UiO : Department of Informatics
University of Oslo

Proceedings of the PhD Symposium at iFM'17 on Formal Methods: Algorithms, Tools and Applications

(PhD-iFM'17)

Erika Ábrahám and Silvia Lizeth Tapia Tarifa (Eds.)

Research report 470, September 2017

ISBN 978-82-7368-435-6
ISSN 0806-3036

Proceedings of the PhD Symposium at iFM'17 on Formal Methods: Algorithms, Tools and Applications

(PhD-iFM'17)

Erika Ábrahám and Silvia Lizeth Tapia Tarifa (Eds.)

September 2017

Contents

On the Synthesis of Guaranteed-Quality Plans for Robot Fleets in Logistics Scenarios via Optimization Modulo Theory

Francesco Leofante*†

University of Genoa, Italy
RWTH Aachen University, Germany
leofante@cs.rwth-aachen.de

Abstract

Industries are on the brink of widely accepting a new paradigm for organizing production by having autonomous robots manage and optimize the in-factory supply chain. Successful symbolic reasoning methods exist to solve the underlying scheduling problem, however they usually offer no guarantees on the quality of the solution. This work proposes to employ Optimization Modulo Theories (OMT), an extension of SMT solving with optimization capabilities, to synthesize guaranteed-quality schedules for multi-robot systems. For this purpose, the RoboCup Logistics League (RCLL) is used as a realistic testbed.

1 The project

With the advent of Industry 4.0, factories are moving from static process chains towards more autonomy, by introducing robots in their production lines. Given this paradigm shift, the problem of managing and optimizing the in-factory supply chain carried out by (fleets of) autonomous robots becomes crucial.

The RoboCup Logistics League (RCLL) [4] has been proposed as a realistic testbed to study the above mentioned problem at a comprehensible and manageable scale. There, groups of robots need to maintain and optimize the material flow according to dynamic orders in a simplified factory environment.

Though there exist successful symbolic reasoning methods towards solving the underlying scheduling problem [3], a disadvantage of these methods is that they provide no guarantees about the quality of the solution.

A promising solution to this problem is offered by the recently emerging field of Optimization Modulo Theories (OMT), where Satisfiability Modulo Theories (SMT) solving is extended with functionalities towards optimization [6, 7, 9].

In this project we study how OMT solving can be employed to compute optimal strategies for multi-robot systems within the RCLL scope. We frame the underlying scheduling problem as a linear (mixed-)integer problem involving Boolean combinations of constraints over the reals. This can be solved and optimized by SMT solvers with optimization functionalities such as `Z3` [1] and `OptiMathSAT` [8].

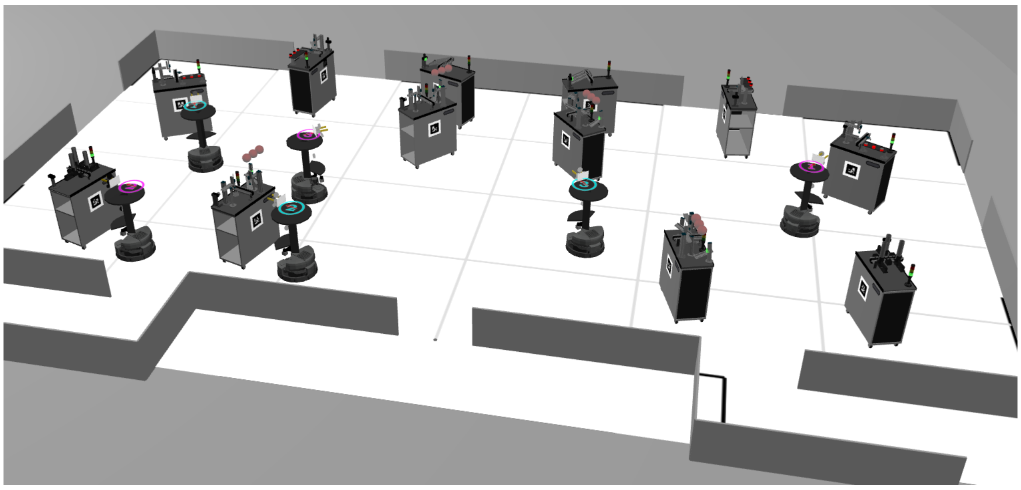


Figure 1: The simulation environment used for our experiments

2 The domain

The example domain chosen for evaluating our approach is based on the Planning Competition for Logistics Robots in Simulation [2] shown in Figure 1. There, the task is to control a team of three robots to transport workpieces among a number of machines. Each machine performs a processing step such as mounting a colored ring or a cap. Orders that denote the products which must be assembled with these operations are posted at run-time and therefore require quick planning and scheduling. Orders come with a delivery time window introducing a temporal component into the problem.

A game in the competition is structured in two phases: (*i*) exploration and, (*ii*) production. This work reports on the exploration phase, during which two teams of robots must roam the environment and determine where the team's own machines are positioned. For this, the playing field is divided into 24 virtual zones, 12 of which belong to each of the two teams. Six of these zones contain machines. Therefore, the task is to efficiently assign the three robots to the 12 zones, identify the zones which contain a machine, and then precisely determine the position of the machine and recognize an industrial light signal for verification. Robots are given 20 minutes to perform exploration (planning time included). This imposes tight timing constraints on the performances of our OMT-based planning module.

3 OMT-based planning and RCLL

We integrated our OMT-based planning module in the RCLL planning framework [5]. The overall architecture is depicted in 2. The CLIPS[1] executive controls the overall execution. At a suitable time (when the game enters the appropriate phase), it triggers the OMT solving process to come up with a plan to explore the machines and encodes the available knowledge in a message. The solver queries travel times of the position for exploration from a navigation

[*] Joint work with E. Ábrahám, G. Lakemeyer, T. Niemueller, A. Tacchella.

[†] Part of this work is supported by the ICT project house Foundations of a Digitized Industry, Economy, and Society of RWTH Aachen University.

[1] CLIPS is a rule-based production system using forward chaining inference [10].

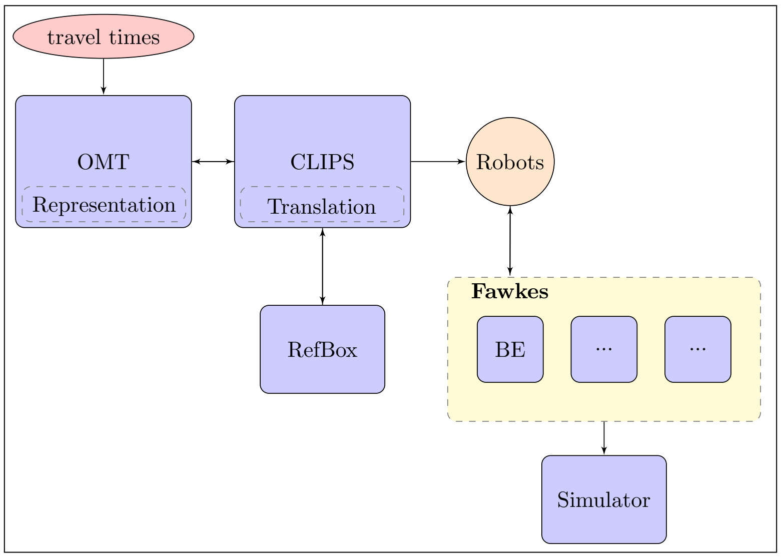


Figure 2: The overall architecture.

graph and uses a domain model phrased suitably for OMT. The result is then represented as a plan and sent to the CLIPS executive, which must translate it into a native representation for execution. This plan is then synchronized with all robots, which then execute their respective partial plans by invoking the appropriate basic behaviors through the behavioral and functional components of the Fawkes[1] software framework.

References

[1] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proc. of TACAS'08*, pages 337–340, 2008.

[2] Tim Niemueller, Erez Karpas, Tiago Vaquero, and Eric Timmons. Planning competition for logistics robots in simulation. In *Proc. of PlanRob@ICAPS'16*, 2016.

[3] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. Incremental task-level reasoning in a competitive factory automation scenario. In *Proc. of AAAI'13 Spring Symposium*, 2013.

[4] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. The RoboCup Logistics League as a benchmark for planning in robotics. In *Proc. of PlanRob@ICAPS'15*, 2015.

[5] Tim Niemueller, Gerhard Lakemeyer, Francesco Leofante, and Erika Ábrahám. Towards CLIPS-based task execution and monitoring with SMT-based decision optimization. In *Proc. of PlanRob@ICAPS'17*, to appear.

[6] Robert Nieuwenhuis and Albert Oliveras. On SAT modulo theories and optimization problems. In *Proc. of SAT'06*, pages 156–169, 2006.

[7] Roberto Sebastiani and Silvia Tomasi. Optimization modulo theories with linear rational costs. *ACM Trans. Comput. Log.*, 16(2):12:1–12:43, 2015.

[8] Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A tool for optimization modulo theories. In *Proc. of CAV'15*, pages 447–454, 2015.

[9] Roberto Sebastiani and Patrick Trentin. Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions. In *Proc. of TACAS'15*, pages 335–349, 2015.

[1]Fawkes is a component-based software framework for robotic real-time applications.

[10] Robert M. Wygant. CLIPS: A powerful development and delivery expert system tool. *Computers & Industrial Engineering*, 17(1–4), 1989.

On the community structure of SAT-BMC problems

Xavier Gillard and Charles Pecheur

Université Catholique de Louvain, Louvain-la-Neuve, Belgium
{xavier.gillard,charles.pecheur}@uclouvain.be

Abstract

A common belief says that modern SAT solvers are efficient because of their ability to exploit structural properties of industrial problems. However, we only have a limited understanding of what is covered by this 'structure'. A recent hypothesis suggests the community structure as a candidate definition for that notion. Our paper proposes a tool helping to understand community structure of SAT instances generated by BMC.

1 Introduction

It is commonly believed in the SAT community that the efficiency of modern SAT solvers stems from their ability to exploit the structure of the original problem. However, that notion of structure is often ill defined and very little is known about the reason solvers perform so well in practice. A recent line of work on SAT suggests that community structure is a good candidate for that 'structure' definition [2, 22, 23]. In that context, community structure is a property reflecting the possibility to split a SAT instance in almost disjoint subproblems. Indeed, the modularity measuring that structure was shown to be strongly correlated with the execution time of CDCL solvers [22]. However, it was proved [21] that some pseudo-industrial instances with a good community structure [2] require an exponential time for all resolution-based solvers. Such instances are therefore hard even for the most sophisticated CDCL solvers. Hence, it is believed that industrial instances exhibit additional features which are not fully explained by the community structure. In this paper we present a tool we developed to help understanding the community structure of industrial SAT problems. In particular, we focus on instances generated by SAT-based Bounded Model Checking (BMC) [9], a well established formal verification method based on SAT. Our paper is structured as follows: we begin with a brief introduction to community structure, modern SAT solvers and BMC. Then we present our tool and discuss some early observations we made. Finally, we propose research perspectives based on our prospective results.

2 Background

Community structure. Informally, the community structure of a graph is a structural property of that graph that decomposes it into *almost disjoint* sets of nodes. More precisely, these sets of nodes are called *communities* and consist of a cluster of nodes that are densely connected with one another and sparsely with the rest of the graph. There exist efficient algorithms capable of decomposing a graph into communities (e.g. the Louvain Method [11]).

When it comes to analyzing an instance of SAT problem, the community structure is generally interpreted over the variable incidence graph (VIG) derived from the boolean formula encoding the problem. Such a VIG is built from the CNF representation of the formula by having one vertex per *variable* of the problem and one edge between two variables if they appear together in one same clause (irrespective of their literal polarity).

Modern SAT solvers. Modern SAT solvers such as MiniSat [16] or Glucose [7] implement a Conflict Driven Clause Learning approach. These build upon the traditional DPLL [15] algorithm to implement a resolution refutation proof system, enhanced with the ability to learn new (redundant) clauses. All these additional clauses are guaranteed to derive from the original problem and are learned by the solver upon conflicts [26]. Additionally, CDCL solvers enrich their resolution algorithm with a set of techniques and heuristics aiming at maximizing the information gained from the learning process [28, 20, 24, 19, 5, 4, 6, 3, 10], which contributes to further pushing the limits of the set of tractable SAT instances.

Meanwhile, the question of why CDCL solvers work so well in practice remains largely unanswered. The traditional approach consists in a proof-complexity-theoretic argument saying that *because* of its capability to learn and remember conflict clauses, CDCL implements a proof system (General Resolution Refutation) that is up to exponentially more efficient than that of DPLL (Tree Like Resolution Refutation) [8, 18, 24]. However, this argument is of no use in understanding the rationale behind the good performance of such solvers on *industrial problems*[1]. Thus, it is widely believed that CDCL solvers are able to exploit some kind of structure of the underlying problem. Hence the recent interest in the community structure of industrial problems.

Bounded Model Checking (BMC). Model checking consists in verifying that a temporal formula ϕ expressed in LTL holds for all the traces of a model M. Bounded Model Checking [9] limits to traces shorter than k (written $M \models_k \phi$). To do so, a BMC model checker tries to disprove $M \models_k \phi$ by finding a trace of length k violating the property ϕ. Concretely, the refutation process is done symbolically by means of a reduction to SAT. In other words, a boolean formula $[\![M, \neg\phi]\!]_k$, which is satisfiable iff there exists a violating trace, is fed to some SAT solver. The solution of that satisfiability test determines whether or not $M \models_k \phi$ and yields an appropriate counter-example if one exists. For our matter, the key element to understand about the generation of $[\![M, \neg\phi]\!]_k$ is that it is generated as a sequence of k "blocks of variables" encoding the possible values of the k states of a bounded trace of M.

3 Experiments

A community analysis tool. The tool we developed[2] in order to analyze the community structure of SAT-BMC problem instances builds upon the PyNuSMV framework which has recently been extended to expose SAT capabilities of NuSMV [17]. Thanks to that framework, our tool is able to generate a series of SAT problem instances[3] corresponding to the various formulas submitted to a solver when performing a bounded verification of some given SMV model. Additionally, it uses Igraph [13] to build, manipulate and visualize the VIG associated with each of the aforementioned SAT problems. More interestingly, our tool combines the features of the two libraries to compute the graph modularity, detect the community structure of the VIG and then reconcile the variables with their semantic meaning in the original SMV model. This permits the use of data mining algorithms to detect patterns of semantic information frequently associated in the various communities. To that end, we used the algorithms implemented in Pymining [14] to detect frequent patterns (with relim) and frequent sequences. The

[1]To explain their lack of efficiency on *random instances*, it has been proved that random k-CNF formulas have exponential resolution complexity [1, 2, 12]. Obviously, that tells nothing about the efficiency of CDCL on industrial instances which is our main focus.

[2]Technical details are found on the project page http://bit.ly/2vWr8bM

[3]And dump them to DIMACS CNF format for later analysis if necessary.

information learned that way suggests what the hidden meaning of the communities could be. Moreover, our tool is also able to collect and plot statistics about the succession of generated instances. The combination of all the above features provides a fairly complete and efficient toolkit to analyze the structure of BMC instances. However, our tool slightly suffers from its inability to associate a semantic meaning to the auxiliary variables introduced by the Tseitin transformation [27] used to encode the raw SAT-BMC problem in CNF.

Early results. So far, a full scale empirical analysis of the community structure of BMC instances has not been realized yet. We have analyzed two small models (dining philosophers and a diagnosability) and one real industrial problem (a railway interlocking). We observed that communities tend to represent either the evolution of an object (ie philosopher, fork) over a certain period of time (time frame 1, 2 and 3) or the synchronization of the model at some point of time (all the forks in the 2nd time frame). Surprisingly though, we observed that often the communities tend to group variables from three consecutive time blocks while we would have expected no more than two to reflect the application of the Kripke Structure's transition relation. We haven't yet found any convincing reason explaining that observation, which remains to be confirmed by further experiments.

We also observed that the modularity of the SAT instances generated for known hard problems increases for small values of k and diminishes for higher bounds, which corroborates the results of [22]. Besides that, we also noted that the modularity of SAT instances generated for problems comprising a model only (formula ϕ is vacuously true) seems to saturate. We believe that this phenomenon might be related to the *diameter*[4] of the model.

4 Related work

Analyzing the community structure of industrial SAT instances has only gained interest very recently and is still actively investigated. Therefore, a tool called SatGraf [23] has been developed to analyze the impact of that structure on the CDCL runtime. Even though it shares some of its feature with the tool presented here, SatGraf does not consider the semantic level of the problem. A reflection somewhat similar in its essence to the one presented here has been held by Shishmarev et al. in [25]; although it took place in a completely different setup than the current one. Indeed, their focus wasn't set on the community structure but solely on getting an understanding of the meaning of learned clauses in the context of constraint programming. Hence, their results cannot be lifted to BMC.

5 Perspectives and conclusion

Encouraged by the above early results, we believe that our tool might greatly help in gaining a better understanding of the community structure that arises from SAT-BMC problems. This objective is a necessary step that could help designing BMC variants that best exploit the capabilities of current SAT solvers. Indeed, having a proper understanding of model features that give rise to the communities might serve the point of designing a preprocessor exploiting these features to generate redundant clauses, thereby easing the resolution without having to go through the lengthy conflict-learning process. Additionally, this kind of knowledge might help to scale SAT-BMC horizontally with the design of a semantically grounded heuristic to split the problem before passing it on to a parallel SAT solver.

[4]The smallest k to to ensure that $M \models_k \phi \iff M \models \phi$.

References

[1] Dimitris Achlioptas. Random satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of satisfiability*, chapter 8, pages 245–270. IOS press, 2009.

[2] Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The community structure of SAT formulas. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 410–423. Springer, 2012.

[3] Gilles Audemard, Benoît Hoessen, Said Jabbour, Jean-Marie Lagniez, and Cédric Piette. Penelope, a parallel clause-freezer solver. In *SAT Challenge 2012: Solver and Benchmarks Descriptions*, pages 43–44, 2012.

[4] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Saïs. On freezing and reactivating learnt clauses. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 188–200. Springer, 2011.

[5] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI*, volume 9, pages 399–404, 2009.

[6] Gilles Audemard and Laurent Simon. Refining restarts strategies for SAT and UNSAT. In *Principles and Practice of Constraint Programming*, pages 118–126. Springer, 2012.

[7] Gilles Audemard and Laurent Simon. Glucose and syrup in the SAT race 2015. *SAT Race*, 2015.

[8] Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[9] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in computers*, 58:117–148, 2003.

[10] Armin Biere and Andreas Fröhlich. Evaluating CDCL restart schemes. In *Proceedings POS-15. Sixth Pragmatics of SAT workshop*, 2015.

[11] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[12] Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM (JACM)*, 35(4):759–768, 1988.

[13] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.

[14] Barthelemy Dagenais. Pymining, a few data mining algorithms in pure python. `https://github.com/bartdag/pymining`, 2015.

[15] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[16] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[17] Xavier Gillard. Adding SAT-based model checking to the pynusmv framework. Master's thesis, M.Sc. Thesis, Université Catholique de Louvain, 2016.

[18] Carla P Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. *Foundations of Artificial Intelligence*, 3:89–134, 2008.

[19] Jinbo Huang et al. The effect of restarts on the efficiency of clause learning. In *IJCAI*, volume 7, pages 2318–2323, 2007.

[20] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[21] Nathan Mull, Daniel J Fremont, and Sanjit A Seshia. On the hardness of SAT with community structure. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 141–159. Springer, 2016.

[22] Zack Newsham, Vijay Ganesh, Sebastian Fischmeister, Gilles Audemard, and Laurent Simon. Impact of community structure on SAT solver performance. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 252–268. Springer, 2014.

[23] Zack Newsham, William Lindsay, Vijay Ganesh, Jia Hui Liang, Sebastian Fischmeister, and Krzysztof Czarnecki. Satgraf: Visualizing the evolution of SAT formula structure in solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 62–70. Springer, 2015.

[24] Lawrence Ryan. *Efficient algorithms for clause-learning SAT solvers.* PhD thesis, Citeseer, 2004.

[25] Maxim Shishmarev, Christopher Mears, Guido Tack, and Maria Garcia de la Banda. Learning from learning solvers. In *International Conference on Principles and Practice of Constraint Programming*, pages 455–472. Springer, 2016.

[26] João P Marques Silva and Karem A Sakallah. GRASP – a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227. IEEE Computer Society, 1997.

[27] Grigori Samuilovich Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constrained Mathematics and Mathematical Logic*, 1968.

[28] Hantao Zhang and Mark Stickel. Implementing the davis–putnam method. *Journal of Automated Reasoning*, 24(1-2):277–296, 2000.

Formal Verification of CNL Health Recommendations

Fahrurrozi Rahman and Juliana Bowles

School of Computer Science, North Haugh, St Andrews KY16 9SX, UK
{fr27,jkfb}@st-andrews.ac.uk

1 Introduction

Clinical texts, such as therapy algorithms, are often described in natural language and may include hidden inconsistencies, gaps and potential deadlocks. We propose an approach to identify such problems with formal verification. From each sentence in a therapy algorithm written in controlled natural language (CNL), we automatically generate a parse tree and derive case frames. From the case frames we construct a state-based representation and use the model checker UPPAAL to verify the model. The choice of timed automata and UPPAAL is motivated by the fact that clinical texts sometimes make reference to timed and periodic events and we want the added flexibility of probabilistic extensions available in the wide range of tools that the UPPAAL family has. We will explore the use of different formal techniques including constraint solvers at later stages of the work.

Our overall aim is to use formal methods to detect gaps and case omissions in clinical pathways, and help to further clarify treatment steps. In the long term, we would like to develop solutions that make it possible for patients to gain a better understanding of the therapy underlying their disease and the options available to them. Our current focus is on the algorithm for blood glucose lowering therapy in adults with type 2 diabetes from the National Institute for Health and Care Excellence (NICE)[1]. This extended abstract is based on [6].

2 Background

Demner-Fushman et al. [2] reports the adaptation of NLP strategies to develop clinical decision support (CDS) systems that may help in decision making—e.g. monitoring of clinical events, processing radiology/pathology text-based reports, or processing a mixture of clinical note—for health care providers as well as the public. Aiming to improve the (cost) effectiveness of colorectal cancer (CRC) screening and surveillance, Imler et al. [4] conducted a study by annotating pathology reports which can give high accuracies to detect CRC, advanced adenoma, conventional adenoma, etc.

In software engineering, Carvalho [1] has shown how to combine NLP and formal methods to automatically generate test cases from the software requirements. The requirements written in CNL are parsed and followed by the construction of case frames which are then transformed into the *data flow reactive system* formalism and then translated into different target formal methods for verification to generate candidates for test cases. Motivated to detect problems at an early stage of development process, Diamantopoulos et al. [3] created a mechanism to automatically map requirements to formal representation using semantic role labelling. Using a model that has been trained from annotated software requirements, several ontology concepts are inferred and then can be used to trace the connection and relationship between them.

[1]https://goo.gl/YDDtQY

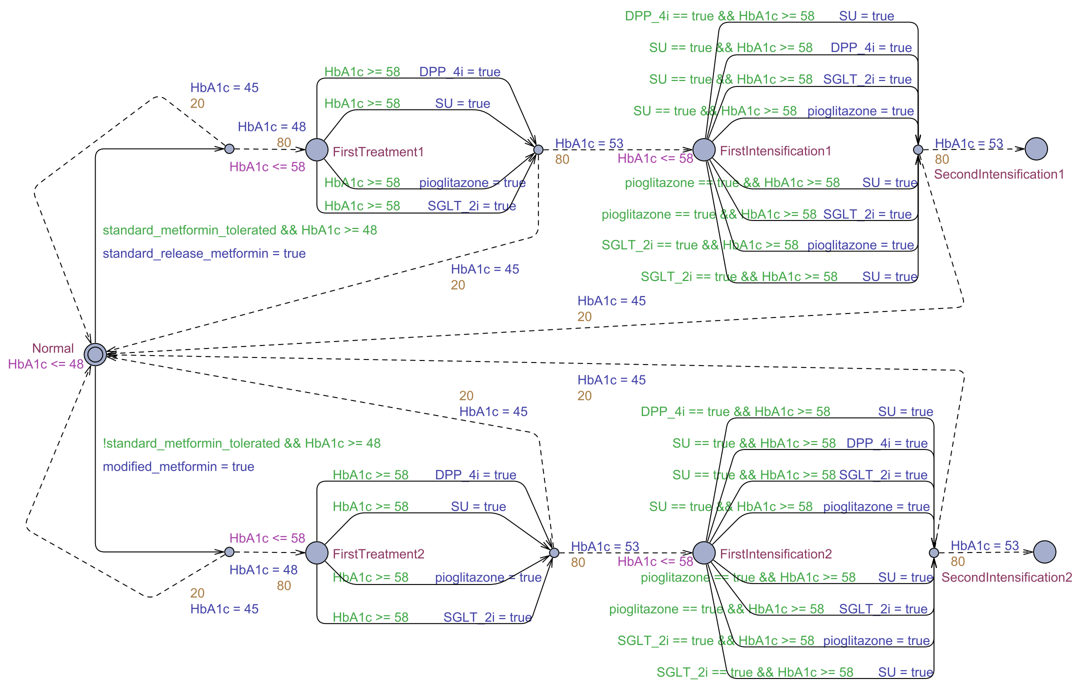

Normal
HbA1c <= 48
FirstTreatment1
FirstTreatment2
FirstIntensification1
FirstIntensification2
SecondIntensification1
SecondIntensification2
standard_metformin_tolerated && HbA1c >= 48
standard_release_metformin = true
!standard_metformin_tolerated && HbA1c >= 48
modified_metformin = true
HbA1c = 45
20
HbA1c = 48
80
HbA1c <= 58
HbA1c >= 58 DPP_4i = true
HbA1c >= 58 SU = true
HbA1c >= 58 pioglitazone = true
HbA1c >= 58 SGLT_2i = true
HbA1c = 53
DPP_4i == true && HbA1c >= 58 SU = true
SU == true && HbA1c >= 58 DPP_4i = true
SU == true && HbA1c >= 58 SGLT_2i = true
SU == true && HbA1c >= 58 pioglitazone = true
pioglitazone == true && HbA1c >= 58 SU = true
pioglitazone == true && HbA1c >= 58 SGLT_2i = true
SGLT_2i == true && HbA1c >= 58 pioglitazone = true
SGLT_2i == true && HbA1c >= 58 SU = true

Figure 1: Model with Branch Points

3 The Problem, Model Generation and Verification

In our work, we process specific clinical texts related to handling a particular disease. Instead of generating output in terms of steps for testing as done in [1] or annotating texts as done in [3], we look for and verify discrepancies that are inherently hidden inside the therapy algorithm.

To process the sentences in the therapy algorithm, we create a CNL (in terms of lexical and grammar rules) to standardise the structure of the sentences. After defining the rules as context free grammar (CFG), the original sentences taken from the therapy algorithm are manually rewritten so they conform to the rules. These sentences are then parsed with the help of modgrammar[2], a library in Python for building parsers using CFG definitions. From the generated parse tree, we construct the case frames following the eight thematic roles defined by Carvalho [1] with two additional roles: one for handling the nested condition in our sentences and another one to mark the sequence of the sentence in the therapy algorithm.

From the constructed case frames, we generate the diabetes automaton by automaticallly traversing the case frames for each sentence. The generated model must agree with UPPAAL's notation: a timed automaton with clocks, locations, location invariants, transitions, and guards. Fig. 1 shows the final (partial) model generated after adding some branch points that takes the system back to the initial state (to model when a patient's condition turns back to normal after a period of time under treatment which was not covered by the original therapy algorithm).

The generated model is verified against several predefined properties some of which can be seen in Table 1. The properties selected are used primarily to evaluate our approach. For

[2]https://bitbucket.org/modgrammar/modgrammar

example,

```
A[] !deadlock
```

is not satisfied because the model at present and as generated from the therapy algorithm contains no further steps after second intensificiation and hence deadlocks at that point. This suggests that further discussions with clinicians are required to understand available options from that point and what is realistic.

Table 1: Verification Result

UPPAAL queries	Verification Result	Remark
`A[] !deadlock`	not satisfied	There is a path that leads to a deadlock.
`E<> !diab.FirstIntensification1 && !diab.FirstIntensification2`	satisfied	There exists a path where first intensification is never reached.
`diab.FirstIntensification2 --> diab.SecondIntensification2`	not satisfied	There is a path in which a second intensification will never be reached from a first intensification.

4 Conclusion

We have presented an approach to analyse therapy algorithms published by NICE automatically. Therapy algorithms contain instructions which we treat as CNL statements and from which we ultimately build a state-based representation, in our case a timed automaton, that can be analysed by a model checker such as UPPAAL. A particular problem that we wanted to investigate was how to detect inconsistencies and gaps in the treatment options. In particular, it was noted that the algorithm did not consider a possible recovery in the different treatment stages that a patient with type 2 diabetes may go through nor what happens after a second intensification. In future collaborations with a general practitioner, we will analyse EHRs for patients with diabetes in Scotland, in order to detect cases or treatments that are not currently captured in the guidelines, and extend the algorithm accordingly. More broadly, we also want to see if one algorithm may have conflicts with another from a different disease for patients with multiple ongoing chronic conditions (cf. [5]).

References

[1] G. Carvalho. *NAT2TEST: Generating Test Cases from Natural Language Requirements based on CSP*. PhD thesis, Universidade Federal de Pernambuco, 2016.

[2] D. Demner-Fushman et al. What can natural language processing do for clinical decision support? *Journal of Biomedical Informatics*, 42(5):760–772, 2009.

[3] T. Diamantopoulos et al. Software requirements as an application domain for natural language processing. *Language Resources and Evaluation*, 51(2):495–524, 2017.

[4] T. D. Imler et al. Multi-center colonoscopy quality measurement utilizing natural language processing. *The American Journal of Gastroenterology*, 110(4):543–552, 2015.

[5] A. Kovalov and J.K.F. Bowles. Avoiding medication conflicts for patients with multimorbidities. In *International Conference on Integrated Formal Methods*, pages 376–390, 2016.

[6] F. Rahman and J.K.F. Bowles. Formal Verification of CNL Health Recommendations. In *International Conference on Integrated Formal Methods*, 2017. To appear in *Lecture Notes in Computer Science 10510*

Modular Heap Shape Analysis for Java Programs[*]

Florian Frohn and Jürgen Giesl

LuFG Informatik 2, RWTH Aachen University, Germany

Abstract

We report on our ongoing work towards an automatic heap shape analysis for Java programs in order to approximate the possible forms of sharing and cycles on the heap. Our analysis is completely modular, i.e., Java methods are analyzed independently and each method is analyzed only once. Moreover, in contrast to similar analyses, our technique does not only analyze cyclicity and reachability, but also sharing, i.e., it checks whether two objects may have a common successor. Finally, our analysis is path-sensitive, i.e., if two objects o and o' share, then our analysis approximates which paths of object fields lead from o and o' to their common successor. Use-cases for such an analysis include (but are not limited to) the modularization of termination analysis, complexity analysis, and data-flow analysis.

1 Introduction

Our tool AProVE [3] has been the most powerful fully automatic termination analyzer for Java programs for many years, as witnessed by the annual termination competition.[1] Its outstanding feature is its precise handling of the heap, which is analyzed via symbolic execution using a rather complex abstract domain [1, 4]. To solve the arising search problems, AProVE uses standard SMT solvers. Recently, AProVE's termination analysis has been adapted for complexity analysis of Java programs [2]. This adaption was driven by our project *CAGE*[2] with Draper Inc. and the University of Innsbruck where AProVE analyzes programs with tens of thousands lines of code. Unsurprisingly, this project revealed bottlenecks w.r.t. the scalability of AProVE's current heap shape analysis (HSA). While we can bypass these restrictions by providing *method summaries* as additional input [2], a more scalable automatic HSA is desirable.

In Sect. 2, we sketch such an analysis after discussing AProVE's current heap shape analysis (which we call $\mathcal{HS}_{\mathsf{AProVE}}$ from now on) and its limitations w.r.t. scalability. Like $\mathcal{HS}_{\mathsf{AProVE}}$, our new analysis $\mathcal{HS}_{reg}$ can be used to analyze sharing as well as cyclicity, but in this paper we focus on sharing for reasons of space. Sect. 3 briefly discusses related work and concludes.

2 An Alternative to **AProVE**'s Heap Shape Analysis

Like many other HSAs, AProVE represents sets of program states using a symbolic heap and a mapping from local variables to symbolic references (i.e., pointers to memory locations). To express information about the heap, it uses predicates over symbolic references [4]:[3] The *points-to* predicate $o \xrightarrow{f} o'$ expresses that the field f of the object referenced by o stores the reference o', the *may-alias* predicate $o =^? o'$ means that o and o' may be equal, and the *may-share* predicate $o \searrow\!\!\swarrow o'$ states that o and o' may have a common successor, i.e., some o'' may be reachable from o as well as o' by dereferencing (possibly empty) sequences of fields. The semantics of the empty set of predicates is that all objects are in disjoint parts of the heap, i.e., sharing and aliasing

*Supported by the DFG grant GI 274/6-1 and the Air Force Research Laboratory (AFRL).

[1]http://termination-portal.org/wiki/Termination_Competition

[2]http://www.draper.com/news/draper-s-cage-could-spot-code-vulnerable-denial-service-attacks

[3]Here, we omit some predicates for reasons of space.

has to be allowed explicitly using predicates. Note that the points-to predicate means that some connection *must* exist, whereas the may-alias and the may-share predicate express that references *may* share or alias, i.e., $\mathcal{HS}_{\mathsf{AProVE}}$ simultaneously performs may- and must-analyses. Consequently, the semantics of $\mathcal{HS}_{\mathsf{AProVE}}$ is rather complicated.

Example 1. *Consider a class for lists with the fields* `value` *and* `next`*. To represent a list* o_1 *with at least two elements whose last value is* o_3*, the predicates* $\{o_1 \xrightarrow{\texttt{next}} o_2, o_2 \searrow\!\!\swarrow o_3\}$ *can be used. Note that the connection from* o_2 *to* o_3 *of the form* `next`*`.value` *cannot be expressed precisely since the may-share predicate is not path-sensitive. In particular,* $o_2 \searrow\!\!\swarrow o_3$ *also allows paths from* o_3 *to* o_2*, i.e., it over-approximates the connection between* o_2 *and* o_3 *rather coarsely.*

The main drawback of $\mathcal{HS}_{\mathsf{AProVE}}$ for scalability is its dependence on the context in which methods are called (context-sensitivity). So the same method often has to be analyzed several times to take differences in the calling states into account. This need for context-sensitivity is closely related to $\mathcal{HS}_{\mathsf{AProVE}}$'s restricted expressivity. As shown in Ex. 1, $\mathcal{HS}_{\mathsf{AProVE}}$ cannot describe the effect of many common heap manipulating methods like appending to the end of a list precisely. Context-sensitivity often allows us to focus on specific cases instead. For example, the result of appending to the end of a list *of known length* can be expressed using the points-to predicate.

Hence, the goal of our new analysis $\mathcal{HS}_{reg}$ is to improve expressivity such that context-sensitivity can be sacrificed without major regressions w.r.t. precision. To this end, $\mathcal{HS}_{reg}$ uses a single path-sensitive predicate $o \xrightarrow{\pi}\!\!\xleftarrow{\tau} o'$. Here, π and τ are regular languages over the set $\mathcal{F}$ of all fields in the program (i.e., $\pi, \tau \subseteq \mathcal{F}^*$). The semantics of $o \xrightarrow{\pi}\!\!\xleftarrow{\tau} o'$ is that o and o' may only share if o reaches their common successor via a path from π and o' reaches this successor via a path from τ. (More precisely, if $o.v = o'.w$ and $o.v' \neq o'.w'$ holds for every prefix v' of v and every prefix w' of w where $v' \neq v$ or $w' \neq w$, then we have $v \in \pi$ and $w \in \tau$). Hence, $o \xrightarrow{\mathcal{F}^*}\!\!\xleftarrow{\mathcal{F}^*} o'$ and $o \xrightarrow{\varepsilon}\!\!\xleftarrow{\varepsilon} o'$ correspond to $\mathcal{HS}_{\mathsf{AProVE}}$'s may-share and may-alias predicates.

Since $\mathcal{HS}_{reg}$ does not provide a points-to predicate, it is a pure may-analysis and hence avoids the complexity that arises from combining may- and must-analyses.

Example 2. *With* $\mathcal{HS}_{reg}$*, the situation in Ex. 1 can be expressed by the predicates* $\{o_1 \xrightarrow{\texttt{next}}\!\!\xleftarrow{\varepsilon} o_2, o_2 \xrightarrow{\texttt{next}^*\texttt{.value}}\!\!\xleftarrow{\varepsilon} o_3\}$*. The path from* o_2 *to* o_3 *is described more precisely than in Ex. 1, since these predicates do not allow paths from* o_3 *to* o_2 *anymore. However, they only express that* o_1*'s field* `next` may *store* o_2*, whereas the predicates from Ex. 1 express that it* must *store* o_2*.*

Ex. 2 shows that $\mathcal{HS}_{\mathsf{AProVE}}$ and $\mathcal{HS}_{reg}$ are orthogonal in general, but $\mathcal{HS}_{reg}$ can describe possible (as opposed to definite) sharing more precisely than $\mathcal{HS}_{\mathsf{AProVE}}$, i.e., $\mathcal{HS}_{reg}$ fits our needs w.r.t. expressivity. This is also true for more complex data structures (e.g., binary trees with the fields `value`, `left`, and `right`, where $o \xrightarrow{(\texttt{left}|\texttt{right})^*\texttt{.value}}\!\!\xleftarrow{\varepsilon} o'$ expresses that o' may be an element of the tree o). However, applying $\mathcal{HS}_{reg}$ in an interprocedural, context-insensitive setting with reasonable precision is non-trivial. The reason is that interprocedural program analyses usually summarize methods using pre- and postconditions. Hence, to ensure that every method is analyzed at most once, each method has to be analyzed with the most general precondition. For $\mathcal{HS}_{reg}$, this means that we have to add the predicate $o \xrightarrow{\mathcal{F}^*}\!\!\xleftarrow{\mathcal{F}^*} o'$ for each pair o, o' of arguments of the analyzed method to the initial state. Clearly, this would diminish the precision of $\mathcal{HS}_{reg}$.

Our solution is to analyze a slightly different property than $\mathcal{HS}_{\mathsf{AProVE}}$ and many other similar analyses. $\mathcal{HS}_{\mathsf{AProVE}}$ analyzes the property "which references may share" for every program position of the analyzed method `m`. Clearly, this property is highly context-sensitive, i.e., it can differ significantly depending on the program state in which `m` is invoked. Instead, $\mathcal{HS}_{reg}$ analyzes the property "which references may share *due to side-effects of* `m`", i.e., it just considers sharing that has been introduced by the currently analyzed method itself. While this property is clearly context-insensitive, it coincides with the property analyzed by $\mathcal{HS}_{\mathsf{AProVE}}$ if there is no sharing in the initial state, as it is the case for the `main` method of Java programs.

Example 3. *Consider a method* `add` *that appends a value o' to the end of a list o. Independently from the states in which* `add` *is called, the property "which references may share when* `add` *returns* due to side-effects of `add`*" can be approximated by the predicate* $o \xrightarrow{\texttt{next}^*.\texttt{value}} \xleftarrow{\varepsilon} o'$.

As a consequence, whenever $\mathcal{HS}_{reg}$ encounters an invocation of a previously analyzed method `m`, it has to incorporate the connections that might be introduced by `m` into the calling state.

Example 3 (continued)**.** *Assume that* `add` *is called in a state whose heap is described by* $o \xrightarrow{\texttt{next}^*} \xleftarrow{\varepsilon} o''$*. To construct the state* after *the invocation of* `add`*,* $\mathcal{HS}_{reg}$ *has to incorporate the new connection* $o \xrightarrow{\texttt{next}^*.\texttt{value}} \xleftarrow{\varepsilon} o'$ *into the predicate* $o \xrightarrow{\texttt{next}^*} \xleftarrow{\varepsilon} o''$ *of the calling state. Since o'' may be reachable from o via* `next`*-pointers, o' may be inserted behind o'', i.e., o' and o'' may share. Hence, the resulting state is* $\{o \xrightarrow{\texttt{next}^*} \xleftarrow{\varepsilon} o'', o \xrightarrow{\texttt{next}^*.\texttt{value}} \xleftarrow{\varepsilon} o', o'' \xrightarrow{\texttt{next}^*.\texttt{value}} \xleftarrow{\varepsilon} o'\}$.

To infer predicates that approximate the side-effects introduced by methods (with loops or recursion), $\mathcal{HS}_{reg}$ requires techniques for generalization and symbolic reasoning with regular languages. We developed a library which accomplishes this task and yields promising results.

3 Conclusion

In this paper, we discussed some shortcomings of AProVE's current heap shape analysis and sketched an alternative analysis to overcome them. The key idea is to use a path-sensitive abstract domain to improve expressivity, which in turn allows us to sacrifice context-sensitivity without major regressions w.r.t. precision. As a result, we obtain a high degree of modularity. First results with a prototypical implementation are promising.

Apart from AProVE's previous approach to HSA, the most closely related techniques are [5,6]. However, these approaches are just field-sensitive, i.e., in contrast to our technique they do not take the order of fields on paths into account. Moreover, they just analyze reachability and cyclicity, and they rely on a field-insensitive sharing analysis, whereas our sharing analysis is also field- (and even path-) sensitive. Finally, our approach is orthogonal to separation logic, which is the base for many pure must-analyses, whereas our analysis is a pure may-analysis.

References

[1] M. Brockschmidt, R. Musiol, C. Otto, and J. Giesl. Automated termination proofs for Java programs with cyclic data. In *Proc. CAV '12*, LNCS 7358, pages 105–122, 2012.

[2] F. Frohn and J. Giesl. Complexity Analysis for Java with AProVE. In *Proc. iFM '17*, LNCS, 2017.

[3] J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Analyzing program termination and complexity automatically with AProVE. *JAR*, 58(1):3–31, 2017.

[4] C. Otto, M. Brockschmidt, C. von Essen, and J. Giesl. Automated termination analysis of Java Bytecode by term rewriting. In *Proc. RTA '10*, LIPIcs 6, pages 259–276, 2010.

[5] E. Scapin and F. Spoto. Field-sensitive unreachability and non-cyclicity analysis. *Sci. Comput. Program.*, 95:359–375, 2014.

[6] D. Zanardini and S. Genaim. Inference of field-sensitive reachability and cyclicity. *ACM Transactions on Computational Logic*, 15(4):33:1–33:41, 2014.

Abstraction-based Model Checking of POMDPs in Motion Planning*

Leonore Winterer[1], Sebastian Junges[2], Ralf Wimmer[1], Nils Jansen[4], Ufuk Topcu[3], Joost-Pieter Katoen[2], and Bernd Becker[1]

[1] BrainLinks-BrainTools Cluster of Excellence
Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany
{winterel | wimmer | becker}@informatik.uni-freiburg.de
[2] RWTH Aachen University, Aachen, Germany
{sebastian.junges | katoen}@informatik.rwth-aachen.de
[3] The University of Texas at Austin, Austin, Texas, USA
utopcu@utexas.edu
[4] Radboud University, Nijmegen, The Netherlands
n.jansen@science.ru.nl

Abstract

Partially observable Markov decision processes (POMDPs) are a natural model for many applications where one has to deal with incomplete knowledge and random phenomena, including, but not limited to, robotics and motion planning. However, many interesting properties of POMDPs are undecidable or otherwise very expensive to decide in terms of both runtime and memory usage. In our work, we develop abstraction-based methods that can deliver safe bounds and good approximations for certain classes of properties.

1 Challenge

In offline motion planning, we aim to find a *strategy* for an agent that ensures certain desired behavior, even in the presence of dynamical obstacles and uncertainties [2]. If random phenomena like uncertainty in the outcome of an action or in the movement of dynamic obstacles need to be taken into account, the natural model for such scenarios are *Markov decision processes* (MDPs). MDPs are are non-deterministic models which allow the agent to perform actions under full knowledge of the current state of the agent and the surrounding environment. In many applications, though, full knowledge cannot be assumed, and we have to deal with *partial observability* [3]. For such scenarios, MDPs are generalized to *partially observable Markov decision processes* (POMDPs). In a POMDP, the agent does not know the exact state of the environment, but only an observation that can be shared between multiple states. Additional information about the likelihood of being in a certain state can be gained by tracking the observations over time. This likelihood is called the belief state. Using an update function mapping a belief state and an action as well as the newly obtained observation to a new belief state, one can construct a (typically infinite) MDP, commonly known as the *belief MDP*.

While model checking and strategy synthesis for MDPs are, in general, well-manageable problems, POMDPs are much harder to handle and, due to the potentially infinite belief space, many problems are actually undecidable [1]. Our aim is to apply different *abstraction and abstraction refinement* techniques to POMDPs in order to get good and safe approximative results for different types of properties.

*This work was (partly) supported by BrainLinks-BrainTools, Cluster of Excellence funded by the German Research Foundation (DFG, grant number EXC 1086) and by the awards ONR # N000141612051, NASA # NNX17AD04G and DARPA # W911NF-16-1-0001.

2 Approach

As a case study, we work with a scenario featuring a controllable agent. Within a certain area, the agent needs to traverse a room while avoiding both static obstacles and randomly moving opponents. Only the positions of the opponents cannot always be observed. The area is modeled as a grid, the static obstacles as grid cells that may not be entered. Our detailed assumption for this scenario is that the agent always knows its own position, but the positions of an opponent is only known if its distance from the agent is below a given threshold and if the opponent is not hidden behind a static obstacle. We assume that the opponents move probabilistically. This directly leads to a POMDP model for our case study. For simplification purposes, we are only dealing with one opponent right now, although the same approach would work with an arbitrary number of opponents as well.

The goal is to find a strategy which maximizes the probability to navigate through the grid from an initial to a target location without collision. For a grid size of $n \times n$ cells and one opponent, the number of states in the POMDP is in $O(n^4)$, i. e., the state space grows rapidly with increasing grid size. In order to handle non-trivial grids, we propose an approach using *game-based abstraction* [4].

Intuitively, we lump together all states that induce the same observation; so we can still distinguish between all states in which the opponent's position is known, but states in which the position is not known are merged into one *far away* state [6]. In order to get a safe approximation of the possible behavior of the opponent, for all of these lumped states we add a non-deterministic choice over the potential positions of the opponent. We formalize this as a *2-player probabilistic game* [4], in which one player controls the actions of the agent, and the other player controls the non-determinism added by the abstraction. This allows both players to optimize according to different goals. The abstraction player can create a worst-case scenario to *over-approximate* the realistic behavior, thus ensuring that the obtained bounds are safe and the resulting strategy cannot perform worse when mapped back to the original scenario.

A comparison with the state-of-the-art POMDP model checker PRISM-pomdp [5] indicates that we can handle grids that are considerably larger than what PRISM-pomdp can handle, while still getting schedulers that induce values which are close to the optimum. Table 1 shows a few of our results for verifying a reach-avoid property on a grid without obstacles. As one can see, the abstraction approach is faster by orders of magnitude than solving the POMDP directly, and the game model also is much smaller for large grids while still getting very good approximations for the actual probabilities. The strategies induce even better values when they are mapped back to the original POMDP. Note that this lifting of strategies can currently only be performed for small benchmarks due to technical limitations of the PRISM tool we use.

While being sound, our approach is still targeting an undecidable problem and as such not complete in the sense that in general no strategy with maximum probability for success can be deduced. In particular for cases with few paths to the goal location, the gap between the obtained bounds and the actual maximum can become large. For those cases, we define a scheme to refine the abstraction, which leads to larger games and accordingly longer computation times, but also to better results. In Table 2 we have demonstrated a first, very coarse step of this refinement. We use a benchmark representing a long, narrow tunnel, in which the agent has to pass the opponent once, but, due to the nature of the abstraction, can actually run into him several times. With longer tunnels, the probability to safely arrive in a goal state diminishes. Adding a refinement which remembers the last known position of the opponent and thus restricting the non-deterministic movement keeps the probability steady for arbitrary tunnel length.

Table 1: Comparing the POMDP solution using PRISM-pomdp with the solution of the PG abstraction using PRISM-games on different sized grids without obstacles for a reach-avoid property. The MDP result is the hard upper limit for the probability. All times are given in seconds.

	POMDP solution				PG solution				Lifting	MDP
Grid size	States	Result	Model Time	Sol. Time	States	Result	Model Time	Sol. Time	Result	
3×3	299	0.8323	0.063	0.26	400	0.8323	0.142	0.036	0.8323	0.8323
4×4	983	0.9556	0.099	1.81	1348	0.9556	0.353	0.080	0.9556	0.9556
5×5	2835	0.9882	0.144	175.94	6124	0.9740	0.188	0.649	0.9825	0.9882
5×6	4390	0.9945	0.228	4215.056	8058	0.9785	0.242	0.518	0.9893	0.9945
6×6	6705	??	0.377	– MO –	10592	0.9830	0.322	1.872	0.9933	0.9970
8×8	24893	??	1.735	– MO –	23128	0.9897	0.527	6.349	0.9992	0.9998
10×10	66297	??	9.086	– MO –	40464	0.9914	0.904	6.882	0.9999	0.9999
20×20	– Time out during model construction –				199144	0.9921	8.580	122.835	0.9999	0.9999
30×30	– Time out during model construction –				477824	0.9921	41.766	303.250	??	0.9999
40×40	– Time out during model construction –				876504	0.9921	125.737	1480.907	??	0.9999
50×50	– Time out during model construction –				1395184	0.9921	280.079	3129.577	??	– MO –

Table 2: Results of the game-based abstraction for benchmarks describing a long, narrow corridor, with and without rudimentary refinement. All times are given in seconds.

	Grid	States	Choices	Trans.	Result	Time
no ref.	4×40	50880	93734	170974	0.9228	19
	4×60	77560	143254	261534	0.8923	67
	4×80	104240	192774	352094	0.8628	115
	4×100	130920	242294	442654	0.8343	164
with ref.	4×40	55300	120848	198088	0.9799	63
	4×60	83820	182368	300648	0.9799	220
	4×80	112340	243888	403208	0.9799	265
	4×100	140860	305408	505768	0.9799	746

3 Conclusion and Further Research

Game-based abstraction has turned out to be a viable way to obtain safe – and in many cases rather tight – bounds on maximal reach-avoid probabilities in our motion planning scenario. It scales much better than the standard analysis algorithms for POMDPs, which are typically based on discretizing the corresponding belief MDP [5].

Currently, our approach is limited to a restricted class of POMDPs based on an agent and its opponents moving inside a connected graph, but we have reason to believe that game-based abstraction is also beneficial for arbitrary POMDPs. We are also going to investigate other classes of properties involving costs and the long-term behavior of agents.

References

[1] Krishnendu Chatterjee, Martin Chmelík, and Mathieu Tracol. “What is decidable about partially observable Markov decision processes with ω-regular objectives”. In: *J. Comput. Syst. Sci.* 82.5 (2016), pp. 878–911. DOI: 10.1016/j.jcss.2016.02.009.

[2] Ronald A. Howard. *Dynamic Programming and Markov Processes.* 1st edition. The MIT Press, 1960.

[3] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. “Planning and acting in partially observable stochastic domains”. In: *Artif. Intell.* 101.1 (1998), pp. 99–134. DOI: 10.1016/S0004-3702(98)00023-X.

[4] Mark Kattenbelt and Michael Huth. “Verification and Refutation of Probabilistic Specifications via Games”. In: *FSTTCS.* Vol. 4. LIPIcs. Schloss Dagstuhl, 2009, pp. 251–262. DOI: http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2009.2323.

[5] G. Norman, D. Parker, and Xueyi Zou. “Verification and Control of Partially Observable Probabilistic Systems”. In: *Real-Time Systems* 53.3 (2017), pp. 354–402. DOI: 10.1007/s11241-017-9269-4.

[6] Leonore Winterer et al. “Motion Planning under Partial Observability using Game-Based Abstraction”. In: *IEEE Conf. on Decision and Control (CDC).* Melbourne, Australia: IEEE, Dec. 2017.

Co-simulation and formal verification of logic-based specifications of components in CPSs

Maurizio Palmieri[1][2], Cinzia Bernardeschi[2], and Andrea Domenici[2]

[1] Dipartimento di Ingegneria dell'Informazione, University of Florence, Florence, Italy
[2] Dipartimento di Ingegneria dell'Informazione, University of Pisa, Pisa, Italy
{maurizio.palmieri,cinzia.bernardeschi,andrea.domenici}@ing.unipi.it

Abstract

Our work aims at the development of a methodology for formal modeling, simulation, and verification of logic-based specifications of components in cyber-physical systems (CPSs). The methodology is supported by automated tools: the logic specification of the CPS controllers is given in the logic of the Prototype Verification System (PVS) tool; logic theories are co-simulated with sub-systems modelled with continuous formalisms in the INTO-CPS co-simulation framework. Co-simulation, besides being a prototyping tool supporting the exploration of user interaction, provides checks at early stages of development. The interactive theorem prover of the PVS framework can be applied for generating formal proofs of the correctness of the proposed controllers. In the work carried out so far, the methodology is applied to simple autonomous vehicles.

1 Introduction

Cyber-Physical Systems (CPS) are complex physical systems operated by digital controllers. From the computational point of view, the existence of digital and physical components requires the use of different kinds of mathematical formalisms, i.e., discrete, logic-based models for controllers and continuous models based on differential equations for plants. In addition, the physical parts of a CPS may have to be modeled with different languages and tools, since a variety of modeling tools has been produced to address different physical aspects of the systems. Simulation in CPS often takes the form of co-simulation, i.e., integrated simulation of different subsystems, each modeled with a specific formalism and simulated by a specific simulation engine. The *Functional Mockup Interface* (FMI) [1] is a standard for co-simulation: sub-models implemented as *functional mockup units* (FMUs), are orchestrated by a master that communicates with them through proxy modules (*FMI wrappers*) whose interfaces are FMI-compliant. Recently, the INTO-CPS project [3] created an integrated co-simulation framework based on FMI. A complementary approch to co-simulation is formal verification. For example, KeYmaera [2] is a theorem prover, recently developed and applied successfully for the verification of CPSs. Its language includes conditions, non-determinism, loops, composition, and continuous dynamics, i.e., behaviours defined by differential equations.

The work we present provides the possibility of co-simulate logic theories given in the higher-order logic of the Prototype Verification System PVS [5] with sub-systems modelled with continuous formalisms in the INTO-CPS co-simulation framework. PVS has been used to verify systems from many fields of application, including, e.g., microprocessor design and air traffic control. Moreover, the interactive theorem prover of the PVS framework can be applied for generating formal proofs of the correctness of the proposed controllors. We show the application of our methodology to a simple example: a unicycle that has the task of reaching an assigned straight path and then follow it, and it can be controlled by varying its turning speed.

2 FMU generation for PVS

The PVS specification provides basic types, such as Booleans, naturals, integers, reals, and others, and type constructors to define more complex types. The mathematical properties of each type are defined axiomatically in a set of fundamental theories, called the *prelude*. Among the complex types, the ones used in this work are *record* types and *predicate subtypes*.

A way to use a PVS theory is to use it as an *executable* model for simulation. This is made possible by the PVSio extension. PVSio [4] is a ground evaluator that computes the value of ground (variablefree) expressions. The evaluator can also compute functions with side effects, such as producing outputs. It should be noted that functions with side effects are logically equivalent to normal (i.e., purely logical) functions of the PVS language, so that they do not interfere with theorem proving. The PVSio evaluator acts as an interpreter for PVS logic language.

The FMU has been extended with a Graphical User Interface (GUI) that, in addition to visualizing the simulation, can forward input from the user to the system controller at run-time.

Introducing user inputs from a GUI requires the controller to work in two alternative modes, automatic or manual. Accordingly, a new variable is added to the state of the controller to keep track of the current mode. The switch between different control modes can be performed in different ways, depending on the specific case. For example, a specific input from the user could trigger the automatic control mode.

3 Case study: a unicycle vehicle

A unicycle is a possible representation of a terrestrial vehicle with a pair of wheels connected through an axle; it is represented by three state variables: x, y, ψ. The pair (x, y) indicates the midpoint position of the axle and ψ indicates the orientation angle from the x-axis and the orthogonal to the axle itself, as shown in Figure 1(left). The linear speed v is commonly given by a preassigned function of time, for simplicity we can assume it to be constant, so the only free variable we can use to achieve our goal is the angular speed ω. In order to deal with the problem of driving the vehicle to follow a pre-defined straight line we can use the following equation:

$$\omega = -\,d\,v\,sinc(\psi - \theta) - \mathrm{k}\,(\psi - \theta) \tag{1}$$

where d is the distance between the vehicle and the desired line and θ is the angular coefficient of the desired line, k is a parameter of the control law and $sinc(\alpha) = sin(\alpha)/\alpha$, if $\alpha \neq 0$; 1, otherwise . The unicycle kinematics is modeled in Simulink, and exported as an FMU. The controller is defined by the unicycle controller theory:

```
unicycle_controller : THEORY
BEGIN
IMPORTING stdmath
 State : TYPE = [#
      y: real, x: real, psi: real, % inputs
      y_0: real, psi_c: real, % target line
      k: real, v: real, % parameters
      w: real #] % output
END
```

The state record type contains the unicycle coordinates and yaw angle coming from the Simulink side, the parameters of the target line, the control law parameters, and the turning

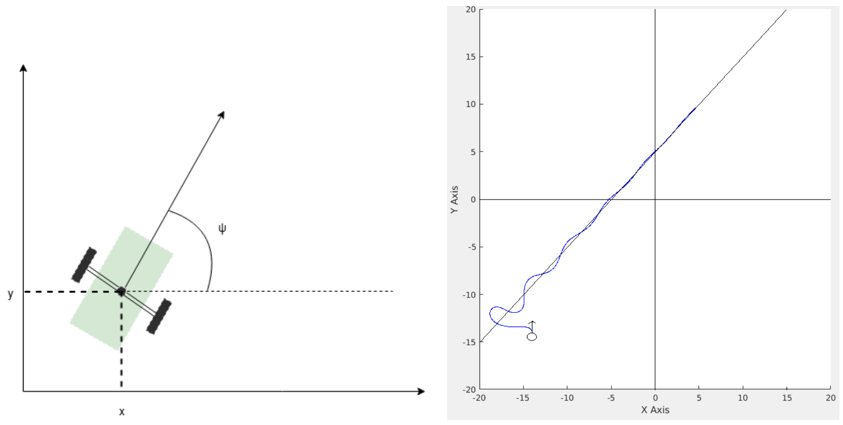


Figure 1: Representation of a unicycle (left); an example of co-simulation (right).

speed to be sent to the Simulink side. The function `step(s:State):State` uses the input and parameter fields of the state record to compute the new value of `w` and replace the previous one in a new copy of the record.

The co-simulation model has been exercised by varying the parameters of the target line and of the control law, and the initial position and orientation of the unicycle. For example, Figure 1(right) shows the path of the unicycle starting at $(-14, -14)$, headed parallel to the ordinate axis, and approaching a straight line shown in the figure. The control law parameters are $k = 0.05$ and $v = 0.2$; the resulting trajectory has an oscillating transient.

Using the interactive theorem prover of PVS, we proved the correctness of the control law in reaching the assigned straight path and then follow it. Eq. 1 can easily be implemented in a PVS theory. We applied a common practice in control theory: the system is linearized at the desired state, the systems Jacobian is formulated, and the asymptotic stability of the state is verified by showing that the eigenvalues are real and negative.

References

[1] Torsten Blochwitz, Martin Otter, Johan Åkesson, Martin Arnold, Christoph Clauß, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauß, Dietmar Neumerkel, Hans Olsson, and Antoine Viel. Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In *Proceedings of the 9th International Modelica Conference*, pages 173–184. The Modelica Association, 2012.

[2] F. Franchetti, T. M. Low, S. Mitsch, J. P. Mendoza, L. Gui, A. Phaosawasdi, D. Padua, S. Kar, J. M. F. Moura, M. Franusich, J. Johnson, A. Platzer, and M. M. Veloso. High-assurance spiral: End-to-end guarantees for robot and car control. *IEEE Control Systems*, 37(2):82–103, April 2017.

[3] Integrated Tool Chain for Model-based Design of Cyber-Physical Systems$^{®}$ project, Horizon 2020, grant number 644047.

[4] C. Muñoz. Rapid prototyping in PVS. Technical Report NIA 2003-03, NASA/CR-2003-212418, National Institute of Aerospace, Hampton, VA, USA, 2003.

[5] S. Owre, S. Rajan, J. Rushby, N. Shankar, and M. Srivas. PVS: combining specification, proof checking, and model checking. In R. Alur and T.A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in LNCS, pages 411–414. Springer-Verlag, 1996.

Towards a Contract-Based Unified Design Process

Cesar A. R. Santos[1,2], Mike Nicolai[1], and Tom Schrijvers[2]

[1] Siemens Industry Software N.V., Leuven, Belgium
[2] KU Leuven, Belgium

Abstract

Mechatronic systems engineering is a complex multidisciplinary activity that presents many challenges. Design decisions from one engineering domain affect the other domains, making it difficult to develop systems in a modular way. The different tools and methodologies used by each domain complicate the problem, as multiple incompatible models must be kept consistent throughout the design process. Effectively coordinating and integrating multiple engineering domains, at all levels of abstraction in the design cycle, is still an open problem.

Contract-Based Design offers a solution to the integration problem. Contracts allow component-wise specification of requirements, at different levels of abstraction, and the formal verification of their composition and refinement. However, contracts can only validate fully connected architectures, limiting them to later phases of the design.

Computational Design Synthesis offers a promising solution to the consistency problem, by generating multiple architectural and simulation models from a single abstract system model, such that they are consistent by construction. Existing synthesis approaches are limited by the integration capabilities of the simulation tools for which they generate models. This relegates integration to near the end of the design phase.

We believe that a combination of Contract-Based Design and Computational Design Synthesis is a better approach. This text argues that this combination enables a large portion of the design cycle to be kept consistent by construction, and also allows for early separation of concerns and safe integration of independently developed parts.

In the presentation examples of the challenges involved in the design of different mechatronic use cases will be presented. Examples will be given based on the standard approaches, showing their limitations, and how this unified methodology could improve upon the state of the art.

1 Introduction

Mechatronic systems engineering is a multidisciplinary field, combining mechanical engineering, electrical engineering, computer science and more. Engineers from different fields work on different parts of the system at different levels of abstraction. Because mechatronic systems are integrated systems, the entire system must be viewed as a whole. [1]

A change in the mechanical parts of a system may have direct consequences on the design of the electrical wiring, for example, or the thermal dissipation, which might require changes to sensors and the software monitoring them. All domains must be taken into account simultaneously, as much as possible, to achieve optimal designs. Each engineering domain has its own tools and methodologies, making coordination and integration difficult. [2]

The design process for a mechatronic system can be subdivided following the RFLP [3] design process. In traditional document-based systems engineering all the artifacts produced during this process had to be manually kept consistent. If a simulation gave unexpected results, the cause had to (potentially) be traced back to a faulty requirement, and then all of the design documents updated to reflect the necessary change.

Model-based systems engineering (MBSE) [4] is a methodology designed to deal with the consistency and integration issues of the typical document-based approach to system engineering, through the centering the design cycle around a single integrated and coherent system model. Ideally this model is a formal and computationally tractable specification of the system, from which all other artifacts can be automatically generated.

In the next section we look the state of the art in MBSE. Specifically, we look at how the combination of Computational Design Synthesis [5] and Contract-Based Design [6], better unifies the design process.

2 State of the Art

The two major challenges facing MBSE are consistency and integration. One approach to maintaining consistency is Computational Design Synthesis. One approach for safer integration is Contract-Based Design.

In **Computational Design Synthesis** the system is modeled at the logical architecture or even functional level [7]. The system is subdivided into abstract components with some interface (a set of input and output ports), and the synthesis tool can generate all possible logical architectures from that specification. If the components are annotated with realizations to different simulation and analyses tools, then those detailed models can also be automatically generated.

Any change to the abstract model is automatically reflected in all the generated models, and as such they are kept consistent by construction. Only documents that do not follow the component model must be handled separately, like requirements.

Contract-Based Design has been proposed in the literature as a way to tackle the problem of integrating independent designs. In the Contract-Based Design paradigm each component is associated with a formal contract, i.e., a clear description of its expected behavior. The contracts specify the assumptions a component makes about its environment, and what that component guarantees under those conditions.

This modular structuring of requirements allows for compositional reasoning, stepwise refinement, and a principled reuse of components that are designed independently.

Particularly, component connections and refinements of abstract models can be computationally checked without requiring detailed simulation.

3 A Unified Process

Because both contracts and design synthesis work on the level of systems, components and their interfaces, the two can be combined. Computational Design Synthesis uses solvers (like SAT [8] solvers) to generate all the possible architectures from a system specification. Contract Based Design uses solvers to verify that contract composition and refinement is valid.

These two solving processes can be merged, such that the synthesis tool only generates logical architectures that result in valid contract compositions or refinements. This means that any implementation of an abstract component that follows its contract will fit any of the generated logical architectures.

With the combination of Computational Design Synthesis and Contract-Based Design, the chances of a project needing to be restarted are much lower. Each separate team can iterate and computationally validate their part independently from the system model. The requirements, being formally verifiable, are far less likely to be inconsistent or incomplete. Finally, all possible architectural solutions can be explored simultaneously. While it is not possible to guarantee that all generated architectures will work, the number of iterations on the complete system are greatly reduced, thus reducing development costs.

4 Conclusions

The MBSE approach aims to solve the integration and consistency problems being faced by mechatronic systems engineering. The MBSE ideal, with a single system model from which everything is derived, does not exist in practice.

Computational Design Synthesis offers excellent support for consistency, while Contract-Based Design offers excellent support for integration. The combination of the two is superior to the sum of their parts, as they also complement each other in other ways, enabling better architectural synthesis, and contracts to be distributed earlier in the process.

The presentation will show examples of the challenges involved in the design of different mechatronic use cases, following a standard MBSE approach, and how this unified methodology could improve the design process.

References

[1] Herman Van der Auweraer et al. "Virtual engineering at work: the challenges for designing mechatronic products". In: *Engineering with Computers* 29.3 (Sept. 2012), pp. 389–408. ISSN: 0177-0667, 1435-5663.

[2] VDC Research. *Volume 1: Automotive/Transportation; Volume 3: Industrial Automation; and Volume 4: Medical Devices.* Tech. rep. 2008.

[3] Sven Kleiner and Christoph Kramer. "Model Based Design with Systems Engineering Based on RFLP Using V6". In: *Smart Product Engineering: Proceedings of the 23rd CIRP Design Conference, Bochum, Germany, March 11th - 13th, 2013.* Ed. by Michael Abramovici and Rainer Stark. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 93–102. ISBN: 978-3-642-30817-8.

[4] Jeff A. Estefan. "Survey of model-based systems engineering (MBSE) methodologies". In: *Incose MBSE Focus Group* 25 (2007).

[5] Clemens Muenzer and Kristina Shea. "Simulation-Based Computational Design Synthesis Using Automated Generation of Simulation Models From Concept Model Graphs". In: *Journal of Mechanical Design* 139.7 (2017), p. 071101.

[6] Albert Benveniste et al. *Contracts for System Design.* Research 8147. INRIA, 2012.

[7] Mike Nicolai et al. "Generating conceptual designs with an architecture-centric design space exploration method". In: *(in preparation)* ().

[8] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. "Solving SAT and SAT Modulo Theories: From an abstract procedure to DPLL (T)". In: *Journal of the ACM (JACM)* 53.6 (2006), pp. 937–977.

An operational semantics for a weak memory model with buffered writes, message passing, and goroutines

Daniel S. Fava,[1] Martin Steffen,[1] Volker Stolz[1,2] and Stian Valle[1]

[1] Dept. of Informatics, University of Oslo
[2] Western Norway University of Applied Sciences

A *memory model* dictates the order in which memory operations appear to execute; it also affects how processes communicate through shared memory in a parallel computing setting. The design of a proper memory model is a balancing act. On one hand, memory models must be lax enough to allow common hardware and compiler optimizations. On the other hand, the more lax the model, the harder it is for developers to reason about their programs. Though the right balance between relaxation and intelligibility is up for debate, models should, in principle, preclude definitely unwanted behavior. One class of unwanted behavior is called the *out-of-thin-air*. These are counter intuitive results that can be justified through circular reasoning. In this work we propose a memory model that precludes out-of-thin-air behavior, is fairly relaxed (allows writes to different memory location to appear out of order), and is based on a calculus with processes and buffered channels.

The calculus we propose is inspired by the Go programming language developed at Google. Go recently gained traction in networking applications, web servers, distributed software and the like. It features goroutines (i.e., asynchronous execution of function calls resembling lightweight threads) and buffered channel communication in the tradition of CSP or Occam. The *happens-before relation* is used in the Go memory model to describe which reads can observe which writes to the same variable. It says, for example, that *within a single goroutine, the happens-before relation boils down to program order.* The language is particularly suited for concurrent programs. If the effects of a goroutine are to be observed by another, a synchronization mechanism must be used in order to establish a relative ordering between events belonging to the different goroutines. The Go memory model advocates channel communication as the main method of synchronization [2], where *a send on a channel happens before the corresponding receive from that channel completes.* Synchronization is also imposed by the boundedness of channels: *The k^{th} receive on a channel with capacity C happens before the $(k+C)^{th}$ send from that channel completes.*

Abstract syntax

The abstract syntax of the calculus is given in Table 1. *Values* are written generally as v and include booleans, integers, and etc (these more obvious values are not explicitly listed on the table). Less obviously, local variables (or registers) are also counted as values and are denoted r. Names (or references) are also considered values and are denoted n. Names are used, for example, when referring to different channels – when presenting the semantics, we will use c for indicating a reference to a channel.

Expressions are written as e. A `load` z represents the reading the shared variable z into the thread. The syntax for reading global variables makes the shared memory access explicit in this representation: Global variables z, unlike local variables r, are not expressions on their own; they can be used only in connection with loading from or storing to shared memory. Therefore, expressions like $x \leftarrow$`load` z or $x \leftarrow z$ are disallowed.

A new channel is created by `make` (`chan` T, v), where T represents the type of values carried by the channel, and the non-negative integer v the channel's capacity. Sending a value over a channel and receiving a value as input from a channel are written respectively as $v_1 \leftarrow v_2$ and $\leftarrow v$. After the operation `close`, no further values can be sent on the specified channel. Attempting to send values on a

$$
\begin{array}{lcl}
v & ::= & r \mid \underline{n} \\
e & ::= & t \mid v \mid \texttt{load}\, z \mid z := v \\
 & \mid & \texttt{make}\,(\texttt{chan}\, T, v) \mid \leftarrow v \mid v \leftarrow v \mid \texttt{close}\, v \mid \underline{\texttt{pend}\, v} \\
 & \mid & \texttt{if}\, v\, \texttt{then}\, t\, \texttt{else}\, t \mid \texttt{go}\, t \\
g & ::= & v \leftarrow v \mid \leftarrow v \mid \texttt{default} \\
t & ::= & \texttt{let}\, r = e\, \texttt{in}\, t \mid \sum_i \texttt{let}\, r_i = g_i\, \texttt{in}\, t_i
\end{array}
$$

Table 1: Abstract syntax

closed channel leads to a panic. The expression `pend` v represents the state immediately after sending a value over a channel. Note that `pend` is part of the *run-time* syntax as opposed to the user-level syntax, i.e., it is used to formulate the operational semantics of the language. On Table 1, run-time syntaxes are underlined.

Starting a new asynchronous activity, called goroutine in Go, is done using the go-keyword.

Select-statements, written using the $\sum$-symbol, consist of a finite set of guarded branches. Only communication statements, i.e., channel sending and receiving, or the `default`-keyword are allowed as guards (with at most one `default` per select). A channel can be mentioned in more than one guard; sending and receiving guards can also be used in the same select-statement.

The `let`-construct `let` $r = e$ `in` t combines sequential composition and the use of scopes for local variables r: after evaluating e, the rest t is evaluated where the resulting value of e is handed over using r. The let-construct is seen as a binder for variable r in t. It becomes *sequential composition* when r does not occur free in t. We use semicolon as syntactic sugar in such situations.

Operational semantics with write buffering

Programs consist of the parallel composition of goroutines $\langle \sigma, t \rangle$, write events $n(\!|z{:=}v|\!)$, and channels $c[q_f, q_b]$. In the current semantics, read accesses to the main memory cannot be delayed; consequently, there are no read events.

Write events are 3-tuples from $N \times X \times Val$; they record the shared variable being written to and the written value, together with a unique identifier n. A write $n(\!|z{:=}v|\!)$ to variable z is said to be *shadowed* if another write event n' to variable z happened after n but before the current point in time.

In addition to the code t to be executed, goroutines $\langle \sigma, t \rangle$ contain local information about earlier memory interaction. In first approximation, the local state contains information about events which occurred earlier. *Local states* σ are are tuples of type $2^{(N \times X)} \times 2^N$ abbreviated as Σ. We use the notation (E_{hb}, E_s) to refer to the tuples. The first component of the local state E_{hb} contains the identities of all write events that have happened before the current stage of the computation of the goroutine. The second component E_s of the local state represents the set of identities of write events that, at the current point, are shadowed.

From a goroutine's point of view, its reads and writes appear in program order. This is guaranteed by the absence of delayed reads and by disallowing reads from obtaining values of writes that have been shadowed. Also, from a goroutine's point of view, writes from other goroutines may appear out of order. This is because writes are placed on a global pool and because subsequent reads can read any write currently in the pool. The reduction rules for reads and writes are on our technical report [1].

Synchronization between goroutines is achieved by communicating via channels as shown in Table 2. A channel is of the form $c[q_f, q_b]$, where c is a name and (q_f, q_b) a pair of queues referred to as *forward* and *backward* queue respectively. For convenience, we use c_f and c_b when referring to channel's c forward and backward queues respectively. In addition to communicating a value, the queues are

$$\frac{\mathit{fresh}(c)}{\langle \sigma, \texttt{let}\ r = \texttt{make}\ (\texttt{chan}\ T, v)\ \texttt{in}\ t\rangle \to \langle \sigma, \texttt{let}\ r = c\ \texttt{in}\ t\rangle \parallel c_f[] \parallel c_b[]}\ \textsc{R-Make}$$

$$\frac{|q| \leq \mathit{cap}(c) \qquad \neg \mathit{closed}(c_f[q])}{\langle \sigma, c \leftarrow v; t\rangle \parallel c_f[q] \to \langle \sigma, \texttt{pend}\ c; t\rangle \parallel c_f[(v,\sigma) :: q]}\ \textsc{R-Send}$$

$$\frac{|q_1| < \mathit{cap}(c) \qquad \sigma' = \sigma + \sigma''}{c_b[q_2 :: \sigma''] \parallel \langle \sigma, \texttt{pend}\ c; t\rangle \parallel c_f[q_1] \to c_b[q_2] \parallel \langle \sigma', t\rangle \parallel c_f[q_1]}\ \textsc{R-Pend}$$

$$\frac{\sigma' = \sigma + \sigma'' \qquad v \neq \bot}{c_f[q_1 :: (v,\sigma'')] \parallel \langle \sigma, \texttt{let}\ r = \leftarrow c\ \texttt{in}\ t\rangle \parallel c_b[q_2] \to c_f[q_1] \parallel \langle \sigma', \texttt{let}\ r = v\ \texttt{in}\ t\rangle \parallel c_b[\sigma :: q_2]}\ \textsc{R-Receive}$$

$$\frac{\neg \mathit{closed}(c_f[q])}{c_f[q] \parallel \langle \sigma, \texttt{close}\ (c); t\rangle \to c_f[(\bot,\sigma) :: q] \parallel \langle \sigma, t\rangle}\ \textsc{R-Close}$$

Table 2: Operational semantics: message passing

managed so that sends and receives trigger exchanges of happened-before and shadowing knowledge between the communicating partners.

Contributions

We propose an operational semantics for a memory model that precludes out-of-thin-air behavior, is fairly relaxed, and is based on a calculus with processes and buffered channels [1]. We have implemented of the calculus in the $\mathbb{K}$ framework [3], an executable semantics framework based on rewriting logic, and we are currently working on a proof that the model exhibits the so called data-race free guarantee. The DRF guarantee says that data-race free programs (i.e. programs that are properly synchronized) have sequentially consistent behavior. Our goal is to further relax the model by introducing delayed reads, yet, at the same time, keeping out-of-thin-air behavior at bay.

References

[1] Daniel Fava, Martin Steffen, Volker Stolz, and Stian Valle. An operational semantics for a weak memory model with buffered writes, message passing, and goroutines. Technical Report 466, 2017.

[2] Go memory model. The Go memory model. https://golang.org/ref/mem, 2016.

[3] Grigore Roşu and Traian Florin Şerbănuţă. An overview of the K semantic framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010. doi: 10.1016/j.jlap.2010.03.012.

Virtually Timed Ambients: Formalisation and Analysis

Johanna B. Stumpf, Einar Broch Johnsen, and Martin Steffen

University of Oslo, Oslo, Norway
{johanbst, einarj, msteffen}@ifi.uio.no

1 Motivation

The ambient calculus is the process algebra of locations and domains, originally developed by Cardelli and Gordon [2] for distributed systems such as the Internet. We extend the ambient calculus with a notion of virtual time as a resource. The resulting calculus can be used for instance to model aspects of virtualization in cloud computing, where different locations, barriers between locations, and barrier crossing are important features, as well as elasticity which allows to provision virtual resources on-demand.

2 Previous work on timed process algebras

Algebraic concurrency theories such as ACP, CCS and CSP have been extended to deal with time-dependent behaviour in various ways (e.g., [1, 7, 4]). All these approaches describe speed as the absolute *duration* of processes, while in our approach speed describes the relative *processing power* of an ambient.

3 Preliminaries on mobile ambients

An ambient represents the location or domain where a process is running. Ambients can be nested, such that a surrounding *parental ambient* contains *subambients*, and the nesting structure can change dynamically. This is specified by three basic capabilities. The input capability $\mathit{in}\ n$ indicates the willingness of a process, respectivly its containing ambient, to enter an ambient named n, running in parallel outside, e.g., $k[\mathit{in}\ n.P] \mid n[Q] \to n[k[P] \mid Q]$. The output capability $\mathit{out}\ n$ enables an ambient to leave its surrounding ambient n, e.g., $n[k[\mathit{out}\ n.P] \mid Q] \to k[P] \mid n[Q]$. The third basic capability $\mathit{open}\ n$ allows to open an ambient named n which is on the same level as the capability, e.g., $k[\mathit{open}\ n.P \mid n[Q]] \to k[P \mid Q]$. This syntax, as well as the semantics we consider, is based on [6] and largely unchanged compared to [2].

4 Virtually timed mobile ambients

We extend mobile ambients with notions of virtual time and resource consumption. Virtual time is a resource, which is made available to a location by its parental location, similar to time slices that an operating system provisions to its processes. Interpreting the locations of ambients as a place of deployment, each timed ambient is modelled to have a certain computing power, determined by its deployment. Thus, our model of timed ambients uses a *local* notion of time, which, however, is *relative* to the computing power of the embedding, parental ambients. The basic model of virtually timed ambients is described in [5].

Timed systems A timed ambient contains one *local clock* and possibly other timed ambients or classic untimed ambients and processes. A *computing environment* is a timed ambient which contains *resources*, as explained below.

Local clocks To represent the outlined time model, each timed ambient is equipped with one local clock responsible for triggering timed behaviour and local resource consumption. Clocks have a *speed*, interpreted *relative* to the speed of the surrounding timed ambient. The speed s of a clock is given by the tuple (p,q), where p is the number of local time slices emitted for a number q of time slices received from the surrounding ambient. Time slices propagate from parental clocks to clocks in the subambients. Thus, the time in a nested ambient is relative to the global time, depending on the speeds of the clocks of the ambients it is nested in. We assume one universal outermost ambient with a *global clock* triggering the clocks of the local subambients recursively. When moving timed ambients, we must update the clocks to guarantee a correct propagation of time slices. As virtual time is made available to an ambient by its surroundings we have to ensure that a clock distributes time slices to all of its current subambients. Thus, we use an *update function* and define timed capabilities **in** n, **out** n, and **open** n for timed systems, corresponding to the similar untimed capabilites.

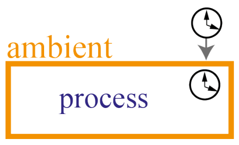


Figure 1: Figurative representation of a virtually timed ambient with a local clock.

Computing resources An ambient's processing power is defined by a *resource process* which transforms the time slices of the local clock into locally consumable resources. Processes expend the processing power of the ambient they are contained in by consuming resources. An ambient with a higher local clock speed produces more resources per parental time slice which in turn allows more work to be done for each parental time slice.

5 Weak bisimulation for timed ambients

We define weak timed bisimulation for virtually timed ambients in [5] as a conservative extension of weak bisimulation for mobile ambients as defined by Merro and Zappa Nardelli [6]. We then define a bisimulation for specific classes of processes which relaxes the condition on timing. This way we can determine if a system is faster than another and give a worst case approximation for this timing difference. We finally show that weak timed bisimulation for ambients completely characterises reduction barbed congruence for virtually timed ambients, extending a result from [6].

6 A Type System with Assumptions and Commitments

Type systems are a common technique to describe the important features of a calculus and provide a way to have the implementation of those features mechanically checked. A basic type system for mobile ambients was first defined in [3] and was mainly concerned with controlling communication and mobility. We are currently working on an assumption and commitment type system with coeffects [8] for virtually timed ambients, concerning time and resources. The type system enables the checking of constraints regarding the capacity of ambients as well as providing an upper bound for the resource usage. Additionally, we aim to obtain a *subject reduction result* for the type system for virtually timed ambients, showing that the upper bounds on resources and the number of subambients are preserved under reduction.

7 Concluding Remarks

Virtualization opens for new and interesting foundational models of computation by explicitly emphasizing deployment and resource management. We introduce virtually timed ambients, a formal model of hierarchical locations of execution with explicit resource provisioning. Resource provisioning for virtually timed ambients is based on virtual time, a local notion of time reminiscent of time slices for virtual machines in the context of nested virtualization. This way, the computing power of a virtually timed ambient depends on its location in the deployment hierarchy. To reason about timed behavior in this setting, we define weak timed bisimulation for virtually timed ambients as a conservative extension of bisimulation for mobile ambients, and show that the equivalence of bisimulation and reduction barbed congruence is preserved by this extension. Introducing an assumption and commitment type system with coeffects allows for the checking of timing and resource constraints on ambients and gives an upper bound on the resources used by a process.

References

[1] J. C. M. Baeten and J. A. Bergstra. *Real Time Process Algebra* . Technical Report CS-R 9053, Centrum voor Wiskunde en Informatica (CWI), 1990.

[2] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Theor. Comput. Sci.,*, 240(1): 177–213, 2000.

[3] Luca Cardelli and Andrew D. Gordon. Types for the Ambient Calculus. In *Information and Computation*, 177(2:)160–194,2002.

[4] Matthew Hennessy and Tim Regan. A process algebra for timed systems. In *Information and Computation*, 117(2):221–239, 1995.

[5] E. B. Johnsen, M. Steffen, and J. B. Stumpf. A calculus of virtually timed ambients. In *Postproceedings of selected contributiions to the 23rd International Workshop on Algebraic Development Techniques (WADT 2016)*, 2017.

[6] Massimo Merro and Francesco Zappa Nardelli. Behavioral theory for mobile ambients. *J. ACM*, 52(6):961–1023, November 2005.

[7] Faron Moller and Chris Tofts. A temporal calculus of communicating systems. In J. C. M. Baeten and J. W. Klop, editors, *Proc. CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*, pages 401–415. Springer, 1990.

[8] T. Petricek, D. Orchard, and A. Mycroft. Coeffects: unified static analysis of context-dependence. In *Proceedings of International Conference on Automata, Languages, and Programming — Volume Part II*, ICALP 2013.

Assessing the Coverage of Formal Specifications

(Extended Abstract)

Dominic Steinhöfel

TU Darmstadt, Dept. of Computer Science, Darmstadt, Germany
steinhoefel@cs.tu-darmstadt.de

Abstract. Deductive program verification is an intricate and time-consuming task, in spite of significant advances in state-of-the-art program provers. While proving the correctness of programs with respect to existing specifications can already be difficult, it can be even more demanding to come up with sensible specifications for methods and especially for loops. Another issue is related to programs heavily making use of software libraries: Their verification can be considered almost infeasible due to the lack of formal specifications of the libraries. We propose a method for assessing the coverage/strength of formal specifications based on "facts" extracted using heavyweight symbolic execution. We envision that this method can be employed for (1) assisting verification engineers in the incremental specification of programs, (2) comparing different specifications for the same program, and (3) obtaining information for specification generation tools. Our approach has been implemented as a prototype for Java which uses the heavyweight symbolic execution system KeY as a backend. We studied its practicability with several small examples and plan to conduct a more extensive case study in the near future.

1 Introduction and Preliminaries

In the past decades, *deductive software verification* techniques evolved from theoretical approaches reasoning about simple while languages to systems such as Spec#, Frama-C, OpenJML and KeY [1] which are capable of proving complex properties about programs in industrial programming languages such as C, C# and Java [2]. Still, one major challenges remains: The creation of suitable formal specifications for the software that should be proven correct. In [2], a sorting routine of the Java standard library, TimSort, is formally specified and the correctness of the routine w.r.t. the specification is shown using the KeY verifier. Although the specification does not even cover the main aspects of sortedness and permutation, a substantial effort was required for their construction. Current approaches to automatic specification generation are not yet capable of creating specifications that can be used for functional verification in the general case. We propose an approach using Symbolic Execution (SE) to extract "facts" about Java methods specified with the Java Modelling Language (JML) and to check the coverage of provided method and loop specifications w.r.t. those facts. One of the main applications of our method is aiding verification engineers in incrementally increasing the strength of their specifications; also, the facts might

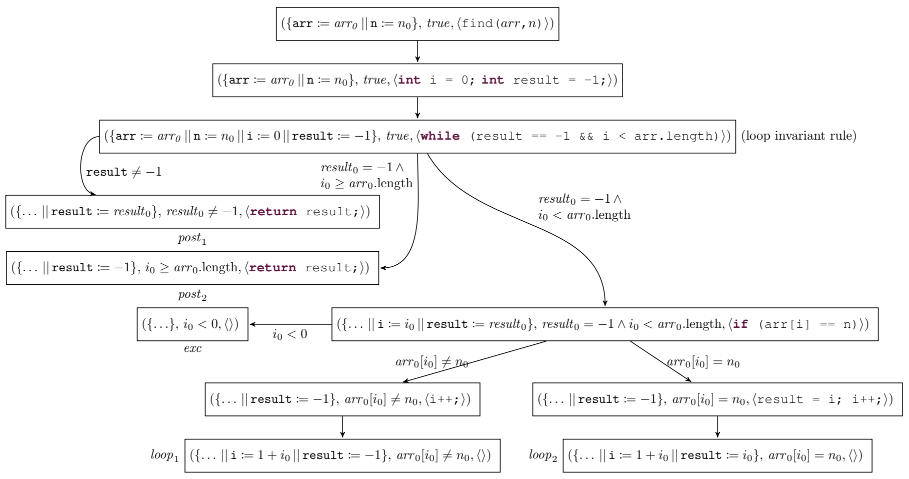

({arr := arr₀ || n := n₀}, true, ⟨find(arr, n)⟩)
({arr := arr₀ || n := n₀}, true, ⟨int i = 0; int result = -1;⟩)
({arr := arr₀ || n := n₀ || i := 0 || result := −1}, true, ⟨while (result == -1 && i < arr.length)⟩)
(loop invariant rule)
result ≠ −1
result₀ = −1 ∧ i₀ ≥ arr₀.length
result₀ = −1 ∧ i₀ < arr₀.length
({… || result := result₀}, result₀ ≠ −1, ⟨return result;⟩)
post₁
({… || result := −1}, i₀ ≥ arr₀.length, ⟨return result;⟩)
post₂
({…}, i₀ < 0, ⟨⟩)
exc
i₀ < 0
({… || i := i₀ || result := result₀}, result₀ = −1 ∧ i₀ < arr₀.length, ⟨if (arr[i] == n)⟩)
arr₀[i₀] ≠ n₀
arr₀[i₀] = n₀
({… || result := −1}, arr₀[i₀] ≠ n₀, ⟨i++;⟩)
({… || result := −1}, arr₀[i₀] = n₀, ⟨result = i; i++;⟩)
loop₁
({… || i := 1 + i₀ || result := −1}, arr₀[i₀] ≠ n₀, ⟨⟩)
loop₂
({… || i := 1 + i₀ || result := i₀}, arr₀[i₀] = n₀, ⟨⟩)

Fig. 1: Simplified SET for the `find` method in Listing 1

be used by specification generation tools to successively create stronger method contracts and loop invariants. One of our visions is that the approach could help comparing different specifications of software libraries submitted to a "collaborative specification" platform, which could be based on an imaginary rewards system in the fashion of StackOverflow.

We introduce some preliminaries, and refer the reader to [1,3] for more details. SE results in an Symbolic Execution Tree (SET) consisting of SE *states*. An SE state is a triple of a *symbolic state* $\{\texttt{x} := t \,\|\, \ldots\}$, a *path condition* (a closed first-order formula determining a path) and a *program counter*. The latter is a dynamic logic formula with a program, determining the following subtree; in the presentation in this paper, we abbreviate it to the next statement to execute.

2 Extracting Facts by Symbolic Execution

```
public int find(int[] arr, int n) {
    int i = 0, result = -1;
    while (result == -1 &&
          i < arr.length) {
        if (arr[i] == n)
            result = i;
        i++;
    }
    return result;
}
```

Listing 1: `find` method

Listing 1 depicts a simple method for finding an integer in an array. Fig. 1 shows the corresponding SET. The tree contains five (symbolically) feasible final states: The states $post_1$ and $post_2$ represent return states after loop termination, while $loop_1$ and $loop_2$ arise from the execution of the loop body. The exceptional state *exc*, in which `i` is negative, is practically infeasible, but still contained in the SET if the loop invariant does not exclude that case. From those states, we extract the following facts: (1) Two *post condition facts* $\texttt{result} \doteq result_0$ (where $result_0 \neq -1$) and $\texttt{result} \doteq -1$ (if n was not found) from the states $post_1$ and $post_2$, (2) two *loop body facts* $\texttt{i} \doteq 1 + i_0$ and $\texttt{result} \doteq -1$ from $loop_1$, (3) another two loop body facts $\texttt{i} \doteq 1 + i_0$ and $\texttt{result} \doteq i_0$ from $loop_2$. In addition, we can obtain two special fact categories depending on the method post condition and loop invariant: (4) A set of *use case facts* from all *post* nodes, containing one fact for each conjunctive element in the specified post condition of the method. An unsatisfied use case fact means that either the post condition cannot be satisfied by the method, or that the loop invariant is too weak. (5) A set of *exception facts* if the method precondition or loop invariant are not excluding potential runtime exceptions to be thrown. Uncovered exception facts are reported by our tool.

For each fact, we distinguish three cases: Whether the fact is *covered*, i.e., implied by the results of SE and the specification elements; whether it is *"trivially" covered*, i.e., without the specification elements and can thus be ignored in strength estimation, or whether it is *overapproximated*, that is, the fact (e.g., $n \doteq 3$) implies the specification (e.g., $n \geq 0$).

3 Applications

For effectively using facts analysis in incremental specification development, we suggest to begin with an initial, intuitive method post condition. Afterward, the loop invariant (and method precondition) should be strengthened until no more exception or use case facts are uncovered. Finally, post condition and loop body facts can successively be taken into account until the specification reaches a satisfactory state. One result of our analysis is a percentage number of covered facts (the *strength*), where overapproximated facts are weighted by a factor smaller than 1. These numbers can be employed to compare different existing specifications, or to guide the search in an automatic approach for specification generation. In the latter case, also the feedback about facts of the program can be helpful. At our companion website `key-project.org/papers/coverage-formspec`, we provide a Java executable of our tool and an evaluation for the `find` method.

4 Future Work and Conclusion

We presented an approach for estimating the strength of formal specifications written in JML for Java programs. To the best of our knowledge, there is no other approach analyzing the strength, or comparing different versions, of formal specifications. We envision that our technique can be useful for both manually and automatically creating method and loop specifications. In the future, we plan to perform an extensive case study to assess the utility of the approach.

References

1. Ahrendt, W., Beckert, B., et al. (eds.): Deductive Software Verification – The KeY Book, LNCS, vol. 10001. Springer International Publishing (2016)
2. Gouw, S.d., Rot, J., et al.: OpenJDK's java.utils.Collection.sort() is broken: The good, the bad and the worst case. In: Kroening, D., Pasareanu, C.S. (eds.) Proc. of the 27th Intl. Conf. on Computer Aided Verification. Springer (2015)
3. Scheurer, D., Hähnle, R., et al.: A General Lattice Model for Merging Symbolic Execution Branches. In: Ogata, K., Lawford, M., et al. (eds.) ICFEM 2016, Proceedings. pp. 57–73. Springer International Publishing (2016)

Versioned Simulation Testing for Concurrent and Distributed Models

Lars Tveito, Einar Broch Johnsen, and Rudolf Schlatte

Department of Informatics, University of Oslo, Oslo, Norway
`{larstvei,einarj,rudi}@ifi.uio.no`

1 Introduction

We address the problem of validating models of concurrent and distributed systems using techniques for systematic path exploration. It is often not enough to validate that a system performs functionally correct; one also wants to know what level of throughput the system can provide, and if the system can maintain a quality of service (QoS) that respects a given service level agreement (SLA) [6]. Our goal is to provide efficient and reliable testing techniques for concurrent models, addressing both functional correctness and whether resource consumption of a system is in accordance with a given SLA.

2 Context

For our purposes, we leverage Real-Time ABS [5], a timed, object-oriented modeling language, featuring distributed concurrent object groups that communicate via asynchronous method calls and futures [3]. This expressive language is suitable for modeling distributed systems at a high level of abstraction, and in particular, it captures deployment decisions in terms of timing properties and the cost of operations in the system. Real-Time ABS has an associated simulation tool to help analyze these models, which scales to real-world industrial use cases [4, 2]. However, a single simulation cannot describe all possible executions of a concurrent system and provides no guarantees for detecting issues that only occur in certain execution paths. In the following section, we show how we have extended the Real-Time ABS simulator to address this problem.

3 Path Exploration for Real-Time ABS

To explore many execution paths by means of the Real-Time ABS simulator, we have developed a system which controls the systematic exploration of different simulations runs. Running many simulations can be computationally expensive, but is highly parallelizable, so the system has been designed to allow them to run simultaneously on different computers to lower the overall execution time of the path exploration.

We have created a naming scheme, which provides stable ABS process identities which persist between runs. A scheduling *trace* is a sequence of such stable ABS process identities and is used to obtain reproducible simulations. If a simulator is given a trace, then it runs deterministically until the end of the trace, after which it runs non-deterministically until termination.

When a simulator runs non-deterministically and makes a scheduling decision, it communicates which processes were and were *not* scheduled to the system; scheduling the unexplored processes will lead to unexplored paths. The system keeps track of paths that are unexplored and dynamically spawns new simulator instances to cover them. The process terminates if there are no more unexplored paths, at which point we have achieved full path exploration.

In addition, the simulators report their current state to the system at each scheduling point, and a state space is constructed which shows how the scheduling decisions affect the state over time. A visual representation is provided, which for small examples is much simpler to analyze than the outputs from the simulators. However, manual inspection is both tedious and error-prone. We plan to address the issue of expressing SLA requirements as formal properties and enable verification of such properties against the results of the simulations.

Currently, we make no effort to prune the number of paths explored. Implementing advanced partial order reduction (POR) techniques, as found in [1, 7], could allow this solution to scale to real-world use cases.

4 Simulation Testing

To improve performance we want to minimize the number of simulations needed, without restricting the number of tests that can be executed. To this end, we want to extend our current system to simulation testing. Simulation testing is the decomposition of simulation and testing, where the results of a simulation are stored in a database and tests are written as queries to the database, rather than embedding the tests in the model directly. The idea is central in a testing library called Simulant[1], but has, to the best of our knowledge, not been discussed in scientific literature.

In contrast to traditional testing, simulation testing does not require the model (or program) to be executed for each test. Most systems should, ideally, be equipped with an extensive test suite (of hundreds or thousands of tests); not having to produce the data that is subject to testing for each test can be a huge performance gain, and can enable systems to embody sizable test suits. When testing concurrent systems, each test must be run against exponentially many different executions, which emphasizes the need to avoid running the executions for each test in our problem domain.

5 Towards Versioned Simulation Testing

We propose a notion of *versioned* simulation testing, which is based on using knowledge of past simulations to avoid rerunning simulations that will not provide new results, even when the model is changed. Our motivation for migrating towards versioned simulation testing is to facilitate incremental simulation testing for models under rapid development.

Pruning the number of paths to explore has the highest potential for significant performance gains in our system. With versioned simulation testing, we want to prune the number of paths based on what simulations have already been executed. To realize this idea, we must be able to detect if a prefix of a trace from an earlier version of the model is *compatible* with the current model. This is the case if the executed code is unaltered between the two versions of the model. For example, if the only change is in the `else`-branch of an `if`-statement, then it is not necessary to do path exploration before the execution reaches that `else`-branch.

The planned system architecture is outlined in Figure 1. The user is responsible for developing the model, initiating path exploration, and writing queries to the database for analysis. The system automatically runs the appropriate simulations and stores information in the database. To detect the simulations that do not need to be executed, the system relies on a normalized

[1]`https://github.com/Datomic/simulant`

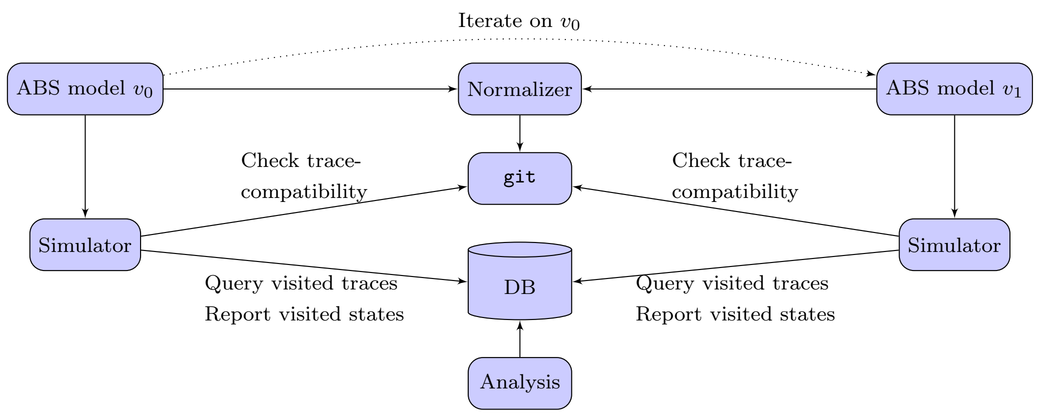


Figure 1: The planned system architecture for versioned simulation testing

version of the model which is under version control. The normalization phase generates a version of the model which makes it easy to check whether the code between two points in the code has been altered by performing a line-by-line comparison.

References

[1] E. Albert, P. Arenas, and M. Gómez-Zamalloa. "Actor- and Task-Selection Strategies for Pruning Redundant State-Exploration in Testing". In: *Formal Techniques for Distributed Objects, Components, and Systems*. Vol. 8461. Lecture Notes in Computer Science. Springer, 2014, pp. 49–65.

[2] E. Albert et al. "Formal modeling and analysis of resource management for cloud architectures: an industrial case study using Real-Time ABS". In: *Service Oriented Computing and Applications* 8.4 (2014), pp. 323–339.

[3] F. S. de Boer, D. Clarke, and E. B. Johnsen. "A Complete Guide to the Future". In: *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007*. Vol. 4421. Lecture Notes in Computer Science. Springer, 2007, pp. 316–330.

[4] F. S. de Boer et al. "Formal Modeling of Resource Management for Cloud Architectures: An Industrial Case Study". In: *Service-Oriented and Cloud Computing - First European Conference, ESOCC 2012*. Vol. 7592. Lecture Notes in Computer Science. Springer, 2012, pp. 91–106.

[5] E. B. Johnsen, R. Schlatte, and S. L. T. Tarifa. "Integrating deployment architectures and resource consumption in timed object-oriented models". In: *Journal of Logical and Algebraic Methods in Programming* 84.1 (2015), pp. 67–91.

[6] B. Nobakht, S. de Gouw, and F. S. de Boer. "Formal Verification of Service Level Agreements Through Distributed Monitoring". In: *Service Oriented and Cloud Computing - 4th European Conference, ESOCC 2015*. Vol. 9306. Lecture Notes in Computer Science. Springer, 2015, pp. 125–140.

[7] S. Tasharofi et al. "TransDPOR: A Novel Dynamic Partial-Order Reduction Technique for Testing Actor Programs". In: *Formal Techniques for Distributed Systems - Joint 14th IFIP WG 6.1 International Conference, FMOODS 2012 and 32nd IFIP WG 6.1 International Conference, FORTE 2012*. Vol. 7273. Lecture Notes in Computer Science. Springer, 2012, pp. 219–234.

Security Functionality of IoT Devices

Elahe Fazeldehkordi[1], Olaf Owe[2] and Toktam Ramezanifarkhani[1]
[1] Department of Informatics/Department of Technological Systems, University of Oslo, Oslo, Norway.
[2] Department of Informatics, University of Oslo, Oslo, Norway.
elahefa@ifi.uio.no, olaf@ifi.uio.no, toktamr@ifi.uio.no

Abstract

Internet of Things (IoT) devices offer different technologies connected to the Internet to provide several applications to connect worldwide physical objects into a unified system. Privacy has been frequently identified as a main concern for IoT devices, when dealing with/managing personal information. Despite this, most existing works on privacy requirements deal with privacy requirements as a special case of security requirements, and there seems to be no established ontology for IoT systems and their privacy. Ontologies are known to be rich sources of knowledge, and, being structured and equipped with reasoning features, they form a powerful tool to handle requirements. Therefore, key aspects of privacy are, usually, overlooked. In this context, wrong design decisions might be made due to insufficient understanding of privacy concerns.

In order to have a good understanding of privacy and its concepts, firstly we need to know the actual difference between privacy and security. In this work, we address IoT security concepts, functionalities and requirements. In the second step, we will address IoT privacy, in particular privacy concepts whose major purpose is to identify the main concepts and relations for capturing privacy requirements and functionalities. We further identify and analyze the main privacy-related functionalities, requirements, and concepts, as well as the corresponding relations, in order to propose an ontology that can be used to capture privacy requirements for IoT devices.

1 Introduction

With the aim of providing an advanced mode of communication between various systems and devices, as well as facilitating the interaction of humans with the virtual environment, Internet of Things (IoT) finds its application in almost any field [17,18,19]. Based on the large number of low-cost sensors and wireless communication abilities, the sensor network technology puts forward new demands to the Internet technology. It will bring huge changes to the future society, change our way of life and business models [1]. Apart from the benefits of IoT devices, as with all things using the Internet infrastructure for information exchange, IoT too is susceptible to various security issues and has some major privacy concerns for the end users. As such IoT, even with all of its advanced capabilities in the information exchange area, is a flawed concept from the security viewpoint and proper steps has to be taken in the initial phase before going for further development of IoT for an effective and widely accepted adoption.

Privacy has been frequently identified as a main concern for IoT devices while dealing with or managing personal information. Privacy breaches might lead to lack of appropriate security policies, bad security practices, attacks, data thefts, etc. [2,12]. Despite this, few works focus on considering privacy during the system design. More specifically, most existing works on privacy requirements often deal with them either as non-functional requirements with no specific criteria on how such requirements can be met, or as a part of security [11,13], i.e., focusing mainly on confidentiality and overlooking important privacy aspects. Therefore, key aspects of privacy are, usually, overlooked. Efforts that have been made to clarify the privacy concept have been linked to more refined concepts such as secrecy, person-hood, control of personal information, etc., but there is no consensus on the definition of these concepts or which of them should be used to analyze privacy [9]. This has resulted in a lot of confusion among designers and stakeholders, which has turned to wrong design decisions. In this context, wrong design decisions might be made due to insufficient understanding of privacy concerns. In other words, dealing with privacy-related concerns is very crucial these days because privacy breaches in IoT devices may result in huge costs as well as other negative consequences in the long term [2,5,6,7,8]. However, most of these breaches can be avoided if privacy requirements of the system-to-be were captured properly during system design, where privacy requirements aim to capture the types and levels of protection necessary to meet the privacy needs of the users. In this context, a well-defined privacy ontology that captures privacy-related concepts along with their interrelations would constitute a great step forward in designing privacy-aware systems.

As used in the knowledge representation domain, the term "ontology" stands for an explicit, formal, machine-readable semantic model that defines the classes, instances of the classes, inter-class relations and data properties relevant to a problem domain [20]. Ontologies have been proven to be a key success factor for eliciting high quality requirements. They can facilitate and improve the job of requirements engineers [4,10,14], since they can reduce the conceptual vagueness and terminological confusion by providing a shared understanding of the related concepts between designers and stakeholders [16]. In addition, the ontology should capture privacy requirements in their social and organizational context. Most complex systems these days (e.g., healthcare systems, smart cities, etc.) are socio-technical systems [3], which consist not only of technical components but also human interfaces along with their interrelations. Focusing on the technical aspects and leaving the social and organizational aspects outside the system's boundary leaves the system open to different kinds of vulnerabilities that might manifest themselves in the social interactions and/or the organizational structure of the system [15].

In order to come up with a more precise and accurate privacy ontology for IoT, it is essential to have a good understanding of privacy, privacy concepts and requirements, and to know the actual difference between privacy and security. First of all, we need to have a clear view of security, including the following questions: What is security? What are security functionalities and their requirements? What is the relationship between security and privacy? What is special for IoT devices? As a possible answer to these questions, we have started our work with clarifying security concepts, functionalities and requirements. For this matter, the terminology and definitions of necessary security concepts have been collected from text books based on the National Institute of Standards and Technology (NIST) and the ISOs 27k family. In a second step we will address privacy, in particular privacy concepts whose major purpose is to identify the main concepts/relations for capturing privacy requirements/functionalities. We will further analyze the selected privacy-related functionalities/requirements/concepts and the corresponding relations, and identify the main ones, in order to propose an ontology that can be used to capture and to deal with privacy requirements. This work is therefore intended to be a starting point to address the problem of identifying a core privacy ontology for IoT devices. Security functionality is the security-related features, functions, mechanisms, services, procedures, and architectures implemented within organizational information systems or the environments in which those systems operate [21]. Security functionality can be obtained by employing within the information systems and supporting infrastructure of the organization, a combination of management, operational, and technical security controls.

Figure 1 presents part of a diagram representing the top-level view of the security functionality to the relevant security controls and requirements based on National Institute of Standards and Technology (NIST) Special Publication 800-53 and International Organization for Standardization/International Electrotechnical Commission (ISO/IEC). This diagram represents our initial work so far and is a first step in the design of an ontology, since the structure of the figure defines concepts and relations that are central in the ontology. Organizations that have implemented or plan to implement the Framework for Improving Critical Infrastructure Cybersecurity can use the mapping of the security functionality. The security functionality mapping information can also be useful to organizations that wish to demonstrate compliance to the security requirements in the context of their established information security programs, when such programs have been built around the NIST or ISO/IEC security controls.

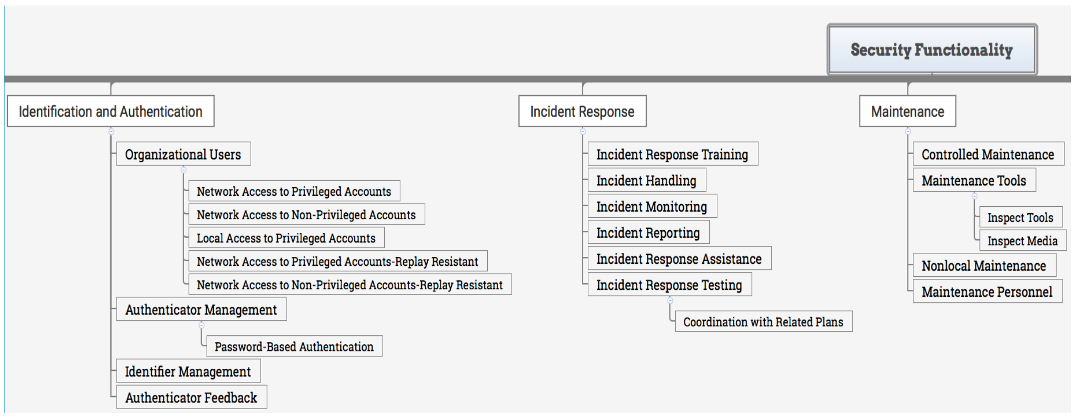

Security Functionality
Identification and Authentication
Organizational Users
Network Access to Privileged Accounts
Network Access to Non-Privileged Accounts
Local Access to Privileged Accounts
Network Access to Privileged Accounts-Replay Resistant
Network Access to Non-Privileged Accounts-Replay Resistant
Authenticator Management
Password-Based Authentication
Identifier Management
Authenticator Feedback
Incident Response
Incident Response Training
Incident Handling
Incident Monitoring
Incident Reporting
Incident Response Assistance
Incident Response Testing
Coordination with Related Plans
Maintenance
Controlled Maintenance
Maintenance Tools
Inspect Tools
Inspect Media
Nonlocal Maintenance
Maintenance Personnel

Figure 1: A top-level view of the security functionality diagram

References

[1] Kumar, J. S., & Patel, D. R. (2014). *A survey on internet of things: Security and privacy issues.* International Journal of Computer Applications, 90(11).
[2] Acquisti, A., Friedman, A., & Telang, R. (2006). *Is there a cost to privacy breaches? An event study*. ICIS 2006 Proceedings, 94.

[3] Churchman, C. W., & Verhulst, M. (Eds.). (1960). *Management Sciences, models and techniques* (Vol. 2). Pergamon.
[4] Dzung, D. V., & Ohnishi, A. (2009, November). *Ontology-based reasoning in requirements elicitation*. In Software Engineering and Formal Methods, 2009 Seventh IEEE International Conference on (pp. 263-272). IEEE.
[5] Gellman, R. (2002). *Privacy, consumers, and costs-how the lack of privacy costs consumers and why business studies of privacy costs are biased and incomplete*. In Digital Media Forum, Ford Foundation.
[6] Camp, L. J. (2002, July). *Designing for trust*. In Workshop on Deception, Fraud and Trust in Agent Societies (pp. 15-29). Springer Berlin Heidelberg.
[7] Campbell, K., Gordon, L. A., Loeb, M. P., & Zhou, L. (2003). *The economic cost of publicly announced information security breaches: empirical evidence from the stock market*. Journal of Computer Security, 11(3), 431-448.
[8] Hong, J. I., Ng, J. D., Lederer, S., & Landay, J. A. (2004, August). *Privacy risk models for designing privacy-sensitive ubiquitous computing systems*. In Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques (pp. 91-100). ACM.
[9] Solove, D. J. (2005). *A taxonomy of privacy*. U. Pa. L. Rev., 154, 477.
[10] Souag, A. (2012, June). *Towards a new generation of security requirements definition methodology using ontologies*. In 24th International Conference on Advanced Information Systems Engineering (CAiSE'12) Gdańsk, Poland, 25-29 June 2012 (pp. 1-8).
[11] Kalloniatis, C., Kavakli, E., & Gritzalis, S. (2008). *Addressing privacy requirements in system design: the PriS method*. Requirements Engineering, 13(3), 241-255.
[12] Labda, W., Mehandjiev, N., & Sampaio, P. (2014, March). *Modeling of privacy-aware business processes in bpmn to protect personal data*. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (pp. 1399-1405). ACM.
[13] Zannone, N. (2006). *A requirements engineering methodology for trust, security, and privacy* (Doctoral dissertation, PhD thesis, University of Trento).
[14] Kaiya, H., & Saeki, M. (2006, September). *Using domain ontology as domain knowledge for requirements elicitation*. In Requirements Engineering, 14th IEEE International Conference (pp. 189-198). IEEE.
[15] Liu, L., Yu, E., & Mylopoulos, J. (2003, September). *Security and privacy requirements analysis within a social setting*. In Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International (pp. 151-161). IEEE.
[16] Uschold, M., & Gruninger, M. (1996). *Ontologies: Principles, methods and applications*. The knowledge engineering review, 11(2), 93-136.
[17] Pontin, J. (2005). *ETC: Bill Joy's Six Webs*. MITTechnology Review, 29.
[18] Shen, G., & Liu, B. (2011, May). *The visions, technologies, applications and security issues of Internet of Things*. In E-Business and E-Government (ICEE), 2011 International Conference on (pp. 1-4). IEEE.
[19] Borgohain, T., Kumar, U., & Sanyal, S. (2015). *Survey of security and privacy issues of Internet of Things*. arXiv preprint arXiv:1501.02211.
[20] Ulicny, B. E., Moskal, J. J., Kokar, M. M., Abe, K., & Smith, J. K. (2014). *Inference and ontologies*. In Cyber Defense and Situational Awareness (pp. 167-199). Springer International Publishing.
[21] Ross, R., VISCUSO, P., GUISSANIE, G., DEMPSEY, K., & RIDDLE, M. (2015). *Protecting controlled unclassified information in nonfederal information systems and organizations*. NIST Special Publication, 800, 171.

Using Coloured Petri Nets for Resource-Aware Program Analysis

Anastasia Gkolfi, Einar Broch Johnsen, and Ingrid Chieh Yu

Department of Informatics, University of Oslo, Norway
{natasa, einarj, ingridcy}@ifi.uio.no

1 Introduction

The research described here, funded by the NFR project CUMULUS, is dedicated to establish semantic foundations for *cloud aware computing*. The goal is to integrate resource management in static analysis for verification of cloud aware programs.

The starting point for our work is the Abstract Behavioural Specification language (ABS) [6], a modelling language with a formal semantics which combines the actor model of concurrency with object orientation to target distributed systems. It supports resource awareness, which make it suitable for cloud applications. In particular, ABS has features like deployment components, cost annotations, time, move of objects, and transfer of resources (see [7]), which allow implementation of deployment scenarios at the programming level.

These novel features of ABS make analysis very challenging. At this time, there are no tools for ABS beyond simulation which support resource management. Our approach is based on a high level of abstraction, achieved by combining Abstract Interpretation [2, 3] with Model Checking [1].

2 The Approach

As mentioned above, the lack of tools supporting resource analysis for ABS programs and the need for automation led to model-driven approach through a higher level of abstraction.

The type of model that has been chosen was coloured Petri nets [5], since they form a mathematically founded formalism that is well known for its suitability to model concurrency, synchronization and resources. In addition, choosing high-level Petri nets (here, coloured Petri nets) was crucial for reducing the size of the model and advantaging of the automation that existing tools can offer (CPN Tools), while hierarchies allow structural separation of the features under consideration.

3 Outline of the Petri Net Model

The model should be able to faithfully over-approximate the cost, hence the communication history of ABS programs. As a consequence, in addition to the in-

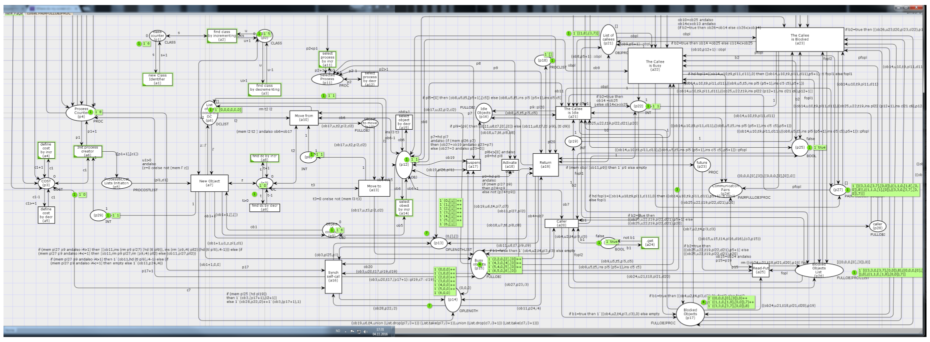
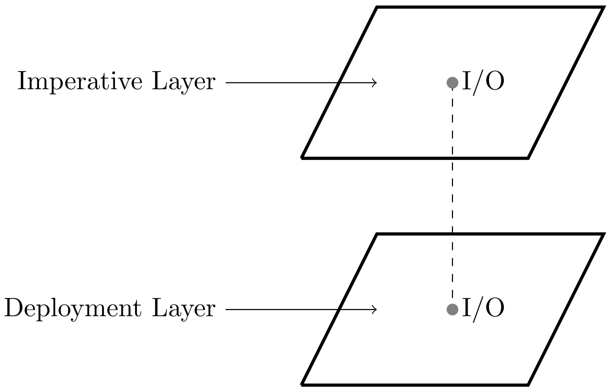


Fig. 1. ABS semantics implemented as a coloured Petri net with CPN Tools

formation about the resources, it should also simulate the communication mechanism of ABS. This led to the construction of the model in two layers:



The imperative layer [4] can over-approximate the communication topology of ABS programs (see Fig. 1) , so static deadlock cycles can be easily detected by using the model checker of CPN Tools. The deployment layer (ongoing work, see Fig. 2) can check resource availability and handle resource management. The two layers exchange information such that object execution (imperative layer) is related to resource availability (deployment layer). Notice here that, since the model is based on the semantics, the markings are in simulation relation with ABS program configurations, so we keep *a single and fixed size model for any ABS program*. In addition to that, a soundness proof supports fully the implementation.

4 Conclusions and Future Work

The idea behind our approach was to construct a sound model based on the semantics of ABS language to abstractly simulate program configurations and use the model checker of CPN Tools as an abstract interpreter. As a first step,

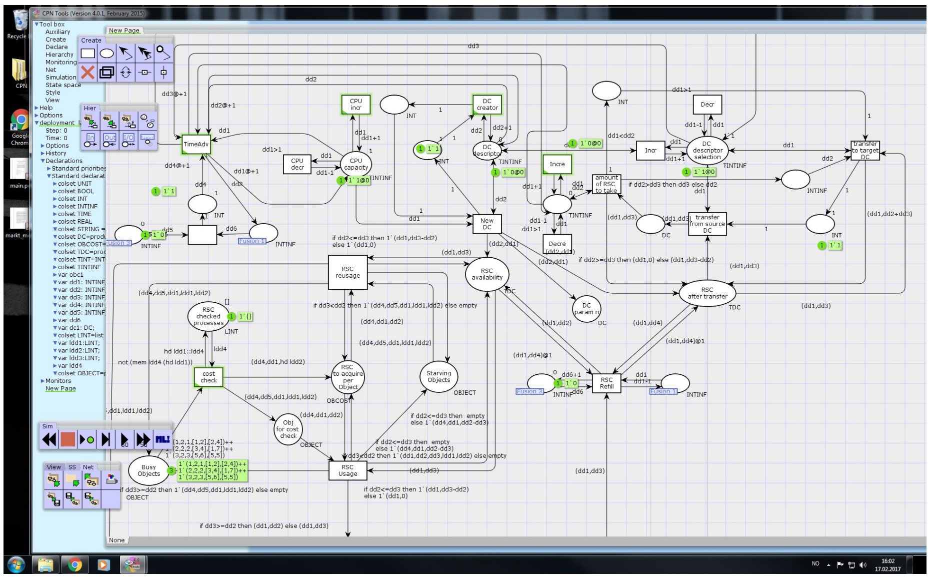


Fig. 2. ABS deployment semantics implemented as a colored Petri net with CPN Tools

we constructed a model of the imperative layer of the language which can detect static deadlocks. Afterwards, we implemented the deployment layer which adds the resource awareness to the imperative layer. Work in progress concerns resource management and optimization at the level of the model as well as a concretization function which over-approximates program states for the optimized resource distribution.

References

1. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 1999.
2. P. Cousot. Semantic foundations of program analysis. In *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Inc., 1981.
3. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Fourth Annual Symposium on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.
4. A. Gkolfi, C. C. Din, E. B. Johnsen, I.C. Yu. Translating Active Objects into Colored Petri Nets for Communication Analysis. To appear in LNCS, Springer, 2017.
5. K. Jensen and L. M. Kristensen. *Coloured Petri Nets – Modelling and Validation of Concurrent Systems*. Springer, 2009.
6. E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen. ABS: A core language for abstract behavioral specification. In *Proc. Formal Methods for Components and Objects (FMCO 2010)*, volume 6957 of *Lecture Notes in Computer Science*, pages 142–164. Springer, 2011.
7. E. B. Johnsen, R. Schlatte, and S. L. Tapia Tarifa. Integrating deployment architectures and resource consumption in timed object-oriented models. *Journal of Logical and Algebraic Methods in Programming*, 84(1):67—91, 2015.