a robotic system is seen as a technical system that takes over "jobs" which otherwise would be in the responsibility of other humans. Moreover, robotic systems are in these discussions either often seen as insufficient interaction partners that will not fulfill the demands of a human on an interaction partner or they are seen as intruders that may even prohibit a human to interact with other humans or to learn to interact appropriately. Both views are however far away from today's reality.

Robotic systems as they are applied today can carry out only a limited amount of jobs that would otherwise be carried out by humans. Such jobs, like working in an industrial environment on the production line and repeatedly performing the same task, are often jobs that require no interaction with humans and are willingly delegated to robotic systems. To introduce robotic systems into highly interactive and social environments is a future challenge that arises fast due to demographic changes. Besides above mentioned important ethic and social discussions, questions about the requirements for everyday and intuitive human-robot interaction have to be stated and answered. This chapter will address mainly two important questions:

1. How must a robotic system be designed to allow *safe* human-robot interaction?
2. How must a robotic system be designed to become an *accepted* interaction partner?

While this chapter cannot present the final answers to these questions it will, however, give an introduction to some relevant research topics. Moreover, it will hopefully show that a broad field of research areas must be addressed to enable safe and intuitive human-robot interaction and to design and build robotic systems that will be accepted as interaction partners. Whether such robotic systems may ever become social interaction partners cannot be answered here although the quality of achievable human-robot interaction is an important prerequisite for the discussion of the possible social role of robotic systems.

The chapter is structured as follows: In Section 2 relevant aspects for safe human-robot interaction such as inherent safety, concepts for required onboard sensor systems, and processing as well as challenges and opportunities, that come from the possibility to apply formal verification methods to proof safe human-robot interaction, are discussed. The focus of Section 3 is on aspects that are relevant for the design of robots which can be accepted as human interaction partner. Approaches such as intuitive interfaces, the usage of implicitly transferred information and the recognition of human intention as well as the role of

learning from humans are discussed. Finally, in Section 4 the chapter is briefly summarized.

2 Safe Human-Robot Interaction in Changing Environments

In current industrial applications an absolute separation of robots' and humans' workspace can be found in most cases. Robots work in cages while humans may only enter the robots' workspace when the robot is turned off. Such applications require no cooperative human-robot interaction and allow the application of conventional industrial robots. If close and collaborative interaction between robots and humans is required, as it is already the case for medical and service robotics, safety has to be considered. Safety can be assured by design and/or control as well as by verification. Robotic design and/or control is addressed by the mechatronic development of safe robotic manipulators and the extensions of their capabilities in sensing their environment and in detecting humans within their workspace. Verifying the proper functioning of such systems is challenging because it incorporates the *interaction* of a robotic system as a cyber physical system not only with its physical *environment* but also with a biological system, the *human*, which can both not fully be modeled. Before addressing this challenge in Section 2.3, a short overview is given about relevant research in safe robotic manipulators (see Section 2.1) and embedded sensor systems (see Section 2.2).

2.1 Inherent Safe Robotic Systems for Human-Centered Robotics

The performance of traditional industrial robots has usually been evaluated looking at their speed, precision, payload, and reachable workspace. When considering robots in close cooperation with humans, another aspect of performance needs to be considered: safety. In classical industrial robots, safety was achieved by enclosing the robot within some fences and avoiding sharing the workspace with humans. However, when exiting those controlled environments in which no interaction with humans takes place, the design of the robot and its cell must be reconsidered to ensure a safety level which enables human-robot interaction. The safety mechanisms to be used will determine the allowed degree of interaction with a human. For example, by introducing sensors and software safety mechanisms to monitor and control the manipulator and/or by covering the manipulator with soft materials to avoid abrasion injuries, a robot may be able to share human spaces without possibly injuring the human. In case of robots with large inertia (thus also weight), it might not be enough to cover a robot with compliant material to really avoid injuries to a cooperating human since the amount of covering that would be required would be too large. In the case of control software based on the monitoring of some sensory signals, questions arise on what happens in case of a software/electrical failure, or in the case of a very fast impact to which the sensors/software might not be able to react on time. Impacts are the most dangerous hazard when working with large inertia

robots at high speed. In case of lightweight robots (see next paragraph), the most dangerous situation is not the (free) impact per se, but the clamping between the robot structure or between robot and a wall [77].

There are two main ways of providing safety while sharing a workspace between robot and human: (1) avoiding the possibility of contact events between robot and human by both robot and workspace supervision or (2) enabling full human-robot interaction (contacts) by designing a new generation of force and power-limiting robots combining low weight and inertia with control algorithms or mechatronic designs that limit the maximum forces the robots can exert. The first option, workspace supervision, can be used for most of classical industrial robots. Basically, it requires a supervision system that monitors the robot's workspace and ensures that a minimum distance between worker and robot is maintained. This can be achieved by safety-rated sensors and software which supervises both robot speed and distance between robot and human, and either slows down or stops the robot in case that a minimum distance is violated. The second option, force-limiting robots, has the potential of enabling fully human-robot interaction by limiting robot's power and force by inherent design. Power and force limitation can be achieved by design of the manipulator or by using control software mechanisms. Namely, by using low-inertia manipulators, choosing suitable material for compliant covering and an appropriate geometry that avoids pinch points and sharp edges, or by adding control functions such as safe collision detection and safe supervision of the robot kinematics.

In recent years we have seen on the market a number of this new generation of industrial (lightweight) robotic manipulators that enable different degrees of interaction based on different control mechanisms and/or design criteria. Among the dual-arm robots which incorporate some of the above mentioned safety features are for example the ABB Yumi (available from Spring 2015)(Fig. 1(b)). The ABB Yumi is an inherently safe system by limiting via design the maximum forces it can exert. However, as a trade-off, that limitation also limits the maximum payload (likely to be around 500 g) which might be though enough for applications such as small parts assembly. Another recent system is the dual-arm Nextage-OPEN (previously known as HIRO) designed by Kawada Industries which can handle around 2 kg (including gripper). In this case, the maximum power of the motors is limited by design to 80W, a value which is based on the requirements mentioned on a previous version (thus not anymore valid) of the standard ISO 10218 to guarantee safety of an industrial robot. Another force limiting robot by design is the Baxter robot from Rethink Robotics (Fig. 1(c)) which uses a further development of the series elastic actuators [1]. The company Universal Robots is also developing force limiting robots (Fig. 1(d))which can carry 5 and 10 kg based on measuring motor currents to estimate joint torques (and a second redundant measurement of the torques using the position measurements at both sides of the gear). In this case, the maximum force the robot can exert is limited to 150N (as the 80W, a value of a previous version of the standard ISO 10218, not anymore valid). Finally, the latest version of the well-known KUKA lightweight robots, the KUKA iiwa (Fig. 1(a)), can lift 7 and

14 kg and makes use of the joint torque sensors to achieve compliant motion and thus safe human-robot interaction.

In summary, many approaches in manipulator design are possible to enable inherent safe human-robot interaction and to open a new field of cooperative interaction which is today only starting to be developed. Cooperative interaction between human and robot is highly relevant for future developments especially in industrial environments which require high flexibility and short down times. Inherent safety is the basis for human-robot interaction which can further be improved as will be discussed in the upcoming chapters.

2.2 Embedded Sensor Systems for Onboard Self and Environment Modeling

A robot that is equipped with a minimum of sensors that perceive contact forces, e.g., torque sensors in each of its joint, can sense direct interaction with the environment and the interacting human. The usage of tactile sensors in human robot interaction can therefore be regarded as safety-critical, as it prevents injuries and damage in the case a contact between the interaction partners cannot be avoided. The upcoming challenges coming alongside full body artificial skins with continuously increasing resolution and modalities are discussed presenting decentralized pre-processing as one solution towards tackling the problem of handling the increasing amount of data.

To enhance safety, the environment is often supervised by, e.g., safety-rated sensing capabilities. Keeping sensing capability in the environment is often sufficient if interaction takes place in static and supervised areas. For robotic systems that interact with humans in changing environments this approach is not applicable. This increase in requirements of the autonomy of robotic systems demands for additional sensing capabilities to cope with unforeseen events. To achieve this, the system has to be able to analyze the gathered information as well as to model its system state as well as its environment online. Increasing these capabilities in a robot is a challenge on integrating both sensors and the processing units for the required processing power.

For human-robot interaction it is further advantageous if the robot could be able to sense and model its environment in a similar fashion as a human can. This would allow easier interaction, communication and cooperation as well as easier adaptation to new environments since a common understanding about the environment could be generated and used to share and exchange knowledge.

Besides visual and auditory capabilities the human skin is a very important sensing system. Sensor cells that are integrated into the skin allow to measure the increment and decrement of pressure, stretching of the skin, acceleration of a pressure stimulus and local pressure and deformation. By these sensors, the human can feel its environment and detect changes in its environment and especially during manipulation in much higher detail and broadness than simple torque sensors could. Integrating broad tactile sensing capabilities into a robotic manipulator is challenging since sensors that sense contact with the environment

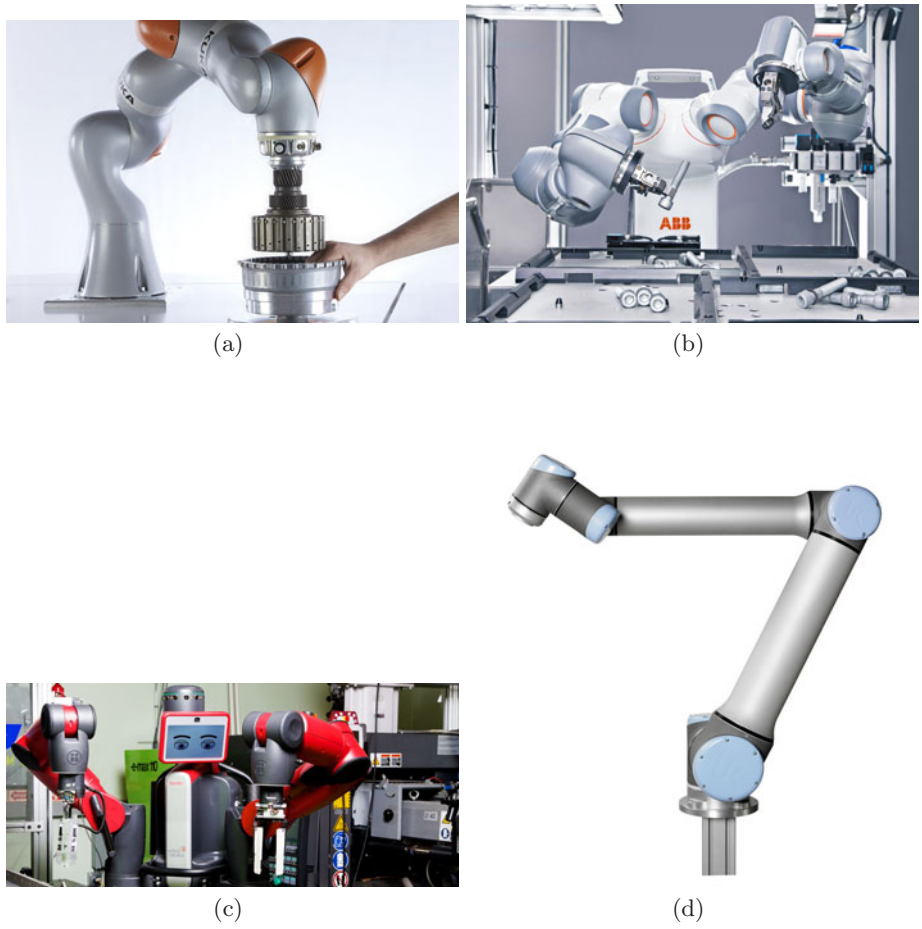


Fig. 1. *(b)* KUKA-iiwa (KUKA), *(d)* Yumi (ABB), *(a)* Baxter (Rethink Robotics), *(c)* UR10 (Universal Robots)

need to be placed at the robots surface, which in case of multi-modal sensing is a highly frequented area.

Nonetheless, several tactile sensing systems tackle the challenge of integrating several modalities at the contact area of the robot. The attempt of [3] is to mimic the capabilities of the human hand. Sensors for acquiring information about texture, geometric and thermal properties are integrated in finger-shape sensor modules. [2] and [4] focus more on the realization of sensor patches that can be combined to cover a complete robotic system with optimized communication and data acquisition.

The challenge of integrating such sensors into robotic end-effectors can be observed from several examples. The hand systems developed by DLR [6, 5] are highly integrated examples of robotic systems that strive to replicate the kinematic capabilities of the human hand as closely as possible. It can be seen that these systems are highly integrated without any space left relevant for tactile skin sensors and its processing. [7] show another example of a five-fingered hand with a tactile skin. Due to limited integration space, the tactile sensors have been placed on top of the kinematic structure of the end-effectors without further shielding and integration into the embedded processing units of the gripper. Commercially available systems show a similar design. The Schunk SDH Hand can be equipped with tactile sensors that thanks to their small dimensions can be integrated into the gripper structure, although the actuators for the direct drive of the joints share some of the integration space.

Regarding the development of the tactile sense in robotics towards highly autonomous systems, perceiving as many modalities of touch as possible is a key feature towards adding object properties like texture, geometry, and hardness during exploration tasks. [8] summarized the sensing modalities for a robotic system and the derived information to gather complete tactile information objects in the contact area. The tactile information flow described comprises of using force sensing arrays, measuring forces and torques as well as dynamic inputs together with thermal properties and joint actuator efforts. By combining these modalities, useful information about material hardness, texture properties, absolute incoming forces and much more can be derived. The presented tactile sensor modules are developments towards this goal, but either some of the information is missing or is derived by sensors originally perceiving different modalities. One example is measurement of overall incoming forces. One approach is to sum up the information from the sensor modality that is responsible of measuring geometric properties. Using this modality results in increased calibration effort, as every sensing element of the required force sensing array needs to be calibrated for measuring absolute forces. This approach is time-consuming, as these have to be recalibrated, or does not lead to optimal results. Hence, the goal is to use multi-modal information to allow to drive and adapt different kinds and levels of interaction based on sensor data that is recorded from embedded sensors and analysis.

The approach is thus to integrate separate force sensing modalities for measuring dynamic, static and geometric properties at the contact area. Besides the

aforementioned challenges of integrating the required sensors into the available integration space, processing the perceived information is another issue. For instance, the tactile sensing system presented in [9] consisting of more than 700 sensing elements generates - due to its camera-based acquisition system - more than 300 Megabytes per second of sensory information. Approaches that sample the sensor information in the end-effector and transmit the unprocessed data to a high-level processing unit cannot be taken anymore to tackle this amount of information. Therefore, approaches have to be developed that allow not only to integrate multimodal sensors into a robotic systems but to also integrate intelligent and space saving analysis systems which requires to develop new and decentralized analysis strategies.

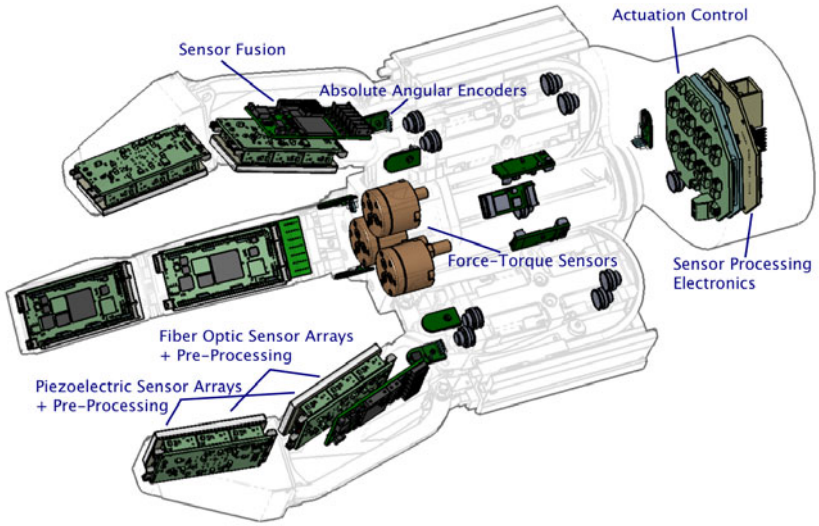


Fig. 2. Highly integrated gripper system with multi-modal tactile force sensing capabilities and embedded pre-processing.

An approach towards a robotic end-effector with a multi-modal tactile force sensing system with decentralized pre-processing integrated into the system (2) is presented in [10]. A three-fingered morphology using two opposable thumbs has been chosen as a good compromise between the amount of realizable grasp types using this setup and the mechanical complexity of the system (compare [11]). Several force sensing modalities have been integrated into the system. For measuring geometric properties, a fiber-optic measurement principle has been selected. By combining the setup of this sensor together with an array of piezoelectric material, both geometric properties and dynamic impacts can be measured by specialized modalities at the same contact area. The exterioceptive

sensing modalities are completed by three force-torque sensors, one is integrated in each finger base. Together with proprioceptive information like joint angles and applied actuation power measured from the actuation of the gripper, this set of sensors fulfills the tactile force sensing requirements of the described tactile information flow in [8]. Pre-processing of the sensor elements is realized as early as possible in the gripper system itself. Therefore, a combination of eleven field-programmable gate arrays (FPGAs), nine programmable system on chips (PSoCs) and one digital signal processor (DSP) is handling the embedded processing of the perceived information. This combination of specialized processing units that are distributed in the overall gripper system allows the generation of high-level information already in the end-effector thus enabling the implementation of local control loops like grasping an object or reacting to slippage of objects without involving high-level processing systems outside the end-effector. This approach enables fast reaction times to external stimuli and enables further reduction of communication between the processing system in the gripper and external processors. Extending this approach to more complex behaviors implemented locally leads for example to the possibility to let the end-effector control the manipulator arm where it is attached to, to follow a contour. The implementation of such a behavior could result in a shift of control paradigms where usually kinematic chains are controlled in a top-down manner by high-level processing units. The shift of processing load and the resulting geometric information of the followed contour delivered by the combination of the joint movements combined with the kinematics of the manipulator arm further leads to a reduction of data transferred to high-level processing units which is a key feature towards robotic systems with increasing complexity due to increasing autonomous capabilities.

By embedding sensing capability into robots, they can become independent of predefined and supervised interaction environments and thus potentially able to interact with humans in changing environments. To achieve such independency requires research not only in the field of sensor technology but also in the field of electronics. Following the definition of today's big data discussion which is based on the relationship of available processing power and the amount of data that needs to be processed, the embedded processing architectures of future robotic systems can be inspired by research from the field of big data handling. Embedded sensing capabilities cannot only be used to replace safety-rated sensing capabilities of a static interaction environment to enhance or even facilitate safe human-robot interaction. It does enable a robot to share a common understanding of the environment of the interacting human or to even analyze the human's behavior or intention as will be discussed in Section 3.

2.3 Formal Verification of Human-Robot Interaction

In robotic application fields where a robot takes over or substitutes for parts of the human's body function safety becomes most relevant. However, in such applications like robotic rehabilitation the functioning and control of a robot as a partly autonomous system often depends on many influencing aspects like the

context of interaction, the human's behavior, and the human's general or current capabilities. Quite often the robot must adapt to changing situations or changes in the human's capabilities. This is for example highly relevant for patients with disabilities. Here support might depend on the patient's capability to be able to, e.g., move a body part by herself or himself or not. The more directly a robotic system, like an orthosis or exoskeleton as a rehabilitation device, interferes with the human's behavior and the more its functionality depends on conditions which are determined by the interacting human and environment, the more complex it becomes to even describe the functionality or to guarantee proper and reasonable functioning. While the formal verification of autonomous robots for functional safety is already a complicated task [12], complex human-robot interaction can per se be seen as a challenging field for the implementation of formal descriptions and formal verification methods. To apply verification methods to applications with intense human-robot interaction is especially interesting since the implementation of complex interaction increases the vulnerability of such approaches for errors that might otherwise not easily be found. There are many reasons for hidden errors in complex interaction. A most important one arises from the human ability to automatically and unconsciously compensate for minor inaccuracies. Formal description and verification could support the detection of such errors. A first step is to assure correct functioning of complex systems with respect to specified application conditions. Even by developing a formal model for an approach which must not necessarily be yet a model that can directly be verified by standard verification methods can help to detect errors within specific applications which would otherwise (due to their low impact on overall functioning or due to given dependencies) not easily be found or hidden [13].

However, new approaches are required that allow to formally describe complex human-machine interaction, i.e., to model the human or human data like physiological data and the interacting cyber-physical system(s) as well as the applied human-machine interfaces (as part of the robot or as independent devices, see Section 3.1). To verify their correct functioning is a very challenging task since today's solutions to formally describe such complex systems and human-machine interaction are not suitable to be directly applied. A rigorous specification of all parts and interfaces might help to break the complex problem down, since verifying complex systems may fail due to their complexity itself as faulty dependencies between or interplay of processes are easily overseen when addressing the system as a whole. Thus, a possible solution is to focus on the *basic functions and processes* of the system or interaction and to verify them in themselves, before addressing the formal verification of their interplay in a second iteration. Here, techniques such as model checking [14] seem to be suitable to be applied. However, further development and extensive research in this field is required to solve this challenge.

3 Robots as Accepted Interaction Partner

Besides designing and building robotic systems that allow safe interaction, robots must be provided with further qualities that are common for humans. Highly relevant for satisfying human-human interaction is that the interaction is intuitive, i.e, that none of the partners has to constantly think of what and how to say or to do something in a certain way and that common interaction pathways can be used (see Section 3.1). This might not always be the case. For example, when people of different cultures interact with each other, mimic, body language and of course language might differ and prohibit intuitive interaction. Moreover, for easy interaction it is relevant that the intention of the interaction partner can be inferred and addressed (see Section 3.2). This is even more relevant for cooperative interaction where it is relevant to infer what the cooperating partner wants to do next to solve a task together. In the long run a robotic system will not be accepted as natural, human-like interaction partner in case it does not learn by itself and during interaction how to improve interaction. A good strategy is to directly learn from the interaction partner, i.e., the human. Therefore, the approach of imitation learning for improving human-robot interaction will be discussed at the end in Section 3.3.

3.1 Intuitive Interfaces and their Application Focus

To interact with technical systems interfaces are required. Such interfaces can be stand-alone interfaces, they can be embedded into the interaction environment, worn by the human interaction partner or can be part of the cyber-physical system, e.g., the robot. Most interfaces serve purely as input device, i.e., as possibility to transfer commands to a technical system. Such interfaces can be quite simple, like a mouse or keyboard. Other systems are more than input devices. For example, an Automatic Virtual Environment (CAVE) is a complex interaction environment. It is an immersive virtual reality facility that allows interaction within spatially engaging environments but also with robotic systems or their simulations (see Fig. 3 for an example). While a mouse or keyboard belongs to the type of interfaces that allow a human to translate commands into input that a computer or other technical systems can understand, a CAVE system allows the human to interact more intuitively and to receive complex feedback.

Interfaces that enable intuitive interaction become more and more relevant. They often allow to make use of communication pathways that are used for human to human interaction and communication. For example, speech is a highly efficient communication channel for humans. Speech interfaces were developed early in the 20th century. Joseph Weizenbaum ($1923-2008$) developed the computer program ELIZA [15]. He discovered that this simple natural language processing system, which was able to carry out humanlike conversations but was actually not able to understand the meaning of human language, could be applied to substitute a human partner. This secondary finding of his research was of high interest for psychiatrists, who even suggested to apply the program

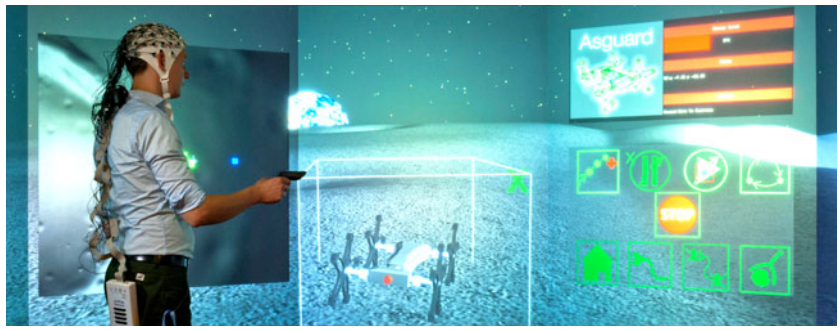


Fig. 3. Immersive virtual multi-robot control using a CAVE supported by embedded brain reading.

ELIZA as an acceptable substitute for human therapy [16]. The given example shows that interfaces for speech generation and understanding that may even imitate the learning of human's speech [15, 17–20] has a high relevance for interaction and that robots equipped with such an interface can easily be accepted as interaction partner.

Another communication channel that is increasingly used for interaction with robots is body language. Especially gesture is used for *explicit* interaction, and allows the intuitive control of a robot [21–23]. Gestures can be used to transfer complex commands by often simple movements of the limbs, usually the upper limb of the human. They are to some degree restricted, since they can only be used for predefined commands which have to be learned by the human. However, by making use of a hierarchical gesture control concept which utilizes different levels of input states and differentiates between command modes and options (see Fig. 4), it is not only possible to greatly ease the process of learning those gestures and their meaning for humans; the same simple gestures can furthermore be used several times while communicating different commands to the robot.

The hierarchical gesture control concept is achieved mainly by separating commands into modes and options (see Fig. 4), i.e., the user would first issue a gesture to select a desired operation mode, e.g. exploration and then is able to set options, e.g. the desired exploration mode (if not default behavior is wanted) for the selected mode by issuing additional gestures. This way, even a reduced set of available gestures can be used to issue a variety of different commands, i.e., by using certain gestures for mode selection and the same gestures for option selection as well.

Furthermore, by using the concept of starting an interaction with a certain simple gesture command which turns the robot from its previous, e.g., idle to an "attention" state (see Fig. 4), it can be clearly defined whether the human does wish to communicate with the robot or not. This allows the human to otherwise behave freely. He or she will not have to avoid movements of the, e.g., arms that would otherwise constitute a gesture. For easy and fluent interaction

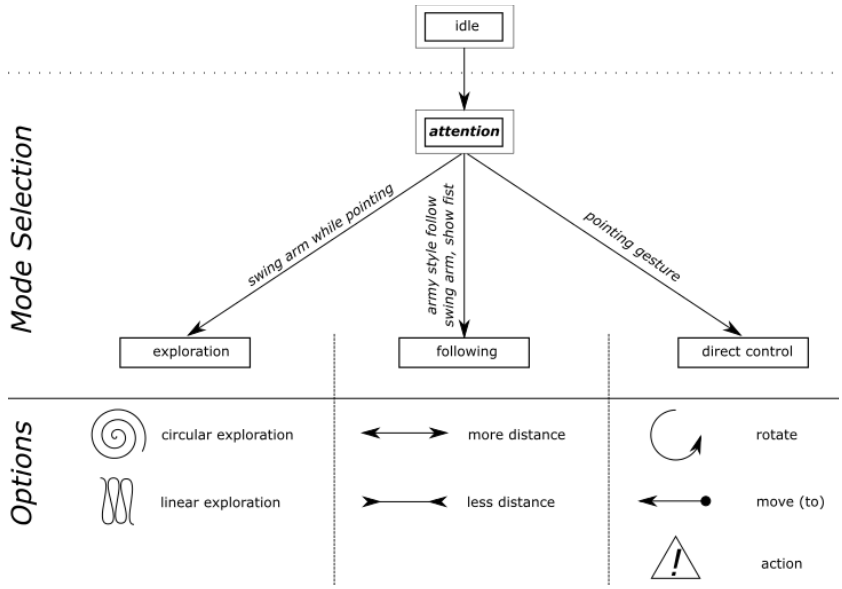


Fig. 4. Simplified sketch of an hierarchical gesture control concept.

it is also comfortable to not to keep the limb in a certain position to drive a responding behavior of the robot. It is a better option to generate a gesture which starts a response of the robot that is performed until a new command is given. Moreover, a very simple and intuitive gesture can be defined to stop any behavior of the robot. Such a gesture makes sense especially in situations where the human might not know what to do next and therefore needs some time to think the situation over. In such situations it might be best that the robot is just doing nothing.

Special applications for interfacing with robots can be found in the field of rehabilitation. Here, a technical system or a robot must support a human by restituting or substituting her or his capabilities. Early interfaces for reestablishing communication or motor function were brain computer interfaces (BCIs) that make use of brain activity [24–27]. Such interfaces like the so-called P300-speller [24] are an alternative to physical interaction, since they allow subjects, including paralyzed Locked-In syndrome patients [28], to communicate words, letters, and simple commands. They even reenable physical interaction by controlling a cyber-physical system, like a prothesis. Recently the usage of orthoses or even exoskeletons in rehabilitation and assistance in daily life has become of high relevance when combined with BCIs. Orthoses or exoskeletons which were originally developed for military purposes to expand human capabilities, i.e., to increase the force a human can apply [29], can combined with a BCI be used to reenable a human to move her or his arms [30–32] or legs [33, 34] again. However, exoskeleton technology cannot only be used for the support of disabled

persons but also for industrial applications. Exoskeletons can be a applied as a very intuitive interface that allow to control complex robots in, e.g., tele-robotics applications [35, 36].

As it was discussed for sensing capabilities (see Section 2.2), for the development of interface technology it becomes more and more relevant that human-robot interaction is not only intuitive but supported in any environment. However, the approach of equipping specific environments with intelligent interfaces, as it is for example done for the support of people in their flats by making use of approaches of assistive daily living [37], is limited to human-controlled environments. In contrast, the approach of equipping humans with *wearable interfaces* (e.g., the *myo*[1] wireless muscular activity sensor for issuing gesture commands) can be applied to in principle any environment. In future, it will however become more and more relevant for human-robot interaction to equip robots with embedded interfaces to enable for example gesture recognition without the requirements that either the interacting human or the environment is equipped with sensing technology that interprets her or his posture. Approaches which will be discussed in Section 3.3 can be applied to read the human's overt behavior. However, before explaining the relevance of behavior recognition and learning of motor behavior from humans for successful human-machine interaction the relevance of intention recognition will be discussed next.

3.2 The Relevance of Implicit Information and Human Intention Recognition

To build robots that are accepted by humans as interaction partner, it is not sufficient that they can be explicitly controlled via intuitive interfaces. The more and better a robot is cooperating with a human the more the interacting human will expect humanlike behavior of the robot. To achieve this, it is relevant that not only explicit information (i.e., a control command) is transferred between human and robot, but that *implicit* information is transferred as well. Implicit information is transferred between humans rather passively. It tells the interaction partner something about, e.g., the emotional state, involvement, or mental load. Studies showed that it is highly relevant that explicit and implicit information do match. Is for example gesture used to implicitly support explicit information transferred by speech it was found that a human interaction partner will not trust information given explicitly by speech in case that the implicitly transferred information does not match [38]. On the other hand, a human interaction partner will more likely accept a system as interaction partner in case that explicitly and implicitly transferred information does match. Therefore, to improve human-robot interaction a robot could be enabled to transmit implicit information by, e.g., gesture or facial expression, like a human [39–41]. In turn the robot or interface must be enabled to analyze the human by decoding implicitly sent information.

[1] https://www.thalmic.com/en/myo/

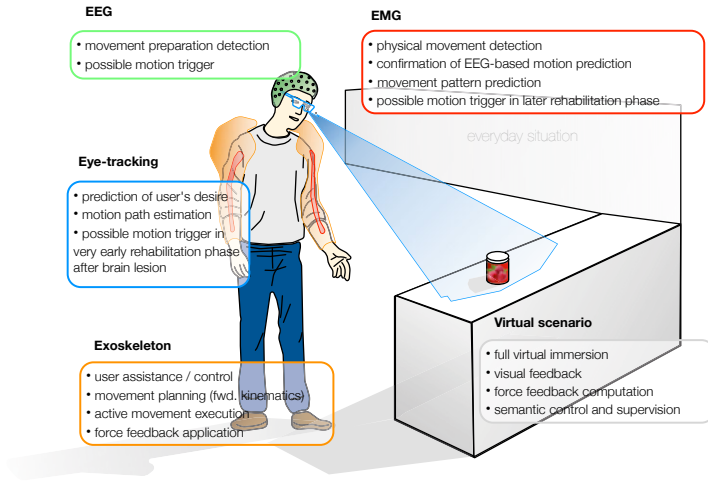


Fig. 5. Multimodal data analysis for embedded brain reading in robotic rehabilitation using exoskeleton technology for active support.

As mentioned afore two of the first fields of robotic applications in which robots directly cooperate with humans were logistics and surgery. Especially the development of surgical assistive devices is already very advanced. They allow to support humans by, e.g., enhancing the speed of specific tasks performed during surgery, like tying a knot [42], by taking over complicated tasks like catheter guidance [43], or by executing "superhuman" performance by adopting the best behavior from several surgeons [44]. However, studies did also show that the prediction of the surgeon's intention is essential to further improve a system's response [45]. This is especially relevant for the support of complicated surgical procedures. Is a robot able to infer the intention of a human it can chose an appropiate behavior that matches the situation and intention of the human interaction partner. The ability to infer a suitable behavior from even incomplete human instructions is discussed to be one prerequisite for future robotic agents [46]. Besides analyzing (incomplete) explicit information with respect to the possible intention of a human interaction partner, intentions can also be inferred from implicitly transferred information (e.g., from gesture or facial expression) or from physiological data. Physiological measures like electrodermal activity or galvanic skin response, blood pressure, respiratory patterns, the electrooculogram (EOG) and the measurement of pupillary responses, the electromyogram (EMG), the electrocardiogram (ECG) and the electroencephalogram (EEG) can be used to passively gain implicit information about the human.

While behavior for explicit interaction and behavior like gestures that transfer implicit information, can be counted as overt data which can be observed without extra devices for recording and analysis physiological measures are

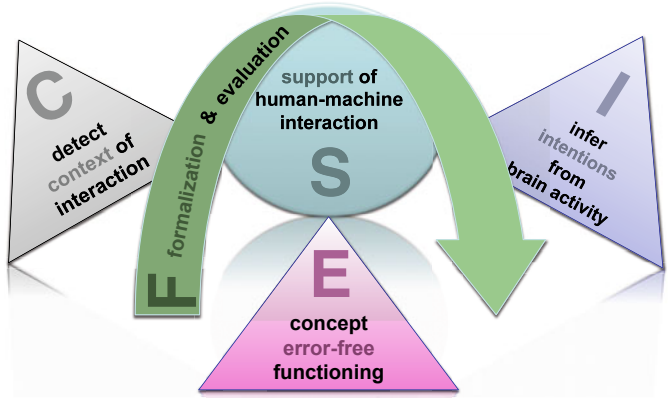


Fig. 6. Concept for embedded brain reading for the support of human-machine interaction based on the context of interaction and inferred intentions.

covert data that require specific recording devices and analysis to make them visible. At the first moment this sounds like a restriction for their usage. However, covert data has the advantage to continuously be available and to be highly reliable, i.e., even more reliable than information gained from overt behavioral data as shown in [47] where the reliability of overt behavioral data was compared with covert EEG data as source of information. Another advantage of using covert physiological data is that for its usage a systems' user does not have to perform any additional task nor a higher mental or cognitive engagement is required, since it can passively be acquired.

Physiological data is for example used for biocybernetic adaptation. Biocybernetic adaptation is an application of physiological computing [48]. It is used to enhance or enrich interaction by transforming physiological signals into real-time computer input to change the functionality of a system regarding, e.g., fatigue or frustration levels of a user. It can enable greater control over complex systems [49]. Such implicit decoding of physiological data does not increase the workload but can subjectively reduce it, while the humans' performance [50] and task engagement [51] are increased.

A special physiological measure is the human brain activity since the brain is controlling most of the human's behavior. Thus, human-machine interfaces that make use of brain activity were always of high interest (see for example its usage for BCI in Section 3.1). Since the brain is not only controlling but also planning behavior, analyzing brain activity is in principle an approach to uncover

preconscious intention like early, preconscious movement intention [52, 53]. The analysis of brain activity can therefore serve as a window into the human mind or brain [54]. There are different measures for brain activity, the most often used one is the electroencephalogram as a direct measure of the brain's electrical activity. The EEG is comparatively easy and cheap to be recorded. However, the human EEG is a very complex signal that consist of overlaying activity and is easily disturbed by external noise and noise generated by the human body. Thus, even by applying advanced signal processing and classification methods (as they are available by open source software frameworks like pySPACE [55]) the outcome of EEG analysis cannot be 100% correct. Therefore, it is very important to supervise the usage of brain activity as source of covert information about the human. By embedded brain reading (eBR) [56], it is possible to use brain activity to either adapt interaction, e.g., with respect to workload in a complex robot control scenario (see Fig. 3) or to even drive interaction, e.g., for robotic rehabilitation purposes (see Fig. 5), by passively reading brain activity while controlling for correct interpretation of it. To supervise correctness is enabled by using brain activity in the context of interaction and by inferring upcoming interaction behavior which can then be detected and used as a control signal or by using other modalities like technical data of the interacting system, behavioral data, or other physiological data [57] of the interacting human to control for the correctness of the predicted brain state. The later, i.e., the usage of multimodal data, e.g., EEG, EMG, eye-tracking and exoskeleton data (see Fig. 5), is most relevant for the application in robotic rehabilitation where a robotic system, like an exoskeleton, is used to substitute human behavior based on inferred intentions.

For future research especially the field of robotic rehabilitation will become a target for new approaches of verification methods (see Section 2.3) to guarantee safety and correct behavior of such systems. Although there are first attempts to apply formal verification approaches to human-robot interaction [13], the research in this field has just started. However, to apply such approaches in complex human-robot interaction seems to be a promising approach and is therefore part of eBR for human-robot interaction (see Fig. 6).

3.3 Improved Interaction based on Learning from Humans

Robots that are situated in complex and unknown environments cannot have a predefined behavior for every possible situation. Thus, robots need an efficient and versatile way of acquiring new behavior that is appropriate for new and unforeseen situations. Machine learning (ML) is a popular approach for this as it is data-driven and thus allows often to cope even with unforeseen situations. ML approaches enable the system to learn models of itself and its environment and to adapt them if required. In the ideal case, learning of knowledge and utilization of the learned knowledge are not conducted strictly sequential, which would require separate training and deployment phases, but are integrated in a continuous, life-long learning approach [59, 60]. ML approaches developed for learning in autonomous robots allow for instance to acquire human-like behavior [39, 61,

62] and are in principle also suitable for improving interaction between robots and humans and to cope with inter- and intra-individual variances in human behavior. Different kinds of ML can be applied in robot learning, for example *imitation learning* [63–65], which allows learning behavior based on human demonstrations, *reinforcement learning* [58, 75], which allows learning based on trial-and-error, and *transfer learning* [69, 70], which aims at reusing and adapting behaviors that have already been acquired for similar situations. In the following, we discuss imitation learning and the required techniques for recording and analysis of human behavior; these methods fit naturally in situations where human-robot interaction is key.

To learn from humans and to improve interaction behavior requires not only to analyze and model the environment; the robot must also gather data from a human demonstrator and be able to analyze and model the human's behavior. For instance, human demonstrations of complex behavior can be broken down into simpler behavioral building blocks which are easier to learn by imitation and more reusable. For this decomposition of behavior, unsupervised machine learning methods for behavior segmentation can be applied [74]. Different techniques can be used for data acquisition in the context of human demonstrations, e.g., motion capturing, which potentially pose their own challenges within the context of imitation learning. For example, while motion capturing is often appropriate for recording a human's whole body posture, the recording of complex, detailed manipulations, like grasping an object, introduces new issues: for instance, the manipulated object may be concealed to the system by the manipulator's hands. To cope with those issues, motion capturing has to be extended by other techniques, specifically fitting for the issues that arise from the task to *record human behavior* instead of plain motion. For example, Fig. 7 shows a setup used for such recordings: the upper image shows an environment for human behavior recording, the lower two images display a hybrid capture technique in which motion capture and visual tracking approaches are combined, i.e., to cope with the above mentioned problem of concealed objects.

Moreover, in order to apply ML approaches for improving interaction, the robot must be able to recognize by itself that learning is required. To enable this, it must "understand" that a specific behavior of a human is relevant for interaction. Furthermore, the robot must infer what kind of behavior would be appropriate to solve a collaborative task and whether it has already learned a behavior that can be transferred as a solution to the unexpected and unknown situation. Thus a robot must be able to store and reuse as well as combine learned behaviors, which suggests using a hierarchical representation of behavior and an explicit mapping from situation to behavior. Recent advances in reinforcement learning in the area of contextual policy search are promising candidates in this direction [75, 76]. Moreover, a robot must be able to autonomously identify situations in which it requires further data to acquire a reliable behavior, either in the form of human demonstrations or by further trial-and-error. Active learning [71] is a promising approach for this, which has recently shown encouraging results when combined with contextual policy search [72, 73].

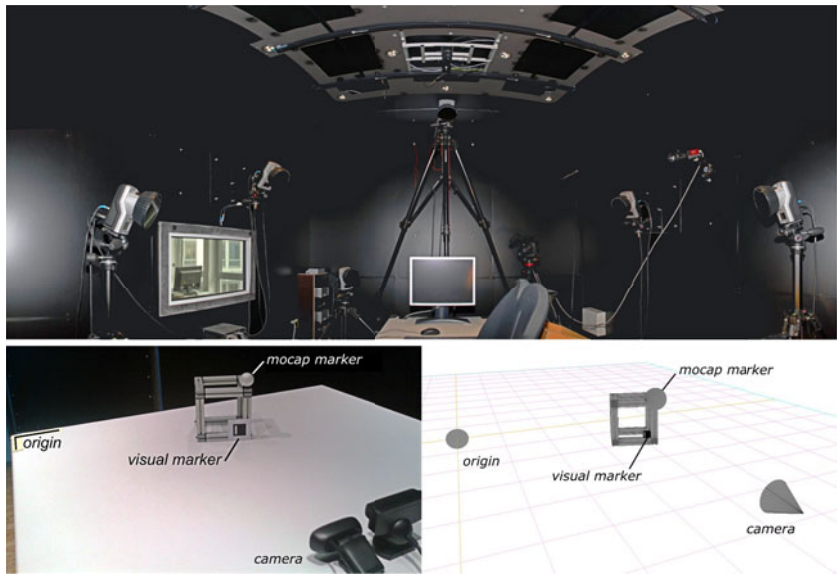


Fig. 7. Motion capture setup for recording human behaviors.

While offline analysis and learning is suitable for skill acquisition from human demonstrations [61, 63–65], life-long learning and learning during an ongoing interaction require the application of online learning methods [66–68]. This also implies that the analysis of the human behavior must also be performed online, which requires that either the environment, the human interaction partner, or the robot are equipped with hardware and software solutions for behavior analysis and ideally also with solutions for intention recognition (see Section 3.2). The preferred approach would be to equip the robot with adequate hardware and software, since by relying on the usage of systems that are installed in the environment like motion tracking systems (see Fig. 7), interaction would again be limited to technically equipped environments. To equip the interacting human with respective hardware is also not desirable since only humans that are wearing the respective devices could interact with the robot. Thus, equipping the robot with the required hardware and analysis devices would in principle be the best solution but does enhance the demands on embedded solutions as discussed in Section 2.2. Therefore, to develop online learning methods for the improvement of human-robot interaction and for learning appropriate interaction behavior, two approaches might at first be chosen or combined: (1) the usage of specific tracking devices in the environment, like environments equipped with motion tracking and visual tracking devices that allow the easy tracking of human and manipulated object (see Fig. 7) and (2) the usage of specific wearable devices that record changes in human posture, like a motion capturing suit (e.g., like

synertial[2] motion capturing suits) or an exoskeleton which cannot only be used to drive movements of the human (as sketched in Fig. 5) but can also be used to passively record posture and changes of it.

4 Summary and Conclusion

To enable intuitive, adaptive human-robot interaction in unknown environments with robotic systems that a human can accept as interaction partner, many aspects in different fields of research have to by considered like: (1) intrinsic safety to avoid any kind of injury of interacting persons by the robot, (2) embedded sensor and analysis capabilities to enable the robot to understand its environment and the interacting human to learn from her or him and to optimize interaction behavior, (3) new approaches that allow to verify the correct functioning of the robot as well as the proper interaction to increase safety especially in domains of human-robot interaction where the correct functioning is highly relevant as in rehabilitation robotics, (4) intuitive interfaces that can easily be used for implicit and explicit interaction, (5) approaches that are able to infer the intentions of an interacting human by the analysis of overt behavioral as well as cover physiological data to predict upcoming interaction behavior that can jointly be performed, (6) behavior analysis and learning methods that enable a robot to identify relevant situations and behavior, to learn from human and to improve behavior online.

While this list does not claim to be complete, it should allow to infer that human-robot interaction cannot be solved with research in a single research domain instead it does require the combination of different research and research solutions. This inherent complexity makes human-robot interaction a very challenging but also interesting field of research that does not only have to consider the interaction of cyber-physical systems with a real and complex physical environment. It also has to consider the interaction of a cyber-physical system with a biological system, the human. Hence, in an even broader view also psychological aspects have to be investigated and ethic questions have to be answered.

This chapter might create the impression that robots as they exist today are still far away from copying humans as interaction partners. However, many challenges are understood and already under research to generate solutions that in their combination can lead to robots which will in future be accepted by humans as humanlike interaction partner.

References

1. Pratt, G. A. and Williamson, M. M. (1995). Series elastic actuators. Proceedings. 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95, pages 399-406.
2. Mittendorfer, P. and Cheng, G. (2011). Humanoid multimodal tactile-sensing modules. Robotics, IEEE Transactions on, 110.

[2] http://www.synertial.com/

3. Wettels, N., Fishel, J., and Loeb, G. (2014). Multimodal Tactile Sensor. The Human Hand as an Inspiration for Robot Hand Development, Springer Tracts in Advanced Robotics (STAR) Series, (0912260), 120.
4. Maiolino, P., Maggiali, M., Cannata, G., Metta, G., and Natale, L. (2013). A Flexible and Robust Large Scale Capacitive Tactile System for Robots, IEEE Sensors Journal, vol. 13, no. 10, pp. 3910-3917.
5. Grebenstein, M., Albu-Schaffer, A., Bahls, Thomas, Chalon, M., Eiberger, O., Friedl, W., Gruber, R., Haddadin, S., Hagn, U., Haslinger, R., Hoppner, H., Jorg, S., Nickl, M., Nothhelfer, A., Petit, F., Reill, J., Seitz, N., Wimbock, T., Wolf, S., Wusthoff, T., and Hirzinger, G. (2011). The DLR hand arm system. Robotics and Automation (ICRA), 2011, 31753182.
6. Liu, H., Wu, K., Meusel, P., Seitz, N., Hirzinger, G., Jin, M. H., and Chen, Z. P. (2008). Multisensory five-finger dexterous hand: The DLR/HIT Hand II. 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, 36923697. doi:10.1109/IROS.2008.4650624.
7. Kawasaki, H., Komatsu, T., and Uchiyama, K. (2002). Dexterous anthropomorphic robot hand with distributed tactile sensor: Gifu hand II. IEEE/ASME Transactions on Mechatronics, 7(3), 296303. doi:10.1109/TMECH.2002.802720.
8. Cutkosky, M. R., Howe, R. D., and Provancher, W. R. (2007). Handbook of robotics, Chapter 19, Force and tactile sensors. Sensors (Peterborough, NH).
9. Kampmann, P. and Kirchner, F. (2012). A Tactile Sensing System for Underwater Manipulation. Proceedings of the workshop on: Advances in Tactile Sensing and Touch based Human-Robot Interaction to be held in conjunction with the 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI 2012), Boston, Massachusetts, USA, o.A., 3/2012.
10. Kampmann, P. and Kirchner, F. (2014). Towards a fine manipulation system with tactile feedback for deep-sea environments. Robotics and Autonomous Systems.
11. Kapandji, I., Tubiana, R., and Honore, L. (2007). The Physiology of the Joints: The upper limb, The Physiology of the Joints, Churchill Livingstone.
12. Täubig, H., Frese, U., Hertzberg, C., Lth, C., Mohr, S., Vorobev, E., and Walter, D. (2012). Guaranteeing Functional Safety: Design for Provability and Computer-Aided Verification. In Autonomous Robots, 32 (3), pp. 303331
13. Kirchner, E. A. and Drechsler, R. (2013). A Formal Model for Embedded Brain Reading. Industrial Robot: An International Journal, 40(6):530–540.
14. Clarke, Jr., E. M., Grumberg, O., and Peled, D. A. (1999). Model Checking. MIT Press.
15. Weizenbaum, J. (1966). Eliza - a computer program for the study of natural language communication between man and machine. Communications of the ACM, 9(1):36–45.
16. Weizenbaum, J. (1976). Computer Power and Human Reason: From Judgment to Calculation. W. H. Freeman & Co.: New York, NY, USA.
17. Wahlster, W. (2000). Mobile Speech-to-Speech Translation of Spontaneous Dialogs: An Overview of the Final Verbmobil System. In Wahlster, W., editor, Verbmobil: Foundations of Speech-to-Speech Translation., pages 3–21. Springer: Berlin, Heidelberg.
18. Nöth, E., Batliner, A., Kieling, A., Kompe, R., and Niemann, H. (2000). Verbmobil: the use of prosody in the linguistic components of a speech understanding system. IEEE Transactions on Speech and Audio Processing, 8(5):519–532.
19. Herzog, G. and Wazinski, P. (1994). Visual translator: Linking perceptions and natural language descriptions. Artificial Intelligence Review, 8(2-3):175–187.

20. Dindo, H. and Zambuto, D. (2010). A probabilistic approach to learning a visually grounded language model through human-robot interaction. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010, pages 790–796.
21. Van den Bergh, M., Carton, D., de Nijs, R., Mitsou, N., Landsiedel, C., Kuhnlenz, K., Wollherr, D., Van Gool, L. J., and Buss, M. (2011). Real-time 3D hand gesture interaction with a robot for understanding directions from humans. In RO-MAN, 2011 IEEE, pages 357–362.
22. Kim, D., Lee, J., Yoon, H.-S., Kim, J., and Sohn, J. (2013). Vision-based arm gesture recognition for a long-range human-robot interaction. The Journal of Supercomputing, 65(1):336–352.
23. Ma, B., Xu, W., and Wang, S. (2013). A robot control system based on gesture recognition using kinect. TELKOMNIKA Indonesian Journal of Electrical Engineering, 11(5):2605–2611.
24. Farwell, L. and Donchin, E. (1988). Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. Electroencephalography and Clinical Neurophysiology, 70(6):510–523.
25. Wolpaw, J. R., Birbaumer, N., McFarland, D. J., Pfurtscheller, G., and Vaughan, T. M. (2002). Brain-computer interfaces for communication and control. Clinical Neuro- physiology, 113(6):767–791.
26. Guger, C., Harkam, W., Hertnaes, C., and Pfurtscheller, G. (1999). Prosthetic control by an EEG-based brain-computer interface (BCI). Proceedings of the 5th European Conference for the Advancement of Assistive Technology (AAATE 5th).
27. Pfurtscheller, G. (2000) Brain oscillations control hand orthosis in a tetraplegic. Neuroscience Letters, 292(3):211–214.
28. Kübler, A., Kotchoubey, B., Kaiser, J., Wolpaw, J., and Birbaumer, N. (2001). Brain-computer communication: unlocking the locked in. Psychological Bulletin, 127(3):358–375.
29. Karlin, S. (2011). Raiding iron mans closet. IEEE Spectrum, 48(8):25–25.
30. Nef, T., Colombo, G., and Riener, R. (2005). Armin. Roboter für die Bewegungstherapie der oberen Extremität. Automatisierungstechnik, 53(12):597–606.
31. Mihelj, M., Nef, T., and Riener, R. (2007). ARMin II - 7 DoF rehabilitation robot: mechanics and kinematics. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 4120–4125.
32. Housman, S. J., Kelly, L., Scott, M., and Reinkensmeyer, D. J. (2009). A Randomized Controlled Trial of Gravity-Supported, Computer-Enhanced Arm Exercise for Individuals With Severe Hemiparesis. Neurorehabilitation and Neural Repair, 23:505–514.
33. Suzuki, K., Mito, G., Kawamoto, H., Hasegawa, Y., and Sankai, Y. (2007). Intention- based walking support for paraplegia patients with Robot Suit HAL. Advanced Robotics, 21(12):1441–1469.
34. Zoss, A., Kazerooni, H., and Chu, A. (2006). Biomechanical design of the Berkeley lower extremity exoskeleton (BLEEX). IEEE/ASME Transactions on Mechatronics, 11(2):128–138.
35. Folgheraiter, M., Bongardt, B., Albiez, J., and Kirchner, F. (2008). A bio-inspired haptic interface for tele-robotics applications. In IEEE International Conference on Robotics and Biomemetics (ROBIO 2008), pages 560–565, Bangkok.
36. Folgheraiter, M., Kirchner, E. A., Seeland, A., Kim, S. K., Jordan, M., Wohrle, H., Bongardt, B., Schmidt, S., Albiez, J., and Kirchner, F. (2011). A multimodal

brain-arm interface for operation of complex robotic systems and upper limb motor recovery. In Vieira, P., Fred, A., Filipe, J., and Gamboa, H., editors, Proceedings of the 4th International Conference on Biomedical Electronics and Devices (BIODEVICES- 11), pages 150–162, Rome. SciTePress.

37. Autexier, S., Hutter, D., and Stahl, C. (2013). In: Juan Carlos Augusto; Reiner Wichert (Hrsg.). Proceedings of the Fourth International Joint Conference on Ambient Intelligence. International Joint Conference on Ambient Intelligence (Aml-2013), December 3-5, Dublin, Ireland, Springer-Verlag, CCIS.
38. Bergmann, K., Kahl, S., and Kopp, S. (2013). Modeling the semantic coordination of speech and gesture under cognitive and linguistic constraints. In Aylett, R., Krenn, B., Pelachaud, C., and Shimodaira, H., editors, Intelligent Virtual Agents, volume 8108 of Lecture Notes in Computer Science, pages 203–216. Springer: Berlin, Heidelberg.
39. Sadeghipour, A. and Kopp, S. (2011). Embodied gesture processing: Motor-based integration of perception and action in social artificial agents. Cognitive Computation, 3(3):419–435.
40. Wimmer, M., MacDonald, B. A., Jayamuni, D., and Yadav, A. (2008). Facial expression recognition for human-robot interaction - a prototype. In Sommer, G. and Klette, R., editors, RobVis, volume 4931 of Lecture Notes in Computer Science, pages 139–152. Springer.
41. Giorgana, G. and Ploeger, P. G. (2011). Facial expression recognition for domestic service robots. In Röfer, T., Mayer, N. M., Savage, J., and Saranli, U., editors, RoboCup, volume 7416 of Lecture Notes in Computer Science, pages 353–364. Springer.
42. Mayer, H., Gomez, F., Wierstra, D., Nagy, I., Knoll, A., and Schmidhuber, J. (2006). A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pages 543–548.
43. Riga, C., Bicknell, C., Cheshire, N., and Hamady, M. (2009). Initial clinical application of a robotically steerable catheter system in endovascular aneurysm repair. Journal of Endovascular Therapy, 16(2):149–153.
44. Van den Berg, J., Miller, S., Duckworth, D., Hu, H., Wan, A., Fu, X.-Y., Goldberg, K., and Abbeel, P. (2010). Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations. In IEEE International Conference on Robotics and Automation (ICRA), 2010, pages 2074–2081.
45. Weede, O., Monnich, H., Muller, B., and Worn, H. (2011). An intelligent and autonomous endoscopic guidance system for minimally invasive surgery. In IEEE International Conference on Robotics and Automation (ICRA), 2011, pages 5762–5768.
46. Tenorth, M. and Beetz, M. (2013). KnowRob–A Knowledge Processing Infrastructure for Cognition-enabled Robots. Part 1: The KnowRob System. International Journal of Robotics Research (IJRR), 32(5):566–590.
47. Gerson, A. D., Parra, L. C. and Sajda, P. (2006). Cortically-coupled computer vision for rapid image search. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 2(14):174–179.
48. Allanson, J. and Fairclough, S. (2004). A research agenda for physiological computing. Interacting with Computers, 16(5):857–878.
49. Woods, D. D. (1996). Decomposing Automation: Apparent Simplicity, Real Complexity, chapter 1, pages 3–17. CRC.

50. Prinzel, L. J., Freeman, F. G., Scerbo, M. W., Mikulka, P. J., and Pope, A. T. (2000). A closed-loop system for examining psychophysiological measures for adaptive task allocation. The International Journal of Aviation Psychology, 10(4):393–410.
51. Freeman, F., Mikulka, P., Prinzel, L., and Scerbo, M. (1999) Evaluation of an adaptive automation system using three EEG indices with a visual tracking task. Biological Psychology, 50(1):61–76.
52. Libet, B., Gleason, C. A., Wright, E. W., and Pearl, D. K. (1983). Time of conscious intention to act in relation to onset of cerebral activity (readiness-potential). The unconscious initiation of a freely voluntary act. Brain, 106(Pt 3):623–642.
53. Shibasaki, H. and Hallett, M. (2006). What is the Bereitschaftspotential? Clinical Neurophysiology, 117(11):2341–2356.
54. Coles, M. (1989). Modern Mind-Brain Reading: Psychophysiology, Physiology, and Cognition. Psychophysiology, 26(3):251–269.
55. Krell, M. M., Straube, S., Seeland, A., Wohrle, H., Teiwes, J., Metzen, J. H., Kirchner, E. A., and Kirchner, F. (2013). pySPACE - a signal processing and classification environment in Python. Frontiers in Neuroinformatics, 7(40).
56. Kirchner, E. A. (2014). Embedded Brain Reading, University of Bremen, Bremen, Germany, `http://nbn-resolving.de/urn:nbn:de:gbv:46-00103734-14`.
57. Kirchner, E. A., Tabie, M., and Seeland, A. (2014). Multimodal movement prediction - towards an individual assistance of patients. PLoS ONE, 9(1):e85060.
58. Kober, J. and Peters, J. (2012). Reinforcement learning in robotics: A survey. In Wier- ing, M. and Otterlo, M., editors, Reinforcement Learning, volume 12 of Adaptation, Learning, and Optimization, pages 579610. Springer: Berlin, Heidelberg.
59. Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. In: L. Steels (ed.) The Biology and Technology of Intelligent Autonomous Agents, 144, pp. 165–196. Springer Berlin Heidelberg.
60. Silver, D. L. and Yang, Q., Li, L. (2013). Lifelong machine learning systems: Beyond learning algorithms. In: 2013 AAAI Spring Symposium Series.
61. Metzen, J. H., Fabisch, A., Senger, L., de Gea Fernández, J. and Kirchner, E. A. (2013). Towards learning of generic skills for robotic manipulation. KI - Kunstliche Intelligenz, pages 1–6.
62. Dindo, H., Chella, A., Tona, G. L., Vitali, M., Nivel, E. and Thórisson, K. R. (2011). Learning problem solving skills from demonstration: An architectural approach. In Schmidhuber, J., Thórisson, K. R., and Looks, M., editors, AGI, volume 6830 of Lecture Notes in Computer Science, pages 194–203. Springer.
63. Argall, B. D., Chernova, S., Veloso, M. and Browning, B. (2009). A survey of robot learning from demonstration. Robotics and Autonomous Systems, 57(5):469–483.
64. Schaal, S. (1997). Learning from demonstration. In Advances in Neural Information Processing Systems 9. MIT Press.
65. Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational approaches to motor learning by imitation. Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences **358**(1431), 537–547.
66. Ito, M. and Tani, J. (2004). On-line Imitative Interaction with a Humanoid Robot Using a Dynamic Neural Network Model of a Mirror System. Adaptive Behavior **12**(2), 93–115.
67. Leòn, A., Morales, E. F., Altamirano, L., and Ruiz, J. R. (2011). Teaching a Robot to Perform Task through Imitation and On-line Feedback. Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, Lecture Notes in Computer Science, 7042, 549-556.

68. Poubel, L. P., Sakka, S., Cehajic, D., and Creusot, D. (2014). Support changes during online human motion imitation by a humanoid robot using task specification. In: IEEE International Conference on Robotics and Automation (ICRA), 1782-1787.
69. Taylor, M. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research, 10(1):16331685.
70. da Silva, B. C., Konidaris, G., and Barto, A. G. (2012). Learning parameterized skills. In: Proceedings of the 29th International Conference on Machine Learning (ICML 2012). Edinburgh, Scotland.
71. Ruvolo, P. and Eaton, E. (2013). Active task selection for lifelong machine learning. In: Twenty-Seventh AAAI Conference on Articial Intelligence.
72. da Silva, B., Konidaris, G., and Barto, A. (2014). Active Learning of Parameterized Skills. In: Proceedings of the 31st International Conference on Machine Learning (ICML 2014).
73. Fabisch, A. and Metzen, J. (2014). Active Contextual Policy Search. Journal of Machine Learning Research, 15:3371-3399.
74. Senger, L., Schröer, M., Metzen, J., and Kirchner, E. A. (2014). Velocity-Based Multiple Change-point Inference for Unsupervised Segmentation of Human Movement Behavior. In: Proceedings of the 22nd International Conference on Pattern Recognition (ICPR 2014).
75. Deisenroth, M. P., Neumann, G., and Peters, J. (2013). A survey on policy search for robotics. Foundations and Trends in Robotics 2(12), 328373.
76. Daniel, C., Neumann, G., and Peters, J. (2013). Learning Sequential Motor Tasks. In: Proceedings of 2013 IEEE International Conference on Robotics and Automation (ICRA).
77. Haddadin, S., Albu-Schffer, A., and Hirzinger, G. (2009). Requirements for safe robots: Measurements, analysis and new insights. In: The International Journal of Robotics Research (IJRR), 28(11–12), 1507–1527.

Physical Safety in Robotics

Sami Haddadin

Institute of Automatic Control (IRT)
Leibniz Universität Hannover
Appelstr. 11 D-30167 Hannover, Germany
haddadin@irt.uni-hannover.de
WWW: http://www.irt.uni-hannover.de

Abstract. Over the last decade, safe physical Human-Robot Interaction (pHRI) has been made possible due to significant advances in mechatronics, control, and planning. One result of these developments were fully integrated safer lightweight robots that are equipped with sophisticated interaction control capabilities. These new robots have even opened up novel and unforeseen application domains, in which human and robot are sought to work and interact with each other. For this, safe physical interaction is prime. This chapter gives a brief overview on two of its central aspects: human safety from an injury and standards standpoint, and control for physical interaction with focus on interaction control and collision handling.

Keywords: robot safety, physical human-robot interaction, injury analysis, collision handling

1 Introduction

Robotics is currently undergoing a fundamental paradigm shift, both in research and real-world applications. Classically, it was dominated for the last decades by possibly dangerous position controlled rigid robots carrying out typical automation tasks such as positioning and path tracking in various applications. Recently, a new generation of mechatronic robots has appeared on the landscape, including novel concepts in general robot design within the soft-robotics context. These trends bring us closer to the long-term goal of safe, seamless pHRI in the real domestic and professional world.

Recent advances in physical Human-Robot Interaction, including the vast progress of 3D perception, have shown the potential and feasibility of robot systems for active and safe workspace sharing and collaboration with humans. The fundamental breakthrough was the human-centered design of robot mechanics and control (soft-robotics), which also induced the novel research stream of intrinsically elastic robots (Series Elastic Actuation (SEA) or its generalization Variable Impedance Actuation (VIA)). By considering the physical contact of the human and the robot in the design phase, possible injuries due to unintentional contacts can be considerably mitigated. Furthermore, taking into account the

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_9,

human's intention and preferences will enable the realization of human-friendly motions and interaction behavior. Some of the most advanced systems that were developed are now entering industrial markets. These technologies serve both industrial and service oriented domains. Possible future applications of these novel devices developed for close interaction with humans are depicted in Fig. 1. They range from industrial co-workers and mobile servants over robots in the

Fig. 1. Application examples for pHRI, ranging from shop floor logistics and manipulation, over professional service robots and assistive devices for the disabled, to service robots in domestic applications.

professional service sector, assistive devices for physically challenged individuals, to service robots for support of general household activities. All of these applications share the common requirement of close, safe and dependable physical interaction between human and robot in a shared workspace. Therefore, such robots need to be carefully designed for human-friendliness. That is, they have to be able to safely sense, reason, learn, and act in a partially unknown world inhabited by humans.

2 Human Safety

Providing safety in pHRI is a multi-faceted challenge and requires an analysis on various levels of abstraction. pHRI aims at the coexistence of humans and robots in a common workspace and at extending their communication modes by physical means. This spatial proximity leads to a variety of *potential threats*, determined by the current state of the system of interest, which consists of the human(s), the robot(s) and their surrounding environment. Understanding the respective threats, in particular regarding potential human injury originating

from physical robot-human contacts, and embedding the insights into according safety standards/regulations is one of the major challenges of nowadays robotics.

2.1 Human Injury in Robotics

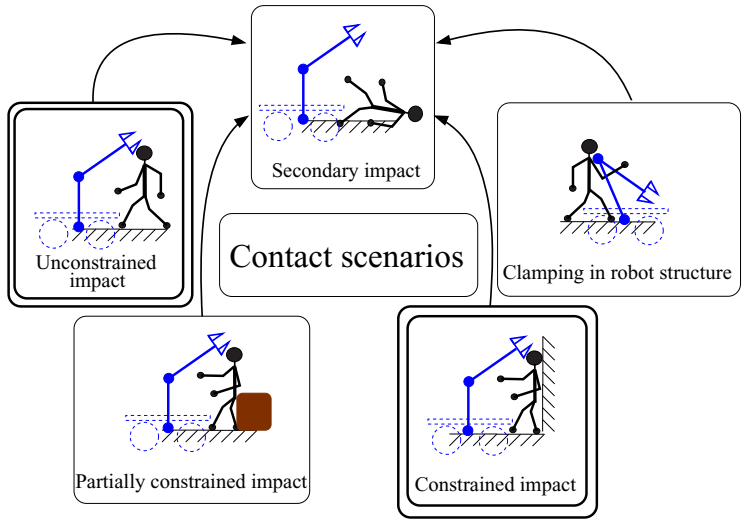


Fig. 2. Robot-human impact scenario classes. Unconstrained and constrained impacts are considered the two main scenarios.

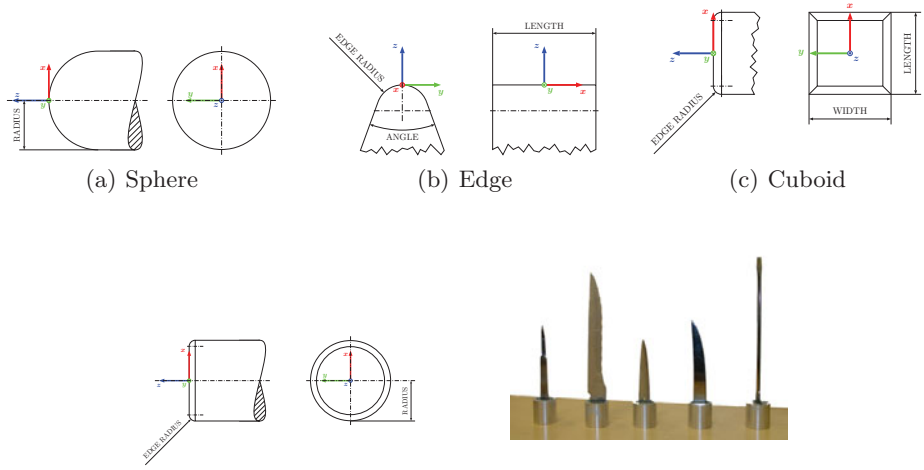
Impact Scenarios In order to quantify human injury that may occur in the context of pHRI, one needs to understand how mechanical forces may cause injury in principle. Figure 2 depicts relevant robot-human impact scenarios. These may involve unconstrained impacts, clamping in the robot structure, constrained impacts, partially constrained impacts, and resulting secondary impacts [1]. Apart from such situational definitions, the most urgent question is how to quantify human injury level that might occur due to a collision between human and robot. The understanding of human injury has been treated in the fields of injury biomechanics and forensics for several decades and the respective studies served for the early work on human injury in robotics. In fact, various injury measures from biomechanics and forensics were applied to human injury analysis in robotics [2], [3],[4], [5], [1], [6], [7]. An overview on the most important existing injury classification metrics and biomechanical injury measures can be found in [8]. The most important results from biomechanics, forensics, and robotics literature are briefly reviewed now.

Overview Biomechanics Literature In order to derive the injury characteristics of different body parts for direct collisions with an impactor, which is the

Table 1. Selected impact experiments from biomechanics and robotics literature.

Body part	Impactor type	Impactor parameters	Collision case	Subject	Mass [kg]	Velocity [m/s]	Reference
	Flat circular						
	Maxilla, Zygoma, Frontal, Temporo-Parietal, Mandible	14.3 mm radius	DC	Cadaver	1.08 - 3.82	2.99 - 5.97	[9], [10]
	Temporo-Parietal	12.7 mm radius	DC	Cadaver	10.6	2.7	[11]
	Nose	14.3 mm radius	DC	Cadaver	3.2	1.56 - 3.16	[12]
	Frontal	35 mm radius	DU	Cadaver	14.3	3.37 - 6.99	[13]
	Edge						
	Nose	12.5 mm radius	DU	Cadaver	32, 64	2.77 - 6.83	[14]
Head	Maxilla, Zygoma, Frontal	10 mm radius	DC	Cadaver	14.5	2.4 - 4.2	[15]
	Frontal	12.7 mm radius	DPC	Cadaver	∞ (human falling on impactor)	2.23 - 3.14	[16]
	Cuboid						
	Temporo-Parietal	50 mm length, 100 mm width	DC	Cadaver	12	4.3	[11]
	Frontal	size not specified, padded	DPC	Cadaver	5.31 - 5.97	3.56 - 9.6	[17]
	Frontal	size not specified	DPC	Cadaver	∞ (human falling on impactor)	2.23 - 3.87	[16]
	Sphere						
	Frontal	120 mm radius	DU, QSC, DPC	Hybrid III dummy	4, 67, 1980	0.2 - 4.2	[18]
	Frontal	203.2, 76.2 mm radius	DPC	Cadaver	∞ (human falling on impactor)	2.87 - 3.5	[16]
	Flat circular						
	Thorax	76.2 mm radius, 12.77 mm edge radius	DU, DC	Cadaver	1.6 - 23.6	4.34 - 14.5	[19], [20]
	Thorax	76 mm radius, rubber padded	DU	Volunteer	10	2.4 - 4.6	[21]
	Thorax	76.2 mm radius, 12.77 mm edge radius	DU	Cadaver	19.27	4.0 - 10.6	[22]
Torso	Abdomen	12.7 mm radius	DU	Cadaver	32, 64	4.9 - 13.0	[23]
	Sphere						
	Thorax	120 mm radius	DU, QSC	Hybrid III dummy	4, 67, 1980	0.2 - 4.2	[18]
	Abdomen	5, 12.5 mm radius	DC	Pig tissue	2 - 10	0.5 - 4.0	[7]
	Edge						
	Abdomen	45° angle, 200 mm length, 0.2 mm edge radius	DC	Pig tissue	2 - 10	0.5 - 4.0	[7]
	Edge						
	Forearm	12.5 mm radius, angle 0°	DC	Cadaver	9.48	3.63	[24]
Upper extremities	Forearm	size not specified	DC	Cadaver	9.75	2.44, 4.23	[25]
	Shoulder, upper arm, forearm	5 mm edge radius, 30° angle	DC	Volunteer	4.16, 8.65	0.45 - 1.25	
	Flat circular						
	Forearm, hand	size not specified	QSC	Cadaver	∞ (velocity control)	25 mm/min	[26]
Lower extremities	**Sharp**	see Fig. 3 (e)	DC	Pig tissue, volunteer	4	0.16 - 0.8	[6]

most relevant case for robotics, countless experiments and publications have been produced over the last 50 years. The investigated impactors used in robotics and biomechanics experiments vary significantly in size and shape. However, from the test setups one can identify and cluster principal geometric primitives. The main primitives and their parameters are depicted in Fig. 3. The z-axis of the coordinate frame associated to each primitive defines the direction of impact $\mathbf{u}$.



(a) Sphere (b) Edge (c) Cuboid

(d) Flat circular (e) Sharp tools

Fig. 3. Typical impactor primitives with according parameters.

Numerous relevant impact experiments with cadavers, volunteers, crash test dummies, and biological tissue for the head, neck, and chest were generated, see Tab. 1. There, for all selected experimental campaigns the collision scenario, impacted body part, impact parameters according to Fig. 3, subject, and impact velocity are listed. For describing the collision scenario, we use following abbreviations: D: dynamic, QS: quasi-static, U: unconstrained, C: constrained, PC: partially constrained. A collision experiment denoted by DU is thus dynamic unconstrained, while quasi-static constrained impacts are labeled QSC (see also Sec. 2.1). The respective impactor type and parameters are listed for comparison.

Next, some essential characteristics of human-robot impacts are elaborated for a more general understanding of the underlying dynamics.

Robot-Human Impacts

Robot Collision Modeling Let us assume that of a serial chain rigid robot consisting of n joints there is at most a single link involved in a collision. Let

$$\dot{\boldsymbol{x}}_c = \begin{bmatrix} \boldsymbol{v}_c \\ \boldsymbol{\omega}_c \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_{c,\mathrm{lin}}(\boldsymbol{q}) \\ \boldsymbol{J}_{c,\mathrm{ang}}(\boldsymbol{q}) \end{bmatrix} \dot{\boldsymbol{q}} = \boldsymbol{J}_c(\boldsymbol{q})\dot{\boldsymbol{q}} \in \mathbb{R}^6 \tag{1}$$

be the stacked (screw) vector of linear velocity at the contact point and angular velocity of the associated robot link, with an associated (geometric) contact Jacobian $\boldsymbol{J}_c(\boldsymbol{q})$ that is a function of the joint angle $\boldsymbol{q}$. Accordingly, the Cartesian collision wrench, being the stacked vector of collision force $\boldsymbol{f}_{\text{ext}}$ and collision moments $\boldsymbol{m}_{\text{ext}}$, is denoted by

$$\boldsymbol{\mathcal{F}}_{\text{ext}} = \begin{bmatrix} \boldsymbol{f}_{\text{ext}} \\ \boldsymbol{m}_{\text{ext}} \end{bmatrix} \in \mathbb{R}^6. \tag{2}$$

When such a collision occurs, the robot dynamics becomes

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\tau}_F = \boldsymbol{\tau} + \boldsymbol{\tau}_{\text{ext}}, \tag{3}$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ is the symmetric and positive definite joint space inertia matrix, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \in \mathbb{R}^n$ is the centripetal and Coriolis vector, and $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$ is the gravity vector. $\boldsymbol{\tau} \in \mathbb{R}^n$ is the motor torque and $\boldsymbol{\tau}_F \in \mathbb{R}^n$ the dissipative friction torque. $\boldsymbol{\tau}_{\text{ext}} \in \mathbb{R}^n$ is the typically unknown external joint torque given by

$$\boldsymbol{\tau}_{\text{ext}} = \boldsymbol{J}_c^T(\boldsymbol{q})\,\boldsymbol{\mathcal{F}}_{\text{ext}}. \tag{4}$$

The effective mass $m_{\boldsymbol{u}}$ of a robot acting in the instantaneous collision direction $\boldsymbol{u}$, which has to be consistent to $\boldsymbol{J}_c(\boldsymbol{q})$, can be deduced from $\boldsymbol{M}(\boldsymbol{q})$ via the Cartesian kinetic energy matrix $\boldsymbol{\Lambda}(\boldsymbol{q})$. This is defined as

$$\boldsymbol{\Lambda}(\boldsymbol{q}) = (\boldsymbol{J}_c(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{J}_c(\boldsymbol{q})^T)^{-1}, \tag{5}$$

where the inverse of $\boldsymbol{\Lambda}(\boldsymbol{q})$ is based on the decomposition of the kinetic energy matrix:

$$\boldsymbol{\Lambda}(\boldsymbol{q})^{-1} = \begin{bmatrix} \boldsymbol{\Lambda}_v(\boldsymbol{q})^{-1} & \overline{\boldsymbol{\Lambda}}_{v\omega}(\boldsymbol{q}) \\ \overline{\boldsymbol{\Lambda}}_{v\omega}(\boldsymbol{q})^T & \boldsymbol{\Lambda}_\omega(\boldsymbol{q})^{-1} \end{bmatrix}, \tag{6}$$

with $\overline{\boldsymbol{\Lambda}}_{v\omega}(\boldsymbol{q}) = \boldsymbol{J}_{c,\text{lin}}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{J}_{c,\text{ang}}(\boldsymbol{q})^T$. Finally, $m_{\boldsymbol{u}}$ is found to be

$$m_{\boldsymbol{u}} = [\boldsymbol{u}^T\boldsymbol{\Lambda}_v(\boldsymbol{q})^{-1}\boldsymbol{u}]^{-1}. \tag{7}$$

It should be noted that the Jacobian has to be the *center of mass Jacobian*. Otherwise, the entire inverse of the Cartesian inertia tensor has to be used and not only its translational component block. More details can be found in [27]. We assume the local impact curvature in $\boldsymbol{u}$-direction to be denoted $c_{\boldsymbol{u}}$.

Characteristic Robot-Human Impact Force Profile A physical collision between robot and human is typically characterized by a distinct force profile that is composed by two consecutive phases[1], see Fig. 4.

1. Phase I is characterized by a very short impact, governed by the robot and human reflected dynamics.

[1] Note that for unconstrained soft-tissue collisions these two phases can simplify into a single *Phase I* impact.

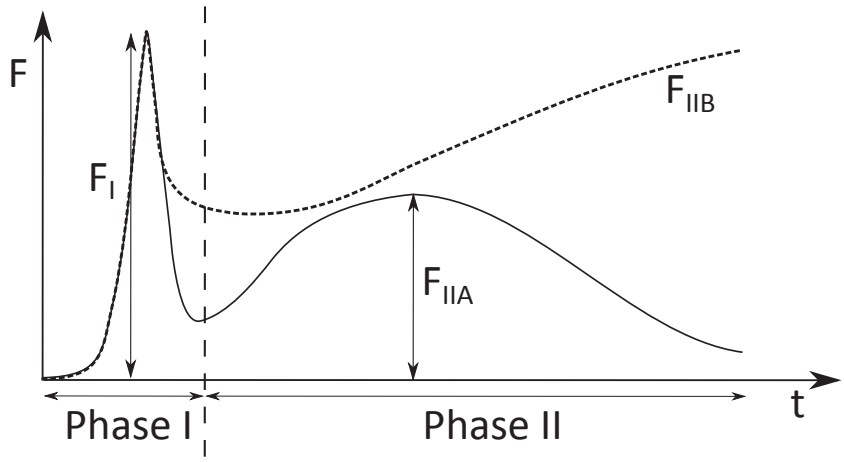


Fig. 4. Typical robot-human collision force profiles.

2. Phase II is characterized by a quasi-static contact event. Without clamping this is a pushing force, whereas if the human is clamped it is a crushing force.

Phase I can be treated from a pure impact physics point of view, i.e. it is determined by the reflected inertia, velocity, and impact curvature $c_{\boldsymbol{u}}$ of the robot together with the characteristics of the respective body part that is being struck. The maximum contact force is denoted F_I.

Phase II, on the other hand, has to be further subdivided into either *clamping* or *no clamping* incident. In case of no clamping, the maximum force is F_{IIA}, for clamping the maximum force is F_{IIB}. In particular, *Phase II* is highly robot control and design dependent and is especially important in case of clamping.

- *Phase IIA: no clamping*
  Typically, for free impacts at robot velocities > 0.3 m/s, F_{IIA} is significantly smaller than F_I. Otherwise, F_I is smaller than F_{IIA} and is governed by the robot actuator torques (active quasi-static pushing) and the reaction of the human body that is mainly governed by its reflected impedance.
- *Phase IIB: clamping*
  In case of clamping the final maximum force F_{IIB} is limited by the maximum motor torques $\boldsymbol{\tau}_{\max}$ of the robot via $\mathcal{F}_{\text{ext}} = J_c^{T\#}\boldsymbol{\tau}_{\max}$, where $J_c^{T\#}$ is the contact Jacobian pseudoinverse. If the robot is powerful enough to generate active contact forces that penetrate or break human tissue/structure, the contact force is of course limited by the human maximum tissue resistance[2].

Next, the influence of robot mass and velocity for the unconstrained impact are described. This analysis is particularly important to understand Phase I.

[2] Please note that singularities need careful treatment, which however, goes beyond the scope of the chapter.

Influence of Robot Mass and Velocity Assume a simple mass-spring-mass model for the impact between human and robot. M_H is the reflected inertia of the human[3]. K_H is the contact stiffness which is in case of a rigid robot mainly the effective stiffness of the human contact area. $\dot{x}_{re}^0$ is the relative impact velocity between the robot and human. Solving the corresponding differential equation leads to the maximum contact force

$$\mathcal{F}_{\text{ext}}^{\max} = \sqrt{\frac{m_{\boldsymbol{u}} M_H}{m_{\boldsymbol{u}} + M_H}} \sqrt{K_H} \dot{x}_{re}^0. \tag{8}$$

The dependency of frontal bone contact force on the robot mass and velocity is depicted in Fig. 5 (upper). It can be observed that collision force (which is a well known bone fracture indicator) generally increases with velocity. For increasing mass, however, a saturation effect takes place. After a certain robot mass has been reached ($m_{\boldsymbol{u}} \approx 20$ kg in Fig. 5), additional weight has only negligible influence on collision force. This inertial saturation effect can also be observed for other impact locations such as contacts with the chest, see Fig. 5 (lower).

If the robot mass is significantly larger than the human head mass, i.e. $m_{\boldsymbol{u}} \gg M_H$, equation (8) reduces to

$$\mathcal{F}_{\text{ext}}^{\max}(m_{\boldsymbol{u}} >> M_H) = \sqrt{K_H M_H} \dot{x}_{re}^0. \tag{9}$$

This shows that for a robot with significantly larger reflected inertia than the human head, only the contact stiffness, the impact velocity, and the mass of the human head are relevant but not the robot mass.

Table 2. Impact data for the lateral surface of the right upper arm.

Impact	Max. impact force (N)	Impact area (mm^2)	Displacement (m)	Tissue stiffness (N/m)	Stress σ (N/mm^2)	Impact velocity (m/s)	Kinetic energy (J)	Energy density (J/mm^2)	AO	VAS
1	9.5	966	0.03	316.7	0.001	0.2	0.08	0.0001	IC1MT1NV1	0
2	19	966	0.037	513.5	0.002	0.44	0.36	0.0007	IC1MT1NV1	0
3	38.1	966	0.044	865.9	0.039	0.65	0.80	0.0016	IC1MT1NV1	0
4	59.6	966	0.055	1083.6	0.062	0.88	1.45	0.003	IC1MT1NV1	0
5	81.4	966	0.058	1403.4	0.084	1.11	2.31	0.005	IC1MT1NV1	1
6	103.5	966	0.060	1725	0.107	1.34	3.37	0.007	IC1MT1NV1	1.5
7	128.1	966	0.064	2001.6	0.133	1.55	4.50	0.009	IC1MT1NV1	2
8	154.1	966	0.069	2233, 3	0.16	1.76	5.81	0.012	IC1MT1NV1	3
9	186.4	966	0.069	2701.4	0.193	2.03	7.73	0.016	IC1MT1NV1	3
10	224.5	966	0.069	3253.6	0.253	2.24	9.41	0.019	IC1MT1NV1	4
11	272.2	966	0.077	3535.1	0.282	2.55	12.2	0.025	IC1MT1NV1	6

Human-Robot Impact Voluntary Testing The behavior of human tissue during collisions is complex. Consequently, surrogates cannot reveal the entire diversity. Accordingly, the conduction of human voluntary experiments is necessary to fully understand human injury and pain dynamics in robotics. The following experimental test, which was carried out by the author, was the first

[3] Assuming a simplifying decoupling of the head from the torso, which holds for the short duration of the impact. For the post-impact phase, neck stiffness and body inertia have to be considered, which complicates the analysis considerably.

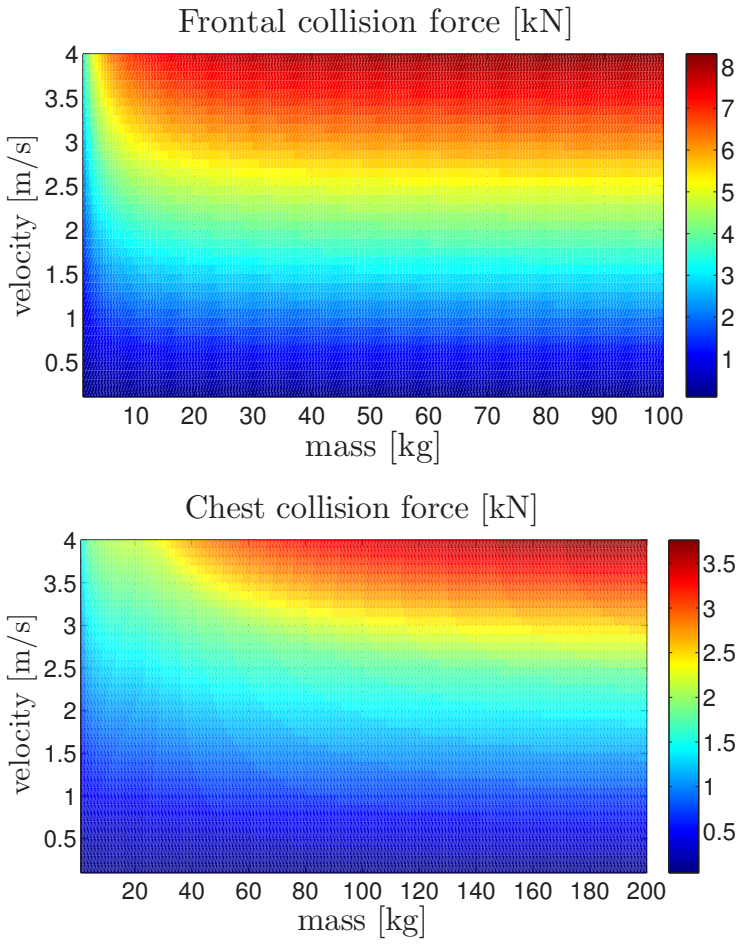


Fig. 5. Mass-velocity dependency for human head and chest contact force. A mass-spring-mass model is used for collisions against the head, where the head mass M_H is 4.5 kg and the approximate contact stiffness of the frontal bone $K_H = 1000$ N/mm [15]. For the chest, the model proposed in [28] is used.

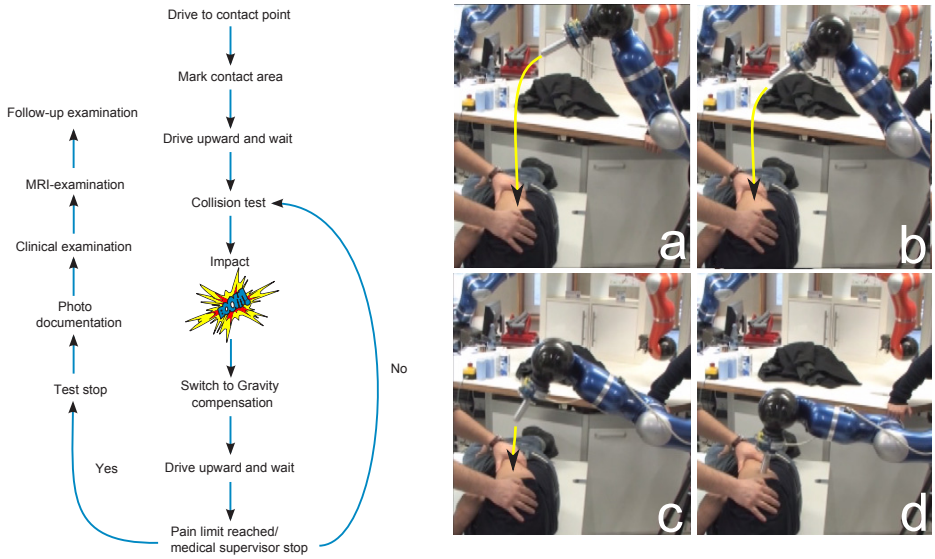


Fig. 6. Flow-chart depicting the basic experimental steps (left). Collision trajectory with subject (right).

systematic analysis in this direction. The voluntary experiments were conducted with a healthy young adult in the year 2011. The collision experiments were performed with the KUKA/DLR LWR and following approaches to injury and pain analysis were carried out: *injury severity analysis according to AO*[4], *biomechanical analysis*, *pain*, and *imaging methods*. The setup and experiment steps are depicted in Fig. 6. The robotic system allows to conduct controlled robot-human collisions in order to analyze input parameters and their effect on output parameters such as pain and injury. Measured impact characteristics and quantities included impact force, impact area, tissue displacement, tissue stiffness, stress, impact velocity, kinetic energy and energy density. The reflected inertia was kept constant at $m_{\boldsymbol{u}} = 3.75$ kg for every test. The used impactor for the resulting Tab. 2 was a sphere with radius of 12.5 mm.

The injury was defined using the AO-classification [29] directly after each test series. Each impact series was carried out at the same location on the human body at increasing impact velocity until the participant initiated a controlled system stop during the experiment. The impact areas were then imaged with a magnetic resonance imaging (MRI) after a time interval of about 4-5 hours. The remaining tissue did not show any pathological signs. Compared to an equivalent drop test in [7] with abdominal pig tissue (large sphere, 4.2 kg, 2.5 m/s), the voluntary experiments provide similar results in terms of injury severity. The maximum velocity of 2.55 m/s is at the border of inducing a contusion. Where

[4] AO stands for "Arbeitsgemeinschaft für Ostheosynthesefragen" [29].

there were no marks immediately after impact, a mild contusion formed at day 1. For the pain tolerance at a VAS of 6/10 an impact force of $F = 272.2$ N was measured. The energy density appears to have the most significant correlation to pain.

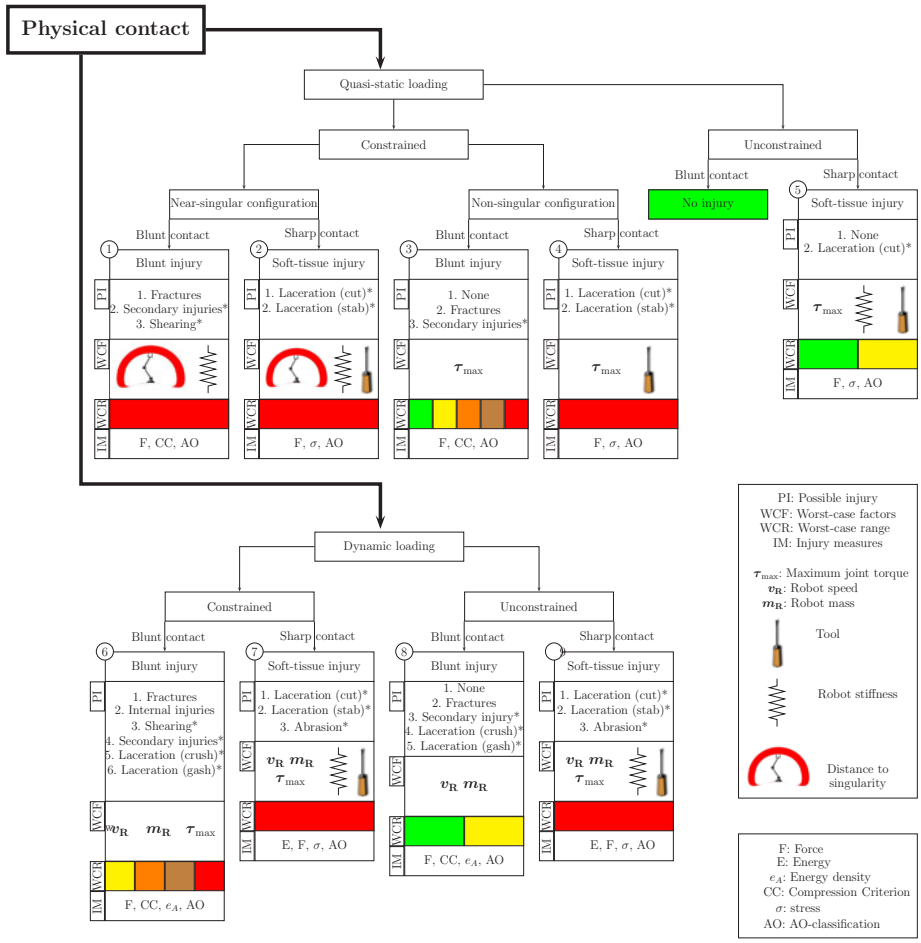


Fig. 7. Safety Tree showing possible injury (PI), major worst-case factors (WCF) and the possible worst-case range (WCR). * indicates still ongoing topics of research. Additionally, relevant injury criteria are given for the head, chest, and soft-tissue injuries.

Synopsis An overview of the potential injury threats depending on the current state of the robot and the human, a classification of these mechanisms, governing factors of the particular process and possible injuries are depicted in Fig. 7.

Physical contact can be divided into two fundamental subclasses: quasi-static and dynamic loading. Fundamental differences in injury severity and mechanisms are observed as well if a human is (partially) constrained or not, leading to the second subdivision. For the quasi-static case it is differentiated between near-singular and non-singular clamping as already outlined. The last differentiation separates injuries caused by blunt contact from the ones induced by tools or sharp surface elements.

Each class of injury is characterized by possible injuries (PI), worst-case factors (WCF) and their worst-case range (WCR). WCF are the main contributors to the worst-case, such as maximum joint torque, the distance to singularity or the robot speed. The worst-case range indicates the maximum possible injury depending on the worst-case factors. In addition to the classification of injury mechanisms for each such class, suggestions for injury measures (IM) are given as well. They are specific injury measures which are appropriate, useful for the classification and measurement of injury potentially occurring during physical Human-Robot Interaction[5].

For example ① represents blunt clamping in the near-singular configuration, see Fig. 7. Even for low-inertia robots this situation could become dangerous and is therefore a possible serious threat with almost any robot on a fixed base within a (partially) confined workspace. Possible injuries are fractures and secondary injuries e.g. caused by penetrating bone structures or an injured neck if the trunk is clamped but the head is free. This would mean that the robot pushes the head further while the trunk remains in its position. Another possible threat is shearing off a locally clamped human along an edge. Appropriate indices are e.g. the contact force and the Compression Criterion (CC) [30]. ③ represents the clamped blunt impact in non-singular configuration. The injury potential is defined by the maximum actuation torque $\boldsymbol{\tau}_{\max}$ and can range from no injury to severe injury or even death for high-inertia (and torque) robots. The robot stiffness does not contribute to the worst-case since a robot without collision detection would simply increase the motor torque to follow the desired trajectory. Therefore, robot stiffness only contributes to the detection mechanism by enlarging the detection time. Also, the contact force and CC are well suited to predict occurring injury. ⑧ denotes the unconstrained impact which was the first injury mechanism investigated in the robotics literature. This process is governed by the impact velocity and (up to a saturation value) by the robot mass. As shown in [4] even a robot of arbitrary mass cannot severely injure a human head by means of impact related criteria from the automobile industry like the head injury criterion (HIC). However, fractures e.g. of facial bones are likely to occur but not all would be classified as a serious injury. Laceration by means of crushes and gashes are worth to be evaluated, especially with respect

[5] Please note that the list of injury measures is not necessarily complete, but these ones are certainly suitable to be applied to a more granular robotics injury analysis. This does not mean that criteria such as the well known Head Injury criterion (HIC) do not provide general insights, they are just not necessarily optimal to understand injury on a more differentiated lower-injury scale.

to service robotics. The contact force and CC are well suited severity criteria for this class and in order to evaluate lacerations the energy density has to be considered.

The preceding overview is intended as a worst-case analysis for the described contact cases. The next step is to ask which actions can be taken against each particular threat. [1] discusses this thoroughly. At this point, however, it shall be noted that instead of quantifying injury in terms of a measurable injury criterion, injury evaluation by a medical expert e.g. via the AO-classification can always be applied and would presumably result in a more exhaustive and precise judgement.

2.2 Safety Standards for Human-Robot Interaction

Robotics standardization made significant progress to establish the underlying regulations for co-working cells in the real world. Safety for industrial robots is addressed in a variety of general standards [31], [32], [33]. The most important industrial robotics standards is the ISO 10218. It was established in recognition of the particular hazards that industrial robots and industrial robot systems pose. The machinery concerned and the extent to which hazards, hazardous situations and events are covered, is indicated in the scope of ISO 10218. In recognition of the variable nature of hazards with different uses of industrial robots, ISO 10218 is divided into two parts. It provides a detailed analysis of mechanical hazards such as impacts (movements of any part of the robot arm), crushing (movement of any part of the robot arm), shearing (movement of additional axes), entanglement (rotation of wrist or additional axes), drawing-in or tapping (between robot arm and any fixed object), cutting or severing (movement or rotation creating scissors action), and contact of persons with live parts (direct contact) [34]. In particular, the introduction of collaborative robots has been a major acknowledgment to the advances made in robotics research in pHRI over the last decade. The recent updates to ISO 10218 (safety requirements for industrial robots) lead to the development of the new TS 15066. It is regarded as a complementary information that concretizes the content of ISO 10218. Generally, ISO/TS 15066 provides guidance for collaborative robot operation where a robot and a person share the same workspace. The TS 15066 considers collaborative modes and requirements such as minimum separation distances, safety-rated monitored stops, speed and separation monitoring, and power and force limiting. In collaborative operations the integrity of the safety-related control system is of major importance, particularly when process parameters such as speed and force are being controlled. A comprehensive risk assessment is required to assess not only the robot system itself, but also the environment in which it is placed, i.e. in the workplace. A key process in the elimination of hazards and reduction of risks is the design of the collaborative robot system and the associated cell layout. Various considerations about the access and clearance of the collaborative workspace are provided. During the design of a robotic system, the maximum space and the restrictions of the collaborative robot system have to be considered. Furthermore, the need for clearances around obstacles and the accessibility

for operators should influence the design. The intended contact(s) between portions of the robot system and an operator play a major role towards a possibly intrinsically safe design. In order to identify the risks resulting from the collaborative action, an appropriate set of collision incidents that can occur during the collaborative work activities and foreseeable misuse has to be determined. This has to include affected body regions and the involved collision areas of the robot. The limit values that may not be exceeded during the collision incident depend on the affected body regions. The geometry of the involved areas of the robot and the biomechanical properties of the affected body regions influence the forces occurring during the collision incident. Therefore, the ISO/TS 15066 describes injury severity criteria that consist of maximum allowable limit values on individual body regions. These limit values are established to prevent the occurrence of skin/tissue penetrations that are accompanied by bleeding wounds, fractures or other skeletal damage [35].

In addition to the industrial standardization efforts in the pHRI domain, the ISO 13482 [36] is the first non-industrial robot safety standard that allows/regulates close pHRI. This international standard specifies requirements and guidelines for the inherent safe design, protective measures, and information for use of so called *personal care robots*. It focuses on three types of personal care robots (mobile servant robots, physical assistant robots and person carrier robots). These robots typically perform tasks to improve the quality of life of intended users irrespective of age or capability. The standard describes hazards associated with the use of these robots and provides requirements to eliminate or reduce the risks associated with these hazards to an acceptable level. Significant hazards are presented and this standard describes how they are to be dealt with for each personal care robot type. Robotic devices used in personal care applications are also covered by this standard and are to be treated as personal care robot.

3 Control for Physical Interaction

For soft and safe pHRI the question arises how to gently handle physical contact in robotics from a controls point of view. As impedance control [37] became the most popular interaction control paradigm in the pHRI world, this particular scheme will be one focus of this section. Its generalization to multi-priority impedance control laws allows the realization of sophisticated robot compliance with multiple objectives via active control. A major advantage of impedance control is that discontinuities like contact-non-contact, do not create stability problems as they occur, for example, with hybrid force control [38]. Its extension to impedance and feed-forward learning and adaptation, for which first works can be found in [39, 40], is discussed after introducing the concept of multi-priority impedance control. Apart from nominal interaction control, a robot sharing its workspace with humans and physically interacting with its environment should be able to quickly detect collisions and safely react to them. In the absence of external sensing, relative motions between robot and environment/human are

unpredictable and unexpected collisions may occur at any location along the robot arm. The state-of-the-art schemes for collision detection and reflex reaction are introduced.

3.1 Interaction Control

Originally developed for robust and compliant object manipulation, impedance and the related admittance control form a paradigm to treat robotic systems from an energetic point of view such that motion and force can be controlled in a unified manner. They offer the advantage over standard hybrid force-motion controllers to provide a framework independent from kinematic work space constraints. These control types popularized by [37] are also especially advantageous in terms of uncertainties and disturbances in unknown environments due to their inherently robust nature [41]. The terms impedance and admittance are derived from electrical system theory where they describe the relationship between voltage and current as input/output pairs. To generalize impedance and admittance such pairs can be defined domain-independently as effort and flow variables. For robotics, the mechanical analogies, i.e. mechanical impedance and admittance are of particular interest.

More details on the conceptual basics of impedance and the dual admittance control can e.g. be found in [42],[43].

The mostly used version of impedance control is to impose a second order dynamics of a mass-spring-damper system (so-called target impedance [37]) on the closed-loop equations. Typically, the control objective is expressed in Operational space coordinates $\boldsymbol{x}$ as

$$\boldsymbol{M}_x\ddot{\tilde{\boldsymbol{x}}} + \boldsymbol{D}_x\dot{\tilde{\boldsymbol{x}}} + \boldsymbol{K}_x\tilde{\boldsymbol{x}} = \boldsymbol{\mathcal{F}}_{ext}, \tag{10}$$

where $\tilde{\boldsymbol{x}} := \boldsymbol{x} - \boldsymbol{x}_d$ is the position error and $\boldsymbol{x}_d$ is called equilibrium position. $\boldsymbol{M}_x$ denotes the desired inertia, while $\boldsymbol{D}_x$ and $\boldsymbol{K}_x$ are the according closed-loop damping and stiffness matrices in Operational space. Assuming rigid body dynamics, the control law to obtain the aforementioned behavior is

$$\begin{aligned}\boldsymbol{\tau}_C = {} & \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{J}(\boldsymbol{q})^T(\boldsymbol{\Lambda}(\boldsymbol{q})\ddot{\boldsymbol{x}}_d + \boldsymbol{\mu}(\dot{\boldsymbol{x}}, \boldsymbol{x})) \\ & -\boldsymbol{J}(\boldsymbol{q})^T(\boldsymbol{\Lambda}(\boldsymbol{q})\boldsymbol{M}_x^{-1}(\boldsymbol{K}_x\tilde{\boldsymbol{x}} + \boldsymbol{D}_x\dot{\tilde{\boldsymbol{x}}})) + \boldsymbol{J}(\boldsymbol{q})^T(\boldsymbol{\Lambda}(\boldsymbol{q})\boldsymbol{M}_x^{-1} - \boldsymbol{I})\boldsymbol{\mathcal{F}}_{ext}. \end{aligned} \tag{11}$$

In order to fully implement this scheme, a wrist force torque sensor is necessary for the inertia shaping part. In [44] a modified impedance controller was designed that uses angle/axis representations for the rotational components of the Operational space. For its derivation, energy contributions with physical interpretation are considered and the end-effector orientation displacement representation is chosen to be in terms of a unit quaternion to avoid singularities.

For redundant robots, it is typically desired to also control the nullspace behavior in order to embed other control objectives $\boldsymbol{\tau}_{N,i}$ into a stacked hierarchy of tasks. For the case of a single nullspace controller $\boldsymbol{\tau}_N$ this torque has to be

projected via the nullspace projector matrix $\boldsymbol{N}(\boldsymbol{q})$ into the nullspace of the task, leading to the overall control law

$$\boldsymbol{\tau} = \boldsymbol{\tau}_C + \boldsymbol{N}(\boldsymbol{q})^T \boldsymbol{\tau}_N. \tag{12}$$

The nullspace projection matrix can be chosen in different ways. The simplest case is $\boldsymbol{N}(\boldsymbol{q}) = \boldsymbol{I} - \boldsymbol{J}(\boldsymbol{q})^{\#}\boldsymbol{J}(\boldsymbol{q})$, where $\boldsymbol{J}(\boldsymbol{q})^{\#}$ denotes the Moore-Penrose pseudo inverse. Alternatively, one may chose the *dynamically consistent* generalized pseudoinverse

$$\boldsymbol{J}(\boldsymbol{q})^{\#} = \boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{J}(\boldsymbol{q})^T \boldsymbol{\Lambda}(\boldsymbol{q}). \tag{13}$$

In particular in the pHRI domain, a multitude of different subtasks $\boldsymbol{\tau}_{N,i}$ are meaningful to be executed simultaneously. These may e.g. involve

- safety (collision anticipation & avoidance, self-collision avoidance, ...)
- physical constraints (joint limits, geometric task constraints)
- task execution (tracking control, ...)
- posture primitives (in particular for humanoids)

To realize consistent behaviors, task hierarchies are constructed such that certain tasks are prioritized over others [45]. In [46] a hierarchy is realized by null space projection techniques, which also prevents discontinuities concerning unilateral constraints by smoothing out transitions.

Extensions to the basic schemes for flexible joint dynamics [47], [48], [49] and for the SEA case [50] were developed as well. Furthermore, Cartesian impedance control has been applied to grasping und multiple-arm robotic systems in [51].

3.2 Collision Handling

One of the core problems in pHRI is the handling of collisions between robots and humans, with the primary motivation of limiting possible human injury due to physical contacts. Various *monitoring signals* can be used to gather context independent information about the event.

The *collision detection phase*, whose binary output denotes whether a robot collision occurred or not, is characterized by the transmission of contact wrenches, often for very short impact durations. The occurrence of a collision, which may happen anywhere along the robot structure, shall be detected as fast as possible. A major practical problem is the selection of a threshold on the monitoring signals, so as to avoid false positives and achieve high sensitivity at the same time. A rather intuitive approach is to monitor the measured currents in robot electrical drives, looking for fast transients possibly caused by a collision [52], [53]. Another proposed scheme compares the actual commanded torques (or motor currents) with the nominal model-based control law (i.e., the instantaneous torque expected in the absence of collision), with any difference being attributed to a collision [54]. This idea has been refined by considering the use of an adaptive compliance control [55], [56]. However, tuning of collision detection thresholds in these schemes is difficult because of the highly varying dynamic characteristics of the control torques.

Knowing which robot part (e.g., which link of a serial manipulator) is involved in the collision is an important information that can be exploited for robot reaction. *Collision isolation* aims at localizing the contact point $\boldsymbol{x}_c$, or at least which link i_c out of the n-body robot collided. One way to obtain both collision detection and isolation is to use sensitive skins [57], [58], [59], [60]. However, it is obviously more practical and reliable to detect and possibly isolate a collision without the need of additional tactile sensors. On the other hand, the previously mentioned monitoring signals used in [52], [53], [54], [55], [56] are in general not able to achieve reliable collision isolation (even when robot dynamics is perfectly known). In fact, they either rely on computations based only on the nominal desired trajectory, or compute joint accelerations by inverting the mass matrix and thus spreading the dynamic effects of collision on a single link, or use acceleration estimates for torque prediction and comparison, which inherently introduces noise (due to double numerical differentiation of position data) and intrinsic delays. The common drawback of these methods is that the effect of a collision on a link propagates to other link variables or joint commands due to robot dynamic couplings, affecting thus the isolation property.

Other relevant quantities about a collision that are deduced during the *collision identification phase* are the directional information and the intensity of the generalized collision force, either in terms of the acting Cartesian wrench $\boldsymbol{\mathcal{F}}_{\text{ext}}(t)$ at the contact, or of the resulting joint torque $\boldsymbol{\tau}_{\text{ext}}(t)$ during the entire physical interaction event. This information characterizes (in some cases, completely) the collision event. The first method that achieved simultaneously collision detection, isolation, and identification was proposed in [61]. The basic idea was to view collisions as faulty behaviors of the robot actuating system, while the detector design took advantage of the decoupling property of the robot *generalized momentum* $\boldsymbol{p} = \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}}$ [62], [63].

During the *collision reaction phase* the robot should react purposefully in response to a collision event, i.e., taking into account available contextual information. Because of the fast dynamics and high uncertainty of the problem, the robot reaction should be embedded in the lowest control level. For instance, the simplest reaction to a collision is to stop the robot. However, this may possibly lead to inconvenient situations, where the robot is unnaturally constraining or blocking the human [1]. To define better reaction strategies, information from collision isolation, identification and classification phases should be used. Some examples of successful collision reaction strategies have been given in [64], [64], [65].

Collision Detection and Identification A recent overview on standard techniques to estimate $\boldsymbol{\tau}_{\text{ext}}$ can be found in [8]. In this chapter, we focus on the main method, namely the monitoring scheme based on the observation of the generalized momentum that was introduced in [61]. The scheme, which is regarded as the standard algorithm, was motivated by the desire of avoiding the inversion of the robot inertia matrix, decoupling the estimation result, and also eliminating the need of an estimate of joint accelerations. Note that from now on, an

estimate of a generic vector $\boldsymbol{x}$ will be denoted by $\hat{\boldsymbol{x}}$. The according disturbance observer based estimator dynamics is defined as

$$\boldsymbol{r}(t) = \boldsymbol{K}_O \left(\hat{\boldsymbol{p}}(t) - \int_0^t \big(\boldsymbol{\tau} - \hat{\boldsymbol{\beta}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{r}\big) \mathrm{d}s - \hat{\boldsymbol{p}}(0) \right), \tag{14}$$

with $\hat{\boldsymbol{p}} = \hat{\boldsymbol{M}}(\boldsymbol{q})\dot{\boldsymbol{q}}$, $\hat{\boldsymbol{\beta}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \hat{\boldsymbol{g}}(\boldsymbol{q}) + \hat{\boldsymbol{C}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \dot{\hat{\boldsymbol{M}}}(\boldsymbol{q})\dot{\boldsymbol{q}}$, and $\boldsymbol{K}_O = \mathrm{diag}\{k_{O,i}\} > \boldsymbol{0}$ being the diagonal gain matrix of the observer. In ideal conditions, $\hat{\boldsymbol{M}} = \boldsymbol{M}$ and $\hat{\boldsymbol{\beta}} = \boldsymbol{\beta}$, the dynamic relation between the external torque $\boldsymbol{\tau}_{\mathrm{ext}}$ and $\boldsymbol{r}$ is

$$\dot{\boldsymbol{r}} = \boldsymbol{K}_O(\boldsymbol{\tau}_{\mathrm{ext}} - \boldsymbol{r}). \tag{15}$$

In other words, $\boldsymbol{r}$ is a stable, linear, decoupled, first-order estimation of the external collision torque $\boldsymbol{\tau}_{\mathrm{ext}}$. Large values of $k_{O,i}$ give small time constants $T_{O,i} = 1/k_{O,i}$ in the transient response of that component of $\boldsymbol{r}$ which is associated to the same component of the external joint torque $\boldsymbol{\tau}_{\mathrm{ext}}$. In the limit, we obtain

$$\boldsymbol{K}_O \to \infty \quad \Rightarrow \quad \boldsymbol{r} \approx \boldsymbol{\tau}_{\mathrm{ext}}. \tag{16}$$

Collision Reflex Reactions After a collision has been detected, suitable collision reflex reaction is needed. Four basic context-independent joint level collision reflexes are discussed next. They lead to significantly different reflex behavior after a contact was detected. In the third and fourth scheme the directional information on contact torques provided by suitable identification schemes such as (14) may be used to safely drive the robot away from the collision location.

Robot Stop The most obvious strategy to react to a collision is to stop the robot. This behavior can e.g. be obtained by setting $\boldsymbol{q}_d = \boldsymbol{q}(t_c)$, where t_c is the instant of collision detection or by simply engaging the robot's brakes. More elaborate braking strategies can be found in [66].

Torque Control with Gravity Compensation One may also react to a collision by switching the controllers. Typically, prior to the collision incident the robot moves along a desired trajectory with a position reference based controller (e.g. position or impedance control). After detection the control mode is switched to a compliance based controller that ignores the previous task trajectory. A particularly useful variant is to switch to torque control mode with gravity compensation $\boldsymbol{\tau} = \boldsymbol{g}(\boldsymbol{q})$. Note that this strategy does not explicitly take into account any information about $\boldsymbol{\tau}_{\mathrm{ext}}$.

Torque Reflex This strategy extends the torque control based strategy by explicitly incorporating the estimation or measurement of $\boldsymbol{\tau}_{\mathrm{ext}}$ into the motor torque $\boldsymbol{\tau}$ via

$$\boldsymbol{\tau} = \boldsymbol{g}(\boldsymbol{q}) + (\boldsymbol{I} - \boldsymbol{K}_r)\boldsymbol{\tau}_{\mathrm{ext}}, \tag{17}$$

where $\boldsymbol{K}_r = \text{diag}\{k_{r,i}\} > \boldsymbol{I}$. It can be shown that such a law is equivalent to scaling of the robot dynamics by $\boldsymbol{K}_v^{-1}$. The closed loop dynamics become

$$\underbrace{\boldsymbol{K}_v^{-1}\boldsymbol{M}(\boldsymbol{q})}_{\boldsymbol{M}'(\boldsymbol{q})}\ddot{\boldsymbol{q}} + \boldsymbol{K}_v^{-1}\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{\tau}_{\text{ext}} = \boldsymbol{0}, \tag{18}$$

where $\boldsymbol{M}(\boldsymbol{q}) > \boldsymbol{M}'(\boldsymbol{q})$ holds component wise.

Admittance Reflex Reference trajectory modification via an admittance type strategy that uses the measurement or estimation of $\boldsymbol{\tau}_{\text{ext}}$ can easily e.g. be realized via

$$\boldsymbol{q}_d(t) = -\int_{t_c}^{T} \boldsymbol{K}_a \boldsymbol{r}\, \mathrm{d}t, \tag{19}$$

where $\boldsymbol{K}_a = \text{diag}\{k_{a,i}\} > \boldsymbol{I}$. With this scheme that requires no control switching the robot quickly drives away from the external torque source and decreases the contact forces till they decay to zero.

4 Conclusion

Physical safety in robotics has become a central discipline in pHRI over the last decade. This is due to the significant progress made in the fields of mechatronics, interaction control, motion planning, and 3D sensing towards highly integrated and sensorized lightweight systems that are able to physically interact with their surrounding. Clearly, the rise of a new generation of commercial robots capable of safe physical interaction has also contributed to the large interest in the field. The robotics research and industrial community expects these systems to open up new markets and to push robotics further towards domestic applications that may also involve even more complex and possibly mobile manipulators. However, despite this recent success in research and also in the commercialization of assistance robots, there are many open research questions that need to be tackled before this class of systems can become a commodity not only in early adopter industrial applications but also on a broader scale: in particular continuing the road towards safe robotics by tightly coupling injury biomechanics and safe interaction control with lightweight and compliant robot design will further push the boundaries and build the foundation of pHRI.

Acknowledgment

I would like to thank Simon Haddadin, Alessandro De Luca, Alin Albu-Schäffer, and Nico Mansfeld for their highly valued collaboration over the last years. Parts of this chapter summarize previously published joint work.

References

1. Haddadin, S., Albu-Schäffer, A., Hirzinger, G.: Requirements for safe robots: Measurements, analysis & new insights. Int. J. of Robotics Research **28**(11-12) (2009) 1507–1527
2. Bicchi, A., Tonietti, G.: Fast and soft arm tactics: Dealing with the safety-performance trade-off in robot arms design and control. IEEE Int. Conf. on Robotics and Automation Mag. **11** (2004) 22–33
3. Zinn, M., Khatib, O., Roth, B.: A new actuation approach for human friendly robot design. Int. J. of Robotics Research **23** (2004) 379–398
4. Haddadin, S., Albu-Schäffer, A., Hirzinger, G.: Safety evaluation of physical human-robot interaction via crash-testing. Robotics: Science and Systems Conference (2007) 217–224
5. Oberer, S., Schraft, R.D.: Robot-dummy crash tests for robot safety assessment. In: IEEE Int. Conf. on Robotics and Automation. (2007) 2934–2939
6. Haddadin, S., Albu-Schäffer, A., Haddadin, F., Roßmann, J., Hirzinger, G.: Study on soft-tissue injury in robotics. IEEE Robotics Automation Mag. **18**(4) (2011) 20–34
7. Haddadin, S., Haddadin, S., Khoury, A., Rokahr, T., Parusel, S., Burgkart, R., Bicchi, A., Albu-Schäffer, A.: On making robots understand safety: Embedding injury knowledge into control. Int. J. of Robotics Research **31** (2012) 1578–1602
8. Haddadin, S.: Towards Safe Robots - Approaching Asimov's 1st Law. Volume 90 of Springer Tracts in Advanced Robotics. Springer (2014)
9. Schneider, D., Nahum, A.: Impact studies of facial bones and skull. SAE Paper No.720965, Proc. 16th Stapp Car Crash Conference (1972) 186–204
10. Nahum, A.M., Gatts, J.D., Gadd, C.W., Danforth, J.: Impact tolerance of the skull and face. In: SAE Paper No.680785, Stapp Car Crash Conf. (1968)
11. Allsop, D., Perl, T.R.and Warner, C.: Force/deflection and fracture characteristics of the temporo-parietal region of the human head. SAE Trans. (1991) 2009–2018
12. Cormier, J., Manoogian, S., Bisplinghoff, J., Rowson, S., Santago, A., McNally, C., Duma, S., Bolte Iv, J.: The tolerance of the nasal bone to blunt impact. In: Annals of Advances in Automotive Medicine/Annual Scientific Conference. Volume 54. (2010) 3
13. Delye, H., Verschueren, P., Depreitere, B., Verpoest, I., Berckmans, D., Vander Sloten, J., Van Der Perre, G., Goffin, J.: Biomechanics of frontal skull fracture. J. of Neurotrauma **24**(10) (2007) 1576–1586
14. Nyquist, G.W., Cavanaugh, J.M., Goldberg, S.J., King, A.I.: Facial impact tolerance and response. SAE Paper No.861896, Proc. 30th Stapp Car Crash Conference (1986) 733–754
15. Allsop, D., Warner, C., Wille, M., Schneider, D., Nahum, A.: Facial impact response - a comparison of the Hybrid III dummy and human cadaver. In: SAE Paper No.881719, Stapp Car Crash Conf. (1988) 781–797
16. Hodgson, V., Thomas, L.: Comparison of head acceleration injury indices in cadaver skull fracture. In: SAE Paper No710854, Stapp Car Crash Conf. (1971) 299–307
17. Nahum, A.M., Smith, R.W.: An experimental model for closed head impact injury. In: SAE Paper No.760825, Stapp Car Crash Conf. (1976)
18. Haddadin, S., Albu-Schäffer, A., Frommberger, M., Rossmann, J., Hirzinger, G.: The "DLR crash report": Towards a standard crash-testing protocol for robot safety - part I+II: Results & discussions. IEEE Int. Conf. on Robotics and Automation (2009) 280–287 + 2663–2670

19. Kroell, C.K., Schneider, D.C., Nahum, A.M.: Impact tolerance and response of the human thorax I. In: SAE Paper No. 710851, Stapp Car Crash Conference. (1971)
20. Kroell, C., Scheider, D., Nahum, A.: Impact tolerance and response of the human thorax II. In: SAE Paper No.741187, Stapp Car Crash Conference. (1974) 383–457
21. Patrick, L.: Impact force deflection of the human thorax. SAE Paper No.811014, Proc. 25th Stapp Car Crash Conference (1981) 471–496
22. Nahum, A.M., Gadd, C.W., Schneider, D.C., Kroell, C.: Deflection of the human thorax under sternal impact, sae technical paper 700400. In: Int. Automot. Saf. Conf. (1970)
23. Cavanaugh, J., Nyquist, G., Goldberg, S., King, A.: Lower abdominal impact tolerance and response. In: SAE Paper No.861878, Stapp Car Crash Conf. (1986)
24. Duma, S., Schreiber, P., McMaster, J., Crandall, J., Bass, C., Pilkey, W.: Dynamic injury tolerances for long bones of the female upper extremity. Int. Research Council on Biomechanics of Injury (IRCOBI1998) (1998) 189–201
25. Duma, S., Crandall, J., Hurwitz, S., Pilkey, W.: Small female upper extremity interaction with the deploying side air bag. In: SAE Paper No.983148, Stapp Car Crash Conf. (1998) 47–63
26. Spadaro, J., Werner, F., Brenner, R., Fortino, M., Fay, L., Edwards, W.: Cortical and trabecular bone contribute strength to the osteopenic distal radius. J. of Orthopaedic Research **12** (1994) 211–218
27. Khatib, O.: Inertial properties in robotic manipulation: An object-level framework. Int. J. of Robotics Research **14**(1) (1995) 19–36
28. Lobdell, T., Kroell, C., Scheider, D., Hering, W.: Impact response of the human thorax. Symp. on Human Impact Response (1972) 201–245
29. Ruedi, T.P., Murphy, W.M., et al.: AO principles of fracture management. Volume 1. AO Publishing (2007)
30. Lau, I., Viano, D.: Role of impact velocity and chest compression in thoraic injury. Avia. Space Environ. Med. **56** (1983) 16–21
31. ISO12100:2010: Safety of machinery – general principles for design – risk assessment and risk reductions (International Organization for Standardization, Geneva 2010)
32. ISO13849-1:2006: Safety of machinery – safety-related parts of control systems – part 1: General principles for design (International Organization for Standardization, Geneva 2006)
33. ISO13855:2010: Safety of machinery – positioning of safeguards with respect to the approach speeds of parts of the human body (International Organization for Standardization, Geneva 2010)
34. ISO10218-1:2011: Robots and robotic devices – safety requirements for industrial robots – part 1: Robots (International Organization for Standardization, Geneva 2011)
35. ISO/TS15066: Robots and robotic devices – Collaborative robots (International Organization for Standardization, unpublished)
36. ISO13482:2014: Robots and robotic devices – safety requirements for personal care robots (International Organization for Standardization, Geneva 2014)
37. Hogan, N.: Impedance control: An approach to manipulation: Part I - theory, Part II - implementation, Part III - applications. J. of Dynamic Systems, Measurement and Control **107** (1985) 1–24
38. Craig, J., Raibert, M.: A systematic method for hybrid position/force control of a manipulator. IEEE Computer Software Applications Conf. (1979) 446–451

39. Yang, C., Gowrishankar, G., Haddadin, S., Parusel, S., Albu-Schäffer, A., Burdet, E.: Human like adaptation of force and impedance in stable and unstable interactions. IEEE Trans. on Robotics **27**(5) (2010) 918–930
40. Stemmer, A., Albu-Schäffer, A., Hirzinger, G.: An analytical method for the planning of robust assembly tasks of complex shaped planar parts. In: IEEE Int. Conf. on Robotics and Automation. (2007) 317–323
41. Hogan, N.: On the stability of manipulators performing contact tasks. IEEE Int. Conf. on Robotics and Automation **4**(6) (1988) 677–686
42. Ott, C., Mukherjee, R., Nakamura, Y.: Unified impedance and admittance control. In: IEEE Int. Conf. on Robotics and Automation. (2010) 554–561
43. Kurfess, T.R.: Robotics and Automation Handbook. CRC press (2010)
44. Caccavale, F., Natale, C., Siciliano, B., Villani, L.: Six-dof impedance control based on angle/axis representations. IEEE Trans. on Robotics and Automation, **15**(2) (1999) 289–300
45. Sentis, L., Khatib, O.: Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. Int. J. of Humanoid Robotics (2005) 505–518
46. Dietrich, A., Wimböck, T., Albu-Schäffer, A.: Dynamic whole-body mobile manipulation with a torque controlled humanoid robot via impedance control laws. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems. (2011) 3199–3206
47. Albu-Schäffer, A., Ott, C., Frese, U., Hirzinger, G.: Cartesian impedance control of redundant robots: Recent results with the DLR-light-weight-arms. In: IEEE Int. Conf. on Robotics and Automation. Volume 3. (2003) 3704–3709
48. Albu-Schäffer, A., Ott, C., Hirzinger, G.: A unified passivity-based control framework for position, torque and impedance control of flexible joint robots. Int. J. of Robotics Research **26** (2007) 23–39
49. Zollo, L., Siciliano, B., De Luca, A., Guglielmelli, E., Dario, P.: Compliance control for an anthropomorphic robot with elastic joints: Theory and experiments. J. of Dynamic Systems, Measurement, and Control **127**(3) (2005) 321–328
50. Platt Jr., R., Abdallah, M., Wampler, C.: Multiple-priority impedance control. In: IEEE Int. Conf. on Robotics and Automation. (2011) 6033–6038
51. Stramigioli, S.: Modeling and IPC control of interactive mechanical systems: a coordinate-free approach. Springer-Verlag New York, Inc. (2001)
52. Suita, K., Yamada, Y., Tsuchida, N., Imai, K., Ikeda, H., Sugimoto, N.: A failure-to-safety "kyozon" system with simple contact detection and stop capabilities for safe human - autonomous robot coexistence. In: IEEE Int. Conf. on Robotics and Automation. (1995) 3089–3096
53. Yamada, Y., Hirasawa, Y., Huang, S., Umetani, Y., Suita, K.: Human-robot contact in the safeguarding space. IEEE/ASME Trans. on Mechatronics **2**(4) (1997) 230–236
54. Takakura, S., Murakami, T., Ohnishi, K.: An approach to collision detection and recovery motion in industrial robot. In: Annual Conference of IEEE Industrial Electronics Society. (1989) 421–426
55. Morinaga, S., Kosuge, K.: Collision detection system for manipulator based on adaptive impedance control law. In: IEEE Int. Conf. on Robotics and Automation. (2003) 1080–1085
56. Kosuge, K., Matsumoto, T., Morinaga, S.: Collision detection system for manipulator based on adaptive control scheme. Trans. of the Soc. of Instrument and Control Engineers **39** (2003) 552–558
57. Lumelsky, V., Cheung, E.: Real-time collision avoidance in teleoperated whole-sensitive robot arm manipulators. IEEE Trans. on Systems, Man and Cybernetics **23**(1) (1993) 194–203

58. Strohmayr, M.: Artificial Skin in Robotics. PhD thesis, Karlsruhe Institute of Technology (2012)
59. De Maria, G., Natale, C., Pirozzi, S.: Force/tactile sensor for robotic applications. Sensors and Actuators A: Physical **175** (2012) 60–72
60. Dahiya, R., Mittendorfer, P., Valle, M., Cheng, G., Lumelsky, V.: Directions toward effective utilization of tactile skin: A review. IEEE Sensors J. **13**(11) (2013) 4121–4138
61. De Luca, A., Albu-Schäffer, A., Haddadin, S., Hirzinger, G.: Collision detection and safe reaction with the DLR-III lightweight manipulator arm. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems. (2006) 1623–1630
62. De Luca, A., Mattone, R.: Actuator fault detection and isolation using generalized momenta. In: IEEE Int. Conf. on Robotics and Automation. (2003) 634–639
63. Kuntze, H.B., Frey, C., Giesen, K., Milighetti, G.: Fault tolerant supervisory control of human interactive robots. In: IFAC Workshop on Advanced Control and Diagnosis. (2003) 55–60
64. Haddadin, S., Albu-Schäffer, A., Luca, A.D., Hirzinger, G.: Collision detection & reaction: A contribution to safe physical human-robot interaction. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems. (2008) 3356–3363
65. Parusel, S., Haddadin, S., Albu-Schäffer, A.: Modular state-based behavior control for safe human-robot interaction: A lightweight control architecture for a lightweight robot. In: IEEE Int. Conf. on Robotics and Automation. (2011) 4298–4305
66. Mansfeld, N., Haddadin, S.: Reaching desired states time-optimally from equilibrium and vice versa for visco-elastic joint robots with limited elastic deflection. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems. (2014) 3904–3911

In-circuit Error Detection with Software-based Error Correction – An Alternative to TMR*

Gökçe Aydos[1,2]
PhD advisor: Görschwin Fey

[1] DLR, Institute of Space Systems, Robert-Hooke-Str. 7, 28359 Bremen Germany
[2] University of Bremen, Institute of Computer Science,
Bibliothekstr. 1, 28359 Bremen Germany
goekce@cs.uni-bremen.de

Abstract. FPGAs are often utilized in space avionics. To protect the FPGA application data against radiation effects in space, data redundancy can be used. A well-known method is to triplicate the circuit and eliminate the erroneous circuit output with a local voter (TMR). Alternatively, in-circuit error detection with software-based error correction can be used, if the FPGA works as a co-module next to a processor running the mission software. In this work, we present an implementation of this method on a commonly used spacecraft data handling architecture.

1 Introduction

Field-programmable gate arrays (FPGAs) are often utilized in space avionics. FPGAs involved in mission critical applications implement fault tolerance mechanisms. One of the reasons is the ionizing radiation in space, which can flip stored bits in *flip-flops* (FFs). In the best case, the flipped bits are masked and overwritten in next cycles, having no effect on the system. In the worst case, this leads to catastrophic failures [4].

Tolerance against bitflips can be implemented using redundancy in space or time, i.e., by instantiating multiple entities of one circuit (space), or repeating the same operation for multiple clock cycles on one circuit (time), or also by combining the both (spacetime) [3]. Mission critical FPGA applications often use space redundancy, mostly in form of *triple modular redundancy* (TMR).

In TMR, a module, e.g. a FF or a whole circuit, is triplicated and the outputs are connected to a voter, which drives the correct output value in case of a failure in a single module. This facilitates correction of an error in the same clock cycle, i.e., by only using space redundancy. In presence of tight space constraints, this overhead can turn into a hurdle for fulfilling the design timing closure and area requirements. Alternatively, if the overhead of time redundancy is feasible for an application, the redundancy in space can be reduced and, in return, the redundancy in time dimension can be increased. We propose an instantiation of

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_10,

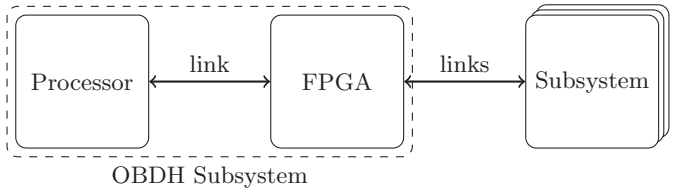


Fig. 1. Simplified model of a typical OBDH architecture.

this approach where error detection is done in space, and error correction in time, only in case of an error. In a system constellation with a processor and an FPGA, the on-demand time redundancy can be easily implemented in software. We call this method *in-circuit error detection with software-based error correction*. In the following, a common spacecraft *on-board data handling* (OBDH) subsystem architecture and an implementation of this method on this architecture will be shown.

2 Application to a Typical OBDH Architecture

The OBDH subsystem of a spacecraft typically handles the communication from the ground station to other subsystems. A simplified model of a typical OBDH architecture is shown on Fig. 1.

The processor runs the mission software and the FPGA implements interface protocol circuits required by various subsystems on board of a satellite. The processor uses the FPGA for communicating with the subsystems.

The FPGA implements the memory-mapped interfaces for the subsystems. Consequently, from the software point of view, the FPGA is a remote memory. To access the remote memory, the software sends memory requests to the FPGA and the FPGA replies every request with a response. If a particular request is not responded (within a timeout), then the software can retry the last request. There are two kinds of memory requests, write and read. Every request can contain memory accesses to particular memory addresses.

In our model, the FPGA design to which the fault tolerance method is not applied, consists of three circuits (A), (B) and (C), shown as white boxes in Fig. 2. (A) stores the requests from the software and responses sent from (B). (B) transforms the requests from the software to actual memory signals for (C).

We assume that the processor, the subsystems, and the circuits (A) and (C) are reliable, i.e., immune against bitflips. So, only (B) has to be hardened.

The gray boxes in Fig. 2 show the circuits for error detection and handling after hardening the design. The error detection circuit checks for data integrity in (B) to detect bitflips, e.g., by using *concurrent error detection* (CED) [2]. An example for CED is to generate parity for every register in (B) and check for data integrity in the next cycle. In case of detected bitflips, the error handling module engages the countermeasures. In this implementation, the error handling

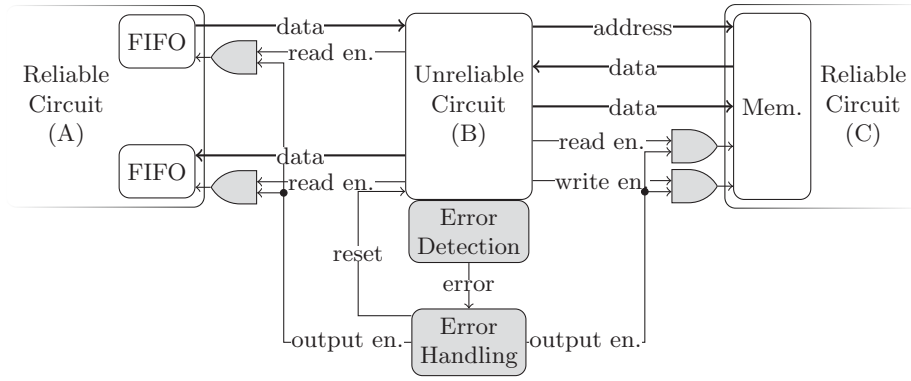


Fig. 2. In-circuit error detection applied on the unreliable circuit (B).

module masks the output signals of (B) and resets it. The masking occurs in one clock cycle, so a fault in (B) does not propagate to the neighboring circuits, i.e., the circuit is isolated.

Through the specified timeout, the software is always aware of a failure in (B). Upon failure of (B), a request is repeated and no request gets lost.

The shown hardening method is useful on circuits like protocol converters, where the circuit acts as an intermediate module. If the circuit interface includes control signals (e.g., write/read enable in Fig 2) which are maskable, then the error can be masked using a relatively short path compared to correcting the error by resetting the circuit.

Like TMR, the method can be applied after the behavioral synthesis on the register transfer level, therefore it is transparent to the application. If the communication protocol between the software and the circuit permits a timeout, then the method is also transparent to the software.

This method concentrates on the circuit bits of an FPGA application and not on the configuration bits, where the FPGA application itself is stored. It is crucial to protect also the configuration bits from the ionizing radiation, if an SRAM-based FPGA is used. If a flash-based FPGA is used, the bitflips on the configuration are negligible [1].

References

1. Battezzati, N., Sterpone, L., Violante, M.: Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications, chap. 7. Springer (2011)
2. Mitra, S., McCluskey, E.J.: Which concurrent error detection scheme to choose? In: International Test Conference Proceedings. pp. 985–994. IEEE (2000)
3. Nicolaidis, M.: Time redundancy based soft-error tolerance to rescue nanometer technologies. In: 17th IEEE VLSI Test Symposium. pp. 86–94 (1999)
4. Petersen, E.: Single Event Effects in Aerospace, chap. 1. John Wiley & Sons (2011)

Behavior Driven Development for Tests and Verification*

Melanie Diepenbeck
PhD advisor: Rolf Drechsler

Group of Computer Architecture, University of Bremen

Abstract. Nowadays, hardware is usually tested and verified at post-design time. The bottom line is that more effort is spend in the validation phases than in the implementation, because it is harder to fix bugs in later design stages than during the implementation of the design. In contrast, test-first approaches such as test driven development (TDD) have become increasingly important for software development. Behavior driven development (BDD) extends TDD by using natural language style scenarios to describe tests. But both approaches miss formal verification methods which are very important in hardware design. This research project presents a new approach based on BDD that combines testing and verification seamlessly.

1 Introduction

In traditional hardware design flows, testing and verifying the design is often more time- and labour-consuming than the actual implementation. Historically, testing and verification are usually applied after most of the design has been implemented. The result is that mandatory major design changes due to serious bugs lead to long design cycles so that time-to-market constraints cannot be met. Hence, the validation of hardware designs should start as early as possible to discover these bugs at design-time where they can be fixed easily.

In the software domain, *test driven development* (TDD, [1]) has become a popular approach that enables short design cycles that includes validation right from the start. Basically in TDD, testing and implementation take turns, whereas after a new test case is introduced the implementation is extended to satisfy the new test case. *Behaviour driven development* (BDD, [2]) enhances the idea of TDD with tests written in natural language that are called *scenarios*. Step by step these scenarios are associated with executable test code, thereby linking the specification and the implementation.

Considering safety critical hardware design, testing is insufficient. Simulating BDD scenarios rarely covers the whole input and state space. Therefore, some bugs might be missed that could be detected with formal methods such as [3, 4]. However, formal methods experts are often needed to define these properties since a high level of mathematical expertise is needed to understand and apply properties to a design.

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_11, © Springer Fachmedien Wiesbaden 2015

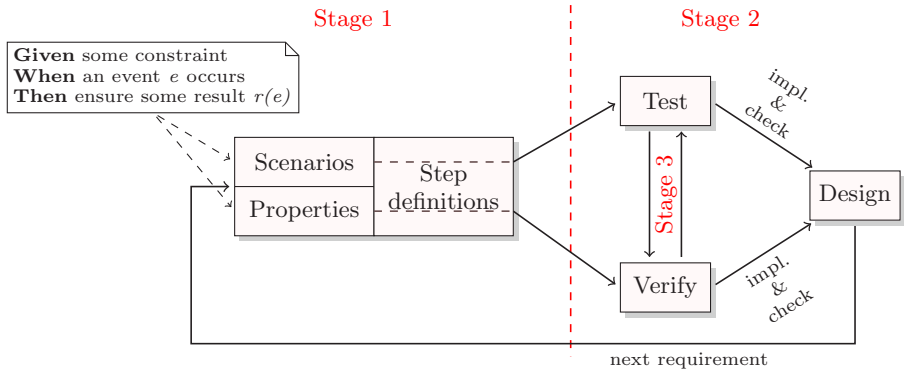


Fig. 1. Proposed Flow

The research project presented in this paper investigates a new hardware design flow that completes the BDD method by complementing tests with formal verification techniques in a flexible and agile way. The traditional test driven flow is enhanced by integrating formal verification as a second driver of the implementation.

2 New Design Flow

The contribution of this research, partly presented in [5, 6], consists of three main ideas; (1) a customized BDD flow that is suitable for circuit design using scenarios and properties alike, (2) generated properties from textual scenarios that can be used for formal verification and (3) generated tests from textual properties that are helpful in explaining complex properties. The advanced BDD flow has three main stages, that are depicted in Figure 1 and are described in the following sections.

Stage 1: Acceptance Tests and Properties

In Stage 1, the features of the individual hardware components are described by *scenarios* or *properties* in natural language using the *Given-When-Then* sentence structure as seen in Figure 1. Each sentence is called a *step*. Writing properties additionally to acceptance tests allows keeping all requirements in a single consistent document throughout the whole design flow [6].

Stage 2: BDD for Tests and Verification

The design is implemented in Stage 2. Following the test-first principle, the implementation is developed iteratively by specifying *step definitions* for each step of Stage 1 and then implementing the design parts that are required by the steps. A step definition contains fragments of test code that describes the behavior of a single step in a scenario. All steps of a scenario are used to compose a testbench that drives the design under verification (DUV).

Stage 3: Generating Properties and Tests

BDD scenarios usually only consider few selected test input data and never cover a scenario exhaustively. To cover the whole input space formal properties come in handy. In order to ease the effort of writing properties, scenarios can be automatically generalised to properties. The resulting property can then be verified using existing state-of-art model checkers. The property is obtained by capturing the verification intent of a scenario and map the underlying test code using the *Given-When-Then* sentence structure to an implication property [5].

One widely acknowledged disadvantage of formal methods is that it can take too long to find a solution. Furthermore, complex properties may be hard to understand, so that examples are necessary. Therefore, this approach also allows to generate scenarios from properties that can help explain the properties and investigate the design further. The generated scenario consists mostly of existing textual steps, such that the validation intent can be easily comprehended by anyone involved in the design flow. Such a scenario is generated by creating a witness for the verification intent of the original property. This witness is then translated to a textual scenario using existing steps.

3 Conclusion

In this paper, a new BDD flow has been presented that complements testing with verification in seamless manner. Furthermore, when tests are not sufficient enough to check the design, they can be generalized to properties, and otherwise, when formal properties do not result a solution in appropriate time, a human-understandable scenario can be generated for further inspections of the design.

References

1. Beck, K.: Test Driven Development. By Example. Addison-Wesley Longman, Amsterdam (November 2003)
2. Wynne, M., Hellesøy, A.: The Cucumber Book: Behaviour-Driven Development for Testers and Developers. The Pragmatic Bookshelf (January 2012)
3. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic Model Checking without BDDs. In: Tools and Algorithms for Construction and Analysis of Systems, Springer (March 1999) 193–207
4. Bradley, A.R.: SAT-based model checking without unrolling. In Jhala, R., Schmidt, D.A., eds.: VMCAI. Volume 6538 of Lecture Notes in Computer Science., Springer (2011) 70–87
5. Diepenbeck, M., Soeken, M., Grosse, D., Drechsler, R.: Behavior driven development for circuit design and verification. In: Int'l Workshop on High Level Design Validation and Test Workshop (HLDVT). (Nov 2012) 9–16
6. Diepenbeck, M., Kühne, U., Soeken, M., Drechsler, R.: Behaviour driven development for tests and verification. In: Tests and Proofs. Springer (2014) 61–77

Semantic Object Recognition Based on Qualitative Probabilistic Spatial Relations*

Malgorzata Goldhoorn
PhD Advisor: Frank Kirchner

University of Bremen, Department of Computer Science,
Bibliothekstr. 1, 28359 Bremen, Germany
malgorzata.goldhoorn@informatik.uni-bremen.de

Abstract. Intelligent systems able to perform everyday tasks in human living environments are going to play an important role in the future. Especially service and assistive robots, which could take over tasks such as fetch and carry, would be of great use. However, dealing with objects in natural environments is not a trivial but rather very challenging task. The robot has to extract the objects from noisy sensor data and give them a meaningful and correct description. The goal of this thesis is to develop an approach for robust semantic object recognition, which can be used for such purposes. In our approach, we take advantage of the spatial contextual information about objects' co-occurrences to perform a robust object recognition. Our approach is unique in that it uses spatial semantics in a probabilistic manner. We also develop a new representation of this information, termed *Spatial Potential Fields*.

1 Introduction

In recent years, the development of low-cost depth-image sensors and new algorithms for depth camera processing has contributed to the improvement of robot perception systems significantly. Nevertheless, current approaches for robot perception have their limitations, such as weakness against occlusions and are based primarily on extracted object features to recognize the given object class. Our goal is to develop a recognition system in which the contextual knowledge about typical spatial relations between objects is used. This knowledge can help a robot to recognize or classify an object reliably, even if the perception has a degree of uncertainty. In this work we extend the qualitative spatial object relations to probabilistics and introduce a new representation form for those contextual relations. Additionally, we make use of very strong domain unspecific a priori knowledge about the spatial relations in which objects are likely to be.

2 Related Work

Semantic object recognition and object search have become popular topics in robotics recently. One of the current works in this field is the semantic labelling

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_12, © Springer Fachmedien Wiesbaden 2015

approach from Annand et al. [1]. The authors used contextual information about geometrical object features and relations between them to perform labelling of object classes. Adymir et al. [2] describe an approach for active visual search, in which the topological relations between objects are used to create a search action and find the objects. Similar to the work in [1] the next-best-view algorithm is applied to deal with occlusion. Haider et al. [3] propose the use of a context model to improve the detection result of other related objects in the scene. In this approach object co-occurrence information is used to perform recognition of the object category. However, most works do not take account of the diversity of the possible objects' relations and try to recognize the objects mostly exclusively based on their appearance.

3 Semantic Object Recognition

In order to recognize objects in large human environments, the system must be able to reason about possible object locations before the recognition process can take place. To achieve this, we would like to equip the system with environmental spatial semantics which can help the robot to optimise its search action. In our method, a priori knowledge about general spatial object relations is used. This knowledge can be understood as a (qualitative) spatial context for the object recognition. Additionally, the spatial relations serve as a heuristic for the object search.

3.1 Probabilistic Spatial Relations

The main contribution in our work is the method in which the typical knowledge about qualitative spatial relations between objects is used in a probabilistic manner to improve the semantic perception process. The probabilistic extension of the relations should improve the recognition, which is often erroneous due to the lack of knowledge or the presence of ambiguities. In the real environment an object might be occluded or the data provided by the sensors might be noisy. Moreover, as soon as part of the information is missing, the reasoning cannot be done by using logic exclusively. Through probability it is possible to describe that one specific object can be located on different places and can have several relations to other objects with certain probabilities. What is more, given the arrangement of the objects, spatial reasoning could tell the robot that the object classified as a mug is more likely to be a computer mouse given its location to the right of the flatscreen. The probability value may change according to the given environment and the robot could learn this a priori knowledge from experience. The relations have the general form:

$$\beta : \varphi \times \eta \times \eta \rightarrow [0; 1] \tag{1}$$

Where $\eta = \{\eta_1, ..., \eta_n\}$ describes a given object class and $\varphi = \{\varphi_1, ..., \varphi_n\}$ contains the possible qualitative relations. Based on the equation (1) the probabilistic qualitative spatial relations can be defined, as shown in examples (2-3).

$$\beta(on, table, mug) = 0.8, \quad \beta(near, table, fridge) = 0.6 \tag{2}$$

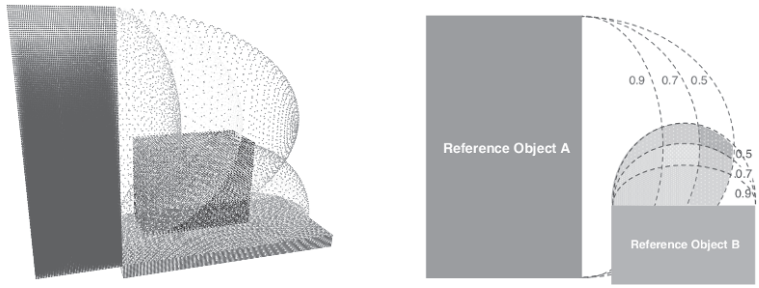


Fig. 1. Left: two potential fields, each in form of three half-ellipsoids with different field intensities and a sought object in the middle. Right: illustration of the spatial potential field concepts for *near* and *above* relations (shown in 2D)

$$\beta(rightOf, mouse, screen) = 0.7, \quad \beta(above, table, shelf) = 0.6 \tag{3}$$

3.2 Spatial Potential Fields

The *Spatial Potential Fields (SPFs)* [4] are used to model and calculate the qualitative topological relations in a probabilistic manner and represent this information on the robot's world model. Our idea was inspired by the potential fields method which has been used in robot navigation for obstacle avoidance [5]. The fields are geometrical spheroids which are aligned to the objects. They may have different form depending on the relations type. An example of such a field is shown in figure (Fig.1). By means of the *SPFs*, the degree of intensity for a given qualitative relation can be calculated and, in turn, used to find the location likely to contain the object.

References

1. Anand, A., Koppula, H.S., Joachims, T., Saxena, A.: Contextually guided semantic labeling and search for three-dimensional point clouds. The International Journal of Robotics Research (2012)
2. Aydemir, A., Sjoo, K., Folkesson, J., Pronobis, A., Jensfelt, P.: Search in the real world: Active visual object search based on spatial relations. In: Robotics and Automation (ICRA), 2011 IEEE International Conference on. (2011) 2818–2824
3. Ali, H., Shafait, F., Giannakidou, E., Vakali, A., Figueroa, N., Varvadoukas, T., Mavridis, N.: Contextual object category recognition for rgb-d scene labeling. Robotics and Autonomous Systems **62** (2014) 241 – 256
4. Goldhoorn, M., Hartanto, R.: Semantic perception using spatial potential fields. In: The 9th International Workshop on Cognitive Robotics (CogRob) of the 21st European Conference on Artificial Intelligence (ECAI). (2014)
5. Borenstein, J., Koren, Y.: The vector field histogram-fast obstacle avoidance for mobile robots. IEEE J. Robotics and Automation, **7** (1991) 278–288

Constraint-based Handling of Component Networks*

Matthias Goldhoorn
PhD advisor: Frank Kirchner

University of Bremen, Department of Computer Science,
Bibliothekstr. 1, 28359 Bremen, Germany
Matthias.Goldhoorn@informatik.uni-bremen.de

Abstract. The ability to reconfigure running complex component networks in a robust manner is the next challenge in robotic software frameworks. The increasing complexity of hard- and software leads to an increasing systems' complexity. The need to have a non-static version of a collection of components that form a behaviour, for one point in time, is caused not least by limiting resources of processing power. It is not possible to have all algorithms running at each point in time. Therefore, the requirement to manage these component networks came up. This work presents a new constraint-based approach to tackle the problem. The aim of this work is to model a constraint-based system that is able to handle state of the art robotic component networks, without increasing the complexity for the system handling and system instantiation.

Keywords: component networks, reconfiguration, system behaviour, planning, robotics

1 Introduction

Robots are becoming more complex, from both a software and hardware point of view. As hardware is getting more capable, more complex algorithms are being developed. However, the drawback is that there is no single algorithm that can handle all problems or requirements to control an autonomous robot system. Therefore, one of the remaining big challenges is the integration of all these single components and/or modules in a complex system (Fig. 1). Defining clean and powerful interfaces can already be achieve by using frameworks like ROS [1] or Rock [2], but the managing of those components is mostly still a open question.

There are solutions available, but none of the known tools seems to be suitable to satisfy the user's needs, from our point of view. The known systems have several drawbacks from either the engineering point of view (they are hard to use), or they are limited in their capabilities, or both.

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_13,

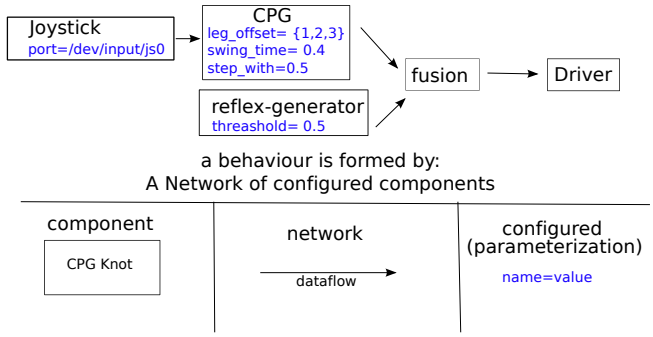


Fig. 1. An abstract example of a component network

2 Related Work

There is a limited number of ongoing work that focuses on the handling of component networks. There are, on the one hand, classical plan managers like *roby/syskit*, mixed planners like *T-Rex* [3], and typical planners like the *knowrob* [4] system. Each system has different advantages and drawbacks.

The most used solution for component based robotics systems seems to be ROS. But its modularization capabilities are limited. Most of the ROS users are following the task state pattern approach for system control [5] which is not suitable to be used in future systems with growing complexity because there is no way to reuse these realizations. This is caused by a missing abstraction layer. *Knowrob* presents a planning based solution which tries to connect several data sources to solve the *symbol grounding problem* [6]. But still the association to concrete algorithms is unsolved or domain specifically hard wired.

3 Constraint-based Approach

In a robotic system there are often multiple algorithms that can fulfil a given job like a trajectory follower. The problem is that often several sensors could be used for these algorithms as data provider. The selection of the right ones is part of the constraint checking process. In our work we are planning to develop a clean constraint-based approach for modelling and handling robotic component based systems. Our goal is to build a seamless integration of a constraint solver for managing component networks. Instead of defining what effect a component has, we are using the component itself as a symbol. The goal is to use components as compositions similar to those described in [7].

The core concept of our work is, in contrast to syskit, the ability to handle ambiguities. Instead of denying ambiguous solutions, we assume that each valid solution is capable to fulfil a requirement. It is the job of the designer to limit the usage by adding constraints to prohibit invalid solutions. This approach enables a constraint solver to search for multiple solutions, and in case of failures, automatically switch to another solution where the failing component is excluded.

We do not have a symbol grounding problem in this kind of setup, each component gets a clear symbol. The job of the constraint solver is to find a right association for algorithms, instead of solving abstract goals. Each *problem* for managing component networks could be transformed to a representation which can be solved by constraint solvers.

References

1. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA workshop on open source software. Volume 3. (2009) 5
2. o. V.: Rock, the Robot Construction Kit (2012) http://www.rock-robotics.org.
3. McGann, C.e.a.: A deliberative architecture for AUV control. IEEE International Conference on Robotics and Automation (2008)
4. Tenorth, M., Beetz, M.: KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. International Journal of Robotics Research (IJRR) **32** (2013) 566 – 590
5. Lütkebohle, I.e.a.: Generic middleware support for coordinating robot software components: The Task-State-Pattern. Journal of Software Engineering in Robotics **2** (2011) 20–39
6. Harnad, S.: The symbol grounding problem. Physica D: Nonlinear Phenomena **42** (1990) 335 – 346
7. Joyeux, S., Albiez, J., et al.: Robot development : from components to systems. In: Control Architecture of Robots. (2011) 1–15

Model-Based Testing Against Complex SysML Models*

Christoph Hilken
PhD advisor: Jan Peleska

University of Bremen, Department of Mathematics and Computer Science,
Bibliothekstr. 1, 28359 Bremen, Germany
chilken@informatik.uni-bremen.de

Abstract. In recent years higher level of abstractions are considered to cope with the complexity in today's system design. Especially, modeling languages such as SysML get increasingly popular in early design phases. Model-based testing allows to use those models for test generation. Test cases are derived from the model and not directly from the specification. In this work a new proposed methodology will be implemented in an existing model-based testing tool. Afterwards this methodology is used to extend the tool to support activities as well as state machines and not only state machines.

Keywords: SysML, Transition Relation, Model Checking, Model-based Testing

1 Introduction

Development of embedded systems is a cumbersome task. Increasing number of components, tighter hardware/software interaction, and the integration of cyper-physical components such as sensors and actuators have led to a significant complexity to be tackled. Verification, validation and testing of such systems is an important task in the development. Designers keep trying to cope with this complexity by higher levels of abstraction such as Register Transfer Level or the Electronic System Level. In the early phases of the design, modeling languages like UML and its profiles SysML and MARTE become increasingly popular. The *OMG Systems Modeling Language* (SysML [4]), for example, provides different descriptions means such as block definition diagrams (for the structure of the system), activity diagrams (for the behavior of operations of the system) or state machines (for control states of a system and their transitions) for specifying the structure and behavior of a system prior to its implementation. Additionally, requirements can be provided as constraints in the *Object Constraint Language* (OCL [5]). Therefore, SysML allows a precise description of the system in an early phase of the development and that can be used for model-based testing.

Instead of deriving test cases directly from the specification, which is a laborious task, especially if the specification or an implementation detail changes, the test cases are derived from a model of the system under test and its environment.

Model-based testing can be structured in the following steps:

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_14,

Modeling: In a first step a model, which is an executable, partial description of the system, usually derived from the specification, has to be created by hand. For this purpose the SysML is used as modeling language.

Generation: Based on the model an abstract test suite and corresponding test oracles are generated.

Compilation: The abstract test suite and test oracles are transformed into executable tests.

Execution: Running the executable tests. Test oracles check test results during and after the test.

The generation of the abstract test suite requires a formal encoding of the behavioral model semantics. This is typically achieved by generating initial state conditions as well as the transition relation of the model. Since SysML provides a multitude of alternative or complementary notations, this poses a significant challenge to the development of corresponding tool support. In this work the existing industrial-strength tool RttTgen [6] will be extended: A new methodology for the generation is introduced to allow an easy extension of the supported description means.

In the next section the proposed methodology is described. The paper closes with a small conclusion and ongoing work.

2 Methodology

The translation from the model to its corresponding transition relation is a cumbersome task, any error will spoil the verification or validation results. In Addition, SysML provides many different description means and for each one a translation to its corresponding transition relation has to be provided. Therefore, a two-step approach is proposed for generating the corresponding transition relation. First, a syntactic model-to-model transformation of the given behavioral descriptions into a unified description of the system's behavior and structure is performed. The unified description consists only of operations with pre- and post-conditions allocated in blocks. Because of this unification, there is only one translation into a transition relation needed, namely the translation of operations with pre- and post-conditions (called model-to-text transformation). Additionally, many semantic variation points can be implemented in the model-to-text transformation and so can be re-used. As a result adding support for new description means is easy: Only the model-to-model transformation has to be implemented.

Figure 1 shows an example of a model-to-model transformation. The behavior of the system components in figure 1a is described by means of state machines. These state machines are replaced by operations in figure 1b and attributes are added to represent implicit informations of the corresponding description means, in this case the active state of the state machine. For generation of the corresponding pre- and post-conditions as well as a more detailed description of the approach please refer to [2].

Besides test generation the transition relation can be used for verification of the model, for example to prove that the specification is free of contradictory requirements (see e.g. [1]) or to check for unwanted behavior to be avoided (see e.g. [7]).

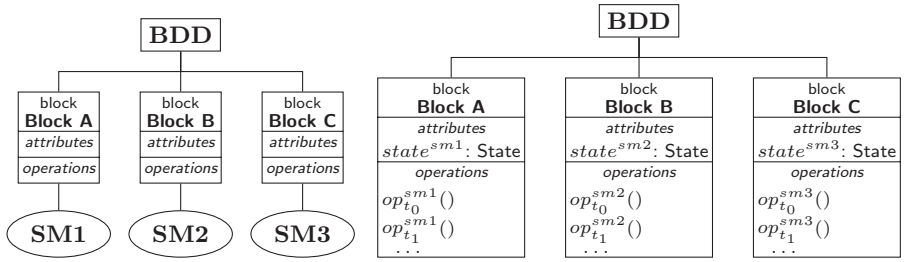


Fig. 1. Proposed model-to-model transformation.

3 Conclusion and Future Work

The proposed two-step approach on the one hand reduces the complexity of the translation into corresponding transition relations, on the other hand new description means can easily be added. Afterwards, the resulting transition relation is used for model-checking and test generation. A basic translation for activities(see [3]) and the proposed two-step approach are already finished. The next step is full support of models consisting of both, state machines and activities.

References

1. Gogolla, M., Kuhlmann, M., Hamann, L.: Consistency, independence and consequences in UML and OCL models. In: Tests and Proofs. pp. 90–104 (Jul 2009)
2. Hilken, C., Peleska, J., Wille, R.: A unified formulation of behavioral semantics for sysml models. In: 3rd International Conference on Model-Driven Engineering and Software Development (2015)
3. Hilken, C., Seiter, J., Wille, R., Kühne, U., Drechsler, R.: Verifying consistency between activity diagrams and their corresponding ocl contracts. In: Forum on Specification & Design Languages (2014)
4. Object Management Group: OMG Systems Modeling Language (OMG SysML^{TM}). Tech. rep., Object Management Group (2010), OMG Document Number: formal/2010-06-02
5. Object Management Group: OMG Object Constraint Language (OCL). Tech. rep., Object Management Group (2012), OMG Document Number: formal/2012-01-01
6. Peleska, J.: Industrial-strength model-based testing - state of the art and current challenges. In: Petrenko, A.K., Schlingloff, H. (eds.) Proceedings Eighth Workshop on Model-Based Testing, Rome, Italy, 17th March 2013. Electronic Proceedings in Theoretical Computer Science, vol. 111, pp. 3–28. Open Publishing Association (2013)
7. Soeken, M., Wille, R., Drechsler, R.: Verifying dynamic aspects of UML models. In: Design, Automation and Test in Europe. pp. 1077–1082. IEEE Computer Society (Mar 2011)

Integrated Model-based Testing and Model Checking with the Benefits of Equivalence Partition Testing*

Felix Hübner
PhD advisor: Jan Peleska

University of Bremen, Department of Mathematics and Computer Science,
Bibliothekstr. 1, 28359 Bremen, Germany
felixh@informatik.uni-bremen.de

Abstract. In safety-critical systems the verification process is one of the most important and most time-consuming tasks. Therefore automated methods are needed to guide the verification process. Typical methods for system verification are model checking and model-based testing. Both methodologies have a lot in common. It seems promising to investigate possible synergies of these two research areas. In this work, the possibilities for the integration of model-based testing with model checking will be investigated. Additionally, a novel model-based testing approach based on equivalence class partitioning has been implemented recently. In this paper a short overview of the implementation is given.

Keywords: Model-based Testing, Model Checking, Partition Testing

1 Introduction

Model-based testing and model checking offer different advantages and are used for different purposes. Model checking can provide full correctness proofs through exhaustive state exploration, but suffers from the state explosion problem. In contrast to that, test suites are rarely able to prove the absence of any remaining errors, but are always feasible, and they still represent the most important verification method in practice. Model checking [1] is used to verify that a model satisfies a specification, which is a set of properties that are often defined in temporal logic, e.g. LTL. Model checking is able to prove (or disprove) that the specification is satisfied globally by the system, which has to be a finite state system in order to guarantee that the model checking algorithms terminate. But in many cases the state explosion problem prevents the application of model checking. In this case abstraction techniques can help to reduce the state space in order to render the usage of model checking algorithms possible. Model-based testing is the method to automatically extract a test suite from a model. This

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_15,

test suite is then applied to the real system (concrete hardware, software, embedded system) and can be used for all kinds of systems, even if the state space is infinite. The general problem of testing is, that neither all possible inputs and combinations of the latter nor infinite behaviours can be checked.

However, in the context of finite state machines (FSM) a variety of complete[1] test methods exist, e.g. [2]. These methods allow to proof the conformance of an implementation to the model and hence allow a statement on the correctness of an implementation by means of testing.

The main contributions of this work are as follows: 1. A novel model-based testing approach has been implemented.[2] This approach provides an automated test data generation algorithm for SysML/UML state machines [4], that is complete under certain restrictions. This leverages the incompleteness of model-based testing by using data abstraction. The implementation is presented in the following section. 2. As shown in the last section of this paper, future work will focus on experimental evaluation of the test strength of the presented strategy. 3. The same equivalence abstraction will be applied to problems of model checking. 4. An integrated framework for model-based testing and model checking will be implemented to combine advantages of both methods.

2 Equivalence Class Partition Testing

This section serves as a brief overview about the implementation of the equivalence class partition testing (ECPT) strategy as proposed in [5]. There it is shown that it is possible to abstract the potentially infinite input domain of a state transition system, in order to derive an FSM abstraction with finite input alphabet. This FSM allows the generation of a finite, but still complete, test suite with respect to a fault domain[3].

Fig. 1 shows the flow of the test data generation for the ECPT approach. A behavioural description of the System-Under-Test modelled by a SysML/UML state machine is first transformed into an internal model representation expressed by a state transition system. This first transformation facilitates the application of the approach to other description languages, whose behavioural semantics can be expressed by a state transition system. Next, an abstraction based on I/O equivalence [7] is performed. This induces an FSM abstraction of the system. On this FSM the well-known W-method [2] is applied, resulting in the test suite. In [6] this novel approach has already been applied to a real-world example from the *European Train Control System (ETCS)*.

[1] Completeness comprises soundness (every correct implementation is accepted) and exhaustiveness (every erroneous implementation is rejected).

[2] The implementation is integrated in the model-based testing component of the industrial strength tool RT-Tester [3].

[3] The fault domain is a collection of systems that fulfill certain properties. The test suite is proven to be complete with respect to this fault domain, i.e. the completeness does only apply for systems that are part of the fault domain. For further information please refer to [6].

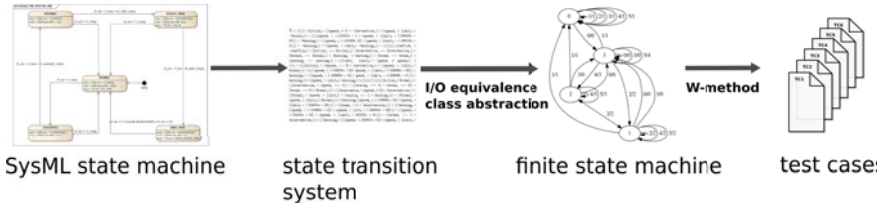


Fig. 1. Flow of the ECPT approach

3 Conclusion and Future Work

The presented ECPT approach is a novel model-based testing technique that is able to automatically generate test suites from SysML/UML state machines, that are complete for a well-defined fault domain. Hence this model-based testing technique is able to allow a statement on the correctness of a System-Under-Test (SUT), contrary to general testing. This statement is only valid, if it can be guaranteed, that the SUT belongs to the fault domain. In general, especially for blackbox tests, this cannot be decided. Hence future work will focus on the experimental evaluation of the test strength of the approach for arbitrary systems. Furthermore, the question how the equivalence classes have to be refined to increase the fault domain (and improve the test strength) will be answered. The use of the I/O equivalence data abstraction in the context of model checking will be evaluated and may result in an integrated framework for model-based testing and model checking.

References

1. Edmund M. Clarke, Orna Grumberg, and Lucent Technologies. *Model Checking*. The MIT Press, Cambridge. Massachusetts, 1999.
2. T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Softw. Eng.*, 4(3):178–187, May 1978.
3. Jan Peleska. Industrial-strength model-based testing - state of the art and current challenges. *Electronic Proceedings in Theoretical Computer Science*, 111:3–28, 2013. arXiv: 1303.1006.
4. Object Management Group. OMG Unified Modeling Language (OMG UML), superstructure, version 2.4.1. Technical report, 2011.
5. Wen-ling Huang and Jan Peleska. Exhaustive model-based equivalence class testing. In *Testing Software and Systems*, number 8254 in Lecture Notes in Computer Science, pages 49–64. Springer Berlin Heidelberg, 2013.
6. Cécile Braunstein, Anne E. Haxthausen, Wen-ling Huang, Felix Hübner, Jan Peleska, Uwe Schulze, and Linh Vu Hong. Complete model-based equivalence class testing for the ETCS ceiling speed monitor. In *Formal Methods and Software Engineering*, number 8829 in Lecture Notes in Computer Science, pages 380–395. Springer Berlin Heidelberg, 2014.
7. Wen-ling Huang and Jan Peleska. Complete model-based equivalence class testing. *International Journal on Software Tools for Technology Transfer*, pages 1–19, 2014.

An SMT-based Approach to analyze Non-Linear Relations of Parameters for Hybrid Systems

Xian Li
PhD advisor: Klaus Schneider

Embedded Systems Group, Department of Computer Science,
University of Kaiserslautern, Germany

Abstract. Deriving constraints over parameters to avoid unexpected system behaviors is extremely important for parametric analysis of hybrid systems. In the long run, our project aims for an SMT-based approach to reveal non-linear relations between parameters for hybrid systems that are specified by parameterized formal models using standard data types (reals, integers and booleans) and affine dynamics. The problem we address is undecidable since the underlying logic consists of boolean combinations of propositional logic atoms as well as atoms from non-linear arithmetic theories over integers and reals with quantifiers. Currently, a symbolic simulation algorithm has been prototypically implemented based on a new developed prototypical constraint solver.

Keywords: SMT, Constraints over Parameters, Parametric Analysis

The hybrid systems discussed in this paper are specified by parameterized formal models supporting standard data types (reals, integers and booleans) and affine dynamics, like *Hybrid Quartz* [1] or HyDI [2]. Our project aims for an SMT-based approach to reveal non-linear relations of parameters for automatic analysis of this kind of models. The problem we address leads to an undecidable satisfiability problem. In [3], we described the satisfiability problem from both the logical and the constraint problems' [4] perspectives.

From the logical point of view, the satisfiability problem that we consider consists of boolean combinations of propositional logic atoms as well as atoms of non-linear arithmetic theories over integers and reals with $\exists$-quantifiers. It is undecidable and strictly included in the combination of the first-order logic of the structure $(\mathbb{R}, +, \cdot, \mathbb{Z}, 0, 1, <)$ and propositional logic.

In the constraint problems' perspective, if we get rid of the boolean variables, then the remaining problem amounts to check whether there exists a solution for a set of Mixed Integer Non-Linear Programming (MINLP) problems. The MINLP problem is one of the most general modeling paradigms in optimization and includes both Non-Linear Programming (NLP) and Mixed Integer Linear Programming (MILP) as subproblems. Linear constraint problems are a subset of NLP problems, while, MILP problems are a subset of MINLP problems.

As shown in Fig. 1, according to the constraint problems they could solve, we classified various available tools for hybrid system analysis, e.g. HySAT [5], MathSAT5 [6], CVC4 [7], Z3 [8], KeYmaera [9], IMITATOR [10], SpaceEx [11],

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_16,

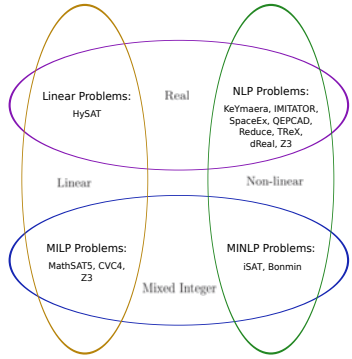


Fig. 1. Tool Classification in Constraint Problems' Perspective

TReX [12], iSAT [13] and dReal [14], together with some algebraic computation tools, e.g. QEPCAD [15], Reduce [16].

Most of the tools cannot handle a single MINLP problem, except for iSAT and some tools for MINLP problems, like Bonmin [17]. In general, a solution for an individual MINLP problem cannot always be found. This might be due to limitations of the tool itself or due to the fact that satisfiability of MINLP problems is undecidable. No solution returned does not mean that no solution exists. Moreover, iSAT and Bonmin are developed in different application areas. It is still unclear which tool performs better to solve the satisfiability problem that we consider.

Therefore, the dual-rail representation is used, where two BDDs represent the three possible values (*true*, *false* or *unknown*) of a node that corresponds to an individual MINLP problem. A new prototypical constraint solver has been developed by integrating Bonmin with a BDD package implemented in our *Averest* system (`www.averest.org`). Based on our new solver, we proposed a counterexample-guided algorithm for symbolic simulation of hybrid systems [3]. The algorithm has been prototypically implemented on top of our *Averest* system. We benefit from our Averest system, since it offers a constantly evolving infrastructure containing tools for compilation, analysis, synthesis, and different techniques for formal verification. It provides algorithms that translate a *Hybrid Quartz* program to a set of guarded actions [18]. The guarded actions are the basis for the system's symbolic representation [1].

The correctness of the prototypical constraint solver is assured by the BDD package and by Bonmin. Its capability is restricted by the two tools as well. After improving the efficiency of the prototypical tool integration, a comparison with iSAT will be conducted by benchmark examples. Moreover, we plan to combine abstraction and linearization techniques to reorganize the system's symbolic representation, so that either MILP problems or NLP problems can be generated to simulate the original MINLP problems. In this way, we avoid the strict requirements for the backend solver, so that various tools can be used, including all those tools that are displayed in Fig. 1. Deriving constraints over parameters to meet the system's specifications could then be achieved by solving the corresponding MILP problems or NLP problems, both of which are decidable.

References

1. Bauer, K., Schneider, K.: From synchronous programs to symbolic representations of hybrid systems. In Johansson, K., Yi, W., eds.: Hybrid Systems: Computation and Control (HSCC), Stockholm, Sweden, ACM (2010) 41–50
2. Cimatti, A., Mover, S., Tonetta, S.: HyDI: A language for symbolic hybrid systems with discrete interaction. In: Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on, IEEE Computer Society (2011) 275–278
3. Li, X., Schneider, K.: A counterexample-guided approach to symbolic simulation of hybrid systems. In: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV), Chemnitz, Germany, In proceeding (2015)
4. Bordeaux, L., Hamadi, Y., Zhang, L.: Propositional satisfiability and constraint programming: A comparative survey. ACM Computing Surveys (CSUR) **38** (2006)
5. Fränzle, M., Herde, C.: HySAT: An efficient proof engine for bounded model checking of hybrid systems. Formal Methods in System Design (FMSD) **30** (2007) 179–198
6. Cimatti, A., Griggio, A., Joost Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Volume 7795 of LNCS., Rome, Italy, Springer (2013) 93–107
7. CVC4. `cvc4.cs.nyu.edu/web/`
8. Z3. `z3.codeplex.com/`
9. Platzer, A.: Logical Analysis of Hybrid Systems – Proving Theorems for Complex Dynamics. Springer (2010)
10. André, É., Fribourg, L., Kühne, U., Soulat, R.: IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In: Proceedings of the 18th International Symposium on Formal Methods (FM'12). Volume 7436 of Lecture Notes in Computer Science., Paris, France, Springer (2012) 33–36
11. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Computer Aided Verification (CAV). Volume 6806 of LNCS., Snowbird, Utah, USA, Springer (2011) 379–395
12. Annichini, A., Bouajjani, A., Sighireanu, M.: TReX: A tool for reachability analysis of complex systems. In: Computer Aided Verification (CAV). Volume 2102 of LNCS., Paris, France, Springer (2001) 368–372
13. Eggers, A., Fränzle, M., Herde, C.: SAT modulo ODE: A direct SAT approach to hybrid systems. In: Automated Technology for Verification and Analysis (ATVA). Volume 5311 of LNCS., Seoul, South Korea, Springer (2008) 171–185
14. Gao, S., Kong, S., Clarke, E.: dReal: An SMT solver for nonlinear theories over the reals. In: Conference on Automated Deduction (CADE). Volume 7898 of LNCS., Lake Placid, NY, USA, Springer (2013) 208–214
15. QEPCAD. `http://www.usna.edu/CS/qepcadweb/B/QEPCAD.html`
16. Reduce. `reduce-algebra.com/`
17. Bonami, P., Biegler, L., Conn, A., CornuéJols, G., Grossmann, I., Laird, C., Lee, J., Lodi, A., Margot, F., Sawaya, N., Wächter, A.: An algorithmic framework for convex mixed integer nonlinear programs. Discret. Optim. 5 (2008) 186–204
18. Schneider, K.: The synchronous programming language Quartz. Internal Report 375, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany (2009)

Analyzing and Simulating Time Descriptions from UML/MARTE CCSL*

Judith Peters
PhD advisor: Rolf Drechsler

Institute of Computer Science, University of Bremen, Bremen 28359, Germany
jpeters@informatik.uni-bremen.de

Abstract. The complexity of modern embedded systems makes it inevitable to consider higher abstraction levels in the design process to overcome problems in acceptable time and effort. In higher abstraction levels, the utilization of functional requirements is quite advanced, while the utilization of non-functional requirements like timing still is an open problem. We aim to address this problem utilizing the timing definitions from UML/MARTE CCSL.

Keywords: CCSL, UML, MARTE, SystemC, Formal Methods

1 Introduction

Modern embedded systems are growing to a huge complexity, making classical design tasks error-prone and time-consuming. As a consequence, novel design flows introduced several abstraction levels to overcome this problem. At this stage, the classical level of highest abstraction is the *Electronic Systems Level* (ESL), where SystemC and other high-level programming languages are used to describe the system. But still, a big gap remains between textual specification and ESL. In the last decade, modeling languages like the *Unified Modeling Language* (UML, [2]) provided a "bridge" between the given specification and its initial implementation [4]. In the design of embedded and cyber-physical systems, particularly the *Modeling and Analysis of Real-time and Embedded systems* profile (MARTE, [1]) finds considerable attention.

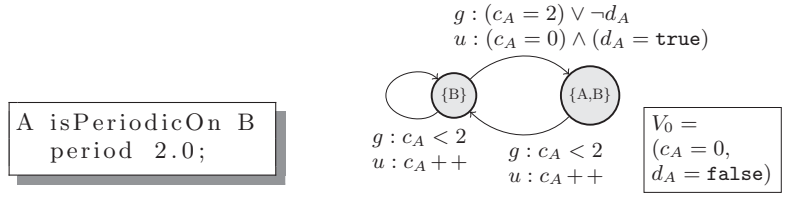
In this field, the utilization of non-functional requirements like timing is still an open problem. MARTE provides a special language for timing specification: the *Clock Constraint Specification Language* (CCSL), which relies on describing clocks and instants. In our project we are going to utilize this language for classical design tasks such as verification or code-generation.

2 Design and More – A Generic Representation of CCSL

To utilize the textual CCSL specification, we introduce a generic automaton representation of it. A lot of approaches have already been proposed to directly

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_17, © Springer Fachmedien Wiesbaden 2015



(a) CCSL constraints (b) Corresponding automaton

Fig. 1. Generic representation of CCSL constraints

transform the CCSL constraints to input for certain model checkers [5] or to ESL descriptions [3]. We transform it to a more generic representation which can be used for several tasks at the same time.

The main concept of our approach is the *ticking set*, which is the set of clocks $c \in \mathcal{C}$ ticking in one simulation step. All clock behavior can be modeled as a movement between these ticking sets, making the ticking sets *states* and the movement between them the *transitions*. These transitions can be restricted according to the CCSL constraints using guard functions over global variables. These variables are manipulated using update functions.

For a more detailed explanation, consider the CCSL specification from Fig. 1(a). It means that `B` is not restricted and can tick whenever it wants, while `A` is a subclock of `B`. Thus, `A` cannot tick alone but has to tick coincidently with `B`. For the resulting automaton, this gives the ticking sets {`B`} and {`A`, `B`} (see Fig. 1(b)). Furthermore, as the period of `A` is 2.0, we can conclude that the ticking set {`A`, `B`} can never be followed by itself, i. e. this transition is not needed. Finally this leads to two states and three transitions (see Fig. 1(b)).

To enforce the periodicity, now a counter c_A for the period is added and additionally a Boolean variable d_A to represent, if the subclock `A` has already ticked. The initial values V_0 are $c_A = 0$ and $d_A = \texttt{false}$. For every transition leading to a state not containing `A`, c_A is increased, while the other transition resets the counter to zero and sets d_A, representing, that `A` has ticked now and the period starts again (see the conditions u in Fig. 1(b)). The transition to the ticking set including `A` can only be taken if the counter is high enough, while the other transitions can only be taken if it is low enough (see g in Fig. 1(b)). Now, the resulting automaton can be used for various design tasks such as e. g. verification or code-generation.

3 Generating SystemC

As the behavior and thereby the automaton and its analysis time is growing exponentially in the clock number, we developed a faster way to simulate and test the constraints together with SystemC applications [3]. This is an alternative to the automaton approach especially for systems with high numbers of clocks and, thus, long verification times. Our code generation scheme extends a given functional implementation in SystemC with timing. To generate the behavior, we extend the existing implementation by a *TimeController* as shown in Fig. 2.

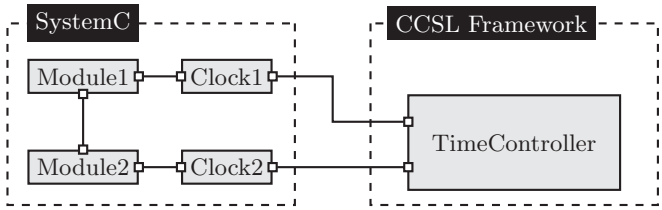


Fig. 2. Example SystemC implementation extended by CCSL Framework

CCSL constraints can be divided in two kinds: combination and future constraints. Combination constraints state, in what combinations clocks may or may not tick, while future constraints define times in the future from a given moment on, where other clocks have to tick or not to tick. To store the future constraint information, we use *ClockMonitor* objects. One object for every clock stores, which constraints it applies to other clocks and what restrictions are applied to itself. The combination constraints are represented by *Bind* objects. Finally, the TimeController uses lists of clocks that *can, cannot* or *must* tick in every step to represent the behavior. These lists are updated according to the constraints, until all clocks are distributed between *must* and *cannot*. The clocks in *must* form finally the ticking set of that simulation step.

4 Conclusion and Future Work

In the recent past, we developed a generic representation of CCSL constraints in terms of automata, which can represent the whole breadth of CCSL constraints. As the analysis of these automata is time-consuming, we developed a simple and fast code-generation scheme, which can be used for fast tests and simulations. Now, we want to combine the two approaches to get better simulations (regarding non-deterministic choices and clock-dependencies) and to improve the automaton evaluation towards bounded model checking and symbolic representation to make its use more feasible.

References

1. Object Management Group: UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Object Management Group (2011)
2. Object Management Group: OMG Unified Modeling Language TM (OMG UML) Superstructure. Object Management Group (2011)
3. Peters, J., Wille, R., Drechsler, R.: Generating SystemC Implementations for Clock Constraints Specified in UML/MARTE CCSL. International Conference on Engineering of Complex Computer Systems (ICECCS), 116–125 (2014)
4. Drechsler, R., Soeken, M., Wille, R.: Formal Specification Level: Towards Verification-driven Design Based on Natural Language Processing. Forum on Specification and Design Languages (FDL), 53–58 (2012)
5. Mallet, F., Yin, L.: Correct Transformation from CCSL to Promela for verification. Institut National de Recherche en Informatique et en Automatique, (2012)

Design and Synthesis of Reversible Circuits using Hardware Description Languages*

Eleonora Schönborn
PhD advisor: Rolf Drechsler

Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
eleonora@informatik.uni-bremen.de

Abstract. Reversible computation recently gained a lot of interest due to applications in areas like quantum computation and low power design. Due to its special properties, the established circuit design flow cannot simply be applied to reversible logic. In this work, we consider hardware description languages for the scalable design and synthesis of reversible circuits. Two complementary directions are discussed, namely (1) exploiting the conventional design flow and mapping the result to a reversible circuit, and (2) developing an entirely new design flow, which considers reversibility right from the beginning.

1 Reversible Computation

Nowadays, computational components are being embedded in more and more objects of our everyday lives. In smartphones, cars, medical equipment, etc. these components are linked closely to their physical environment using sensors and actors. Connected via networks they form cyber-physical systems. The expectations on these integrated circuits are rising with their number of applications. Especially low energy consumption has become a crucial design goal. While established power management techniques are reaching their limits, technologies alternative to CMOS are becoming more important day by day.

Many alternative technologies and applications currently investigated are based on reversible computation. Examples include quantum computation [1], low power design [2], and adiabatic circuits [3].

Reversible computation is a computing paradigm which only allows reversible (i.e. bijective) operations. Thus, each gate in a reversible circuit represents a bijection. Conventional gate libraries cannot be applied here, and new libraries of reversible gates have been introduced. Furthermore, fanout and feedback are generally not allowed in reversible circuits. As a consequence, design methods cannot simply be transferred from conventional circuit design, but have to be adapted or developed from scratch.

Most existing methods for reversible circuits work on the gate level, so almost no support for reversible systems on the specification level, the electronic system level, or the register transfer level exists yet. Moreover, most of the existing approaches for synthesis only accept specifications provided in terms of Boolean

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_18,

function descriptions like truth tables or Boolean decision diagrams, making them hardly scalable.

In this work, we investigate the use of hardware description languages (HDLs) for the design and synthesis of reversible circuits. Two complementary directions are discussed, namely (1) designing reversible circuits by exploiting the conventional design flow first, and afterwards mapping the result to a reversible circuit, and (2) applying an entirely new design flow to be developed, which considers reversibility right from the beginning through all abstraction levels.

2 Exploiting the Conventional Design Flow

The design flow for conventional circuits has been continually improved over decades and offers many powerful design tools and algorithms. Here, we consider using these methods for the design of reversible circuits. To be precise, the first steps of the design process follow the conventional design flow. The resulting conventional design will then automatically be mapped to a reversible circuit description by methods yet to be developed.

When following this direction, the most important questions are:

- At which abstraction level should the conventional design be mapped to a reversible circuit description?
- What mapping method will create the best reversible circuits regarding criteria like gate cost, delay, and number of signals?
- How can the mapping be done efficiently w.r.t. runtime and memory usage?

Mapping at a low abstraction level like the gate level can be realized straightforwardly. Each conventional gate is substituted by a template of reversible gates realizing the same function or, in the case of irreversible functions, embedding the function in a bijection using additional circuit lines. However, since each gate is mapped individually without regarding global information, the resulting circuits are usually far from optimal.

We consider an approach mapping from the register transfer level instead. The mapping scheme is similar to the one described for the gate level, but instead of single gates, complete modules have to be substituted. For this purpose, past accomplishments in the design of reversible building blocks for various data flow operations like adders, multipliers, etc. can be exploited. This way, circuit lines and/or gate cost can be saved compared to the gate level mapping.

Mapping from an higher level of abstraction, like the HDL description, would enable the use of even more global information and thus further reductions in the resulting circuits. However, this would require a complex mapping scheme.

3 Developing a Specific Design Flow

The second direction aims for the development of an entirely new design flow which considers reversibility from the specification and through all following abstraction levels. Special characteristics of reversible functions could be exploited

this way. Theoretically, there would be no need for embedding. On the downside, the whole design flow has to be redeveloped.

For the specification of large and/or complex reversible systems, HDLs supporting the characteristics of reversible logic have to be developed. Thus far, only preliminary versions of such HDLs are available (e.g. [4, 5]).

These languages do not allow for direct assignments such as `a=b` as this would be irreversible. Instead, reversible assignment operations are used, and can be combined with not necessarily reversible expressions. Despite significant differences like this between reversible and conventional HDLs, these languages enable the design of complex systems in reversible logic as we showed in [6].

In [4] an algorithm was introduced which allows for the synthesis of a reversible circuit directly from the HDL description. In this algorithm, a statement like `c^=a*b` is realized by cascading building blocks for the operations (multiplication and XOR-assignment). Since non-reversible parts of the overall reversible statement are synthesized separately, additional circuit lines are required for embedding. Hence, this synthesis scheme suffers from similar problems as the mapping methods discussed in Section 2. But in contrast, the initial reversible description allows for un-computing temporary results and thus for saving some of the additional lines, as we discussed in [7].

4 Conclusion

To efficiently design reversible logic, we need to investigate high abstraction levels like HDL. In this work, two directions are considered: Exploiting the conventional design flow and developing a new flow according to the properties of reversible circuits. Which direction should be taken is not obvious and may depend on the application. The goal of this work is to investigate both directions, develop new methods as well as compare and improve existing ones.

References

1. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge Univ. Press (2000)
2. Berut, A., Arakelyan, A., Petrosyan, A., Ciliberto, S., Dillenschneider, R., Lutz, E.: Experimental verification of Landauer's principle linking information and thermodynamics. Nature **483** (2012) 187–189
3. Patra, P., Fussell, D.: On efficient adiabatic design of MOS circuits. In: Workshop on Physics and Computation, Boston (1996) 260–269
4. Wille, R., Offermann, S., Drechsler, R.: SyReC: a programming language for synthesis of reversible circuits. In: Forum on Specification & Design Languages. (2010) 184–189
5. Thomsen, M.K.: A functional language for describing reversible logic. In: Forum on Specification & Design Languages. (2012) 135–142
6. Wille, R., Soeken, M., Große, D., Schönborn, E., Drechsler, R.: Designing a RISC CPU in Reversible Logic. In: Int'l Symp. on Multiple-Valued Logic. (May 2011) 170–175
7. Wille, R., Soeken, M., Schönborn, E., Drechsler, R.: Circuit line minimization in the HDL-based synthesis of reversible logic. In: IEEE Annual Symposium on VLSI. (2012) 213–218

Dynamic Rebound Control and Human Robot Interaction of a Ball Playing Robot*

Dennis Schüthe
PhD advisor: Udo Frese

University of Bremen, schuethe@informatik.uni-bremen.de

Abstract. Building a ball playing entertainment robot with the main contribution of a task level optimal control which handles trajectory planning and optimal control in a single controller and takes redundancy and elasticity into account. For faster actuation the feedback gain is used and the controller distributes into two parts. Human-Robot-Interaction(HRI) completes the entertainment aspect.

1 Introduction

Playing ball games is challenging for humans of all ages, especially if the ball should be played back precisely to an opponent. When children start learning it, you see a lot of divergence at the target position. For a robot such a game is a demanding task due its physical limitations and timing aspects. The controller for this task is presented herein and how to interact with the audience for a more lively scenery and more attractive robot for people.

2 The Physical System

The entertainment robot is called Doggy(c. f. Figure 1). It is an enhanced development of the first version Piggy[4]. The head of Doggy is a 40 cm styrofoam sphere added at the end of a carbon rod. The head is used as racket, this makes the orientation of the head unimportant. The rod itself is attached to the third Axis of the robot, which moves the head sidewards — for Doggy we use only revolute joints. The second Axis holds Axis 3 to turn the head forward and backward. Together with the head, Axes 2 and 3 describe a partial hollow sphere as workspace, which is due to joint limitations. Both Axes are mounted on the first Axis, which acts like a hip and can turn the workspace around. This results in a redundant Degree of Freedom, see Figure 1. In upright position the robot is 2.1 m tall. The axes of the robot are driven by DC-motors through tooth belts. This leads to elasticity between motor and joint and has to be taken into account by our controller.

Moreover, a stereo camera system is added to the robot at its hip. The system is used to track balls[2] and calculating a position, where the balls intersact with the robot's workspace. For the HRI the cameras can also be used to track people. Additionally, a stereo microphone is attached to the system to make the dog hear and respond on noise.

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_19, © Springer Fachmedien Wiesbaden 2015

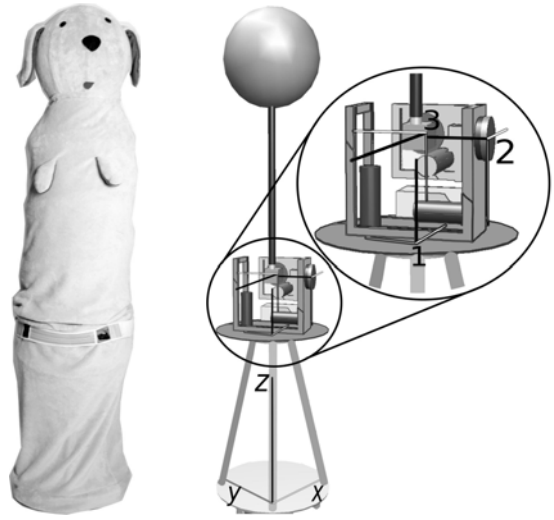


Fig. 1: (Left) The robot Doggy. (Right) Principle drawing of axes and interior.

3 Task Level Optimal Control

The most challenging part is the design of the controller. As we are talking about ball games with accurate rebound we have to fulfill following conditions: Being with the head at time T at position p with the velocity v. Moreover, we have constraints given by the joint limitations and the physical motor limitations.

For that we use Task Level Optimal Control[6]. Compared to most implementations, we use the finite horizon linear quadratic regulator in an adapted way, where the horizon is an additional input parameter, similar to [3]. In contrast to this work, we propagate the feedback gain for a number of steps instead of an online computed control value for every step [5]. This leads to a faster computation as we do not need to recalculate the feedback gain for every step.

We distribute the controller into two parts, a fast acting linear optimal and a nonlinear optimal controller. The latter runs on a computer to calculate the feedback gain for the linearized nonlinear model of the robot at an update rate of 50 Hz given by the stereo camera system. For the linearization points we predict our system behavior, where the starting point is the actual system state. The state consists of motor and joint velocities and positions. The motor positions and velocities can be measured directly, where the joint values have to be estimated using inertia measurement units data. On a microcontroller we compute input commands by the feedback gain, at rate of 1 kHz. A disturbance would then be recognized and countered first on the microcontroller in a linear and very fast behavior. Deviations from the calculated path would be recognized on the computer at the next update step. This will recompute the linearized dynamics with the actual state as starting point. The controller adapts to the new situation and the updated feedback gain is propagated to the Microcontroller.

A second advantage is to define a whole task instead of calculating a trajectory and follow it by a controller. Therefore, we propagate T, p, and v to the controller, which automatically decides the optimal way. Moreover, our definition of optimality is to reach the desired p and v with lowest energy input to the system, i. e. minimization of the input torque. The achievement of our work is a controller that combines the regulation of the plant and the trajectory planning in one optimal controller, which is more efficient than a seperated implementation.

Furthermore, we want to include the physics of playing back the ball into the optimization process, such that the target position is passed to the controller and not the intersection position and velocity.

4 Human-Robot-Interaction

To complete the entertainment aspect, the robot interacts with the audience to get their acceptance as an opponent player. First, there is the gaming scenario, which is mainly to hit the ball back to the player who has thrown it. But there could also be variations of this game. The robot could hit the ball to another person which has been recognized by cameras. This would take more attention of the audience, as the ball might be thrown to them. Furthermore, the detection algorithm should recognize if a person is watching the game and can be included.

Another part is the acoustic. The robot should act on noise as a human would do, i. e. react to noise which has not been there before by turning the head to the source and adapt to the noise level in environments where a lot of people talk or music is playing. A first version is able to handle these things and turns the robots head towards the sound of interest[1].

References

1. Bartsch, M.: Sound of Interest. Ein Ballspielroboter hört stereo. Master's Thesis (2014)
2. Birbach, O., Frese, U.: A Precise Tracking Algorithm Based on Raw Detector Responses and a Physical Motion Model. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 4746-4751. (2013)
3. Goretkin, G., Perez, A., Platt, R., Konidaris, G.: Optimal Sampling-Based Planning for Linear-Quadratic Kinodynamic Systems. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 2429–2436. (2013)
4. Laue, T., Birbach, O., Hammer, T., Frese, U.: An Entertainment Robot for Playing Interactive Ball Games. In: RoboCup 2013: Robot Soccer World Cup XVII, pp.171–182. Springer (2013)
5. Mare J. B., De Don J. A.: Solution of the input-constrained {LQR} problem using dynamic programming. In: Systems & Control Letters, pp. 342–348. (2007)
6. Schüthe, D., Frese,U.: Task Level Optimal Control of a Simulated Ball Batting Robot. In: 11th International Conference on Informatics in Control, pp. 45–56. SCITEPRESS (2014)

Development of Consistent Formal Models*

Julia Seiter
PhD advisor: Rolf Drechsler

Institute of Computer Science, University of Bremen, Germany

Abstract. Formal models can be used in the system design process to find design errors as soon as possible and to reduce the time-to-market and the development costs. Several methods for the verification of such models have been proposed in the past. However, developing such a formal model usually requires several iterations in a so-called *refinement* process. Between each of these steps, the consistency of the models' behavior has to be ensured as new errors can be introduced. Additionally, coverage metrics are necessary to determine if the model can be implemented yet or requires further consideration. The major contributions of this thesis are (1) a formally sound approach for the verification of model refinements, (2) a technique to retrieve correct and formal relations between the iterations of the refinement, and (3) a coverage metric for formal models.

1 Formal Modeling for Complex Systems

Modeling languages are well-established concepts in the field of software design. Methodologies such as model-driven architecture (MDA, [7]) put their focus on the development of abstract models which can be employed for various purposes. The most prominent example for such a language is the Unified Modeling Language (UML, [8]) which offers various diagrams for the specification of all kinds of different aspects of a software system. However, not only the domain of software design makes use of modeling languages. With the increasing complexity of embedded systems, high-level descriptions become more and more important for hardware systems or mixed hardware/software systems. Figure 1(a) shows an example of a formal model in UML.

A formal model of a combined hardware/software system offers various advantages. It facilitates graphic documentation in a way that is understandable for both, designers and stakeholders. Additionally, code generation techniques can be applied to create a description in, for example, a hardware description language. Finally, early validation and verification are rendered possible if the model is constructed accordingly.

2 Verification of Formal System Models

The verification of formal models differs in certain aspects from the verification of hardware designs. Usually, such models are not checked for particular functional properties but for consistency, either of a single model instance or of

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_20,

(a) Abstract model (b) Refined model

Fig. 1. Example of a model and its refinement

the model's dynamic behavior. Verification tasks for formal models may include static consistency (a single state in which all global constraints are satisfied), the reachability of a good/bad system state, or the executability of a single operation.

3 Consistent Model Development

So far, we have considered single instances of formal models and their correctness [11]. The development of a model which can be used for implementation, however, is an iterative process in which detail is added to an abstract model to make it more specific. This process is called *refinement*.

Refinement verification is very crucial as a correct refinement may save the developer time and effort. If it can be proven that a refinement step preserves the abstract model's behavior in the refined model, then certain verification results from the abstract model are preserved as well. However, with each refinement step, new errors can be introduced and already proven properties may be rendered invalid.

Until now, most approaches to refinement verification require manual interaction by employing for example theorem proving [1]. This demands a highly specialized and experienced verification engineer. Additionally, it can be a very time-consuming process. However, automatic techniques for the verification of formal models do exist. By extending those approaches such that they become applicable to more than one model, refinement verification is rendered possible.

As a basis, the relation between the abstract and the refined model has to be formalized. Then, several correctness criteria can be defined on this relation to ensure that the abstract behavior is preserved during the refinement. Figure 3 shows an example for a model refinement. The abstract model in Figure 1(a) has been refined to the model in Figure 1(b) by the strengthening of an operation constraint. The consistency of such a refinement can be proven by showing

1. that the initial states of the models match/are consistent and
2. that after the execution of a refined operation, the constraints of the abstract operation are still valid.

We provide a more detailed description of the approach in [9]. Having proven a refinement consistent, the remaining verification effort might be even further reduced by the application of optimizations such as slicing where the model is reduced based on the verification task [10].

Refinement, however, can also occur in another form in the design process. Since the UML provides so many different diagram types for the specification

from different viewpoints, it is also necessary to ensure consistency between these different diagrams. In [6], we consider different diagram types to describe structure on the one hand and behavior on the other, but both on the same level of abstraction. By transforming the behavioral description into constraints and the structural model, their consistency can be proven.

Unfortunately, all of these approaches are only applicable when the relation between the models is known. Without a formal refinement relation, the verification approach mentioned above cannot be used. Accordingly, methods to derive such a relation from two models are required. In the past, several works have considered how to retrieve traceability information, but most of them focus on the structure and ignore the models' behavior. Thus, we propose a fully automated retrieval methods for correct refinement relations, based on the criteria presented in [9].

Last but not least, the question remains when the resulting model is ready for implementation. In hardware verification, coverage metrics have been proposed to measure the completeness of the verification or test process [2, 5, 3, 12]. However, since verification on formal models is very different from verification on lower levels of abstraction, new metrics are required. In [4], we propose a coverage metric for formal specifications which ensures the executability of all operations. A second coverage metric in this work considers the relevance of the constraints on the operations.

References

1. Abrial, J.R.: The B-book: assigning programs to meanings. Cambridge University Press, New York, NY, USA (1996)
2. Bormann, J.: Complete functional verification. In: Formal Methods in Computer Aided Design (FMCAD), Industrial Experience Report (2009)
3. Claessen, K.: A coverage analysis for safety property lists. In: Formal Methods in Computer Aided Design (FMCAD). pp. 139–145. IEEE (Nov 2007)
4. Drechsler, R., Seiter, J., Soeken, M.: Coverage on the formal specification level. In: Int'l Workshop on Design and Implementation of Formal Tools and Systems (2014)
5. Große, D., Kühne, U., Drechsler, R.: Analyzing functional coverage in bounded model checking. Computer-Aided Design of Circuits and Systems (TCAD) pp. 1–11 (2008), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4544863
6. Hilken, C., Seiter, J., Wille, R., Kühne, U., Drechsler, R.: Verifying consistency between activity diagrams and their corresponding ocl contracts. In: Forum on specification and Design Languages (2014)
7. Object Management Group: Model driven architecture - mda guide rev. 2.0. Tech. rep. (2014)
8. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language reference manual. Addison-Wesley Longman, Essex, UK (Jan 1999)
9. Seiter, J., Wille, R., Kühne, U., Drechsler, R.: Automatic refinement checking for formal system models. In: Forum on specification and Design Languages (2014)
10. Seiter, J., Wille, R., Soeken, M., Drechsler, R.: Determining relevant model elements for the verification of uml/ocl specifications. In: Design, Automation and Test in Europe (2013)
11. Soeken, M., Wille, R., Drechsler, R.: Verifying dynamic aspects of UML models. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011. p. 16 (2011)
12. Tasiran, S., Keutzer, K.: Coverage metrics for functional validation of hardware designs. Design and Test of Computers 18, 36–45 (2001)

Formal Verification of Robustness*

Niels Thole[1,2]
PhD advisor: Görschwin Fey

[1] Institute of Computer Science, University Bremen, Germany
nthole@informatik.uni-bremen.de
[2] Institute of Space Systems, German Aerospace Center, Germany

Abstract. Due to the decreasing size of transistors, the probability of transient errors and the variability of the transistor's characteristics in electrical circuits are continuously increasing. These issues demand for techniques to check the robustness of circuits and their behavior under transient faults and variability. Furthermore, the implementation of methods that provide robustness are prone to implementation errors. Checks are needed to verify that the nominal behavior of the system did not change due to modifications that are meant to provide robustness. Solutions for both problems are presented in this work.

1 Introduction

New technology decreases the size and the energy consumption of transistors. While this development enables to create more advanced systems, they become more susceptible to *transient faults*. External noise like cosmic radiation can produce glitches in the system, which can lead to erroneous behavior. If a fault leads to observable erroneous behavior, an *error* occurs.

To prevent errors, several hardening techniques have been developed [1, 4]. A system that can prevent an error under a fault is called *robust*. All hardening techniques cause additional costs. The used techniques are often complex and implementation errors can arise. For this reason, the decision if, where, and how hardening techniques are implemented, should be taken carefully. To support the decision, an analysis of the robustness of the system is required. Based on this analysis, qualified decisions about robustness can be made. After a system has been modified to include hardening techniques, another analysis can be used to verify the effects of the new implementation.

This procedure enables a systematic approach to locate critical locations within the system and add hardening techniques without wasting resources by implementing unneeded hardening techniques.

The presented PhD topic is the formal verification of robustness. Different levels of abstractions are handled.

A part of the work describes the conservative analysis of a transient fault in a circuit on gate level. This work considers logical, timing, and electrical masking to provide a statement about the robustness of the circuit. If our algorithm decides that a circuit is robust, the conservative examination ensures that this decision is correct.

* This work was supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_21, © Springer Fachmedien Wiesbaden 2015

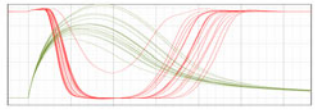


Fig. 1: Original Error Signals

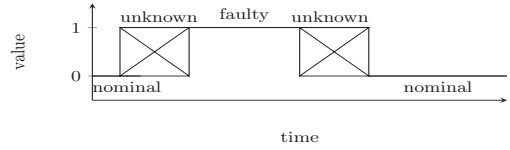


Fig. 2: SET-Model using a three-valued fault signal

Another part provides a method for *Equivalence Checking* between two systems that are modeled as *Finite State Machines* (FSM). The work can be used to verify that the added hardening techniques do not change the nominal behavior of the system. The current implementation can verify the equivalence between two C++ classes or provide a counterexample that shows that the models are not equivalent.

Using both methods, we can verify that added hardening techniques lead to a robust system and we can prove that the behavior remains the same.

2 Conservative Analysis of SETs

Some approaches for robustness checking use a coarse abstraction to enable the use on larger circuits while others are very precise but are limited to smaller circuits. Our work with Garcia-Ortiz [6] proposes a model that considers logical, timing, and electrical masking as well as gate variability. We provide an approach to determine if a *Single Event Transient* (SET) can lead to erroneous behavior by modeling the circuit including the SET as SAT-formula. Our conservative approach aims to ensure that a circuit is robust even under variability of parameters if the approach states that the circuit is robust.

In our work, we consider the circuit at gate level. This level cannot exactly represent the behavior of a circuit like the transistor level. To ensure the conservative analysis, we use three-valued logic to approximately describe the circuit's behavior.

At the physical level, signals have a voltage level in a continuous range which is interpreted as 1 or 0 if the voltage level is above or below certain thresholds. When the charge of a signal changes due to an SET, the voltage level of the signal changes gradually. Due to random variations in the gates, the effect of a soft error has a large variability as seen in Figure 1. To consider this uncertainty without modeling the exact voltage level, we consider the value to be unknown during that time and to be 0 or 1 otherwise like shown in Figure 2.

In our model, a signal is interpreted as a waveform, i.e., a vector of variables and an offset value, similar to WAVEsat [3]. The offset value describes the offset of the variables from timestep 0 while the different variables describe the change of the signal over time. We iteratively compute the waveform for each gate. Each waveform is further modified to include variable delays of the gates as well as electrical masking. If the changed signal transmits into the sampling window of an output, an error may occur and our approach returns this information.

Experiments were done to compare our algorithm with spice simulations and to evaluate the performance. Whenever the spice simulation detected an error, our algorithm found this error as well. Deciding the robustness within a time limit of 120 minutes was possible for all circuits of ISCAS-85 [2] except for the multiplier c6288 and few SETs in c7552 and c3540.

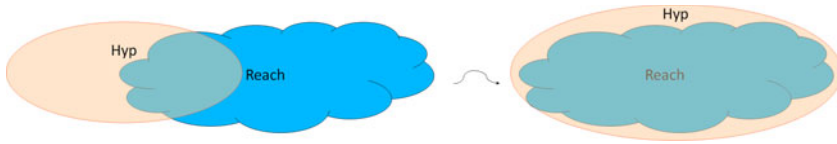


Fig. 3: Modifying the hypothesis

3 Equivalence Checking

Using the method for robustness checking, we can check a system for problematic gates that require hardening and can verify the robustness afterwards. But we cannot guarantee that the modified system behaves equivalently to the original system.

In our work [5], two abstract FSMs that describe hardware modules are checked for equivalence. The two models are equivalent if and only if the same sequence of inputs on both models always leads to equivalent outputs. A developer usually has additional knowledge about the hardware. We exploit that knowledge by formulating it as a hypothesis that approximates all equivalent states of the two models. A good hypothesis increases the performance of the equivalence check significantly.

Stepwise induction is used to prove the equivalence between the two models. Thus, we try to prove that each pair of states that fulfills the hypothesis produces equivalent output under equivalent input and the succeeding states also fulfill the hypothesis. If the proof is successful, the equivalence of the models is proved. Otherwise, the two models are not equivalent or the hypothesis is too weak. The hypothesis is then modified until a decision is possible like shown in Figure 3.

The approach is implemented and tested for hardware described at the system level by modeling a hardware module as a C++ class.

References

1. Black, J.D., Cressler, J.D., Mantooth, H.A.: Best practices in radiation hardening by design: CMOS. In: Extreme Environment Electronics, pp. 475–483. CRC Press (2013)
2. Bryan, D.: The ISCAS'85 benchmark circuits and netlist format. North Carolina State University (1985)
3. Sauer, M., Czutro, A., Polian, I., Becker, B.: Small-delay-fault ATPG with waveform accuracy. In: Proceedings of the International Conference on Computer-Aided Design. pp. 30–36 (2012)
4. Sterpone, L., Sonza Reorda, M., Violante, M., Kastensmidt, F., Carro, L.: Evaluating different solutions to design fault tolerant systems with SRAM-based FPGAs. Journal of Electronic Testing pp. 47–54 (2007)
5. Thole, N., Fey, G.: Equivalence checking on system level using stepwise induction. In: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen. pp. 197–200 (2014)
6. Thole, N., Fey, G., Garcia-Ortiz, A.: Analyzing an set at gate level using a conservative approach. Accepted at *Testmethoden und Zuverlässigkeit von Schaltungen und Systemen* (2015)

Pose and Posture Estimation using Inertial Sensor Data*

Felix Wenk
PhD advisor: Udo Frese

German Research Center for Artificial Intelligence, Felix.Wenk@dfki.de

Abstract. This paper discusses the estimation of the position and orientation, i.e. the pose, of a rigid body and the posture of a human using inertial sensor data only, i.e. without absolute pose information. Since this is impossible in general, specific assumptions as to the rigid body motion and the skeleton's structure are introduced. What to expect of estimates obtained using such assumptions is also briefly covered.

1 Introduction

Pose and Posture estimation are at the heart of various cyber-physical systems. A pose is the position and orientation of a rigid body. The relative poses of the bodies of a skeleton, a system of rigid bodies, are the skeleton's posture. Determining the posture of a human is the technical foundation to, for instance, let characters in animation films move naturally, do motion analysis of athletes[3], or measure stress on the skeleton of manual workers.

Poses are estimated whenever the orientation and position of a rigid body are tracked over time. This could be a flying ball, the robot hand to catch it[1], or the head of a person wearing virtual reality goggles[5].

My work's contributions, which this paper provides a brief overview of, are procedures to obtain an estimate of a rigid-body pose, which does not suffer from unlimited drift, from inertial sensor and magnetometer data only and to estimate postures of humans from inertial sensor data only, i.e. without magnetometers.

To estimate a pose or a posture, the rigid body or the skeleton is equipped with different sensors, depending on the application. To determine the orientation, magnetometers and inertial sensors, i.e. combinations of gyrometer and accelerometer, are a common choice. Cameras or GPS are candidates to observe the position.

To measure a human posture, the human in question typically wears a suit equipped with a lot of markers filmed by cameras. Alternatively as in the project SIRKA, whose objective is to build such a suit and which my work is part of, inertial sensors and magnetometers may be used.

In SIRKA, external sensors are avoided, i.e. no GPS satellites, cameras or mechanical sensors are allowed. This, for example, will enable us to estimate the posture of workers of the Meyer Werft, a german shipyard, who will use the estimates to identify and detect postures that potentially lead to occupational

* This work was supported by the BMBF project SIRKA and by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy.

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_22,

diseases. Because magnetic fields in ship bodies are constantly changing, the SIRKA suit won't use magnetometers.

2 Pose Estimation

Formally, estimating a pose of an object in a frame of reference, "world coordinates", means determining the coordinate transform from a coordinate system attached to the object to world coordinates. This transform is made of an orientation and a translation.

All it takes to estimate the pose is to find the orientation and translation that are most probable given the inertial and magnetometer measurements. The main difficulty is to identify the relationship between the pose and the sensor measurements. This is reasonably easy for the gyrometer, which measures angular velocity and thus relates two successive orientations in time.

It is not as easy for the accelerometer and magnetometer, which measure acceleration and the magnetic field, respectively. The acceleration must be related to the translation of the pose, because it is its second time derivative. Let the object with the accelerometer attached lie on the floor. The translation clearly does not change and its second time derivative, the acceleration, is zero. Interestingly, the accelerometer does not measure zero. It measures the acceleration exerted by the floor onto the sensor to counter gravity, preventing the object with the sensor to fall through that floor. So if at rest, the accelerometer measurement provides the direction of gravity, so it determines how much the sensor and object are rolled and tilted.

The magnetometer measurements are analogous to the non-accelerating accelerometer, but instead of the gravitational field, the magnetic field is measured. Assuming that this is earth's magnetic field, one also measures how much the object is panned.

The accelerometer, magnetometer and gyrometer measurements can be probabilistically fused into orientation estimates, even though the sensor measurements are noisy and the object is moving, provided that on average the accelerometer measures the negative gravity[2].

With that orientation estimate, gravity can be approximately be removed from the accelerometer measurement to obtain the linear acceleration, which may be integrated to a translation.

In contrast to the orientation, the inertial sensors provide no absolute measurement of the object's position, so there is no obvious way to compensate for accumulating drift due to sensor noise, as there was for the orientation. The key observation is that the orientation estimator only works because we only consider cases in which the assumption holds, that the accelerometer on average measures negative gravity. If we were to estimate the orientation of an object relative to a space station orbiting the earth, the orientation estimation approach just explained would not work.

It turned out that there is a class of motions compatible with a different assumption, on which an estimator for the complete pose can be built: If it is known that the object stays at the same position on long-term average, one can estimate the entire pose including short-term translations relative to the long-

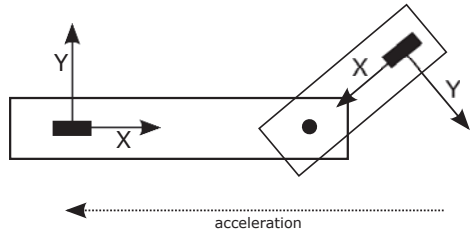


Fig. 1. Top-down view on two bodies (rectangles) connected over a joint (circle), each equipped with an accelerometer (filled rectangles). The acceleration is measured by both sensors on different axes, thus determining their relative orientation.

term average position. Readers interested in why that works and how this is probabilistically modelled should read [4].

3 Posture Estimation

What makes the collection of bodies a skeleton is that the bodies are connected to each other with joints. If displacements of the joints relative to fixed points on the rigid bodies the they are attached to are known, which they approximately are, then the poses of the bodies relative to each other can be determined by estimating their relative orientations. With the orientations known, adding the displacements vectorially yields the positions of (fixed points on) the bodies.

There is one catch. For the angle around the vertical, i.e. to tell where "forward" is, the orientation estimator relies on a magnetometer, which does not work in the environments the SIRKA posture estimator is to be used in.

For a skeleton's posture, we're interested only in the relative orientations of its bodies. The *relative* orientation is obtainable without a magnetometer if multiple bodies are considered at once. Let two bodies, as in figure 1, be subject to a linear acceleration. This determines their relative orientation except for the angle around the direction of the linear acceleration, which changes over time.

References

1. Birbach, O., Frese, U., Bauml, B.: Realtime perception for catching a flying ball with a mobile humanoid. In: Robotics and Automation (ICRA), 2011 IEEE International Conference on. pp. 5955–5962. IEEE (2011)
2. Kraft, E.: A quaternion-based unscented Kalman filter for orientation tracking. In: Proceedings of the Sixth International Conference of Information Fusion. vol. 1, pp. 47–54 (2003)
3. Roetenberg, D., Luinge, H., Slycke, P.: Xsens mvn: full 6dof human motion tracking using miniature inertial sensors. Tech. rep., Xsens Motion Technologies BV (2009)
4. Wenk, F., Frese, U.: Pose Estimation from Inertial Sensor Data and Prior Information. IEEE Transactions on Robotics (2014), under review
5. Yao, R., Heath, T., Davies, A., Forsyth, T., Mitchell, N., Hoberman, P.: Oculus VR Best Practices Guide. Oculus VR (2014), `http://static.oculus.com/sdk-downloads/documents/OculusBestPractices.pdf`

Reconfigurable Hardware-Based Acceleration for Machine Learning and Signal Processing ⋆

Hendrik Woehrle[1]
PhD Advisor: Frank Kirchner[1,2]

[1] German Research Center for Artificial Intelligence, DFKI Bremen, Robotics Innovation Center, Robert-Hooke-Str. 1, 28359 Bremen, Germany
hendrik.woehrle@dfki.de,
[2] University of Bremen, Faculty 3 – Mathematics and Computer Science, Robotics Lab, Robert-Hooke-Str.1, 28359 Bremen, Germany,

Abstract. Certain application areas of signal processing and machine learning, such as robotics, impose technical limitations on the computing hardware, which make the use of generic processors unfeasible. In this paper we propose a framework for the development of dataflow accelerators as a possible solution. The approach is based on model based development and code generation to allow a rapid development of the accelerators and perform a functional verification of the overall system.

Keywords: Robotics, Embedded Systems, FPGA, Hardware Acceleration, Dataflow

1 Introduction

Machine learning and signal processing methods are used for a wide range of applications nowadays. Furthermore, we can currently observe two important trends: the amount of data that has to be processed is constantly growing, while mobile and robotic systems become increasingly important. They impose additional requirements, such as limited physical space, power consumption and real time constraints, which make the usage of standard *generic* processors suboptimal. Here, combining a *generic* mobile CPU with *application specific hardware accelerators* for high performance computations are a reasonable approach. Field Programmable Gate Arrays (FPGAs) are a flexible possibility to implement such hardware accelerators. FPGAs consist of logic elements that can be configured to form specific circuits and implement an algorithm *in hardware* in order to achieve significant performance improvements. FPGAs become more and more important for applications such as digital signal processing, since vendors integrate components such as DSP slices. However, a major problem is the design complexity, which has prevented FPGAs from being widely used for machine learning and signal processing. Several *specific* FPGA implementations for machine learning algorithms exist. However, most of them are usually not generic

⋆ This work was supported by the *German Federal Ministry of Economics and Technology* (BMWi, grants FKZ 50 RA 1012 and FKZ 50 RA 1011).

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7_23,

or transferable. A generic framework for machine learning and robotics has been proposed in [1], but it is only suitable for stationary high performance, but not mobile, systems. In this paper, we propose the *re*configurable *S*ignal *P*rocessing *A*nd *C*lassification *E*nvironment (*reSPACE*) to overcome these problems. The first author is the main designer and developer of the presented framework.

2 Accelerator Hardware Architecture

The proposed framework is based on the *static heterogeneous synchronous dataflow* computing paradigm, which is popular for machine learning [2, 3]. In the dataflow computing paradigm, data is streamed through a sequence of algorithms, which are implemented as so-called *nodes*, and transformed on its way through them [4]. It is possible to combine the different nodes to a heterogeneous *dataflow accelerator* (DFA). Since the structure is pre-defined for a specific application, it is a *static* DFA. In contrast to software dataflow concepts, the data is shifted by one step through the flow on each clock tick of the system, resulting in a *synchronous* design. To implement a *complete system*, we provide two different, model-based alternatives. Either a *library of predefined, parametrizable nodes* can be used, that contains basic, widely used algorithms such as FIR and IIR filters, direct current offset removal, standardization, etc.. They are provided as directly usable nodes that can be combined to build up the DFA. Otherwise, a *customizable circuit generation for matrix-multiply based algorithms* can be used, which generates a node according to the specification given by a domain specific language. The operations are mapped directly to the DSP slices to perform resource and power efficient parallel matrix operations. There are at least two possibilities to use such DFAs: inside a *System on Chip (SoC)*, where the DFA is connected to a CPU via a bus system, or by *direct access* inside another hardware system. In the in SoC setup, the DFA can be used to accelerate a specific software task. Usually, an operating system like Linux is running on the host CPU. Hence, to access the DFA from a software application, device drivers are needed that interface with the DFAs and run as kernel modules. They can be automatically generated according to meta-informations that describe the interface of the DFA. It is also possible to generate interfaces to high-level languages, e.g., Python, to integrate the DFA into frameworks such as pySPACE [3]. Still, the design and implementation of DFAs can be a complex and error-prone task, due to the FPGA properties, like cycle-accurate timing and numerical issues due to fixed-point computations that need to be addressed. Therefore, reSPACE supports the automatic generation of testbenches, which allow the verification and evaluation of the DFA in simulation and directly on the target system.

3 Example Application: Biomedical Signal Processing

Exoskeletons can be used as an intuitive input device for the teleoperation of robotic systems [5] or as rehabilitation devices [6]. To enhance the smoothness of interaction with the exoskeleton, the joint control algorithms can be modulated

by integrating predictions of upcoming movements based on the online analysis of the human operator's EEG [7] using a movement prediction system, that can be *embedded* in the exoskeleton. EEG data is high-dimensional, and the online analysis requires a wide range of different computationally demanding signal processing and machine learning operations. In this case, reSPACE can be used to map the operations to FPGAs to provide the necessary computational power. **Further Applications:** There are various application areas in machine learning, robotics and autonomous systems that would gain a substantial benefit from FPGA-based accelerators. Examples are various image processing techniques or enhancing the control of robots [8].

4 Conclusion and Future Work

In this paper, we proposed an holistic approach for the rapid development of dataflow accelerators for machine learning and signal processing. The techniques, i.e., model-based hardware development, software generation and functional verification, are bundled in the framework reSPACE. In future, we will apply it in a wide range of different applications and provide the framework as open source.

References

1. Graf, H.P., Cadambi, S., Durdanovic, I., Jakkula, V., Sankaradass, M., Cosatto, E., Chakradhar, S.T.: A massively parallel digital learning processor. In: NIPS. (2008)
2. Zito, T., Wilbert, N., Wiskott, L., Berkes, P.: Modular toolkit for Data Processing (MDP): a Python data processing framework. Frontiers in Neuroinformatics **2**(8) (2008)
3. Krell, M.M., Straube, S., Seeland, A., Wöhrle, H., Teiwes, J., Metzen, J.H., Kirchner, E.A., Kirchner, F.: pySPACE - a signal processing and classification environment in Python. Frontiers in Neuroinformatics **7**(40) (2013)
4. Flagg, M.: Dataflow principles applied to real-time multiprocessing. In: COMPCON Spring'89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers., IEEE (1989)
5. Folgheraiter, M., Kirchner, E.A., Seeland, A., Kim, S.K., Jordan, M., Woehrle, H., Bongardt, B., Schmidt, S., Albiez, J., Kirchner, F.: A multimodal brain-arm interface for operation of complex robotic systems and upper limb motor recovery. In: Proc. of the 4th International Conference on Biomedical Electronics and Devices (BIODEVICES-11), Rome (2011)
6. Kirchner, E.A., Albiez, J., Seeland, A., Jordan, M., Kirchner, F.: Towards assistive robotics for home rehabilitation. In Chimeno, M.F., Solé-Casals, J., Fred, A., Gamboa, H., eds.: Proceedings of the 6th International Conference on Biomedical Electronics and Devices (BIODEVICES-13), Barcelona, ScitePress (2013)
7. Seeland, A., Woehrle, H., Straube, S., Kirchner, E.A.: Online movement prediction in a robotic application scenario. In: 6th International IEEE EMBS Conference on Neural Engineering (NER), San Diego, California (2013)
8. Langosz, M., von Szadkowski, K., Kirchner, F.: Introducing particle swarm optimization into a genetic algorithm to evolve robot controllers. In: Proceedings of the 16th Annual Conference on Genetic and Evolutionary Computation. GECCO '14, ACM (2014)

Author Index

Aydos, Gökçe, 272

Becker, Bernd, 122

Chakrabarty, Krishnendu, 190

de Gea Fernandez, Jose, 224
Diepenbeck, Melanie, 275

Frehse, Goran, 50

Glesner, Sabine, 1
Goldhoorn, Malgorzata, 278
Goldhoorn, Matthias, 281

Hübner, Felix, 287
Haddadin, Sami, 249
Havelund, Klaus, 151
Haxthausen, Anne Elisabeth, 82
Herber, Paula, 1
Hilken, Christoph, 284

Ibrahim, Mohamed, 190

Kampmann, Peter, 224
Kirchner, Elsa Andrea, 224
Kirchner, Frank, 224

Li, Xian, 290
Li, Zipeng, 190

Mallet, Frédéric, 26
Metzen, Jan Hendrik, 224

Peleska, Jan, 82
Peters, Judith, 293

Reger, Giles, 151

Sauer, Matthias, 122
Schönborn, Eleonora, 296
Schüthe, Dennis, 299
Scholl, Christoph, 122
Schröer, Martin, 224
Seiter, Julia, 302

Thole, Niels, 305

Wenk, Felix, 308
Wimmer, Ralf, 122
Woehrle, Hendrik, 311

R. Drechsler, U. Kühne (eds.), *Formal Modeling and Verification of Cyber-Physical Systems,*
DOI 10.1007/978-3-658-09994-7, © Springer Fachmedien Wiesbaden 2015