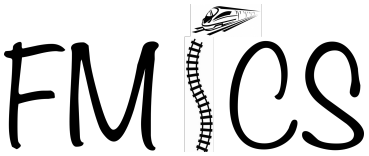
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche

Technical Report 2016–TR–010

Research Ideas
FMICS-AVoCS 2016

International Workshop on
Formal Methods for Industrial Critical Systems
and Automated Verification of Critical Systems



ISTI–CNR, Pisa, Italy
26–28 September 2016

Editors
Maurice H. ter Beek, Stefania Gnesi, and Alexander Knapp

Table of Contents

Preface

This technical report of ISTI contains the research ideas presented at the International Workshop on Formal Methods for Industrial Critical Systems and Automated Verification of Critical Systems (FMICS-AVoCS), which was held in Pisa, Italy, September 26–28, 2016. FMICS-AVoCS 2016 combines the 21st International Workshop on Formal Methods for Industrial Critical Systems and the 16th International Workshop on Automated Verification of Critical Systems.

The aim of the FMICS workshop series is to provide a forum for researchers who are interested in the development and application of formal methods in industry. In particular, FMICS brings together scientists and engineers that are active in the area of formal methods and interested in exchanging their experiences in the industrial usage of these methods. The FMICS workshop series also strives to promote research and development for the improvement of formal methods and tools for industrial applications.

The aim of the AVoCS workshop series is to contribute to the interaction and exchange of ideas among members of the international research community on tools and techniques for the verification of critical systems. It covers all aspects of automated verification, including model checking, theorem proving, SAT/SMT constraint solving, abstract interpretation, and refinement pertaining to various types of critical systems which need to meet stringent dependability requirements (safety-critical, business-critical, performance-critical, etc.).

FMICS-AVoCS 2016 encouraged the submission of research ideas in order to stimulate discussions at the workshop. These research ideas typically concern reports on ongoing work or surveys on work published elsewhere related to the topics of interest to FMICS-AVoCS, which include but are not limited to:

- Design, specification, refinement, code generation and testing of critical systems based on formal methods
- Methods, techniques and tools to support automated analysis, certification, debugging, learning, optimization and transformation of critical systems, in particular distributed, real-time systems and embedded systems
- Automated verification (model checking, theorem proving, SAT/SMT constraint solving, abstract interpretation, etc.) of critical systems
- Verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability (e.g., scalability and usability issues)
- Tools for the development of formal design descriptions
- Case studies and experience reports on industrial applications of formal methods, focussing on lessons learned or identification of new research directions
- Impact of the adoption of formal methods on the development process and associated costs
- Application of formal methods in standardization and industrial forums

We received 5 research abstracts, which we decided to accept all for a short presentation during the workshop and inclusion in this technical report. This technical report accompanies the formal proceedings of FMICS-AVoCS 2016 published by Springer as volume 9933 of their Lectures Notes in Computer Science series.

We are very grateful to our sponsors, the European Research Consortium for Informatics and Mathematics (ERCIM), Formal Methods Europe (FME), and Springer International Publishing AG. We also thank Tiziana Margaria (University of Limerick & LERO, the Irish Software Research Center, Ireland), the coordinator of the ERCIM working group FMICS, and the other board members, as well as the steering committee of AVoCS, all listed below, for their continuous support during the organization of FMICS-AVoCS. We acknowledge the support of EasyChair for assisting us in managing the complete process from submission to this technical report.

Finally, we thank all authors for their submissions and all attendees of the workshop for their participation.

September 2016

Maurice ter Beek
Stefania Gnesi
Alexander Knapp

Organization

General Chair

Maurice H. ter Beek	ISTI–CNR, Pisa, Italy

Program Committee Co-chairs

Stefania Gnesi	ISTI–CNR, Pisa, Italy
Alexander Knapp	Universität Augsburg, Germany

Program Committee

Maria Alpuente	Universitat Politècnica de Valéncia, Spain
Jiri Barnat	Masarykova Univerzita, Czech Republic
Michael Dierkes	Rockwell Collins, Blagnac, France
Cindy Eisner	IBM Research, Haifa, Israel
Alessandro Fantechi	Università di Firenze, Italy
Francesco Flammini	Ansaldo STS, Naples, Italy
María del Mar Gallardo	Universidad de Málaga, Spain
Michael Goldsmith	University of Oxford, UK
Gudmund Grov	Heriot-Watt University, UK
Matthias Güdemann	Diffblue ltd., Oxford, UK
Marieke Huisman	Universiteit Twente, The Netherlands
Gerwin Klein	NICTA and University of New South Wales, Australia
Peter Gorm Larsen	Aarhus Universitet, Denmark
Thierry Lecomte	ClearSy, Aix-en-Provence, France
Tiziana Margaria	University of Limerick and LERO, Ireland
Radu Mateescu	INRIA Grenoble Rhône-Alpes, France
David Mentré	Mitsubishi Electric R&D Centre Europe, Rennes, France
Stephan Merz	INRIA Nancy and LORIA, France
Manuel Núñez	Universidad Complutense de Madrid, Spain
Peter Ölveczky	Universitetet i Oslo, Norway
Charles Pecheur	Université Catholique de Louvain, Belgium
Marielle Petit-Doche	Systerel, Aix-en-Provence, France
Ralf Pinger	Siemens AG, Braunschweig, Germany
Jaco van de Pol	Universiteit Twente, The Netherlands
Markus Roggenbach	Swansea University, UK
Matteo Rossi	Politecnico di Milano, Italy
Marco Roveri	FBK-irst, Trento, Italy
Thomas Santen	Microsoft Research Advanced Technology Labs Europe, Aachen, Germany

Bernhard Steffen	Universität Dortmund, Germany
Jun Sun	University of Technology and Design, Singapore
Helen Treharne	University of Surrey, UK

FMICS WG Board Members

Álvaro Arenas	IE Business School, Madrid, Spain
Luboš Brim	Masarykova Univerzita, Czech Republic
Alessandro Fantechi	Università di Firenze, Italy
Hubert Garavel	INRIA Grenoble Rhône-Alpes, France
Stefania Gnesi	ISTI–CNR, Pisa, Italy
Diego Latella	ISTI–CNR, Pisa, Italy
Tiziana Margaria	University of Limerick and LERO, Ireland
Radu Mateescu	INRIA Grenoble Rhône-Alpes, France
Pedro Merino	Universidad de Málaga, Spain
Jaco van de Pol	Universiteit Twente, The Netherlands

AVoCS Steering Committee

Michael Goldsmith	University of Oxford, UK
Stephan Merz	INRIA Nancy and LORIA, France
Markus Roggenbach	Swansea University, UK

Sponsors

ERCIM
European Research Consortium for Informatics and Mathematics

FORMAL METHODS EUROPE

Springer

Towards Multi-Core Symbolic Model Checking of Timed Automata*

Arnd Hartmanns, Sybe van Hijum, Jeroen Meijer, and Jaco van de Pol

University of Twente, Enschede, The Netherlands

1 Introduction

Timed automata (TA) are widely used for modeling, analysis and optimisation. Typical applications, where timing is crucial, include distributed protocols, embedded systems, and hard real-time control. Their analysis is supported by model checkers like Uppaal [1] or Rabbit [3]. Model checking is an exhaustive technique, limiting its scalability. One solution direction is to reduce resource usage by partial order reduction (POR) or by concise symbolic representations using decision diagrams (BDDs). An orthogonal direction is to exploit more resources efficiently by high-performance model checking. LTSmin [7] provides distributed and multi-core algorithms for model checking, compatible with POR, and supporting multi-core decision diagrams through Sylvan [5]. Earlier, we applied LTSmin to TA through Opaal [4], an open-source code generator for Uppaal models. The Pins-API of LTSmin supports multiple specification languages, by an on-the-fly next-state function. We applied this for reachability analysis and LTL model checking [8] on TA. Clock zones were encoded as difference bound matrices (DBM) in a single monolithic variable, and the Pins API was extended with a subsumption check on DBMs. This led to a competitive, scalable explicit-state model checker for TA.

In this extended abstract, we investigate whether reductions like POR and BDD representations can be combined with high-performance model checking of TA. Technically, this comes for free with the implementation of the Pins API. Yet in practice, those techniques are only effective for specifications with high locality, i.e. when the dependency matrix between state variables and transition groups is sparse. Viewing DBMs as a monolithic variable, which by the nature of time affects all transition groups, destroys locality.

2 Zone Representations

Opaal provides an interface to a TA's zone graph using an explicit representation of discrete state variables plus DBMs for zones. A **DBM** explicitly stores the relationships between all pairs of clocks. To represent zones in a more space-efficient way, specialised variants of decision diagrams have been developed:

- A **CDD** [2] represents the difference between a pair of clocks using a single node. Its multiple outgoing edges are labelled with disjoint intervals. CDD algorithms need to re-establish disjointness after every step.

* This work is partly supported by the 3TU project "Big Software on the Run".

– A **DDD** [9] uses multiple nodes per clock difference. Each node represents an upper bound constraint on the difference and has two edges: the constraint is true or false. False edges lead to another constraint on the same clocks.
– A **CRD** [10] also uses nodes for upper bounds on clock differences, but allows multiple edges with (overlapping) upper bounds. There are several normal forms for CRDs with different performance characteristics.
– **CMD**s [6] combine CDDs, CRDs and DBMs into a single structure. They have multiple constraints per edge, leading to fewer nodes but complex edges.
– The direct use of BDDs has also been proposed [11]. Subsumption and inclusion cannot be checked with standard BDD algorithms. RABBIT [3] also uses BDDs directly, but is based on digitisation so does not support open clock constraints.

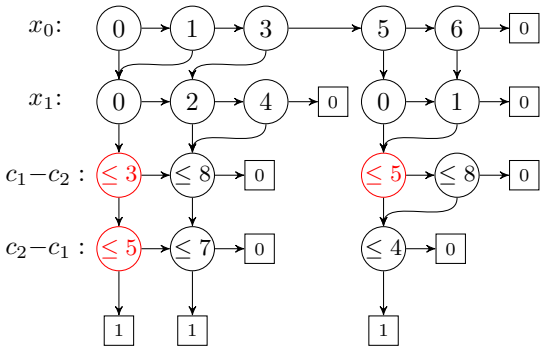
A known challenge in decision diagrams is to find a good variable ordering. Reordering experiments have been conducted for CRDs only, so far.

3 Design

In [8], the state vector consists of all discrete model variables, plus one timed variable, which is a reference to a monolithic DBM managed by OPAAL. LTSMIN calls PINS' next-state function, which is implemented by OPAAL and returns the set of successor state vectors, including their DBM. By locality, LTSMIN only needs to re-explore a transition if one of its dependent state variables changed. We now show how we can gradually increase locality by a migration path of four steps:

1. We first flatten the DBM into n^2 separate state variables, indicating for n clocks a bound v on their differences, $c_i - c_j \leq v$, hopefully exposing more locality.
2. We store the state space on the LTSMIN/SYLVAN side in a List Decision Diagram (LDD). An LDD represents a set of integer vectors canonically. Level i of the diagram holds all nodes for variable x_i, consisting of a value v, a down-pointer to the next level (if $x = v$) and a next-pointer in the same level (if $x > v$).
3. We change the interpretation of the nodes representing the DBM in the LDD, taking into account that they actually represent clock differences. The result is a mix of LDD nodes (for integer variables) and DDD nodes (for clock differences). This enables new reductions, e.g. both red nodes labeled ≤ 5 can be eliminated.
4. The final step is to implement symbolic DDD operations for all time computations: the delay operator, invariant constraining, clock resets, and extrapolation.

This diagram shows a mixed LDD/DDD, as in Step 3. The main challenge towards completing Step 4 is to redefine the PINS interface. We would need alternative ways to communicate the appropriate invariants and clock resets, without compromising specification language independence.



4 Preliminary Results and Outlook

We show the runtime (t_{LDD}, t_{DDD}, in seconds) and number of decision diagram nodes (n_{LDD}, n_{DDD}) of our current prototype implementation for four standard benchmarks on the right. We see that DDD lead to smaller diagrams here, so they indeed appear to be more memory-efficient. In terms of runtime, the comparison is not so clear; we expect improvements here as the implementation progresses.

model	t_{LDD}	n_{LDD}	t_{DDD}	n_{DDD}
fisher_5	5.9	17,131	10.2	5,535
critreg_3	2.2	3,743	3.6	3,825
csmacd_8	22.8	324,047	10.4	36,098
viking_{10}	31.5	304	34.0	304

There are several venues for further experimentation and design improvements. Since Sylvan is a multi-core LDD package, one could now investigate the speedup of multi-core symbolic TA model checking. Also, one could investigate the effectiveness of existing variable reordering heuristics (which we used already for the LDD columns above), or design specialised heuristics for clock variables. The flattened DBMs also provide a first step to experiment with partial-order reduction for TA.

More fundamental improvements would be to further improve the locality, redesign the Pins-API for timed constraints and clock updates, and implement more symbolic timed operations. One could extend the design with other decision diagram types, like CRDs, CDDs and CMDs. Finally, this line of research could be compared with the use of symbolic model checking for TA with discretised clocks [3].

References

1. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: FORMATS. LNCS 3185, Springer (2004)
2. Behrmann, G., Larsen, K.G., Pearson, J., Weise, C., Yi, W.: Efficient timed reachability analysis using clock difference diagrams. In: CAV. LNCS, vol. 1633. Springer (1999)
3. Beyer, D., Lewerentz, C., Noack, A.: Rabbit: A tool for BDD-based verification of real-time systems. In: CAV. LNCS, vol. 2725. Springer (2003)
4. Dalsgaard, A., Hansen, R., Jørgensen, K., Larsen, K., Olesen, M., Olsen, P., Srba, J.: opaal: A lattice model checker. In: NFM. LNCS 6617, Springer (2011)
5. van Dijk, T., van de Pol, J.: Sylvan: Multi-core decision diagrams. In: TACAS. LNCS 9636 (2015)
6. Ehlers, R., Fass, D., Gerke, M., Peter, H.: Fully symbolic timed model checking using constraint matrix diagrams. In: RTSS. IEEE Comp. Soc. (2010)
7. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: High-performance language-independent model checking. In: TACAS. LNCS 9636 (2015)
8. Laarman, A., Olesen, M.C., Dalsgaard, A.E., Larsen, K.G., van de Pol, J.: Multi-core emptiness checking of timed Büchi automata using inclusion abstraction. In: Sharygina, N., Veith, H. (eds.) CAV. LNCS 8044, Springer (2013)
9. Møller, J.B., Lichtenberg, J., Andersen, H.R., Hulgaard, H.: Fully symbolic model checking of timed systems using difference decision diagrams. ENTCS 23(2) (1999)
10. Wang, F.: Efficient verification of timed automata with BDD-like data-structures. In: VMCAI. LNCS, vol. 2575. Springer (2003)
11. Zhang, H., Du, J., Cao, L., Zhu, G.: A full symbolic reachability analysis algorithm of timed automata based on BDD. In: ISADS. IEEE Comp. Soc. (2015)

Parametric Interval Markov Chains: Synthesis Revisited

Laure Petrucci[1⋆] and Jaco van de Pol[2⋆⋆]

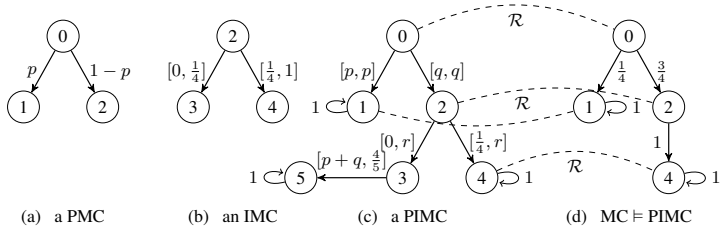
[1] LIPN, CNRS UMR 7030, Université Paris 13, Sorbonne Paris Cité, Villetaneuse, France
[2] Formal Methods and Tools, University of Twente, Enschede, The Netherlands

1 Introduction

Markov Chains (MC) are widely used to model stochastic systems, like randomised protocols, failure and risk analysis, and modelling phenomena in molecular biology. Here we focus on discrete time MCs, where transitions between states are governed by a state probability distribution, denoted by $\mu : S \times S \to [0, 1]$. Practical applications are often hindered by the fact that the probabilities $\mu(s, t)$, to go from state s to t are unknown. Several solutions have been proposed, for instance Parametric Markov Chains (PMC) [3] and Interval Markov Chains (IMC) [7], in which unknown probabilities are replaced by parameters or intervals, respectively; see the diagram (a), (b). Following [4], we study their common generalization, Parametric Interval Markov Chains (PIMC), which allow intervals with parametric bounds. PIMCs allow to study the boundaries of admissible probability intervals, which is useful in the design exploration phase. This leads to the study of parameter synthesis for PIMCs, started in [6].

IMCs can be viewed as specifications of MCs. An IMC is consistent if there exists some MC that implements it. The main requirements of this relation are: (1) all behaviour of the MC can be simulated by the IMC; (2) the transition probabilities out of each state sum up to 1. The consistency synthesis problem for PIMCs computes all parameter values leading to a consistent IMC. E.g., PIMC (c) is consistent when $q = 0, p = 1$, or $p + q = 1, r = 1$. The implementation MC (d) corresponds to $r = 1, p = \frac{1}{4}, q = \frac{3}{4}$.

Our contribution is a simplification of the theory in [6], together with a formal verification in PVS. In particular, we provide a simple co-inductive definition of consistency, and prove that it coincides with the inductive notion of n-consistency in [6]. Finally, based on these definitions, we provide algorithms for the consistency check of IMCs and the consistency parameter synthesis of PIMCs. We implemented the latter as a Constraint Logic Program over the reals, in SWI-prolog with clp(R).



(a) a PMC (b) an IMC (c) a PIMC (d) MC ⊨ PIMC

⋆ This work has been partially supported by project PACS ANR-14-CE28-0002.
⋆⋆ This research was performed when the author was invited professor at Université Paris 13.

2 Consistency of Parametric Interval Markov Chains

We first define (Parametric Interval) Markov Chains and their relation. For PIMCs, P denotes the set of parameters and $\mathsf{Int}[0,1](P)$ denotes intervals $[p,q]$ with $p,q \in P \cup [0,1]$. For IMCs, $P = \varnothing$ and PMCs only have point intervals $[p,p]$.

Definition 1 (MC, PIMC, $\models$, consistency [7,4]). *A* Markov Chain *(MC) is a tuple $\mathcal{M} = (S, s_0, \mu)$, with initial state $s_0 \in S$ and probability distribution $\mu : S \times S \to [0,1]$, i.e., $\forall s \in S : \sum_{s' \in S} \mu(s,s') = 1$. A* Parametric Interval Markov Chain *(PIMC) is a tuple $\mathcal{I} = (T, t_0, P, \varphi)$, with initial state $t_0 \in T$, parameters P, and parametric probability distribution $\varphi : T \times T \to \mathsf{Int}[0,1](P)$.*
$\mathcal{M}$ implements *$\mathcal{I}$ ($\mathcal{M} \models \mathcal{I}$) if there exists a simulation relation $\mathcal{R} \subseteq S \times T$, such that for all $s\mathcal{R}t$, there exists a probabilistic correspondence $\delta : S \times T \to \mathsf{Int}[0,1]$, with:*

1. $\forall t' \in T : \sum_{s' \in S} \mu(s,s') \cdot \delta(s',t') \in \varphi(t,t')$
   (the total contribution of the implementing transitions satisfies the specification)
2. $\forall s' \in S : \mu(s,s') > 0 \Rightarrow \sum_{t' \in T} \delta(s',t') = 1$
   (the implementing transitions yield a probability distribution)
3. $\forall s' \in S, t' \in T : \delta(s',t') > 0 \Rightarrow s'\mathcal{R}t'$
   (corresponding successors are in the simulation relation)

A PIMC $\mathcal{I}$ is consistent if for some parameters P and MC $\mathcal{M}$, we have $\mathcal{M} \models \mathcal{I}(P)$.

We write $\varphi(t)(t')$ for $\varphi(t,t')$. For $\nu = \varphi(t)$, lower/upper bounds are denoted by $\nu(t') = [\nu_\ell(t'), \nu_u(t')]$. The support $sup(\nu) = \{t' \mid \nu_u(t') \neq 0\}$. From now on, we assume that $sup(\varphi(t))$ is a finite set, so all sums below are well-defined.

Local consistency of state s w.r.t. a selected subset $X \subseteq T$ indicates compatibility with the associated $\nu = \varphi(s)$: The lower and upper bounds of the selected transitions are properly ordered and summing them up admits the probability mass 1. The unselected transitions should allow probability 0.

Definition 2. *We define* local consistency constraints *$LC(\nu, X)$ as a conjunction of:*

$$up(\nu,X) = \textstyle\sum_{t \in X} \nu_u(t) \geq 1$$
$$low(\nu,X) = \textstyle\sum_{t \in X} \nu_\ell(t) \leq 1$$
$$local(\nu,X) = (\forall t \in X : \nu_\ell(t) \leq \nu_u(t)) \wedge (\forall t \notin X : \nu_\ell(t) = 0)$$

We now provide three characterisations of consistency for IMCs: $Cons$ contains those states whose set of $Cons$-successors are locally consistent. States in $Cons_n$ can perform n consistent steps in a row. The third definition will be needed for PIMCs.

Definition 3. *Let $\mathcal{I} = (T, t_0, \varphi)$ be an IMC. We define consistency as:*
co-inductive: *the largest set $Cons \subseteq T$ satisfying $\forall t : t \in Cons \equiv LC(\varphi(t), Cons)$.*
n-recursive: $t \in Cons_n \equiv n > 0 \Rightarrow LC(\varphi(t), Cons_{n-1})$
n-recursive (X): $t \in \mathcal{C}ons_n^X \equiv n > 0 \Rightarrow \exists X : LC(\varphi(t), X) \wedge X \subseteq \mathcal{C}ons_{n-1}^X$

Theorem 1. *Let $\mathcal{I} = (T, t_0, \varphi)$ be an IMC.*

1. *$t_0 \in \mathcal{C}ons$ if and only if $\mathcal{M} \models \mathcal{I}$ for some MC $\mathcal{M} = (S, s_0, \mu)$.*
2. *For all $t \in T$, $t \in Cons$ if and only if $\forall n \in \mathbb{N} : t \in Cons_n$.*
3. *For all $t \in T$, $t \in Cons_n$ if and only if $t \in \mathcal{C}ons_n^X$.*
4. *If all simple paths from t have length $\leq m$, then $Cons_m(t) \equiv \forall n : Cons_n(t)$.*

The full version of this paper will contain the proofs, which have been checked in the interactive theorem prover PVS, based on higher-order classical logic. Here we remark that Theorem 1(2) essentially requires the assumption that $\varphi(t)$ has finite support ($\forall t$).

3 Algorithms for Consistency Checking and Synthesis

Theorem 1(1) leads to the backward Algorithm 1 to check IMC consistency (cf. Appendix A), while Thm. 1(2) justifies the forward Algorithm 2. Thm. 1(3) uses finite quantifications only, so it provides a basis to generate a consistency constraint over the parameters as a nested conjunction/disjunction over linear inequalities, as in Alg. 3. Finally, Thm. 1(4) lowers the upper bound on n from $|T|$ [6] to the length of the longest simple path. For instance, in binary trees the length of longest path is $\log(|T|)$.

Theorem 2. *Let (S, s_0, φ) be an IMC, with maximal outdegree d.*

1. *Algorithm 1 checks consistency in time $\mathcal{O}(|\varphi| \cdot d)$.*
2. *Algorithm 2 checks consistency in time $\mathcal{O}(|S| \cdot |\varphi|)$.*
3. *Algorithm 3 synthesises consistency constraints in time $\mathcal{O}(|S|^3 \cdot 2^d)$.*

We prototyped Algorithm 3 as a Constraint Logic Program (linear arithmetic over reals). Appendix A contains a direct implementation in SWI-prolog over clp(R). It uses backtracking to enumerate all subsets X. The constraint solver is used to prune inconsistent computations early. However, note that this corresponds to unfolding the constraints to a disjunctive normal form, so the polynomial time bound will be lost.

4 Conclusion

The algorithms can be further improved by storing intermediate results, pruning irrelevant parts of the computation earlier, and treating strongly connected components in isolation. We are implementing the algorithms, in order to run them on realistic PMCs from Prism [1] and IMCs from Tulip [2]. We would like to study the scalability of parallel synthesis algorithms as well. Finally, it would be interesting to generalise our results to Parametric Interval Markov Decision Processes, thus generalising results for APA [5].

References

1. Prism – probabilistic model checker. `http://www.prismmodelchecker.org`
2. Tulip – Interval Markov Chains. `http://tulip.lenhardt.co.uk/index.jsp`
3. Daws, C.: Symbolic and Parametric Model Checking of Discrete-Time Markov Chains. In: ICTAC'04. pp. 280–294 (2004)
4. Delahaye, B.: Consistency for Parametric Interval Markov Chains. In: SynCoP'15. pp. 17–32 (2015)
5. Delahaye, B., Katoen, J., Larsen, K.G., Legay, A., Pedersen, M.L., Sher, F., Wasowski, A.: Abstract probabilistic automata. Information and Computation 232, 66–116 (2013)
6. Delahaye, B., Lime, D., Petrucci, L.: Parameter Synthesis for Parametric Interval Markov Chains. In: VMCAI'16. pp. 372–390 (2016)
7. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: LICS'91. pp. 266–277 (1991)

A Algorithms and SWI Prolog-clp(R) implementation

Algorithm 1 $ConsBwd(s)$

Require: Initialise $t.cons := \top$

```
Q := S
while Q ≠ ∅ do
    pick s from Q
    X := {t∈sup(φ(s)) | t.cons=⊤}
    if ¬LC(φ(s), X) then
        s.cons := ⊥
        for all t s.t. φ(t, s) > 0 do
            if t.cons = ⊤ then
                Q := Q ∪ {t}
            end if
        end for
    end if
end while
```

Algorithm 2 $ConsFwd(m)(s)$

Require: Initialise $t.cons := 0$

```
if m ≤ s.cons then
    return true
end if
C := ∅
for all t ∈ sup(φ(s)) do
    if ConsFwd(m − 1)(t) then
        C := C ∪ {t}
    end if
end for
if LC(φ(s), C) then
    s.cons := m
    return true
else
    return false
end if
```

Algorithm 3 $Synth(m)(s)$

Require: Initialise $t.cons[m] := \bot$

```
if s.cons[m] ≠ ⊥ then
    return s.cons[m]
else
    L := sup(φ(s))
    D := false
    for all X ⊆ L do
        C := LC(φ(s), X)
        for all t ∈ X do
            C:=C∧Synth(m−1)(t)
        end for
        D := D ∨ C
    end for
    s.cons[m] := D
    return D
end if
```

```
% IMC is a list of state distributions, each state distribution
% is of the form state(source,[ trans(low,high,target), ...])

:- use_module(library(clpr)).
distribution(S,Dist,IMC) :- member(state(S,Dist),IMC).

lc(Dist,Set) :- low(Dist,Set), up(Dist,Set), local(Dist,Set).
low(Dist,Set) :- low(Dist,Set,Expr), {Expr =< 1}.
low([],_,0).
low([trans(A,_,State)|T],Set,A+B) :-
    member(State,Set),!,low(T,Set,B).
low([_|T],Set,B) :- low(T,Set,B).
up(Dist,Set) :-
    up(Dist,Set,Expr), {Expr >= 1}.
up([],_,0).
up([trans(_,A,State)|T],Set,A+B) :-
    member(State,Set),!,up(T,Set,B).
up([_|T],Set,B) :- up(T,Set,B).
local([],_).
local([trans(L,U,State)|T],Set) :-
    member(State,Set),!, {0=<L,L=<U,U=<1}, local(T,Set).
local([trans(L,_,_)|T],Set) :-
    {L=0}, local(T,Set).

subsets([],[]).
subsets(L,[_|K]) :- subsets(L,K).
subsets([A|L],[trans(_,_,A)|K]) :- subsets(L,K).
cons(0,S,IMC) :- !.
cons(N,S,IMC) :- distribution(S,Dist,IMC), subsets(Succ,Dist),
                 lc(Dist,Succ),N1 is N-1,consSucc(N1,IMC,Succ).
consSucc(_,_,[]).
consSucc(N,IMC,[S|Succ]) :- cons(N,S,IMC), consSucc(N,IMC,Succ).
```

Parameter Synthesis: Algorithm 3 in SWI-prolog with CLP(R).

```
example([P,Q,R],
 [ state(0,[trans(P,P,1),trans(Q,Q,2)]),
   state(1,[trans(0,1,1)]),
   state(2,[trans(0,R,3),trans(0.25,R,4)]),
   state(3,[trans(P+Q,0.8,5)]),
   state(4,[trans(0,1,4)]),
   state(5,[trans(0,1,5)])
 ]).

?- example([P,Q,R],PIMC),cons(3,0,PIMC).
P=0, Q=R=1 ;
P=1, Q=0   ;
P+Q=1, R=1 ;
no.
```

Example run, computing $Cons_3(0)$ on the PIMC (c) of the running example. Prolog synthesizes three answers. Note that the first answer is subsumed by the third answer. So the complete parameter space for which (c) is consistent is:

$$(p = 1 \wedge q = 0) \vee (p + q = 1 \wedge r = 1)$$

Towards a High-Level Model Checking Language: Object-orientation, Data Structures and Local Variable Pruning

Paolo Milazzo, Giovanni Pardini, and Giovanna Broccia

Dipartimento di Informatica, Università di Pisa
{milazzo,pardinig,broccia}@di.unipi.it

Abstract. We propose Objective/MC, a high-level object-oriented language for the modelling of finite-state systems. The language features and data structures are selected with the aim of making modelling a friendly task and of enabling the translation of models into compact transition systems, amenable to efficient verification via Model Checking.
At the moment, we have defined the imperative core of Objective/MC, developed a local variable pruning technique and developed a compiler of the core language into the input language of the PRISM probabilistic model checker. As future work, we plan to add concurrency, data structures, object-oriented features and probabilistic/stochastic features.

1 Motivations

Modelling is often a challenging task. It requires: (i) understanding the mechanisms that govern the dynamics of the system, (ii) performing suitable abstractions in order to express such mechanisms in a concise way, and (iii) constructing an unambiguous representation at the chosen abstraction level. In the case of model checking, the system dynamics is often formalized as a transition system (or *Kripke structure*). The way in which a transition system is expressed in the input languages of model checking tools can vary significantly from tool to tool. In many cases the input language allows transitions to be specified as updates of the system's variables to be performed when certain conditions are satisfied (as if-then clauses or rewrite rules). However, if-then clauses and rewrite rules are often too low level as a modelling paradigm to express systems' mechanisms in a natural way. They can lead to huge and complex descriptions that are difficult to read and error-prone.

In this paper we describe the proposal of Objective/MC, a new language for the specification of system models to be analysed by means of model checking.

2 The Objective/MC Modelling Language

2.1 Desired language features

Objective/MC aims to be a general purpose modelling language. It will have an imperative semantics with standard control flow constructs. These are the main

features we aim to include in the language (preliminarly discussed in [2] with, actually, game modelling in mind):

Concurrency We plan to include the possibility of defining parallel processes that can synchronize on specific labels (CSP-like).

Object-oriented features (with statically allocated objects) We plan to include class definitions with standard Java-like inheritance. Objects will be defined as statically allocated global variables (the will be no `new` operator).

Probabilistic/stochastic choices We plan to include a selection statement (like a C `switch`) in which cases are chosen on the basis either of a user-defined probability distribution or of stochastic rates.

Arrays and graphs (statically allocated) We plan to include (fixed size) arrays and a primitive notion of graph (with a fixed structure). Moreover, we wish to include array filtering and graph exploration primitives.

The presence of data structures in the language calls for the presence of local variables also (to be used, e.g., as array indexes). One of the main characteristics of the language is that it makes a very different treatment of global and local variables. Global variables are assumed to actually describe the state of the modelled system, whereas local variables are only used to ease the specification of the system's internal mechanisms, and can be pruned from the model.

The semantics of the language is hence a transition system whose states correspond to different evaluations of the global variables and transitions correspond to the assignments to global variables as specified in the Objective/MC model. In addition, a hidden global variable `pc` is considered in order to put sequential assignments in the correct order in the transition system.

2.2 Current State of Development

At the moment, we have defined the imperative core of Objective/MC, with bounded integers and booleans as data types, arrays as the only data structure, procedures (void functions) and a basic notion of class (analogous to a C struct). The syntax of the language (apart from classes) is presented through the example of job processor scheduler in Figure 1. Objective/MC is a C-like language. The non-standard constructs are the bounded integer type (denoted `int(n..m)`), the `times` loop (that iterates a fixed number of times), the `run` block (analogous to the `main` function in C), the `select(n,m)` operator (representing the non-deterministic choice of a value in $[n, m]$) and the `yield` statement (to be used in the body of a while loop when no global variable assignments are reachable).

An implementation of the compiler of Objective/MC into the modelling language of the PRISM model checker is available [1]. The compiler transforms the model in order to prune all local variables. Hence, the generated transition system will consist of transitions that correspond to the updates of only the global variables. For the sake of local variable pruning the language includes some syntactic constraints: (i) a local variable cannot be read after a global variable it depends upon may have been modified, (ii) a local variable declared outside of

```
1  int(0..10)[5] P;          // array of five processors
2  int(0..10) nextjob;       // duration of the next job
3  void execute_step() {     // decreases the load of each processor by one unit
4      int(0..4) i = 0;
5      times (5) {
6          if (P[i] > 0) { P[i] = P[i] as int(1..10) - 1; }
7          i = (i < 4 ? i + 1 : i) as int(0..4); }
8  }
9  run {
10     while (true) {
11         execute_step();
12         if (nextjob == 0) nextjob = select(0,3);
13         int(0..4) min_idx = 0;
14         int(0..4) i = 1;
15         times (4) {
16             if (P[i] < P[min_idx]) min_idx = i;
17             i = (i < 4 ? i + 1 : i) as int(0..4); }
18         if (P[min_idx] + nextjob <= 10) {
19             P[min_idx] = (P[min_idx] + nextjob) as int(0..10);
20             nextjob = 0;
21         } else yield;
22     }
23 }
```

Fig. 1. An Objective/MC model of a job processor scheduler.

a `while` loop cannot be updated inside the loop body, and (iii) each iteration of a `while` loop must include at least one global variable assignment (or a `yield`).

Constraints (i) and (ii) may look very restrictive. However, they deal with local variables and are necessary in order to ensure that the value of each local variable can be at any time computed by the current values of the global variables. A model that does not satisfy one of the two constraints can be trivially fixed by turning the problematic local variable into a global variable.

The local variable pruning technique implemented in the Objective/MC compiler consists in a sequence of transformations applied to the Control Flow Graph of the model, as exemplified in the diagram in Fig. 2. The output of the compiler is a model expressed in the input language of the PRISM model checker, that is then used to build the transition system. The choice of PRISM as the target model checking tool is related with the aim of including probabilistic/stochastic features in the language. Moreover, PRISM has been chosen among probabilistic/stochastic model checkers since it is one of the most used tools and since we are familiar with it. In principle, the Objective/MC compiler could be extended in order to deal with more than one translation targets.

More detailed information about the current definition of Objective/MC and the local variable pruning technique can be found in [3].

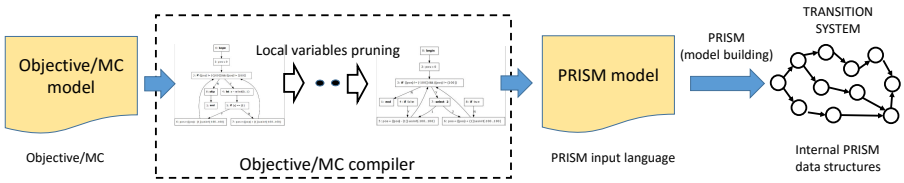


Fig. 2. Schematic representation of the translation of an Objective/MC model into a transition system. The Objective/MC compiler computes the control flow graph of the model and applies a number of transformations to it aimed at pruning local variables. The obtaned graph is then translated into a model in the PRISM input language. Finally, the transition system is built (and analysed) by using the standard PRISM facilities.

References

1. ObjMC: The Objective/MC compiler, `http://pages.di.unipi.it/pardini/ObjMC`
2. Broccia, G.: Un Domain Specific Language per la modellazione e l'analisi di giochi. Master's thesis, University of Pisa, Italy (2015), `https://etd.adm.unipi.it/t/etd-09032015-150400/`
3. Pardini, G., Milazzo, P.: A high-level model checking language with compile-time pruning of local variables. In: Proc. of DataMod 2016. Lecture Notes in Computer Science, vol. 9946. Springer (in press), `http://pages.di.unipi.it/datamod/wp-content/uploads/sites/8/2016/06/DataMod_2016_paper_4_Pardini.pdf`

Formal Verification of Non-Linear Hybrid Systems Using Ariadne

Luca Geretti[1], Davide Bresolin[2], and Tiziano Villa[1]

[1] Università di Verona, Verona, Italy {luca.geretti, tiziano.villa}@univr.it
[2] Università di Bologna, Bologna, Italy davide.bresolin@unibo.it

1 Introduction

Hybrid systems are dynamical systems that exhibit both a discrete and a continuous behaviour. In order to model and specify hybrid systems in a formal way, the notion of *hybrid automata* has been introduced [1]. Intuitively, a hybrid automaton is a "finite-state automaton" with continuous variables that evolve according to dynamics characterizing each discrete state (called a *location*). However, such model enables the definition of monolithic systems only, with no support for the decomposition of the state space. On the other hand, *hybrid I/O automata* [10] extend hybrid automata to support the composition of multiple (simpler) automata that operate on subsets of the continuous state.

Of particular importance in the analysis of hybrid (I/O) automata is the computation of the *reachable set*, i.e., the set of all states that can be reached under the dynamical evolution starting from a given initial state set. Many approximation techniques and tools to estimate the reachable set have been proposed in the literature. Most of the available software packages, like PhaVer [8] and SpaceEx [9], are limited to affine dynamics. Others, like HSOLVER [13] can handle more complex dynamics, but are based on abstractions which prevent the designer from observing the dynamical evolution of the system. Finally, most of these tools use a numerical engine which does not guarantee correctness in respect to numerical rounding.

To overcome such limitations, we recently proposed a development environment for hybrid systems verification, called Ariadne [4], which differs from existing tools by being based on the theory of computable analysis [6]. Such theory provides a rigorous mathematical semantics for the numerical analysis of dynamical systems, suitable for implementing formal verification algorithms, enabling tasks such as dominance checking [3] and parametric controller synthesis. However, in the following we will discuss about verification of *safety properties*, which is currently the main focus of Ariadne.

2 Safety Verification with Ariadne

Suppose we wish to verify that a safety property φ holds for a hybrid automaton H; in other words, that φ remains true for all possible executions starting from a set X_0 of initial states. Then we only need to prove that $\mathit{ReachSet}_{\mathcal{A}}(X_0) \subseteq$

$\mathrm{Sat}(\varphi)$, where $\mathrm{Sat}(\varphi)$ is the set of states where φ is true. Unfortunately, the reachability problem is not decidable in general [1]. Nevertheless, formal verification methods can be applied to hybrid automata: suppose we can compute an over-approximation $S \supseteq \mathit{ReachSet}_{\mathcal{A}}(X_0)$. Then if S is a subset of $\mathrm{Sat}(\varphi)$, then so is the reachable set and the automaton H respects the property. Conversely, if we can compute an under-approximation $S \subseteq \mathit{ReachSet}_{\mathcal{A}}(X_0)$ that turns out to contain at least one point outside $\mathrm{Sat}(\varphi)$, we have proved that H does not respect the safety property φ.

Finite-time evolution is obtained by integration of a (non-linear) vector field $\dot{X} = f(X)$: starting from a set X_i and choosing an integration step t_i, we obtain the next set X_{i+1} and the corresponding partial reached set $R_{i,i+1}$. This represents a continuous step of evolution, as opposed to a discrete step where a transition is activated, changing the discrete state and possibly the continuous state. Evolution up to a time T therefore becomes a simple sequence of continuous and discrete transitions.

In order to account for numerical errors, all the computed sets are over-approximations of the actual sets. These over-approximations are called *enclosures*, and are represented as Taylor expansions to offer an accurate and flexible representation of the included set. In our theory, evolution can be computed according to two different semantics:

- *upper semantics*: if we evolve the system for a finite time, the set of points that we obtain is a superset of the reachable set;
- *lower semantics*: if we evolve the system for a finite time, each point that we obtain has a bounded distance ε to a point of the reachable set.

Lower semantics in particular is a *bounded approximation*, rather than an under-approximation of the reachable set, which is not computable in general. If we prove that the evolution under lower semantics reaches a point outside $\mathrm{Sat}(\varphi)$ by at least ε, then $\mathit{ReachSet}_{\mathcal{A}}(X_0)$ has a point outside $\mathrm{Sat}(\varphi)$ and the property φ is not satisfied.

While the computation of evolution for a finite time is straightforward, the same cannot be said for the case of infinite time. The termination condition for infinite time evolution is that no additional reachable region can be obtained after an evolution step. Such check requires to perform inclusion test, intersection and subtraction of enclosures. Unfortunately, the subtraction and inclusion tests are not computable, while the intersection is very inefficient. Consequently, enclosures are *discretised* onto a set of non-overlapping *cells* of a *grid*, internally represented by means of a binary decision diagram.

Summarizing, given a hybrid automaton $\mathcal{A}$, and an initial set X_0, Ariadne can compute two kinds of approximations to the reachable set, for both finite and infinite-time evolution:

- An *outer approximation* O of the reachable set using upper semantics. Formally, a *closed set* O such that the *closure* of $\mathit{ReachSet}_{\mathcal{A}}(X_0)$ is strictly contained in the *interior* of O.

– An ε-*lower approximation* L_ε of the reachable set using lower semantics. Formally, an *open set* L_ε where for every point $x \in L_\varepsilon$ there exists a point $y \in \mathit{ReachSet}_{\mathcal{A}}(X_0)$ such that $|x - y| < \varepsilon$.

In the case of O, the evolution can be performed either in the *forward* or *backwards* direction. On the contrary, backwards evolution for L_ε, while computable, has not been implemented, since lower semantics causes backwards transitions to yield very coarse enclosures. This results in a large bound on the approximation, which is ineffective for reachability analysis.

3 Ongoing Work

Formal verification of non-linear hybrid systems is still in its infancy, but tools like Ariadne show promising results on industrial-strength case studies. Experimentally, we are applying the tool to model and verify robotic surgery tasks [11, 5]. In such a field, formally proving safety properties is of the utmost importance in order to implement reliable controllers for semi-automatic surgery operations.

From the technical viewpoint, we have discussed how Ariadne is able to compute both outer approximations (sets O) and bounded approximations (sets L_ε) of the reachable set, allowing to observe the system evolution with fine detail. However, these features are responsible for bottlenecks, in particular with respect to scalability and accuracy of the approximations. To overcome the current limitations of the tool, we are working on the following improvements:

– improve automated tuning of the accuracy parameters in order to control the over-approximation error; this applies to the integration step and the maximum Taylor expansion order;
– improve heuristics for discretization events, to minimize convergence for infinite time evolution;
– address scalability by means of counterexample-based abstraction refinement techniques [2].

In addition, we plan to extend the model to *differential inclusions*, based on the work of [14], to enable the representation of noisy inputs. The obvious application of differential inclusions is to model uncertainty, but they are needed also to enable *contract-based design*[12]: given a complex system, we can replace the actual input of an automaton with an input having *partially defined behavior*. Such decoupling of automata ultimately allows to analyze automata separately, trading-off system complexity for verification accuracy.

Further information on the Ariadne framework can be found in [7] about functional calculus, [3] regarding the reachability routines, and [4] for advanced verification strategies. We are currently preparing a first official release of the library, providing user documentation and tutorial resources.

References

1. R. Alur, C. Courcoubetis, T. A. Henzinger, and P. H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer, 1993.
2. R. Alur, T. Dang, and F. Ivančić. Counterexample-guided predicate abstraction of hybrid systems. *Theor. Comput. Sci.*, 354(2):250–271, 2006.
3. L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti, and T. Villa. Ariadne: Dominance checking of nonlinear hybrid automata using reachability analysis. In *Reachability Problems*, volume 7550 of *LNCS*, pages 79–91. Springer, 2012.
4. L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti, and T. Villa. Assume-guarantee verification of nonlinear hybrid systems with ARIADNE. *Int. J. Robust. Nonlinear Control*, 24(4):699–724, 2014.
5. D. Bresolin, L. Geretti, R. Muradore, P. Fiorini, and T. Villa. Formal verification of robotic surgery tasks by reachability analysis. *Microprocess. Microsyst.*, 39(8):836–842, 2015.
6. P. Collins. Semantics and computability of the evolution of hybrid systems. *SIAM J. Control Optim.*, 49:890–925, 2011.
7. P. Collins, D. Bresolin, L. Geretti, and T. Villa. Computing the evolution of hybrid systems using rigorous function calculus. In *Proc. 4th IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'12)*, volume 45(9) of *IFAC Proceedings Volumes*, pages 284–290. Elsevier, 2012.
8. G. Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *Int. J. Softw. Tools Technol. Transf.*, 10:263–279, 2008.
9. G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 379–395. Springer, 2011.
10. N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Inform. Comput.*, 185(1):105–157, 2003.
11. R. Muradore, D. Bresolin, L. Geretti, P. Fiorini, and T. Villa. Robotic Surgery: Formal Verification of Plans. *IEEE Robot. Autom. Mag.*, 18(3):24–32, 2011.
12. P. Nuzzo, A. L. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa. A platform-based design methodology with contracts and related tools for the design of cyber-physical systems. *Proc. IEEE*, 103(11):2104–2132, 2015.
13. S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. *ACM Trans. Embed. Comp. Syst.*, 6(1), 2007.
14. S. Zivanovic and P. Collins. Numerical solutions to noisy systems. In *Proc. 49th IEEE Conference on Decision and Control (CDC'10)*, pages 798–803, 2010.

Automation of Test Case Assessment in SPLs — Experiences and Open Questions

Josh Taylor[1], Alexander Knapp[2],
Markus Roggenbach[1], and Bernd-Holger Schlingloff[3]

[1] Swansea University, Wales, United Kingdom
joshdt001@gmail.com
[2] Universität Augsburg, Germany
[3] Humboldt Universität and Fraunhofer FOKUS, Germany

Abstract. This research idea turns a theory for test case assessment in the model-based development of multi-variant systems, so called Software Products Lines (SPL), into practice. To this end, we provide a tool chain for automated test case assessment, validate it on the example of a coffee machine product line, and finally, successfully apply it to "The Body Comfort System" product line from the automotive domain.

1 Introduction

The concept of a software product line originates by the work of D. Parnas [6]. It has gained much attention by the research and consultancy of the Carnegie Mellon University Software Engineering Institute [1,5]. According to the CMU-SEI definition, "a software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"[4]. SPLs are abundant in today's software-intensive systems: most electronic control units, e.g., in cars or trains, come in multiple variants, as well as consumer products such as coffee machines, dishwashers, mobile phones, etc.

A challenge common to the development of such systems is that their built-in software is similar, but not identical in all products; there are slight differences according to the features exhibited by a particular product. Sources of variability include planned diversity for different user groups, evolution and enhancement of products, and re-use of modules from one product in another one.

SPL engineering addresses this challenge. The main goal of SPL development is the strategic re-use of software artefacts. There have been various approaches to re-use: by copy and paste, macros, subroutines, modules, objects, components and services. The common problem in all of these approaches is that re-use increases the risk of errors. Therefore, quality assurance for SPLs is of utmost importance.

Given an SPL and a software test suite, not all tests apply to all possible products. Often, it is clear from the context whether a test can be executed with a product or

[4] CMU Software Engineering Institute: Product Line Web Page. http://www.sei.cmu.edu/productlines/(2015/01)

not. However, there are cases where this is not obvious: consider the case that a certain additional feature blocks some behaviour present in the 'standard' product. In general, for a sufficiently expressive specification language the problem of test case assessment is undecidable.

In [2,3], some of the authors presented a theory for test case assessment in the model-based development of multi-variant systems. This deals with both positive (green) and negative (red) test cases, and introduces a third colour (yellow) for test cases whose outcome is not determined with a given product model. This means that it is needless to execute them with products based on this model. This approach thus allows to assess and select those test cases from a universal test suite which are relevant for a given product.

2 Tool Chain

Our tool chain can be separated into two main parts: the modelling and the test colouring part. It re-uses existing tools, where our contribution lays in the automation of their co-operation.

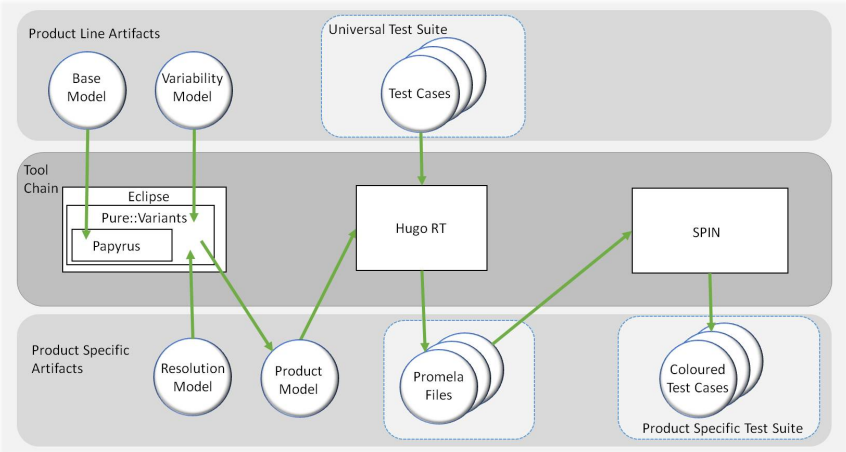


Fig. 1. Tool chain for test case colouring for SPLs

For the modelling part, we completely rely on existing technology. We utilise the UML model editor PAPYRUS[5] to define a base model (consisting of class diagrams, state machines, and a composite structure diagram) and PUREVARIANTS[6] to deal with variability, namely to create a feature model and a variability model (above shown as one integrated variability model). In PUREVARIANTS, the user then realises a resolution model by selecting which features to include. This results in a product model in UML.

[5] www.eclipse.org/papyrus/(2016/08)
[6] www.pure-systems.com/products/pure-variants-9.html(2015/06)

The test colouring uses two tools, namely HUGO/RT[7] and the SPIN model-checker[8]. Here, we extended HUGO/RT to automatically take a list of test cases and a product model in order to translate these into executable PROMELA code, which can be checked by SPIN. Each applicable test case results in a separate PROMELA file. This file is then processed with SPIN in up to three passes for different reachability properties of the encoded automaton.

Validation: Coffee Machine. In [2,3], some of the authors presented a small Software Product Line of an automated coffee machine. A number of test cases were presented, and manually coloured against various products. Our tool chain was able to produce all test colourings as predicted – except one, where it turned out after careful analysis that the manual process, due to human error, had resulted in a wrong colouring. Overall, this result increases trust in our tool chain. Furthermore, it demonstrates the need for tools: even for a relatively simple test case human error occurred.

3 An Industrial Size Case Study: Body Comfort System

As a case study of industrial size, we have chosen the "Body Comfort System Case Study" developed by Lity et al. [4] in order to demonstrate their delta-oriented SPL test method. The system is made up of a mandatory interface, a mandatory door system and an optional security component. Lity et al. give in total 64 test cases represented as Message Sequence Charts. We have represented a significant subset of these test cases in our approach, i.e., we encoded them as sequences of occurrences and dispatches of UML events and were running them through our tool chain in order to assess their colour. This resulted in having multiple test cases for each product model. Here, we studied all 18 product models considered in [4].

Looking at each product model, the first question was if a test case was applicable, i.e., we checked if the alphabet of events of the test cases was a subset of the alphabet of the product model. Here it turned out that the notion of applicability in Lity et al. is the same as ours. Furthermore, in the average, only about 1/3 of the test cases are applicable to the product models. If a test was applicable, we were running our test colouring procedure.

Concerning performance, the colouring time depends on a number of different aspects. Time is growing with the length of a test case: short test cases consisting of 3 or 4 events take 1–2 seconds, while longer ones of about 20 events can take 3–4 minutes. Also, time grows with the number of different state machines involved within the test case. Finally, time grows with the number of transitions in these state machines. Overall, colouring time is below 10 minutes per test case. In this respect, we consider our approach to be practically feasible, as most test cases can be coloured fast, and about two thirds of them are excluded due to the basic applicability criterion.

Concerning colouring, it turns out that all test cases from [4] are actually green ones, i.e., they express desired behaviour. It does not come as a surprise that there are

[7] `www.informatik.uni-augsburg.de/en/chairs/swt/sse/hugort/`(2016/08)
[8] `spinroot.com/spin/whatispin.html`(2016/08)

no yellow test cases: the UML models are close to implementation where all relevant design decisions have been resolved. That there are no red test cases is a question of methodology: apparently, Lity et al. were concentrating on demonstrating expected behaviour rather than trying to demonstrate that certain error situations have been avoided.

4 Conclusions and Open Questions

We have successfully implemented a tool chain for SPL test assessment according to the theory presented in [2,3]. We validated the tool chain on a simple example, and showed that the developed implementation scales up. However, there are a number of research questions arising:

1. What are appropriate coverage metrics for SPL models and coloured test cases?
2. Currently we are focusing on individual products for colouring. However, in principle, it should be possible to colour classes of products. This would reduce colouring time. How to refine the base model adequately?
3. As we obtain only green test cases in our case study, we presume that Lity et al. were performing testing for functionality. How to extend this to testing for safety, i.e., "show whether or not each software module performs its intended function and does not perform unintended functions" (IEC 61508)? A first step towards this would be to formulate safety properties as test objectives, that one then would turn into red test cases.

Overall, it is still an open question of how to relate our approach with incremental development methods such as step-wise refinement and software evolution.

References

1. P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.
2. A. Knapp, M. Roggenbach, and B.-H. Schlingloff. On the Use of Test Cases in Model-based Software Product Line Development. In S. Gnesi, A. Fantechi, P. Heymans, J. Rubin, K. Czarnecki, and D. Dhungana, editors, *Proc. 18th Intl. Software Product Line Conf. (SPLC'14)*, pages 247–251. ACM, 2014.
3. A. Knapp, M. Roggenbach, and B.-H. Schlingloff. Automating Test Case Selection in Model-Based Software Product Line Development. *Intl. J. Softw. Inform.*, 9(2):153–175, 2015.
4. S. Lity, R. Lachmann, M. Lochau, and I. Schaefer. Delta-oriented Software Product Line Test Models — The Body Comfort System Case Study. Informatik-Bericht 2012-07, v1.2, Technische Universität Carolo-Wilhemina zu Braunschweig, 2015.
5. J. D. McGregor, L. M. Northrop, S. Jarrad, and K. Pohl. Initiating Software Product Lines. *IEEE Softw.*, 19(4):24–27, 2002.
6. D. L. Parnas. On the Design and Development of Program Families. *IEEE Trans. Softw. Eng.*, 2(1):1–9, 1976.