2nd International Workshop on Synthesis of Complex Parameters

SynCoP'15, April 11, 2015, London, UK

Edited by

Étienne André
Goran Frehse

Editors

Étienne André
Université Paris 13, Sorbonne Paris Cité, LIPN
CNRS, UMR 7030
F-93430, Villetaneuse, France
`Etienne.Andre@lipn.univ-paris13.fr`

Goran Frehse
Universite Joseph Fourier Grenoble 1
Verimag
Grenoble, France
`Goran.Frehse@imag.fr`

ACM Classification 1998
D.2.4 Software/Program Verification

ISBN 978-3-939897-82-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-82-8.

Publication date
November 2015

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/OASIcs.SynCoP.2015.i

ISBN 978-3-939897-82-8 **ISSN 2190-6807** **http://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

ISSN 2190-6807

www.dagstuhl.de/oasics

Contents

2nd International Workshop on Synthesis of Complex Parameters (SynCoP'15).
Editors: Étienne André and Goran Frehse
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the proceedings of the 2nd International Workshop on Synthesis of Complex Parameters (SynCoP'15). The workshop was held in London, UK on April 11th, 2015, as a satellite event of the 18th European Joint Conferences on Theory and Practice of Software (ETAPS'15).

SynCoP aims at bringing together researchers working on verification and parameter synthesis for systems with discrete or continuous parameters, in which the parameters influence the behavior of the system in ways that are complex and difficult to predict. Such problems may arise for real-time, hybrid or probabilistic systems in a large variety of application domains. The parameters can be continuous (e.g., timing, probabilities, costs) or discrete (e.g., number of processes). The goal can be to identify suitable parameters to achieve desired behavior, or to verify the behavior for a given range of parameter values.

The scientific subject of the workshop covers (but is not limited to) the following areas:

- parameter synthesis,
- parametric model checking,
- regular model checking,
- robustness analysis,
- parametric logics, decidability and complexity issues,
- formalisms such as parametric timed and hybrid automata, parametric time(d) Petri nets, parametric probabilistic (timed) automata, parametric Markov Decision Processes, networks of identical processes,
- interactions between discrete and continuous parameters,
- applications to major areas of computer science and control engineering.

Program

This volume contains nine contributions: two invited talks, six regular papers and three abstracts of informal presentations. The two invited talks are:

- Cut-offs in Parameterized Verification (Parosh Abdulla)
- Parameter synthesis for probabilistic real-time systems (Marta Kwiatkowska)

Each regular paper was reviewed by at least three different reviewers. The six accepted papers are:

- Consistency for Parametric Interval Markov Chains (Benoît Delahaye)
- Guaranteed control of switched control systems using model order reduction and state-space bisection (Adrien Le Coënt, Florian De Vuyst, Christian Rey, Ludovic Chamoin and Laurent Fribourg)
- Game-based Synthesis of Distributed Controllers for Sampled Switched Systems (Laurent Fribourg, Ulrich Kühne and Nicolas Markey)

- Parameter and Controller Synthesis for Markov Chains with Actions and State Labels (Bharath Siva Kumar Tati and Markus Siegle)
- Parametric Verification of Weighted Systems (Peter Christoffersen, Mikkel Hansen, Anders Mariegaard, Julian T. Ringsmose, Kim G. Larsen and Radu Mardare)
- Tuning PI controller in nonlinear uncertain closed-loop systems with interval analysis (Julien Alexandre Dit Sandretto, Alexandre Chapoutot and Olivier Mullier)

Furthermore, three informal presentations were made at the workshop:

- Discrete Parameters in Petri Nets (Nicolas David, Claude Jard, Didier Lime, and Olivier H. Roux)
- Enhanced Distributed Behavioral Cartography of Timed Automata (Étienne André, Camille Coti and Hoang Gia Nguyen)
- Parameter Synthesis with IC3 (Alessandro Cimatti, Alberto Griggio, Sergio Mover and Stefano Tonetta)

Support and Acknowledgement

SynCoP 2015 was partially supported by the ANR (Agence Nationale de la Recherche) research project ANR-14-CE28-0002 PACS (Parametric Analyses of Concurrent Systems), with the additional support of Université Paris 13 (Villetaneuse, France) and Verimag (Grenoble, France). We would like to thank these various entities for their generous financial and organizational support.

We thank the two invited speakers for their presence and their interesting presentations, the authors for their contributions, the program committee members for reviewing and selecting the papers, and Paulo Oliva from the ETAPS organizing committee for its support.

Finally, we would like to thank the editorial board of the OASIcs proceedings.

In Villetaneuse and Grenoble,

Étienne André and Goran Frehse

Program Committee Chairs

Étienne André	Villetaneuse, France
Goran Frehse	Grenoble, France

Publicity Chair

Benoît Delahaye	Nantes, France

Proceedings Chair

Didier Lime	Nantes, France

Program Committee

Étienne André	Villetaneuse, France (co-chair)
Alessandro Cimatti	Trento, Italy
Benoît Delahaye	Nantes, France
Giorgio Delzanno	Genova, Italy
Alexandre Donzé	Berkeley, USA
Goran Frehse	Grenoble, France (co-chair)
Laurent Fribourg	Cachan, France
Antoine Girard	Grenoble, France
Peter Habermehl	Paris, France
Claude Jard	Nantes, France
Sumit Kumar Jha	Orlando, USA
Vineet Kahlon	Austin, USA
Kim Larsen	Aalborg, Denmark
Didier Lime	Nantes, France
Wojciech Penczek	Warszawa, Poland
Laure Petrucci	Villetaneuse, France
Olivier H. Roux	Nantes, France
Jiří Srba	Aalborg, Denmark
Jun Sun	Singapore
Ashish Tiwari	USA
Tayssir Touili	Villetaneuse, France
Tomáš Vojnar	Brno, Czech Republic

Additional Reviewers

Giorgio Bacci	Aalborg, Denmark
Giovanni Bacci	Aalborg, Denmark
Lukáš Holík	Brno, Czech Republic
Radu Mardare	Aalborg, Denmark
Maciej Szreter	Warsaw, Poland

List of Authors

Parosh A. Abdulla
Uppsala University
Uppsala, Sweden

Julien Alexandre dit Sandretto
Unité d'Informatique et d'Ingénierie des Systèmes
ENSTA ParisTech, Université Paris-Saclay
Palaiseau, France

Étienne André
Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France
Villetaneuse, France

Ludovic Chamoin
LMT Cachan, ENS Cachan & CNRS
Cachan, France

Alexandre Chapoutot
Unité d'Informatique et d'Ingénierie des Systèmes
ENSTA ParisTech, Université Paris-Saclay
Palaiseau, France

Peter Christoffersen
Aalborg University
Aalborg, Denmark

Alessandro Cimatti
Fondazione Bruno Kessler
Trento, Italy

Camille Coti
Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France
Villetaneuse, France

Nicolas David
University of Nantes, LINA
Nantes, France

Florian de Vuyst
CMLA, ENS Cachan & CNRS
Cachan, France

Benoît Delahaye
Université de Nantes / LINA
Nantes, France

Laurent Fribourg
LSV, ENS Cachan & CNRS
Cachan, France

Alberto Griggio
Fondazione Bruno Kessler
Trento, Italy

Mikkel Hansen
Aalborg University
Aalborg, Denmark

Fréderic Haziza
Uppsala University
Uppsala, Sweden

Lukáš Holík
Brno University of Technology
Brno, Czech Republic

Claude Jard
University of Nantes, LINA
Nantes, France

Ulrich Kühne
Group of Computer Architecture
University of Bremen
Bremen, Germany

Marta Kwiatkowska
Department of Computer Science
University of Oxford
Oxford, UK

Kim Guldstrand Larsen
Aalborg University
Aalborg, Denmark

Adrien Le Coënt
CMLA, ENS Cachan & CNRS
Cachan, France

Didier Lime
École Centrale de Nantes, IRCCyN
Nantes, France

Radu Mardare
Aalborg University
Aalborg, Denmark

Anders Mariegaard
Aalborg University
Aalborg, Denmark

Nicolas Markey
LSV, ENS Cachan & CNRS
Cachan, France

Sergio Mover
Fondazione Bruno Kessler
Trento, Italy

Olivier Mullier
Unité d'Informatique et d'Ingénierie des Systèmes
ENSTA ParisTech, Université Paris-Saclay
Palaiseau, France

Hoang Gia Nguyen
Université Paris 13, Sorbonne Paris Cité,
LIPN, CNRS, UMR 7030, F-93430,
Villetaneuse, France
Villetaneuse, France

Christian Rey
LMT Cachan, ENS Cachan & CNRS
Cachan, France
Safran Tech
Magny les Hameaux, France

Julian Trier Ringsmose
Aalborg University
Aalborg, Denmark

Olivier H. Roux
École Centrale de Nantes, IRCCyN
Nantes, France

Markus Siegle
Universität der Bundeswehr München
Dept. of Computer Science
Munich, Germany

Bharath Siva Kumar Tati
Universität der Bundeswehr München
Dept. of Computer Science
Munich, Germany

Stefano Tonetta
Fondazione Bruno Kessler
Trento, Italy

View Abstraction – A Tutorial*

Parosh A. Abdulla[1], Fréderic Haziza[2], and Lukáš Holík[3]

1 **Uppsala University**
`parosh@it.uu.se`
2 **Uppsala University**
`frederic.haziza@it.uu.se`
3 **Brno University of Technology**
`holik@fit.vutbr.cz`

Abstract

We consider *parameterized verification*, i.e., proving correctness of a system with an unbounded number of processes. We describe the method of *view abstraction* whose aim is to provide a *small model property*, i.e., showing correctness by only inspecting instances of the system consisting of a small fixed number of processes. We illustrate the method through an application to the classical Burns' mutual exclusion protocol.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases program verification, model checking, parameterized systems

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.1

Category Invited Paper

1 Introduction

The behavior of many types of systems can be described using one or more *parameters* such as the number of processes, or the sizes of the data structures that the system uses. The goal of *parameterized verification* is to prove (or refute) the correctness of the system for all values of the parameters. There are numerous applications where parameterized systems arise naturally:

- *Number of processes.* In a mutual exclusion protocol, an arbitrary number of processes may participate in a given session of the protocol. In a cache coherence protocol, an arbitrary number of threads may share a cache line. In a Petri net, there is no bound on the number of tokens that are generated during a run of the net.
- *Sizes of data structures.* The behaviors of recursive programs can be captured using unbounded stacks [18]. Data link protocols can be modeled by processes communicating through unbounded FIFO queues [8]. The latter have also recently been used to encode the behavior of programs running on weak memory models such TSO, PSO, and POWER [12, 2, 28].
- *Multiple parameters.* Timed Petri Nets (TPN) [10] extend the model of Petri nets by equipping each token with a real-valued clock. A TPN induces a system that is infinite in two dimensions. More precisely, a run of a TPN may generate an unbounded number

* This work was partially supported by UPMARC, The Uppsala Programming for Multicore Architectures Research Center.

of tokens each of which is a real-valued clock. The implementations of concurrent stacks, queues, and sets are infinite in three dimensions [5]. First, an unbounded number of threads may operate on the data structure. Furthermore, there is no bound on the size of the data structure. Finally, the values stored inside the data structure are usually fetched from an infinite domain such as the set of integers.

In this tutorial, we concentrate on systems where the parameterization arises due to the number of processes. This class of systems can itself be divided into several subclasses depending on the following three parameters.

- *Processes.* The processes may be finite-state or infinite-state. Even in the case of finite-state processes, the state space of the system is unbounded. This is true since the state space contains all states we get as we vary the parameter (size of the system). The individual processes may be infinite-state since they may operate on variables ranging over infinite domains (e.g., the natural numbers). In such a case we get a state space that is infinite in two dimensions.
- *Topology.* On the one hand, the system may consist of a set of processes without any structure. On the other hand, the system topology may have a certain pattern. For instance, the processes may be organized as a linear array. Then, a process may refer to its left/right neighbors, or to all the processes to its left/right. The processes may also be organized in a ring, tree, or a general graph.
- *Communication Primitives.* A simple form of communication is when two processes perform a *rendezvous* which involves both processes changing state simultaneously. Another form of communication is through *shared variables* that can be read from and written to by all/some processes in the system. We may have *broadcast transitions* where an arbitrary number of processes change state simultaneously. Furthermore, the transitions of a process may be conditioned by *global conditions*. An example of a (universal) global condition, in a system with linear topology, is that "all processes to the left of a given process i should satisfy a property ϕ". In this case, process i is allowed to perform the transition only in the case where all processes with indices $j < i$ satisfy ϕ.

In this paper, we consider a class of systems where we have finite-state processes that are organized in a linear array. The processes communicate through global transitions. For such systems, we consider the verification of *safety properties*. Intuitively, a safety property states that nothing bad will happen during the execution of the system. For a a mutual exclusion protocol, a typical safety property is that no two processes should be in their critical sections at the same time. Checking a given safety property reduces to checking the reachability of a set of *bad* configurations, namely those that violate the property.

The work of this tutorial is based on the ideas presented in [4] that introduces *view abstraction*. View abstraction is inspired by strong empirical evidence that parameterized systems often enjoy a *small model property*. More precisely, it analyzes only a small number of processes (rather than the whole family) and shows that this is sufficient to capture the (un)reachability of bad configurations. On the one hand, bad configurations can often be characterized by minimal conditions that are possible to specify through a fixed number of *witness* processes. For instance, in a mutual exclusion protocol, a bad configuration contains *two* processes in their critical sections; and in a cache coherence protocol, a bad configuration contains *two* cache lines in their *exclusive* states. In both cases, having the two witnesses is sufficient to make the configuration bad (regardless of the actual size of the configuration). On the other hand, it is usually the case that such bad patterns (if existing) appear already in small instances of the system. View abstraction shows also that it is often the case that correctness can be established by only inspecting a small number of processes. We illustrate the method through an application to the classical Burns' mutual exclusion protocol.

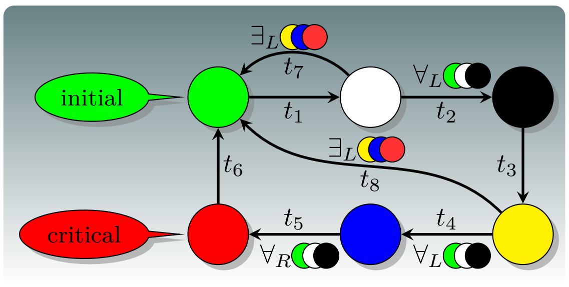


Figure 1 A process in Burns' Protocol.

Related Work.

Regular model checking [24, 15] performs parameterized verification by encoding the set of configurations using finite-state automata. The method has been augmented with techniques such as widening [13, 29], abstraction [14], and acceleration [9]. All these works rely on computing the transitive closure of transducers or on iterating them on regular languages.

There are numerous techniques less general than regular model checking, but that are lighter and more dedicated to the problem of parameterized verification. The idea of *counter abstraction* is to keep track of the number of processes which satisfy a certain property [22, 16, 17, 27]. In general, counter abstraction is designed for systems with unstructured or clique architectures, but may be used for systems with other topologies too.

Several works reduce parameterized verification to the verification of finite-state models. Among these, the *invisible invariants* method [11, 26] and the work of [25] exploit cut-off properties to check invariants for mutual exclusion protocols.

Monotonic abstraction [6, 7, 30] combines regular model checking with abstraction in order to produce systems that have monotonic behaviors wrt. a well quasi-ordering on the state-space. The method of [21, 20] and its reformulated, generic version of [19] come with a complete method for well-quasi ordered systems which is an alternative to backward reachability analysis based on a forward exploration.

Parameterized systems whose behaviors are conditioned by time or data constraints are described in [3, 1].

2 Model

We consider parameterized systems where the processes are modeled as finite-state automata arranged in a linear array. The processes may perform local or global (existential or universal) transitions. We illustrate our model through the classical Burns' protocol.

2.1 Processes

A process in Burns' protocol, depicted in Fig. 1, is defined by a finite-state automaton. The automaton has six states, namely 🟢, ⚪, ⚫, 🟡, 🔵, and 🔴. The process starts its execution form the *initial* state 🟢, and tries to reach its critical section 🔴. The automaton contains three types of transitions. From the state 🟢, the process can perform the *local* transition t_1 in which it changes state to ⚪ regardless of the states of the other processes. From the state ⚪, the process can perform the *existential global* transition t_7 in which it changes state to 🟢 provided that there *exists* a process to its *left* (and hence the notation $\exists_L$) whose state is either 🟡, 🔵, or 🔴. From the state ⚪, the process can also perform the *universal*

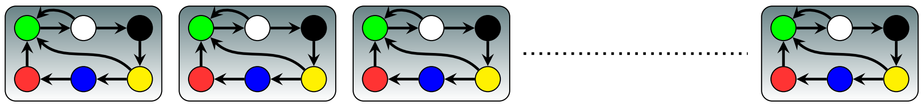


Figure 2 Parameterized Burns' Protocol.

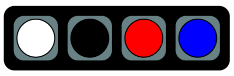


Figure 3 A configuration.

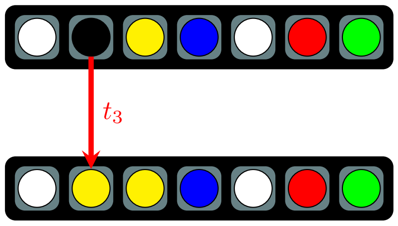


Figure 4 A local transition.

global transition t_2 in which it changes state to ⚫ provided that the states of *all* processes to its *left* (and hence the notation $\forall_L$) are either 🟢, ⚪, or ⚫. A process starts its execution from (state) 🟢 and can immediately cross to ⚪. At ⚪ it looks left and performs a test where it checks whether all processes to its left are in one of the states 🟢, ⚪, or ⚫. If the test succeeds it crosses to ⚫; otherwise it goes back to the initial state 🟢. Form ⚫ it can perform a local transition and move to 🟡. At 🟡, the process looks left again and performs the same test as before. If successful, it will now move to 🔵. At 🔵, the process now looks right, and checks whether all processes to its right are in one of the states 🟢, ⚪, or ⚫. If the test succeeds it crosses to its critical section 🔴, from which it can go back to the initial state 🟢.

The parameterized version of Burns' protocol (Fig. 2) consists of an arbitrary number of processes. The goal is to show that if we start from a configuration where all the processes are in state 🟢 then it is not possible to reach a configuration where two or more processes are in state 🔴.

2.2 Transition System

We define the transition system induced by the parameterized version of Burns' protocol. More precisely, we define the set of *configurations* and the *transition relation*.

A configuration gives the states of the processes in a given instance of the system. A configuration in Burns' protocol is depicted in Fig. 3, corresponding to an instance of the system with four processes that are in states ⚪, ⚫, 🔴, and 🔵 respectively. Notice that the set of configurations is infinite since there is no bound of the number of processes.

The transition relation is induced on the set of configurations by the above mentioned three types of transitions. When performing a transition, a process, called the *active process* changes state while the other processes remain passive (although their states may help enable/disable the move of the active process).

Fig. 4 depicts a local transition, where the active process performs t_3 and changes state from ⚫ to 🟡 while the states of the other processes remain unchanged.

An existential global transition is shown in Fig. 5. Here, the active process performs t_7 and changes state from ⚪ to 🟢. The transition is enabled since there is a witness in state 🟡 (marked by a green arrow). On the other hand, in Fig. 6, the transition is not enabled since there is no witness with the appropriate state to the left of the active process.

A universal global transition is shown in Fig. 7. The active process performs t_5 and changes state from 🔵 to 🔴. The transition is enabled since all processes to the right of the

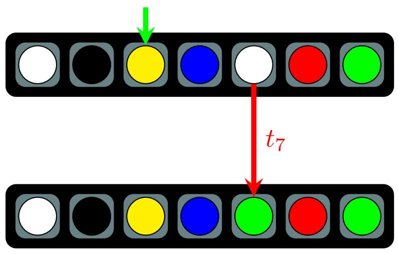


Figure 5 An existential transition.

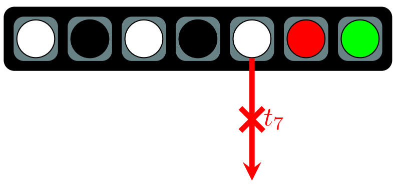


Figure 6 A disabled existential transition.

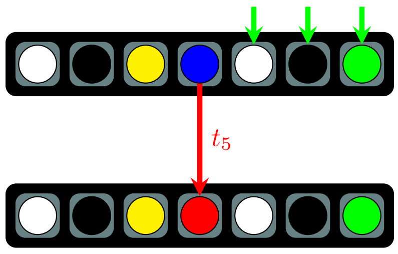


Figure 7 A universal transition.

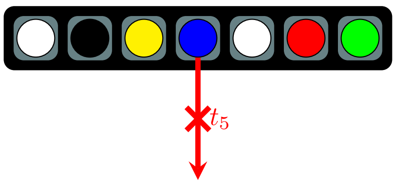


Figure 8 A disabled universal transition.

active process (marked by green arrows) are in one of the states 🟢, ◯, or ●, as required by the condition of the transition. In Fig. 8, the same transition is not enabled since there is one process in state 🔴 to the right of the active process (thus violating the condition of the transition).

For a configuration c, we use `post`(c) to be the set of configurations that we can reach from c through the application of a single transition.

2.3 Safety Properties

Checking a safety property amount to checking whether an instance of the system, starting from an *initial configuration*, can reach a *bad configuration* through repeated applications of the post operator.

In Burns' protocol, an initial configuration (Fig. 9) contains only processes in the state 🟢. The set `Init` of initial configurations is infinite since there is one initial configuration for each size of the system. the set `Init` can be characterized by a (very simple) regular expression, namely (🟢)$^+$, i.e., one or more processes in state 🟢.

A bad configuration (Fig. 10) is one which contains two or more processes in their critical sections (in state 🔴). This set `Bad` of bad configurations is also infinite. The set `Bad` *upward closed* wrt. the subword relation. Here, we say that a configuration c_1 is subword of a configuration c_2 if c_1 occurs (not necessarily contiguously) in c_2. Obviously if a configuration contains at least two processes in state 🔴 then any larger configuration (wrt. the subword relation) will also contain at least two processes in state 🔴, and hence belongs to the set of bad configurations. In fact, the set `Bad` is often characterized by its (finite) set $\texttt{Bad}_{min}$ of minimal elements. In the case of Burns' protocol, $\texttt{Bad}_{min}$ is the singleton containing the configuration 🔴🔴.

Mutual exclusion is a safety property. Checking it amounts to checking whether there is a sequence of transition that leads from an initial configuration to a bad configuration.

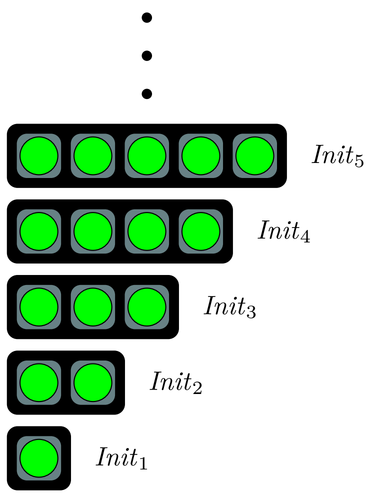


Figure 9 The set of initial configurations.

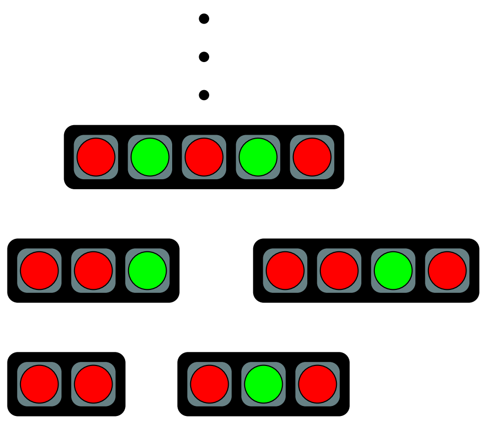


Figure 10 Examples of bad configurations.

3 Verification

We describe a scheme that allows to carry out parameterized verification automatically. The scheme consists of two procedures that can be performed in parallel, independently of each other. The first procedure is an under-approximation of the set of reachable configurations that is performed in the concrete domain. The second procedure is an over-approximation that is based on *view abstraction*. Both procedures are parameterized by a natural number $k \geq 1$ that defines the degree of the precision of the approximation.

3.1 Under-approximation

For a given $k \in \mathbb{N}$, we perform reachability analysis on the set of configurations of size k (Fig. 11). We start from the initial configuration of size k and generate all reachable configurations R_k of size k. This amounts to performing standard reachability analysis on a finite-state system since there are only finitely many configurations of size k. We can inspect R_k and search for bad configurations. We start from $k = 1$, and increase the value of k successively. If there is a bad configuration (of some size k) that is reachable, then it will be detected by the under-approximation procedure when inspecting R_k. Consequently, if the system does not satisfy the safety property then this will be reported by the under-approximation procedure. However, the procedure is not able to prove correctness of the system, since this would require computing R_k for all $k \in \mathbb{N}$.

3.2 Over-approximation

The over-approximation procedure is based on an abstract interpretation scheme, called *view abstraction*, that is parameterized by a natural number $k \in \mathbb{N}$. The concrete domain consists of the configurations of the system, while the abstract domain consists of objects that we call *views*. As we shall see below, each view is a subword of a configuration. We define an abstraction function α_k, a concretization function γ_k, and an abstract post operator $\texttt{Apost}_k$, all of which are parameterized by k.

For a configuration c, the abstraction $\alpha_k(c)$ is the set of views (subwords of c) of size up to k (Fig. 12). For a set C of configurations, we define $\alpha_k(C) := \cup_{c \in C} \alpha_k(c)$. Consider a set

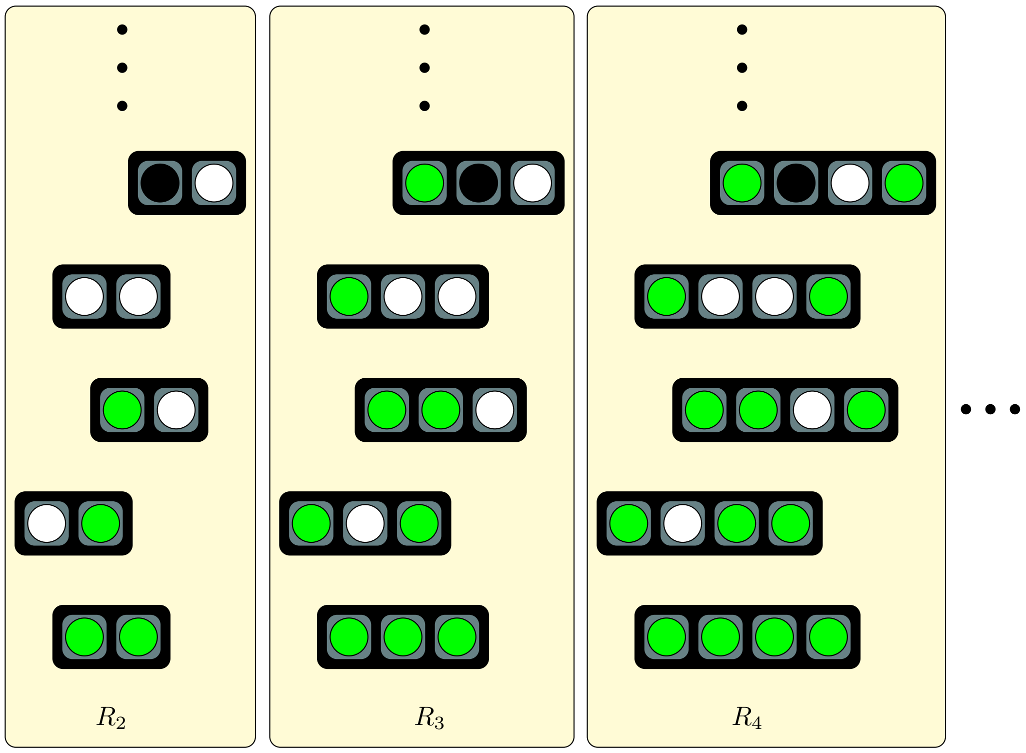


Figure 11 The under-approximation procedure.

of views of size up to k such that X is downward closed, i.e., if X contains a view x then each subword of x is also included in X. We define the *concretization* $\gamma_k(X)$ of X to be the set of configurations c such that that $\alpha_k(c) \subseteq X$, i.e. all members of the the abstraction of c are included in X (see Fig. 13.) Notice that, even for a finite set X and for a fixed $k \in \mathbb{N}$, the set $\gamma_k(X)$ is in general infinite (as is the case with $\gamma_2(X)$ in Fig. 13).

For $k \in \mathbb{N}$, we define the abstract post operator $\mathtt{Apost}$ such that for a set of views of size up to k, we have $\mathtt{Apost}_k(X) := \alpha_k(\mathtt{post}(\gamma_k(X)))$. In other words, we first take the concretization of X, then apply the concrete post operator, and finally take the abstraction of the result. We will preform reachability analysis on the set of views using a fixpoint iteration, parameterized with $k \in \mathbb{N}$. More precisely, we define the set $V_k := \mu X.\alpha_k(\mathtt{Init}) \cup \mathtt{Apost}_k(X)$. Before we describe how we compute V_k, we will give two of its properties (illustrated in Fig. 14). Let R be the set of reachable configurations. First, the concretization of V_k is an over-approximation of R, i.e., $R \subseteq \gamma_k(V_k)$ for all $k \in \mathbb{N}$. Furthermore, the precision of the abstraction increases with k in the sense that $\gamma_{k+1}(V_{k+1}) \subseteq \gamma_k(V_k)$ for all $k \in \mathbb{N}$. We use these two properties to define the following scheme for proving correctness of the system. Suppose that the system is correct, i.e., $R \cap \mathtt{Bad} = \emptyset$. We consider the sequence of sets $\gamma_1(V_1) \supseteq \gamma_2(V_2) \supseteq \gamma_3(V_3) \supseteq \cdots$. We start with $\gamma_1(V_1)$ and check whether $\gamma_1(V_1) \cap \mathtt{Bad} = \emptyset$. If the answer is positive then we know that the system is correct (since $R \subseteq \gamma_k(V_k)$ and hence $R \cap \mathtt{Bad} = \emptyset$.) On the other hand, if $\gamma_1(V_1) \cap \mathtt{Bad} \neq \emptyset$ (which is the case in Fig.14) then we are not sure whether $R \cap \mathtt{Bad} \neq \emptyset$ holds or not. Therefore, we increase k and repeat the procedure for $\gamma_2(V_2)$. In Fig. 14, the intersection $\gamma_2(V_2) \cap \mathtt{Bad}$ is still not empty. Therefore, we increase k yet again. The procedure will terminate if and when we reach a k where $\gamma_k(V_k) \cap \mathtt{Bad} = \emptyset$. In Fig. 14 such a k exists and $k = 3$.

Computing $\mathtt{Apost}_k(X)$ efficiently is not straightforward. The reason is that the set $\gamma_k(X)$ is in general infinite even if the set of views X is finite. We solve this problem by showing

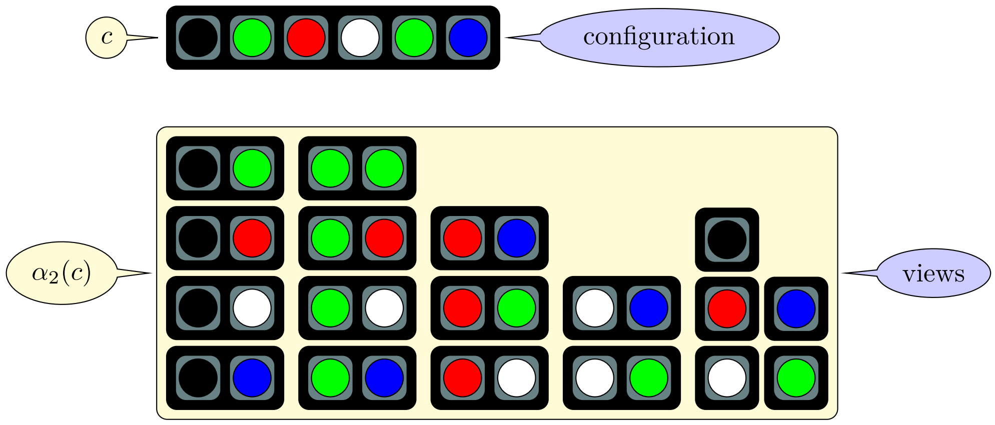


Figure 12 A configuration and its α_2-abstraction.

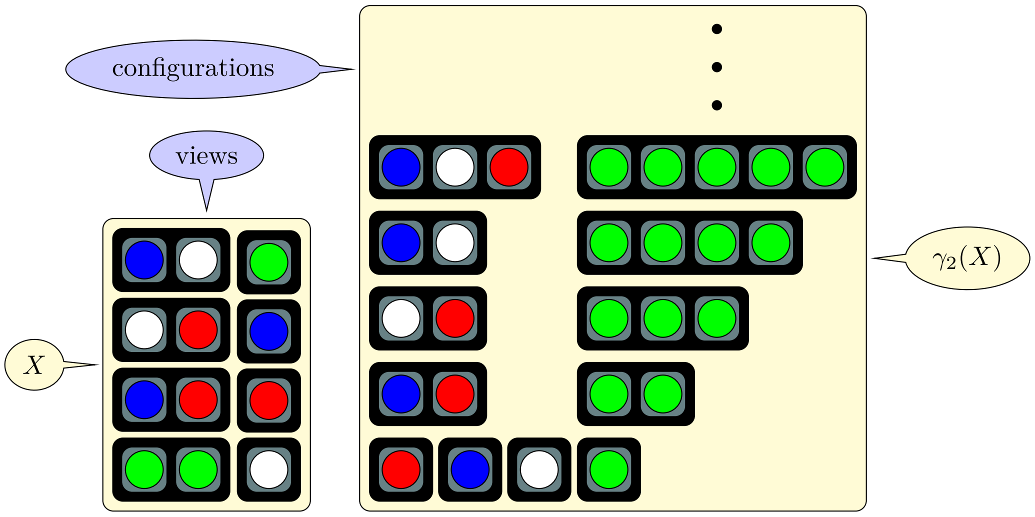
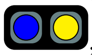
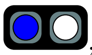
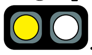
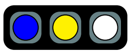
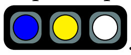
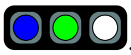
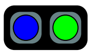
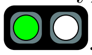
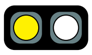


Figure 13 Concretization of a set of views.

that we only need to perform a simpler operation $\mathtt{Apost}_k^{k+1}(X)$. For $1 \leq \ell \leq k$, we define $\mathtt{Apost}_k^{\ell}(X) := \alpha_k(\mathtt{post}(\gamma_k^{\ell}(X)))$, where $\gamma_k^{\ell}(X)$ is the subset of $\gamma_k(X)$ containing only views of size up to ℓ. Notice that for any ℓ (and in particular for $\ell = k+1$), the set $\gamma_k^{\ell}(X)$ is finite and (easily) computable. We will illustrate why $\mathtt{Apost}_k(X) = \mathtt{Apost}_k^{k+1}(X)$ through the following example which shows a part of computing $\mathtt{Apost}_2(X)$. Assume that X contains (among others) the views , , and . We first compute the set $\gamma_2^3(X)$, i.e., we include all members of the concretization of X of size up to 3. For instance, the configuration  is a member of $\gamma_2^3(X)$. Then, we apply the concrete post operator $\mathtt{post}$ on the set $\gamma_2^3(X)$. In particular we apply $\mathtt{post}$ on the configuration . For instance, if the active process is then transition t_8 is enabled due to the existence of the witness to the left of the active process. As a result, we obtain the configuration . Finally, we apply the abstraction α_2, obtaining, among others, the new views  and . In particular, the latter view was obtained from the view  which

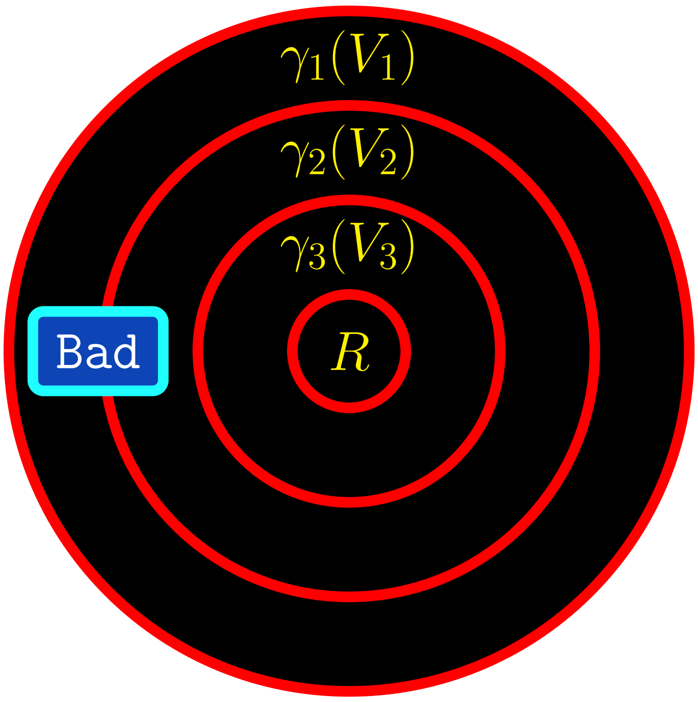


Figure 14 Abstract Analysis.

was part of X, through applying the transition t_8. There are two interesting aspects of this transition to observe. First, the transition needed the witness 🔵 which was not part of the original view. Second, the witness consists of a *single* process. Thus, in order to derive the new view 🟢⚪ we needed to add *one* extra process in order to accommodate the witness. This explains the reason why we need to consider concrete configurations of larger sizes than the views (to ensure the inclusion the witnesses), but also the reason why we need to only consider configurations whose sizes are larger by *one* (since witnesses are of size one).

3.3 Scheme

Our verification scheme consists of performing a number of iterations, where each iteration corresponds to a particular value of $k \in \mathbb{N}$ (Fig. 15). We start with $k = 1$. During each iteration, we run the under- and over-approximation procedures for the given value of k. If the under-approximation procedure is conclusive, i.e., $R_k \cap \texttt{Bad} \neq \emptyset$ then we terminate and declare the system *unsafe*. Otherwise, we run the over-approximation procedure to compute V_k. If this is conclusive, i.e., $V_k \cap \texttt{Bad} = \emptyset$ then we terminate and declare the system *safe*. Notice that if either of the two procedures is conclusive then the current k is a *cut-off* point beyond which we need not continue (since we have either concluded correctness or incorrectness of the system). If none of the procedures is conclusive, we increase the precision, by increasing the value of k, and perform the next iteration.

We show how to perform each operation that is required for implement the scheme.

- *Checking whether* $R_k \cap \texttt{Bad} \neq \emptyset$. As mentioned above, computing R_k amount to performing reachability analysis on a finite-state system. Furthermore, to check $R_k \cap \texttt{Bad} \neq \emptyset$ we need only to consider elements of `Bad` whose sizes are up to k. The latter is finite and easily computable since `Bad` is upward closed. Thus, we need to consider the intersection of two finite sets.

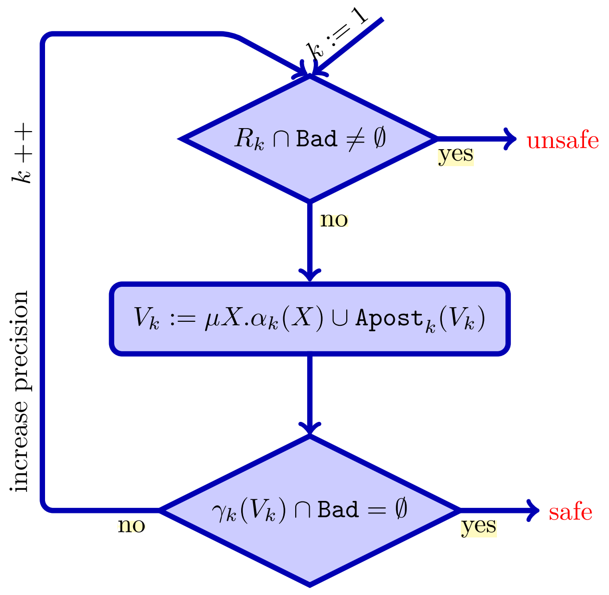

$k := 1$
$R_k \cap \texttt{Bad} \neq \emptyset$
yes
unsafe
no
$V_k := \mu X.\alpha_k(X) \cup \texttt{Apost}_k(V_k)$
$\gamma_k(V_k) \cap \texttt{Bad} = \emptyset$
yes
safe
no
$k++$
increase precision

Figure 15 Abstract Analysis.

- *Computing* V_k. We can compute the fixpoint as follows.
  - Since the set of initial configurations $\texttt{Init}$ is regular, computing $\alpha_k(\texttt{Init})$, for any $k \in \mathbb{N}$, amounts to generating the subwords of members of $\texttt{Init}$ of size up to k. This is a task that can accomplished using simple automata operations.
  - As mentioned above, for a given set of views X, we need only to compute $\texttt{Apost}_k^{k+1}(X)$ rather than $\texttt{Apost}_k(X)$. The former can be computed as follows: (i) we compute the finite set of configurations $C := \gamma_k^{k+1}(X)$ by matching the different members of X. (ii) We compute the set $C' := \texttt{post}(C)$ by applying the transition relation on C. Notice that C' is finite. (iii) We compute the finite set of views $V' := \alpha_k(C')$ which amounts to computing the abstraction of a finite set.
- *Checking whether* $V_k \cap \texttt{Bad} = \emptyset$. This amounts to checking whether there is a minimal configuration c in $\texttt{Bad}$ (i.e., $c \in \texttt{Bad}_{min}$) such that $\alpha_k(c) \subseteq V_k$. Recall that $\texttt{Bad}_{min}$ is finite and given, and hence we can perform the test by systematically going through all members of $\texttt{Bad}_{min}$.

4 Application

We illustrate the verification scheme by applying it to Burns' protocol.

Iteration 1: Under-Approximation.

We start from the initial configuration 🟩 of size one. The set R_1 of reachable configurations of size one can be shown to contain all configurations of size one. However, none of these configurations belongs to $\texttt{Bad}$ since each member of $\texttt{Bad}$ contains at least two processes (in state 🔴). Hence, the under-approximation scheme is inconclusive for $k = 1$.

Iteration 1: Over-approximation.

The set $\alpha_1(\texttt{Init})$ contains a single view, namely 🟢. This will be added to V_1. In order to compute the fixpoint, we apply $\texttt{Apost}_1^2$ repeatedly. Let us consider its first application. The set γ_1^2 contains the two configurations 🟢 and 🟢🟢. Applying $\texttt{post}$ to these two configurations gives the configurations ⚪, 🟢⚪, and ⚪🟢. Applying α_2 on the derived set of configurations gives the set containing the two views 🟢 and ⚪ that will now be added to V_1. Applying $\texttt{Apost}_1^2$ repeatedly in this manner will yield the set of all views of size one. Applying γ_2 to this set gives the set of *all* configurations, and in particular this set will intersect with the set $\texttt{Bad}$. Therefore, the over-approximation scheme is inconclusive for $k = 1$.

Iteration 2: Under-Approximation.

We start from the initial configuration 🟢🟢 of size two. The set R_2 of reachable configurations of size two can be computed using finite-state reachability analysis. We leave the computation of the set (as an easy exercise) to the reader. None of the configurations in R_2 belongs to $\texttt{Bad}$, and hence, the under-approximation scheme is inconclusive also for $k = 2$.

Iteration 2: Over-approximation.

The set $\alpha_2(\texttt{Init})$ contains the two views 🟢 and 🟢🟢. Thus, these views will be added to V_2. In order to compute the fixpoint, we apply $\texttt{Apost}_2^3$ repeatedly. Let us consider its first application. The set γ_2^3 contains the configurations 🟢, 🟢🟢, and 🟢🟢🟢. Applying $\texttt{post}$ to these three configurations gives the configurations ⚪, 🟢⚪, ⚪🟢, ⚪🟢🟢, 🟢⚪🟢, and 🟢🟢⚪. Applying α_2 to the derived set of configurations gives the set containing the views 🟢 , ⚪, ⚪🟢, 🟢⚪, and 🟢🟢. Applying $\texttt{Apost}_2^3$ repeatedly will yield the set of all views of size two *except* the views 🔴🔵 and 🔴🔴. In particular, the absence of the second view implies that applying γ_2 gives a set of configurations that does not intersect with the set $\texttt{Bad}$. This means that we can terminate and conclude that the system is safe. Notice that the cut-off point for Burns' protocol is $k = 2$.

5 Completeness

We will give examples of systems for which our method is complete (for which the scheme is guaranteed to terminate). First, we give the definitions of upward and downward closed sets, and then give a sufficient condition that guarantees termination. Finally, we describe a class of systems, namely *monotonic systems*, that satisfy the condition.

5.1 Downward and Upward Closed Sets

Let $\preceq$ be the subword relation on configurations. The relation $\preceq$ is a *well quasi-ordering* [23]: for any infinite sequence $c_0, c_1, c_2, \ldots$ of configurations, there are $i < j$ such that $c_i \preceq c_j$. A set D of configurations is said to be *downward closed* if $c_1 \in D$ and $c_2 \preceq c_1$ implies $c_2 \in D$. A set U of configurations is said to be *upward closed* if $c_1 \in U$ and $c_1 \preceq c_2$ implies $c_2 \in U$. For a set C of configurations, we define $C\!\uparrow := \{c_2 \mid \exists c_1 \in C.\, c_1 \preceq c_2\}$ to be the upward closure of C. Let $\min(C)$ be the set of minimal elements of C. The set $\min(C)$ is always finite. Notice that, for an upward closed set U, we have that $U\!\uparrow = U$, and that U can be characterized by its minimal elements in the sense that $U = \min(U)\!\uparrow$. For instance, the set `Bad` in Burns' protocol is upward closed, and $\min(\texttt{Bad})$ is a singleton containing the configuration 🔴🔴. Observe that the complement $\neg D$ of a downward closed set D is upward closed, and vice versa.

5.2 Sufficient Condition

Let D be a set of configuration. We say that D is a *good downward closed invariant* if it satisfies the following conditions: (i) *downward closed*: D is downward closed; and (ii) *invariant*: D is inductive, i.e., D contains the set `Init` of initial configuration, and D is closed under the application of the transition relation (for any $c \in D$, a transition from c leads to a configuration inside D again.) (iii) *good*: $D \cap \texttt{Bad} = \emptyset$. If D satisfies conditions (i) and (ii) then we can show that there is a k such that $\gamma_k(V_k) \subseteq D$. In fact, we can define k to be the size of a largest configuration in $\min(\neg D)$, i.e., the size of a largest configuration in the minimal set of configurations in the complement of D. Recall that $\neg D$ is upward closed. If D also satisfies condition (iii) then the over-approximation procedure is guaranteed to terminate. More precisely, $D \cap \texttt{Bad} = \emptyset$ implies that $\gamma_k(V_k) \cap \texttt{Bad} = \emptyset$, which means that the procedure declares correctness of the system at step k (if not earlier). As a side remark, notice also that if the set R of reachable configurations is downward closed then $\gamma_k(V_k) = R$ for some k (we know from the previous section that $R \subseteq \gamma_k(V_k)$ for all k.)

We will motivate that if R is downward closed then our procedure is guaranteed to terminate implying its completeness. We consider two cases. If $R \cap \texttt{Bad} \neq \emptyset$, then there is a k such that $R_k \cap \texttt{Bad} \neq \emptyset$ and the under-approximation procedure declares the system unsafe during the k^{th} iteration. If $R \cap \texttt{Bad} \neq \emptyset$ then since R is an invariant, i.e., it satisfies the conditions (i), (ii), and (iii), and hence the over-approximation procedure will terminate.

In the case of Burns' protocol, the set R is characterized by the regular expression (🟢 + ⚪ + ⚫ + 🟡 + 🔵)* $\cdot$ (🔴 + ϵ) $\cdot$ (🟢 + ⚪ + ⚫ + 🟡)*. Our method will not compute this regular expression explicitly. However, the language of the expression is downward-closed and is equal to the set $\neg$ { 🔴🔴 , 🔴🔵 }$\uparrow$. The size of the largest elements in the set of minimal configurations is equal to 2, which explains why the over-approximation procedure terminates at $k = 2$ for Burns' protocol.

5.3 Monotonic Systems

A system is said to be *monotonic* if, for all configurations c_1, c_2, c_3, whenever $c_2 \in \texttt{post}(c_1)$ and $c_1 \preceq c_3$ then there is a configurations c_4 such that $c_2 \preceq c_4$ and $c_4 \in \texttt{post}(c_3)$. In other words, the relation $\preceq$ is a simulation wrt. the transition relation. For monotonic systems, it is the case that $R \cap \texttt{Bad} = \emptyset$ iff $R\!\downarrow \cap \texttt{Bad} = \emptyset$. This follows from the assumption that `Bad` is an upward closed set. Therefore, taking the downward closure of the set of reachable configurations does not cause any imprecision. This means that we can work with a new transition relation in which we allow the system to be *lossy*: a configuration may, at any point of time, lose an arbitrary number of processes. The new transition relation will reach `Bad` if and only if the old one does. Furthermore, the new set of reachable configurations is downward closed which means that our procedure is guaranteed to terminate. In fact, there is a wide class of systems that induce transition relations that are monotonic with respect to a well quasi-ordering. Our scheme is complete for such systems. Examples include lossy channel systems [8], Petri nets (our procedure solves the coverability problem for them), timed Petri nets [10], etc.

6 Conclusion

We have presented a method for automatic verification of parameterized systems. The method proves or refutes whether a given safety property is satisfied by only inspecting small instances of the system. More precisely, we run two procedures in parallel. The first computes the (finite) set of reachable configurations of size up k for some natural number k. The second procedure carries out an abstract interpretation scheme, called *view abstraction*, in which precision is defined (and can be increased) using a natural number k. The latter is guaranteed to terminate in case there is a good invariant that is downward closed. This implies that the whole method is guaranteed to terminate in case the set of reachable configurations is downward closed and hence that termination is guaranteed in case the transition relation is monotonic on the set of configurations.

The framework can be extended in a straightforward manner to other types of topologies such as multisets, rings, and trees, and also extended to the case where global transitions are not assumed to happen atomically [4].

References

1 P. A. Abdulla and G. Delzanno. On the coverability problem for constrained multiset rewriting. In *Proc. AVIS'06, 5th Int. Workshop on on Automated Verification of Infinite-State Systems*, 2006.

2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Carl Leonardsson, and Ahmed Rezine. Counter-example guided fence insertion under tso. In *Proc. TACAS '08, 18th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *Lecture Notes in Computer Science*, 2012.

3 Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. Multi-clock timed networks. In *Proc. LICS '04, 20th IEEE Int. Symp. on Logic in Computer Science*, pages 345–354, 2004.

4 Parosh Aziz Abdulla, Frédéric Haziza, and Lukás Holík. All for the price of few (parameterized verification through view abstraction). In *VMCAI*, volume 7737 of *LNCS*, pages 476–495, 2013.

5 Parosh Aziz Abdulla, Frédéric Haziza, Lukás Holík, Bengt Jonsson, and Ahmed Rezine. An integrated specification and verification technique for highly concurrent data structures. In *TACAS*, volume 7795 of *LNCS*, pages 324–338, 2013.

6 Parosh Aziz Abdulla, Noomene Ben Henda, Giorgio Delzanno, and Ahmed Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *Proc. TACAS '07, 13th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 721–736. Springer Verlag, 2007.

7 Parosh Aziz Abdulla, Noomene Ben Henda, Giorgio Delzanno, and Ahmed Rezine. Handling parameterized systems with non-atomic global conditions. In *Proc. VMCAI '08, 9th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, 2008.

8 Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *LICS*, pages 160–170. IEEE Computer Society, 1993.

9 Parosh Aziz Abdulla, Axel Legay, Julien d'Orso, and Ahmed Rezine. Simulation-based iteration of tree transducers. In *Proc. TACAS '05, 11th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *Lecture Notes in Computer Science*, 2005.

10 Parosh Aziz Abdulla and Aletta Nylén. Timed Petri nets and BQOs. In *Proc. ICATPN'2001: 22nd Int. Conf. on application and theory of Petri nets*, volume 2075 of *Lecture Notes in Computer Science*, pages 53 –70, 2001.

11 T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions. In Berry, Comon, and Finkel, editors, *Proc. 13th Int. Conf. on Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 221–234, 2001.

12 M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *POPL*, pages 7–18. ACM, 2010.

13 Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large. In *Proc. 15th Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235, 2003.

14 A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *Proc. 16th Int. Conf. on Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386, Boston, July 2004. Springer Verlag.

15 D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, 2001.

16 G. Delzanno. Automatic verification of cache coherence protocols. In Emerson and Sistla, editors, *Proc. 12th Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 53–68. Springer Verlag, 2000.

17 Giorgio Delzanno. Verification of consistency protocols via infinite-state symbolic model checking. In *FORTE'00*, volume 183 of *IFIP Conference Proceedings*, pages 171–186. Kluwer, 2000.

18 J. Esparza and S. Schwoon. A bdd-based model checker for recursive programs. In *CAV*, volume 2102 of *LNCS*, pages 324–336. Springer, 2001.

19 Pierre Ganty, Jean-François Raskin, and Laurent Van Begin. A Complete Abstract Interpretation Framework for Coverability Properties of WSTS. In *VMCAI'06*, volume 3855 of *LNCS*, pages 49–64. Springer, 2006.

20 Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Expand, enlarge and check... made efficient. In *CAV'05*, volume 3576 of *LNCS*, pages 394–407. Springer, 2005.

21 Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Expand, enlarge and check: New algorithms for the coverability problem of wsts. *J. Comput. Syst. Sci.*, 72(1):180–203, 2006.

22 S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.

23 G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7):326–336, 1952.

24 Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256:93–112, 2001.

25 Kedar S. Namjoshi. Symmetry and completeness in the analysis of parameterized systems. In *VMCAI'07*, volume 4349 of *LNCS*, pages 299–313. Springer, 2007.

26 A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In *Proc. TACAS '01, 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031, pages 82–97, 2001.

27 A. Pnueli, J. Xu, and L. Zuck. Liveness with (0,1,infinity)-counter abstraction. In *Proc. 14th Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, 2002.

28 Susmit Sarkar, Peter Sewell, Jade Alglave, Luc Maranget, and Derek Williams. Understanding power multiprocessors. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 175–186, 2011.

29 T. Touili. Regular Model Checking using Widening Techniques. *Electronic Notes in Theoretical Computer Science*, 50(4), 2001. Proc. of VEPAS'01.

30 N. Yonesaki and T. Katayama. Functional specification of synchronized processes based on modal logic. In *IEEE 6th International Conference on Software Engineering*, pages 208–217, 1982.

Parameter synthesis for probabilistic real-time systems*

Marta Kwiatkowska

Department of Computer Science, University of Oxford
Department of Computer Science
University of Oxford
Parks Road
Oxford OX1 3QD
UK
marta.kwiatkowska@cs.ox.ac.uk

Abstract

The parameter synthesis problem aims to find parameter valuations that guarantee that a given objective is satisfied for a parametric model. Applications range from automated model repair to optimisation. This lecture will focus on models with probability and real-time and give an overview of recent results concerning parameter synthesis from quantitative temporal logic objectives. Existing algorithmic approaches and experimental results will be discussed, and future research challenges outlined.

This lecture is based on [2, 1, 3].

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases quantitative verification, timed automata, parameter synthesis

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.16

Category Invited Paper

References

1 Milan Ceska, Frits Dannenberg, Marta Z. Kwiatkowska, and Nicola Paoletti. Precise parameter synthesis for stochastic biochemical systems. In Pedro Mendes, Joseph O. Dada, and Kieran Smallbone, editors, *Proceedings of the 12th International Conference in Computational Methods in Systems Biology (CMSB 2014)*, volume 8859 of *Lecture Notes in Computer Science*, pages 86–98. Springer, 2014.

2 Marco Diciolla, Chang Hwan Peter Kim, Marta Z. Kwiatkowska, and Alexandru Mereacre. Synthesising optimal timing delays for timed I/O automata. In Tulika Mitra and Jan Reineke, editors, *Proceedings of the 14th International Conference on Embedded Software (EMSOFT 2014)*, pages 16:1–16:10. ACM, 2014.

3 Aleksandra Jovanović and Marta Z. Kwiatkowska. Parameter synthesis for probabilistic timed automata using stochastic game abstractions. In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *Proceedings of the 8th International Workshop on Reachability Problems (RP 2014)*, volume 8762 of *Lecture Notes in Computer Science*, pages 176–189. Springer, 2014.

* This work is supported by ERC Advanced Grant VERIWARE and Proof of Concept Grant VERIPACE.

Consistency for Parametric Interval Markov Chains

Benoît Delahaye

Université de Nantes / LINA
Nantes, France
`benoit.delahaye@univ-nantes.fr`

Abstract

Interval Markov Chains (IMCs) are the base of a classic probabilistic specification theory by Larsen and Jonsson in 1991. They are also a popular abstraction for probabilistic systems. In this paper we introduce and study an extension of Interval Markov Chains with parametric intervals. In particular, we investigate the consistency problem for such models and propose an efficient solution for the subclass of parametric IMCs with local parameters only. We also show that this problem is still decidable for parametric IMCs with global parameters, although more complex in this case.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.1.1 Models of Computation, G.3 Probability and Statistics

Keywords and phrases specification, parameters, Markov chains, consistency

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.17

1 Introduction

Interval Markov Chains (IMCs for short) extend Markov Chains, by allowing to specify intervals of possible probabilities on state transitions instead of precise probabilities. IMCs have been introduced by Larsen and Jonsson [16] as a *specification* formalism—a basis for a stepwise-refinement-like modeling method, where initial designs are very abstract and underspecified, and then they are made continuously more precise, until they are concrete. Unlike richer specification models such as Constraint Markov Chains [6] or Abstract Probabilistic Automata [9], IMCs are difficult to use for compositional specification due to lack of basic modeling operators. Nevertheless, IMCs have been intensively used in order to model real-life systems in domains such as systems biology, security or communication protocols [2, 12, 5, 19, 11].

The extension of Markov Chains into Interval Markov chains was motivated by the fact that, when modelling real-life systems, the actual exact value of transition probabilities may not be known precisely. Indeed, in most cases, these values are measured from observations or experiments which are subject to imprecision. In this case, using intervals of probabilities that take into account the precision of the measures makes more sense than using an arbitrary but precise value. We now take this reasoning a step further.

Complex systems are most often built by assembling multiple components. Assume that one of these components may fail with a given probability that depends on the quality of the materials involved in its fabrication. In practice, a prototype of the component is built and the failure probability of this component is measured by experiment with a certain

This work has been partially supported by project PACS ANR-14-CE28-0002-04.

imprecision. This failure probability and the subsequent imprecision are then taken into account in the model of the system by using an interval of probability. When one analyzes this model, every result will depend on the failure rate of this component, which itself depends on the choice of the quality of materials. If the conclusions of the analysis are that the failure probability is too high for the whole system to be viable, then a new prototype of the failing component is built using different materials and the modeling and analysis phase starts over. This process is repeated until a satisfactory failing component is identified.

Instead of using this “trial and error” methodology, we propose a new extension of Interval Markov Chains that allows using parameters in the definition of intervals of probability. In our example, the failure probability of the failing component is clearly a parameter of the system. The developer is interested in whether there exists a maximal value of this probability that will ensure that the whole system is satisfactory. When this value is identified, one can choose the materials of the failing component accordingly in order to produce a prototype with a lower maximal failing probability.

Therefore, we introduce in this paper the new formalism called parametric Interval Markov Chains (pIMCs), which extends IMCs by allowing the use of parameters as lower or upper endpoints of probability intervals. We also show that the problem of deciding whether a given pIMC is consistent (i.e. admits a valid implementation) is decidable and propose algorithms in order to solve this problem. In particular, we identify a subclass of pIMCs – *local pIMCs* – for which an efficient algorithm is proposed. In the rest of the paper, we limit ourselves to closed intervals. Nevertheless, all the results we propose can be extended with minor modifications to open/semi-open intervals whose lower/upper endpoints contain linear combinations of parameters and constants.

Related work. To the best of our knowledge, there is no existing work on parametric probabilistic specification theories as such, where parameters range over probability values. Still, classes of systems where parameters give some latitude on probability distributions, such as parametric Markov models [17] have been studied in the literature [18, 13]. The activity in this domain has yielded decidability results [15], parametric probabilistic model-checking algorithms [8] and even tools [14]. Continuous-time parametric and probabilistic models have also been considered in some very restricted settings [7]. Networks of probabilistic processes where the number of processes is a parameter have also been studied in [3, 4], and probabilistic timed automata with parameters in clock constraints and invariants have been studied in [1].

The paper proceeds as follows. In Section 2, we begin by introducing concepts and notations that will be used throughout the paper. Section 3 introduces the new formalism of parametric Interval Markov Chains, studies their relations to (Interval) Markov Chains and discusses what we call the *range* of parameters. In Section 4, we present original solutions to the consistency problem for IMCs and pIMCs. Finally, Section 5 concludes the paper and discusses future work.

2 Background

Throughout the paper, we use the notion of parameters. A parameter $p \in P$ is a variable ranging through the interval $[0,1]$. A valuation for P is a function $\psi : P \to [0,1]$ that associates values with each parameter in P. We write $\texttt{Int}_{[0,1]}(P)$ for the set of all closed intervals of the form $[x,y]$ with $x, y \in [0,1] \cup P$. When $P = \emptyset$, we write $\texttt{Int}_{[0,1]} = \texttt{Int}_{[0,1]}(\emptyset)$ to denote closed intervals with real-valued endpoints. Given an interval I of the form

$I = [a,b]$, $\texttt{Low}(I)$ and $\texttt{Up}(I)$ respectively denote the lower and upper endpoints of I, i.e. a and b. Given an interval $I = [a,b] \in \texttt{Int}_{[0,1]}$, we say that I is well-formed whenever $a \leq b$. In the following, we abuse notations and write $\emptyset$ for the empty interval, meaning any not well-formed interval. Given a finite set S, we write $\texttt{Dist}(S)$ for the set of distributions over S, i.e. the set of functions $\rho : S \to [0,1]$ such that $\sum_{s \in S} \rho(s) = 1$. In the rest of the paper, we assume that all the states in our structures are equipped with labels taken from a fixed set of atomic propositions A. A state-labelling function over S is thus a function $V : S \to 2^A$ that assigns to each state a set of labels from A.

We recall the notion of Markov Chains (MCs), that will act as models for (parametric) IMCs. An example of a Markov Chain is given in Figure 1a.

▶ **Definition 1** (Markov Chain)**.** A Markov Chain is a tuple $\mathcal{M} = (S, s_0, M, A, V)$, where S is a finite set of states containing the initial state s_0, A is a set of atomic propositions, $V : S \to 2^A$ is a labeling function, and $M : S \times S \to [0,1]$ is a probabilistic transition function such that $\forall s \in S, \sum_{t \in S} M(s,t) = 1$.

We now recall the notion of Interval Markov Chain (IMC), adapted from [10]. IMCs are a specification formalism that allows one to represent an infinite set of MCs. Roughly, IMCs extend MCs by replacing exact probability values on transitions with intervals of allowed probability values. An example of an IMC is given in Figure 1b.

▶ **Definition 2** (Interval Markov Chain [10])**.** An Interval Markov Chain (IMC) is a tuple $\mathcal{I} = (S, s_0, \varphi, A, V)$, where S, s_0, A and V are as for MCs, and $\varphi : S \times S \to \texttt{Int}_{[0,1]}$ is a transition constraint that associates with each potential transition an interval of probabilities.

The following definition recalls the notion of satisfaction introduced in [10]. Satisfaction (also called implementation in some cases) allows to characterize the set of MCs represented by a given IMC specification. Crucially, satisfaction abstracts from the syntactic structure of transitions in IMCs: a single transition in the implementation MC can contribute to satisfaction of more than one transition in the specification IMC, by distributing its probability mass against several transitions. Similarly many MC transitions can contribute to satisfaction of just one specification transition.

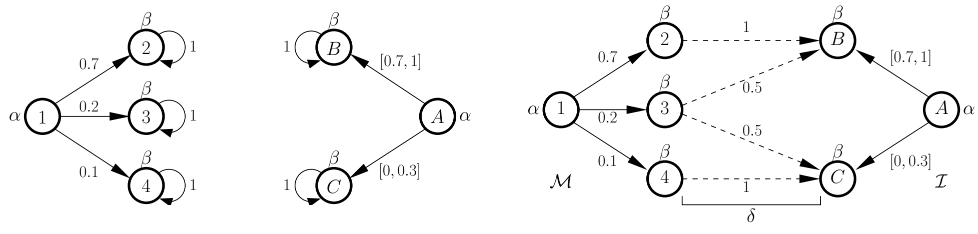
▶ **Definition 3** (Satisfaction Relation [10])**.** Let $\mathcal{I} = (S, s_0, \varphi, A, V^I)$ be an IMC and $\mathcal{M} = (T, t_0, M, A, V^M)$ be a MC. A relation $\mathcal{R} \subseteq T \times S$ is a *satisfaction relation* if whenever $t\mathcal{R}s$,

1. the valuations of s and t agree: $V^M(t) = V^I(s)$,
2. there exists a function $\delta : T \to (S \to [0,1])$ such that
   a. for all $t' \in T$ such that $M(t,t') > 0$, $\delta(t')$ is a distribution on S,
   b. for all $s' \in S$, we have $(\sum_{t' \in T} M(t,t') \cdot \delta(t')(s')) \in \varphi(s,s')$, and
   c. for all $t' \in T$ and $s' \in S$, if $\delta(t')(s') > 0$, then $(t',s') \in \mathcal{R}$.

   We say that $\mathcal{M}$ satisfies $\mathcal{I}$ (written $\mathcal{M} \models \mathcal{I}$) iff there exists a satisfaction relation containing (t_0, s_0).

The set of MCs satisfying a given IMC $\mathcal{I}$ is written $[\![\mathcal{I}]\!]$. Formally, $[\![\mathcal{I}]\!] = \{\mathcal{M} \mid \mathcal{M} \models \mathcal{I}\}$. In the rest of the paper, we write $\perp$ for the *empty* IMC, i.e.$\perp = (\emptyset, \emptyset, \emptyset, A, \emptyset)$. By construction, we have $[\![\perp]\!] = \emptyset$.

The notion of satisfaction between the MC $\mathcal{M}$ from Figure 1a and the IMC $\mathcal{I}$ from Figure 1b is illustrated in Figure 1c.



(a) A Markov Chain $\mathcal{M}$. **(b)** An IMC $\mathcal{I}$. **(c)** An example of satisfaction relation.

Figure 1 Markov Chain, Interval Markov Chain and satisfaction relation [10].

3 Parametric Interval Markov Chains

In this section, we propose a new formalism, called parametric Interval Markov Chains (pIMC) that extends IMCs by allowing parameters as the lower/upper endpoints of the transition intervals. We start by giving the main definitions of pIMCs and their relations with IMCs and MCs, and then distinguish two subclasses of interest of pIMCs: *local* and *global* pIMCs.

3.1 pIMCs and their relations to IMCs/MCs

We now propose an extension of IMCs that allows using parameters in the definition of intervals.

▶ **Definition 4** (Parametric Interval Markov Chain). A parametric Interval Markov Chain (pIMC) is a tuple $\mathcal{I}^P = (S, s_0, \varphi_P, A, V, P)$, where S, s_0, A and V or as for IMCs, P is a set of variables (parameters) ranging over $[0, 1]$ and $\varphi_P : S \times S \to \texttt{Int}_{[0,1]}(P)$ associates to each potential transition a (parametric) interval.

In the following, we abuse notations and also write $\perp$ for the *empty* pIMC, i.e. $\perp = (\emptyset, \emptyset, \emptyset, A, \emptyset, \emptyset)$.

Roughly, an instance of a pIMC $\mathcal{I}^P$ is a pair $(\mathcal{I}, \psi)$, where $\mathcal{I}$ is an IMC that respects the structure and labels of $\mathcal{I}^P$ and such that its transition constraint is the instantiation of φ_P according to the valuation for the parameters ψ.

▶ **Definition 5** (Instance of a pIMC). An instance of pIMC $\mathcal{I}^P = (S, s_0, \varphi_P, A, V, P)$ is a pair $(\mathcal{I}, \psi)$ (written $(\mathcal{I}, \psi) \vdash \mathcal{I}^P$), where $\mathcal{I} = (S, s_0, \varphi, A, V)$ is an IMC respecting the structure and labels of $\mathcal{I}^P$, $\psi : P \to [0, 1]$ is a valuation for the parameters, and $\varphi \equiv \varphi_P[p \leftarrow \psi(p)]$.

We sometimes write $\mathcal{I} \vdash_\psi \mathcal{I}^P$ instead of $(\mathcal{I}, \psi) \vdash \mathcal{I}^P$ and say that $\mathcal{I}$ is an instance of $\mathcal{I}^P$ through ψ. We say that $\mathcal{I}$ is an instance of $\mathcal{I}^P$, written $\mathcal{I} \vdash \mathcal{I}^P$, whenever there exists a valuation ψ such that $\mathcal{I} \vdash_\psi \mathcal{I}^P$.

A MC $\mathcal{M} = (T, t_0, M, A, V^M)$ implements pIMC $\mathcal{I}^P$, written $\mathcal{M} \models \mathcal{I}^P$, iff there exists an instance $\mathcal{I}$ of $\mathcal{I}^P$ such that $\mathcal{M} \models \mathcal{I}$. We write $[\![\mathcal{I}^P]\!]$ for the set of MCs implementing $\mathcal{I}^P$.

▶ **Example 6.** Consider pIMC $\mathcal{I}^P$ given in the left of Figure 2. $\mathcal{I}^P$ represents a family of dispensable probabilistic beverage machines (dpbm) that have a probability greater or equal to 0.5 of delivering tea and a probability lower or equal to 0.5 of delivering coffee. In addition, we use parameter p to model the fact that the machine can fail to deliver anything with probability at most p. The value of p depends on the quality of a potentially failing

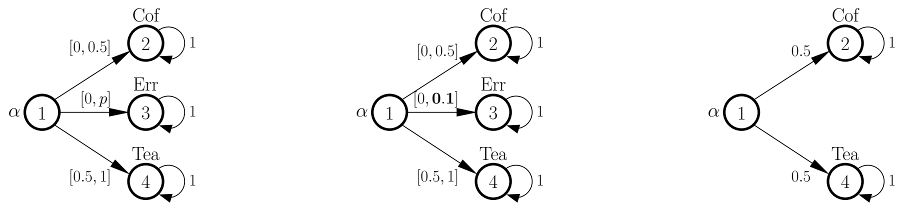


Figure 2 pIMC $\mathcal{I}^P$ (left) with one of its instances $\mathcal{I}$ (middle) and an implementation $\mathcal{M}$ (right).

component. The IMC $\mathcal{I}$ given in the middle of Figure 2 depicts the family of dpbm for which the potentially failing component has a maximal failure probability of 0.1. Finally, MC $\mathcal{M}$ given in the right of Figure 2 depicts a given dpbm of this family, where the actual probabilities of delivering tea and coffee are fixed to 0.5 and 0.5 respectively, and where the potentially failing component does not fail.

As for IMCs, one question of interest for pIMCs is to decide whether they admit at least one implementation – the so-called consistency problem. Given the definition of implementation, deciding whether a pIMC is consistent amounts to verifying whether it admits at least one instance that is itself consistent. Nevertheless, we will see in Section 4, that in the case of local pIMCs, consistency can be decided using a polynomial algorithm on the pIMC itself without having to go through any of its instances.

3.2 Local VS Global Parameters

We now propose two subclasses that distinguish different trends in the use of parameters throughout a given structure. Parameters can be used at two different levels in a given pIMC: either in a local fashion – reflecting small tuning artifacts in a model; or in a global fashion – reflecting potential design choices. In the following, we formally define these subclasses.

Local parameters. Parameters are said to be *local* if they only appear in transition probabilities outgoing from a unique state. In this sense they reflect small tuning artifacts because of their small impact on the structure of the pIMC. The pIMC $\mathcal{I}^P$ in Figure 2 illustrates this notion: in $\mathcal{I}^P$, parameter p is local as it only appears in transitions outgoing from a single state (State 1). In essence, p models the failure probability of a single component, only used once in pIMC $\mathcal{I}^P$.

We write $\mathtt{Range}(p)$ for the range of a given parameter p, i.e. the set of states s such that p is either the lower or the upper endpoint of the probability interval associated with an outgoing transition of s. Formally, given pIMC $\mathcal{I}^P = (S, s_0, \varphi_P, A, V, P)$, $\mathtt{Range}_{\mathcal{I}^P}(p) = \{s \in S \mid \exists s' \in S \text{ s.t. } p \in \mathtt{Low}(\varphi_P(s,s')) \cup \mathtt{Up}(\varphi_P(s,s'))\}$. When clear from the context, we write $\mathtt{Range}(p)$ instead of $\mathtt{Range}_{\mathcal{I}^P}(p)$. We say that a parameter $p \in P$ is *local* in $\mathcal{I}^P$ iff $|\mathtt{Range}(p)| \leq 1$. A pIMC $\mathcal{I}^P = (S, s_0, \varphi_P, A, V, P)$ is *local* iff all its parameters are local.

Since all parameters are local in local pIMCs, it is very easy to check whether the outgoing transitions of a given state are *consistent* in the sense that it is possible to find out easily whether there exist values of the parameters such that the outgoing intervals of a given state are not empty.

Global parameters. Parameters are *global* if they are not local, i.e. if they appear in the outgoing probability intervals of at least two states.

Formally, we say that parameter $p \in P$ is *global* in $\mathcal{I}^P$ iff $|\mathtt{Range}_{\mathcal{I}^P}(p)| > 1$. We say that pIMC $\mathcal{I}^P = (S, s_0, \varphi_P, A, V, P)$ is *global* iff at least one of its parameters is global.

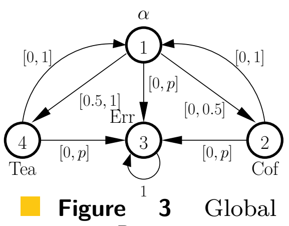


Figure 3 Global pIMC $\mathcal{I}_2^P$ with global parameter p.

▶ **Example 7.** pIMC $\mathcal{I}_2^P$ from Figure 3 depicts a family of beverage machines in which all modules use the same potentially failing component. The maximal probability of failure of this component is modeled using a parameter p. This parameter is *global* in $\mathcal{I}_2^P$ as it appears in the outgoing transitions of several states (States 1, 2, 4). In $\mathcal{I}_2^P$, the choice of a value for p has more impact on the potential behaviors of the global pIMC than in the case of pIMC $\mathcal{I}^P$ from Figure 2, where parameter p was only local.

In the case of global pIMCs, checking whether the outgoing transitions of a given state are consistent becomes more tricky, since the potential values of the parameters may be subject to constraints coming from other states.

4 Consistency

As said in Section 3, one question of interest given a pIMC $\mathcal{I}^P$ is to decide whether it admits at least one instance that itself admits at least one implementation. This is what we call the consistency problem. In this section, we start by recalling the consistency problem in the case of IMCs and solutions to this problem that have been proposed in the literature. We propose an alternative solution to the consistency problem for IMCs and then extend it to the case of local pIMCs. Finally, we show that the problem is more complex in the case of pIMCs with global parameters.

4.1 Consistency of IMCs

The consistency problem for IMCs has already been studied in the literature [10] and it has been proven that it is decidable and can be solved in polynomial time. We first recall one of the existing algorithms and then propose an alternative, more direct solution.

In [10], the consistency problem for IMCs has been considered as a special case of the *common implementation* problem, which consists in deciding, given a finite number of IMCs, whether there exists at least one implementation satisfying them all. One can solve the consistency problem for a given IMC $\mathcal{I}$ by deciding whether $\mathcal{I}$ admits a common implementation with itself. The proposed solution to the consistency problem is based on the notion of consistency relation, also introduced in [10]. It is shown that an IMC $\mathcal{I}$ is consistent iff there exists a consistency relation between $\mathcal{I}$ and itself, which can be decided in polynomial time.

As explained in [10], a consistency relation allows one state of a given IMC to contribute to the consistency of other states. Although this was justified by the fact that satisfaction abstracts from the structure of transitions in IMCs, we show in the following theorem that whenever an IMC is consistent, it admits one implementation with the same structure. As a consequence, one transition in this implementation only contributes to satisfying the exact same transition in the specification IMC, which will allow us to avoid the use of consistency relations in the rest of the paper.

▶ **Theorem 8.** *An IMC $\mathcal{I} = (S, s_0, \varphi, A, V)$ is consistent iff it admits an implementation of the form $\mathcal{M} = (S, s_0, M, A, V)$ where, for all reachable state s in $\mathcal{M}$, it holds that $M(s, s') \in \varphi(s, s')$ for all s'.*

Proof. Let $\mathcal{I} = (S, s_0, \varphi, A, V)$ be an IMC. One direction of this theorem is trivial: if $\mathcal{I}$ admits an implementation of the form $\mathcal{M} = (S, s_0, M, A, V)$ where, for all reachable state s in $\mathcal{M}$, it holds that $M(s, s') \in \varphi(s, s')$ for all s', then $\mathcal{I}$ is consistent. We now prove the other direction.

Assume that $\mathcal{I}$ is consistent and let $\mathcal{M}' = (T, t_0, M', A, V')$ be a MC such that $\mathcal{M}' \models \mathcal{I}$ with satisfaction relation $\mathcal{R}$. From $\mathcal{M}'$, we build a new implementation of $\mathcal{I}$ of the desired form. Let $f : S \to T$ be a function that associates to each state of $\mathcal{I}$ one of the states in $\mathcal{M}'$ contributing to its satisfaction, if any. Formally, f is such that for all $s \in S$, if $f(s)$ is defined, then $(f(s), s) \in \mathcal{R}$. Let $\delta_{(f(s),s)}$ be the function given by $\mathcal{R}$ (item 2 of Definition 3). We now define the desired implementation $\mathcal{M} = (S, s_0, M, A, V)$. Let $S' = \{s \in S \mid \exists t \in T, (t, s) \in \mathcal{R}\}$ and $M(s, s') = \sum_{t \in T} \delta_{f(s),s}(t)(s') \cdot M'(f(s), t)$ if $s \in S'$ and 0 otherwise.

We observe that, by definition of $\mathcal{R}$ we have $M(s, s') \in \varphi(s, s')$ for all $(s, s') \in S' \times S$. Moreover, whenever $M(s, s') > 0$, there exists at least one state $t \in T$ such that $\delta_{f(s),s}(t)(s') > 0$ and $M'(f(s), t) > 0$. Therefore, by definition of δ, we have $(t, s') \in \mathcal{R}$ and thus $s' \in S'$. It thus follows that only states from S' can be reachable in $\mathcal{M}$.

Consider the identity relation $\mathcal{R}'$ over S' and let $(s, s) \in \mathcal{R}'$. Let $\delta' : S \to (S \to [0, 1])$ be such that $\delta'(s')(s'') = 1$ whenever $s' \in S'$ and $s'' = s'$, and 0 otherwise.

- Let $s' \in S$ be such that $M(s, s') > 0$. By construction, we have $s' \in S'$ and thus $\delta'(s')$ is a distribution on S.
- Let $s' \in S$ and consider $\sum_{s'' \in S} M(s, s'') \cdot \delta(s'')(s')$.
  - If $s' \notin S'$, then $\sum_{s'' \in S} M(s, s'') \cdot \delta'(s'')(s') = 0$ and we know by $\mathcal{R}$ that $0 \in \varphi(s, s')$ (because there is no $t \in T$ such that $\delta(t)(s') > 0$).
  - Otherwise, we have $\sum_{s'' \in S} M(s, s'') \cdot \delta'(s'')(s') = M(s, s') \in \varphi(s, s')$
- For all $s', s'' \in S$ such that $\delta'(s')(s'') > 0$, we have $s' = s''$ and $s' \in S'$, therefore $(s', s'') \in \mathcal{R}'$.

We conclude that $\mathcal{R}'$ is a satisfaction relation between $\mathcal{M}$ and $\mathcal{I}$. Moreover, we know by construction that $(t_0, s_0) \in \mathcal{R}$, thus $s_0 \in S'$. ◀

The fact that a consistent IMC necessarily admits an implementation with the same structure implies that using a cross-product such as introduced in the notion of consistency relation in order to prove consistency is not necessary. Therefore, one does not need to search for local inconsistencies in $S \times S$, as is done in [10], but only needs to check and avoid local inconsistencies on S.

We thus propose an alternative solution to the consistency problem for IMCs. Our solution is based on the notion of pruning. The aim of pruning is to detect and remove from a given structure all the states that cannot contribute to any of its implementations. Such states are called *inconsistent*. The algorithm we propose will follow the same principle: it will detect and propagate local inconsistencies (i.e. states whose transition intervals cannot be satisfied) through the state-space of the IMC until either the initial state is locally inconsistent – the IMC is thus inconsistent – or only consistent states are reachable, implying that the IMC is consistent. Because of Theorem 8, an implementation of the original IMC $\mathcal{I}$ can be directly derived from its pruned version.

A pruning algorithm was also proposed in [10], but it was based on the notion of consistency relation, therefore using the cross-products we are trying to avoid. In [10], a quadratic number of iterations is needed in order to build the consistency relation, each iteration being itself quadratic in the number of states. A linear number of iterations is then needed in order to prune the consistency relation. In contrast, the algorithm we propose in the following only needs a linear number of iterations, each iteration being linear itself.

The pruning operator we propose is based on the notion of local state-consistency.

▶ **Definition 9.** Given an IMC $\mathcal{I} = (S, s_0, \varphi, A, V)$, a state $s \in S$ is *locally consistent* if there exists a distribution $\rho \in \mathtt{Dist}(S)$ such that for all $s' \in S$, $\rho(s') \in \varphi(s, s')$.

Being able to check whether a given state in an IMC is locally consistent is thus of paramount importance. Fortunately, this can be done quite easily: checking whether a state is locally consistent amounts to solving a set of linear inequations. Indeed, assuming that $S = \{s_0, s_1, \ldots s_n\}$, checking whether $s_i \in S$ is consistent amounts to deciding whether the following system of inequations admits a solution.

$$\exists x_0, \ldots x_n,\ x_0 + \ldots + x_n = 1 \wedge\ x_0 \in \varphi(s_i, s_0) \wedge \ldots \wedge\ x_n \in \varphi(s_i, s_n)$$

In fact, one does not need to solve the system in order to decide whether it admits a solution. If φ contains intervals that are not well-formed, then s_i is trivially inconsistent. Otherwise, assuming all the intervals in φ are well-formed, then one only needs to check whether the sum of all lower endpoints is below 1 and whether the sum of all upper endpoints is above 1.

▶ **Proposition 10.** *Given an IMC $\mathcal{I} = (S, s_0, \varphi, A, V^I)$, a state $s \in S$ is locally consistent iff $\varphi(s, s')$ is well-formed for all s', and $\sum_{s' \in S} \mathit{Low}(\varphi(s, s')) \leq 1 \leq \sum_{s' \in S} \mathit{Up}(\varphi(s, s'))$.*

Checking whether a state is locally consistent can thus be done in linear time. Once locally inconsistent states have been identified, they will be made unreachable in $\mathcal{I}$ by iterating the following pruning operator β. In the following, we say that a state s of IMC $\mathcal{I} = (S, s_0, \varphi, A, V^I)$ is *inconsistent* iff there is no implementation of $\mathcal{I}$ in which s is satisfied. In practice, s is inconsistent iff it is locally inconsistent or there are transitions with non-zero probability leading from s to another inconsistent state s', i.e. such that $0 \notin \varphi(s, s')$. In order to keep track of inconsistent states that have already been processed, we equip IMCs with a marking function $\lambda : S \to \{0, 1\}$. States s such that $\lambda(s) = 1$ are inconsistent states that have already been identified and made unreachable in a previous iteration of β. The notion of satisfaction is not impacted by this marking function.

▶ **Definition 11** (Pruning operator β for IMCs)**.** Let $\mathcal{I} = (S, s_0, \varphi, A, V, \lambda)$ be an IMC. The pruning operator β is defined as follows. Let $\lambda_0(S) = \{s \in S \mid \lambda(s) = 0\}$.

1. If $\lambda_0(S)$ does not contain any locally inconsistent state or if $\mathcal{I} = \bot$, then $\beta(\mathcal{I}) = \mathcal{I}$.
2. Else, if s_0 is locally inconsistent, then $\beta(\mathcal{I}) = \bot$.
3. Otherwise, let $s_i \in \lambda_0(S)$ be a new locally inconsistent state in $\mathcal{I}$. We then define $\beta(\mathcal{I}) = (S, s_0, \varphi', A, V, \lambda')$, with $\lambda'(s_i) = 1$ and $\lambda'(s) = \lambda(s)$ for all $s \neq s_i$, $\varphi'(s, s') = \varphi(s, s')$ if $s' \neq s_i$, and
$$\varphi'(s, s_i) = \begin{cases} \varphi(s, s_i) & \text{if } \lambda(s) = 1 \\ [0, 0] & \text{if } \lambda(s) = 0 \text{ and } 0 \in \varphi(s, s_i) \\ \emptyset & \text{otherwise} \end{cases}$$

As seen in the above definition, the pruning operator does not remove inconsistent states but makes them unreachable. When 0 is an allowed probability for incoming transitions, β enforces this choice by modifying the subsequent intervals to $[0, 0]$. When 0 is not allowed, then the only possibility is to modify the interval probabilities to $\emptyset$, which propagates local inconsistency to predecessors. The first application of β should always be done with an empty marking function, i.e. assigning 0 to all states.

Since pruning potentially propagates local inconsistencies to predecessor states, the pruning operator β has to be applied iteratively until it converges to a fixpoint. The IMC obtained in this fixpoint is either $\perp$ or an IMC with no reachable locally inconsistent states (item 1 of Definition 11 above). Since at least one inconsistent state is detected and made unreachable at each iteration (items 2 and 3 of Definition 11), the number of iterations needed in order to converge is bounded by $|S|$. The complexity of applying pruning to $\mathcal{I}$ until it converges is thus polynomial. The result of this iteration on IMC $\mathcal{I}$ is written $\beta^*(\mathcal{I})$ in the rest of the document.

▶ **Theorem 12.** *For all IMC $\mathcal{I} = (S, s_0, \varphi, A, V)$ and marking function λ such that $\lambda(s) = 0$ for all $s \in S$, it holds that $[\![\beta^*((S, s_0, \varphi, A, V, \lambda))]\!] = [\![\mathcal{I}]\!]$.*

Proof. Let $\mathcal{I} = (S, s_0, \varphi, A, V)$ be an IMC and let λ be a Marking function such that $\lambda(s) = 0$ for all $s \in S$. Let $\mathcal{I}' = (S, s_0, \varphi', A, V, \lambda') = \beta^n((S, s_0, \varphi, A, V, \lambda))$ for some $n \in \mathbb{N}$. We show that for all MC $\mathcal{M}$, we have $\mathcal{M} \models \mathcal{I}' \iff \mathcal{M} \models \beta(\mathcal{I}')$.

If $\{s \in S \mid \lambda'(s) = 0\}$ does not contain any inconsistent state or if s_0 is inconsistent, then the result is trivial. We thus assume that an inconsistent state $s_i \in \{s \in S \mid \lambda'(s) = 0\}$ is found and made unreachable by β.

We start by observing that, by construction, all states $s \in S$ such that $\lambda'(s) = 1$ are such that for all $s' \in S$ with $\lambda(s') = 0$, we have either $\varphi'(s', s) = [0, 0]$ or $\varphi'(s', s) = \emptyset$.

$\boxed{\Rightarrow}$ Let $\mathcal{M} = (T, t_0, M, A, V^M)$ be a MC such that $\mathcal{M} \models \mathcal{I}'$. Let $\mathcal{R}$ be the associated satisfaction relation. We show that $\mathcal{R}$ is still a satisfaction relation between $\mathcal{M}$ and $\beta(\mathcal{I}') = (S, s_0, \varphi'', A, V, \lambda'')$. Let $(t, s) \in \mathcal{R}$.

1. Since β has no effect on valuations, we still have $V^M(t) = V(s)$.
2. Let $\delta : T \to (S \to [0, 1])$ be the function given by $\mathcal{R}$. By construction, it holds that
   a. for all $t' \in T$ such that $M(t, t') > 0$, $\delta(t')$ is a distribution on S,
   b. for all $s' \in S$, we have $(\sum_{t' \in T} M(t, t') \cdot \delta(t')(s')) \in \varphi'(s, s')$, and
   c. for all $t' \in T$ and $s' \in S$, if $\delta(t')(s') > 0$, then $(t', s') \in \mathcal{R}$.

Items 2.*a*. and 2.*c*. are not impacted by β. We now show that Item 2.*b*. still holds.

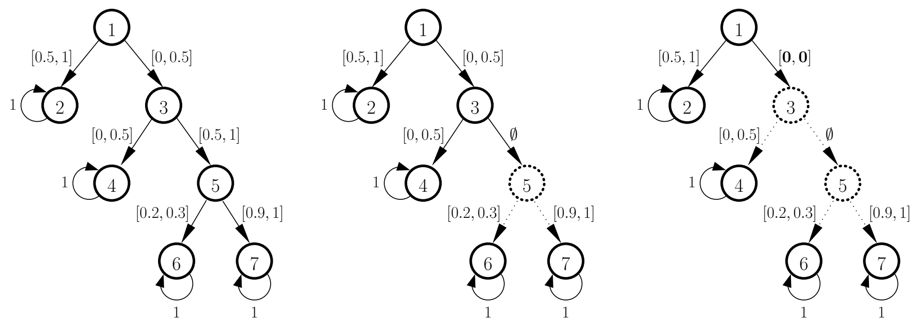
For all $s' \in S$ such that $s' \neq s_i$, we have $\varphi''(s, s') = \varphi'(s, s')$ and Item 2.*b*. still holds. Furthermore, since s_i is inconsistent in $\mathcal{I}'$, we necessarily have that for all $t' \in T$, $(t', s_i) \notin \mathcal{R}$, and thus $\delta(t')(s_i) = 0$. Therefore, we have $(\sum_{t' \in T} M(t, t') \cdot \delta(t')(s_i)) = 0$.

- If s is such that $0 \in \varphi'(s, s_i)$, then we still have $0 \in \varphi''(s, s_i)$ since $\varphi''(s, s_i)$ is either $\varphi'(s, s_i)$ or $[0, 0]$.
- Otherwise, if $0 \notin \varphi'(s, s_i)$, then we have $(\sum_{t' \in T} M(t, t') \cdot \delta(t')(s')) \notin \varphi'(s, s')$, which is a contradiction w.r.t. the definition of $\mathcal{R}$. As a consequence, there exists no $t \in T$ such that $(t, s) \in \mathcal{R}$ and the modification of $\varphi'(s, s_i)$ into $\varphi''(s, s_i) = \emptyset$ has no consequence on Item 2.*b*.

Finally, $\mathcal{R}$ is still a satisfaction relation between $\mathcal{M}$ and $\beta(\mathcal{I}')$ and therefore $\mathcal{M} \models \beta(\mathcal{I}')$.

$\boxed{\Leftarrow}$ Let $\mathcal{M} = (T, t_0, M, A, V^M)$ be a MC such that $\mathcal{M} \models \beta(\mathcal{I}') = (S, s_0, \varphi'', A, V, \lambda'')$. Let $\mathcal{R}$ be the associated satisfaction relation. We show that $\mathcal{R}$ is also a satisfaction relation between $\mathcal{M}$ and $\mathcal{I}'$. Let $(t, s) \in \mathcal{R}$ and let $\delta : T \to (S \to [0, 1])$ be the function given by $\mathcal{R}$. As above, β has no effect on valuations and on Items 2.*a*. and 2.*c*. of the definition of a satisfaction relation. We show that Item 2.*b*. also holds between $\mathcal{M}$ and $\mathcal{I}'$.

For all $s' \in S$ such that $s' \neq s_i$, we have $\varphi''(s, s') = \varphi'(s, s')$ and Item 2.*b*. trivially holds. Furthermore, since s_i is inconsistent in $\mathcal{I}$, it is also inconsistent in $\mathcal{I}'$. As a consequence, we necessarily have that for all $t' \in T$, $(t', s_i) \notin \mathcal{R}$, and thus $\delta(t')(s_i) = 0$. Therefore, we have $(\sum_{t' \in T} M(t, t') \cdot \delta(t')(s_i)) = 0$.



(a) An IMC $\mathcal{I}$. **(b)** $\beta(\mathcal{I})$. **(c)** $\beta^2(\mathcal{I}) = \beta^*(\mathcal{I})$.

Figure 4 Iterative application of the pruning operator β to IMC $\mathcal{I}$ until convergence.

- If s is such that $0 \in \varphi'(s, s_i)$, then $(\sum_{t' \in T} M(t,t') \cdot \delta(t')(s_i)) \in \varphi'(s, s_i)$ and Item 2.*b*. holds.
- Otherwise, if $0 \notin \varphi'(s, s_i)$, then we have $\varphi''(s, s_i) = \emptyset$. As a consequence $(\sum_{t' \in T} M(t,t') \cdot \delta(t')(s_i)) \notin \varphi''(s, s_i)$, which is a contradiction. Therefore, there exists no $t \in T$ such that $(t, s) \in \mathcal{R}$ and $0 \notin \varphi'(s, s_i)$.

Finally, $\mathcal{R}$ is also a satisfaction relation between $\mathcal{M}$ and $\mathcal{I}'$ and therefore $\mathcal{M} \models \mathcal{I}'$. ◀

▶ **Example 13.** Figure 4 illustrates the iteration of pruning operator β on an IMC. Consider IMC $\mathcal{I}$ from Figure 4a. Applying β on $\mathcal{I}$ consists in two steps: (1) searching for a locally inconsistent state in $\mathcal{I}$, and (2) modifying $\mathcal{I}$ in order to make the selected locally inconsistent state unreachable. At first, the only locally inconsistent state in $\mathcal{I}$ is State 5. As a consequence, applying β will either reduce all incoming interval transition probabilities to $[0, 0]$ when 0 is already allowed or to $\emptyset$ when this is not the case. The only incoming transition for State 5 is equipped with interval $[0.5, 1]$, therefore it is replaced with $\emptyset$. $\beta(\mathcal{I})$ is then depicted in Figure 4b. In the second iteration of β, State 3 is identified as locally inconsistent because it has an outgoing transition equipped with $\emptyset$. State 3 only has one incoming transition, which is equipped with interval $[0, 0.5]$. Since this interval contains 0, it is replaced with $[0, 0]$. $\beta^2(\mathcal{I})$ is represented in Figure 4c. Since $\beta^2(\mathcal{I})$ does not contain any reachable locally inconsistent state, the fixed point is reached and $\beta^*(\mathcal{I}) = \beta^2(\mathcal{I})$.

4.2 Consistency of pIMCs

We now move to the setting of pIMCs. Recall that a pIMC $\mathcal{I}^P$ is consistent iff it admits at least one consistent instance, i.e. $\exists \mathcal{I}, \exists \mathcal{M} \mid \mathcal{M} \models \mathcal{I}$ and $\mathcal{I} \vdash \mathcal{I}^P$.

As we will see later, the difficulty of deciding whether a given pIMC $\mathcal{I}^P$ is consistent highly depends on the nature of the parameters in $\mathcal{I}^P$. This is due to the fact that the notion of local state-consistency only makes sense for states whose transition probability intervals only contain local parameters. Indeed, in the case of global parameters, the local consistency of one state might be incompatible with the local consistency of another due to the incompatible choice of parameter valuations. In the following, we propose an intuitive and efficient solution for deciding whether a local pIMC is consistent. We then show that

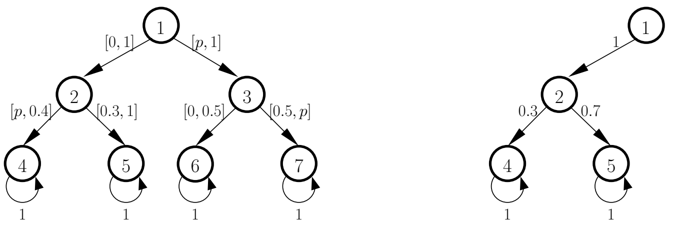


Figure 5 Global pIMC $\mathcal{I}^P$ with global parameter p (left) and one of its implementations (right).

consistency is also decidable in the case of global parameters although the algorithm we propose is more complex.

▶ **Example 14.** Consider pIMC $\mathcal{I}^P$ given on the left of Figure 5. Parameter p in $\mathcal{I}^P$ is global as it affects outgoing transitions of States 1, 2 and 3. If one is checking local state-consistency, it looks like all states in $\mathcal{I}^P$ are locally consistent. Indeed, outgoing transitions of State 1 can be satisfied with $p = 0$; outgoing transitions of State 2 can be satisfied with $p = 0$; and outgoing transitions of State 3 can be satisfied with $p = 1$. From a purely local point of view, it thus seems that $\mathcal{I}^P$ is consistent. However, it also appears that State 2 requires that $p \leq 0.4$ while State 3 requires that $p \geq 0.5$. One could therefore conclude that $\mathcal{I}^P$ is inconsistent. Despite of this fact, we claim that $\mathcal{I}^P$ is consistent: if p is set to 0, then we can reduce the transition interval from State 1 to State 3 to $[0, 0]$, which makes State 3 unreachable. Therefore, one no longer needs to have $p \geq 0.5$ and a correct implementation of $\mathcal{I}^P$ can be found. Such an implementation is Given on the right of Figure 5.

Consistency of local pIMCs. In order to check consistency of local pIMCs, a similar algorithm to the one used for checking consistency of IMCs can be used. In fact, due to Theorem 8, one does not need to consider particular instances of $\mathcal{I}^P$ in order to find out whether $\mathcal{I}^P$ is consistent: since all instances of $\mathcal{I}^P$ share the same structure, $\mathcal{I}^P$ will be consistent iff there exists an implementation that shares this structure. Since the notion of local state-consistency makes sense in the case of local pIMCs, we adapt the pruning algorithm presented in Section 4.1 to local pIMCs.

Let $\mathcal{I}^P = (S, s_0, \varphi_P, A, V, P)$ be a local pIMC and let $s \in S$. We write $\texttt{param}(s) = \{p \in P \mid \texttt{Range}(p) = \{s\}\}$ for the set of parameters appearing in the outgoing transition intervals of s. We then say that s is *locally consistent* iff there exists a valuation ψ over $\texttt{param}(s)$ and a distribution $\rho \in \texttt{Dist}(S)$ such that for all $s' \in S$, $\rho(s') \in \varphi_P(s, s')[p \leftarrow \psi(p)]$. Recall that the only parameters potentially appearing in $\varphi_P(s, s')$ are necessarily from $\texttt{param}(s)$.

In a similar fashion to the case of IMCs, local consistency of state $s_i \in S$ can be reduced to checking whether a system of inequations admits a solution. In order to facilitate presentation, we assume that $S = \{s_1, \ldots, s_n\}$ and we use the parameters in $\texttt{param}(s_i)$ as variables taking values in $[0, 1]$. The system is then as follows:

$$\sum_{j=1}^{n} \texttt{Low}(\varphi_P(s_i, s_j)) \leq 1 \wedge \sum_{j=1}^{n} \texttt{Up}(\varphi_P(s_i, s_j)) \geq 1 \wedge \texttt{Low}(\varphi_P(s_i, s_1)) \leq \texttt{Up}(\varphi_P(s_i, s_2)) \\ \wedge \ldots \wedge \texttt{Low}(\varphi_P(s_i, s_n)) \leq \texttt{Up}(\varphi_P(s_i, s_n))$$

In this system, the first two inequations reflect the definition of local state-consistency while the other inequations ensure that all the intervals expressed using parameters are well-formed. In the case of IMCs, we were able to remove this check by assuming beforehand that our IMCs were well-formed. In the case of pIMCs, we cannot assume the same as well-formedness will depend on the actual value given to parameters. Nevertheless, solving such a system of inequations can be done in polynomial time w.r.t. $|S|$ and $|P|$.

We now propose a pruning algorithm for local pIMCs based on the notion of local state-consistency. The outline of this algorithm is similar to the algorithm for IMCs, and only the modification of interval probabilities following the discovery of a new locally inconsistent state s_i is slightly modified: We start by identifying the set of parameters appearing as the lower bound of a transition interval leading to s_i and then enforce the value of these parameters to be 0 in order to be able to make s_i unreachable. Formally, we write $\texttt{enforce}(s_i) = \{p \in P \mid \exists s \in S, p = \texttt{Low}(\varphi_P(s, s_i))\}$ for this set of parameters. As for IMCs, we use a marking function $\lambda : S \to \{0, 1\}$ in order to keep track of locally inconsistent states that have already been processed. The notions of instantiation and satisfaction are not impacted by this marking function.

▶ **Definition 15** (Pruning operator β for pIMCs)**.** Let $\mathcal{I}^P = (S, s_0, \varphi_P, A, V, P, \lambda)$ be a pIMC. The pruning operator β for pIMCs is defined as follows. Let $\lambda_0(S) = \{s \in S \mid \lambda(s) = 0\}$.

1. If $\lambda_0(S)$ does not contain any locally inconsistent state or if $\mathcal{I}^P = \perp$ then $\beta(\mathcal{I}^P) = (\mathcal{I}^P)$.
2. Else, if s_0 is locally inconsistent, then $\beta(\mathcal{I}^P) = \perp$.
3. Otherwise, let $s_i \in \lambda_0(S)$ be a new locally inconsistent state in $\mathcal{I}^P$. We then define $\beta(\mathcal{I}^P) = (S, s_0, \varphi'_P, A, V, P, \lambda')$, with $\lambda'(s_i) = 1$ and $\lambda'(s) = \lambda(s)$ for all $s \neq s_i$, $\varphi'_P(s, s') = \varphi_P(s, s')[\texttt{enforce}(s_i) \leftarrow 0]$ if $s' \neq s_i$, and

$$\varphi'_P(s, s_i) = \begin{cases} \varphi_P(s, s_i)[\texttt{enforce}(s_i) \leftarrow 0] \text{ if } \lambda(s) = 1 \\ [0, 0] \text{ if } \lambda(s) = 0 \text{ and } \varphi_P(s, s_i)[\texttt{enforce}(s_i) \leftarrow 0] = [0, .] \\ \emptyset \text{ otherwise} \end{cases}$$

As for IMCs, the pruning operator β for pIMCs propagates local inconsistencies to predecessor states. Therefore, β has to be applied iteratively until a fixpoint if reached. The pIMC obtained in this fixpoint is either $\perp$ or a pIMC with no reachable locally inconsistent state (item 1 of Definition 15). Since at least one inconsistent state is identified and made unreachable at each iteration (items 2 and 3 of Definition 15), the number of iterations needed in order to converge is bounded by $|S|$. Therefore, the complexity of applying pruning to a given local pIMC until convergence is polynomial in $|S|$ and $|P|$. The result of this iteration on pIMC $\mathcal{I}^P$ is written $\beta^*(\mathcal{I}^P)$.

▶ **Example 16.** Figure 6 illustrates the pruning operator for local pIMCs. Consider local pIMC $\mathcal{I}^P$ given in the left of Figure 6. We start by searching for locally inconsistent states in $\mathcal{I}^P$: State 3 is chosen. The first application of pruning operator β will therefore try to make State 3 unreachable by forcing all incoming transition intervals to $[0, 0]$. This can only be done if either 0 is already the lower bound of the incoming interval or if a parameter p is the lower bound of the incoming interval and p can be forced to 0 throughout the whole pIMC. In $\mathcal{I}^P$, State 3 only has one incoming transition, which is equipped with interval $[p, 1]$. Parameter p is thus forced to 0 in all other transitions and the incoming interval to State 3 is reduced to $[0, 0]$. The result $\beta(\mathcal{I}^P)$ is given in the right of Figure 6. Since there are no more locally inconsistent states in $\beta(\mathcal{I}^P)$, we have $\beta^*(\mathcal{I}^P) = \beta(\mathcal{I}^P)$.

We now show that the result of iterating β on a given pIMC $\mathcal{I}^P$ is a pIMC with the same set of implementations as $\mathcal{I}^P$.

▶ **Theorem 17.** *For all local pIMC $\mathcal{I}^P = (S, s_0, \varphi_P, A, V, P)$ and marking function λ such that $\lambda(s) = 0$ for all $s \in S$, it holds that for all MC $\mathcal{M}$, $\mathcal{M} \models \mathcal{I}^P$ iff $\mathcal{M} \models \beta^*((S, s_0, \varphi_P, A, V, P, \lambda))$.*

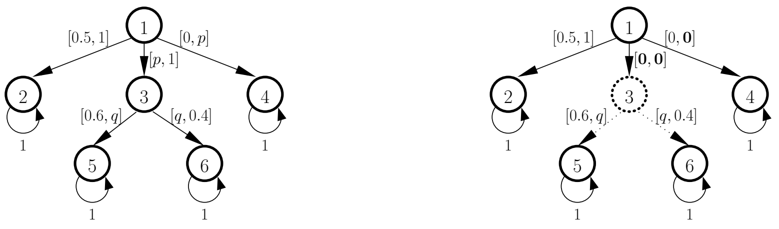


Figure 6 Iterative application of the pruning operator to pIMC $\mathcal{I}^P$ (left) until convergence (right).

Proof. Let $\mathcal{I}^P = (S, s_0, \varphi_P, A, V, P)$ be a local pIMC and let λ be a marking function such that $\lambda(s) = 0$ for all $s \in S$. Let $\mathcal{I}^{P'} = (S, s_0, \varphi'_P, A, V, P, \lambda') = \beta^n((S, s_0, \varphi_P, A, V, P, \lambda))$ for some $n \in \mathbb{N}$. We show that for all MC $\mathcal{M}$, we have $\mathcal{M} \models \mathcal{I}^{P'} \iff \mathcal{M} \models \beta(\mathcal{I}^{P'})$.

If $\{s \in S \mid \lambda'(s) = 0\}$ does not contain any locally inconsistent state or if s_0 is locally inconsistent, then the result is trivial. We thus assume that a locally inconsistent state $s_i \in \{s \in S \mid \lambda'(s) = 0\}$ is found and made unreachable by β.

We start by observing that, by construction, all states $s \in S$ such that $\lambda'(s) = 1$ are such that for all $s' \in S$ with $\lambda(s') = 0$, we have either $\varphi'_P(s', s) = [0, 0]$ or $\varphi'_P(s', s) = \emptyset$.

$\boxed{\Rightarrow}$ Let $\mathcal{I} = (S, s_0, \varphi, A, V, P)$ be an IMC and let $\psi : P \to [0, 1]$ be a valuation for the parameters such that $\mathcal{I} \vdash_\psi \mathcal{I}^{P'}$. Let $\mathcal{M} = (T, t_0, M, A, V^M)$ be a MC such that $M \models \mathcal{I}$ with satisfaction relation $\mathcal{R} \subseteq T \times S$. We show that there exists an IMC $\mathcal{I}'$ such that $M \models \mathcal{I}'$ and $\mathcal{I}' \vdash \beta(\mathcal{I}^{P'})$.

The proof proceeds in two steps: we first build the IMC $\mathcal{I}'$ and show that $\mathcal{I}' \vdash \beta(\mathcal{I}^{P'})$ and then show that $\mathcal{M} \models \mathcal{I}'$.

Let $\psi' : P \to [0, 1]$ be a new valuation for the parameters such that $\psi'(p) = 0$ if $p \in \texttt{enforce}(s_i)$ and $\psi'(p) = \psi(p)$ otherwise. Let $\mathcal{I}' = (S, s_0, \varphi', A, V, P)$ be such that $\varphi'(s, s') = \varphi_P(s, s')[p \leftarrow \psi'(p)]$ if $s' \neq s_i$ or if $\lambda(s) = 1$, $\varphi'(s, s_i) = [0, 0]$ if $\lambda(s) = 0$ and $\varphi_P(s, s_i)[\texttt{enforce}(s_i) \leftarrow 0] = [0, .]$, and $\varphi'(s, s_i) = \emptyset$ otherwise. By construction, it follows that $\mathcal{I}' \vdash_{\psi'} \beta(\mathcal{I}^{P'})$.

We now show that $\mathcal{R}$ is a satisfaction relation between $\mathcal{M}$ and $\mathcal{I}'$. Let $(t, s) \in \mathcal{R}$.

1. Since β has no effect on valuations, we have $V^M(t) = V(s)$.
2. Let δ be the function given in item 2 of Definition 3. By construction, it holds that
   a. for all $t' \in T$ such that $M(t, t') > 0$, $\delta(t')$ is a distribution on S,
   b. for all $s' \in S$, we have $(\sum_{t' \in T} M(t, t') \cdot \delta(t')(s')) \in \varphi(s, s')$, and
   c. for all $t' \in T$ and $s' \in S$, if $\delta(t')(s') > 0$, then $(t', s') \in \mathcal{R}$.

Items 2.*a*. and 2.*c*. are not impacted by β. We now show that item 2.*b*. still holds when considering φ' instead of φ.

Remark that since s_i is locally inconsistent in $\mathcal{I}'$, we necessarily have that for all $t' \in T$, $(t', s_i) \notin \mathcal{R}$ and therefore $\delta(t')(s_i) = 0$. Let $s' \in S$ and consider $\varphi'(s, s')$.

- If $s' \neq s_i$, we have $\varphi'(s, s') = \varphi_P(s, s')[p \leftarrow \psi'(p)]$. If $\varphi_P(s, s') = [x, y]$ with $x \in [0, 1] \cup P$ and $y \in [0, 1] \cup (P \setminus \texttt{enforce}(s_i))$, then either $\varphi'(s, s') = \varphi(s, s')$ or $\varphi(s, s') \subseteq \varphi'(s, s')$ and therefore $(\sum_{t' \in T} M(t, t') \cdot \delta(t')(s')) \in \varphi'(s, s')$. The only difficulty appears when $y = p \in \texttt{enforce}(s_i)$. In this case, there must exist $s'' \in S$ such that $\varphi_P(s'', s_i) = [p, .]$. Moreover, since $\mathcal{I}^P$ is *local*, we must have $s = s''$. By $\mathcal{R}$, we know that $(\sum_{t' \in T} M(t, t') \cdot$

$\delta(t')(s_i)) \in \varphi(s, s_i)$. Since $\delta(t')(s_i) = 0$ for all $t' \in T$, we have that $0 \in \varphi(s, s_i) = [\psi(p), .]$, thus $\psi(p) = \psi'(p) = 0$ and $\varphi'(s, s') = \varphi(s, s')$. As a consequence, item 2.*b*. still holds.

- If $s' = s_i$, then since s_i is inconsistent, we have $\sum_{t' \in T} M(t, t') \cdot \delta(t')(s_i) = 0 \in \varphi(s, s_i)$. As a consequence, we necessarily have $\varphi(s, s_i) = [0, .]$ and thus $\varphi_P(s, s_i)[\mathtt{enforce}(s_i) \leftarrow 0] = [0, .]$. Therefore, by construction, we still have $0 \in \varphi'(s, s_i)$ and item 2.*b*. holds.

Finally, $\mathcal{R}$ is still a satisfaction relation between $\mathcal{M}$ and $\mathcal{I}'$ and therefore $\mathcal{M} \models \mathcal{I}'$.

$\boxed{\Leftarrow}$ The proof of $\Leftarrow$ is straightforward with symmetric arguments. ◀

Consistency of global pIMCs. Unfortunately, the pruning algorithm we propose above cannot be ported to the setting of global pIMCs. Indeed, as illustrated in Example 14, the notion of local state consistency does not make sense in this setting, as restrictions on the values of parameters given by the local consistency of a given state can impact the local consistency of another. Nevertheless, consistency of global pIMCs is decidable: one can derive another, more complex, pruning algorithm from the one proposed in Definition 15. Since this algorithm is not optimal and only serves to prove decidability, we only present the outline of the algorithm without going into too much details.

Since fixing the value of given parameters may impact several states, we propose to group states that share given parameters and check inter-consistency of this group of states instead of local consistency of all states taken separately. We thus define groups of states that share parameters and propose a system of inequations that will decide whether this group of states is *inter-consistent*. Formally, given global pIMC $\mathcal{I}^P = (S, s_0, \varphi_P, A, V, P)$ and states $s_1, s_2 \in S$, we say that s_1 and s_2 are inter-dependent, written $s_1 \leftrightarrow s_2$ iff either $\mathtt{param}(s_1) \cap \mathtt{param}(s_2) \neq \emptyset$ or there exists s_3 such that $s_1 \leftrightarrow s_3$ and $s_3 \leftrightarrow s_2$. The groups of states we consider for the new notion of *inter-consistency* will thus be equivalence classes under $\leftrightarrow$.

Given such an equivalence class $\overline{s}$, we say that $\overline{s}$ is inter-consistent iff the system of inequations consisting of all inequations for local consistency of all states in $\overline{s}$ admits a solution. When $\overline{s}$ is **not** inter-consistent, the pruning algorithm will nondeterministically choose one of the states in $\overline{s}$, try to make it unreachable as in Definition 15 and mark it. From this point, if pruning goes on until $\mathcal{I}^P$ is proven consistent, then we can conclude positively. However, if the initial state is ultimately proven inconsistent, then we cannot conclude and the algorithm will backtrack and try making another state from $\overline{s}$ unreachable instead until all possible combinations of states in $\overline{s}$ have been considered. Only then can we conclude that $\mathcal{I}^P$ is inconsistent. Since there are only finitely many combinations of states in S, the algorithm will ultimately converge and allow deciding whether global pIMC $\mathcal{I}^P$ is consistent.

5 Concluding remarks

In this paper, we have introduced the new formalism of parametric Interval Markov Chains, that extends Interval Markov Chains by allowing the use of parameters as lower or upper bounds to the interval probabilities of transitions. We have also shown that the consistency problem is decidable for pIMCs and proposed an efficient algorithm for checking consistency of pIMCs with local parameters only. While we limit ourselves to intervals where parameters can only appear as lower or upper bound, our work can be directly extended to intervals with linear expressions over parameters and constants. In fact, this change does not impact any of the proposed solutions for local or global pIMCs : the systems of inequations we propose for deciding local or inter-consistency and the subsequent pruning algorithms remain unchanged.

The first direction for future work is to design better-suited algorithms for solving the consistency problem in the case of global pIMCs. Our second direction for future work is to consider other problems of interest for pIMCs, e.g. parameter synthesis with respect to some optimality criterion such as reachability. Finally, as has been argued in the literature, IMCs are quite limited as a specification theory as they are not closed under compositional operators such as parallel composition or conjunction. Therefore, we plan to extend our reasoning to more expressive specification theories such as Constraint Markov Chains [6] or Abstract Probabilistic Automata [9].

References

1 É. André, L. Fribourg, and J. Sproston. An extension of the inverse method to probabilistic timed automata. *Formal Methods in System Design*, (2):119–145, 2013.

2 R. Barbuti, F. Levi, P. Milazzo, and G. Scatena. Probabilistic model checking of biological systems with uncertain kinetic rates. *Theor. Comput. Sci.*, 419(0):2 – 16, 2012.

3 N. Bertrand and P. Fournier. Parameterized verification of many identical probabilistic timed processes. In *FSTTCS*, volume 24 of *LIPIcs*, pages 501–513, 2013.

4 N. Bertrand, P. Fournier, and A. Sangnier. Playing with probabilities in reconfigurable broadcast networks. In *FoSSaCS*, volume 8412 of *LNCS*, pages 134–148. Springer, 2014.

5 F. Biondi, A. Legay, B.F. Nielsen, and A. Wasowski. Maximizing entropy over markov processes. In *LATA*, volume 7810 of *LNCS*, pages 128–140. Springer, 2013.

6 B. Caillaud, B. Delahaye, K.G. Larsen, A. Legay, M.L. Pedersen, and A. Wasowski. Constraint markov chains. *Theor. Comput. Sci.*, 412(34):4373–4404, 2011.

7 N. Chamseddine, M. Duflot, L. Fribourg, C. Picaronny, and J. Sproston. Computing expected absorption times for parametric determinate probabilistic timed automata. In *QEST*, pages 254–263. IEEE Computer Society, 2008.

8 C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *ICTAC*, volume 3407 of *LNCS*, pages 280–294. Springer, 2004.

9 B. Delahaye, J-P. Katoen, K.G. Larsen, A. Legay, M.L. Pedersen, F. Sher, and A. Wasowski. Abstract probabilistic automata. *Inf. Comput.*, 232:66–116, 2013.

10 B. Delahaye, K.G. Larsen, A. Legay, M.L. Pedersen, and A. Wasowski. Consistency and refinement for interval markov chains. *J. Log. Algebr. Program.*, 81(3):209–226, 2012.

11 L.M. Ferrer Fioriti, E.M. Hahn, H. Hermanns, and B. Wachter. Variable probabilistic abstraction refinement. In *ATVA*, volume 7561 of *LNCS*, pages 300–316. Springer, 2012.

12 R. Gori and F. Levi. An analysis for proving probabilistic termination of biological systems. *Theor. Comput. Sci.*, 471(0):27 – 73, 2013.

13 E.M. Hahn, T. Han, and L. Zhang. Synthesis for PCTL in parametric Markov decision processes. In *NSFM*, volume 6617 of *LNCS*, pages 146–161. Springer, 2011.

14 E.M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. PARAM: A model checker for parametric Markov models. In *CAV*, volume 6174 of *LNCS*, pages 660–664. Springer, 2010.

15 E.M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric Markov models. *Software Tools for Technology Transfer*, 13(1):3–19, 2011.

16 B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *LICS*, pages 266–277. IEEE Computer, 1991.

17 R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Decidability results for parametric probabilistic transition systems with an application to security. In *SEFM*, pages 114–121. IEEE Computer Society, 2004.

18 R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Aspects of Computing*, 19(1):93–109, 2007.

19 K. Sen, M. Viswanathan, and G. Agha. Model-checking markov chains in the presence of uncertainties. In *TACAS*, volume 3920 of *LNCS*, pages 394–410. Springer, 2006.

Guaranteed control of switched control systems using model order reduction and state-space bisection

Adrien Le Coënt[1], Florian de Vuyst[1], Christian Rey[2,3], Ludovic Chamoin[2], and Laurent Fribourg[4]

1 **CMLA, ENS Cachan & CNRS**
61 Av. du Président Wilson, 94235 Cachan Cedex, France
`adrien.le-coent,devuyst@cmla.ens-cachan.fr`

2 **LMT Cachan, ENS Cachan & CNRS**
61 Av. du Président Wilson, 94235 Cachan Cedex, France
`rey,chamoin@lmt.ens-cachan.fr`

3 **Safran Tech**
1 Rue Geneviève Aubé, 78772 Magny les Hameaux, France
`christian.rey@safran.fr`

4 **LSV, ENS Cachan & CNRS**
61 Av. du Président Wilson, 94235 Cachan Cedex, France
`fribourg@lsv.ens-cachan.fr`

Abstract

This paper considers discrete-time linear systems controlled by a *quantized* law, i.e., a piecewise constant time function taking a finite set of values. We show how to generate the control by, first, applying *model reduction* to the original system, then using a “state-space bisection” method for synthesizing a control at the reduced-order level, and finally computing an upper bound to the deviations between the controlled output trajectories of the reduced-order model and those of the original model. The effectiveness of our approach is illustrated on several examples of the literature.

1998 ACM Subject Classification B.1.2 Control Structure Performance Analysis and Design Aids, B.1.3 Control Structure Reliability, Testing, and Fault-Tolerance, B.2.3 Reliability, Testing, and Fault-Tolerance

Keywords and phrases Model Order Reduction, guaranteed control, stability control, reachability control, error bounding

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.33

1 Introduction

We are focusing here on switched control systems, a class of hybrid systems recently used with success in various domains such as automotive industry and power electronics. Several strategies have been developped to design control laws for such systems; we use here the invariance analysis [9, 8, 10]. The associated algorithms are very expensive and require a limited state space dimension; we thus use a model order reduction in order to synthesize a controller at the reduced-order level. Two methods are proposed to apply the controller to the full-order system. Offline and online controls are enabled, and the computation of upper bounds of the error induced by the reduction allowed to guarantee the effectiveness of the controller.

Comparison with related work. Model order reduction techniques for hybrid or switched systems are classically used in *numerical simulation* in order to construct at the reduced level trajectories which cannot be computed directly at the original level due to complexity and large size dimension [3].

Han and Krogh make use of model reduction in order to perform *set-based reachability analysis* [16]. They do not construct isolated trajectories issued from isolated points, but (an overapproximation of) the infinite set of trajectories issued from a dense set of initial points. This allows them to perform formal verification of properties such as *safety*. In both approaches, the control is *given* as an input of the problem. In contrast here, the control is *synthesized* using set-based methods in order to achieve by construction properties such as *convergence* and *stability*.

The problem of control synthesis for hybrid and switched systems has been widely studied and various tools exist. The Multi-Parametric Toolbox (MPT 3.0 [17]) for example solves optimal control problems using operations on polytopes. Most approaches make use of Lyapunov or the so-called "multiple Lyapunov functions" to solve the problem of control synthesis for switched systems - see for example [24]. The approximate bisimulation approach abstracts switched systems under the form of a discrete model [14, 12] under certain Lyapunov-based stability conditions. The latter approach has been implemented in PESSOA [18] and CoSyMA [20]. The approach used in this paper avoids using Lyapunov functions and relies on the notion of "(controlled) invariant" [7].

Plan. In Section 2, we give some preliminaries about linear controlled systems and reachability sets. In Section 3, we recall the principles of the state-space bisection method. In Section 4, we explain how to construct a reduced model, apply the state-space bisection method at this level, and compute upper bounds to the error induced at the original level. In Section 5, we propose two methods of control synthesis allowing to synthesize (either offline or online) a controller at the reduced-order level and apply it to the full-order system. In Section 6, we apply our approach to several examples of the literature. We conclude in Section 7.

2 Background

We consider a class of control systems composed of a plant and a controller defined as follows. The plant is a discrete-time linear time invariant (DLTI) system Σ defined by (see [4, 24] for more information on DLTI and sampled switched systems):

$$\Sigma : \begin{cases} x(t+\tau) & = A_d x(t) + B_d u(t), \\ y(t) & = C_d x(t). \end{cases} \tag{1}$$

Here, the state x is an n-vector, the control input u a p-vector, the output y an m-vector, and A_d an $n \times n$-matrix, B_d an $n \times p$-matrix, C_d a $m \times n$ matrix. The real positive constant τ is the time sampling parameter. All the coefficients are reals. The system Σ is obtained from the temporal discretization of the continuous linear time invariant (LTI) system:

$$\begin{cases} \dot{x}(t) & = Ax(t) + Bu(t), \\ y(t) & = Cx(t), \end{cases}$$

with

$$A_d = e^{A\tau}, \quad B_d = \int_0^\tau e^{A(\tau - t)} B dt, \quad C_d = C.$$

We consider here the case of a *quantized* control (see, e.g. [22]). This means that the control law $\mathbf{u} \equiv u(\cdot)$ is a *piecewise constant* function that changes its value at each sampling time $0, \tau, 2\tau, \ldots$ Furthermore, $\mathbf{u}$ can take only a *finite* number of vector values. We denote by U the finite set of values taken by $\mathbf{u}$, every element of U is called a *mode*. The problem of (discretized) quantized control is to find a *state-dependent* law $u(t)$ which, at every sampling time, finds the element of U that allows to achieve a given goal, such as the stabilization around an objective point y_{obj}. Actually, the stabilization of such systems cannot be perfect [22], and we thus look for *practical* stability: we do not look for an equilibrium point y_{obj}, but only for a neighborhood of y_{obj} in which we confine the system. We note that the controller in the above model implements a state feedback law and has only discrete-state dynamics.

The entries of the problem are the following:

1. a subset $R_x \subset \mathbb{R}^n$ of the state space, called *interest set*,
2. a subset $R_y \subset \mathbb{R}^m$ of the output space, called *objective set*.

The objective is to find a law $u(\cdot)$ which, for any initial state $x_0 \in R_x$, stabilizes the output y in the set R_y.

Remark: In [10], we have proposed a procedure, called *state-space bisection procedure*[1], in order to achieve a similar goal. The context was simpler since we considered there only the state equation $x(t+\tau) = A_d x(t) + B_d u(t)$ without the output equation $y(t) = C_d x(t)$. Here, the stabilization problem is naturally extended to take into account the output $y(t)$. Note that even if the initial state x_0 belongs to R_x, the corresponding output y_0 does not necessarily belong to R_y.

The state-space bisection procedure being subject to the so-called *curse of dimensionality*, it will be applied here to a reduced order model $\hat{\Sigma}$ of dimension $n_r < n$ in order to stabilize its output in the objective set R_y. We will show that the control law synthesized on the reduced system $\hat{\Sigma}$ still stabilizes the output of the original system Σ with a tolerance $\varepsilon > 0$. We now introduce some notations required to explain the state-space bisection procedure.

Some notations. We will use $\mathbf{x}(t, x, u)$ to denote the point reached by Σ at time t under mode $u \in U$ from the initial condition x. This gives a transition relation $\rightarrow^\tau_u$ defined for x and x' in $\mathbb{R}^n$ by: $x \rightarrow^\tau_u x'$ iff $\mathbf{x}(\tau, x, u) = x'$. Given a set $X \subset \mathbb{R}^n$, we define the *successor set* of a set $X \subset \mathbb{R}^n$ of states under mode u as:

$$Post_u(X) = \{x' \mid x \rightarrow^\tau_u x' \text{ for some } x \in X\}.$$

The set $Post_u(X)$ is then the result of the affine transformation $A_d X + B_d u$. Likewise, we define the *output successor set* of a set $X \subset \mathbb{R}^n$ of states under mode u as:

$$Post_{u,C}(X) = \{Cx' \mid x \rightarrow^\tau_u x' \text{ for some } x \in X\}.$$

An *input pattern* named Pat is defined as a finite sequence of modes. A *k-pattern* is an input pattern of length at most k. The successor set of $X \subset \mathbb{R}^n$ using $Pat \equiv (u_1 \cdots u_k)$ is defined by

$$Post_{Pat}(X) = \{x' \mid x \rightarrow^\tau_{u_1} \cdots \rightarrow^\tau_{u_k} x',\ x \in X\}.$$

[1] In [9, 8, 10], this method was called “state-space decomposition”, but we use here “state-space bisection” in order to avoid ambiguity with model reduction methods.

The mapping $Post_{Pat}$ is itself an affine transformation. The output successor set of $X \subset \mathbb{R}^n$ using $Pat \equiv (u_1 \cdots u_k)$ is defined by

$$Post_{Pat,C}(X) = \{Cx' \mid x \rightarrow^{\tau}_{u_1} \cdots \rightarrow^{\tau}_{u_k} x',\ x \in X\}.$$

Given an input pattern Pat of the form $(u_1 \cdots u_m)$, and a set $X \subset \mathbb{R}^n$, the *unfolding of* X *via* Pat, denoted by $Unf_{Pat}(X)$, is the set $\bigcup_{i=0}^{m} X_i$ with:

- $X_0 = X$,
- $X_{i+1} = Post_{u_{i+1}}(X_i)$, for all $0 \leq i \leq m-1$.

The unfolding thus corresponds to the set of all the intermediate states produced when applying the input pattern Pat to the states of X.

3 Control Synthesis by State-Space Bisection

We now explain the method that we are going to use in order to find a control law $\mathbf{u}$ that stabilizes the state x of the system Σ into a given zone $R_x \subset \mathbb{R}^n$ [10], and makes the output y reach a given zone $R_y \subset \mathbb{R}^n$.

3.1 x-stabilization and y-convergence requirements

Given an interest set R_x and an objective set R_y, we can define the notion of "x-stabilization" and "y-convergence" in this context as follows.

▶ **Definition 1.** Given a system Σ, a set R_x subset of $\mathbb{R}^n$, a set R_y subset of $\mathbb{R}^m$, and a positive integer k, an *x-stabilizing and y-convergent control* for (R_x, R_y, k) with respect to Σ is a function that associates to each $x \in R_x$ a k-pattern Pat such that:

- $Post_{Pat}(\{x\}) \subseteq R_x$,
- $Post_{Pat,C}(\{x\}) \subseteq R_y$.

Note that we use the term "y-convergence" because the output corresponding to the initial state can possibly be outside R_y, whereas the initial state necessarily belongs to R_x. Given a system Σ, an x-stabilizing and y-convergent control guarantees that all the trajectories starting at R_x return to R_x within k steps and that the output is sent into R_y. In order to find an x-stabilizing and y-convergent control, we can adapt the method of "state-space bisection" introduced in [8].

▶ **Definition 2.** Given a system Σ, two sets R_x and R_y respectively subsets of $\mathbb{R}^n$ and $\mathbb{R}^m$, and a positive integer k, a *successful decomposition of* (R_x, R_y, k) *w.r.t.* Σ is a set Δ of the form $\{V_i, Pat_i\}_{i \in I}$, where I is a finite set of indices, every V_i is a subset of R_x, and every Pat_i is a k-pattern such that:

(a) $\bigcup_{i \in I} V_i = R_x$,
(b) for all $i \in I$: $Post_{Pat_i}(V_i) \subseteq R_x$,
(c) for all $i \in I$: $Post_{Pat_i,C}(V_i) \subseteq R_y$.

Remark: Note that in pratice, the condition $Post_{Pat_i,C}(V_i) \subseteq R_y$ is verified by verifying that the bounding box of $Post_{Pat_i,C}(V_i)$ (that is the smallest square box containing $Post_{Pat_i,C}(V_i)$) belongs to R_y. All the set-based operations are carried out with zonotopes. See [1] for the computation of the bounding box of a zonotope and operations realized on zonotopes.

A successful decomposition $\Delta = \{(V_i, Pat_i)\}_{i \in I}$ naturally induces a *state-dependent control* on R_x. The control induced by Δ is defined as follows: consider an initial point x_0

of R_x; we know that $x_0 \in V_i$ for some $i \in I$ (since $R_x = \bigcup_{i \in I} V_i$). One thus applies Pat_i to x_0, which gives a new state x_1 that belongs itself to R_x, and the associated output belongs to R_y (since $Post_{Pat_i}(V_i) \subseteq R_x$ and $Post_{Pat_i,C}(V_i) \subseteq R_y$). The process can then be repeated on x_1, and so on iteratively. Obviously, the induced control is an x-stabilizing and y-convergent control for (R_x, R_y, k). Formally, we have:

▶ Proposition 1. Suppose that Δ is a successful decomposition of (R_x, R_y, k) w.r.t. Σ. Then the control induced by Δ is an x-stabilizing and y-convergent control for (R_x, R_y, k) w.r.t. Σ.

The problem of finding an x-stabilizing and y-convergent controller thus reduces to the problem of finding a successful decomposition. The latter problem can be solved by using the method of *state-space bisection* [8], as explained below.

3.2 Bisection method

We give here a simple algorithm, adapted from [8], called Bisection algorithm. Given two sets R_x and R_y, and a positive integer k, the algorithm, when it succeeds, provides for a successful decomposition Δ of (R_x, R_y, k) w.r.t. Σ of the form $\{V_i, Pat_i\}_{i \in I}$. The input sets R_x and R_y are given under the form of *boxes* of $\mathbb{R}^n$ and $\mathbb{R}^m$ (i.e., cartesian products of n closed intervals for R_x, and cartesian products of m closed intervals for R_y). The subsets V_i, $i \in I$, of R_x are boxes that are obtained by repeated bisection. At the beginning, the Bisection procedure calls sub-procedure Find_Pattern in order to get a k-pattern Pat such that $Post_{Pat}(R_x) \subseteq R_x$ and $Post_{Pat,C}(R_x) \subseteq R_y$. If it succeeds, then it is done. Otherwise, it divides R_x into 2^n sub-boxes $V_1, \dots, V_{2^n}$ of equal size. If for each V_i, Find_Pattern gets a k-pattern Pat_i such that $Post_{Pat_i}(V_i) \subseteq R_x$ and $Post_{Pat_i,C}(V_i) \subseteq R_y$, it is done. If, for some V_j, no such input pattern exists, the procedure is recursively applied to V_j. It ends with success when a successful decomposition of (R_x, R_y, k) is found, or failure when the maximal degree d of bisection is reached. The algorithmic form of the procedure is given in Algorithms 1 and 2. The main procedure Bisection(W, R_x, R_y, D, K) is called with R_x as input value for W, d for input value for D, and k as input value for K; it returns either $\langle \{(V_i, Pat_i)\}_i, True \rangle$ with $\bigcup_i V_i = W$, $\bigcup_i Post_{Pat_i}(V_i) \subseteq R_x$, $\bigcup_i Post_{Pat_i,C}(V_i) \subseteq R_y$ when it succeeds, or $\langle \_, False \rangle$ when it fails. Procedure Find_Pattern(W,R_x,R_y,K) looks for a K-pattern Pat for which $Post_{Pat}(W) \subseteq R_x$ and $Post_{Pat,C}(W) \subseteq R_y$: it selects all the K-patterns by increasing length order until either it finds such an input pattern Pat (output: $\langle Pat, True \rangle$), or none exists (output: $\langle \_, False \rangle$). The correctness of the procedure is stated as follows.

▶ **Theorem 3.** *If Bisection(R_x,R_x,R_y,d,k) returns $\langle \Delta, True \rangle$, then Δ is a successful decomposition of (R_x, R_y, k) w.r.t. Σ.*

In [8], we have developed a tool that implements the Bisection procedure, using zonotopes (see [13]); it is written in Octave [21].

4 Model Order Reduction

Actually, because of the computational cost of the bisection procedure, the application of the bisection method at the full-order level n becomes rapidly intractable (typically for $n \geq 15$). Therefore, it is interesting to apply *projection-based* model order reduction methods (see [3]), then construct decompositions (hence control laws) at the reduced state level of dimension $n_r < n$ rather than at the full-order state level of dimension n. For many

Algorithm 1: Bisection(W, R_x, R_y, D, K).
/* with additional input ε_x for online version */

```
Input: A box $W$, a box $R_x$, a box $R_y$, a degree $D$ of bisection, a length $K$ of input pattern
Output: $\langle\{(V_i, Pat_i)\}_i, True\rangle$ with $\bigcup_i V_i = W$, $\bigcup_i Post_{Pat_i}(V_i) \subseteq R_x$ and $\bigcup_i Post_{Pat_i,C}(V_i) \subseteq R_y$, or $\langle\_, False\rangle$
1  $(Pat, b) := Find\_Pattern(W, R_x, R_y, K)$
2  if $b = True$ then
3      return $\langle\{(W, Pat)\}, True\rangle$
4  else
5      if $D = 0$ then
6          return $\langle\_, False\rangle$
7      else
8          Divide equally $W$ into $(W_1, \ldots, W_{2^n})$
9          for $i = 1 \ldots 2^n$ do
10             $(\Delta_i, b_i) :=$ Bisection$(W_i, R_x, R_y, D-1, K)$
11         return $(\bigcup_{i=1\ldots 2^n} \Delta_i, \bigwedge_{i=1\ldots 2^n} b_i)$
```

applications and engineering problems, it is observed that the systems can be reduced while being accurate enough. This is due to the underlying regularity of the solutions or, in other words, to the low dimension of the actual trajectory submanifold.

Given a full-order system Σ, an interest set $R_x \subset \mathbb{R}^n$ and an objective set $R_y \subset \mathbb{R}^m$, we will construct a reduced-order system $\hat{\Sigma}$ using a projection π of $\mathbb{R}^n$ to $\mathbb{R}^{n_r}$. If $\pi \in \mathbb{R}^{n\times n}$ is a projection, it verifies $\pi^2 = \pi$, and π can be written as $\pi = \pi_L \pi_R$, where $\pi_L \in \mathbb{R}^{n\times n_r}$, $\pi_R \in \mathbb{R}^{n_r\times n}$ and $n_r = rank(\pi)$. The DLTI system $\hat{\Sigma}$ is defined by the matrices $\hat{A}_d$, $\hat{B}_d$, $\hat{C}_d$, and can be written:

$$\hat{\Sigma} : \begin{cases} \hat{x}(t+\tau) & = \hat{A}_d \hat{x}(t) + \hat{B}_d u(t), \\ y_r(t) & = \hat{C}_d \hat{x}(t), \end{cases}$$

with

$$\hat{A}_d = e^{\hat{A}\tau}, \quad \hat{B}_d = \int_0^\tau e^{\hat{A}(\tau - t)} \hat{B} dt, \quad \hat{C}_d = \hat{C}.$$

The matrices $\hat{A}$, $\hat{B}$ and $\hat{C}$ are defined as follows:

$$\hat{A} = \pi_R A \pi_L, \quad \hat{B} = \pi_R B, \quad \hat{C} = C \pi_L.$$

Here, $\hat{x}$ is an n_r-vector, u a p-vector, y_r an m-vector, and $\hat{A}_d$ an $n_r \times n_r$-matrix, $\hat{B}_d$ an $n_r \times p$-matrix, $\hat{C}_d$ a $m \times n_r$ matrix. The reduced system is obtained with the change of variable: $\hat{x} = \pi_R x$. Accordingly, $\hat{R}_x = \pi_R R_x \subset \mathbb{R}^{n_r}$ is the projection of R_x. The objective set R_y is kept unchanged. Using the method described in Section 3, one generates a successful decomposition $\hat{\Delta}$ of $(\hat{R}_x, R_y, k)$ w.r.t. $\hat{\Sigma}$ for some given k. This leads to a reduced-order control $\mathbf{u}_{\hat{\Delta}}$. By Theorem 3, $\mathbf{u}_{\hat{\Delta}}$ sends the output y_r in R_y. When this control is applied to the full-order system Σ, this leads to a trajectory $y(t)$. The difference between the two trajectories $y(t)$ and $y_r(t)$ will be denoted by $e(t)$. An upper bound of $\|e(t)\|$ will be computed in order to assess the deviation from R_y. We will denote by ε_y^j an upper bound of this error for $t = j\tau$, and we will denote by ε_y^∞ the maximum value of this bound: $\varepsilon_y^\infty = \sup_{j\geq 0} \varepsilon_y(j\tau)$ (see Appendix for the calculation of these bounds).

```
Algorithm 2: Find_Pattern($W, R_x, R_y, K$).
                              /* with additional input $\varepsilon_x$ for online version */
Input: A box $W$, a box $R_x$, a box $R_y$, a length $K$ of input pattern
Output: $\langle Pat, True\rangle$ with ,$Post_{Pat}(W) \subseteq R_x$,$Post_{Pat,C}(W) \subseteq R_y$ and
        $Unf_{Pat}(W) \subseteq S$, or $\langle \_, False\rangle$ when no input pattern maps $W$ into $R_x$ and
        $CW$ into $R_y$
1 for $i = 1 \dots K$ do
2     $\Pi$ := set of input patterns of length $i$
3     while $\Pi$ is non empty do
4         Select $Pat$ in $\Pi$
5         $\Pi := \Pi \setminus \{Pat\}$
6         if $Post_{Pat}(W) \subseteq R_x$ and $Post_{Pat,C}(W) \subseteq R_y$ /* condition modified for online
          version */ then
7             return $\langle Pat, True\rangle$
8 return $\langle \_, False\rangle$
```

5 Reduced Order Control

In this section, we first explain the procedure of control synthesis, then we propose a method to guarantee that the obtained controller sends the output of the full-order system in R_y with a tolerance ε_y^∞.

5.1 Guaranteed offline control

Suppose that we are given a system Σ, an interest set R_x, and an objective set R_y. The procedure first consists in reducing the system Σ of order n to a system $\hat{\Sigma}$ of order $n_r < n$ (see section 4). Here, the classical method of balanced truncation [5, 2, 19, 6] is used to construct π.

We apply the state-space bisection procedure to the reduced-order system $\hat{\Sigma}$, we obtain a successful decomposition $\hat{\Delta}$ of $(\hat{R}_x, R_y, k)$ w.r.t. $\hat{\Sigma}$. Therefore, the procedure returns a successful decomposition $\hat{\Delta}$ of the form $\{\hat{V}_i, Pat_i\}_{i\in I}$ such that:

1. I is a finite set of indices,
2. every $\hat{V}_i$ $(i \in I)$ is an interval product of dimension n_r such that $\bigcup_{i\in I} \hat{V}_i = \hat{R}_x$,
3. every Pat_i $(i \in I)$ is a k-pattern such that for all $i \in I$: $Post_{Pat_i}(\hat{V}_i) \subseteq \hat{R}_x$ and $Post_{Pat_i,\hat{C}}(\hat{V}_i) \subseteq R_y$.

The successful decomposition $\hat{\Delta}$ induces a control $u_{\hat{\Delta}}$ on $\hat{R}_x$. This control $u_{\hat{\Delta}}$ is $\hat{x}$-stabilizing and y_r-convergent for $(\hat{R}_x, R_y, k)$ w.r.t. $\hat{\Sigma}$. Let x_0 be an initial condition in R_x. Let $\hat{x}_0 = \pi_R x_0$ be its projection belonging to $\hat{R}_x$, $\hat{x}_0 = \pi_R x_0$ is the initial condition for the reduced system $\hat{\Sigma}$: $\hat{x}_0$ belongs to $\hat{V}_{i_0}$ for some $i_0 \in I$; thus, after applying Pat_{i_0}, the system is led to a state $\hat{x}_1$; $\hat{x}_1$ belongs to $\hat{V}_{i_1}$ for some $i_1 \in I$; and iteratively, we build, from an initial state $\hat{x}_0$, a sequence of states $\hat{x}_1, \hat{x}_2, \dots$ obtained by application of the sequence of k-patterns Pat_{i_0}, Pat_{i_1}, ... (steps (1), (2) and (3) of Figure 1).

The sequence of k-patterns is computed for the reduced system $\hat{\Sigma}$, but it can be applied to the full-order system Σ: we build, from an initial point x_0, a sequence of points x_1, x_2,... by application of the k-patterns Pat_{i_0},Pat_{i_1},... (steps (4), (5) and (6) of Figure 1). For

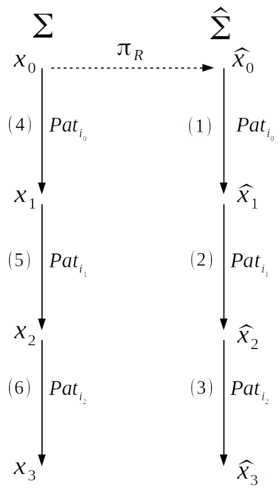


Figure 1 Diagram of the offline procedure for a simulation of length 3.

all $x_0 \in R_x$ and for all $t \geq 0$, the error $\|y(x_0,u,t) - y_r(\pi_R x_0,u,t)\|$ is bounded by ε_y^∞, as defined in Appendix). This leads to the following proposition:

▶ Proposition 2. Let us consider a DLTI system Σ, an interest set R_x, and an objective set R_y. Let $\hat{\Sigma}$ be the projection by balanced truncation of Σ. Let $\hat{\Delta}$ be a successful decomposition of ($\hat{R}_x$,R_y,k) w.r.t. $\hat{\Sigma}$. Then, for all $x_0 \in R_x$, the induced control $u_{\hat{\Delta}}$ applied to the full-order system Σ in x_0 is such that for all $j > 0$, the output of the full-order system $y(t)$ returns to $R_y + \varepsilon_y^\infty$ after at most k τ-steps.

Here, $R_y + \varepsilon_y^\infty$ denotes the set containing R_y with a margin of ε_y^∞. More precisely, if R_y is an interval product of the form $[a_1,b_1] \times \cdots \times [a_m,b_m]$, then $R_y + \varepsilon_y^\infty$ is defined by $[a_1 - \varepsilon_y^\infty, b_1 + \varepsilon_y^\infty] \times \cdots \times [a_m - \varepsilon_y^\infty, b_m + \varepsilon_y^\infty]$.

Remark: Here, we ensure that $y(x_0,u,t)$ is in $R_y + \varepsilon_y^\infty$ at the end of every input pattern, but an easy improvement is to ensure that $y(x_0,u,t)$ stays in a given *safety set* $S_y \supset R_y$ at every step of time τ. Indeed, as explained in [8], we can ensure that the unfolding of the output trajectory stays in S_y. In order to guarantee that $y(x_0,u,t)$ stays in S_y, we just have to make sure that $y_r(\pi_R x_0,u,t)$ stays in the reduced safety set $S_y - \varepsilon_y^\infty$. We thus have to add the condition: "and $Unf_{Pat}(CW) \subset S_y - \varepsilon_y^\infty$" in the line 6 of Algorithm 2.

5.2 Guaranteed Online control

Up to this point, the procedure of control synthesis consists in computing a complete sequence of input patterns on the reduced order model $\hat{\Sigma}$ for a given initial state x_0, and applying the input pattern sequence to the full-order model Σ. The control law is thus computed offline. However, using the bisection method applied to the reduced system $\hat{\Sigma}$, we can use the decomposition $\hat{\Delta}$ online as follows: Let x_0 be the initial state in R_x and $\hat{x}_0 = \pi_R x_0$ its projection belonging to $\hat{R}_x$ (step (1) of Figure 2); $\hat{x}_0$ belongs to $\hat{V}_{i_0}$ for some $i_0 \in I$; we can thus apply the associated input pattern Pat_{i_0} to the full-order system Σ, which yields a state $x_1 = Post_{Pat_{i_0}}(x_0)$ (step (2) of Figure 2); the corresponding output is sent to $y_1 = Post_{Pat_{i_0},C}(x_0) \in R_y + \varepsilon_y^{\ell_0}$; in order to continue to step (3), we have to guarantee that $\pi_R Post_{Pat_i}(x)$ belongs to $\hat{R}_x$ for all $x \in R_x$ and for all $i \in I$. As explained below, this is possible using the computation of an upper bound to the error $\|\pi_R Post_{Pat_i}(x) - Post_{Pat_i}(\pi_R x)\|$

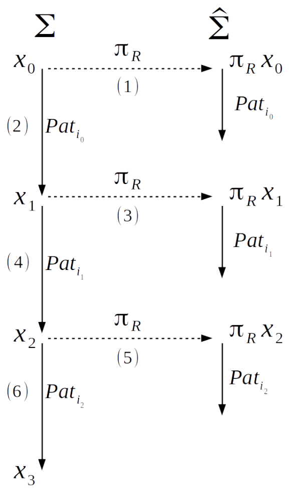


Figure 2 Diagram of the online procedure for a simulation of length 3.

and a reinforcement of the procedure for taking into account this error.

Let ε_x^j be an upper bound to $\|\pi_R Post_{Pat}(x) - Post_{Pat}(\pi_R x)\|$, j being the length of the input pattern Pat (see Appendix for the calculation of this bound). We modify the Algorithms 1 and 2 by adding a new input $\varepsilon_x = (\varepsilon_x^1, \dots, \varepsilon_x^k)$, k being the maximal length of the input patterns. With such an additional input, we define an ε-decomposition as follows:

▶ **Definition 4.** Given a system Σ, two sets R_x and R_y respectively subsets of $\mathbb{R}^n$ and $\mathbb{R}^m$, a positive integer k, and a vector of errors $\varepsilon_x = (\varepsilon_x^1, \dots, \varepsilon_x^k)$, an ε-decomposition of $(R_x, R_y, k, \varepsilon_x)$ w.r.t. Σ is a set Δ of the form $\{V_i, Pat_i\}_{i \in I}$, where I is a finite set of indices, every V_i is a subset of R_x, and every Pat_i is a k-pattern such that:
(a') $\bigcup_{i \in I} V_i = R_x$,
(b') for all $i \in I$: $Post_{Pat_i}(V_i) \subseteq R_x - \varepsilon_x^{|Pat_i|}$,
(c') for all $i \in I$: $Post_{Pat_i,C}(V_i) \subseteq R_y$.

Note that condition (b') is a strenghtening of condition (b) of definition 2. Accordingly, line 6 of Algorithm 2 is modified as follows:

```
6   if $Post_{Pat}(W) \subseteq R_x - \varepsilon_x^i$ and $Post_{Pat,C}(W) \subseteq R_y$ then
```

The computation of an ε-decomposition with the modified algorithms enables to guarantee that the projection $\pi_R x$ of the full-order system state always stays in $\hat{R}_x$. We can thus perform the online control as follows:

Since $Post_{Pat_{i_0}}(\hat{V}_{i_0}) \subseteq \hat{R}_x - \varepsilon_x^{\ell_0}$ and $\pi_R x_0 \in \hat{V}_{i_0}$, we have $Post_{Pat_{i_0}}(\pi_R x_0) \in \hat{R}_x - \varepsilon_x^{\ell_0}$; thus $\pi_R x_1 = \pi_R Post_{Pat_{i_0}}(x_0)$ belongs to $\hat{R}_x$, because $\varepsilon_x^{\ell_0}$ is a bound of the maximal distance between $Post_{Pat_{i_0}}(\pi_R x_0)$ and $\pi_R Post_{Pat_{i_0}}(x_0)$; since $\pi_R x_1$ belongs to $\hat{R}_x$, it belongs to V_{i_1} for some $i_1 \in I$; we can thus compute the input pattern Pat_{i_1}, and we can thus reapply the procedure and compute an input pattern sequence Pat_{i_0},Pat_{i_1},...

We finally have the proposition:

▶ Proposition 3. Let us consider a DLTI system Σ, an interest set R_x, and an objective set R_y. Let $\hat{\Sigma}$ be the projection by balanced truncation of Σ. Let $\hat{\Delta} = \{\hat{V}_i, Pat_i\}_{i \in I}$ be an ε-decomposition for $(\hat{R}_x$,R_y,k,$\varepsilon_x)$ w.r.t. $\hat{\Sigma}$, ε_x being defined as above. Then:

$$\forall x \in R_x, \exists i \in I: \quad \pi_R Post_{Pat_i}(x) \in \hat{R}_x \quad \wedge \quad Post_{Pat_i,C}(x) \in R_y + \varepsilon_y^{|Pat_i|}.$$

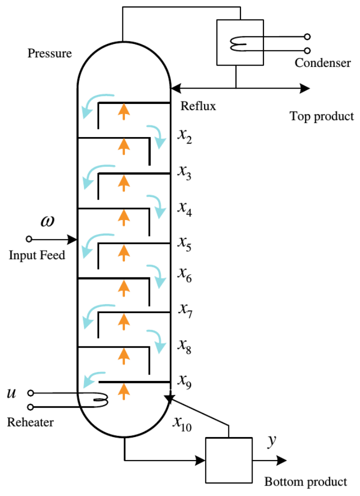


Figure 3 The 11-order distillation column system.

Using the decomposition $\hat{\Delta}$, we can perform an online control as explained above. We have:

▶ Proposition 4. Let us consider a DLTI system Σ, an interest set R_x, and an objective set R_y. Let $\hat{\Sigma}$ be the projection by balanced truncation of Σ. Let $\hat{\Delta}$ be a ε-decomposition for $(\hat{R}_x, R_y, k, \varepsilon_x)$ w.r.t. $\hat{\Sigma}$. Then, for all $x_0 \in R_x$, the induced online control $u_{\hat{\Delta}}$ yields an output sequence of points $y_1, y_2, \ldots$ which belong to $R_y + \varepsilon_y^{\ell_0}, R_y + \varepsilon_y^{\ell_1}, \ldots$ where ℓ_0, ℓ_1, $\ldots$ are the lengths of the input patterns successively applied.

The advantage of such an online control is that the estimated errors $\varepsilon_y^{\ell_0}, \varepsilon_y^{\ell_1}, \ldots$ are dynamically computed, and are smaller than the static bound ε_y^{∞} used in the offline control. The price to be paid is the strenghtening of the $\hat{x}$-stabilization condition (b') of definition 4.

6 Case Studies

6.1 Distillation Column

We consider a linearized model of a distillation column system [25] written under the form of a DLTI system (1). The state $x = (x_1, x_2, \ldots, x_{11})^\top$ of the system is of dimension 11: $x_1, x_2, \ldots, x_{10}$ correspond to the composition of the most volatile component in the different stages of the column, and x_{11} corresponds to the pressure at the top of the column. The perturbation of input feed ω is neglected. The control variable $u \in U = \{0, 1\}$ corresponds to the state turned on (1) or turned off (0) of the reheater. The output y is of dimension 1 and corresponds to the composition of the most volatile component in the bottom product. The system is reduced from $n = 11$ to $n_r = 2$. The sampling time is set to $\tau = 100$ s.

The matrices A,B, and C are the following:

$$A = 10^{-2} \times \begin{bmatrix} -1.4 & -0.43 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.95 & -1.38 & 0.46 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.05 \\ 0 & 0.95 & -1.41 & 0.63 & 0 & 0 & 0 & 0 & 0 & 0 & 0.02 \\ 0 & 0 & 0.95 & -1.58 & 1.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.95 & -3.12 & 1.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.02 & -3.52 & 2.2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.02 & -4.22 & 2.8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2.02 & -4.82 & 3.7 & 0 & 0.02 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.02 & -5.72 & 4.2 & 0.05 \\ 2.55 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.55 & -1.85 \end{bmatrix},$$

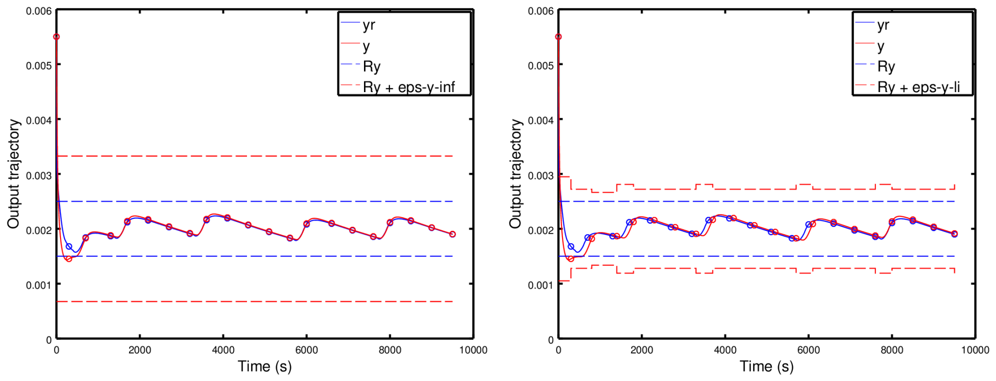


Figure 4 Simulation of $y(t) = Cx(t)$ and $y_r(t) = \hat{C}\hat{x}(t)$ from the initial condition $x_0 = (0.55)^{11}$. Left: guaranteed offline control; right: guaranteed online control.

$$B = [\,0\ 0\ 0\ 0\ 0.01\ 0\ 0\ 0\ 0\ 0\ 0\,]^T \quad \text{and} \quad C = [\,0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0.01\ 0\,]^T\,.$$

The interest set is $R_x = [0,1]^{11}$. The objective set is $R_y = [0.0015, 0.0025]$. The two methods presented in this paper give the results of Figure 4. The circles in the simulations correspond to the end of input patterns. The simulations have been performed with MINIMATOR (an Octave code available at https://bitbucket.org/alecoent/minimator_red) on a 2.80 GHz Intel Core i7-4810MQ CPU with 8 GB of memory. The offline and online methods led to the same decomposition because of the contractive behaviour of the system. The decompositions were obtained in 106 seconds.

Figure 4 shows simulations of the offline and online methods. The initial point $x_0 = (0.55)^{11}$ is in R_x. Note that the corresponding output $y_0 = Cx_0 = 5.5 \times 10^{-3}$ lies outside R_y. For the offline method (on the left), the output y_r of the reduced system (continuous blue) is sent into R_y (dashed blue), and the output y of the full-order system (continuous red) is sent in $R_y + \varepsilon_y^\infty$ (dashed red). For the online method (on the right), the output y_r of the reduced system (continuous blue) is sent into R_y (dashed blue), and the output y of the full-order system (continuous red) is sent in $R_y + \varepsilon_y^{\ell_i}$ (dashed red).

Both methods are thus efficient, the offline method is guaranteed but implies a relatively pessimistic tolerance. The online method is very efficient and the tolerances are much more satisfying. A shift can however appear between the full-order model and the reduced order model, this is an unavoidable consequence of the fact that the reduced order model does not represent the dynamic of the model as accurately as the full-order model.

6.2 Square Plate

We consider here the problem of controlling the central node temperature of a metal square plate discretized by finite elements. This example is taken from [15]. The square plate is subject to the heat equation: $\dfrac{\partial T}{\partial t}(x,t) - \alpha^2 \Delta T(x,t) = 0$. After discretization, the system is written under the form of a DLTI system (1). The plate is insulated along three edges, the right edge is open. The left half of the bottom edge is connected to a heat source. The exterior temperature is set to 0 °C, the temperature of the heat source is either 0 °C (mode 0) or 1 °C (mode 1). The heat transfers with the exterior and the heat source are modelled by a convective transfer. The full-order system state corresponds to the nodal temperatures. The output is the temperature of the central node. The system is reduced from $n = 897$

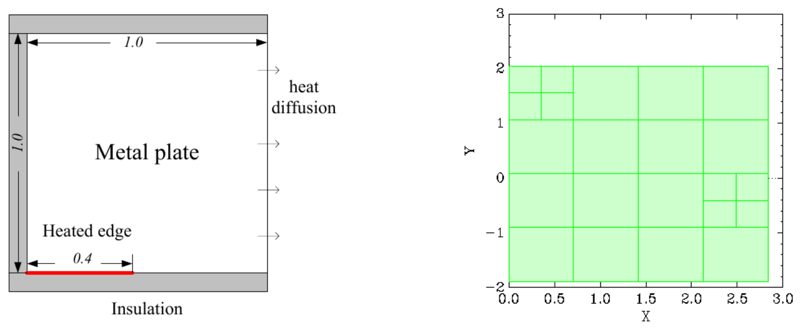


Figure 5 Geometry of the square plate (left) and decomposition of $\hat{R}_x = \pi_R R_x$ in the plane $(\hat{x}_1, \hat{x}_2)$ with the offline procedure (right).

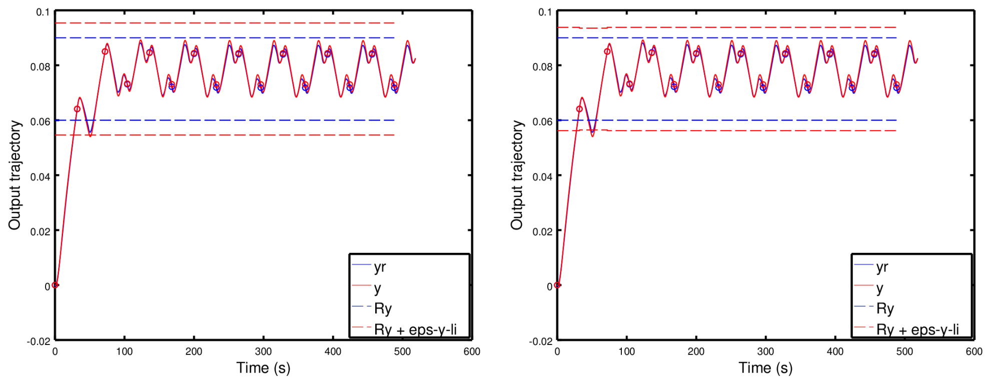


Figure 6 Simulation of $y(t) = Cx(t)$ and $y_r(t) = \hat{C}\hat{x}(t)$ from the initial condition $x_0 = (0)^{897}$. Left: guaranteed offline control; right: guaranteed online control.

to $n_r = 2$. The interest set is $R_x = [0, 0.15]^{897}$, the objective set $R_y = [0.06, 0.09]$. The sampling time is set to $\tau = 8$ s. The geometry of the system and the decomposition obtained with the offline procedure are given in Figure 5. The decompositions were obtained in 5 seconds. Simulations of the offline and online methods are given in Figure 6. We notice that the trajectory y (resp. y_r) exceeds the objective set R_y (resp. $R_y + \varepsilon_y^{\ell_i}$) during the application of the second pattern, yet the markers corresponding to the end of input patterns do belong to objective sets.

7 Final Remarks

Two methods have been proposed to synthesize controllers for switched control systems using model order reduction and the state-space bisection procedure. An offline and an online use are enabled, both uses are efficient but they present different advantages. The offline method allows to obtain the same behaviour as the reduced-order model, but the associated bound is more pessimistic, and the controller has to be computed before the use of the real system. The online method leads to less pessimistic bounds but implies a behaviour slightly different from the reduced-order model, and the limit cycles may be different from those computed on the reduced system. The behaviour of the full-order system is thus less known, but its use can be performed in real time.

There are still some open questions associated to the methods proposed here. During a real online use, only the output y of the system Σ is known, this implies that a reconstruction of the reduced state $\hat{x}$ has to be performed online, either by reconstructing the full-order state x and projecting it, or by reconstructing directly the projected state. Until now, the reconstruction is supposed exact. Our future work will be devoted to the online reconstruction of $\hat{x}$, and this will be done with the use of extended Kalman filters [23, 11].

Acknowledgement. This work is supported by Institut Farman FR3311 through the project SWITCHDESIGN, and by the French National Research Agency (ANR) through the "iCODE Institute project" funded by the IDEX Paris-Saclay, ANR-11-IDEX-0003-02.

References

1 M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of linear systems with uncertain parameters and inputs. In *Proc. of the 46th IEEE Conference on Decision and Control*, pages 726–732, 2007.

2 A. Antoulas and D. C. Sorensen. Approximation of large-scale dynamical systems: an overview. *International Journal of Applied Mathematics and Computer Science*, 11(5):1093–1121, 2001.

3 A. Antoulas, D. C. Sorensen, and S. Gugercin. A survey of model reduction methods for large-scale systems. *Contemporary Mathematics*, 280:193–219, 2000.

4 E. Asarin, O. Bournez, D. Thao, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. *Proc. of the IEEE*, 88(7):1011–1025, July 2000.

5 P. Benner, J.-R. Li, and T. Penzl. Numerical solution of large-scale lyapunov equations, riccati equations, and linear-quadratic optimal control problems. *Numerical Linear Algebra with Applications*, 15(9):755–777, 2008.

6 P. Benner and A. Schneider. Balanced truncation model order reduction for lti systems with many inputs or outputs. In *Proc. of the 19th International Symposium on Mathematical Theory of Networks and Systems*, pages 1971–1974, 2010.

7 F. Blanchini. Set invariance in control. *Automatica*, 35(11):1747 – 1767, 1999.

8 L. Fribourg, U. Kühne, and R. Soulat. Finite controlled invariants for sampled switched systems. *Formal Methods in System Design*, 45(3):303–329, December 2014.

9 L. Fribourg and R. Soulat. *Control of Switching Systems by Invariance Analysis: Application to Power Electronics.* Wiley-ISTE, July 2013. 144 pages.

10 L. Fribourg and R. Soulat. Stability controllers for sampled switched systems. In *Proc. of the 7th Workshop on Reachability Problems in Computational Models (RP)*, volume 8169 of *Lecture Notes in Computer Science*, pages 135–145. Springer, September 2013.

11 B. P. Gibbs. *Advanced Kalman Filtering, Least-Squares and Modeling: A Practical Handbook.* John Wiley & Sons, 2011.

12 A. Girard, G. Pola, and P. Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *Automatic Control, IEEE Transactions on*, 55(1):116–126, Jan 2010.

13 Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Proc. of the 8th International Workshop on Hybrid Systems: Computation and Control (HSCC)*, volume 3414 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2005.

14 Antoine Girard. Synthesis using approximately bisimilar abstractions: state-feedback controllers for safety specifications. In *Proc. of the 13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 111–120. ACM, 2010.

15 A. Han and B.H. Krogh. Reachability analysis of large-scale affine systems using low-dimensional polytopes. In *Proc. the 9th International Workshop on Hybrid Systems: Computation and Control (HSCC)*, volume 3927 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2006.

16 Z. Han and B. H. Krogh. Reachability analysis of hybrid systems using reduced-order models. In *American Control Conference*, pages 1183–1189. IEEE, 2004.

17 M. Herceg, M. Kvasnica, C.N. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzerland, July 17–19 2013.

18 M. Mazo Jr., A. Davitian, and P. Tabuada. Pessoa: A tool for embedded controller synthesis. In *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 566–569. Springer, 2010.

19 B. C. Moore. Principal component analysis in linear systems: Controllability, observability and model reduction. *IEEE Transaction on Automatic Control*, 26(1), 1981.

20 S. Mouelhi, A. Girard, and G. Goessler. CoSyMA: a tool for controller synthesis using multi-scale abstractions. In *Proc. of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 83–88. ACM, April 2013.

21 Octave Web page. http://www.gnu.org/software/octave/.

22 B. Picasso and A. Bicchi. On the stabilization of linear systems under assigned i/o quantization. *IEEE Trans. Automat. Contr.*, 52(10):1994–2000, 2007.

23 K. Reif, S. Günther, E. Yaz Sr., and R. Unbehauen. Stochastic stability of the discrete-time extended kalman filter. *IEEE Transactions on Automatic Control*, 44(4):714–728, 1999.

24 Paulo Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach.* Springer, 2009.

25 D. Tong, W. Zhou, A. Dai, H. Wang, X. Mou, and Y. Xu. h_∞ model reduction for the distillation column linear system. *Circuits Syst Signal Process*, 33:3287–3297, 2014.

A Appendix: Model order reduction and error bounding

A.1 Error bounding for the output trajectory

Here, a scalar *a posteriori* error bound for e is given (adapted from [16]). An upper bound of the Euclidean norm of the error over all possible initial conditions and controls can be formalized as the solution of the following optimal control problem:

$$\varepsilon_y(t) = \sup_{u\in U, x_0\in R_x} \|e(x_0,u,t)\| = \sup_{u\in U, x_0\in R_x} \|y(x_0,u,t) - y_r(\pi_R x_0, u, t)\|$$

Since the full-order and reduced systems are linear, the error bound can be estimated as $\varepsilon_y(t) \le \varepsilon^{x_0=0}(t) + \varepsilon^{u=0}(t)$ where $\varepsilon_y^{x_0=0}$ is the error of the zero-state response, given by

$$\varepsilon_y^{x_0=0}(t) = \max_{u\in U} \|u\| \cdot \|e(x_0=0,u,t)\| = \max_{u\in U} \|u\| \cdot \|y(x_0=0,u,t) - y_r(x_0=0,u,t)\|, \quad (2)$$

and $\varepsilon_y^{u=0}$ is the error of the zero-input response, given by

$$\varepsilon_y^{u=0}(t) = \sup_{x_0\in R_x} \|e(x_0,u=0,t)\| = \sup_{x\in R_x} \|y(x_0,u=0,t) - y_r(\pi_R x_0, u=0,t)\|. \quad (3)$$

Using some algebraic manipulations, one can find a precise bound for $\varepsilon_y^{x_0=0}$ and $\varepsilon_y^{u=0}$ (see [16]). We have:

$$\varepsilon_y^j = \varepsilon_y(j\tau) = \|u(\cdot)\|_\infty^{[0,j\tau]} \int_0^{j\tau} \| \begin{bmatrix} C & -\hat{C} \end{bmatrix} \begin{bmatrix} e^{tA} & \\ & e^{t\hat{A}} \end{bmatrix} \begin{bmatrix} B \\ \hat{B} \end{bmatrix} \| dt \; + \sup_{x_0\in R_x} \| \begin{bmatrix} C & -\hat{C} \end{bmatrix} \begin{bmatrix} e^{j\tau A} & \\ & e^{j\tau\hat{A}} \end{bmatrix} \begin{bmatrix} x_0 \\ \pi_R x_0 \end{bmatrix} \|. \quad (4)$$

The bound $\varepsilon_y^\infty = \sup_{j\ge 0} \varepsilon_y(j\tau)$ is computable when the modulus of the eigenvalues of $e^{\tau A}$ and $e^{\tau\hat{A}}$ is strictly inferior to one, which we suppose here.

A.2 Error bounding for the state trajectory

We recall and introduce some notations, j being the length of the input pattern Pat tested by the bisection method:

$$Post_{Pat}(x) = e^{j\tau A}x + \int_0^{j\tau} e^{A(j\tau - t)}Bu(t)dt,$$

$$Post_{Pat}(\pi_R x) = e^{j\tau\hat{A}}\pi_R x + \int_0^{j\tau} e^{\hat{A}(j\tau - t)}\hat{B}u(t)dt,$$

Using an approach similar to the construction of the bounds (2) and (3), we obtain the following bound, which depends on the length j of the input pattern Pat:

$$\|\pi_R Post_{Pat}(x) - Post_{Pat}(\pi_R x)\| \le \varepsilon_x^j, \quad (5)$$

with

$$\varepsilon_x^j = \|u(\cdot)\|_\infty^{[0,j\tau]} \int_0^{j\tau} \| \begin{bmatrix} \pi_R & -I_{n_r} \end{bmatrix} \begin{bmatrix} e^{tA} & \\ & e^{t\hat{A}} \end{bmatrix} \begin{bmatrix} B \\ \hat{B} \end{bmatrix} \| dt \; + \sup_{x_0\in R_x} \| \begin{bmatrix} \pi_R & -I_{n_r} \end{bmatrix} \begin{bmatrix} e^{j\tau A} & \\ & e^{j\tau\hat{A}} \end{bmatrix} \begin{bmatrix} x_0 \\ \pi_R x_0 \end{bmatrix} \|. \quad (6)$$

Game-based Synthesis of Distributed Controllers for Sampled Switched Systems

Laurent Fribourg[1], Ulrich Kühne[2], and Nicolas Markey[1]

1 LSV, ENS Cachan & CNRS, France
{fribourg,markey}@lsv.ens-cachan.fr

2 Group of Computer Architecture, University of Bremen, Germany
ulrichk@cs.uni-bremen.de

Abstract

Switched systems are a convenient formalism for modeling physical processes interacting with a digital controller. Unfortunately, the formalism does not capture the distributed nature encountered e.g. in cyber-physical systems, which are organized as networks of elements interacting with local controllers. Most current methods for control synthesis can only produce a centralized controller, which is assumed to have complete knowledge of all the component states and can interact with all of them. In this paper, we consider a centralized-controller synthesis technique based on state-space decomposition, and use a game-based approach to extend it to a distributed framework.

1998 ACM Subject Classification B.1.2 Automatic Synthesis

Keywords and phrases cyber-physical systems, controller synthesis, games, robustness, partial observation

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.48

1 Introduction

Hybrid systems are a powerful formalism for modeling and reasoning about cyber-physical systems. They mix the continuous and discrete natures of the evolution of computerized systems. Switched systems are a special kind of hybrid systems, with restricted discrete behaviours: those systems only have finitely many different modes of (continuous) evolution, with isolated switches between modes. Such systems provide a good balance between expressiveness and controllability, and are thus in widespread use in large branches of industry such as power electronics and automotive control.

The control law for a switched system defines the way of selecting the modes during the run of the system. Controllability is the problem of (automatically) synthezing a control law in order to satisfy a desired property, such as safety (maintaining the variables within a given zone) or stabilisation (confinement of the variables in a close neighborhood around an objective point) [6].

In [12], a solution is proposed in order to achieve practical stabilization of discrete-time switched systems. It is based on the repeated bisection of the region of interest surrounding the objective point. Each resulting tile of the bisection is to be associated with a sequence of modes (or *pattern*) that should map the tile inside the zone of interest. Upon success, this naturally induces a control law that becomes cyclic and stabilizes the system along a corresponding limit cycle. However, the decomposition method is proposed in a centralized framework, and assumes that the state of the global system is entirely known. In practice, many industrial systems, such as cyber-physical systems, are implemented in a *distributed* manner, using actuators that are *locally* controlled. Furthermore, these controllers cannot observe the full state of the system, but have only access to the *partial* information conveyed by local sensors.

In the distributed context, the local controllers can be seen as *players*: each player has to achieve a local objective of stabilization. A sufficient condition of global stabilization is obtained when the strategy of any player meets its local objective whatever the strategy of the others. We will adopt this game-oriented view and modify the basic decomposition procedure in order to account for local controllability and partial observability.

Related work. The model of hybrid automata [16] has proven very powerful for combining discrete models issued from software and continuous models issued from the physical world. The topic of synthesizing controllers that, by construction, enforce properties such as safety or reachability, has soon attracted a lot of attention. In [3], a generic methodology is given for constructing safety controllers using iterative computation of reachable states in a backward manner. An alternative approach is proposed in [27], using a game-theoretic formulation and theory of optimal control.

Among hybrid automata, *timed automata* are a very useful class of models where all the state variables correspond to symbolic clocks evolving uniformly at the same rate [1]. In [7], an efficient control synthesis method has been proposed, extending an algorithm for solving games for finite-state systems [19]. It has been implemented in tool UPPAAL-TIGA and applied to industrial case studies [8].

As mentioned above, switched systems constitute another important subclass of hybrid systems well-suited to the modeling of many engineered systems. For this class of systems, a paradigm based on *approximate bisimulation* [25, 14, 15] allows to construct an approximately equivalent discrete model. The original control-synthesis problem can thus be solved at a discrete level, which amounts to computing winning strategies in parity games [23].

Most of the work on controller synthesis in the framework of hybrid systems has focused to the centralized framework. An exception is [20], which gives a methodology based on optimal theory in a multi-agent setting where the agents try to make optimum use of a common resource.

2 Background

2.1 Sampled Switched Systems and Decomposition Method

A *switched system* is a digital quantized control system that consists of a finite family of continuous subsystems, together with a rule that controls the switching between subsystems. Formally, a *switched system* over a set Var of variables can be described by

- a differential equation of the form $\dot{x} = f_\sigma(x)$, where $\{f_u \mid u \in U\}$ is a family of sufficiently regular functions from $\mathbb{R}^{\mathsf{Var}}$ to $\mathbb{R}^{\mathsf{Var}}$ that is parametrized by some finite index set U, called the set of *modes*,
- and a piecewise-constant function $\sigma\colon [0,\infty) \to U$, called *switching rule* [18].

We assume that the system variables have no discrete jump, i.e. the solution $x(\cdot)$ is everywhere continuous. We assume furthermore here that all the individual subsystems are affine, so we obtain an affine switched system of the form $\dot{x} = A_\sigma x + B_\sigma$. Finally, we suppose that σ changes its values *periodically*, at times $k \cdot \tau$, where $\tau \in \mathbb{R}_{>0}$ and k ranges over $\mathbb{N}$. We say that such switched systems are *sampled*. We regard these systems as *discrete-time systems*, observing the state of the system only at the switching instants $k \cdot \tau$. The integration of the continuous equation for mode u during τ still yields an affine equation of the form $x(t+\tau) = \hat{A}_u x(t) + \hat{B}_u$.

In [12], a procedure was designed in order to synthesize a state-dependent control rule that makes all the (discrete-time) trajectories starting from a given set $R \subseteq \mathbb{R}^{\mathsf{Var}}$ repeatedly return to R (the set R is refered to as the *target set* hereafter). In our setting, we require that each variable $v \in \mathsf{Var}$ has its own target zone R_v, so that $R = \prod_{v \in \mathsf{Var}} R_v$ is a rectangular set. The method consists in decomposing R by iterative bisection until (possibly) finding, for each resulting tile, a corresponding pattern (i.e., a sequence of modes) that maps it into R. This guarantees that, starting from any tile W of R, the application of the corresponding pattern π yields a trajectory that ends within R. The crux of the method relies on a simple procedure that, given a tile W, enumerates by increasing length all the patterns, using all possible combinations of modes, until one of them, say π, maps W into R (or until we have exhausted the whole set of patterns up to a given length, in which case the system is declared uncontrollable for the given length). Formally, we write $\mathsf{Post}_\pi(W) \subseteq R$ where Post_π corresponds to the successive application of each mode composing the pattern π (note that Post_π is an affine transformation, because each mode is affine). When such a π exists, we say that the tile W is *successful*.

Each global decomposition Δ of R can be written under the form $(W_i, \pi_i)_{i \in I}$ where I is a finite set of indices, W_i is a tile, and π_i a pattern, such that $\bigcup_{i \in I} W_i = R$, and $\mathsf{Post}_{\pi_i}(W_i) \subseteq R$. For any such decomposition, one can then define an operator Post_Δ as $\mathsf{Post}_\Delta(X) = \bigcup_{i \in I} \mathsf{Post}_{\pi_i}(X \cap W_i)$, for any $X \subseteq R$. Our method aims to find a decomposition Δ of R such that $\mathsf{Post}_\Delta(R) \subseteq R$.

All along the computation of Δ, it may reveal useful to ensure not only that the trajectories return to R after application of each pattern π of the decomposition, but also that the intermediate points of trajectories, obtained after application of each single mode composing π, be confined into a given rectangular set $S \subseteq \mathbb{R}^{\mathsf{Var}}$ containing R. Such a set S is called *safety set*. In order to achieve this additional objective, we strengthen the condition for being successful by requiring the existence of a pattern satisfying:

$$\mathsf{Post}_\pi(W) \subseteq R \quad \text{and} \quad \mathsf{Interm}_\pi(W) \subseteq S \tag{1}$$

where Interm_π refers to the union of all the intermediate sets obtained by the application of the successive prefixes of π. Again, upon success, the resulting decomposition Δ enforces

$$\mathsf{Post}_\Delta(R) \subseteq R \quad \text{and} \quad \mathsf{Interm}_\Delta(R) \subseteq S, \tag{2}$$

where Interm_Δ is defined by $\mathsf{Interm}_\Delta(X) = \bigcup_{i \in I} \mathsf{Interm}_{\pi_i}(X \cap W_i)$ for all $X \subseteq R$. In other terms, it corresponds to a controller enforcing that the global system never leaves the safety zone S, and repeatedly visits the target zone R.

The decomposition method has been implemented in the tool MINIMATOR [21]. This tool makes use of zonotopes [17], and has been written in Octave [22]. At the top-level, the procedure `Decomposition` recursively bisects the target set R until, for each tile, a pattern has been found. It calls the procedure `Find_Pattern`, which implements the search for a correct pattern for a given tile W of the current decomposition. Upon success, the tool constructs a successful decomposition Δ. The tool has been used on various case studies [11, 13].

► **Example 1.** In this paper, we consider as a running example a two-room house equipped with a heating system (a more refined example is described in Section 4). There are heat exchanges between the rooms (characterized by parameters $\alpha_{i,j}$), and heat losses to the outside (with parameters $\alpha_{e,i}$), as schematically depicted on Fig. 1.

Each heater has two modes (`on` and `off`). When turned on, a heater immediately gets hot (to temperature T_f) and heats the room with heat transfer coefficient α_f. The system has

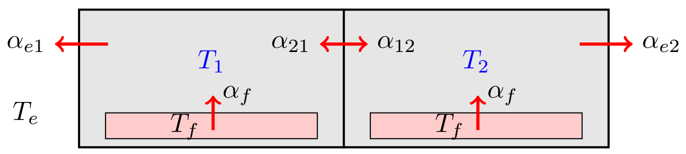


Figure 1 Two-room heating system.

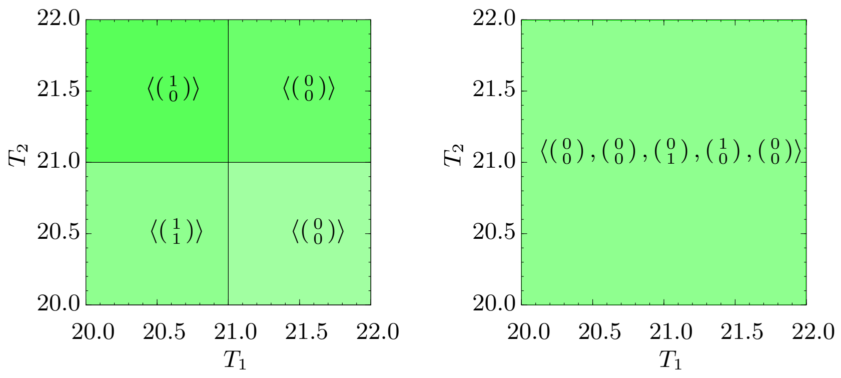


Figure 2 Two valid global controllers for Example 1.

two variables T_1 and T_2, which correspond to the temperatures in rooms 1 and 2. Writing $X = (T_1; T_2)^T$ for the global state of the system, and writing u_1 and u_2 for the modes of the corresponding heaters (assuming u_1 and u_2 belong to $\{0, 1\}$), we get the following equation[1] representing the evolution of the temperatures:

$$\dot{X} = f_{\left(\begin{smallmatrix}u_1\\u_2\end{smallmatrix}\right)}(X) = \begin{pmatrix} -\alpha_{e1} - \alpha_{21} & \alpha_{21} \\ \alpha_{12} & -\alpha_{e2} - \alpha_{12} \end{pmatrix} \cdot X + \begin{pmatrix} u_1\alpha_f T_f + \alpha_{e1}T_e \\ u_2\alpha_f T_f + \alpha_{e2}T_e \end{pmatrix}$$

The objective of our controller is to try to maintain the temperatures in both rooms within a comfort zone $R = [20; 22] \times [20; 22]$, and to ensure that both values never leave the safety zone $S = [19; 23] \times [19; 23]$. Using MINIMATOR, we are able to compute a correct controller for this problem, as depicted in Fig. 2. In fact, the two controllers in the figure represent different trade-offs: The left controller has four different tiles with patterns of length 1, while the controller on the right hand side uses a single uniform pattern of length 5.

2.2 Game-based controller synthesis

Games provide another approach to controller synthesis: in that setting, the controller is seen as one protagonist, playing against other components of the system. A strategy for a player in such a game dictates how the corresponding component must behave, and her winning condition represents the conditions under which the component is said to behave properly.

There is a huge literature on game-based techniques for synthesis [26, 2]. A very large part of these work considers *two-player zero-sum games*: zero-sum games are purely antagonist games, where the objectives of the two players are opposite. This setting corresponds to worst-case controller synthesis: the controller must behave correctly whatever the other players do. Winning strategies then correspond to correct controllers, ensuring correct

[1] For this example, we use $T_e = 10$, $T_f = 50$, $\alpha_{e1} = 0.005$, $\alpha_{e2} = 0.0033$, $\alpha_f = 0.0083$, $\alpha_{12} = \alpha_{21} = 0.05$ and $\tau = 5$.

behavior against any behavior of the environment. In various settings, notably for finite-state systems and ω-regular winning conditions, winning strategies can be computed.

Non-zero-sum games have been considered less intensively: in non-zero-sum games, the objectives are not opposite, and the players may then be interested in cooperating. Such games can be used to reason about the synthesis of *distributed systems*, where several components have their own objectives. In this setting however, winning strategies need not exist (and they would not take into account the possible cooperation between components). Instead, various notions of equilibria can be defined and studied, including the most famous *Nash equilibrium*. Several results exist in this setting; for instance, the existence of pure-strategy Nash equilibria is in general decidable in finite-state games with ω-regular objectives [5], but their existence is undecidable when considering randomized strategies.

An interesting feature of game-based controller synthesis is the ability to take *partial observability* into account. Indeed, in most applications (and especially for distributed control), the components are not able to observe the exact state of the whole system. In the game-based model, this can be taken into account in the definition of *strategies*, requiring them to return the same action for any two situations that are observationally equivalent.

Partial observation can be dealt with when considering zero-sum games with ω-regular objectives [24, 9], but it makes the problem undecidable in more complex settings [10, 4].

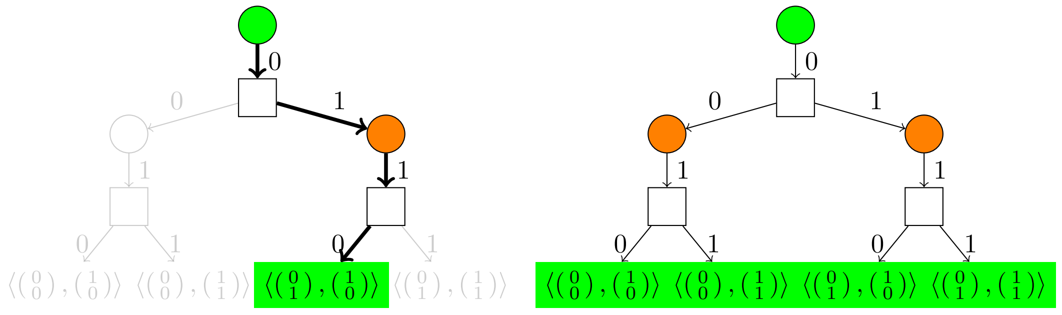
3 Distributed control of sampled switched systems

The invariant-based approach to controller synthesis (depicted in Section 2.1) generates a *centralized* controller, that is, a unique global strategy for the whole system, selecting a global mode u at each time interval τ. This approach assumes *full control* and *full observability* of the whole system. This is due to the structure of the synthesized control algorithm, where the mode switching depends on the local tile to which the system state belongs. Furthermore, we assume that at any switching instant, the controller can choose an arbitrary u from the set of modes U. In many applications, these assumptions are not realistic or would result in an overly complex communication and control infrastructure.

In our running example of a heating system, the controller computed in Example 1 using MINIMATOR selects the mode of the thermostats in both rooms, based on the global state of the system. Such a centralized controller might be *fragile*, in the sense that it is only (or at least was only proven) correct in the case where both thermostats obey the controller strategy. If for some reason the mode selected in one of the rooms is not the expected one (imprecision of the temperature sensor) or is not applied correctly (failure of some component), we have no guarantee about the behavior of the rest of the system.

We address these problems by combining the invariant analysis implemented in MINIMATOR with a game-based view, in order to generate (whenever possible) individual controllers that are correct even if the other components of the system do not behave as expected (but still achieve their objectives[2]), hence adding *robustness* to the whole system. This approach still assumes full observation of the system, and requires some technical restrictions that we explain below. We will then propose a second approach, which assumes partial observation of the system, ending up with robust and fully distributed controllers. Notice that both approaches mainly amount to modifying the notion of being successful for a tile of the decomposition.

[2] Notice that if we do not require the other components to meet their obligations, there is no way of ending up in the target set R.


0
0
1
1
1
0
1
0
1
$\langle\binom{0}{0},\binom{1}{0}\rangle$ $\langle\binom{0}{0},\binom{1}{1}\rangle$ $\langle\binom{0}{1},\binom{1}{0}\rangle$ $\langle\binom{0}{1},\binom{1}{1}\rangle$
0
0
1
1
1
0
1
0
1
$\langle\binom{0}{0},\binom{1}{0}\rangle$ $\langle\binom{0}{0},\binom{1}{1}\rangle$ $\langle\binom{0}{1},\binom{1}{0}\rangle$ $\langle\binom{0}{1},\binom{1}{1}\rangle$

(a) Single global pattern. **(b)** Individual pattern correct in any completion.

Figure 3 Finding successful patterns.

Figure 3 illustrates the difference between our approach and the classical approach of MINIMATOR. The left-hand part of the figure represents the behavior of MINIMATOR: it looks for a pattern (of length 2 in this example) that maps the current set of the decomposition into R (represented in green), and such that at all intermediary steps remain in S (in orange). In this example, a valid pattern is $\langle\binom{0}{1},\binom{1}{0}\rangle$, since the corresponding branch ends up in a green state (representing R) and visits an orange state (corresponding to S) at the intermediate position.

The right-hand part depicts what our algorithm does for checking if pattern $\pi = \langle 0, 1\rangle$ is correct: it has to check that, for any completions of the pattern π with modes for the other components, the image of the original set by this completion is in R, and that it is in S at all intermediary steps. The pattern $\langle 0, 1\rangle$ is then a correct pattern here, as all the completions lead to green configurations (meaning that the taregt zone of the considered player is reached), while all intermediary configurations are orange (corresponding to states in the safety set of the considered player).

3.1 Problem Statement

Consider a sampled switched system as defined in Section 2.1. Distributed control of such a system involving m agents is based on m sets of local modes U_p, with $1 \le p \le m$, which are related to the global modes U by means of a function $\gamma\colon U_1 \times \ldots \times U_m \to U$. In its most simple form, this setting can be implemented by considering that $U = \prod_{1\le p\le m} U_i$, and that $\gamma(u_1, ..., u_m) = (u_1, ..., u_m)$. Since each of the agents has only limited control over the system's behavior, we define local objectives that need to be fulfilled. In this work, we only consider projections on sub-spaces of the problem domain. For this purpose, we assume that the set Var of variables is divided among the players: $\mathsf{Var} = \bigcup_{1\le p\le m} G_p$. We write Γ_p for the set $\mathbb{R}^{G_p}$ of valuations of the variables of agent p. We denote the projection of X on the dimensions in Γ_p by $X \downarrow \Gamma_p$. Each agent p then has to take care of the variables in their set G_p, maintaining them in $S \downarrow \Gamma_p$ and visiting $R \downarrow \Gamma_p$ infinitely often. Notice that since R is a rectangular set, and since for each variable in Var is in some G_p, the following holds: whenever a set X satisfies $(X \downarrow \Gamma_p) \in (R \downarrow \Gamma_p)$ for all $1 \le p \le m$, then $X \subseteq R$. The same holds of S.

3.2 Robust Local Control with Global Observation

In this first approach, the *control* will be localized, while *each controller still has the ability to measure the global system state.* A straightforward solution would be the simple decomposition of a standard controller: this boils down to synthesizing a global controller using the approach of [12], which gives a decomposition and global patterns for each tile of this decomposition. Then each pattern can be projected into m local patterns. Since all agents are acting simultaneously, this results again in a valid controller.

As already explained, this results in a very fragile solution: each controller depends on all other agents in the system. If one agent deviates from this strategy, no guarantees can be made on the global and local objectives (even if the new strategy of the deviating agent is a valid one). A central goal in the synthesis of distributed control is *robustness*: each agent should be able to enforce its own local objective, regardless (to a certain extent) of what the other agents are doing. Thus, we assume that each agent has no knowledge of the strategies of the other agents.

In order to test if a local pattern π is robust, we need to take into account *any possible behavior* of all other agents. For this purpose, the notion of *completion* of a local pattern is used:

▶ **Definition 1.** Given a player p and a local mode $u \in U_p$, the completion of u is defined as

$$\mathsf{Compl}_p(u) = \{w \in U \mid \exists u_1, ..., u_m \text{ s.t. } u_i \in U_i,\ u_p = u,\ \text{and } w = \gamma(u_1, \dots, u_m)\}$$

This notion can easily be extended to patterns. Given a local pattern $\pi \in U_p^+$ with $|\pi| = k$, it is completed to the set of global patterns

$$\mathsf{Compl}_p(\pi) = \{\phi \in U^k \mid \forall 1 \leq j \leq k.\ \phi_j \in \mathsf{Compl}_p(\pi_j)\}.$$

In fact, computing the completion of a local pattern naturally corresponds to exploring a game tree, as explained in the previous section. The leaf nodes of the tree in Fig. 3b correspond to the completion of the local pattern $\langle 0, 1\rangle$ of Player 1. Now, using this definition, we can state what it means for a local pattern π to be robust for a tile W:

▶ **Definition 2.** Player p has a robust strategy for a tile $W \subseteq R$ if there exists a pattern $\pi \in U_p^+$ such that, for all global patterns $\psi \in \mathsf{Compl}_p(\pi)$, the following holds:
1. $\mathsf{Post}_\psi(W) \downarrow \Gamma_p \subseteq R \downarrow \Gamma_p$,
2. $\mathsf{Interm}_\psi(W) \downarrow \Gamma_p \subseteq S \downarrow \Gamma_p$.

We then say that a tile W is successful if all the agents have a robust strategy for W.

A procedure to compute a robust pattern is shown in Algorithm 1. It can be used to compute a distributed control of a sampled switched system based on the basic procedure from [12]. In Algorithm 1, the outer loop searches for a tuple of patterns of *uniform length ℓ*. It does so by enumerating all valid local patterns and checking them for robustness. The procedure terminates successfully if for all agents, a robust pattern of some uniform length ℓ has been found. In order to find a successful decomposition, Algorithm 1 is embedded in the top-level procedure `Decomposition`, which upon success returns a decomposition $(W_i)_{i\in I}$ of R and, for each $i \in I$ and each $1 \leq p \leq m$, a pattern π_i^p. Due to space limitations, this procedure is not shown here, and we refer to [12] for more details.

The fact that we are searching for patterns with uniform length needs to be explained. If we consider the resulting distributed control system in action, then each agent will behave as follows: at some time instant t, it will measure the global system state $X \in R$. According to its local control table and to the tile containing X, each agent will select a pattern of some length ℓ. After $\ell \cdot \tau$ time units, this procedure will be repeated. Now, due to the robustness property of each local control, it is guaranteed that the resulting *global system state* at $t + \ell \cdot \tau$ will again be in R, hence in some tile W of the decomposition, and thus each agent will find a suitable entry in its control table. However, consider a situation where some agent p would play a pattern π of length $\ell' < \ell$ (say). Then, after $\ell' \cdot \tau$ time units, when agent p

```
Algorithm 1: Find_Pattern_Simul($W, R, S, K, m, (\Gamma_p)_p$).
  Input: Sets $W$,$R$,$S$; maximum length $K$; number of players $m$; sub-spaces $(\Gamma_p)_p$
  Output: Patterns $(\pi_1, \ldots, \pi_m)$ of uniform length, where each pattern $\pi_p$ robustly
          satisfies the objective of agent $p$, or $\bot$ if no such patterns exist
1 for $\ell = 1 \ldots K$ do
2     for $p = 1 \ldots m$ do
3         $\Pi := U_p^\ell$;                               // set of local patterns of length $\ell$
4         $\pi_p := \bot$;
5         for $\pi \in \Pi$ do
6             $\Psi := \mathsf{Compl}_p(\pi)$;                  // set of global completions of $\pi$
7             for $\psi \in \Psi$ do
8                 if $\mathit{Post}_\psi(W) \downarrow \Gamma_p \nsubseteq R \downarrow \Gamma_p$ then next $\pi$;
9                 if $\mathit{Interm}_\psi(W) \downarrow \Gamma_p \nsubseteq S \downarrow \Gamma_p$ then next $\pi$;
10            $\pi_p := \pi$; break;                            // valid pattern for agent $p$
11        if $\pi_p = \bot$ then next $\ell$;            // no pattern of length $\ell$ for agent $p$
12    return $(\pi_1, \ldots, \pi_m)$;
13 return $\bot$;
```

measures the system state again in order to find the next pattern to play, it may happen that $X \in S \setminus R$, since the other agents have not finished applying their patterns. Thus, agent p would not be able to choose a new pattern, because the control is limited to R.

Overall, Algorithm 1 computes local patterns that are successful in the sense of Def. 2. Regarding the global controller obtained by this approach, we can definitely assert that it is correct (in the sense of Equation (2)): indeed, the individual patterns computed by our algorithm above can be combined (as they have the same length), and the global pattern obviously satisfies Equation (1), by construction. Given our construction, we would like to assert that each individual pattern is correct against any deviation of the other agents. However, for the same reasons as above, this is only correct w.r.t deviations that use the same pattern lengths, and as long as they achieve their objectives:

▶ **Proposition 3.** Assume that our procedure returns a successful decomposition $\Delta = (W_i, (\pi_i^p)_{1 \leq p \leq m})_{i \in I}$ of R. Then

- for any agent p, for any $i \in I$, any $X \in W_i$, and any completed pattern $\phi \in \mathsf{Compl}_p(\pi_i^p)$, it holds

$$\mathsf{Post}_\phi(X) \downarrow \Gamma_p \in R \downarrow \Gamma_p \qquad \mathsf{Interm}_\phi(X) \downarrow \Gamma_p \subseteq S \downarrow \Gamma_p$$

- for any $i \in I$ and any $X \in W_i$, for the pattern ϕ obtained by combining the patterns π_i^p of all the agents (which is a valid completion of each individual patterns, as they all have the same length), it holds

$$\mathsf{Post}_\phi(X) \in R \qquad \mathsf{Interm}_\phi(X) \subseteq S.$$

- Finally,

$$\mathsf{Post}_\Delta(R) \subseteq R \qquad \mathsf{Interm}_\Delta(R) \subseteq S.$$

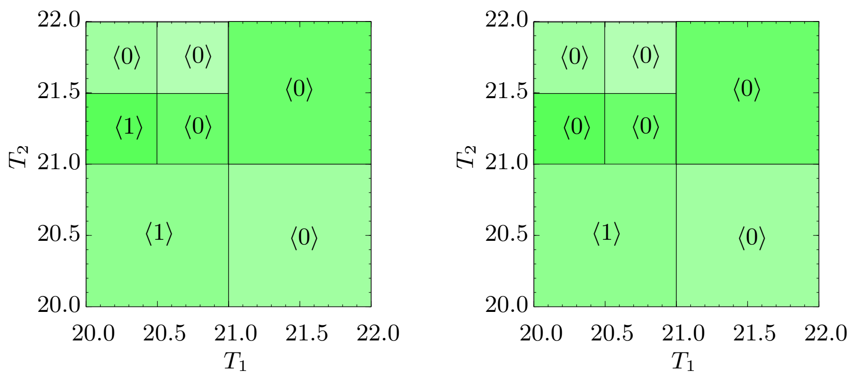


Figure 4 Distributed robust controllers for Example 1 for Players 1 (left) and 2 (right).

▶ **Example 1** (Contd). Consider again the heated rooms from Example 1. A distributed control could be synthesized by separating the problem into a two-player game: the first player controls variable u_1, and has $G_1 = \{T_1\}$; the second player controls u_2, and his objective is on $G_2 = \{T_2\}$. Possible controllers computed by our algorithm are shown in Fig. 4.

3.3 Local Observation

We now extend the above approach to local observation. We assume that each agent can only observe a dedicated sub-space, and can make his decisions only on the basis of this local observations. Compared to the previous approach, this will have two advantages:

- each agent measures only a sub-space, which is more realistic in many cases and allows for simpler (low-dimensional) controllers;
- as a side effect, the patterns can now be desynchronized (and have different lengths), which allows for more admissible controllers.

Some changes are necessary with respect to Algorithm 1. First of all, the local observations allow us to decouple the computation of valid patterns and the decomposition of the target set R: instead of one common decomposition for all m agents, the procedure will compute, for each agent, a successful decomposition of its observed space. In each run, the target set R will only be decomposed along the observed dimensions, according to the following definition:

▶ **Definition 4.** Given a variable $w \in \mathsf{Var}$ and a rectangular set $R \subseteq \mathbb{R}^{\mathsf{Var}}$, writing $R(v) = [a_v, b_v]$ for all $v \in \mathsf{Var}$, the *split of R along variable w* is the set $\mathsf{Split}_{\mathbb{R}^w}(R) = \{R_{\text{left}}, R_{\text{right}}\}$, where

$$R_{\text{left}}(v) = \begin{cases} [a_v, b_v] & \text{if } v \neq w \\ [a_v, \frac{a_v+b_v}{2}] & \text{otherwise} \end{cases} \qquad R_{\text{right}}(v) = \begin{cases} [a_v, b_v] & \text{if } v \neq w \\ [\frac{a_v+b_v}{2}, b_v] & \text{otherwise} \end{cases}$$

This notion is easily extended to a set of variables $V \subseteq \mathsf{Var}$, resulting in a set $\mathsf{Split}_{\mathbb{R}^V}(R)$ of $2^{|V|}$ boxes covering R.

For each agent p, we define its set $O_p \subseteq \mathsf{Var}$ of *observed* variables. ***We require that the set O_p of observed variables be included in his set G_p of variables defining his objective***: for all $1 \leq p \leq m$, we must have $O_p \subseteq G_p$. This condition is needed for the correctness of our procedure: as we explain below, this is precisely the condition that allows us to drop the uniform-length requirement. We write Ω_p for the set $\mathbb{R}^{O_p}$. We also

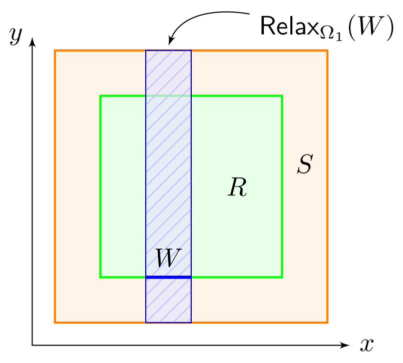


Figure 5 The set $\mathsf{Relax}_{\Omega_1}(W)$ (hatched area), where agent 1 observes only variable x.

write $\Omega_{\bar{p}} = \mathbb{R}^{\mathsf{Var}\setminus O_p}$. As previously, we write $X \downarrow \Omega_p$ for the projection of the set X on Ω_p. Our algorithm will precisely try to compute a successful decomposition of $R \downarrow \Omega_p$. In order to reconstruct the set of possible states that correspond to a given observation, we define the converse of the projection on Ω_p, as follows:

▶ **Definition 5.** Consider a tile $W \subseteq R \downarrow \Omega_p$ observed by an agent p. Its relaxation is the set

$$\mathsf{Relax}_{\Omega_p}(W) = W \times (S \downarrow \Omega_{\bar{p}}).$$

The above definition is visualized in Fig. 5. By relaxing all non-observed dimensions to the invariant set S, we guarantee that the local controller can start a new pattern even if one or several of the other agents have not finished their current pattern (provided their patterns enforce their safety constraints). With these modifications, the patterns that we are looking for can be characterized as follows:

▶ **Definition 6.** Agent p has a strongly-robust strategy for a tile $W \subseteq R \downarrow \Omega_p$ if there exists a pattern $\pi \in U_p^+$ such that, for all global patterns $\psi \in \mathsf{Compl}_p(\pi)$, the following holds:
1. $\mathsf{Post}_\psi(\mathsf{Relax}_{\Omega_p}(W)) \downarrow \Gamma_p \subseteq R \downarrow \Gamma_p$,
2. $\mathsf{Interm}_\psi(\mathsf{Relax}_{\Omega_p}(W)) \downarrow \Gamma_p \subseteq S \downarrow \Gamma_p$.

In this setting, we say that a tile W is successful if all the agents have a strongly robust strategy for W.

The procedure for finding a strongly-robust pattern for some tile W and agent p is shown in Algorithm 2. The top-level procedure, which computes a successful decomposition of a tile W for some agent p, is shown in Algorithm 3. It tries to find a pattern for the whole tile W by calling `Find_Pattern_Local`. If no such pattern can be found, it recurses by splitting the tile W wrt. the observed dimensions. When invoked at some level D of decomposition, the next finer decomposition is called with level $D-1$, and the recursion stops as soon as the finest decomposition has been reached at $D = 0$ without finding a valid pattern. Computing local controllers for m agents boils down to calling `Decomposition`$(R, R, S, K, D, p, \Gamma_p)$ for each agent $p \in \{1, ..., m\}$.

In comparison to the strategy described in the previous section, we can establish stronger guarantees for the overall system's robustness, while slightly relaxing the guarantees wrt. to the target set. Each agent p only measures variables in O_p, while guaranteeing the local objective that the system will return to the projection of R on the variables in G_p. Since $O_p \subseteq G_p$, the subsequent measure of the variables in O_p will be in $R \downarrow \Omega_p$. The only assumption on the other dimensions is their continuous containment inside S. Thus, the

Algorithm 2: `Find_Pattern_Local`$(W, R, S, K, p, \Gamma_p, \Omega_p)$.

```
Input: Sets $W$,$R$,$S$; maximum length $K$; agent $p$; sub-spaces $\Gamma_p$, $\Omega_p$
Output: Pattern $\pi$ robustly satisfying objective of agent $p$; $\perp$ if no such pattern exists
1  $W' := \mathsf{Relax}_{\Omega_p}(W, S)$
2  for $\ell = 1 \dots K$ do
3      $\Pi := U_p^\ell$;                          // set of local patterns of length $\ell$
4      for $\pi \in \Pi$ do
5          $\Psi := \mathsf{Compl}_p(\pi)$;              // set of global completions of $\pi$
6          for $\psi \in \Psi$ do
7              if $\mathit{Post}_\psi(W') \downarrow \Gamma_p \not\subseteq R \downarrow \Gamma_p$ then next $\pi$ ;
8              if $\mathit{Interm}_\psi(W') \downarrow \Gamma_p \not\subseteq S \downarrow \Gamma_p$ then next $\pi$ ;
9          return $\pi$;
10 return $\perp$;
```

global control resulting from the cooperation of the local controllers will remain valid even if any of the controllers are replaced by any strategy that guarantees containment in S.

On the other hand, since the patterns of the distributed controllers can now be played in a decoupled manner, we can no longer guarantee that the global state will return to R. However, a slightly weaker property can be established for each infinite run of the global control system: for each player p, the local objective—the state viewed in the player's sub-space returns to the projection of R—will hold infinitely often.

▶ **Proposition 7.** Assume that our procedure returns successful decompositions $\Delta_p = (W_i^p, \pi_i^p)_{i \in I_p}$ of $R \downarrow \Omega_p$, for each agent p. Then

- for any agent p, for any $i \in I_p$, any $X \in \mathsf{Relax}_{\Omega_p}(W_i^p)$, and any completed pattern $\phi \in \mathsf{Compl}_p(\pi_i^p)$, it holds

$$\mathsf{Post}_\phi(X) \downarrow \Gamma_p \in R \downarrow \Gamma_p \qquad\qquad \mathsf{Interm}_\phi(X) \downarrow \Gamma_p \subseteq S \downarrow \Gamma_p$$

- fix an agent p, an index $i \in I_p$, and some state $X \in \mathsf{Relax}_{\Omega_p}(W_{i_p}^p)$. We define the tree $\mathcal{T}_{p,X}$ inductively as follows:
  - its root is labelled with X, and with the (non-empty) pattern π_i^p;
  - pick a node n with no descendant in the currently-constructed tree; assume that it is labelled with some state Y, and with some non-empty pattern $\rho = u \cdot \rho'$, where u is the first mode of ρ. We then extend the tree by adding sons to n as follows: for each completion w of u, we add a son m_w. We label m_w with $Z = \mathsf{Post}_w(Y)$. We also label it with a pattern, selected as follows:
    - if $Z \notin S$, we label m_w with the empty pattern ϵ;
    - if $Z \in S$ and ρ' is not empty, m_w is labelled with ρ';
    - if $Z \in S$ and ρ' is empty, and if $Z \downarrow \Omega_p \subseteq W_j^p$ for some $j \in I_p$, then we label m_w with π_j^p;
    - finally, if $Z \in S$ and ρ' is empty, but $Z \downarrow \Omega_p \not\subseteq R \downarrow \Omega_p$, m_w is labelled with the empty pattern ϵ.

  We claim that this tree is infinite (i.e., it contains infinite branches), and any infinite branch visits only states in S, and it visits $R \downarrow \Gamma_p$ infinitely many times.

Proof. The first claim is straightforward. The second claim can be proven by noticing that when ρ' becomes empty, agent p has completed his pattern, and provided that the other agents

Algorithm 3: Decomposition$(W, R, S, K, D, p, \Gamma_p, \Omega_p)$.

```
Input: Sets $W$,$R$,$S$; maximum length $K$; depth $D$; agent $p$; sub-spaces $\Gamma_p$, $\Omega_p$
Output: Successful decomposition $\Delta$, or $\perp$ if no such decomposition exists
1  $\pi :=$ Find_Pattern_Local$(W, R, S, K, p, \Gamma_p, \Omega_p)$;
2  if $\pi \neq \perp$ then
3      return $(W, \pi)$;
4  else
5      if $D = 0$ then
6          return $\perp$;                          // finest decomposition reached
7      else
8          Dec := Split$_{\Gamma_p}(W)$; $\Delta := \varnothing$;      // decompose and recursive call
9          for $V \in Dec$ do
10             $\Delta_V =$ Decomposition$(W, R, S, K, D-1, p, \Gamma_p, \Omega_p)$;
11             if $\Delta_V = \perp$ then return $\perp$ ;
12             else $\Delta := \Delta \cup \Delta_V$ ;
13         return $\Delta$;
```

have maintained their variables in their safety sets, the corresponding state Z is in S and is such that $Z \downarrow \Gamma_p \subseteq R \downarrow \Gamma_p$. Since $O_p \subseteq G_p$, it follows[3] that $Z \downarrow \Omega_p \subseteq R \downarrow \Omega_p$, so that the node m_w will be labelled with a non-empty pattern, and the construction can continue. ◀

In the end, let $\Delta = (W_i, (\pi_i^p)_{1 \leq p \leq m})_{i \in I}$ be the decomposition obtained by merging the individual decompositions $(\Delta_p)_{1 \leq p \leq m}$. From Prop. 7, we deduce that if all the agents follow the strategy given by decomposition Δ, then the outcome from any state X will be infinite, it will visit only safe states in S, and each individual target set $R \downarrow \Gamma_p$ will be visited infinitely many times. Again notice that since patterns may have incompatible lengths, we cannot ensure that R itself is visited infinitely many times.

▶ **Example 1** (Contd). We consider our example of the heating system in this setting of local observation, with $O_p = G_p$ for $1 \leq p \leq 2$. The controllers obtained in this setting are depicted on Fig. 6.

4 A more realistic case study

The distributed local controllers obtained for our running example (Fig 6) are very simple: following the natural intuition, their strategy amounts to turning the heater on when the temperature is too low, using only patterns of length one. The only interesting information we get is the exact temperature at which we should turn the heater on or off. In this section, we develop this example a bit further, by assuming that the heaters are reacting slowly.

▶ **Example 2.** Consider a water underfloor heating system: hot water circulates in pipes under the floor, and the controller can open or close the valves. Hot water will first heat the floor, which then in turn transfers heat to the room. The heaters will start to heat up to

[3] If some variable v were in O_p but not in G_p (then it would be in $G_{p'}$ for another agent p'), we would need that agents p and p' play patterns of the same length, in order to ensure $Z \downarrow \Omega_p \subseteq R \downarrow \Omega_p$ when agent p terminates his pattern.

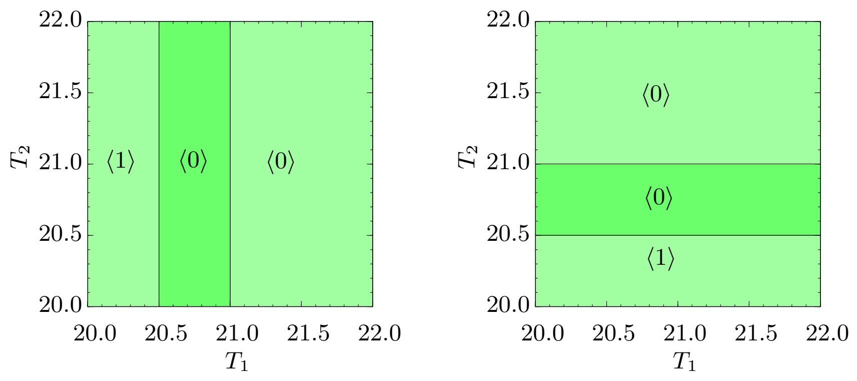


Figure 6 Local controllers for Example 1 for Players 1 (left) and 2 (right).

temperature T_f when switched on. The state $X = (H_1, T_1, H_2, T_2)^T$ of this model is formed by the temperatures of the two rooms (T_1, T_2) and the heaters (H_1, H_2). The dynamics of the model can be described[4] by the equation $\dot{X} = A_{\binom{u_1}{u_2}} X + B_{\binom{u_1}{u_2}}$ with

$$A_{\binom{u_1}{u_2}} = \begin{pmatrix} -\beta_1 - u_1\alpha_f & \beta_1 & 0 & 0 \\ \gamma_1 & -\alpha_{e1} - \gamma_1 - \alpha_{21} & 0 & \alpha_{21} \\ 0 & 0 & -\beta_2 - u_2\alpha_f & \beta_2 \\ 0 & \alpha_{12} & \gamma_2 & -\alpha_{e2} - \gamma_2 - \alpha_{12} \end{pmatrix} \quad B_{\binom{u_1}{u_2}} = \begin{pmatrix} u_1\alpha_f T_f \\ \alpha_{e1} T_e \\ u_2 \alpha_f T_f \\ \alpha_{e2} T_e \end{pmatrix}$$

where $u_1, u_2 \in \{0, 1\}$ indicate the state of the heaters $(0 = \text{off}, 1 = \text{on})$. By discretization with sample time τ, we obtain a switched system $X(t + \tau) = \hat{A}_u \cdot X(t) + \hat{B}_u$.

The global objective of a controller is to keep both rooms at a temperature between 20° and 22°, and the heaters in the comfort zone between 20° and 30°. There are safety margins for the room temperature of 1°. The heaters should not be colder that 15° and should not exceed the maximum of 40°. In other words, the target set is given by $R = ([20, 30] \times [20, 22])^2$, while the safety set is given by $S = ([15, 40] \times [19, 23])^2$. Obviously, $R \subseteq S$.

In order to construct a distributed control for the two rooms, the global state space is projected to the respective dimensions $G_1 = O_1 = \{H_1, T_1\}$ for the first room and $G_2 = O_2 = \{H_2, T_2\}$ for the second room. For all experiments, we used a maximum pattern length of $K = 6$ and a maximum decomposition depth of $D = 3$.

The original implementation of MINIMATOR computes a global controller with a decomposition into 16 tiles (corresponding to a single split in all four dimensions) and patterns of up to three steps. The computation time is 10.45 s, where most of the time is spent on the (failing) attempt to find a single pattern for the whole target set. The approach described in Section 3.2 results in two controllers of similar complexity: 16 tiles with pattern length up to 3. The computation time is slightly higher (13.43 s) due to the more complex exploration of the completed local patterns. Finally, using the approach based on local observations described in Section 3.3, we obtain two simple controllers, each with four (2-dimensional) tiles, as shown in Fig. 8. The computation time was 27.75 s.

[4] For this example, we use the following parameters: $T_e = 10$, $T_f = 40$, $\alpha_{e1} = 0.005$, $\alpha_{e2} = 0.0033$, $\alpha_f = 0.12$, $\alpha_{12} = \alpha_{21} = 0.006$, $\beta_1 = \beta_2 = 0.083$, $\gamma_1 = \gamma_2 = 0.0083$, and $\tau = 5$.

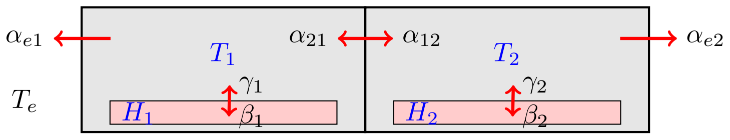


Figure 7 Two-room water underfloor heating system.

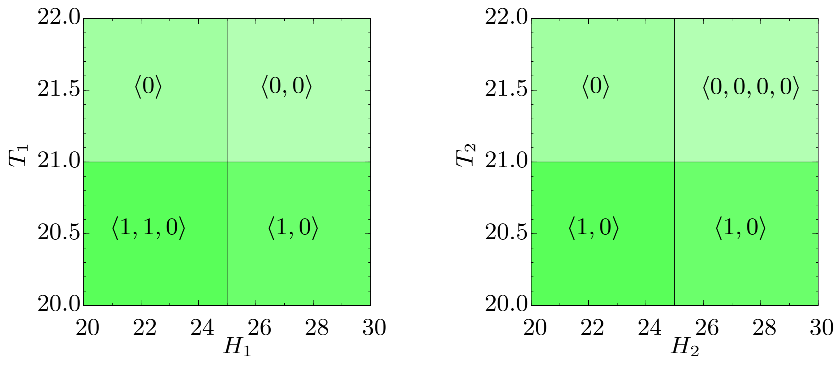


Figure 8 Local controllers for 4-dimensional case study (left: Player 1; right: Player 2).

5 Conclusion

We proposed an extension of the decomposition method for the control of sampled switched systems. The improvement is two-fold: first, we synthesize robust, distributed controllers, which are able to cope with changes in the behaviors of the other controllers of the system; second, our approach can deal with partially observable systems, where controllers may only observe (and base their decisions on) part of the system.

This works opens many directions for future research: following classical results in game theory, it would be natural to make the controller *reconstruct information* about the global state of the system from the evolution of the variables it can observe. This would require that we introduce *memory* in our strategies, and seems to be more than a simple extension of our current approach. Another relevant direction would be to try to use *the same controller* (with partial observation) in all the rooms. This would preserve the partial-observation part of our present approach, but would drop the robustness aspect as the controllers would certainly make use of the fact that all components follow the same strategy. Finally, we would like to extend our approach with *costs*, in order to look for cheap controllers. Notice that just considering the cheapest pattern would only optimize "locally", while it might be more profitable to take a more expensive pattern in order to reach a zone from which control might be cheaper.

Acknowledgments This work has been supported by French network DIGITEO, FP7 project Cassting (FP7-ICT-601148), and iCODE Institute project (ANR-11-IDEX-0003-02).

References

1 R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

2 K. Apt and E. Grädel. *Lectures in Game Theory for Computer Scientists.* Cambridge University Press, 2011.

3 E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli. Effective Synthesis of Switching Controllers for Linear Systems. *Proc. of the IEEE*, 88(7):1011–1025, 2000.

4 P. Bouyer, N. Markey, and S. Vester. Nash equilibria in symmetric games with partial observation. In *SR'14*, EPTCS 146, p. 49–55, Grenoble, France, 2014.

5 R. Brenguier. *Nash equilibria in concurrent games – Applications to timed games.* Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, 2012.

6 R. W. Brockett. Asymptotic stability and feedback stabilization. *Differential geometric control theory*, 27:181–191, 1983.

7 F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR'05*, LNCS 3653, p. 66–80. Springer, 2005.

8 F. Cassez, J. J. Jessen, K. G. Larsen, J. Raskin, and P. Reynier. Automatic synthesis of robust and optimal controllers - an industrial case study. In *HSCC'09*, LNCS 5469, p. 90–104. Springer, 2009.

9 K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games with imperfect information. In *CSL'06*, LNCS 4207, p. 287–302. Springer, 2006.

10 C. Dima and F. L. Ţiplea. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. Research Report 1102.4225, arXiv, 2011.

11 G. Feld, L. Fribourg, D. Labrousse, B. Revol, and R. Soulat. Correct-by-design control synthesis for multilevel converters using state space decomposition. In *FSFMA'14*, EPTCS 156, p. 5–16, 2014.

12 L. Fribourg, U. Kühne, and R. Soulat. Finite controlled invariants for sampled switched systems. *Formal Methods in System Design*, 45(3):303–329, 2014.

13 L. Fribourg and R. Soulat. *Control of Switching Systems by Invariance Analysis: Application to Power Electronics.* Wiley-ISTE, 2013. 144 pages.

14 A. Girard. Synthesis using approximately bisimilar abstractions: state-feedback controllers for safety specifications. In *HSCC'10*, p. 111–120. ACM Press, 2010.

15 A. Girard, G. Pola, and P. Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Trans. on Automatic Control*, 55:116–126, 2010.

16 T. A. Henzinger. The theory of hybrid automata. In *LICS '96*, p. 278–292. IEEE, 1996.

17 W. Kühn. Zonotope dynamics in numerical quality control. In *Mathematical Visualization*, p. 125–134. Springer, 1998.

18 D. Liberzon and A. S. Morse. Basic problems in stability and design of switched systems. *IEEE Control Systems Magazine*, 19:59–70, 1999.

19 X. Liu and S. A. Smolka. Simple linear-time algorithms for minimal fixed points (extended abstract). In *ICALP'98*, LNCS 1443, p. 53–66, London, UK, 1998. Springer.

20 J. Lygeros, D. N. Godbole, and S. Sastry. Multiagent hybrid system design using game theory and optimal control. In *CDC'96*, p. 1190–1195, 1996.

21 Minimator Web page. `https://bitbucket.org/ukuehne/minimator/wiki/Home`.

22 Octave Web page. `http://www.gnu.org/software/octave/`.

23 P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77(1):81–98, 1989.

24 J. H. Reif. The complexity of two-player games of incomplete information. *J. Computer and System Sciences*, 29(2):274–301, 1984.

25 P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach.* Springer Publishing Company, Incorporated, 1st edition, 2009.

26 W. Thomas. On the synthesis of strategies in infinite games. In *STACS'95*, LNCS 900, p. 1–13. Springer, 1995.

27 C. J. Tomlin, J. Lygeros, and S. S. Sastry. A game theoretic approach to controller design for hybrid systems. *Proc. IEEE*, 88:949–970, 2000.

Parameter and Controller Synthesis for Markov Chains with Actions and State Labels

Bharath Siva Kumar Tati and Markus Siegle

Universität der Bundeswehr München, Dept. of Computer Science, Germany
`Bharath.Tati@unibw.de, Markus.Siegle@unibw.de`

Abstract

This paper introduces a novel approach for synthesizing parameters and controllers for Markov Chains with Actions and State Labels (ASMC). Requirements which are to be met by the controlled system are specified as formulas of asCSL, which is a powerful temporal logic for characterizing both state properties and action sequences of a labeled Markov chain. The paper proposes two separate – but related – algorithms for untimed *until* type and untimed general asCSL formulas. In the former case, a set of transition rates and a common rate reduction factor are determined. In the latter case, a controller which is to be composed in parallel with the given ASMC is synthesized. Both algorithms are based on some rather simple heuristics.

1998 ACM Subject Classification C.4 Performance of Systems, D.2.4 Software/Program Verification (Model checking), G.3 Probability and Statistics (Markov processes)

Keywords and phrases Markov chains with actions and state labels, parameter synthesis, controller synthesis, probabilistic model checking

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.63

1 Introduction

Markov chains are widely used to model systems with stochastic behavior and to analyze their quantitative properties such as performance (e.g. utilization, throughput or response time) or dependability (e.g. availability or mean time to failure). Models are usually obtained by transforming from high-level descriptions such as Petri nets or process algebraic descriptions etc. into low-level Markov chains. ASMCs are continuous-time Markov chains extended with actions and state labels. To specify requirements of ASMCs, the temporal logic asCSL [3], which is an extension of CSL [1, 4], has been developed. In particular, asCSL makes it possible to specify complex path-based behavior with the help of regular expressions over state properties and action labels. The process of model checking ASMCs is explained in [3].

The topic of this paper is how to create a controller (also called a supervisor) that controls a given ASMC (also called plant) such that it will satisfy the given path-based requirements specified in asCSL. To this aim, we first study the special case of untimed until-type formulas and then proceed to general untimed path-based requirements. In the former case, we propose an algorithm for parameter synthesis which determines a subset of the ASMC's transitions and a common factor by which those transition rates are to be reduced. In the latter case, a controller is synthesized via a product automaton construction borrowed from the asCSL model checking algorithm where, again, rate reduction plays an important part in this construction. Composing the controller in parallel with the original plant will ensure that the requirement is satisfied. It is important to note that the controller will not only change a set of transition rates, but also potentially change the structural behavior of the plant. We have made the deliberate decision to work with rate reduction factors (as opposed to rate acceleration), since we advocate that it is in general possible to slow down a process (e.g. by reducing the speed of a machine or the capacity of a server), whereas speeding up a process may not be possible, since this would require additional resources. The present paper only considers the controller synthesis problem for untimed asCSL properties, i.e. here we do not consider time-bounded or time-interval-bounded requirements.

Related work: As early as 1993, Lawford and Wonham described an algorithm for synthesizing probabilistic supervisors for a class of probabilistic discrete event systems where a subset of the events is controllable [13], initiating a strand of research that is still active today (see e.g. [14]). The related area of model checking parametric Markov chains has been studied for more than a decade [6]. Some of that work is devoted to the problem of how to deal with the growing symbolic size of the rational functions obtained for the reachability probabilities of interest. This was addressed in [8] for Markovian models with rewards and nondeterminism, and associated tools have been provided [7]. Some approaches for rate parameter synthesis use a discretization of one-dimensional or multi-dimensional parameter ranges over a grid, together with refinement and/or sampling techniques [9]. The recent paper [15] synthesizes rate parameters such that either a given CSL time-bounded property should hold or that the probability of satisfaction is maximized. Their algorithms rely on uniformization combined with the computation of lower and upper bounds as described in [5], and also use parameter range refinement and sampling.

Our approach described in this paper is different in that we do not work with parametric Markov chains, but with Markov chains whose rates are given as constant values. Our problem then is to determine a subset of the transition rates to be modified, and a common reduction factor for those rates, such that a given requirement will be satisfied. In this paper, we consider only untimed requirements, but we deal with the full generality of asCSL-type path properties (without nested probabilistic path operators). This requires the synthesis of a controller which is to be composed in parallel with the given plant, thereby adapting the plant's behavior according to the requirement, possibly also changing its structural behavior.

The rest of the paper is organized as follows: Sec. 2 introduces the fundamental concepts used in this paper. Sec. 3 explains the algorithm to synthesize parameters for untimed until-type formulas, Sec. 4 explains the algorithm to synthesize both parameters and a controller for general untimed asCSL formulas, and Sec. 5 concludes the paper.

2 Preliminaries

This section explains the fundamental concepts used in rest of the paper. A Markov chain with state labels and actions (ASMC) is defined as follows [3]:

▶ **Definition 2.1** (ASMC)**.** An ASMC $\mathcal{M}$ is a tuple (S, Σ, R, L) where
- S is a finite set of states
- Σ is the set of action labels over transitions
- $R : S \times \Sigma \times S \mapsto \mathbb{R}_{\geq 0}$, is the transition function
- $L : S \to 2^{AP}$ is a state labeling function, where AP is a finite set of atomic propositions

A finite untimed *path* σ in an ASMC $\mathcal{M}$ is a finite sequence $\sigma = [(s_0, a_0), (s_1, a_1), \cdots, (s_{n-1}, a_{n-1}), s_n] \in (S \times \Sigma)^* \times S$ and with $\mathit{Paths}(s)$ we denote the set of all finite paths originating from state s. Probabilities are assigned to sets of finite paths by the usual cylinder set construction on sets of infinite paths. An ASMC without action labels is called a state-labeled CTMC. So, the underlying CTMC for an ASMC is given by the tuple (S, R', L), which is a result of removing the action labels and accumulating the rates of parallel transitions, i.e., $R'(s, s') = \sum_{a \in \Sigma} R(s, a, s')$. In order to specify user requirements and characterize execution paths of ASMCs, we use the logic asCSL [3] (without time bounds and without the steady-state operator), which is an extension of the purely state-based logic CSL (continuous stochastic logic) [4].

▶ **Definition 2.2** (State formulas of asCSL)**.** The grammar for untimed asCSL state formulas is given as:

$$\Phi \ ::= \ q \mid \neg\Phi \mid \Phi \vee \Phi \mid P_{\sim b}(\alpha)$$

where $q \in AP$ is an atomic proposition, $\neg$ denotes negation, $\vee$ denotes disjunction, $b \in (0,1)$ denotes a probability value, $\sim \in \{<, \leq, >, \geq\}$ a comparison operator and α is a program as defined in Def 2.3. $P_{\sim b}(\alpha)$ asserts that the probability measure of the set of paths satisfying α meets the bound given by $\sim b$. The program α specifies the property for finite paths.

Remark 1. In contrast to [3], this paper considers probability bounds $b \in (0,1)$ instead of $b \in [0,1]$, since the approach presented here does not aim to turn a non-zero probability into zero, or to turn a probability smaller than one into one. Thus, we do not treat requirements of the form $P_{\leq 0}(\alpha)$ or $P_{\geq 1}(\alpha)$, and for similar reasons we also do not treat requirements of the form $P_{>0}(\alpha)$ or $P_{<1}(\alpha)$.

▶ **Definition 2.3** (Program)**.** asCSL-programs are defined by the following grammar:

$$\alpha \ ::= \ \varepsilon \mid (\phi, b) \mid \alpha; \alpha \mid \alpha \cup \alpha \mid \alpha^*$$

Formally, programs are regular expressions over the alphabet $\Omega = \Phi \times \big(\Sigma \cup \{\surd\}\big) = \big\{(\phi, b) \mid \phi \in \Phi \wedge b \in \big(\Sigma \cup \{\surd\}\big)\big\}$. The operator ; denotes sequential composition, $\cup$ denotes alternative choice, and $*$ denotes Kleene star. Intuitively, program (ϕ, b) means that the current state s should satisfy ϕ, and then the next action taken along the path should be b. If $b \in \Sigma$, an outgoing b-transition has to be taken, and if $b = \surd$ (pseudo-action $\surd \notin \Sigma$), no transition is taken. The full formal semantics of asCSL is given in [3].

Untimed asCSL is an extension of untimed CSL, so every CSL formula can be expressed in asCSL. The syntax and semantics of untimed *Until* formulas are as explained in [4]. For our purpose we consider only the *Until* operator, because the parameter synthesis for the *Next* operator follows trivially from the algorithm for the *Until* operator. For the sake of completeness we provide the semantics of CSL until-type path formulas.

▶ **Definition 2.4** (Untimed Until)**.** The satisfaction relation $\models$ for untimed *Until* path formulas is defined as:

$$\sigma \models \Phi_1 \,\mathcal{U}\, \Phi_2 \quad \text{iff } \exists k \geq 0 : \sigma[k] \models \Phi_2 \wedge \forall (0 \leq i < k) : \sigma[i] \models \Phi_1$$

where Φ_1, Φ_2 are state formulas, and $\sigma[k]$ denotes the k-th state on path σ.

From here on, untimed *Until* is simply called *Until*. Any CSL untimed *Until* property can be expressed in asCSL as $\Phi_1 \,\mathcal{U}\, \Phi_2 = (\Phi_1, \Sigma)^*; (\Phi_2, \surd)$, which follows from Prop. 12 in [3].

Let $Sat(\Phi)$ denote the set of states fulfilling state formula Φ. Partitioning of the ASMC state space is required to accomplish the process of parameter and controller synthesis. In particular, during the synthesis procedure, our attention will be on the states of the so-called *transit* class. This motivates the following definition:

▶ **Definition 2.5** (Partitioning of ASMC)**.** Given an ASMC $\mathcal{M}$ and an asCSL requirement $\Phi = P_{\sim b}(\alpha)$, the states with:

- $Pr(s, \alpha) = 0$ are placed in *invalid* class

- $0 < Pr(s, \alpha) < 1$ are placed into *transit* class and
- $Pr(s, \alpha) = 1$ are placed into *target* class

where, $Pr(s, \alpha)$ is defined as the probability measure of the set of paths $Pr(s, \alpha) = Pr(\sigma \in Paths(s) \mid \sigma \models \alpha)$.

in the given state formula, $\sim$ and b have no influence on the partitioning of these three classes. Furthermore, for an ASMC $\mathcal{M}$ and a state formula Φ, we introduce the following satisfaction relation:

$$\mathcal{M} \models_{transit} \Phi \iff \forall s \in transit : s \models \Phi$$

So, the given user requirement Φ is said to be satisfied by ASMC $\mathcal{M}$, iff all the states of the *transit* class satisfy Φ. The reason for this viewpoint is as motivated in *Remark 1*.

After controller synthesis, in order to satisfy the user requirement, parallel composition of the plant and the controller is necessary (see Sec. 4). Therefore we now provide a definition for parallel composition of ASMCs. Note that different stochastic process algebras possess different semantics for parallel composition [10]. For our purpose, in case of synchronization the resulting rate of two actions with rates λ and μ shall be determined by their product $\lambda \cdot \mu$ (where, in practice, one of the two factors is either equal to one or a slowdown factor $0 < k \leq 1$).

▶ **Definition 2.6** (Parallel composition in ASMCs)**.** The parallel composition of two ASMCs P and Q is defined by the following rules (analogous to [2, 12]):

$$\frac{P \xrightarrow{a,\lambda} P', Q \xrightarrow{a,\mu} Q'}{P \parallel_{\Sigma_{syn}} Q \xrightarrow{a,\lambda\cdot\mu} P' \parallel_{\Sigma_{syn}} Q'} \quad (a \in \Sigma_{syn})$$

and

$$\frac{P \xrightarrow{a,\lambda} P'}{P \parallel_{\Sigma_{syn}} Q \xrightarrow{a,\lambda} P' \parallel_{\Sigma_{syn}} Q} \quad (a \notin \Sigma_{syn})$$

and a third rule, symmetric to the second one, where Q makes a move while P remains stable. In these rules, $a \in \Sigma_{syn} \subseteq \Sigma$ is a synchronizing action, and λ, μ are the transition rates. The labeling of a state (p_i, q_j) in the product ASMC is defined to be the union of the labellings of p_i and q_j.

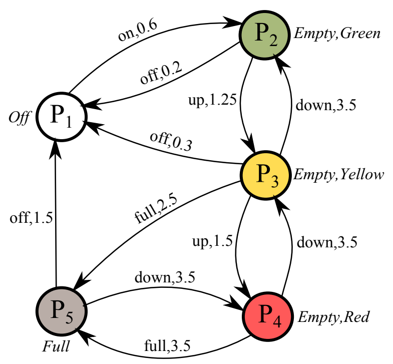
When we employ parallel composition in Sec. 4, one process will be the plant $\mathcal{P}$, the other process will be the controller $\mathcal{C}$, the set of synchronizing actions Σ_{syn} will be equal to the action set $\Sigma_{\mathcal{C}}$ of the controller, and all rates of the controller will be either equal to one or equal to a common reduction factor k, with $0 < k \leq 1$.

3 Parameter Synthesis for "Until"-type requirements

For until-type requirements, the ASMC parameter synthesis problem is intuitively explained as computing a reduction factor k for a subset of the transition rates in the original plant, so as to modify some reachability probabilities as needed. We will start with a simple example.

3.1 Example

This example considers a gas tank with an automatic filling pump, which can be turned off or on based on the levels of the tank.



$$R'_P = \begin{pmatrix} 0 & 0.6 & 0 & 0 & 0 \\ 0.2 & 0 & 1.25 & 0 & 0 \\ 0.3 & 3.5 & 0 & 1.5 & 2.5 \\ 0 & 0 & 3.5 & 0 & 3.5 \\ 1.5 & 0 & 0 & 3.5 & 0 \end{pmatrix}$$

Figure 1 Tank $\mathcal{P}$ shown as an ASMC with respective accumulated rate matrix.

3.1.1 Unrestricted plant $\mathcal{P}$

Fig. 1 shows the gas tank $\mathcal{P}$ modeled as an ASMC along with the respective transition rates. Nodes represent different states of the tank and edges represent transitions between states. Initially, the gas tank can be in any state. *Empty* and *Full* represent different levels of the tank, and level *Empty* is further divided into *Green, Yellow, Red* for easy level reading. Note that the action labeling of transitions is irrelevant for this section.

The user requirement on $\mathcal{P}$ is given as an untimed *Until* formula Φ,

$$\Phi \;=\; P_{\leq 0.7}\,(\varphi), \text{where } \varphi = \mathit{Empty}\,\mathcal{U}\,\mathit{Full} \tag{1}$$

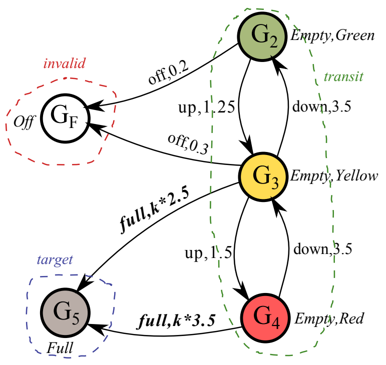
which checks whether the probability to reach a *Full* state from an *Empty* state, possibly via intermediate *Empty* states is at most 0.7. By using the PRISM tool [11], we computed the probabilities of the states in ASMC $\mathcal{P}$ to be

$$Pr(P_1, \varphi) = p_{15} = 0 < 0.7 \tag{2}$$
$$Pr(P_2, \varphi) = p_{25} = 0.69473 < 0.7 \tag{3}$$
$$Pr(P_3, \varphi) = p_{35} = 0.80589 > 0.7 \tag{4}$$
$$Pr(P_4, \varphi) = p_{45} = 0.90294 > 0.7 \tag{5}$$
$$Pr(P_5, \varphi) = p_{55} = 1 \tag{6}$$

where p_{i5} = *Prob(to reach state* P_5 *from state* P_i *via a satisfying path)* and $Sat(\Phi) = \{P_1, P_2\}$. According to Definition 2.5, $\mathit{invalid} = \{P_1\}$, $\mathit{target} = \{P_5\}$ and $\mathit{transit} = \{P_2, P_3, P_4\}$. From the above equations, we know state P_2 already satisfies Φ, whereas P_3 and P_4 do not. Hence, parameter synthesis is required on $\mathcal{P}$. This is done by reducing some of the transition rates in $\mathcal{P}$ by a factor of k. To determine k, we create a reduced automaton $\mathcal{G}$.

3.1.2 Obtain the reduced automaton $\mathcal{G}$ from $\mathcal{P}$

Fig. 2 shows the ASMC $\mathcal{G}$, which has been partitioned according to the definition 2.5. The difference of $\mathcal{G}$ and $\mathcal{P}$ lies in making the *invalid* and *target* classes absorbing in $\mathcal{G}$. According to the heuristics explained in Sec. 3.2.3, in R'_G the rates of the transitions leading from the *transit* class towards the *target* class should be reduced by the factor k. The satisfying range of k between 0 and 1 should be obtained, such that the probabilities p_{35} and p_{45} fall below 0.7 as required by equation (1).



$$R'_G = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0 & 1.25 & 0 & 0 \\ 0.3 & 3.5 & 0 & 1.5 & \boldsymbol{k} \times 2.5 \\ 0 & 0 & 3.5 & 0 & \boldsymbol{k} \times 3.5 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 2 Reduced automaton $\mathcal{G}$ of $\mathcal{P}$, and its rate matrix R'_G.

3.1.3 System of equations from $\mathcal{G}$

For each state G_i in *transit* class, we now construct a new equation (depending on k) for the probability p_{i5}. In our example, equations for p_{25}, p_{35} and p_{45} are obtained from the reduced automaton $\mathcal{G}$ in Fig. 2 as follows:

$$p_{25} = \frac{1.25}{1.45} \times p_{35} \tag{7}$$

$$p_{35} = \frac{3.5}{k \times 2.5 + 5.3} \times p_{25} + \frac{1.5}{k \times 2.5 + 5.3} \times p_{45} + \frac{k \times 2.5}{k \times 2.5 + 5.3} \tag{8}$$

$$p_{45} = \frac{3.5}{k \times 3.5 + 3.5} \times p_{35} + \frac{k \times 3.5}{k \times 3.5 + 3.5} \tag{9}$$

According to equation (1), the following constraints need to be met:

$$0 < k \leq 1 \qquad 0 \leq p_{25} \leq 0.7 \qquad 0 \leq p_{35} \leq 0.7 \qquad 0 \leq p_{45} \leq 0.7$$

Upon solving the system of equations in (7),(8),(9) along with the constraints, we obtain the satisfying range of k to be $0 < k \leq 0.326244$. Thus, the solution of the parameter synthesis problem consists of changing $\mathcal{P}$ by multiplying the transition rates from P_3 to P_5 and from P_4 to P_5 by such a factor of k. We now proceed to the general algorithm and heuristics required to solve the parameter synthesis problem.

3.2 General Algorithm

We assume that the original plant is defined as an ASMC, $\mathcal{P}=(S_P, \Sigma_P, R'_P, L_P)$, and the user requirement is specified by $\Phi = P_{\sim b}(\Phi_1 \,\mathcal{U}\, \Phi_2)$. Algorithm 1 shows the procedure of how to determine the set of transition rates to be reduced and how to synthesize the reduction factor k. For the algorithm to deliver the correct result, the user-given *Until* formula should not contain nested probabilistic formulas. Note that this algorithm provides a simple way to control a plant according to the user requirement, but other, more distinguished, approaches would be also possible (see Sec. 5).

3.2.1 Generating the reduced automaton $\mathcal{G}$

The first step towards solving the parameter synthesis problem is to create a reduced automaton $\mathcal{G}$ from $\mathcal{P}$.

Algorithm 1 Parameter synthesis algorithm for untimed *Until* formulas.

```
Input: Plant $\mathcal{P}$ expressed as ASMC and requirement $\Phi = P_{\sim b}(\Phi_1 \,\mathcal{U}\, \Phi_2)$
Output: Set of transitions to be modified and the values of reduction factor $k$
if $transit = \emptyset$ then quit                    ▷ No states whose probabilities can be modified
else
    if $\mathcal{P} \models_{transit} \Phi$ then quit       ▷ No need of par-synthesis, since req. is already fulfilled
    else
        Construct $\mathcal{G} = (S_G, \Sigma_G, R'_G, L_G)$                                 ▷ Refer Sec. 3.2.1
        Find set of trans. rates $T$ to be reduced and reduction factor $k$  ▷ Refer Sec. 3.2.2
        Change $\mathcal{P}$ to $\mathcal{P}_{mod}$ by reducing the rates of $T$ by factor $k$
    end if
end if
```

▶ **Definition 3.1** (Reduced automaton $\mathcal{G}$)**.** The reduced automaton $\mathcal{G}$ is defined as a tuple $(S_G, \Sigma_G, R'_G, L_G)$ where:

- $S_G = S_P$
- $\Sigma_G = \Sigma_P$
- Partitioning of P is done according to Def. 2.5
- $R'_G = R'_P \setminus \{(s, a, s') \mid s \notin target \vee s \notin invalid\}$
- $L_G = L_P$

As explained in Algorithm 1, $\mathcal{G}$ is created only if $transit \neq \emptyset$ and $\mathcal{P} \not\models_{transit} \Phi$. Once $\mathcal{G}$ has been constructed according to Def. 3.1, some rates of R'_G need to be chosen for synthesis based upon some heuristics (sec 3.2.3) on the given property Φ.

3.2.2 Obtain k from $\mathcal{G}$

Parameter synthesis is based upon some rules as follows:

1. Transition rates in R'_G cannot exceed the original rates in R'_P and cannot be made zero.
2. Some transition rates in R'_G will be reduced by a common factor k (where $0 < k \leq 1$), to make sure that the probability during model checking will satisfy the bound given in Φ. The set of transition rates to be multiplied with k is determined according to the heuristics explained in Sec. 3.2.3.
3. Create a system of (rational polynomial) equations for model checking. There is one equation for each state from *transit* class, representing its probability to reach the *target* class via a satisfying path.
4. Impose the constraints on probabilities according to the given Φ.
5. Solve these equations and constraints for the probabilities, and the resultant k leads to a model that satisfies the user requirement.

3.2.3 Heuristics on $\mathcal{G}$

In R'_G, some of the rates need to be reduced by a common factor k, such that $\mathcal{G}$ satisfies the requirement Φ. The following heuristics are applied to modify the rates. If in the given formula Φ

1. the probability bound is a lower bound, i.e. $P_{\geq b}(\varphi)$ or $P_{>b}(\varphi)$, then the rates of all transitions leading from class *transit* to class *invalid* should be multiplied by the reduction factor k,

2. or, if the probability bound is an upper bound, i.e. $P_{\leq b}(\varphi)$ or $P_{<b}(\varphi)$, then the rates of all transitions leading from class *transit* to class *target* should be multiplied by the reduction factor k.

These heuristics determine the set of transition rates to be reduced, and the common reduction factor k $(0 < k \leq 1)$ can then be found by solving the set of equations with the imposed constraints.

▶ **Theorem 3.2.** *Assume that transit $\neq \emptyset$. Then the set of constrained equations constructed according to the heuristics in Sec. 3.2.3 will always have a solution.*

Proof. For the reduction factor k it holds that $0 < k \leq 1$. This means that one can make the chosen set of transition rates arbitrarily close to zero (without completely disabling the transition). The heuristics distinguishes three classes *transit*, *target* and *invalid*, of which the latter two classes are absorbing. The *target* and the *invalid* class are reachable from the *transit* class by definition. Hence, the uniform reduction of rates between two classes will make the transitions between the other two classes more likely and this will always lead to a solution. ◀

We end this section by characterizing the relation between the ASMC $\mathcal{P}$ and its reduced automaton $\mathcal{G}$. If $\mathcal{G}$ satisfies Φ, that implies that the modified version of $\mathcal{P}$ will also satisfy Φ, because of the fact that $\mathcal{G}$ focuses only on *transit* class of $\mathcal{P}$. This is stated by the following Theorem.

▶ **Theorem 3.3.** *For any ASMC $\mathcal{P}$ and its reduced automaton $\mathcal{G}$, and for any untimed until-type formula Φ without nested probabilistic path operators, it holds that*

$$\mathcal{G} \models_{trans} \Phi \implies \mathcal{P}_{mod} \models_{trans} \Phi$$

where $\mathcal{P}_{mod}$ is the modified version of $\mathcal{P}$ by applying the reduction factor k to the selected transitions.

Proof. The proof follows directly from the construction of $\mathcal{G}$ and from the semantics of until-type formulas. ◀

4 Controller synthesis for untimed asCSL requirements

Having seen the procedure to synthesize parameters for until-type requirements, we now propose an algorithm to synthesize a controller for general untimed asCSL formulas. Given a plant $\mathcal{P}$ and a user requirement in the form of an asCSL formula, we propose a novel approach to synthesize a controller $\mathcal{C}$, which is another ASMC to be composed in parallel with the plant. Controller synthesis will also involve the synthesis of parameters as a subtask. To better understand the synthesis procedure, we again start this section with an example, followed by the general algorithm.

4.1 Example

Fig. 3 shows a plant modeled as an ASMC $\mathcal{P}$. The states can have atomic propositions, but they are not relevant for this example. The transitions shown among the states are labeled with the actions followed by their respective rates. The plant consists of 4 states and has the ability to start in any state.

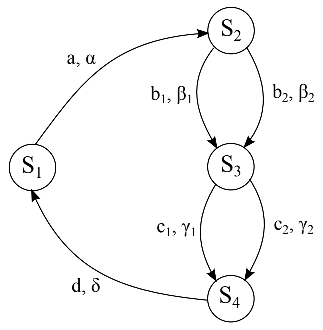


Figure 3 ASMC of plant $\mathcal{P}$.

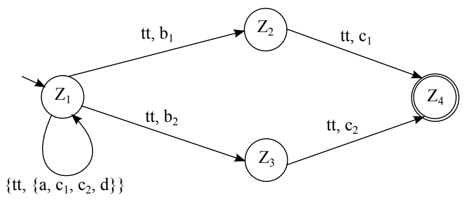


Figure 4 NPA $\mathcal{Z}_\alpha$ for the given asCSL program α.

Assume that we wish to ensure that in $\mathcal{P}$, once action b_1 has taken place, it will be followed by action c_1 with high probability, and the same for actions b_2 and c_2. For this criterion, the asCSL user requirement is given as follows,

$$\Phi \; = \; P_{>0.5}(\alpha) \tag{10}$$

where, α is the following asCSL program:

$$\alpha \; = \; ((tt,a) \cup (tt,c_1) \cup (tt,c_2) \cup (tt,d))^*; \Big(\big((tt,b_1);(tt,c_1) \big) \cup \big((tt,b_2);(tt,c_2) \big) \Big) \tag{11}$$

Formula Φ states that the probability to take action b_i followed by c_i, where $i \in \{1,2\}$, should be greater than 0.5, and *tt* stands for *true*. The first part of α (covered by the Kleene star), states that the initial behavior before either action b_1 or b_2 occur can be arbitrary. For model checking ASMCs against an asCSL requirement, a NPA $\mathcal{Z}_\alpha$ (non-deterministic program automaton) is constructed from the program α. The automaton $\mathcal{Z}_\alpha$ for the given α is shown in Fig. 4, and it happens to be deterministic for this example. According to the asCSL model checking algorithm [3], we now construct the product automaton $\mathcal{Q}$ for $\mathcal{P}$ and $\mathcal{Z}_\alpha$. Fig. 5 shows this product automaton $\mathcal{Q}$, with the accepting state Q_8 and the absorbing fail state Q_7. For model checking we considered the rates in $\mathcal{P}$ to be $\alpha = 2$, $\beta_1 = 4$, $\beta_2 = 6$, $\gamma_1 = 6$, $\gamma_2 = 4$ and $\delta = 3$. Model checking the product automaton (with PRISM [11]) gave that the probability of $\mathcal{P}$ satisfying the given Φ is 0.48 (the same for all states in $\mathcal{P}$), which is a violation of our requirement in equation (10). Hence, we need to synthesize a controller. A controller has the ability to modify both the transition rates and also the structural behavior of an ASMC, which is achieved by using parameter synthesis and controller synthesis respectively. For parameter synthesis, we need to construct a blueprint automaton $\mathcal{B}$ (similar to the purpose of $\mathcal{G}$ in Sec. 3) from the product automaton $\mathcal{Q}$, which is useful for applying heuristics and also to synthesize the controller. Hence, the name blueprint automaton.

4.1.1 Blueprint automaton $\mathcal{B}$ for this example

In the product automaton $\mathcal{Q}$, partitioning of the state space is performed according to the Def. 2.5, with $\varphi = tt\,\mathcal{U}\,target$. Furthermore, states starting from the initial state of $\mathcal{Z}_\alpha$ are exclusively placed into set *initial*. The division of the state space is as below,

1. *target* class contains all states where $Pr\{q,\varphi\} = 1$, i.e. state Q_8,
2. *invalid* class contains those states where $Pr\{q,\varphi\} = 0$, i.e. state Q_7,
3. *transit* class contains the states where $0 < Pr\{q,\varphi\} < 1$, i.e. states Q_1, Q_2, Q_3, Q_4, Q_5 and Q_6.

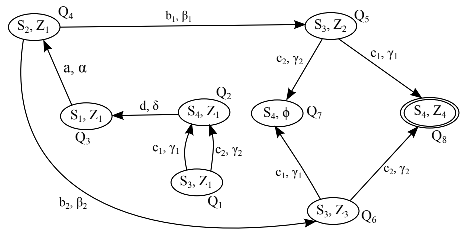


Figure 5 Product automaton $\mathcal{Q}$.

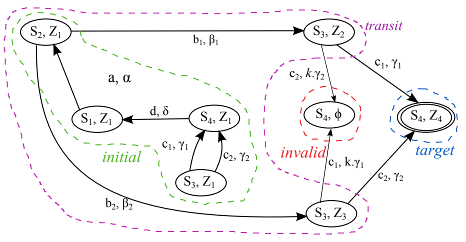


Figure 6 Blueprint automaton $\mathcal{B}$ from $\mathcal{Q}$.

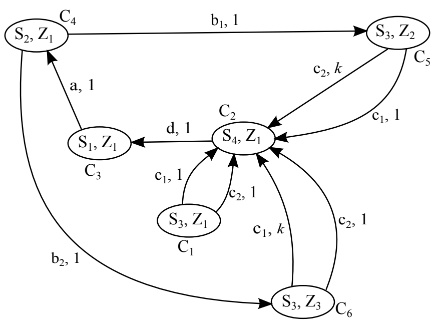


Figure 7 Controller $\mathcal{C}$ for the plant $\mathcal{P}$ and requirement Φ.

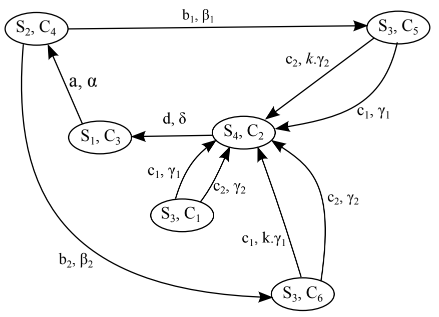


Figure 8 Result of parallel composition $\mathcal{P} \|_{\Sigma_\mathcal{C}} \mathcal{C}$.

4. *initial* set contains only those states which start from the initial states of $\mathcal{Z}_\alpha$, i.e. states Q_1, Q_2, Q_3 and Q_4.

Note that the *initial* set is not disjoint from the other classes (e.g. some *initial* states can be *invalid* or *transit*). The reason for this classification is to identify those states which should satisfy the probability bound given in Φ (10). Fig. 6 shows the classes of $\mathcal{B}$ obtained from $\mathcal{Q}$. From the automaton $\mathcal{B}$, some of the rates are then chosen for parameter synthesis using heuristics (Sec. 4.2.2). In order to increase the probability from 0.48 to above 0.5 as in the given Φ, the rates leaving from *transit* to *invalid* class are considered for parameter synthesis. The aim now is to find a suitable value k $(0 < k \leq 1)$ as the common reduction factor for these rates. According to the model checking algorithm for asCSL [3], in the product automaton we only check for $\varphi = tt\,\mathcal{U}\,target$ formula. As we will see later, to make $\mathcal{P}$ satisfy Φ, it is sufficient if only the states in the *initial* set of $\mathcal{B}$ satisfy Φ, but the states Q_1, Q_2, Q_3 will not affect the probability as they have single outgoing transition. Hence, only state Q_4 needs to be considered. The probability is computed using the equations below.

$$Pr(Q_4, \varphi) = p_{48} = \frac{\beta_1}{\beta_1 + \beta_2} \times \frac{\gamma_1}{\gamma_1 + k \times \gamma_2} + \frac{\beta_2}{\beta_1 + \beta_2} \times \frac{\gamma_2}{k \times \gamma_1 + \gamma_2} \tag{12}$$

$$\text{with constraints:} \qquad 0 < k \leq 1 \qquad 0.5 < p_{48} \leq 1 \tag{13}$$

Solving this constrained quadratic inequality for p_{48} yields k to be $(0 < k < 0.92)$. Hence any k value from this range satisfies the user given Φ.

4.1.2 Creating controller $\mathcal{C}$ for $\mathcal{P}$

The controller will restrict the behavior of the plant according to the user requirements. We create a controller $\mathcal{C}$, such that $(\mathcal{P} \|_{\Sigma_\mathcal{C}} \mathcal{C}) \models_{init \,\cap\, transit} (P_{\sim b}(\alpha))$. Hereby, the notation $\models_{init \,\cap\, transit} (P_{\sim b}(\alpha))$ means that those states $s = (S_i, C_j)$ of $\mathcal{P} \|_{\Sigma_\mathcal{C}} \mathcal{C}$, where $0 < Pr(s, \varphi) <$

Algorithm 2 Controller synthesis algorithm for general untimed asCSL formulas.

```
Input: Plant $\mathcal{P}$ modeled as ASMC and an untimed asCSL specification $\Phi = P_{\sim p}(\alpha)$
Output: Controller $\mathcal{C}$ such that $(\mathcal{P} \|_{\Sigma_\mathcal{C}} \mathcal{C}) \models_{init \cap transit} \Phi$
Construct product automaton $\mathcal{Q}$ from $\mathcal{P}$ and $\mathcal{Z}_\alpha$ as in [3] and classify its states
Let $\varphi = (tt\, \mathcal{U}\, target)$
if $transit = \emptyset$ then quit                    ▷ No states whose probabilities can be modified
else
    if $\forall s \in transit \cap initial : s \models \Phi$ then quit          ▷ No need of contr. synthesis
    else
        Construct $\mathcal{B} = (S_\mathcal{B}, \Sigma_\mathcal{B}, R_\mathcal{B}, L_\mathcal{B})$ from $\mathcal{Q}$          ▷ Refer Sec. 4.2.1
        From $\mathcal{B}$, find set of trans. to be modified and the red. factor $k$   ▷ Refer Sec. 4.2.1
        Modify the absorbing states in $\mathcal{B}$ and multiply the selected rates with $k$
        The result is controller $\mathcal{C}$
    end if
end if
```

1 and C_j contains an initial state of Z_α will satisfy Φ. We use $\mathcal{B}$ as a blueprint for the controller, but in order to make its behavior non-blocking we make all the absorbing states of $\mathcal{B}$ non-absorbing as follows:

- The absorbing states $(S_4, \emptyset)$ and (S_4, Z_4) are removed and all the transitions leading to them are diverted to the state (S_4, Z_1), because when the plant $\mathcal{P}$ has reached state S_4 and whether or not the previous trajectory has satisfied the asCSL program α, the controlling needs to start again from the initial state of $\mathcal{Z}_\alpha$.
- The rates which we obtained by parameter synthesis are replaced by the reduction factor k and the rest of the rates are intentionally set to 1.

Fig. 7 shows the controller $\mathcal{C}$ created for the plant $\mathcal{P}$. The transition rates obtained via parameter synthesis from the states C_5 and C_6 to C_2 are multiplied by the reduction factor k, and the rest of the rates are unchanged, hence the multiplication factor is 1. We already obtained the value of k. When the parallel composition is performed between the original plant $\mathcal{P}$ and the controller $\mathcal{C}$, synchronizing over all the actions in $\Sigma_\mathcal{C}$ as per the ASMC parallel composition rules (Def. 2.6), the resultant product automaton satisfies the user requirement Φ. The result of parallel composition of plant $\mathcal{P}$ and the controller $\mathcal{C}$ is shown in Fig. 8.

4.2 General Algorithm for parameter synthesis for untimed asCSL

In Algorithm 1, we explained the general procedure to synthesize the parameters for *until* type requirements. Now we give an algorithm to synthesize a controller for general untimed asCSL requirements (which also includes parameter synthesis). A prerequisite for the algorithm to work correctly is that the asCSL program α should not contain any nested probabilistic formulas.

4.2.1 Blueprint automaton $\mathcal{B}$

The blueprint automaton $\mathcal{B}$ is based on the product automaton $\mathcal{Q}$ of the plant $\mathcal{P}$ and $\mathcal{Z}_\alpha$.

▶ **Definition 4.1** (Blueprint automaton $\mathcal{B}$). The blueprint automaton $\mathcal{B}$ is obtained from the product automaton $\mathcal{Q}$, along with the following modifications:

- In $\mathcal{Q}$, partitioning of state space is done according to Def. 2.5 for $\varphi = tt\,\mathcal{U}\,target$.
- Once the partitioning is done, a new set called *initial* is identified which contains all the states starting from the initial states of NPA $\mathcal{Z}_\alpha$.
- Some transition rates of $\mathcal{Q}$ are multiplied by the reduction factor k as per the heuristics in Sec. 4.2.2.

From the resultant automaton $\mathcal{B}$, a set of equations is created, taking into account Th. 27 in [3] which states that for an ASMC $\mathcal{P}$ and an asCSL-program α it holds that

$$Prob^{\mathcal{P}}(s,\alpha) = Prob^{\mathcal{P}\times\mathcal{Z}_\alpha}(\langle s, Z_0\rangle, tt\,\mathcal{U}\,target)$$

where s is a state in $\mathcal{P}$ and Z_0 is the set of initial states of NPA $\mathcal{Z}_\alpha$. From this theorem it follows that for an ASMC $\mathcal{P}$ to satisfy the probability bound in the asCSL formula Φ, it suffices to make the states of the form $\langle s, Z_0\rangle$ in the product automaton $\mathcal{Q}$ satisfy the probability bound. Therefore, we construct one equation for each state in $transit \cap initial$, representing its probability to reach the *target* class via a satisfying path. Solving these equations yields the value of k.

4.2.2 Heuristics on $\mathcal{B}$

The heuristics on $\mathcal{B}$ are similar to those of $\mathcal{G}$ (Sec. 3.2.3). It applies between the classes *transit*, *invalid* and *target*, as the new set *initial* will not play any role in heuristics.

4.2.3 Controller $\mathcal{C}$

The controller is derived from the blueprint automaton $\mathcal{B}$, but to make the controller non-blocking, all absorbing states in $\mathcal{B}$ should be made non-absorbing (by redirecting their incoming transitions). The controller $\mathcal{C}$ is thus obtained by modifying $\mathcal{B}$ as follows:

1. Replace all the states from the *invalid* class of the form $(S_i, \emptyset)$ with states of the kind (S_i, Z_0), where Z_0 is the set of starting states of the automaton $\mathcal{Z}_\alpha$.
2. All the states from the *target* class like (S_i, Z_j) (Z_j is a set containing an accepting state of $\mathcal{Z}_\alpha$) are replaced with (S_i, Z_0).

5 Conclusion

For a given ASMC model (the plant) and an asCSL path-based requirement, we have studied the problem of controlling the plant in such a way that its states will satisfy the requirement, wherever possible at all. Satisfaction can be achieved by either a reduction of a subset of the plant's transition rates, or by parallel composition with a controller. We have presented two algorithms: Algorithm 1 performs parameter synthesis for untimed until-type requirements, and Algorithm 2 extends this concept to controller synthesis for general untimed asCSL formulas.

In this paper, we have restricted our attention to requirements without nested probabilistic path operators. Such nesting requires special care, since changing some parameters in order to satisfy an inner probabilistic path formula can have either a positive or an adverse effect on the satisfiability of the enclosing formula. As a simple example, the strategy to maximize the satisfaction set of $P_{\geq b_1}(\Phi_1\ \mathcal{U}\ P_{\geq b_2}(\Phi_2\ \mathcal{U}\ \Phi_3))$ will not be the same as for $P_{\leq b_1}(\Phi_1\ \mathcal{U}\ P_{\geq b_2}(\Phi_2\ \mathcal{U}\ \Phi_3))$, since in the latter case the set of states satisfying the overall formula will be larger if fewer states satisfy the inner formula. We intend to elaborate on this in a forthcoming paper.

The paper has presented feasible solutions, but did not address the question of optimality. Solutions which are "better" in some sense could be obtained by applying more complex heuristics than the ones described in this paper, for instance by allowing non-uniform reduction factors or by also reducing some rates within the class *transit*. However, how do characterize the optimal solution and how to obtain it is future work. Another important issue for future work is the control problem for *time-bounded* problems which will involve the computation of transient state probabilities with the method of uniformization.

Acknowledgments. We would like to thank the anonymous reviewers for their valuable constructive feedback and also our colleague Alexander Gouberman for his suggestions.

References

1 A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model Checking Continuous Time Markov chains. In *Computer-Aided Verfication*, 1996.

2 J. Bachmann, M. Riedl, J. Schuster, and M. Siegle. An efficient symbolic elimination algorithm for the stochastic process algebra tool CASPA. In *35th Int. Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM'09)*, pages 485–496. Springer LNCS 5404, 2009.

3 C. Baier, L. Cloth, B. Haverkort, M. Kuntz, and M. Siegle. Model checking Markov chains with Actions and State labels. *IEEE Trans. on Software Engineering*, 33(4):209–224, 2007.

4 C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE Trans. on Software Engineering*, 29(7), July 2003.

5 L. Brim, M. Češka, S. Dražan, and D. Šafránek. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, volume 8044 of *LNCS*, pages 107–123. Springer Berlin Heidelberg, 2013.

6 C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *Theor. Aspects of Computing - ICTAC 2004*, volume 3407 of *LNCS*, pages 280–294. Springer Berlin Heidelberg, 2005.

7 E.M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. Param: A model checker for parametric Markov models. In *Computer Aided Verification*, volume 6174 of *LNCS*, pages 660–664. Springer Berlin Heidelberg, 2010.

8 E.M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric Markov models. *Int. Journal on Software Tools for Technology Transfer*, 13(1):3–19, 2011.

9 T. Han, J.-P. Katoen, and A. Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *Real-Time Systems Symposium*, pages 173–182. IEEE, 2008.

10 H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274:43–87, 2002.

11 A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. 12th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS' 06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

12 M. Kuntz and M. Siegle. Deriving symbolic representations from stochastic process algebras. In *Process Algebra and Probabilistic Methods, Proc. PAPM-PROBMIV'02*, pages 188–206. Springer, LNCS 2399, 2002.

13 M. Lawford and W.M. Wonham. Supervisory control of probabilistic discrete event systems. In *Proc. of the 36th Midwestern Symposium on Circuits and Systems, 1993.*, pages 327–331, vol.1, 1993.

14 V. Pantelic, M. Lawford, and S. Postma. A framework for supervisory control of probabilistic discrete event systems. In *12th Int. Workshop on Discrete Event Systems (WODES 2014)*, pages 477–484, vol.12, 2014.

15 M. Češka, F. Dannenberg, M. Kwiatkowska, and N. Paoletti. Precise parameter synthesis for stochastic biochemical systems. In *Computational Methods in Systems Biology*, volume 8859 of *LNCS*, pages 86–98. Springer International Publishing, 2014.

Parametric Verification of Weighted Systems

Peter Christoffersen, Mikkel Hansen, Anders Mariegaard, Julian Trier Ringsmose, Kim Guldstrand Larsen, and Radu Mardare

Aalborg University
Selma Lagerlöfs Vej 300, Denmark
`{pchri10, mhan10, amarie10, jrings10} @student.aau.dk`
`{kgl, mardare} @cs.aau.dk`

Abstract

This paper addresses the problem of parametric model checking for weighted transition systems. We consider transition systems labelled with linear equations over a set of parameters and we use them to provide semantics for a parametric version of weighted CTL where the until and next operators are themselves indexed with linear equations. The parameters change the model-checking problem into a problem of computing a linear system of inequalities that characterizes the parameters that guarantee the satisfiability. To address this problem, we use *parametric* dependency graphs (PDGs) and we propose a global update function that yields an assignment to each node in a PDG. For an iterative application of the function, we prove that a fixed point assignment to PDG nodes exists and the set of assignments constitutes a well-quasi ordering, thus ensuring that the fixed point assignment can be found after finitely many iterations. To demonstrate the utility of our technique, we have implemented a prototype tool that computes the constraints on parameters for model checking problems.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases parametric weighted transition systems, parametric weighted CTL, parametric model checking, well-quasi ordering, tool

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.77

1 Introduction

Specification and modelling formalisms that address non-functional properties of embedded and distributed systems have been intensively studied in the last decades. In particular the modelling formalism of timed automata [3] has established itself as a very useful formalism for expressing and analysing timing constraints of systems with respect to timed logics such as TCTL [2] and MTL [20]. This has naturally sparked interest for formal verification of functionality, and model checking techniques have often been used. However, timing constraints are not the only non-functional properties of interest in applications. Often modelling resources that can be consumed and must be monitored during the evolution of a system, such as energy, is an important issue in applications. This initially led to weighted extensions of timed automata [7, 5] and more recently to energy automata [9]. Such formalisms are typically abstracted as weighted transition systems, which are transition systems having the transition labelled by real numbers (or vectors of reals) that represent the resource consumption; and weighted versions of temporal logics are used to encode and verify logical properties.

In this paper we consider a parametric extension of the concept of weighted transition system by allowing the transition labels to be not only numbers, but also linear equations over a given set of parameters. Similarly, a parametric version of weighted CTL is defined, where

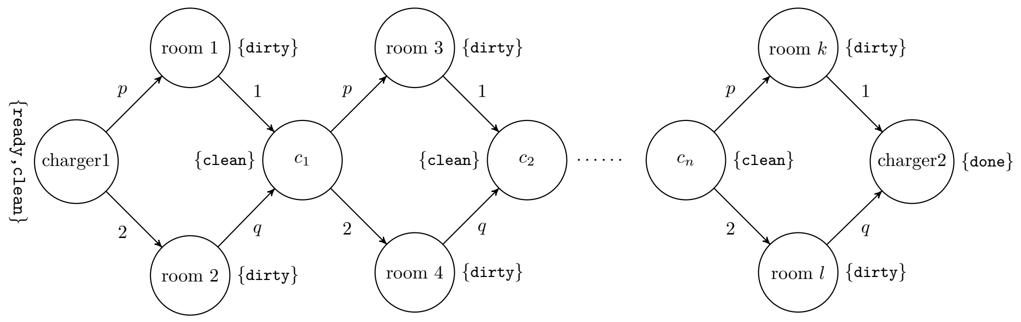


Figure 1 Model $\mathcal{M}_{ex}$ for the robotic vacuum cleaner example.

the temporal operators are themselves indexed with linear equations representing upper bounds for the computation traces. In this context, we address the problem of parametric model checking: given a parametric model and a parametric logical property, compute a linear system of inequalities that describe the possible values of the parameters that will guarantee that the model satisfies the property.

As an example, consider a robotic vacuum cleaner. We want to know whether, in a given situation, our robot can reach and clean a dirty room within some time frame. While we know how many rooms there are, we do not have full knowledge of the time spent by the robot from room to room and neither do we know how long time a room takes to clean. The knowledge of the problem can be represented graphically as depicted in Figure 1, where the parameters p, q represent our lack of knowledge. We want to check whether the robot can finish the job under some linear constraints over the parameters. This is the kind of problems we aim to address and solve in this paper.

The main contribution of this work, is the definition of a global function which iteratively computes a fixed point on the so-called *parametric dependency graphs* (PDGs). The PDGs in this paper are structures that handle parametric quantitative properties in a manner inspired by the Symbolic Dependency Graphs proposed in [19]. We use PDGs to represent problems by stating the dependencies between a given problem and its sub-problems along with their parametric quantitative constraints on the solution to the problem. We use Tarski's Fixed Point Theorem [24] to show that a fixed point exists. In this regard, we also prove that it can be reached in a finite number of steps using Dickson's Lemma [13] for well-quasi orders. Using these results, we show how to build PDGs for encoding of model checking, where parameters exist on both model and formula. Parametric model checking then entails finding constraints on parameter interpretation that makes a property satisfied. In some cases the constraints may be either trivially satisfied or not satisfiable, making it possible to give an yes/no answer. However, in the general case the answer is the set of parameter constraints obtained as a system of linear inequalities. Our technique can be further used to solve related issues such as bisimulation checking for parametric models ([10]).

In addition to the theoretical work of this paper, we have implemented a prototype tool (PVTool) that computes the fixed point for verification of parametric properties on parametric models. The input for the tool is a model that may have parametric weights on transitions, together with a state and a formula. Outputted is then a derivation of the exact constraints for interpretations of the parameters in the model and formula for the property to be satisfied by the model. Finally, we have conducted a series of preliminary performance experiments using the tool.

Related Work. In recent years, various extensions to modeling formalisms and logics have been developed. In [23], a Parametric Kripke Structure and parametric CTL is presented along with algorithms computing constraints on parameters that must be satisfied for a parametric formula to be satisfied in a state. This inspired the authors of this paper to study the behavior of parametric weighted transition systems in [8] where various notions of bisimulation on parametric systems are discussed. In the world of automata, [4] proposes an extension to Timed Automata with parametric guards and invariant called Parametric Timed Automata. For parameter synthesis [6] uses an *inverse* method relying on an initial *good* parameter valuation for each parameter used to compute a constraint on parameters. It is then shown that if some parameter valuation satisfies the constraint, the system is behaviorally equivalent (identical traces) to the system under the initial valuation. This method is in contrast to traditional methods such as [15] based on *Counterexample Guided Abstraction Refinement* (CEGAR, [12]) where the goal is to avoid a set of *bad* states given beforehand. Tool support for various formalisms have also been developed. In [16], parameter synthesis of Linear Hybrid Automata in HyTech is discussed and [18] provides an extension to UPPAAL (see [21]) capable of generating parameter constraints that are necessary and sufficient for a reachability or invariant property to hold for Linear Parametric Timed Automata. Related to our work is also [19] where Dependency Graphs, introduced by Liu and Smolka (see [22]), are extended to Symbolic Dependency Graphs suitable for model checking using Weighted CTL and Weighted Kripke Structures. They compute a fixed point on *assignments* to nodes, being simply the cost of satisfying a formula in the weighted setting. We extend this to the parametric setting where we guarantee termination by exploiting that assignments can be interpreted as a well-quasi ordering. Refer to [14, 1] for a discussion of well-quasi orders for termination of algorithms operating on infinite structures.

2 Parametric Weighted Models and Logic

We now introduce various concepts used throughout this paper. We let Q denote either universal- or existential-quantification, i.e. $Q \in \{A, E\}$. Furthermore, we let $\min\{\emptyset\} = \infty$ and $\max\{\emptyset\} = 0$ and denote the set of natural numbers, including zero, by $\mathbb{N}$.

A *Parametric Weighted Transition System* (PTS) is an extension of weighted transition systems with linear expressions of parameters as labels on the transitions. From here on, $\mathcal{P}$ denotes a fixed finite set of *parameters*. $\mathcal{E}$ is the set of all linear expressions on the form $\sum x_i p_i + y$ where $x_i, y \in \mathbb{N}$ and $p_i \in \mathcal{P}$ for any $i \in \mathbb{N}$.

Finally, PTS label states with a set of atomic propositions from the fixed finite set $\mathcal{AP}$.

We now proceed by formally defining a PTS:

▶ **Definition 1.** A *Parametric Weighted Transition System (PTS)* $\mathcal{M}$ is a triple

$$\mathcal{M} = (M, \rightarrow, \boldsymbol{\ell}), \text{ where}$$

- M is a finite non-empty set of *states*.
- $\rightarrow \subseteq M \times \mathcal{E} \times M$ is the transition relation.
- $\boldsymbol{\ell} : M \longrightarrow 2^{\mathcal{AP}}$ is a labeling function mapping states in M to a set of atomic propositions

Whenever $(m, e, m') \in \rightarrow$ we use the shorthand notation $m \xrightarrow{e} m'$. We will use $\mathfrak{M}$ to denote the set of all PTSs.

A run ρ in a PTS $\mathcal{M} = (M, \rightarrow, \boldsymbol{\ell})$ is a possibly infinite sequence $\rho = m_0 e_1 m_1 e_2 m_2 ...$ where $\forall i \in \mathbb{N}, m_i \in M$ and for $i > 0, e_i \in \mathcal{E}$ we have $m_{i-1} \xrightarrow{e_i} m_i$. A run is *maximal* if it is infinite or the last state in the run has no outgoing transitions. *Runs*(m) denotes all maximal runs ρ starting from m and $Runs(\mathcal{M})$ denote all maximal runs in $\mathcal{M}$. $\rho(i)$ denotes the state of ρ with index i. Finally $|\rho|$ denotes the length of a run ρ given by the number of states. Runs over PTSs accumulate sums of linear expressions from the transitions in the run. Given a position $j \in \mathbb{N}$ in a run ρ, let $\rho(j) = m_j$. The *accumulated weight*, $\mathcal{AW}_\rho(j)$, of ρ at position j is then defined by $\mathcal{AW}_\rho(j) = \sum_{i=1}^{j} e_i$ if $j > 0$ and $\mathcal{AW}_\rho(j) = 0$ if $j = 0$. We denote by $\boldsymbol{out}(m)$ the set of all expression-successor pairs of outgoing transitions from m.

The notion of interpretations, presented next, was first introduced in [8]. We will briefly explain the concept of interpretations; for further discussions, see [8]. Interpretations on the set of parameters is a mapping of each parameter to a natural number. As in [8], we will first define the simplest form of interpretations, namely the interpretation directly on parameters.

▶ **Definition 2.** $\boldsymbol{i} : \mathcal{P} \longrightarrow \mathbb{N}$ is a function mapping each parameter to a natural number.

We denote the set of all interpretations by $\mathfrak{I}$.

Interpretations are extended to the domain $\mathcal{E}$, by letting for an arbitrary $x \in \mathbb{N}$, $\boldsymbol{i}(x) = x$ and by requiring that $\boldsymbol{i}$ preserves polynomial structures, i.e., for arbitrary $p_1, ..., p_k \in \mathcal{P}$ and $x_1, ...x_{k+1} \in \mathbb{N}$:

$$\boldsymbol{i}(x_1 p_1 + ... + x_k p_k + x_{k+1}) = x_1 \boldsymbol{i}(p_1) + ... + x_k \boldsymbol{i}(p_k) + x_{k+1}$$

To reason about parametric properties of PTS, we define a parametric extension of Weighted Computation Tree Logic (WTL) called *Parametric Temporal Logic* (PTL) and give a satisfiability relation of formulae w.r.t. PTS. Interpretations are used to interpret both formulae bounds and PTS edge expressions.

▶ **Definition 3.** The set of PTL *state formulae* are given by the abstract syntax:

$\Phi, \Psi ::= \top \mid \bot \mid \mathsf{a} \mid \Phi \wedge \Psi \mid \Phi \vee \Psi \mid E\varphi \mid A\varphi$

and the set of PTL *path formulae* are given by the abstract syntax:

$\varphi ::= X_{\leq e}\Phi \mid \Phi U_{\leq e}\Psi$

where $\mathsf{a} \in \mathcal{AP}$ and $e \in \mathcal{E}$.

Whether a PTL formula is *satisfied* by a state m or a run ρ of some PTS $\mathcal{M}$ with an interpretation $\boldsymbol{i}$, is given by the satisfiability relation $\models_{\boldsymbol{i}}$. We denote this by $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi$ and $\mathcal{M}, \rho \models_{\boldsymbol{i}} \varphi$, respectively.

▶ **Definition 4.** Given a PTL formula, a PTS $\mathcal{M} = (M, \rightarrow, \boldsymbol{\ell})$, a state $m \in M$ or a run $\rho \in Runs(\mathcal{M})$ and an interpretation $\boldsymbol{i} \in \mathfrak{I}$, the *satisfiability relation* $\models_{\boldsymbol{i}}$ is inductively defined as:

$\mathcal{M}, m \models_{\boldsymbol{i}} \top$		always
$\mathcal{M}, m \models_{\boldsymbol{i}} \bot$		never
$\mathcal{M}, m \models_{\boldsymbol{i}} \mathsf{a}$	iff	$\mathsf{a} \in \boldsymbol{\ell}(m)$
$\mathcal{M}, m \models_{\boldsymbol{i}} \Phi \wedge \Psi$	iff	$\mathcal{M}, m \models_{\boldsymbol{i}} \Phi$ and $\mathcal{M}, m \models_{\boldsymbol{i}} \Psi$
$\mathcal{M}, m \models_{\boldsymbol{i}} \Phi \vee \Psi$	iff	$\mathcal{M}, m \models_{\boldsymbol{i}} \Phi$ or $\mathcal{M}, m \models_{\boldsymbol{i}} \Psi$
$\mathcal{M}, m \models_{\boldsymbol{i}} E\varphi$	iff	there exists $\rho \in Runs(m)$, such that $\mathcal{M}, \rho \models_{\boldsymbol{i}} \varphi$
$\mathcal{M}, m \models_{\boldsymbol{i}} A\varphi$	iff	for all $\rho \in Runs(m)$, it is the case that $\mathcal{M}, \rho \models_{\boldsymbol{i}} \varphi$
$\mathcal{M}, \rho \models_{\boldsymbol{i}} X_{\leq e}\Phi$	iff	$\|\rho\| > 0$, $\boldsymbol{i}(\mathcal{AW}_\rho(1)) \leq \boldsymbol{i}(e)$ and $\mathcal{M}, \rho(1) \models_{\boldsymbol{i}} \Phi$
$\mathcal{M}, \rho \models_{\boldsymbol{i}} \Phi U_{\leq e}\Psi$	iff	there exists $j \in \mathbb{N}$ s.t. $\mathcal{M}, \rho(j) \models_{\boldsymbol{i}} \Psi$ where $\boldsymbol{i}(\mathcal{AW}_\rho(j)) \leq \boldsymbol{i}(e)$ and $\mathcal{M}, \rho(j') \models_{\boldsymbol{i}} \Phi$ for all $j \in \mathbb{N}, j' < j$

3 Analysis of Parametric Properties

In this section we present a global method for analysis of various parametric problems using fixed point computations on Parametric Dependency Graphs (PDG). In this work we show how to abstract model checking problems into finding minimal fixed point assignments on PDGs specifically built for model checking. Refer to the technical report [10] for missing proofs.

3.1 Fixed Point Computations on Parametric Dependency Graphs

As presented in [19], Symbolic Dependency Graphs (SDG) can be used as an abstraction of problems with quantitative dependencies. We give a parametric extension to SDGs called *Parametric Dependency Graph* (PDG). This section will introduce PDGs in a general and formal fashion without context. Intuition and example of application is given in Section 3.2, where we show how to encode the model checking problem.

As the name suggests, Dependency Graphs encode dependencies. These dependencies may arise from the optimal substructure of a given problem. We encode this through the notion of *hyper-edges* with multiple targets, one for each dependency. In the most general sense the notion of a *cover-edge* states an upper bound constraint for the *cost* of some sub-property to be true, where cost is encoded as expression weights on the hyper-edges.

▶ **Definition 5.** A *Parametric Dependency Graph* (PDG) is a tuple $\mathcal{G} = (N, H, C)$, where:

- N is a finite set of *nodes*,
- $H \subseteq N \times 2^{\mathcal{E} \times N}$ is a finite set of *hyper-edges*.
- $C \subseteq N \times \mathcal{E} \times N$ is a finite set of *cover-edges*.

Whenever $(n, T) \in H$ we refer to n as the *source* node and T as the *target-set*. For each $n' \in T$ we refer to n' as a *target* node, or simply *target*. We will use $n \overset{e}{\dashrightarrow} n'$ whenever $(n, e, n') \in C$.

The set of all PDGs will be denoted by $\mathfrak{G}$. We will now proceed to define assignments which are used to encode the parametric cost for reaching a truth value in PDGs and note that the assignments form a complete lattice.

▶ **Definition 6.** Given a PDG $\mathcal{G} = (N, H, C)$, an assignment

$$A : N \longrightarrow (\mathfrak{I} \longrightarrow \mathbb{N} \cup \{\infty\})$$

on $\mathcal{G}$ is a mapping from each node $n \in N$ to a function that, given a parameter interpretation, yields a natural number or ∞.

We denote the set of all assignments $\mathfrak{A}$. The partial ordering over $\mathfrak{A}$ is defined as follows:

▶ **Definition 7.** $(\mathfrak{A}, \sqsubseteq)$ is a poset such that for $A_1, A_2 \in \mathfrak{A}$:

$$A_1 \sqsubseteq A_2 \quad \text{iff} \quad \forall n \in N \, \forall \boldsymbol{i} \in \mathfrak{I} : A_1(n)(\boldsymbol{i}) \geq A_2(n)(\boldsymbol{i})$$

$(\mathfrak{A}, \sqsubseteq)$ is clearly a complete lattice with A^0 and A^∞ as top and bottom element, respectively.

Let A^0 denote the assignment that maps to each node a function that assigns the value 0 regardless of parameter interpretations, i.e. $\forall n \in N \, \forall \boldsymbol{i} \in \mathfrak{I} : A^0(n)(\boldsymbol{i}) = 0$. Similarly A^∞ denotes the assignment that maps to each node a function that assigns the value ∞ regardless of parameter interpretations.

Generally when computing assignments, we use ∞ to represent negative results (infinite cost) and 0 (no cost) to represent positive results. As usual, when computing a minimal fixed point, we start from the bottom element, A^∞ and similarly from the top element, A^0, for maximal fixed points.

We are now ready to define the global update function, which applied iteratively, updates PDG node assignments. Note that this function only considers PDG where for any node there is at most one outgoing *cover*-edge. This is not a limitation for this work as we only consider problems where one *cover*-edge is sufficient and one can easily extend the function to consider multiple *cover*-edges while preserving all results.

▶ **Definition 8.** Given a PDG $\mathcal{G} = (N, H, C)$, $\boldsymbol{F} : \mathfrak{A} \longrightarrow \mathfrak{A}$ is a function that, given an assignment A on $\mathcal{G}$, produces a new assignment on $\mathcal{G}$ for any $n \in N, \boldsymbol{i} \in \mathfrak{I}$, as follows:

$$\boldsymbol{F}(A)(n)(\boldsymbol{i}) = \begin{cases} \begin{cases} 0 & \text{if } A(n')(\boldsymbol{i}) \leq \boldsymbol{i}(e) \\ \infty & \text{otherwise} \end{cases} & \text{if } n \overset{e}{\dashrightarrow} n' \\ \\ \min\limits_{(n,T)\in H} \{ \max\limits_{(e,n')\in T} \{A(n')(\boldsymbol{i}) + \boldsymbol{i}(e)\}\} & \text{otherwise} \end{cases}$$

We use $\boldsymbol{F}^i(A)$ to denote i repeated applications of the function $\boldsymbol{F}$ on A, i.e
$\boldsymbol{F}^i(A) = \boldsymbol{F}(\boldsymbol{F}^{i-1}(\ldots \boldsymbol{F}^1(A)))$ for $i > 0$ and $\boldsymbol{F}^0(A) = A$.

To show that $\boldsymbol{F}$ has a minimal fixed point, we observe by case inspection of Definition 8 that $\boldsymbol{F}$ is monotone with respect to the complete lattice $(\mathfrak{A}, \sqsubseteq)$ as stated by the following lemma.

▶ **Lemma 9.** *The update function $\boldsymbol{F}$ is monotone on the complete lattice $(\mathfrak{A}, \sqsubseteq)$.*

By Tarski's Fixed Point Theorem, we can conclude that $\boldsymbol{F}$ has a minimal and maximal fixed point. The assignment corresponding to these fixed points are denoted by A^{min} and A^{max}. We now proceed to show that A^{min} corresponds to $\boldsymbol{F}^i(A)$ for some $i \in \mathbb{N}$, ensuring that the minimal fixed point is computable in a finite number of steps. The key to this is the notion of well-quasi orders [14].

In general, assignments are functions that, given a node and an interpretation, compute a number. As our function is monotone w.r.t. $\sqsubseteq$ we know that the sequence of assignments computed by our function, given an interpretation, is decreasing for any node. We can therefore pick any interpretation and interpret an assignment A as a tuple $(x_0, x_1, \ldots, x_k)$ where $k+1$ is the number of nodes in the PDG and $x_i \in \mathbb{N} \cup \{\infty\}$ for all $0 \leq i \leq k$. Each iteration can then be interpreted as a function computing the next tuple in a (possibly) infinite sequence of tuples. Let the set of all such tuples be denoted by $\mathfrak{A}^T$ and let $A_i^T \in \mathfrak{A}^T$ denote the tuple computed by the i'th iteration and $\leq$ the component-wise ordering of tuples in $\mathfrak{A}^T$.

As $((\mathbb{N} \cup \{\infty\}), \leq)$ is a well-quasi order, we can use [17] (Theorem 2.3) saying that the Cartesian product of a finite number of well-quasi-ordered spaces is well-quasi-ordered, to state the following lemma.

▶ **Lemma 10.** *$(\mathfrak{A}^T, \leq)$ is a well-quasi-order.*

By the well quasi-ordering of $(\mathfrak{A}^T, \leq)$, we can now state that the fixed point computation ends in a finite number of steps, when applied iteratively on the bottom element A^∞.

▶ **Theorem 11.** *There exists a natural number i such that $A^{min} = \boldsymbol{F}^i(A^\infty)$.*

A similar result for the maximal fixed point does not exist. Consider a PDG consisting of only one node with a non-zero self loop. In this case, the next assignment is the old one plus the loop expression weight, thereby never reaching a fixed point.

▶ **Lemma 12.** *There exists a PDG such that $A^{max} \neq \boldsymbol{F}^i(A^0)$ for any $i \in \mathbb{N}$.*

3.2 Parametric Dependency Graphs For Model Checking

The verification of PTL properties differs from the usual notion of model checking in that it may not always be possible to give a Boolean answer. Instead we seek constraints on parameter interpretations that makes a PTL formula satisfiable by a given PTS state.

The construction rules in Figure 2 define the PDG construction for model checking.

As PDGs in this context are used to encode constraints on satisfiability of PTL formulae, they are constructed over a PTS model $\mathcal{M}$, a state m in the model and a PTL formula Φ. Each node $n \in N$ of a PDG $\mathcal{G}$ will be a pair $\langle m, \Phi \rangle$, where Φ is interpreted as a formula that may be satisfied in state m. A node n therefore represents $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi$. We notice that the problem may depend on sub-formulae of Φ and successors of m. We therefore use hyper-edges to connect the root $\langle m, \Phi \rangle$ to nodes representing these dependencies. For PTL path formulae we encode the parametric upper bounds as cover-edges in the PDG.

It should be clear from the PDG construction that by applying the update function $\boldsymbol{F}$ on an assignment to the PDG, any node that has an outgoing hyper-edge with an empty target set, i.e. nodes representing $\mathcal{M}, m \models_{\boldsymbol{i}} \top$ or $\mathcal{M}, m \models_{\boldsymbol{i}} \mathtt{a}$ where $\mathtt{a} \in \boldsymbol{\ell}(m)$, gets the value $\min\{\max\{\emptyset\}\} = 0$ thus representing satisfiability whereas nodes with no outgoing edges gets the value $\min\{\emptyset\} = \infty$ thus representing non-satisfiability. This shows the intuition behind the semantics of the values, that 0 represents satisfiability and ∞ represents non-satisfiability. In the context of model checking we compute a minimal fixed point, meaning that we start from the bottom element, A^{∞}.

We now state the correctness theorem of the developed model-checking algorithm. The proof follows the correctness proof in [19], as models and formulae can be converted to a non-parametric setting using an interpretation. We denote by $\boldsymbol{B}(\mathcal{M}, m, \Phi)$ the PDG constructed by the rules in Figure 2 for the model checking problem $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi$.

▶ **Theorem 13** (Correctness of Model-Checking Algorithm). *Let $\mathcal{M} = (M, \rightarrow, \ell)$ be a PTS, $m \in M$ a state, $\boldsymbol{i} \in \mathfrak{I}$ an interpretation and let $\mathcal{G} = \boldsymbol{B}(\mathcal{M}, m, \Phi) = (N, H, C)$ where $\langle m, \Phi \rangle \in N$. Then*

$$\mathcal{M}, m \models_{\boldsymbol{i}} \Phi \textit{ iff } A^{min}(\langle m, \Phi \rangle)(\boldsymbol{i}) = 0$$

The following example shows the construction of a PDG given a PTL formula and a PTS model and the application of Theorem 13 to derive parameter constraints.

▶ **Example 14.** Given the PTS $\mathcal{M}$ (Figure 3) and the formula

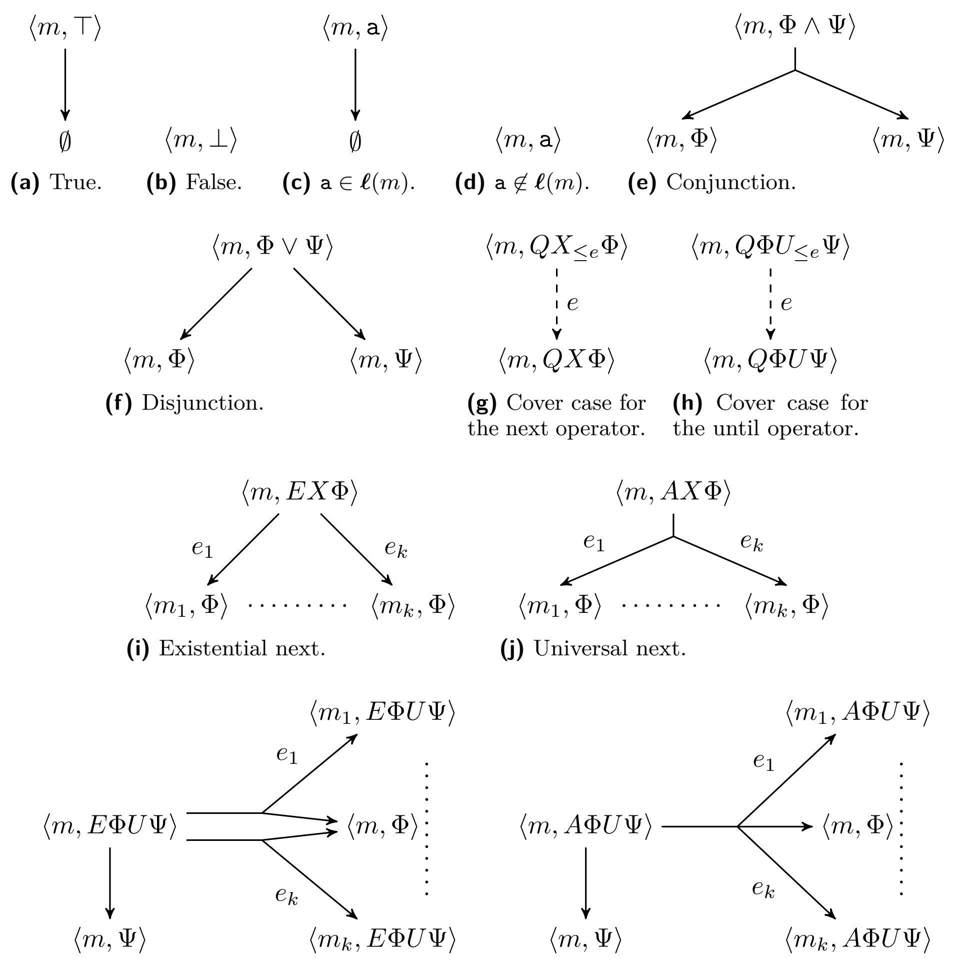
$$\Phi_{ex} = E\mathtt{b}U_{\leq 5}(\mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}),$$

we now apply the construction rules in Figure 2 to encode the query $\mathcal{M}, m \models_{\boldsymbol{i}} \Phi_{ex}$ as the PDG $\mathcal{G}$ in Figure 4.

By repeated application of the monotone function $\boldsymbol{F}$, we arrive at a fixed point after 7 iterations. Using the correctness theorem (and a few trivial simplifications) we get the exact constraints $\boldsymbol{i}(p) \leq 7 + \boldsymbol{i}(q)$ and $\boldsymbol{i}(q) \leq 5$ for $\boldsymbol{i}$ to be a valid interpretation. A quick look at the PTS should convince the reader of the correctness of these constraints.

4 Prototype Tool (PVTool)

We now present a proof of concept tool (PVTool [11]), being entirely web-based, for verification of parametric properties using the technique developed in this work. In Section 1, Figure 1 we presented an example of a parametric system which depicts a robotic vacuum cleaner. We now want to verify whether or not the cleaner can move from the starting location to


$\langle m, \top \rangle$
$\emptyset$
$\langle m, \bot \rangle$
$\langle m, \mathtt{a} \rangle$
$\emptyset$
$\langle m, \mathtt{a} \rangle$
$\langle m, \Phi \wedge \Psi \rangle$
$\langle m, \Phi \rangle$
$\langle m, \Psi \rangle$
$\langle m, \Phi \vee \Psi \rangle$
$\langle m, \Phi \rangle$
$\langle m, \Psi \rangle$
$\langle m, QX_{\leq e}\Phi \rangle$
e
$\langle m, QX\Phi \rangle$
$\langle m, Q\Phi U_{\leq e}\Psi \rangle$
e
$\langle m, Q\Phi U\Psi \rangle$
$\langle m, EX\Phi \rangle$
e_1
e_k
$\langle m_1, \Phi \rangle$
$\langle m_k, \Phi \rangle$
$\langle m, AX\Phi \rangle$
e_1
e_k
$\langle m_1, \Phi \rangle$
$\langle m_k, \Phi \rangle$
$\langle m_1, E\Phi U\Psi \rangle$
e_1
$\langle m, E\Phi U\Psi \rangle$
$\langle m, \Phi \rangle$
e_k
$\langle m, \Psi \rangle$
$\langle m_k, E\Phi U\Psi \rangle$
$\langle m_1, A\Phi U\Psi \rangle$
e_1
$\langle m, A\Phi U\Psi \rangle$
$\langle m, \Phi \rangle$
e_k
$\langle m, \Psi \rangle$
$\langle m_k, A\Phi U\Psi \rangle$

(a) True. **(b)** False. **(c)** $\mathtt{a} \in \ell(m)$. **(d)** $\mathtt{a} \notin \ell(m)$. **(e)** Conjunction.

(f) Disjunction. **(g)** Cover case for the next operator. **(h)** Cover case for the until operator.

(i) Existential next. **(j)** Universal next.

(k) Existential until. **(l)** Universal until.

Figure 2 Let $\{(e_1, m_1), (e_2, m_2) \ldots (e_k, m_k)\} \in \boldsymbol{out}(m)$ and $Q \in \{A, E\}$.

one of the next rooms within 10 minutes units and clean it within 20 minutes. This question can be stated as follows:

$$\mathcal{M}_{ex}, charger1 \models_{\boldsymbol{i}} EX_{\leq 10}[A\ \mathtt{dirty}\ U_{\leq 20}\ \mathtt{clean}]$$

In the online PVTool we can now construct a model of the vacuum cleaner problem and state the PTL query and in return get the parameter constraints for the property to be satisfied in the model as depicted in Figure 5. We will briefly explain the model syntax used in the tool. The line "`charger1 := {clean, ready} <p>room1 + <2>room2;`" declares a state with the symbolic name *charger1* and two atomic propositions: *clean* and *ready*. The state has two outgoing transitions; one going to the state *room1* with the weight p and one going to the state *room2* with the weight 2. Subsequent lines declare states in a similar manner. The formulae syntax is almost identical to the syntax used in the paper, with the exceptions that all sub-formulae must be enclosed in square brackets, bounds on path formulae must be enclosed on curly brackets and we do not write the $\leq$ sign. A thorough explanation of the syntax can be found on PVTool's web page.

To examine performance of the implementation we experiment with models that have a variable number of states. All experiments use models with a structure similar to the

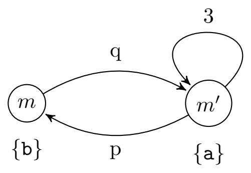


Figure 3 PTS $\mathcal{M}$. $\boldsymbol{\ell}(m) = b, \boldsymbol{\ell}(m') = a$.

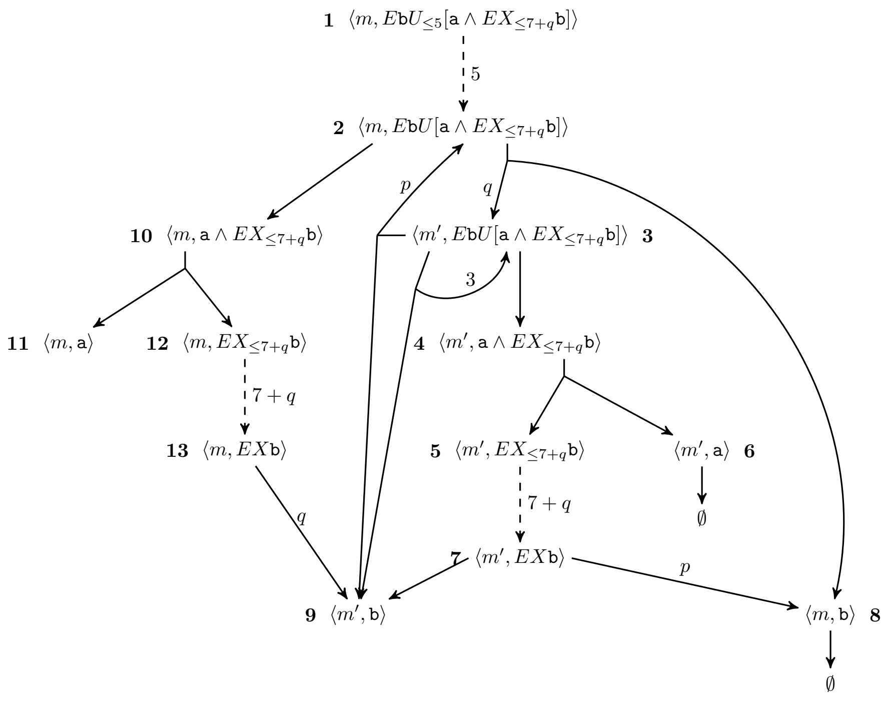

1 $\langle m, E\mathtt{b}U_{\leq 5}[\mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}]\rangle$
5
2 $\langle m, E\mathtt{b}U[\mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}]\rangle$
p
q
10 $\langle m, \mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}\rangle$
$\langle m', E\mathtt{b}U[\mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}]\rangle$ 3
3
11 $\langle m, \mathtt{a}\rangle$
12 $\langle m, EX_{\leq 7+q}\mathtt{b}\rangle$
4 $\langle m', \mathtt{a} \wedge EX_{\leq 7+q}\mathtt{b}\rangle$
7 + q
13 $\langle m, EX\mathtt{b}\rangle$
5 $\langle m', EX_{\leq 7+q}\mathtt{b}\rangle$
$\langle m', \mathtt{a}\rangle$ 6
q
7 + q
$\emptyset$
7 $\langle m', EX\mathtt{b}\rangle$
p
9 $\langle m', \mathtt{b}\rangle$
$\langle m, \mathtt{b}\rangle$ 8
$\emptyset$

Figure 4 PDG $\mathcal{G}$.

model depicted in Figure 1. We simply change the amount of rooms and intermediate "clean" states. For all experiments we use the PTL query

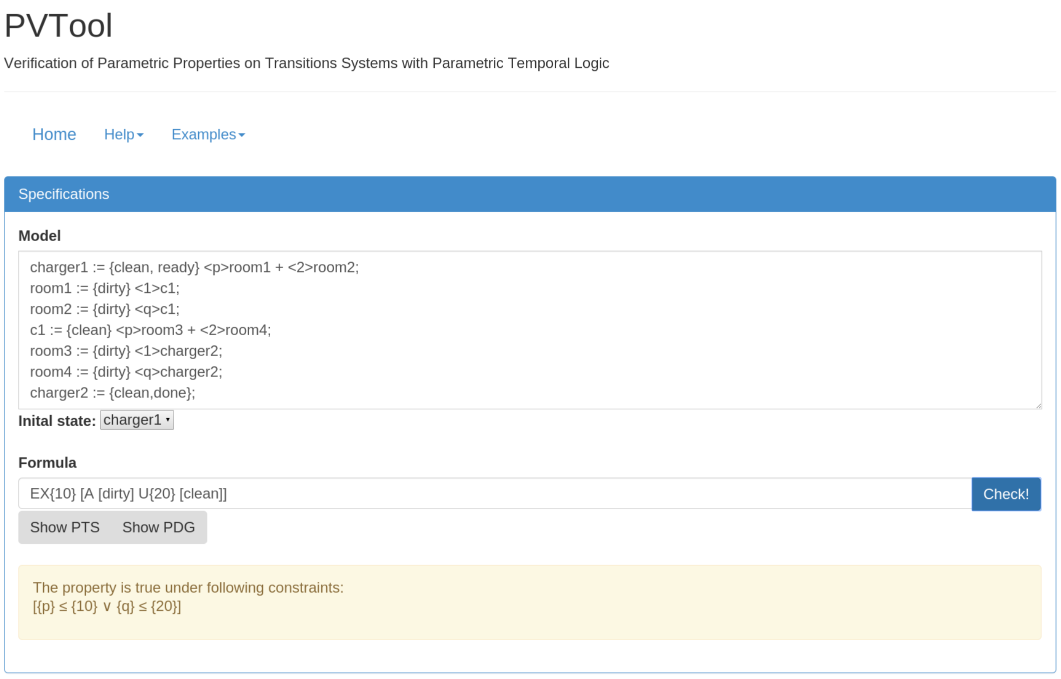
$$\mathcal{M}_{ex}, charger1 \models_{\boldsymbol{i}} E\ [A\ \mathtt{dirty}\ U_{\leq s}\ \mathtt{clean}]\ U_{\leq r}\ \mathtt{done}$$

Refer to Figure 6 to see the performance results. Memory and time consumption increase exponentially in the number of PTS states, reflecting the exponential increase in number of possible paths when scaling the PTS.

5 Conclusion and Future Work

We have defined a variant of Weighted CTL which uses parametric linear expressions as upper bounds on transition weights to allow reasoning about unknown behavior. We call this logic Parametric Temporal Logic (PTL). The semantics of PTL is defined for Parametric Weighted Transition Systems (PTS) that allow parametric linear expressions as weights on transitions.

For encoding of parametric propositional dependencies we present Parametric Dependency Graphs (PDGs). We associate to each node in a PDG a parametric cost and demonstrate


PVTool

Verification of Parametric Properties on Transitions Systems with Parametric Temporal Logic

Model

```
charger1 := {clean, ready} <p>room1 + <2>room2;
room1 := {dirty} <1>c1;
room2 := {dirty} <q>c1;
c1 := {clean} <p>room3 + <2>room4;
room3 := {dirty} <1>charger2;
room4 := {dirty} <q>charger2;
charger2 := {clean,done};
```

Inital state: charger1

Formula

```
EX{10} [A [dirty] U{20} [clean]]
```

The property is true under following constraints:
[{p} ≤ {10} ∨ {q} ≤ {20}]

Figure 5 Screen shot of verification of PTL property on the model in Figure 1 using PVTool.

PTS sts	PDG nds	Iter.	Mem (kB)	Time (s)
7	41	9	1,004	0.0015
13	77	13	1,504	0.017
19	113	17	3,808	0.19
25	149	21	14,264	2.5
31	185	25	60,548	35
34	221	29	263,832	781

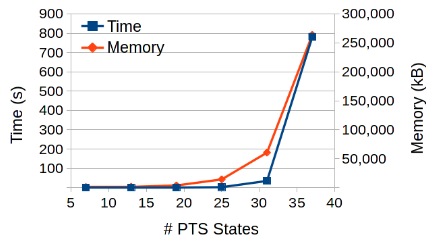


Figure 6 Experiments on a 4 core Intel Xeon E3-1245 v2 3.4 GHz processor with 16 GB RAM.

that a maximal and minimal fixed point on these costs exists. We also show that the minimal fixed point can always be computed in a finite number of steps. Furthermore we show that deciding model checking PTS with PTL properties corresponds to finding the minimal fixed point in a PDG abstraction of model checking problems.

For verification of PTL properties on PTS we have implemented a model checking tool, PVTool ([11]) - available online, in which it is possible to define a PTS, give a PTL formula and get as output the constraints on parameters for a state in the PTS to satisfy the formula. In this context, preliminary experiments were made using an easily scalable PTS model to assess memory and time consumption which scaled exponentially in the size of the PTS, as expected.

As extension on this work, we propose to further look into other methods for updates on assignments to PDG nodes. In [19], the authors present a local fixed point algorithm for dependency graphs. If something similar can be done in the parametric setting we expect significant performance improvements. It would also be interesting to investigate whether a complete characterisation of PDGs for which the maximal fixed point can be computed in a finite number of steps, can be made. Finally one could look for new or existing problem domains where our method can be applied. A first step in this direction was done in the draft [10], for a variant of bisimilarity checking.

References

1 Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Logic in Computer Science, 1996. LICS'96. Proceedings., Eleventh Annual IEEE Symposium on*, pages 313–321. IEEE, 1996.

2 Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.

3 Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *ICALP*, pages 322–335, 1990.

4 Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 592–601, 1993.

5 Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In *HSCC*, pages 49–62, 2001.

6 Étienne André, Thomas Chatain, Laurent Fribourg, and Emmanuelle Encrenaz. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(05):819–836, 2009.

7 Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *HSCC*, pages 147–161, 2001.

8 Sine Viesmose Birch, Peter Christoffersen, Mikkel Hansen, Anders Mariegaard, and Julian T. Ringsmose. Weighted transition system: From boolean to parametric analysis. 8th semester project at Aalborg University, Department of Computer Science, 2014.

9 Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jirí Srba. Infinite runs in weighted timed automata with energy constraints. In *FORMATS 2008. Proceedings*, pages 33–47, 2008.

10 Peter Christoffersen, Mikkel Hansen, Anders Mariegaard, and Julian T. Ringsmose. Parametric verification of weighted systems. Unpublished Technical Report. `http://pvtool.dk/tech_draft.pdf`.

11 Peter Christoffersen, Mikkel Hansen, Anders Mariegaard, and Julian T. Ringsmose. Prototype tool. `http://pvtool.dk/`.

12 Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample guided abstraction refinement. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, pages 154–169, 2000.

13 Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):pp. 413–422, 1913.

14 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1):63–92, 2001.

15 Goran Frehse, Sumit Kumar Jha, and Bruce H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In Magnus Egerstedt and Bud Mishra, editors, *Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 187–200. Springer Berlin Heidelberg, 2008.

16 Thomas A. Henzinger and Howard Wong-Toi. Using hytech to synthesize control parameters for a steam boiler. In Jean-Raymond Abrial, Egon Börger, and Hans Langmaack, editors, *Formal Methods for Industrial Applications*, volume 1165 of *Lecture Notes in Computer Science*, pages 265–282. Springer Berlin Heidelberg, 1996.

17 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(1):326–336, 1952.

18 Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *J. Log. Algebr. Program.*, 52-53:183–220, 2002.

19 Jonas Finnemann Jensen, Kim Guldstrand Larsen, Jiří Srba, and Lars Kaerlund Oestergaard. Local model checking of weighted CTL with upper-bound constraints. In *Model Checking Software*, pages 178–195. Springer, 2013.

20 Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

21 Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

22 Xinxin Liu and Scott A. Smolka. Simple linear-time algorithms for minimal fixed points. In *Automata, Languages and Programming*, pages 53–66. Springer, 1998.

23 Chaiwat Sathawornwichit and Takuya Katayama. A parametric model checking approach for real-time systems design. In *Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific*, pages 8–pp. IEEE, 2005.

24 Alfred Tarski et al. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2):285–309, 1955.

A Fixed point computation results

In this appendix we show the fixed point computation from Example 14 in Table 1. For readability we have taken the liberty of simplifying the assignments based on properties of expressions and min / max functions valid under any interpretation of parameters. If any assignment to a node does not change (modulus our simplification rules) during the entire computation, we have omitted the node from the table. Let $A(n)$ be the assignment to some node n and $e \in \mathcal{E}$ an expression.

- $\max\{\emptyset\} = \min\{0, \ldots\} = 0$
- $\min\{\emptyset\} = \max\{\infty, \ldots\} = \infty + A(n) = e + \infty = \infty$
- $0 + A(n) = A(n)$
- $e + 0 = e$
- Min/max of a single element is the element itself

Finally, let $\boldsymbol{F}^i(n)$ be a shorthand for $\boldsymbol{F}^i(A^\infty)(n)$. Using the correctness theorem we can now derive constraints that must be satisfied by an interpretation $\boldsymbol{i}$ for $\mathcal{M}, m \models_{\boldsymbol{i}} E\mathtt{b}U_{\leq 5}[\mathtt{a} \land EX_{\leq 7+\mathtt{q}}\mathtt{b}]$ to be true. From our simplified assignments it is easy to see that the constraints for $\boldsymbol{F}^7(\mathbf{1})(\boldsymbol{i}) = \boldsymbol{F}^8(\mathbf{1})(\boldsymbol{i}) = 0$ given some interpretation $\boldsymbol{i}$ must be $\boldsymbol{i}(p) \leq 7 + \boldsymbol{i}(q)$ and $\boldsymbol{i}(q) \leq 5$.

Table 1 Updating assignments.

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$A^\infty(n)(\boldsymbol{i})$	∞	∞	∞	∞	∞	∞	∞	∞
$\boldsymbol{F}^1(n)(\boldsymbol{i})$	∞	∞	∞	∞	∞	0	∞	0
$\boldsymbol{F}^2(n)(\boldsymbol{i})$	∞	∞	∞	∞	∞	0	$\boldsymbol{i}(p)$	0
$\boldsymbol{F}^3(n)(\boldsymbol{i})$	∞	∞	∞	∞	$\begin{cases} 0 & \text{if } \boldsymbol{i}(p) \leq 7 + \boldsymbol{i}(q) \\ \infty & \text{otherwise} \end{cases}$	0	$\boldsymbol{i}(p)$	0
$\boldsymbol{F}^4(n)(\boldsymbol{i})$	∞	∞	∞	$\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$	$\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$	0	$\boldsymbol{i}(p)$	0
$\boldsymbol{F}^5(n)(\boldsymbol{i})$	∞	∞	$\boldsymbol{F}^4(4)(\boldsymbol{i})$	$\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$	$\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$	0	$\boldsymbol{i}(p)$	0
$\boldsymbol{F}^6(n)(\boldsymbol{i})$	∞	$\boldsymbol{i}(q) + \boldsymbol{F}^5(\mathbf{3})(\boldsymbol{i})$	$\boldsymbol{F}^4(4)(\boldsymbol{i})$	$\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$	$\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$	0	$\boldsymbol{i}(p)$	0
$\boldsymbol{F}^7(n)(\boldsymbol{i})$	$\begin{cases} 0 & \text{if } \boldsymbol{F}^6(\mathbf{2})(\boldsymbol{i}) \leq 5 \\ \infty & \text{otherwise} \end{cases}$	$\boldsymbol{i}(q) + \boldsymbol{F}^5(\mathbf{3})(\boldsymbol{i})$	$\boldsymbol{F}^4(4)(\boldsymbol{i})$	$\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$	$\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$	0	$\boldsymbol{i}(p)$	0
$\boldsymbol{F}^8(n)(\boldsymbol{i})$	$\begin{cases} 0 & \text{if } \boldsymbol{F}^6(\mathbf{2})(\boldsymbol{i}) \leq 5 \\ \infty & \text{otherwise} \end{cases}$	$\boldsymbol{i}(q) + \boldsymbol{F}^5(\mathbf{3})(\boldsymbol{i})$	$\boldsymbol{F}^4(4)(\boldsymbol{i})$	$\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$	$\boldsymbol{F}^3(\mathbf{5})(\boldsymbol{i})$	0	$\boldsymbol{i}(p)$	0

Tuning PI controller in non-linear uncertain closed-loop systems with interval analysis[*]

Julien Alexandre dit Sandretto, Alexandre Chapoutot, and Olivier Mullier

Unité d'Informatique et d'Ingénierie des Systèmes,
ENSTA ParisTech, Université Paris-Saclay,
828 bd des Maréchaux, 91762 Palaiseau cedex France
`alexandre@ensta.fr`, `chapoutot@ensta.fr`, `mullier@ensta.fr`

Abstract

The tuning of a PI controller is usually done through simulation, except for few classes of problems, e.g., linear systems. With a new approach for validated integration allowing us to simulate dynamical systems with uncertain parameters, we are able to design guaranteed PI controllers. In practical, we propose a new method to identify the parameters of a PI controller for non-linear plants with bounded uncertain parameters using tools from interval analysis and validated simulation. This work relies on interval computation and guaranteed numerical integration of ordinary differential equations based on Runge-Kutta methods. Our method is applied to the well-known cruise-control problem, under a simplified linear version and with the aerodynamic force taken into account leading to a non-linear formulation.

1998 ACM Subject Classification B.5.2 Simulation; I.6.6 Simulation Output Analysis; G.1.7 Initial Value Problem; G.1.0 Interval Arithmetic

Keywords and phrases PID Tuning, huaranteed numerical integration, non-linear ordinary differential equations

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.91

1 Introduction

Recently [2], we developed a new tool for validated simulation [17, 6, 14, 15] of Ordinary Differential Equations (ODE). This tool, by using affine arithmetic [9], is able to handle ODEs with uncertain (and bounded) parameters. This new capability allows us to simulate a dynamical system controlled by a proportional integral (PI), whose parameters K_p (proportional gain), K_i (integral gain) are not well-known. The direct advantage is to use the simulation process to validate or reject some parameter values. This approach is, in philosophy, similar to Ziegler–Nichols approach [20]. Another method, the model of Broida [7], is also based on identification of parameters but works only for a linear problem in open loop. All the existing methods are empirical and do not provide any guarantee on the behavior of the system, and cannot consider uncertainties on the physical part. Interval analysis is often used in applications like robust control [13]. Indeed, the methods provided by the interval formalization are able to consider any kind of bounded uncertainties, manage with non-linear models and offer a guarantee on the numerical computation.

[*] This research benefited from the support of the «Chair Complex Systems Engineering - Ecole Polytechnique, THALES, DGA, FX, DASSAULT AVIATION, DCNS Research, ENSTA ParisTech, Télécom ParisTech, Fondation ParisTech and FDO ENSTA».

2nd International Workshop on Synthesis of Complex Parameters (SynCoP'15).
Editors: Étienne André and Goran Frehse; pp. 91–102
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Until now, to the best of our knowledge, interval analysis for robust control only considers linear systems [19, 4]. Indeed for this class of problems, robust control problem is cast into a problem of stability and performance based on the characteristic polynomial associated to the transfer function of the closed-loop system. Unfortunately, for non-linear closed-loop systems this approach is no longer possible.

The paper is organized as followed. Main interval analysis tools are recalled in Section 2. The tuning algorithm is presented in Section 3 including also a remainder of the PI controller theory. Some experimental results are presented in Section 4 before concluding in Section 5.

2 Interval analysis tools

We recall in this section the main tools coming from interval analysis and particularly affine arithmetic to tighten the issue of interval arithmetic used in our work.

2.1 Interval arithmetic

The simplest and most common way to represent and manipulate sets of values is *interval arithmetic* [16]. An interval $[x_i] = [\underline{x_i}, \overline{x_i}]$ defines the set of reals x_i such that $\underline{x_i} \leq x_i \leq \overline{x_i}$. $\mathbb{IR}$ denotes the set of all intervals. The size or the width of $[x_i]$ is denoted by $w([x_i]) = \overline{x_i} - \underline{x_i}$.

Interval arithmetic [16] extends to $\mathbb{IR}$ elementary functions over $\mathbb{R}$. For instance, the interval sum (i.e., $[x_1] + [x_2] = [\underline{x_1} + \underline{x_2}, \overline{x_1} + \overline{x_2}]$) encloses the image of the sum function over its arguments, and this enclosing property basically defines what is called an *interval extension* or an *inclusion function*.

▶ **Definition 1** (Extension of a function to $\mathbb{IR}$)**.** Consider a function $f : \mathbb{R}^n \to \mathbb{R}$, then $[f] : \mathbb{IR}^n \to \mathbb{IR}$ is said to be an **extension** of f to intervals if

$$\forall [x] \in \mathbb{IR}^n, \quad [f]([x]) \supseteq \{f(x),\ x \in [x]\} \ .$$

2.2 Affine arithmetic

Interval arithmetic provides a good solution to manage with uncertainties. Nevertheless, this representation usually produces too much over-approximated results in particular because of the *dependency problem*. An iterative scheme, such as a mathematical series or an integration scheme, leads typically to a dependency problem: each step depends on the previous ones.

▶ **Example 2.** Consider the ordinary differential equation $\dot{x}(t) = -x$ solved with the Euler's method with an initial value ranging in the interval $[0, 1]$ and with a step-size of $h = 0.5$. For one step of integration, we have to compute with interval arithmetic the expression $e = x + h \times (-x)$ which produces as a result the interval $[-0.5, 1]$. Rewriting the expression e such that $e' = x(1 - h)$, we obtain the interval $[0, 0.5]$ which is the exact result. Unfortunately, we cannot in general rewrite expressions with only one occurrence of each variable. More generally, it can be shown that for most integration schemes the width of the result can only grow if we interpret sets of values as intervals [18]. ■

To avoid this problem we use an improvement over interval arithmetic named *affine arithmetic* [9] which can track linear correlation between program variables. A set of values is represented by an *affine form* $\hat{x}$, i.e., a formal expression of the form $\hat{x} = \alpha_0 + \sum_{i=1}^{n} \alpha_i \varepsilon_i$ where the coefficients α_i are real numbers, α_0 being called the *center* of the affine form, α_i, $i \geqslant 1$, are called *partial deviations* and the ε_i, called *noise symbols*, are independent components of the total uncertainty on $\hat{x}$ with unknown values ranging over the interval

$[-1,1]$. Obviously, an interval $a = [a_1, a_2]$ can be seen as the affine form $\hat{x} = \alpha_0 + \alpha_1\varepsilon$ with $\alpha_0 = (a_1 + a_2)/2$ and $\alpha_1 = (a_2 - a_1)/2$.

Affine arithmetic extends usual operations on real numbers in the expected way. For instance, the affine combination of two affine forms $\hat{x} = \alpha_0 + \sum_{i=1}^{n} \alpha_i\varepsilon_i$ and $\hat{y} = \beta_0 + \sum_{i=1}^{n} \beta_i\varepsilon_i$ with $a, b, c \in \mathbb{R}$, is an affine form given by

$$a\hat{x} \pm b\hat{y} + c \quad = \quad (a\alpha_0 \pm b\beta_0 + c) + \sum_{i=1}^{n}(a\alpha_i \pm b\beta_i)\varepsilon_i \ . \tag{1}$$

However, unlike the addition, non linear operations create new noise symbols. Multiplication for example can be defined by

$$\hat{x} \times \hat{y} \quad = \quad \alpha_0\alpha_1 + \sum_{i=1}^{n}(\alpha_i\beta_0 + \alpha_0\beta_i)\varepsilon_i + \nu\varepsilon_{n+1} \quad \text{with} \quad \nu = \left(\sum_{i=1}^{n}|\alpha_i|\right) \times \left(\sum_{i=1}^{n}|\beta_i|\right) \ . \tag{2}$$

Operations as sin or exp are translated into affine forms with Chebyshev polynomials [9]. For practical details on soundness w.r.t. floating-point computations and the performance of this arithmetic w.r.t. the number of noise symbols see [5].

▶ **Example 3.** Consider again $e = x + h \times (-x)$ with $h = 0.5$ and $x = [0, 1]$ which is associated to the affine form $\hat{x} = 0.5 + 0.5\varepsilon_1$. Evaluating e with affine arithmetic without rewriting the expression, we obtain $[0, 0.5]$ as a result. ■

Note that the set-based evaluation of an expression only consists in substituting all the mathematical operators, like $+$ or sin, by their counterpart in affine arithmetic. We denote by $\text{Aff}(e)$ the evaluation of the expression e using affine arithmetic.

2.3 Guaranteed numerical integration with Runge-Kutta methods

In this section, we recall our previous work [5] on which the extension in [2] is based on.

▶ **Definition 4** (Initial Value Problem (IVP))**.** Consider an Ordinary Differential Equation (ODE) with a given initial condition

$$\dot{y}(t) = f\left(t, y(t), d\right) \quad \text{with} \quad y(0) \in Y_0, \tag{3}$$

with $f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ assumed to be continuous in t and d and globally Lipschitz in y. We assume that parameters d are constant and bounded. An IVP consists in finding a function $y(t)$ described by the ODE for all d and satisfying the initial condition.

A numerical integration method computes a sequence of approximations (t_n, y_n) of the solution $y(t; y_0)$ of the IVP defined in Equation (3) such that $y_n \approx y(t_n; y_{n-1})$.

The simplest method is Euler's method in which $t_{i+1} = t_i + h$ for some step-size h and $y_{i+1} = y_i + h \times f(t_i, y_i, d)$; so the derivative of y at time t_i, $f(t_i, y_i, d)$, is used as an approximation of the derivative on the whole time interval to perform a linear interpolation. This method is very simple and fast, but requires small step-sizes. More advanced methods coming from the Runge-Kutta family use a few intermediate computations to improve the approximation of the derivative. The general form of an explicit s-stage Runge-Kutta formula, that is using s evaluations of f, is

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \ , \tag{4a}$$

$$k_1 = f\big(t_n,\, y_n, d\big) \ , \qquad k_i = f\Big(t_n + c_i h,\, y_n + h\sum_{j=1}^{i-1} a_{ij}k_j, d\Big) \ , \quad i = 2, 3, \dots, s \ . \tag{4b}$$

The coefficients c_i, a_{ij} and b_i fully characterize the method. To make Runge-Kutta validated, the challenging question is how to compute a bound on the distance between the true solution and the numerical solution, defined by $y(t_n; y_{n-1}) - y_n$. This distance is associated to the *local truncation error* (LTE) of the numerical method.

To bound the LTE, we rely on *order condition* [12] respected by all Runge-Kutta methods. This condition states that a method of this family is of order p iff the $p+1$ first coefficients of the Taylor expansion of the solution and the Taylor expansion of the numerical methods are equal. In consequence, LTE is proportional to the Lagrange remainders of Taylor expansions. In previous work [5], LTE is defined by

$$y(t_n; y_{n-1}) - y_n = \frac{h^{p+1}}{(p+1)!}\left(f^{(p)}\left(\xi, y(\xi; y_{n-1}, d)\right) - \frac{d^{p+1}\phi}{dt^{p+1}}(\eta)\right)$$
$$\xi \in]t_k, t_{k+1}[\text{ and } \eta \in]t_n, t_{n+1}[\ . \quad (5)$$

The function $f^{(n)}$ stands for the n-th derivative of function f w.r.t. time t that is $\frac{d^n f}{dt^n}$ and $h = t_{n+1} - t_n$ is the step-size. The function $\phi : \mathbb{R} \to \mathbb{R}^n$ is defined by $\phi(t) = y_n + h \sum_{i=1}^{s} b_i k_i(t)$ where $k_i(t)$ are defined as Equation (4b).

The challenge to make Runge-Kutta integration schemes safe w.r.t. the true solution of IVP is then to compute a bound of the result of Equation (5). In other words we have to bound the value of $f^{(p)}\left(\xi, y(\xi; y_{n-1}), d\right)$ and the value of $\frac{d^{p+1}\phi}{dt^{p+1}}(\eta)$. The latter expression is straightforward to bound because the function ϕ only depends on the value of the step-size h, and so does its $(p+1)$-th derivative. The bound is then obtain using the affine arithmetic.

However, the expression $f^{(p)}\left(\xi, y(\xi; y_{n-1}), d\right)$ is not so easy to bound as it requires to evaluate f for a particular value of the IVP solution $y(\xi; y_{n-1})$ at an unknown time $\xi \in]t_n, t_{n+1}[$. The solution used is the same as the one found in [17, 6] and it requires to bound the solution of IVP on the interval $[t_n, t_{n+1}]$. This bound is usually computed using the Banach's fixpoint theorem applied with the Picard-Lindelöf operator, see [17]. This operator is used to compute an enclosure of the solution $[\tilde{y}]$ of IVP over a time interval $[t_n, t_{n+1}]$, that is for all $t \in [t_n, t_{n+1}]$, $y(t; y_{n-1}) \in [\tilde{y}]$. We can hence bound $f^{(p)}$ substituting $y(\xi; y_{n-1})$ by $[\tilde{y}]$.

The main drawback of the previous approach is that implicit Runge-Kutta methods cannot be validated because ϕ is defined implicitly as a function of k_i. Implicit Runge-Kutta methods are important to deal with *stiff ordinary differential equations* and they have very good stability properties that make them suitable for validated numerical integration. Moreover, as function ϕ is defined over f then $\frac{d^{p+1}\phi}{dt^{p+1}}$ involves time derivatives of f. Hence computing separately $\frac{d^{p+1}\phi}{dt^{p+1}}$ and $f^{(p)}$, without taking into account shared intermediate computation, increases the simulation time. In [2], we define a new formula for LTE for any Runge-Kutta methods based on Fréchet derivatives, see [8] for more details. The new result in [2] is the definition of new validated numerical integration methods based on implicit Runge-Kutta methods.

▶ **Example 5.** Here we present an example coming from the `System 61` in the Vericomp database [3], which is defined by:

$$\begin{pmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 1 \\ x_2 \\ \frac{1}{6}x_1^3 - x_1 + 2\sin(d \cdot x_0) \end{pmatrix} \quad (6)$$

with $d = [2.78, 2.79]$ and the initial condition is $x_0(0) = x_1(0) = x_2(0) = 0$. Using our guaranteed integration method to compute the value of $x(10)$, we obtain $x(10) =$

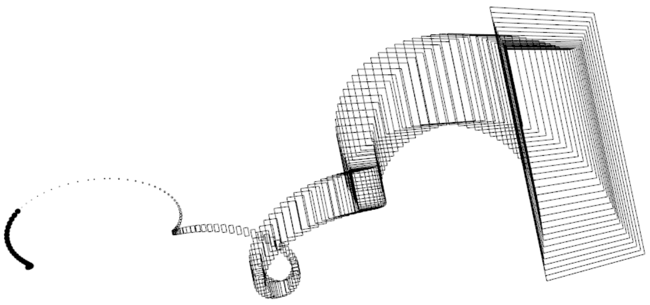


Figure 1 Complete simulation of `System 61`, see Eq. (6), in the Vericomp database over the time interval $[0, 10]$.

$([10, 10], [-1.6338, 1.69346], [-1.55541, 1.4243])^T$, with 4 rejected Picard-Lindelöf and 196 accepted ones. The minimum step-size is $h_{\min} = 0.00636859$ and the maximum step-size $h_{\max} = 0.070553$. Finally the maximum truncature error of the method is 8.9278×10^{-8}. Figure 1 presents the complete simulation of Equation (6). ∎

2.4 Paving

We call paving of a set $\mathcal{S} \subset \mathbb{R}^n$ the list of non-overlapping[1] boxes $[\mathbf{x_i}]$ with a non null width, such that each boxes $[\mathbf{x_i}] \subset \mathcal{S}$ [13]. This tool can be used to describe a set, by a list of inner boxes ($[\mathbf{x_i}] \subset \mathcal{S}$), the outer boxes ($[\mathbf{x_i}] \not\subset \mathcal{S}$) and the frontier, i.e. a list of boxes for which we cannot conclude of the membership to $\mathcal{S}$ in an acceptable computation time. For example, if the set $\mathcal{S}$ describes a ring such as $\mathcal{S} = \{(x, y) \mid x^2 + y^2 \in [1, 2]\}$, the paving of $\mathcal{S}$ in the box $[-2, 2] \times [-2, 2]$ gives the Figure 2.

3 Validated tuning method for PI controllers

3.1 PID controllers

PID controllers are widely used in industrial setting for integrating processes (see, e.g., this survey [10]). Therefore many techniques for their design and tuning have been proposed. In the next section we deal with the design of guaranteed PID controllers.

▶ **Definition 6.** A proportional-integral-derivative controller (PID), see [1] for more details, is represented by a tuple of tuning parameters $(K_p, K_i, K_d) \in \mathbb{R}^3$ designed to compute an error value

$$e(t) = r(t) - y(t) \tag{7}$$

with $r(t)$ a desired setpoint and $y(t)$ a measured process. The general mathematical description of PID is

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) \tag{8}$$

[1] This is true in $\mathbb{R}^n$ but no longer stands with a floating point representation due to rounding preserving over approximations.

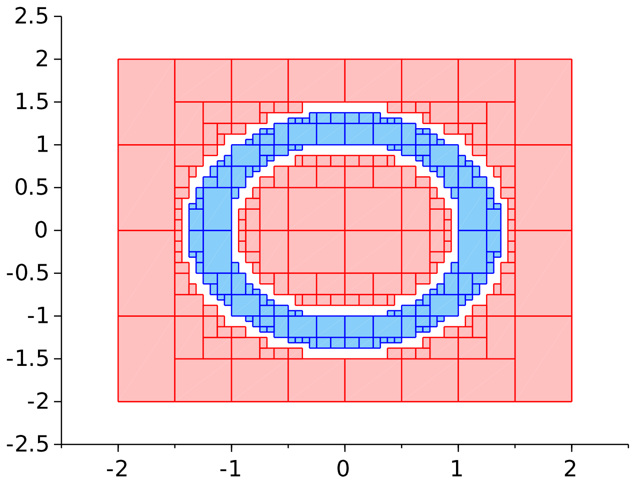


Figure 2 Paving of $x^2 + y^2 \in [1, 2]$, in blue boxes included in $\mathcal{S}$ and in red boxes which do not intersect $\mathcal{S}$.

where τ is the integration variable, $u(t)$ is the input signal of the plant model, the controller parameters are the proportional gain (K_p), the integral gain (K_i), and the derivative gain (K_d).

The choice of the tuple (K_p, K_i, K_d) is directed by the desire of obtaining a trade-off between fast response (convergence to the desired value) and good stability (no unbounded oscillation) of the control system.

3.2 Contribution

For our methods we restrict ourselves to the case where the derivative of the error is not taken into account. In this case we want to compute the set of validated PI parameters $(K_p, K_i) \in [P] \times [I]$ for a given control system. A dynamical system controlled by a PI can be written as

$$\dot{y}(t) = f(y, K_p, K_i, r), \tag{9}$$

with r the constant setpoint. Our method then consists on the computation of a paving of the set of validated PI parameters. For that, we simulate the dynamical system controlled by a PI whose parameters are in an interval. A PI controller defined by (K_p, K_i) is validated if it satisfies

$$\begin{cases} y(t_{end}) \in [r - \alpha\%, r + \alpha\%],\ 100 \geqslant \alpha > 0 \text{ (the desired setpoint is reached);} \\ \dot{y}(t_{end}) \in [-\epsilon, \epsilon],\ \epsilon > 0 \text{ (the system reached the stability zone).} \end{cases} \tag{10}$$

Algorithm 1 describes the computation of the paving of validated PI controllers. It produces two stacks of guaranteed boxes, the one of the accepted parameters and the one of the rejected parameters using a branch algorithm. For a given box $[x]$ of PI controller parameters, if the constraints defined in Equation (10) are satisfied then $[x]$ is put in the stack of guaranteed boxes, otherwise if the diameter of $[x]$ is greater than a given tolerance,

Algorithm 1 Compute the paving of PI parameters.

```
Require: $Stack = \emptyset$, $Stack_{accepted} = \emptyset$, $Stack_{rejected} = \emptyset$, $[x]_0 = ([P], [I])$
  Push $[x]_0$ in $Stack$
  while $Stack \neq \emptyset$ do
    Pop a $[x]$ from $Stack$
    Compute $y(t_{end}), \dot{y}(t_{end})$ using validated simulation of controlled plant with $[x]$
    if $(y(t_{end}) \in [r - \alpha\%, r + \alpha\%])$ && $(\dot{y}(t_{end}) \in [-\epsilon, \epsilon])$ then
      Push $[x]$ in $Stack_{accepted}$
    else if $(width([x]) > tol)$ && $(y(t_{end}) \ni setpoint)$ then
      $([x]_{left}, [x]_{right}) = Bisect([x])$
      Push $[x]_{left}$ in $Stack$
      Push $[x]_{right}$ in $Stack$
    else if $width([x]) >$ tol then
      Push $[x]$ in $Stack_{rejected}$
    else
      $[x]$ forgotten (cannot conclude)
    end if
  end while
```

$[x]$ is split into two boxes that are to be treated in the same way $[x]$ was. If $[x]$ does not meet the tolerance, it is rejected.

4 Experiments

In this section, we apply our PI tuning tool on the classic problem of cruise control. Our algorithm is developed with the IBEX library[2]. We developed a validated IVP solver inside this library, which will be released soon. This solver is the main brick of a quite classical branch and prune algorithm, already available in the library.

4.1 Modeling of the cruise-controller

To demonstrate the computation of PI controller parameters, our method is applied to the problem of automatic cruise control of a vehicle. The goal of this control is to maintain a constant vehicle speed despite external disturbances, such as change in wind or road grade. Action on the throttle has to be made if the vehicle speed is not the desired one.

4.1.1 In simplified form

The first considered example is the modeling of the cruise-controller with a simplified form of the vehicle dynamics. The vehicle to be controlled has a mass m, a velocity v, and is acted by a control force u representing the force generated at the road/tire interface. It is assumed that control on u can be done directly, that neglects the dynamics of the powertrain, tires, etc. In this simplified form is also considered that the resisting forces bv of the rolling resistance and wind drag vary linearly with v and act in the opposite direction of the vehicle's

[2] IBEX is a C++ library for constraint processing based on interval arithmetic: http://www.ibex-lib.org/.

motion. The vehicle system is described by $m\dot{v} + bv = u$ which can be rewritten as the ODE

$$\dot{v} = \frac{(u - bv)}{m}$$

The setpoint v_{set} on the speed is defined and corresponds to the particular velocity we want for the vehicle. So the error $e(t) = v_{set} - v$ and the force needed to make the vehicle go with a velocity in v_{set} is given by the PI

$$u = K_p(v_{set} - v) + K_i \int (v_{set} - v)ds,$$

and then the controlled ODE is

$$\dot{v} = \frac{(K_p(v_{set} - v) + K_i \int (v_{set} - v)dt - bv)}{m} \quad . \tag{11}$$

Let $\text{int}_{\text{err}} = \int (v_{set} - v)dt$ then ODE in Equation (11) can be written as the system

$$\begin{cases} \dot{v} = \dfrac{(K_p(v_{set} - v) + K_i \text{int}_{\text{err}} - bv)}{m} \\ \dfrac{d\text{int}_{\text{err}}}{dt} = v_{set} - v \end{cases}$$

4.1.2 With aerodynamic force

The case where aerodynamic force is not neglected is also considered here. Air particles flow over the hood of the vehicle causing the aerodynamic drag which can be modeled by

$$F_{drag} = (1/2)\rho CdAv^2$$

where ρ is the density of air, CdA is the coefficient of drag for the vehicle times the reference area, and v is the velocity of the vehicle. Here the density of the air is considered equal to 1.2041 kg/m^3 obtained with a temperature of 20°C and an atmospheric pressure of 101kPa. The consideration of aerodynamic drag leads to a non-linear differential system

$$\begin{cases} \dot{v} = \dfrac{(k_p(v_{set} - v) + k_i \text{int}_{\text{err}} - bv - (1/2)\rho CdAv^2)}{m} \\ \dfrac{d\text{int}_{\text{err}}}{dt} = v_{set} - v \end{cases}$$

4.2 Results of paving of PI parameters

Results on the application of our method on the previously described problem are now discussed. The values taken into account for the problem are $m \in [990, 1010]$, $v_{set} = 10$, $v_0 = 0$,$b = 50$, $t_{end} = 10$, $\alpha = 2\%$ and $\epsilon = 0.2$, $(1/2)\rho CdA = 0.4$, $PI_0 = ([1, 4000], [1, 120])$, tol $= 1$.

Simulation with interval parameters.

Firstly, to validate our simulation tool, we integrate the dynamical system with the simplified form of the vehicle dynamics with interval parameters for the PI controller. We start a simulation with $K_p = [900, 950]$ and $K_i = [35, 45]$, from $t = 0$ to $t = 15$. The result is shown in Figure 3. On this figure, we can see the boxes computed by each steps of integration, and that a part of them cross the limit at $t = 10$. It means that some of the parameters are not satisfying for the purpose. During the paving computation, the interval parameters would be split and two new simulations would start.

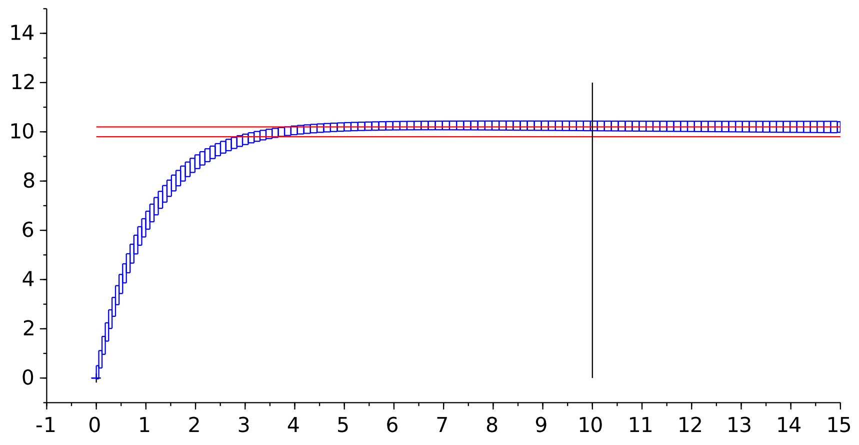


Figure 3 Simulation of the cruise-controller with linear dynamics and interval PI parameters.

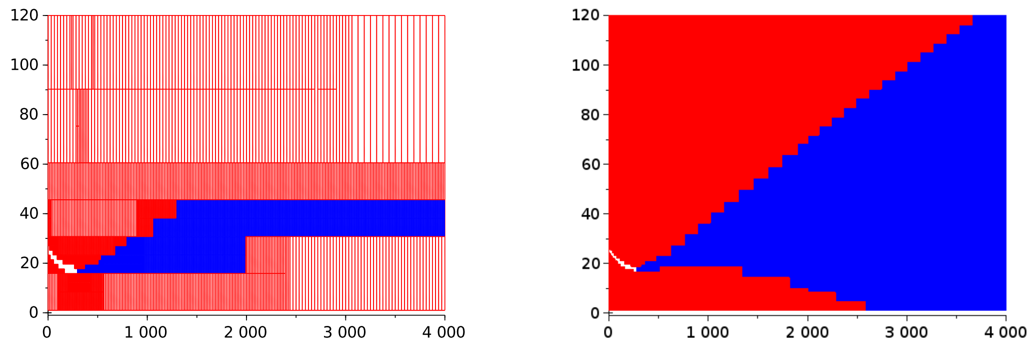


Figure 4 Paving of PI parameters for the linear (on the left) and non-linear (on the right) cruise-controller – in blue accepted and in red rejected.

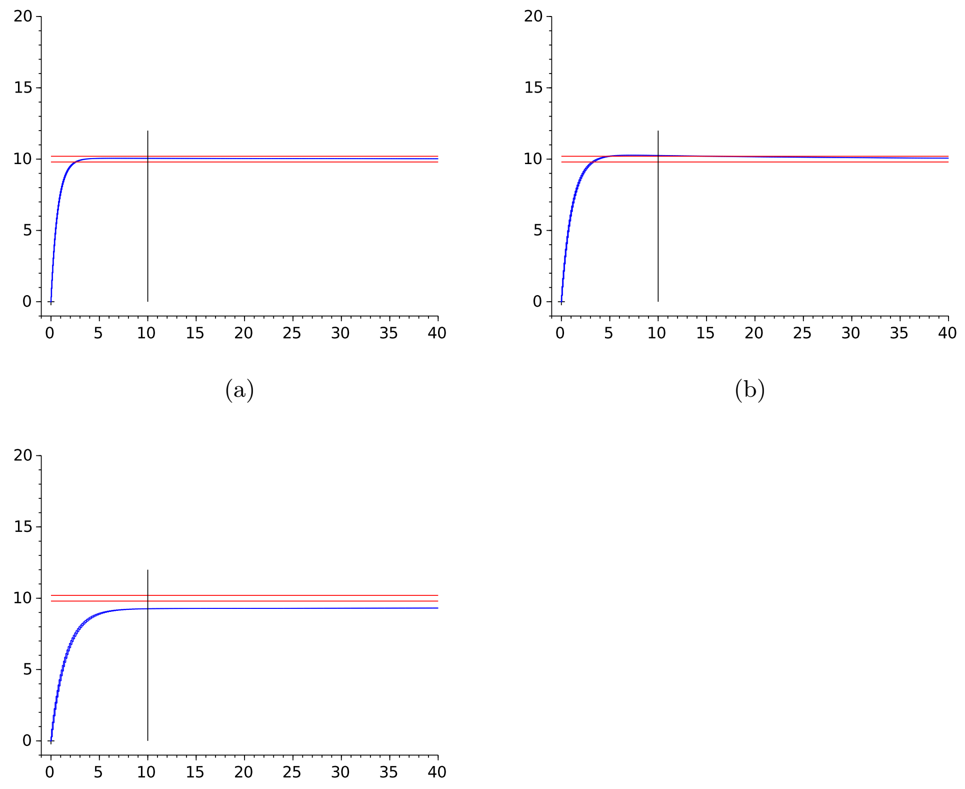
Complete paving.

Figure 4 presents the result of paving for the guaranteed PI parameters for the linear and non linear modeling of the automatic cruise controller.

4.3 Response of controller along time

To verify our results, we plot the response of the cruise controller along time, that is to say the validated simulation of speed of the vehicle from 0 to 40 seconds. We recall that the setpoint is 10km/h and that we would like to attain this value at 10s. Figure 5 gathers the responses for the linear model of the cruise controller, with a validated set of parameters (a), a rejected set of parameters (b) and a set of parameters found in the literature [11] (c). The results of our tool clearly match the constraints for setpoint and stability. The latter set of parameters, if it is correct for an infinite time, does not lead to the setpoint at an acceptable response time.

Figure 6 gathers the responses for the non linear model of the cruise controller (considering the aerodynamic force), with a validated set of parameters (a), a rejected set of parameters (b)



(a) (b) (c)

Figure 5 Response of controller for linear plant with parameters validated ($K_p = 1400$ and $K_i = 35$, in (a)), rejected ($K_p = 900$ and $K_i = 40$, in (b)) and from literature [11] ($K_p = 600$ and $K_i = 1$, in (c)) – in blue the guaranteed response, in red the supervision of the objective, and in black the deadline.

and a set of parameters found in the literature [11] (c). This set coming from an optimization process of a PID which leads to $K_d = 0$. The results of our tool clearly match the constraints for setpoint and stability. For the set of parameters coming from literature (c), the response is correct in term of objective, but this set of parameters leads to a large overshoot, and a strong instability. We can conclude from these results and the current literature that our tool can provide guarantees on sets of parameter for a controller of a non-linear system.

5 Conclusion

We presented a new method for the tuning of PI controllers. Our approach is based on the guaranteed simulation of controlled systems and on the paving of the K_p and K_i parameter space. We can then guarantee a set of parameters for which the response validates two conditions: reach the setpoint and reach the stability zone after a given time lapse. Our tool used to simulate the plant allowing us to perform non-linear integration, we applied it to the non-linear cruise controller. Our results are compared with some values found in the literature and lead to think that our approach is promising. This work may be improved in many ways. We can extend the kind of properties a PI controller must satisfy as a maximum overshoot. Moreover, we will consider PID controllers in the future.

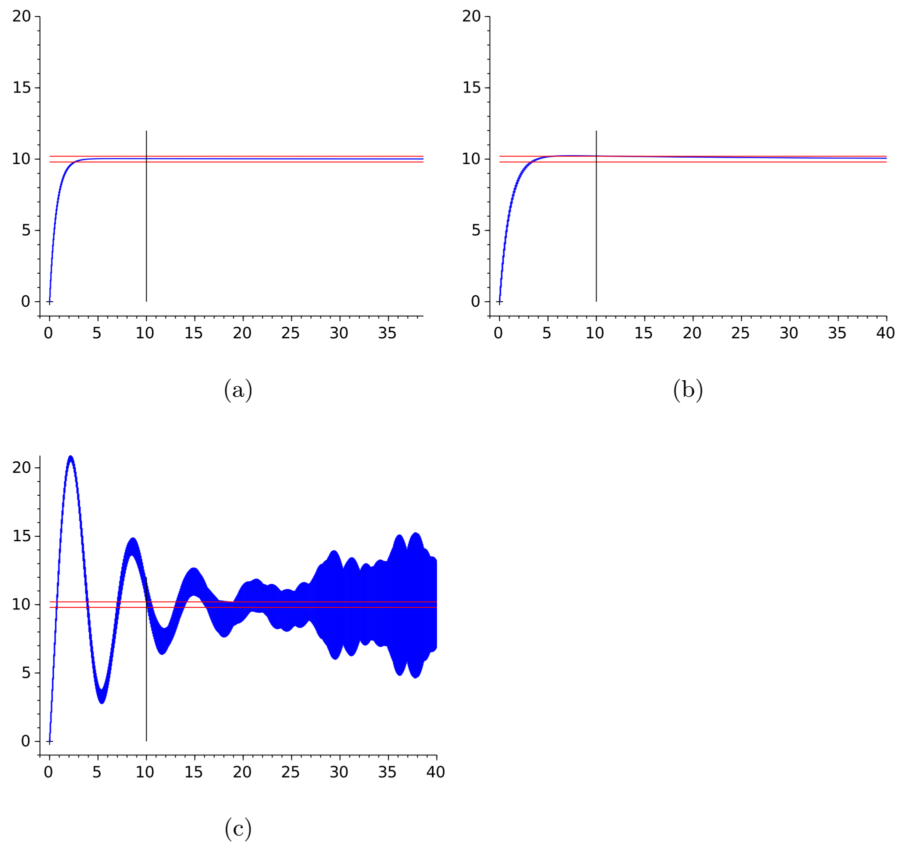


Figure 6 Response of controller for non-linear plant with parameters validated ($K_p = 1400$ and $K_i = 35$, in (a)), rejected ($K_p = 900$ and $K_i = 40$, in (b)) and from literature [11] ($K_p = 232.58$ and $K_i = 1000$ in (c)) – in blue the guaranteed response, in red the supervision of the objective, and in black the deadline.

References

1 K. J. Ãström and T. Hägglund. PID controllers: theory, design, and tuning. *Instrument Society of America, Research Triangle Park, NC*, 1995.

2 J. Alexandre dit Sandretto and A. Chapoutot. Validated Solution of Initial Value Problem for Ordinary Differential Equations based on Explicit and Implicit Runge-Kutta Schemes. Research report, ENSTA ParisTech, January 2015.

3 E. Auer and A. Rauh. Vericomp: a system to compare and assess verified IVP solvers. *Computing*, 94(2-4):163–172, 2012.

4 J. Bondia, M. Kieffer, E. Walter, J. Monreal, and J. Picó. Guaranteed tuning of PID controllers for parametric uncertain systems. In *Decision and Control*, page 2948–2953. IEEE, 2004.

5 O. Bouissou, A. Chapoutot, and A. Djoudi. Enclosing temporal evolution of dynamical systems using numerical methods. In *NASA Formal Methods*, number 7871 in LNCS, pages 108–123. Springer, 2013.

6 O. Bouissou and M. Martel. GRKLib: a Guaranteed Runge Kutta Library. In *Scientific Computing, Computer Arithmetic and Validated Numerics*, 2006.

7 V. Broida. Extrapolation des résponses indicielles apériodiques. *Automatisme*, XVI, 1969.

8 J. C. Butcher. Coefficients for the study of Runge-Kutta integration processes. *Journal of the Australian Mathematical Society*, 3:185–201, 5 1963.

9 L. H. de Figueiredo and J. Stolfi. *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium monographs. IMPA/CNPq, 1997.

10 L. Desborough and R. Miller. Increasing customer value of industrial control performance monitoring - Honeywell's experience. In *AIChE Symposium Series*, pages 169–189, 2002.

11 A. Dowling. Modeling and PID controller example - cruise control for an electric vehicle.

12 E. Hairer, Syvert P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, 2nd edition, 2009.

13 L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.

14 Y. Lin and M. A. Stadtherr. Validated solutions of initial value problems for parametric odes. *Applied Numerical Mathematics*, 57(10):1145–1162, 2007.

15 R. J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. *Computer Arithmetic*, pages 255–286, 1987.

16 R. Moore. *Interval Analysis*. Prentice Hall, 1966.

17 N. Nedialkov, K. Jackson, and G. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. and Comp.*, 105(1):21 – 68, 1999.

18 A. Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confidence regions. *Computing Supplementum*, 9, 1993.

19 J. Vehì, I. Ferrer, and M. À. Sainz. A survey of applications of interval analysis to robust control. In *IFAC World Congress*, 2002.

20 J.G. Ziegler and N.B. Nichols. Optimum settings for automatic controllers. *Journal of dynamic systems, measurement, and control*, 115(2B):220–222, 1993.

Discrete Parameters in Petri Nets*

Nicolas David[1], Claude Jard[1], Didier Lime[2], and Olivier H. Roux[2]

1 **University of Nantes, LINA**
Nantes, France
nicolas.david1@univ-nantes.fr ; claude.jard@univ-nantes.fr

2 **École Centrale de Nantes, IRCCyN**
Nantes, France
didier.lime@ec-nantes.fr ; olivier-h.roux@irccyn.ec-nantes.fr

Abstract

With the aim of significantly increasing the modeling capability of Petri nets, we suggest models that involve parameters to represent the weights of arcs, or the number of tokens in places. We call these Petri nets *parameterised nets* or *PPNs*. Indeed, the introduction of parameters in models aims to improve genericity. It therefore allows the designer to leave unspecified aspects, such as those related to the modeling of the environment. This increase in modeling power usually results in greater complexity in the analysis and verification of the model. Here, we consider the property of coverability of markings. Two general questions arise: "Is there a parameter value for which the property is satisfied?" and "Does the property hold for all possible values of the parameters?". We first study the decidability of these issues, which we show to be undecidable in the general case. Therefore, we also define subclasses of parameterised networks, based on restriction of the use of parameters, depending on whether the parameters are used on places, input or output arcs of transitions or combinations of them. Those subclasses have therefore a dual interest. From a modeling point of view, restrict the use of parameters to tokens, outputs or inputs can be seen as respectively processes or synchronisation of a given number of processes. From a theoretical point of view, it is interesting to introduce those subclasses of PPN in a concern of completeness of the study. We study the relations between those subclasses and prove that, for some subclasses, certain problems become decidable, making these subclasses more usable in practice.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Petri net, parameters, coverability

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.103

Category Informal Presentation

* This work has been presented to Petri Nets 2015.

Enhanced Distributed Behavioral Cartography of Parametric Timed Automata

Étienne André, Camille Coti, and Hoang Gia Nguyen

Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France

Abstract

Parametric timed automata (PTA) [1] allow the specification and verification of timed systems incompletely specified, or featuring timing constants that may change either in the design phase, or at runtime.

The behavioral cartography of PTA (BC) [4] relies on the idea of covering a bounded parameter domain with tiles, i.e., parts of the parameter domain in which the discrete behavior is uniform. This is achieved by iterating the inverse method (IM) [2] on the (yet uncovered) integer parameter valuations ("points") of the bounded parametric domain: given a reference point, IM generalizes the behavior corresponding to this point by synthesizing a constraint containing other (integer and real-valued) points with the same discrete behavior. Then, given a linear time property, it is easy to partition the parametric domain into a subset of "good" tiles and a subset of "bad" ones (which correspond to good and bad behaviors).

Useful applications of BC include the optimization of timing constants, and the measure of the system robustness (values around the reference parameters) w.r.t. the untimed language. In practice, a parameter domain with a large number of integer points will require a long time to compute BC. To alleviate that, our goal is to take advantage of powerful distributed architectures.

Distributing BC is theoretically easy, since it is trivial that two executions of IM from two different points can be performed on two different nodes. However, distributing it efficiently is challenging. For example, calling two executions of IM from two contiguous integer points has a large probability to yield the same tile in both cases, resulting in a loss of time for one of the two nodes. Thus, the critical question is how to distribute efficiently the point on which to call IM.

In a previous work [3], a master-worker scheme is proposed, where the master assigns points to each worker process, which is called a point-based distribution scheme. In this point-based distribution scheme, choosing the point distribution approach on the master side is the key point that will decide the algorithm performance. Since the master has no ability to foresee the tiles on cartography (the "shape" of a cartography is unknown in general), two or more processes can receive close points, that then yield the same result, leading to a loss of efficiency. Besides that, two or more tiles can overlap each other; hence, the question is whether we stop an going process starting from a point that is already covered by another tile. Finally, a very large parameter domain (with many integer points) can cause a bottleneck phenomenon on the master side since many worker processes ask for point, while the master is busy to find uncovered points.

From the previous problems, we proposed an enhanced master-worker distributed algorithm, based on a domain decomposition scheme. The main idea is that the master splits the parameter domain into subdomains, and assigns them to the workers. Then workers will work on their own set of points, hence reducing the probability of choosing close points since the workers work as far as possible from each other. Then, when a worker finishes the coverage of its subdomain, it asks the master for a new subdomain: the master splits a slow worker's subdomain into two parts, and sends it to the fast worker. Furthermore, we used a heuristic approach to decide whether to stop a process working on a point that has been covered by a tile of another worker. In all our experiments, our enhanced distributed algorithm outperforms previous algorithms.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Formal methods, model checking, distributed algorithmic, real-time systems, parameter synthesis

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.104

Category Informal Presentation

References

1 Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.

2 Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, 2009.

3 Étienne André, Camille Coti, and Sami Evangelista. Distributed behavioral cartography of timed automata. In *EuroMPI/ASIA*, pages 109–114. ACM, 2014.

4 Étienne André and Laurent Fribourg. Behavioral cartography of timed automata. In *RP*, volume 6227 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2010.

Parameter Synthesis with IC3

Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta

Fondazione Bruno Kessler
Trento, Italy
`{cimatti,griggio,mover,tonettas}@fbk.eu`

Abstract

Parametric systems arise in many application domains, from real-time systems to software to cyber-physical systems. Parameters are fundamental to model unknown quantities at design time and allow a designer to explore different instantiation of the system (i.e. every parameter valuation induces a different system), during the early development phases.

A key challenge is to automatically synthesize all the parameter valuations for which the system satisfies some properties. In this talk we focus on the parameter synthesis problem for infinite-state transition systems and invariant properties. We describe the synthesis algorithm ParamIC3 [1], which is based on IC3, one of the major recent breakthroughs in SAT-based model checking, and lately extended to the SMT case.

The algorithm follows a general approach that first builds the set of "bad" parameter valuations and then obtain the set of "good" valuations by complement. The approach enumerates the counterexamples that violate the property, extracting from each counterexample a region of bad parameter valuations, existentially quantifying the state variables.

ParamIC3 follows the same principles, but it overcomes some limitations of the previous approach by exploiting the IC3 features. First, IC3 may find a set of counterexamples $s_o, \ldots, s_k$, where each state in s_i is guaranteed to reach some of the bad states in s_k in $k - i$ steps; this is exploited to apply the expensive quantifier elimination on shortest, and thus more amenable, counterexamples. Second, the internal structure of IC3 allows our extension to be integrated in a fully incremental fashion, never restarting the search from scratch to find a new counterexample.

While various approaches already solve the parameter synthesis problem for several kind of systems, like infinite-state transition systems, timed and hybrid automata, the advantages ParamIC3 are that: it synthesizes an optimal region of parameters, it avoids computing the whole set of the reachable states, it is incremental and applies quantifier elimination only to small formulas.

We present the results of an experimental evaluation performed on benchmarks from the timed and hybrid systems domain. We compared the approach with similar SMT-based techniques and with techniques based on the computation of the reachable states. The results show the potential of our approach.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases parameter synthesis, infinite-state transition systems, satisfiability modulo theories, IC3

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.106

Category Informal Presentation

References

1 Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Parameter synthesis with IC3. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 165–168, 2013.