Høgskulen på Vestlandet

HVL-rapport nr. 14
2020

Proceedings of the PhD Symposium at iFM'19 on Formal Methods: Algorithms, Tools and Applications (PhD-iFM'19)

Violet Ka I Pun and Volker Stolz

Faculty of Engineering and Science
Department of Computer Science, Electrical Engineering and Mathematical Sciences

Høgskulen på Vestlandet
2020

HVL-rapport frå Høgskulen på Vestlandet nr. 14

ISSN 2535-8103
ISBN 978-82-93677-34-5

Preface

This research report contains the proceedings of the PhD Symposium at iFM'19 on Formal Methods: Algorithms, Tools and Applications (PhD-iFM'19), which was held on 3 December, 2019 at Western Norway University of Applied Sciences, Bergen, Norway. The program of the symposium consisted of an invited talk by Andreas Griesmayer (ARM, Cambridge, UK) and 11 short presentations. Each short presentation received advices and feedbacks from a senior researcher. Among the 11 short presentations, 5 submitted their contributions in the form of extended abstracts, which were included in this report.

Table of Contents

Static Detection of Distributed Denial of Service Attacks in Active Object Systems
Elahe Fazeldehkordi, Olaf Owe, and Toktam Ramezanifarkhani 1

Learning From Families: Inferring Behavioral Variability From Software Product Lines
Carlos Diego Nascimento Damasceno 4

Coverage Analysis of SML Expressions in CPN Models
Faustin Ahishakiye, Volker Stolz, and Lars Michael Kristensen 7

Automatic Loop Invariant Generation Using Predicate Abstraction for Dependence Analysis
Asmae Heydari Tabar, Richard Bubel, and Reiner Hähnle 10

Towards Better Data Structures for Numerics such as Optimal Transport
Justus Sagemüller, Olivier Verdier, and Volker Stolz 13

Static Detection of Distributed Denial of Service Attacks in Active Object Systems

Elahe Fazeldehkordi, Olaf Owe, and Toktam Ramezanifarkhani

Department of Technology Systems / Department of Informatics, University of Oslo, Norway
{elahefa,olaf,toktamr}@ifi.uio.no

Extended abstract

Denial of Service (DoS) and Distributed DoS (DDoS) attacks, with even higher severity, are among the major security threats for distributed systems, and in particular in the financial sector where trust is essential.

In this paper, our aim is to develop an additional layer of defense in distributed agent systems to combat such threats. We consider a high-level imperative and object-oriented language for distributed systems, based on the actor model with support of asynchronous and synchronous method interaction. This setting is appealing in that it naturally supports the distribution of autonomous concurrent units, and efficient interaction, avoiding active waiting and low-level synchronization primitives such as explicit signaling and lock operations. It is therefore useful as a framework for modeling and analysis of distributed service-oriented systems.

Our language supports efficient interaction by features such as asynchronous and non-blocking method calls and first-class futures, which are popular features applied in many distributed systems today. However, these mechanisms make it even easier for an attacker to launch a DDoS attack, because undesirable waiting by the attacker can be avoided with these mechanisms. We propose an approach consisting of static analysis. We identify and prevent potential vulnerabilities in asynchronous communication that directly or indirectly can cause call-based flooding of agents. More precisely, we adapt a general algorithm for detecting call flooding to the setting of security analysis and for detection of distributed denial of service attacks adding support for one-to-many and many-to-one attacks. The algorithm detects call cycles that might cause overflow in the incoming queues of one or more communicating agents. Each cycle may involve any number of agents, possible involving the attacked agent(s). The high-level framework considered here is relevant for a large class of programming languages and service-oriented systems.

The approach is limited to the software level, and we do not consider the network layer nor the data transport layer. As we are using static detection, there will be a degree of over-approaximation. The static approach could be complemented with runtime DDoS detection checking, including also the network layer and the data transport layer. For more detail see a full version of this paper [1].

As an example of a possible DoS attack, we have considered an attack on customers that can be caused by a financial institution. This kind of attacks can be intentional by attackers or unintentional due to a mistake from a financial institution. Here we considered the attack as a result of an update from the financial institution that was supposed to give better efficiency. We imagine that the financial institution has a subscription *service* for customers, so that customers can subscribe, to receive the latest news about shares, or unsubscribe. In order to handle many customers, a number of *proxies* have been used. In the first version, there is a *get* method on the proxy side, with the effect that the proxy should wait for the information related to shares and funds to be available, and then pass it on to the customers. Therefore,

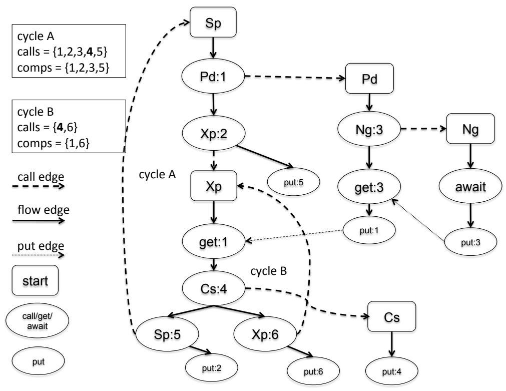

Figure 1: The graph and call/comp sets for the original version of the program.

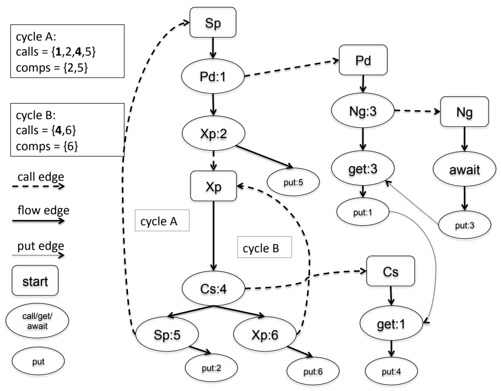

Figure 2: The graph and call/comp sets for the modified version of the program.

there is a waiting time here in the proxy and also heavy network traffic since copies of the information is transferring over network to all subscribers, however, customers might not need the information so often. Therefore, the financial institution may think of another solution for better efficiency and shift the *get* method to the customer side, which means that the customer will wait for the information to be available, instead of proxy, and when it's available and desired he/she can get it from the future reference. Now, the problem of heavy network traffic has been solved, and also the *proxy* does not need to wait and can respond to other requests at any time. Nevertheless this solution can cause flooding cycles in the program and this can lead to DoS attack. Following [2], the static analysis of the first version and the modified version are shown in Figures 1 and 2. Cycles A and B in the modified version are both dangerous and cause a DoS attack on the system since in contrast to the first version, there is no *get* method in the cycles that can regulate the speed of the cycles.

References

[1] E. Fazeldehkordi, O. Owe, and T. Ramezanifarkhani. A language-based approach to prevent DDoS attacks in distributed financial agent systems. In *1st International Workshop on Security for Finan-*

cial Critical Infrastructures and Services (FINSEC'19), ESORICS'19. To appear in LNCS, Springer, Sep., 2019. (16 pages).

[2] O. Owe and C. McDowell. On detecting over-eager concurrency in asynchronously communicating concurrent object systems. *J. Logical and Alg. Methods in Prog.*, 90:158 – 175, 2017.

Learning From Families: Inferring Behavioral Variability From Software Product Lines

Carlos Diego Nascimento Damasceno

University of Sao Paulo (BR) and University of Leicester (UK)
damascenodiego@usp.br

Abstract

Family models are behavioral specifications extended with variability constraints that enable efficient model-based analysis of software product lines (SPL). Albeit reasonably efficient, the creation and maintenance of family models are time-consuming and error-prone, especially if there are large models or crosscutting features. In this PhD project, we investigate the problem of learning family models from SPLs. Our initial contributions are two-fold: (1) partial-Dynamic $\mathtt{L}^*_{\mathtt{M}}$, a novel adaptive algorithm to speed up automata learning by exploring models from alternative software versions *on-the-fly*; and (2) $\mathtt{FFSM}_{\mathtt{Diff}}$, a fully automated technique to learn family models by comparing, merging and annotating finite state machines with variability constraints. Our experiments have shown that our techniques are more efficient than the state-of-the-art of adaptive learning in terms of queries and that succinct family models with fewer states can be learnt, especially if there is high feature reuse. We envisage that our studies can leverage model-based techniques to cases where models are non-existent or outdated and will scale better than independently exploring several versions of evolving systems or product models from configurable systems.

1 Introduction

The modeling and analysis of software product lines (SPL) are known to be challenging; they should incorporate variability to express product-specific behavior to avoid/minimize redundant computations of shared assets and cater for feature interactions [8]. Thus, substantial effort has been spent for developing analysis techniques specifically tailored to product families.

Family-based analysis operates on a single artifact, referred to as *family model*, that is annotated with variability constraints to express variability in terms of states and transitions specific to product configurations. This modeling approach paves the way for efficient model-based testing and verification of SPLs. Nevertheless, the creation and maintenance of test models are known to be time consuming and error-prone, especially if there are large SPLs or crosscutting features; and the traceability between the family- and variability models can be complex due to crosscutting features [3]. Added to this, as requirements change and product instances evolve, the lack of maintenance may render models outdated [2]. To tackle these issues, we proposed this PhD project to investigate how automata model learning [9] can be lifted to the family-based level to support the extraction of family models from SPLs.

Model learning has been a popular approach to automatically derive behavioral models from a system under learning (SUL) by posing tests as queries, i.e., *transfer* and *separating* sequences, to reach and distinguish states [9]. It has been harnessed for a wide range of problems [1]. However, there is a lack of studies about how to cope with variability in time and space [6].

2 Approach

Applying model learning to real systems can be hampered by constant changes along their life-cycle [5], as it may require *learning from scratch*. Adaptive learning attempts to speed up learning by reusing knowledge (i.e., sequences) from alternative/previous versions of a SUL. Studies have shown that reusing sequences from pre-existing models can reduce the cost for learning models from updates. However, after several changes, old separating and transfer sequences may render *redundant* and *deprecated* queries, respectively [2].

2.1 Learning to Reuse: Adaptive Learning for Evolving Systems [2]

We improve upon the state-of-the-art by introducing `partial-Dynamic` $\mathtt{L}^*_{\mathtt{M}}$ ($\partial\mathtt{L}^*_{\mathtt{M}}$), an adaptive algorithm that runs an *on-the-fly* exploration of reused models to avoid irrelevant queries [2].

To achieve this, our algorithm explores *transfer sequences* to find redundancy. Then, given a subset of "useful" transfer sequences, we designed an optimization technique to find deprecated *separating sequences* and hence, the smallest subset with equivalent separating capability, named *experiment cover.* These subsets of transfer and separating sequences initialize the $\mathtt{L}^*_{\mathtt{M}}$ algorithm for learning Mealy machines [7]. Our experiments showed that $\partial\mathtt{L}^*_{\mathtt{M}}$ is less sensitive to evolution and more efficient (i.e., requires less queries) than the state-of-the-art for adaptive learning. The paper has been published at the iFM'19 [2].

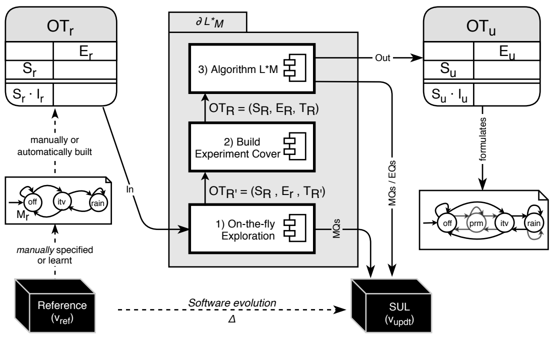

Figure 1: `partial-Dynamic` $\mathtt{L}^*_{\mathtt{M}}$

2.2 Learning from Difference: An Automated Approach for Learning Family Models [3]

Within SPLs, similar challenges may emerge and hamper family-based testing and verification. Families of software products share a common and managed set of features and hence, their behavior tend to have commonalities and variabilities [8]. Thus, model learning for SPLs should avoid redundant effort, and at the same time cater for variability and feature interaction.

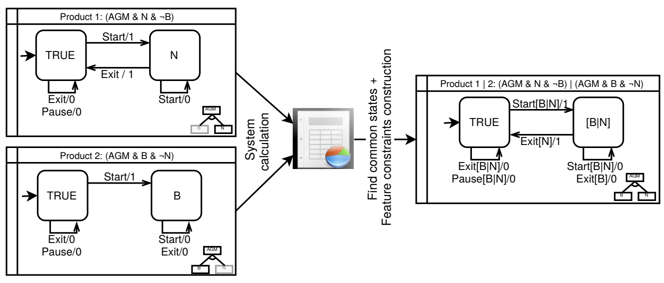

Figure 2: $FFSM_{Diff}$

We have designed $FFSM_{Diff}$, an automated technique to learn family models by comparing, merging and annotating product models. Our technique is presented in terms of featured finite state machines (FFSM) [4], a family-based notation that unifies Mealy Machines from SPLs by annotating states and transitions with variability constraints. A schematic representation of our approach is depicted in Figure 2.

Our technique allows to (i) learn succinct FFSMs from two product models, and (ii) include novel product-specific behavior into an existing FFSM. Our results support the hypothesis that family models can be effectively merged into succinct FFSMs with fewer states, especially if there is high feature sharing among products. These findings indicate that $FFSM_{Diff}$ paves the way for family model learning techniques, which are still understudied; and efficient family-based analysis, even if there are no models specified *a priori.* The full paper has been published at the 23rd International Systems and Software Product Line Conference (SPLC'19) [3].

3 Final Remarks and Next Steps

Real systems pass through many changes along their life-cycle and, as we often do not know how states may have changed, their models tend to become outdated. To deal with these issues, we have designed two techniques for learning models from evolving systems [2] and product families [3]. Our techniques improve upon the state-of-the-art and are complementary to each other in the sense that they pave the way for an *active family model learning* framework.

As the next step of this PhD project, we will propose the concept of active family model learning. In Figure 3, we show our vision of active family model learning. Our vision of *active family model learning* stands for a framework where SPLs can have their family models harvested by re-using *partial family models*, i.e., family models describing subsets of valid product instances from SPLs, to steer an active model learning process [9]. We envisage that such variability-aware model learning framework will scale better than exhaustively and independently applying adaptive learning to product instances or software releases.

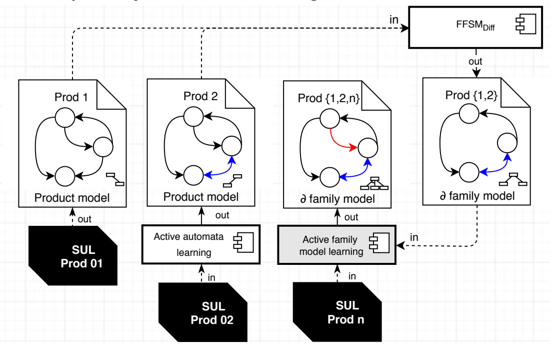

Figure 3: Active family model learning

References

[1] Bernhard K. Aichernig, Wojciech Mostowski, Mohammad Reza Mousavi, Martin Tappler, and Masoumeh Taromirad. Model learning and model-based testing. In Amel Bennaceur, Reiner Hähnle, and Karl Meinke, editors, *Machine Learning for Dynamic Software Analysis: Potentials and Limits: International Dagstuhl Seminar 16172*, pages 74–100, Cham, 2018. Springer.

[2] Carlos Diego N. Damasceno, Mohammad Reza Mousavi, and Adenilso Simao. Learning to reuse: Adaptive model learning for evolving systems. In *Integrated Formal Methods - 15th International Conference, IFM 2019, Bergen, Norway, December 2-6, 2019, Proceedings*. Springer, 2019.

[3] Carlos Diego Nascimento Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simao. Learning from difference: An automated approach for learning family models from software product lines. In *Proceeedings of the 23rd International Systems and Software Product Line Conference - Volume 1, SPLC 2019*, Paris, France, 2019. ACM Press.

[4] Vanderson Hafemann Fragal, Adenilso Simao, and Mohammad Reza Mousavi. *Validated Test Models for Software Product Lines: Featured Finite State Machines*, pages 210–227. Springer International Publishing, Cham, 2017.

[5] David Huistra, Jeroen Meijer, and Jaco van de Pol. Adaptive learning for learn-based regression testing. In Falk Howar and Jiri Barnat, editors, *Formal Methods for Industrial Critical Systems*, Lecture Notes in Computer Science, pages 162–177, Switzerland, 9 2018. Springer.

[6] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, Secaucus, NJ, USA, 2005.

[7] Muzammil Shahbaz and Roland Groz. Inferring mealy machines. In Ana Cavalcanti and Dennis R. Dams, editors, *FM 2009: Formal Methods*, pages 207–222, Berlin, Heidelberg, 2009. Springer.

[8] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. A classification and survey of analysis strategies for software product lines. *ACM Comput. Surv.*, 47, June 2014.

[9] Frits Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, January 2017.

Coverage Analysis of SML Expressions in CPN Models

Faustin Ahishakiye, Volker Stolz, and Lars Michael Kristensen

Western Norway University of Applied Sciences, Bergen, Norway
firstname.lastname@hvl.no

1 Introduction

Most of the software are being developed using model-driven software approaches. This allows to make a step from code-driven to model-driven systems preserving the combination of programming language and some models to obtain a scalable modeling language for concurrent systems. This includes, for example, the use of the standard meta-language (SML) expressions in a model. One potential candidate to ensure this combination is the colored Petri nets (CPNs) model [1], which has extensively been used mainly in concurrent and distributed systems [6]. CPNs provide the formal foundation for modeling concurrency and synchronization and a programming language provides the primitives for modeling data manipulation and creating compact and parameterizable models. Coverage analysis is important for programs in relation to fault detection and different structural coverage criteria are required for software safety and quality design assurance [3]. However, there is no coverage analysis of the SML expressions integrated into a CPN model, and the same structural coverage criteria used for source code can also be applied here. In this paper, we combine the dynamic analysis (simulation and state spaces) of CPN models with coverage analysis of SML guard and arc expressions. We present some examples using CPN Tools based on modified condition decision coverage (MC/DC) criterion. According to the definition of MC/DC [2, 5], each condition in a decision has to show an independent effect on that decision's outcome by: (1) varying just that condition while holding fixed all other possible conditions or (2) varying just that condition while holding fixed all other possible conditions that could affect the outcome. MC/DC is required by certification standards such as, the DO-178C [4] in the domain of avionic software systems, as a coverage criterion for safety critical software at the conditions level. It is highly recommended due to its advantages of requiring few test cases ($n + 1$ for n conditions) and its uniqueness due to the independence effect of each condition. To observe individual sub-expressions of guards and arcs, we have to add transitions to the CPN, which do not affect the overall workings of the net. Our state space exploration (SSE) and simulation results show full MC/DC coverage of the arc and guard expression. However, the guard expression requires an extra transition to acquire the false outcome before analyzing MC/DC. Results show that coverage analysis can be a useful feature in a CPN model.

2 Arc and guard SML expressions coverage analysis

Based on MC/DC criterion, we intend to show that coverage analysis can be a useful feature for a CPN model and how to collect coverage statistics. Consider the example of CPN model in Figure 1. A set of inputs of type integer are the initial markings (inputs tokens to start place) of our model and are equally distributed to the transition through the variable m and mapped to three variables a, b and c. These variables are so far listed as conditions ($a \geq 2, b \leq 2, c \geq 5$) on the arc expression in order to find a set of input test cases as a Boolean list. The test cases are evaluated through the arc SML expression $(a \geq 2 \land b \leq 2) \lor c \geq 5$ (highlighted in Figure

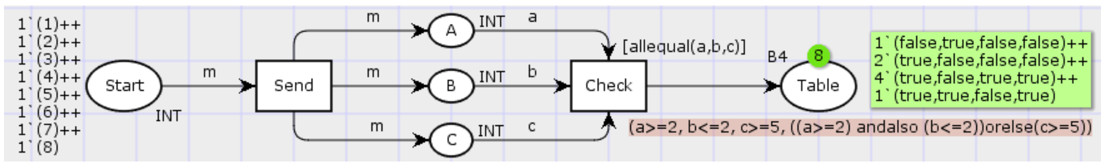

Figure 1: CPN model with SML arc expression

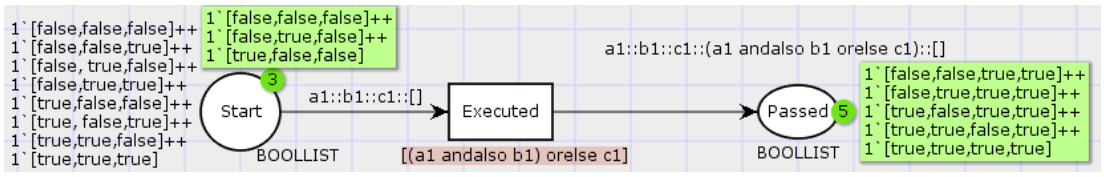

Figure 2: CPN model with SML guard expression

1) which is appended to the tuple of conditions $a \geq 2, b \leq 2$, and $c \geq 5$. The place called "*table*" contains all conditions evaluations and the number of inputs which evaluated each row is appended at the beginning of that row. One of the challenges with CPN is that to analyze coverage, we need to modify the original CPN model. An observational function defining the MC/DC criterion is written in a separate SML file and is called during the simulation. The coverage analysis is summarized in the following steps:

1. For a CPN model with an arc SML expression, add the MC/DC coverage analyzer (consisting of additional transitions and places to check coverage of each condition)

2. Start the state space exploration (SSE)

3. Run the simulation: the table contains evaluations of all conditions and the outcome. The MC/DC covered conditions have the token which has moved from the not covered place to the covered place. Otherwise the tokens remain in the not covered place.

Let the Boolean guard SML expression be $(a1 \wedge b1) \vee c1$, highlighted in Figure 2, the "*Executed*" transition fires only if the predicate (decision) in the guard expression is evaluated to true. Consequently, only the test cases with true outcome pass and the test cases with false outcome are blocked. To evaluate MC/DC, we add a parallel transition named "*Blocked*" with a negated guard SML expression (highlighted in Figure 3) with respect to the first transition guard expression, to allow all blocked test cases to pass through it. A table combining all the test cases from both transitions is constructed as shown in Figure 3. At this stage, the MC/DC coverage analysis proceeds in the same manner as explained in steps 2 and 3 above.

Table 1: SSE of standard CPN behavioral properties

Properties	Nodes	Arcs	Execution time (seconds)
Arc SML expression without MC/DC	23781	122221	221
Arc SML expression with MC/DC	48837	189641	300
Guard SML expression without MC/DC	6561	34992	27
Guard SML expression with MC/DC	11052	64632	65

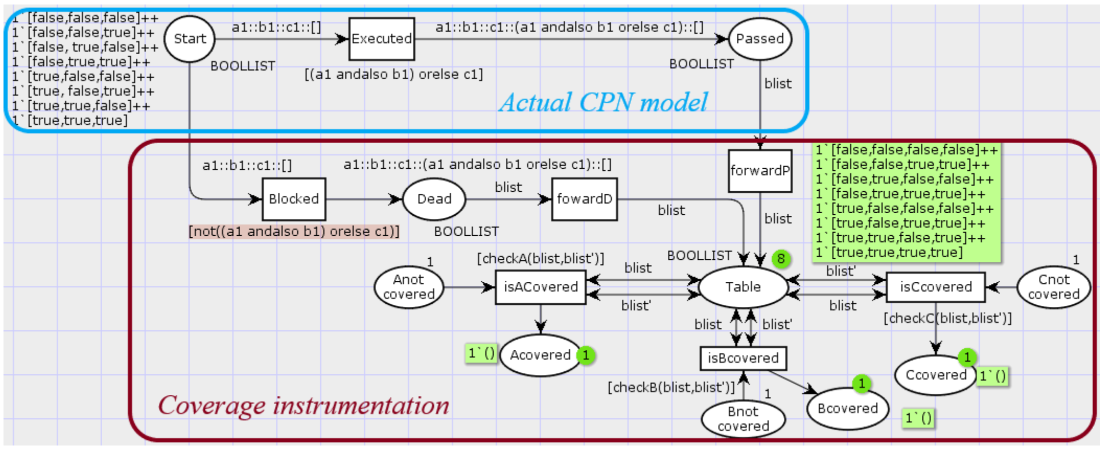

Figure 3: CPN model with MC/DC coverage analysis of SML guard expression

3 Performance evaluation and conclusion

SSE is the main approach to the verification of CPN models. It provides information regarding the visited nodes, statistics on state space, and strongly connected components. It gives information on standard behavioral properties such as liveness (list of home markings, dead and live transitions), boundedness (bounds for the number of tokens on each place), and statistical information regarding size and time used for state space generation. Table 1 compares results of the SSE for arc and guard expression model with and without MC/DC coverage analyzer. The MC/DC coverage analyzer requires additional cost in terms of time and memory due to the increasing number of arcs and nodes compared to the CPN model without the coverage analyzer. The number of arcs, nodes, and execution time almost double when the coverage analyzer is added. However, it provides the means to know if all bindings in the guard and arc expressions are covered as shown in Figure 3. For the future work, we will reduce the number of arcs and nodes added by extraction from state space. In addition, we will extend the coverage analysis to if-then-else expressions, consider other coverage criteria such as branch coverage and check related properties. Moreover, we plan to investigate how this analysis can be automated.

References

[1] Kurt Jensen and Lars M. Kristensen. Colored Petri Nets: A graphical language for formal modeling and validation of concurrent systems. *Communications of the ACM*, 58:61–70, 05 2015.

[2] Chilenski John J. and Miller Steven P. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, 9(5):193–200, 1994.

[3] Hayhurst Kelly J., Veerhusen Dan S., Chilenski John J., and Rierson Leanna K. A Practical Tutorial on Modified Condition/Decision Coverage. Technical report, NASA, 2001.

[4] Frederic Pothon. DO-178C/ED-12C versus DO-178B/ED-12B: Changes and Improvements. Technical report, AdaCore, 2012.

[5] Leanna Rierson. *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance.* CRC Press, 2013.

[6] Hong Zhu and Xudong He. A theory of testing high level Petri nets. In *Proceedings of International Conference on Software: Theory and Practice, 16th IFIP World Computer Congress*, 2000.

Automatic Loop Invariant Generation Using Predicate Abstraction for Dependence Analysis[*]

Asmae Heydari Tabar, Richard Bubel, and Reiner Hähnle

Technische Universität Darmstadt, Department of Computer Science,
64289 Darmstadt, Germany
{heydaritabar,bubel,haehnle}@cs.tu-darmstadt.de

Abstract

Analysis or reasoning about data dependences must be fully automatically to be used as part of compilers or parallelization tools. To this extent, we present our current work-in-progress on automated reasoning of our logic-based dependence analysis –presented in our main paper at iFM 2019 (15th International Conference on integrated Formal Methods)– in presence of loops. For being able to reason about loops our approach relies on loop invariants that have to be provided manually by the user. Here, we report on our ongoing work on generating loop invariants for data dependence properties automatically based on predicate abstraction.

1 Introduction

Analyzing and reasoning about data dependences in programs is necessary, for instance, to perform sound program optimizations or to parallelize programs while preserving the computation behavior of the original program. In our main paper [3] we present a program logic and calculus tailored towards dependence analysis. For a dependence analysis to be used in compilers (for program optimization) or parallelization tools (e.g. DiscoPop [8]) high automation is crucial.

Our approach in [3] can be used to reason (almost) fully automatically about data dependences while maintaining high precision. A major drawback is that in presence of loops, loop invariants are required that have to be provided by the user. Here, we want to focus on how to reduce (or even eliminate) this hidden interaction by automatically generating loop invariants for data dependence properties.

2 A Program Logic for Data Dependence Analysis

In [3] we extend the program logic JavaDL [1] for sequential Java with a formal semantics of read and write data dependences. State-of-the-art tools for parallelization use approaches that over- as well as under-approximate to compute dependences and they lack a formal foundation. But our approach can reason about dependences soundly and with full precision. It has been implemented in the deductive verification tool KeY [1] for the target language Java.

A standard programming language semantics based on traces, i.e. finite or infinite sequences of states, is insufficient to characterize read and write dependences. Rather than supplying a special purpose semantic construct, we decided to give a general solution. It is well-known (e.g., [2]) that *non-functional* properties (such as dependences) can often be formally specified with the help of *ghost variables.* These are memory locations not part of the program under verification that record meta properties of program execution (e.g., memory access). In our

[*]This work was funded by the Hessian LOEWE initiative within the Software-Factory 4.0 project.

semantics we introduced a *single* ghost variable `ma` (for ***m**emory **a**ccess*) that records the whole *history* of memory accesses in the current execution as a finite sequence. But how are properties about the content of `ma` expressed, given that it is not directly accessible in the syntax? This is achieved with *memory access predicates* that allow us to express data dependence properties of locations: `noRAW`, `noWAR`, `noWAW`, `noR`, `noW`.

These predicates take a set of program locations ls as argument and evaluate to true iff in the memory access history of the current state for *no* location in ls there is a read-after-write, write-after-read, write-after-write, write or read, respectively. The axiomatization of these predicates defines the relation between the memory accesses recorded by `ma` and the presence or absence of their respective data dependence property.

3 Automatic Loop Invariant Generation

Using the defined predicates made us able to prove the absence of data dependences in loop-free programs fully automatically. But for reasoning about loops, the user has to provide loop invariants. This lack of automation in dependence analysis of programs containing loops was one of the issues we mentioned as future work in the iFM paper [3].

Currently, we are investigating how to make use of loop invariant generation techniques to derive the necessary invariants automatically. There are many ways for automated loop invariant generation. In our case heuristics-driven techniques such as predicate abstraction [6] looks promising. Predicate abstraction is a well-known abstract interpretation [4] technique in which the abstract domain is constructed from a finite set of predicates over program variables. These predicates can be generated in a heuristic manner from the program text or specified manually by the programmer [5, 9]. Loop invariants are the Boolean combination of the given set of predicates.

As pointed out above, we are optimistic to be able to generate loop invariants mostly automatically via predicate abstraction, because: (i) the domain is specific since the loop invariants mainly capture properties about memory access; (ii) main properties can be specified by predicates; (iii) arguments of predicates (memory locations) can be easily abstracted or approximated.

We want to use the introduced data dependence predicates (`noRAW`, `noWAR`, etc.) to construct a set of atomic formulas from which the loop invariant is generated. Arguments of them can be obtained based on the accessed program locations in the loop. Then, refined Boolean combination of these predicates can form the part of the loop invariant which is concerned about dependence properties. Functional properties of loops also have to be considered, for instance to ensure that array accesses are within bounds. For these we rely on standard techniques from predicate abstraction approaches, like [9].

R.Q. 1: How to determine an initial set of atomic data dependence formulas for predicate abstraction? For the data dependence part we use the set of dependence predicate symbols as starting point. To construct atomic formulas these predicates need to be equipped with a set of program locations as argument.

A naive approach is to produce all combinations based on over-approximations of the accessed program locations in the loop. To reduce the number of candidate program location sets to be considered, we plan to apply techniques that allow, for instance, to describe partitions of an array following an approach like [7].

Hence, the above research question reduces to "How can we compute a safe abstraction or approximation of program locations while having an acceptable level of precision?"

R.Q. 2: How to reduce the number of loop invariant candidates in presence of data dependence formulas? From the set S of atomic data dependence formulas determined in R.Q. 1 loop invariant candidates are generated as Boolean combinations of the formulas in S. A naive approach again leads to redundant combinations. For example, if we know that $\texttt{noR}(\{o, f\})$ is an invariant of the loop then dependent predicates like $\texttt{noRAW}(\{o, f\})$ and $\texttt{noWAR}(\{o, f\})$ do not need to be considered as part of the loop invariant as they are implied. To answer this research question we will investigate means to reduce the number of candidates by exploiting these kind of subsumption relations.

References

[1] Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M. (eds.): Deductive Software Verification—The KeY Book: From Theory to Practice, LNCS, vol. 10001. Springer (2016)

[2] Albert, E., Bubel, R., Genaim, S., Hähnle, R., Díez, G.R.: A formal verification framework for static analysis—as well as its instantiation to the resource analyzer COSTA and formal verification tool KeY. Software & Systems Modeling 15(4), 987–1012 (2016)

[3] Bubel, R., Hähnle, R., Tabar, A.H.: A program logic for dependence analysis. In: Ahrendt, W., Tapia Tarifa, S.L. (eds.) Integrated Formal Methods - 15th International Conference, IFM 2019. Lecture Notes in Computer Science, vol. 11918, pp. 83–100. Springer (2019)

[4] Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Graham, R.M., Harrison, M.A., Sethi, R. (eds.) Conference Record of the Fourth Symposium on Principles of Programming Languages. pp. 238–252. ACM (1977)

[5] Flanagan, C., Qadeer, S.: Predicate abstraction for software verification. In: Launchbury, J., Mitchell, J.C. (eds.) The 29th Symposium on Principles of Programming Languages. pp. 191–202. ACM (2002)

[6] Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: Grumberg, O. (ed.) Computer Aided Verification, 9th Intl. Conf., CAV, Haifa, Israel. LNCS, vol. 1254, pp. 72–83. Springer (1997)

[7] Hähnle, R., Wasser, N., Bubel, R.: Array abstraction with symbolic pivots. In: Ábrahám, E., Bonsangue, M.M., Johnsen, E.B. (eds.) Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 9660, pp. 104–121. Springer (2016)

[8] Huda, Z.U., Atre, R., Jannesari, A., Wolf, F.: Automatic parallel pattern detection in the algorithm structure design space. In: 30th IEEE Intl. Parallel and Distributed Processing Symp. pp. 43–52. IEEE Computer Society (2016)

[9] Weiß, B.: Predicate abstraction in a program logic calculus. In: Leuschel, M., Wehrheim, H. (eds.) Integrated Formal Methods, 7th International Conference, IFM 2009. Lecture Notes in Computer Science, vol. 5423, pp. 136–150. Springer (2009)

Towards Better Data Structures for Numerics such as Optimal Transport

Justus Sagemüller[1], Olivier Verdier[1], and Volker Stolz[1]

Western Norway University of Applied Sciences, Bergen, Norway
`{jsag,over,vsto}@hvl.no`

Abstract

In machine learning, it is often necessary to compare data distributions. One way of doing that is *Optimal transport*. OT is a useful tool for assessing the difference/divergence (*Wasserstein* or *Earth mover distance*) between probability distributions, histograms etc., or for interpolating between them, particularly in case of non-overlapping distributions.

The *Cuturi-Sinkhorn algorithm* is an efficient means of calculating OT for arbitrary metrics on the base space. However it is in practice carried out only on a *discretised representation* of the distributions.

We implement the Sinkhorn algorithm on a data structure which handles the infinite dimensionality of the continuous distribution space through lazy evaluation. Apart from safer, easier handling of what resolution is necessary, this includes the ability to express with *types* the mathematical meaning of a function or distribution.

Our Sinkhorn implementation does work, but floating-point instability is observed in certain cases. Interestingly, it depends on whether the distributions are considered as functions or *dual-functions*, a distinction that does not appear in the conventional a-priori finite-dimensional reduction. We discuss this and whether it is possible to use types to prevent such issues, or to warn about them.

Probability distributions are the foundation of statistics and its applications. In the discrete case, such distributions are readily represented by a concrete probability value for each possible event – i.e. as a function from the set of events X to probabilities $[0, 1]$. Many applications however require, at least conceptually, probability distribution on *continuous* spaces X (real intervals, Euclidean planes, manifolds etc.). Standard procedure is to discretise such spaces to a finite-dimensional approximation and then carry out any computer algorithms on the resulting discrete space of distributions.

The resulting view suggests that distributions can still be understood as functions on X, as *probability density functions*. Indeed this view has some merit thanks to the Riesz-Fréchet representation theorem, but fundamentally, distributions are better understood as (normalised) functionals: as linear mappings from the space of functions $X \to \mathbb{R}$ to $\mathbb{R}$. That includes in particular also discrete, point-distributions on the continuous space (the prototype being the *Dirac distribution*, which expresses that all the events happen at a single point on the real line, for example all at 0). Such distributions do not correspond to any function $X \to \mathbb{R}$, but they do correspond to functionals $(X \to \mathbb{R}) \to \mathbb{R}$, namely pointwise evaluation.

A natural question to ask is, given two distributions $\mathbb{P}_\mathrm{r}$ and $\mathbb{P}_\mathrm{g}$, how similar or dissimilar they are. This is of immediate importance in machine learning. In the function-view of distributions, a naïve attempt would be to sum/integrate the point-wise difference between them ($\mathcal{L}^1$ difference); in practice the Kullback-Leibler [3] divergence family is more common, including the Jensen-Shannon divergence which can be used as a proper *metric* on the distribution space. All of those share the problem that they do not take the topology of the base space X into account. In particular, point-distributions are almost always classified as infinitely far apart, even when the points lie arbitrarily close in X. This is for example a problem in generative adversarial networks, leading to mode collapse. Resolution-limit / smearing can avoid this, but at the obvious cost of loss of resolution and without addressing the underlying problem. What can address it [1] is switching to a metric that does consider the topology of X. The *Wasserstein metric* or *earth mover distance* measures how far the "mass" in the distribution $\mathbb{P}_\mathrm{r}$ needs to be moved across X in order to obtain $\mathbb{P}_\mathrm{g}$. This movement process, with the minimum movement-distance, is called *optimal transport*.

Sinkhorn algorithm. *The* optimal transport between $\mathbb{P}_\mathrm{r}$ and $\mathbb{P}_\mathrm{g}$ can be seen as a joint distribution γ, i.e. a distribution on $X \times X$, such that the marginal on one side is $\mathbb{P}_\mathrm{r}$ and on the other $\mathbb{P}_\mathrm{g}$, and the integrated cost is minimal. The *Sinkhorn OT algorithm* [2] obtains this iteratively. The following is a novel formulation of this algorithm, taylored towards distributions as duals of abstract function spaces.

A function space is a *commutative algebra*, with both addition and multiplication defined point-wise. Let A and B be such algebras; we consider $\mathbb{P}_\mathrm{r} \in A^*$ and $\mathbb{P}_\mathrm{g} \in B^*$, i.e. in the dual space. There is now a multiplication operation available

$$\begin{aligned} & (\cdot) : A^* \times A \to A^* \\ \text{or} \qquad & (\cdot) : B^* \times B \to B^* \\ & (u \cdot \psi)\phi := u(\psi \cdot \phi) \end{aligned}$$

Here, the functional $u \cdot \psi$ is defined by its result for a function ϕ, and $\psi \cdot \phi$ denotes the pointwise multiplication, i.e. standard multiplication in the algebra A or B, respectively.

Furthermore there are division operations of the same types, that use the point-wise reciprocal of ϕ. This, like the multiplication, is linear in its left argument.

Now, given a positive "matrix", in the form of a linear mapping $K : A^* \to B$ and its adjoint $K^* : B^* \to A$, and two (normalised) distributions $\mathbb{P}_\mathrm{r} \in A^*$, $\mathbb{P}_\mathrm{g} \in B^*$, the *Sinkhorn algorithm* [5] provides a uniquely-defined $\gamma : A \to B^*$ of the form

$$\gamma = (v\cdot) \circ K \circ (u\cdot)^*$$

with $u \in A^*, v \in B^*$ such that $\gamma(\mathbf{1}_A) = \mathbb{P}_\mathrm{g}$ and $\gamma^*(\mathbf{1}_B) = \mathbb{P}_\mathrm{r}$. It does this by iteratively, alternatingly updating

$$u \leftarrow {}^{\mathbb{P}_\mathrm{r}}/_{K^* v}; \quad v \leftarrow {}^{\mathbb{P}_\mathrm{g}}/_{Ku}.$$

The linear operators for premultiplication, $(u\cdot)$ and $(v\cdot)$, are more commonly described as diagonal matrices.

The *Cuturi-Sinkhorn algorithm* [2] uses this to obtain an optimal transport. For $A = B = \mathcal{C}(X)$ (continuous functions), it encodes the metric $\|\ \|$ on X into K as

$$K : (\mathcal{C}(X))^* \to \mathcal{C}(X), \quad K(w)(x) := w(\backslash y \mapsto e^{-\lambda \cdot \|x-y\|}).$$

Then, the fixpoint of the Sinkhorn iteration will be an *entropy-limited approximation* to the optimal transport. In the limit $\lambda \to \infty$, this approximation becomes arbtrarily good.

Data structures. The usual way Sinkhorn-Cuturi is used is on discrete X or with discretised representations of the functions on it. In this case, $\mathcal{C}(X)$ and $\mathcal{C}(X)^*$ are both types of vectors/arrays of numbers (in practice usually floating-point). The multiplication operation is just element-wise multiplication in those arrays. However, natural as this may seem, it is known from numerical applications (in particular, differential equations) that such a choice of spatial-points sampling can be suboptimal. To name one issue, point-wise multiplication generally leads to frequency aliasing.

In Sagemüller and Verdier 2019 [4], we proposed a tree data structure to represent functions and their duals *without* any a-priori resolution choice. It is a simple transformation closely related to the standard Haar wavelet transform.

The subject of this work is the use of that structure as the representation for distributions in Sinkhorn optimal transport. The first attempt was to treat the distributions *as functions* in this wavelet expansion, and use the multiplication directly pointwise on those functions, analogous to pointwise on array on the standard discretised implementations. That however leads to numerical instability when transporting non-overlapping distributions: namely, the marginals oscillate in an uncontrolled manner rather than converging towards the desired ones (as Sinkhorn guarantees they should do, given exact calculations). This appears to arise from floating-point inaccuracies in connection with the point-wise cancellation that is required to represent confined distributions in wavelets (or other bases that are not completely local by construction).

Unlike with finite-dimensional vectors, the structure for duals/distributions in our Haar representation is not exactly the same as for functions: they have both different strictness, and different weighting of the coefficients. It turns out that switching to the dual-space view of Sinkhorn-Cuturi outlined in the previous section does both promise allowing to represent confined distributions better (even point/Dirac-distributions are possible), it is also less susceptible to the floating-point problems. We discuss reasons for this behaviour. A main way in which both representations differ is that in the function-space view, the wavelet coefficients describing a narrow peak grow as the tree branches narrow down (so that numbers of very *different* magnitude are summed up), whereas the dual space always only cares about the full integral across a subdomain, which does not change when zooming in so that *similar*-magnitude numbers are summed up, which is generally more precise in floating-point.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[2] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transportation distances. *arXiv e-prints*, page arXiv:1306.0895, 2013.

[3] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, mar 1951.

[4] J. Sagemüller and O. Verdier. Lazy evaluation in infinite-dimensional function spaces with wavelet basis. In *Proceedings of the 8th ACM SIGPLAN International Workshop on Functional High-Performance and Numerical Computing - FHPNC 2019*. ACM Press, 2019.

[5] R. Sinkhorn. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.