Report of Dagstuhl Seminar 15451

Verification of Evolving Graph Structures

Edited by

Parosh Aziz Abdulla[1], Fabio Gadducci[2], Barbara König[3], and Viktor Vafeiadis[4]

1 **Uppsala University, SE,** parosh@it.uu.se
2 **University of Pisa, IT,** gadducci@di.unipi.it
3 **Universität Duisburg-Essen, DE,** barbara_koenig@uni-due.de
4 **MPI-SWS – Kaiserslautern, DE,** viktor@mpi-sws.org

Abstract

This report documents the programme and the outcome of Dagstuhl Seminar 15451 "Verification of Evolving Graph Structures".

The aim was to bring together researchers from different communities (shape analysis, separation logic, graph transformation, verification of infinite-state systems) who are interested in developing techniques for the analysis of graph manipulations, i.e., methods that are able to handle the challenges that arise in current verification problems.

Apart from scientific talks, the programme also included four tutorial talks and four working groups, which are summarized in this report.

Seminar November 1–6, 2015 – http://www.dagstuhl.de/15451
1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.2 Grammars and Other Rewriting Systems
Keywords and phrases dynamic systems, graph transformation, graphs, heap analysis, separation logic, shape analysis, static analysis, verification
Digital Object Identifier 10.4230/DagRep.5.11.1
Edited in cooperation with Christina Jansen and Eugenio Orlandelli

1 Summary

Parosh Aziz Abdulla
Fabio Gadducci
Barbara König
Viktor Vafeiadis

Despite significant progress in recent years, verification still remains a challenging task for hardware and software systems. A particularly complex verification problem is the analysis of graph-like structures that may modify their topology during runtime. The main reason for the difficulty is that some features give rise to infinite state spaces. Examples include variables ranging over unbounded domains, timing constraints, dynamic process creation, heap manipulation, multi-threading, and dynamically allocated data structures. An additional source of complication is that the underlying graphs may be continuously evolving. There is no a priori bound on the size of the graphs that may arise when modelling the run of a program, and the graph shapes may change during a given execution.

Verification of Evolving Graph Structures, *Dagstuhl Reports*, Vol. 5, Issue 11, pp. 1–28
Editors: Parosh Aziz Abdulla, Fabio Gadducci, Barbara König, and Viktor Vafeiadis

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This challenge has prompted several successful lines of research, developing novel techniques such as shape analysis, separation logic, forest automata, and several graph transformation-based approaches. Although specialized tools have been developed in each application area, a considerable amount of effort is needed to develop uniform frameworks that yield efficient yet general solutions.

This seminar brought together researchers interested in developing precise and scalable techniques for the analysis of graph manipulations, i.e., techniques that are able to handle the challenges that arise in current verification problems. These challenges require novel developments and the combination of techniques from a wide range of different areas including model checking and dynamic and static program analysis. By creating collaboration opportunities we hope to substantially increase the size of the systems that can be tackled and the precision of analysis that can be achieved.

Hence the main goal of this seminar was to enhance common understanding and cross-fertilization, highlighting connections among the approaches via tutorials and working groups, with the explicit purpose to enhance interaction. Discussion topics included:

- the definition of uniform frameworks in which to integrate methods for graph analysis that have been proposed by the different research communities;
- the development of new abstraction techniques for pushing the state-of-the-art of graph algorithms in program verification and model checking applications; and
- the identification of research areas in which the analysis of graph manipulation may play an important role, such as the analysis of security protocols, social networks, adaptive networks, and biological systems.

We invited four representatives of the different communities to give tutorial talks in order to introduce fundamental concepts and techniques. Specifically, the following four tutorial talks took place on the first day of the seminar:

- Tomas Vojnar: Shape Analysis via Symbolic Memory Graphs and Its Application for Conversion of Pointer Programs to Container Programs
- Giorgio Delzanno: Graphs in Infinite-State Model-Checking
- Arend Rensink: Verification Techniques for Graph Rewriting
- Viktor Vafeiadis: Separation Logic

On Tuesday and Thursday we organized the following working groups in order to discuss more specific topics which were of interest to a substantial part of the participants:

- Benchmarks and Application Domains
- Specification Languages for Graphs
- Ownership
- Graph Rewriting for Verification

The organizers would like to thank all the participants and speakers for their inspiring talks and many interesting discussions. Furthermore we would like to acknowledge Christina Jansen and Eugenio Orlandelli who helped to write and prepare this report. A special thanks goes to the Dagstuhl staff who were a great help in organizing this seminar.

2 Table of Contents

3 Overview of Talks

3.1 Verification of Dynamic Register Automata

Mohamed-Faouzi Atig (Uppsala University, SE)

Joint work of Parosh Aziz Abdulla; Mohamed Faouzi Atig; Ahmet Kara; Othmane Rezine
Main reference P. A. Abdulla, M. F. Atig, A. Kara, O. Rezine, "Verification of Dynamic Register Automata," in Proc. of the 34th Int'l Conf. on Foundation of Software Technology and Theoretical Computer Science (FSTTCS'14), LIPIcs, Vol. 29, pp. 653–665, Schloss Dagstuhl, 2014.
URL http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2014.653

We consider the verification problem for Dynamic Register Automata (DRA). DRA extend classical register automata by process creation. In this setting, each process is equipped with a finite set of registers in which the process IDs of other processes can be stored. A process can communicate with processes whose IDs are stored in its registers and can send them the content of its registers. The state reachability problem asks whether a DRA reaches a configuration where at least one process is in an error state. We will first show that this problem is in general undecidable. This result holds even when we restrict the analysis to configurations where the maximal length of the simple paths in their underlying (un)directed communication graphs are bounded by some constant. Then we will introduce the model of degenerative DRA which allows non-deterministic reset of the registers. We will prove that for every given DRA, its corresponding degenerative one has the same set of reachable states. While the state reachability of a degenerative DRA remains undecidable, we will show that the problem becomes decidable with nonprimitive recursive complexity when we restrict the analysis to strongly bounded configurations, i.e. configurations whose underlying undirected graphs have bounded simple paths. Finally, we will consider the class of strongly safe DRA, where all the reachable configurations are assumed to be strongly bounded. We show that for strongly safe DRA, the state reachability problem becomes decidable.

3.2 Verification Linearizability for SLL-based Concurrent Data Structures

Parosh Aziz Abdulla (Uppsala University, SE), Bengt Jonsson (Uppsala University, SE), and Cong-Quy Trinh

We present a framework for automated verification of linearizability for concurrent data structures that implement sets, stacks, and queues. We use a specification formalism for linearization policies which allows the user to specify complex patterns including non-fixed linearization points. We define abstraction techniques that allow to make the size of the data domain and the number of threads finite. In order to reason about dynamically allocated memory, we use a combination of shape graphs and thread-modular reasoning. Based on our method, we have verified linearizability for a number of algorithms., including all implementations of concurrent sets, stacks, and queues based on singly-linked lists that are known to us from the literature.

3.3 A Causal View on Non-Interference

Paolo Baldan (University of Padova, IT)

Joint work of Alessandro Beggiato; Alberto Carraro
Main reference P. Baldan, A. Carraro, "A Causal View on Non-Interference". Fundam. Inform. 140(1):1–38, 2015.
URL http://dx.doi.org/10.3233/FI-2015-1243

The concept of non-interference has been introduced to characterise the absence of undesired information flows in a computing system. Although it is often explained referring to an informal notion of causality – the activity involving the part of the system with higher level of confidentiality should not cause any observable effect at lower levels – it is almost invariably formalised in terms of interleaving semantics. In this talk, focusing on Petri nets, we discuss the possibility of providing a causal characterisations of non-interference based on a true concurrent semantics. The investigation can have a conceptual interest – as it clarifies the relation between causality and non-interference, and a practical value as the verification phase can take advantage of partial order techniques.

3.4 Observable Under-Approximations

Aiswarya Cyriac (Uppsala University, SE)

Joint work of Cyriac Aiswarya; Paul Gastin; K. Narayan Kumar
Main reference A. Cyriac, P. Gastin, K. N. Kumar, "Controllers for the Verification of Communicating Multi-Pushdown Systems," in Proc. of the 25th Int'l Conf. on Concurrency Theory (CONCUR'14), LNCS, Vol. 8704, pp. 297–311, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-662-44584-6_21

An under-approximation is observable/controllable/monitorable/diagnosable if it can be decided in run-time whether the current behaviour has exceeded the under-approximation. Behaviours of interest are graphs which, along a run, are monotonously increasing, by means of adding new vertices and edges. We illustrate the notion of observable under-approximation by an example on message sequence charts. We conclude the short presentation with some open questions. Is bounded tree-width observable as an under-approximation, even in special classes of graphs with an underlying linear order and uniformly bounded degree? Which classical under-approximations are observable?

3.5 Graphs in Infinite-State Model Checking (Tutorial)

Giorgio Delzanno (University of Genova, IT)

Joint work of P. A. Abdulla; M. F. Atig; N. Ben Henda; N. Bertrand; B. König; A. Rezine; O. Rezine; A. Sangnier; J. Stückrath; R. Traverso; G. Zavattaro
Main reference G. Delzanno, A. Sangnier, G. Zavattaro, "Parameterized Verification of Ad Hoc Networks," in Proc. of the 21th Int'l Conf. on Concurrency Theory (CONCUR'10), LNCS, Vol. 6269, pp. 313–327, Springer, 2010.
URL http://dx.doi.org/10.1007/978-3-642-15375-4_22

We present a survey on the application of graph theory in the field of infinite-state and parameterized verification. We consider different types of formalisms like transition systems, Petri nets, automata, and rewriting and discuss verification methods based on abstractions and symbolic state exploration.

References

1 P. A. Abdulla, M. F. Atig, and O. Rezine. Verification of directed acyclic ad hoc networks. In *FMOODS/FORTE*, pages 193–208, 2013.
2 P. A. Abdulla, G. Delzanno, and A. Rezine. Approximated parameterized verification of infinite-state processes with global conditions. *Formal Methods in System Design*, 34(2):126–156, 2009.
3 P. A. Abdulla, G. Delzanno, and A. Rezine. Automatic verification of directory-based consistency protocols with graph constraints. *Int. J. Found. Comput. Sci.*, 22(4), 2011.
4 P. A. Abdulla, G. Delzanno, O. Rezine, A. Sangnier, and R. Traverso. On the verification of timed ad hoc networks. In *FORMATS'11*, volume 6604 of *LNCS*, pages 256–270. Springer, 2011.
5 P. A. Abdulla, N. Ben Henda, G. Delzanno, and A. Rezine. Handling parameterized systems with non-atomic global conditions. In *VMCAI'08*, volume 4905 of *LNCS*, pages 22–36. Springer, 2008.
6 N. Bertrand, G. Delzanno, B. König, A. Sangnier, and J. Stückrath. On the decidability status of reachability and coverability in graph transformation systems. In *RTA*, pages 101–116, 2012.
7 N. Bertrand, P. Fournier, and A. Sangnier. Playing with probabilities in reconfigurable broadcast networks. In *FoSSaCS*, pages 134–148, 2014.
8 G. Delzanno, C. Di Giusto, M. Gabbrielli, C. Laneve, and G. Zavattaro. The *kappa*-lattice: Decidability boundaries for qualitative analysis in biological languages. In *CMSB*, pages 158–172, 2009.
9 G. Delzanno, A. Sangnier, and R. Traverso. Parameterized verification of broadcast networks of register automata. In *RP*, pages 109–121, 2013.
10 G. Delzanno, A. Sangnier, R. Traverso, and G. Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *FSTTCS'12*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012.
11 G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR'10*, volume 6269 of *LNCS*, pages 313–327. Springer, 2010.
12 G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FOSSACS'11*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.
13 G. Delzanno, A. Sangnier, and G. Zavattaro. Verification of ad hoc networks with node and communication failures. In *FORTE/FMOODS'12*, volume 7273 of *LNCS*, pages 235–250. Springer, 2012.

14 G. Delzanno and R. Traverso. Decidability and complexity results for verification of asynchronous broadcast networks. In *LATA*, pages 238–249, 2013.

15 G. Ding. Subgraphs and well quasi ordering. *J. of Graph Theory*, 16(5):489–502, 1992.

16 E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *LICS'98*, pages 70–80. IEEE Computer Society, 1998.

17 E. Allen Emerson and V. Kahlon. Parameterized model checking of ring-based message passing systems. In *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20-24, 2004, Proceedings*, pages 325–339, 2004.

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

19 S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. In *CAV'08*, volume 5123 of *LNCS*, pages 214–226. Springer, 2008.

20 K. S. Namjoshi and R. J. Trefler. Uncovering symmetries in irregular process networks. In *VMCAI*, pages 496–514, 2013.

21 M. Saksena, O. Wibling, and B. Jonsson. Graph grammar modeling and verification of ad hoc routing protocols. In *TACAS*, pages 18–32, 2008.

3.6 PSYNC: A Partially Synchronous Language for Fault-Tolerant Distributed Algorithms

Cezara Dragoi (IST Austria – Klosterneuburg, AT), Damien Zufferey, and Tom Henzinger

Fault-tolerant distributed algorithms play an important role in many critical/high-availability applications. These algorithms are notoriously difficult to implement correctly, due to asynchronous communication and the occurrence of faults, such as the network dropping messages or computers crashing. We introduce PSYNC, a domain specific language based on the Heard-Of model, which views asynchronous faulty systems as synchronous ones with an adversarial environment that simulates asynchrony and faults by dropping messages. We define a runtime system for PSYNC that efficiently executes on asynchronous networks. We formalize the relation between the runtime system and PSYNC in terms of observational refinement. This high-level synchronous abstraction introduced by PSYNC simplifies the design and implementation of fault-tolerant distributed algorithms and enables automated formal verification. We have implemented an embedding of PSYNC in the SCALA programming language with a runtime system for partially synchronous networks. We show the applicability of PSYNC by implementing several important fault-tolerant distributed algorithms and we compare the implementation of consensus algorithms in PSYNC against implementations in other languages in terms of code size, runtime efficiency, and verification.

3.7 Symbolic Abstract Data Types

Constantin Enea (University of Paris VII, FR)

Formal specification is a vital ingredient to scalable verification of software systems. In the case of efficient implementations of concurrent objects like atomic registers, queues, and locks, symbolic formal representations of their abstract data types (ADTs) enable efficient modular reasoning, decoupling clients from implementations. Writing adequate formal specifications, however, is a complex task requiring rare expertise. In practice, programmers write reference implementations as informal specifications.

In this work we demonstrate that effective symbolic ADT representations can be automatically generated from the executions of reference implementations. Our approach exploits two key features of naturally-occurring ADTs: violations can be decomposed into a small set of representative patterns, and these patterns manifest in executions with few operations. By identifying certain algebraic properties of naturally-occurring ADTs, and exhaustively sampling executions up to a small number of operations, we generate concise symbolic ADT representations which are complete in practice, enabling the application of efficient symbolic verification algorithms without the burden of manual specification. Furthermore, the concise ADT violation patterns we generate are human-readable, and can serve as useful, formal documentation.

3.8 From Decision Procedures to Full Model-Checking: the MCMT Experience

Silvio Ghilardi (University of Milan, IT)

In this talk, we briefly report both experience and case studies in the development of our logic-based models checker, called MCMT, 'Model Checker Modulo Theories' (see http://users.mat.unimi.it/users/ghilardi/mcmt). During past years, many people (besides the author of the present contribution) contributed to implementation, theoretical advances or experiments; among them, let us mention F. Alberti, R. Bruttomesso, A. Carioni, E. Nicolini, A. Orsini, E. Pagani, S. Ranise, N. Sharygina, D. Zucchelli.

The basic idea in the development of MCMT is the revisitation, in a declarative perspective, of classic results concerning well structured transition systems (WSTS) [2]. The WSTS framework is a formal framework covering a large class of systems and their evolution; in concrete applications, WSTS arise from finitely presented models of rather simple logical theories. These theories are array theories obtained from the combination of a theory for process 'topology' and of theories for (local and shared) data. The notion of an *array-based system* formalizes this intuition [18, 19]. In array-based systems backward search can be implemented in a purely symbolic way and a model-checker exploiting this idea can rely on state-of-the-art SMT-solvers to discharge the proof-obligations needed for fixpoint and safety tests (no ad hoc data structures need to be invented). Monotonic abstraction techniques [3, 4, 5, 1] can be implemented declaratively too by using syntactic constructions like quantifier instantiations and quantifiers relativizations [11].

The framework of array-based system is quite flexible and can be adapted to cope with various kinds of distributed [19], timed [17, 16] and fault-tolerant systems [9, 10]. Both abstraction [6, 8] and acceleration [14] techniques can be integrated with it, making the approach quite suitable for dealing also with array-manipulating sequential programs [7, 12]. For future, we expect even more progress taking advantage from recent advances in the decision procedures concerning quantified fragments of array theories [15, 20, 13, 14].

References

1 P. A. Abdulla. Forcing monotonicity in parameterized verification: From multisets to words. In *Proceedings of SOFSEM '10*, pages 1–15. Springer-Verlag, 2010.
2 P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proc. of LICS*, pages 313–321, 1996.
3 P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezine. Regular model checking without transducers. In *TACAS*, volume 4424 of *LNCS*, pages 721–736, 2007.
4 P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *CAV*, pages 145–157, 2007.
5 P. A. Abdulla, N. B. Henda, G. Delzanno, and A. Rezine. Handling parameterized systems with non-atomic global conditions. In *Proc. of VMCAI*, volume 4905 of *LNCS*, pages 22–36, 2008.
6 F. Alberti, R. Bruttomesso, S. Ghilardi, S. Ranise, and N. Sharygina. Lazy Abstraction with Interpolants for Arrays. In *LPAR-18*, pages 46–61, 2012.
7 F. Alberti, R. Bruttomesso, S. Ghilardi, S. Ranise, and N. Sharygina. SAFARI: SMT-Based Abstraction for Arrays with Interpolants. In *CAV*, pages 679–685, 2012.
8 F. Alberti, R. Bruttomesso, S. Ghilardi, S. Ranise, and N. Sharygina. An extension of lazy abstraction with interpolation for programs with arrays. *Formal Methods in System Design*, pages 63–109, 2014.
9 F. Alberti, S. Ghilardi, E. Pagani, S. Ranise, and G. P. Rossi. Automated support for the design and validation of fault tolerant parameterized systems – a case study. In *Proc. of AVOCS*, 2010.
10 F. Alberti, S. Ghilardi, E. Pagani, S. Ranise, and G. P. Rossi. Brief announcement: Automated support for the design and validation of fault tolerant parameterized systems – a case study. In *DISC*, pages 392–394, 2010.
11 F. Alberti, S. Ghilardi, E. Pagani, S. Ranise, and G.P. Rossi. Universal guards, relativization of quantifiers, and failure models in model checking modulo theories. *JSAT*, 8(1/2):29–61, 2012.
12 F. Alberti, S. Ghilardi, and N. Sharygina. Booster : an acceleration-based verification framework for array programs. In *ATVA*, pages 18–23, 2014.
13 F. Alberti, S. Ghilardi, and N. Sharygina. Decision procedures for flat array properties. In *TACAS*, pages 15–30, 2014.
14 F. Alberti, S. Ghilardi, and N. Sharygina. A new acceleration-based combination framework for array properties. In *FroCoS*, 2015.
15 A.R. Bradley, Z. Manna, and H.B. Sipma. What's decidable about arrays? In *VMCAI*, pages 427–442, 2006.
16 R. Bruttomesso, A. Carioni, S. Ghilardi, and S. Ranise. Automated Analysis of Parametric Timing Based Mutual Exclusion Protocols. In *NASA Formal Methods Symposium*, 2012.
17 A. Carioni, S. Ghilardi, and S. Ranise. MCMT in the land of parameterized timed automata. In *In proc. of VERIFY*, 2010.
18 S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Towards SMT Model-Checking of Array-based Systems. In *Proc. of IJCAR*, LNCS, 2008.

19 S. Ghilardi and S. Ranise. Backward Reachability of Array-based Systems by SMT solving: Termination and Invariant Synthesis. *LMCS*, 6(4), 2010.
20 P. Habermehl, R. Iosif, and T. Vojnar. A logic of singly indexed arrays. In *LPAR*, pages 558–573, 2008.

3.9 Invariant Checking for Graph Transformation: Applications & Open Challenges

Holger Giese (Hasso-Plattner-Institut – Potsdam, DE) and Leen Lambers (Hasso-Plattner-Institut – Potsdam, DE)

Graph transformation can be used as a formal foundation for modeling different kinds of evolving graph structures. In particular, we present two application domains, cyber-physical systems (CPS) and model-driven engineering (MDE), where graph transformation has been used successfully to formally model different scenarios. Furthermore, we employ inductive invariant checking for graph transformation [1] as a verification technique for the different scenarios of the two domains. In the CPS domain the invariance of important safety properties can be shown. In the MDE domain, behavior preservation of model transformations can be reduced to invariant checking [2]. We give an overview of how invariant checking has been applied in these two domains on different scenarios. We present the strengths and weaknesses of this verification technique and conclude with some open challenges.

References

1 Basil Becker, Dirk Beyer, Holger Giese, Florian Klein, and Daniela Schilling. Symbolic invariant verification for systems with dynamic structural adaptation. In *Proc. of the* 28^th *International Conference on Software Engineering (ICSE), Shanghai, China*. ACM Press, 2006.
2 Holger Giese and Leen Lambers. Towards automatic verification of behavior preservation for model transformation via invariant checking. In *Proceedings of International Conference on Graph Transformation (ICGT'12)*, volume 7562 of *LNCS*, pages 249–263. Springer, 2012.

3.10 Approaching the Coverability Problem Continuously

Christoph Haase (ENS – Cachan, FR)

Joint work of Michael Blondin; Alain Finkel; Serge Haddad
Main reference M. Blondin, A. Finkel, C. Haase, S. Haddad, "Approaching the Coverability Problem Continuously," arXiv:1510.05724v2 [cs.LO], 2016.
URL http://arxiv.org/abs/1510.05724v2

The coverability problem for Petri nets plays a central role in the verification of concurrent shared-memory programs. However, its high EXPSPACE-complete complexity poses a challenge when encountered in real-world instances. In this talk, I will present a new approach to this problem which is primarily based on applying forward coverability in continuous Petri nets as a pruning criterion inside a backward-coverability framework. A

cornerstone of the approach is the efficient encoding of a recently developed polynomial-time algorithm for reachability in continuous Petri nets into SMT. The effectiveness of the approach is demonstrated on standard benchmarks from the literature, which shows that it decides significantly more instances than any existing tool and is in addition often much faster, in particular on large instances.

3.11 Dijkstra-style Verification of Graph Programs

Annegret Habel (Universität Oldenburg, DE)

Joint work of Annegret Habel; Karl-Heinz Pennemann; Hendrik Radke; Nils Erik Flick

We investigate Dijkstra-style verification of graph programs relative to several notions of graph conditions (nested, recursively nested, M, HR*) and show: (1) For all notions of graph conditions, there is a transformation Wp such that for every graph program P and every postcondition post, Wp(P,post) is a weakest precondition of P relative to post. (2) For nested and recursively nested graph conditions, there is a semi-decider for the implication problem. In particular, the theorem prover for nested graph conditions is much better than the theorem provers for first-order graph formulas getting the transformed graph condition as input.

References

1 Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions.Mathematical Structures in Computer Science, 19:245–296, 2009.
2 Annegret Habel and Hendrik Radke. Expressiveness of graph conditions with variables. Electronic Communications of the EASST, 30, 2010.
3 Nils Erik Flick. On correctness of graph programs relative to recursively nested conditions. In Graph Computation Models (GCM 2015), volume 1403, pages 97–112. CEUR-WS.org, 2015.

3.12 Modelling Evolving Graph Structures by Differential Equations

Reiko Heckel (University of Leicester, GB)

Joint work of Reiko Heckel; Mudhafar Hussein
Main reference Mudhafar Hussein, Reiko Heckel, Vincent Danos, Pawel Sobocinski, "Modelling Adaptive Networks: The Case of the Petrified Voters," Electronic Communications of the EASST, Vol. 67, pp. 1–12, 2014.
URL http://dx.doi.org/10.14279/tuj.eceasst.67.950

From a stochastic graph transformation system modelling an evolving network it is possible to derive a system of differential equations describing the average evolution of the network. The key concept is an approximation of complex patterns such as they appear in rules' left- and right-hand sides, including negative application conditions and attribute constraints, by combinations of simpler ones. Such approximations is correct in the sense that, for large random graphs where occurrences of patterns and attribute values are independent, they

converge towards the actual number of occurrences of the complex patterns. We illustrate this approach by an example and discuss how these assumptions can be validated using simulations.

References

1 Mudhafar Hussein, Reiko Heckel, Vincent Danos, Pawel Sobocinski. *Modelling Adaptive Networks: The Case of the Petrified Voters*. Proceedings of the 13th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2014)

3.13 A Graph-Based Semantics Workbench for Concurrent Asynchronous Programs

Alexander Heußner (Universität Bamberg, DE) and Chris Poskitt (ETH Zürich, CH)

Joint work of Claudio Corrodi; Alexander Heußner; Christopher M. Poskitt

This talk presents a pathway to a "semantics workbench", with which multiple alternative and possibly contradicting semantics of state-of-the-art concurrency abstractions can be formalised, analysed, and compared. We also raise some (new) fundamental research questions in the areas of graph transformation systems and verification.

3.14 Inverse Monoid of Higher Dimensional Strings

David Janin (University of Bordeaux, FR)

Halfway between graph transformation theory and inverse semigroup theory, we define higher dimensional strings as bi-deterministic graphs with distinguished sets of input roots and output roots. We show that these generalized strings can be equipped with an associative product so that the resulting algebraic structure is an inverse semigroup. Its natural order is shown to capture existence of root preserving graph morphism. A simple set of generators is characterized. As a subsemigroup example, we show how all finite grids are finitely generated. Finally, simple additional restrictions on products lead to the definition of subclasses with decidable Monadic Second Order (MSO) language theory.

References

1 D. Janin. Inverse monoids of higher-dimensional strings. In *Int. Col. on Theor. Aspects of Comp. (ICTAC)*, volume 9399 of *LNCS*, 2015.

3.15 Verifying Pointer Programs using Graph Grammars

Christina Jansen (RWTH Aachen, DE), Joost-Pieter Katoen (RWTH Aachen, DE), Christoph Matheja, and Thomas Noll (RWTH Aachen, DE)

Joint work of Jonathan Heinen; Christina Jansen; Joost-Pieter Katoen; Christoph Matheja; Thomas Noll
Main reference J. Heinen, C. Jansen, J.-P. Katoen, T. Noll, "Juggrnaut: Using graph grammars for abstracting unbounded heap structures," Formal Methods in System Design, 47(2):159–203, 2015.
URL http://dx.doi.org/10.1007/s10703-015-0236-1

This talk presents an abstraction framework for heap data structures. It employs graph grammars, more precisely context-free hyperedge replacement grammars. Our approach aims at extending finite-state verification techniques to handle pointer-manipulating programs operating on complex dynamic data structures that are potentially unbounded in their size. We will see which subset of hyperedge replacement grammars provides sound abstractions and briefly elaborate on its relation to Separation Logic. In addition, a small tool comparison comprising our prototypical tool Juggrnaut as well as each a tool from the area of shape analysis, Separation Logic and general graph transformation is presented.

3.16 Bounded Time-Stamping for Message-passing Systems: Beyond Channel Bounds

Narayan Kumar Krishnan (Chennai Mathematical Institute, IN)

Joint work of Cyriac Aiswarya; Paul Gastin; K. Narayan Kumar

Consider distributed systems consisting of a number of processes communicating with each other by sending messages via FIFO channels. It is crucial for such systems that every process can maintain deterministically the latest information about other processes. To do so, any process p, upon receiving a message from a process q, should determine for every process r, whether the latest event on r that p knows of is more recent than the latest event on r that q knows of. Solving this problem, while storing and exchanging only a bounded amount of information, is very challenging and not always possible. This is known as the gossip problem. A solution to this is the key to solving numerous important problems on distributed systems. We provide a solution that simplifies an existing algorithm for this problem and also extends it to a richer class, going beyond a priori channel bounds.

3.17 Spatio-Temporal Model Checking

Michele Loreti (University of Firenze, IT)

Joint work of Vincenzo Ciancia; Stephen Gilmore; Gianluca Grilletti; Diego Latella; Michele Loreti; Mieke Massink

The interplay between process behaviour and spatial aspects of computation has become more and more relevant in Computer Science, especially in the field of collective adaptive systems, but also, more generally, when dealing with systems distributed in physical space.

Traditional verification techniques are well suited to analyse the temporal evolution of programs; properties of space are typically not explicitly taken into account. We propose a methodology to verify properties depending upon physical space. We define an appropriate logic, stemming from the tradition of topological interpretations of modal logics, dating back to earlier logicians such as Tarski, where modalities describe neighbourhood and surrounding. We lift the topological definitions to a more general setting, also encompassing discrete, graph-based structures. A spatial extension of the global model checking algorithm of the temporal logic CTL is also presented. More precisely, we add to CTL the new spatial operators. The interplay of space and time permits one to define complex spatio-temporal properties.

References

1 Specifying and Verifying Properties of Space V. Ciancia, D. Latella, M. Loreti, M. Massink IFIP TCS 2014, Lecture Notes in Computer Science, 8705, pp. 222–235, 2014.
2 Spatio-Temporal Model-Checking Of Vehicular Movement In Public Transport Systems V. Ciancia, S. Gilmore, G. Grilletti, D. Latella, M. Loreti and M. Massink. Submitted for journal publication, 2014.
3 A spatio-temporal model-checker V. Ciancia, G. Grilletti, D. Latella, M. Loreti and M. Massink. VERY* 2015, Lecture Notes in Computer Science, to appear.
4 Qualitative and Quantitative Monitoring of Spatio-Temporal Properties L. Nenzi, L. Bortolussi, V. Ciancia, M. Loreti and M. Massink. RV 2015, Lecture Notes in Computer Science, 9333, pp. 21–37, 2015.

3.18 Pointer Race Freedom

Roland Meyer (TU Kaiserslautern, DE)

Joint work of Frédéric Haziza; Lukas Holik; Roland Meyer; Sebastian Wolff
URL http://arxiv.org/abs/1511.00184

We propose a novel notion of pointer race for concurrent programs manipulating a shared heap. A pointer race is an access to a memory address which was freed, and it is out of the accessor's control whether or not the cell has been re-allocated. We establish two results. (1) Under the assumption of pointer race freedom, it is sound to verify a program running under explicit memory management as if it was running with garbage collection. (2) Even the requirement of pointer race freedom itself can be verified under the garbage-collected semantics. We then prove analogues of the theorems for a stronger notion of pointer race needed to cope with performance-critical code purposely using racy comparisons and even racy dereferences of pointers. As a practical contribution, we apply our results to optimize a thread-modular analysis under explicit memory management. Our experiments confirm a speed-up of up to two orders of magnitude.

3.19 Modular Analysis of Concurrent Pointer Programs Using Graph Grammars

Thomas Noll (RWTH Aachen, DE), Christina Jansen (RWTH Aachen, DE), and Jens Katelaan

Programs with shared-memory concurrency are inherently difficult to get right: they are prone to all the memory-related errors that are familiar from the single-threaded setting, such as null pointer dereferences and unintended aliasing. In addition, the possible interference between parallel execution threads gives rise to new classes of errors, such as data races. As thread interleaving is nondeterministic in nature and heap-manipulating programs generally have an unbounded state space due to dynamic memory allocation, the application of formal methods is challenging in this setting.

In this talk we develop a static analysis for proving properties such as shape invariants, absence of null pointer dereferences, as well as data-race freedom of programs with fork-join parallelism. To this end, we develop a formal semantics based on hypergraphs and access permissions, and derive an abstract interpretation that uses hyperedge replacement grammars to safely approximate the program's semantics. The result is a fully automatic, thread-modular analysis for proving the above properties in the presence of recursive data structures and dynamic (possibly recursive) thread creation.

3.20 On Graphical Logics for Reasoning about Graph Properties

Fernando Orejas (UPC – Barcelona, ES)

Joint work of Leen Lambers; Marisa Navarro; Fernando Orejas; Elvira Pino

By graphical logics, we mean logics whose formulas are not text, but they consist of graphs and graph morphisms, that are used to express graph properties. In this presentation, I will review previous work on this area and, moreover, I will present the main problems found when extending the logic to allow for the specification of the existence of paths in given graphs. In particular, I will present a proof calculus for this extension that is shown to be sound and it is conjectured to be complete.

References

1 F. Orejas, H. Ehrig and U. Prange: *Reasoning with Graph Constraints*, Form. Asp. Computing 2010.
2 A. Rensink: *Representing First-Order Logic Using Graphs*. ICGT 2004: 319–335
3 A. Habel, K. H. Pennemann: *Correctness of high-level transformation systems relative to nested conditions*. Math. Structures in Comp. Science 2009.
4 K. H. Pennemann: *Development of Correct Graph Transformation Systems*, Ph. D. Thesis, 2009.
5 L. Lambers, F. Orejas: *Tableau-Based Reasoning for Graph Properties*. ICGT 2014.
6 M. Navarro, F. Orejas, E. Pino: *Satisfiability of Constraint Specifications on XML Documents*. Festschrift José Meseguer (2015).

3.21 Interactive Verification of Parameterized Systems

Oded Padon (Tel Aviv University, IL)

Joint work of Ken McMillan; Oded Padon; Mooly Sagiv

Verification of infinite-state and parameterized systems is a long standing research goal. Automated verification tools for such systems often solve an intractable to undecidable problem, so some failures of automation are unavoidable. This work is an an attempt to combine user interaction with automated verification heuristics. We use the decidable EPR fragment of FOL to obtain predictability of the automated analysis, and engage the user to help the system generalize from counter-examples to induction. The user interaction is obtained via graphical visualization, and interactive heuristics. By combining powerful invariant inference heuristics with user interaction, we hope to make verification of infinite-state and parameterized systems more practical.

3.22 Hoare-Style Verification for GP 2

Detlef Plump (University of York, GB) and Christopher M. Poskitt (ETH Zürich, CH)

Joint work of Christopher M. Poskitt; Detlef Plump
Main reference C. M. Poskitt, D. Plump, "Hoare-Style Verification of Graph Programs", Fundamenta Informaticae, 118(1–2): 35–175, 2012.
URL http://dx.doi.org/10.3233/FI-2012-708

GP 2 is an experimental non-deterministic programming language for solving problems on graphs and graph-like structures. The language is based on graph transformation rules, allowing visual programming at a high level of abstraction. We introduce GP 2 and present a Hoare-style proof system for assertional reasoning about programs. The pre- and postconditions of our calculus are nested graph conditions with extensions for properties of attributes and monadic second-order graph structure. This allows us to reason about global properties of graphs, such as 2-colourability, existence of paths, or connectedness. Our proof system is sound with respect to the operational semantics of GP 2.

References

1 C. M. Poskitt, D. Plump, Verifying monadic second-order properties of graph programs, in: Proc. International Conference on Graph Transformation (ICGT 2014), Vol. 8571 of LNCS, Springer, 2014, pp. 33–48.
2 C. M. Poskitt, D. Plump, Hoare-style verification of graph programs, Fundamenta Informaticae 118 (1-2) (2012) 135–175.

3.23 Verification Techniques for Graph Rewriting (Tutorial)

Arend Rensink (University of Twente, NL)

This tutorial paints a high-level picture of the concepts involved in verification of graph transformation systems. We distinguish three fundamentally different application scenarios for graph rewriting: (1) as grammars (in which case we are interested in the language, or set, of terminal graphs for a fixed start graph); (2) as production systems (in which case we are interested in the relation between start and terminal graphs); or (3) as behavioural specifications (in which case we are interested in the transition system as a whole). We then list some types of questions one might want to answer through verification: confluence and termination, reachability, temporal properties, or contractual properties. Finally, we list some techniques that can help in providing answers: model checking, unfolding, assertional reasoning, and abstraction.

3.24 Refining Orderings for Parameterized Verification

Ahmed Rezine (Linköping University, SE)

Joint work of Ahmed Rezine; Zeinab Ganjei; Yu-Fang Chen; Parosh Abdulla; Giorgio Delzanno; Petru Elese; Zebo Peng
Main reference Z. Ganjei, A. Rezine, P. Eles, Z. Peng, "Abstracting and counting synchronizing processes," in Proc. of the 16th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'15), LNCS, Vol. 8931, pp. 227–244, Springer, 2015.
URL http://dx.doi.org/10.1007/978-3-662-46081-8_13
Main reference Z. Ganjei, A. Rezine, P. Eles, Z. Peng, "Lazy Constrained Monotonic Abstraction," in Proc. of the 17th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'16), LNCS, Vol. 9583, pp. 147–165, Springer, 2015.
URL http://dx.doi.org/10.1007/978-3-662-49122-5_7

Multi-threaded programs may synchronise in subtle ways. For instance, they can use integer variables to count the number of threads satisfying some property in order to implement dynamic barriers or to organise their interleaved execution. We address the problem of automatically establishing deadlock freedom and safety in general for multi-threaded programs generating an arbitrary number of concurrent processes. For this purpose, we explain how we leverage on simple techniques to derive "counting invariants", i.e., invariants that relate the number or processes in a given location to the values of the program variables. We use these invariants and leverage on predicate abstraction techniques in order to generate non-monotonic counter machine reachability problems that faithfully capture the correctness of the safety property.

We describe how we check reachability for non-monotonic counter machines. The idea is to localise the refinement of well quasi orderings in order to allow for a decidable reachability analysis on possibly infinite abstractions that are well structured wrt. these orderings. The orderings can be refined based on obtained false positives in a CEGAR like fashion. This allows for the verification of systems that are not monotonic and are hence inherently beyond the reach of classical well-structured-systems-based analysis techniques. Unlike classical lazy predicate abstraction, we show the feasibility of the approach even for systems with infinite control. Our heuristics are applicable both in backward and in forward as shown by our experiments.

References

1 Z. Ganjei, A. Rezine, P. Eles, and Z. Peng. Lazy Constrained Monotonic Abstraction. VMCAI 2016.
2 Z. Ganjei, A. Rezine, P. Eles, and Z. Peng. Abstracting and counting synchronizing processes. VMCAI 2015.
3 Alastair F. Donaldson, Alexander Kaiser, Daniel Kroening, Thomas Wahl. Symmetry-Aware Predicate Abstraction for Shared-Variable Concurrent Programs. CAV 2011.
4 Parosh Aziz Abdulla, Yu-Fang Chen, Giorgio Delzanno, Frédéric Haziza, Chih-Duo Hong, Ahmed Rezine. Constrained Monotonic Abstraction: A CEGAR for Parameterized Verification. CONCUR 2010.
5 Parosh Aziz Abdulla, Giorgio Delzanno, Ahmed Rezine. Parameterized Verification of Infinite-State Processes with Global Conditions. CAV 2007.

3.25 Shape Analysis for Unstructured Sharing

Xavier Rival (ENS – Paris, FR)

Joint work of Huisong Li; Bor-Yuh Evan Chang; Xavier Rival

Shape analysis aims to infer precise structural properties of imperative memory states and has been applied heavily to verify safety properties on imperative code over pointer-based data structures. It is often applied to dynamic structures such as lists and trees, that can be summarised using inductive predicates. Unfortunately, data structures with unstructured sharing, such as graphs, are challenging to describe and reason about in such frameworks. In particular, when the sharing is unstructured, it cannot be described inductively and in a purely local manner. In this talk, we will describe a global abstraction of sharing based on set-valued variables that when integrated with inductive definitions enables the specification and shape analysis of structures with unstructured sharing.

3.26 Local Strategies in Selective Broadcast Networks

Arnaud Sangnier (University of Paris VII, FR)

Joint work of Nathalie Bertrand; Paulin Fournier
Main reference N. Bertrand, P. Fournier, A. Sangnier, "Distributed Local Strategies in Broadcast Networks," in Proc. of the 26th Int'l Conf. on Concurrency Theory (CONCUR'15), LIPIcs, Vol. 42, pp. 44-57, Schloss Dagstuhl, 2015.
URL http://dx.doi.org/10.4230/LIPIcs.CONCUR.2015.44

We study the problems of reaching a specific control state, or converging to a set of target states, in networks with a parameterized number of identical processes communicating via broadcast. To reflect the distributed aspect of such networks, we restrict our attention to executions in which all the processes must follow the same local strategy that, given their past performed actions and received messages, provides the next action to be performed.

3.27 Compositional Reasoning and Symmetry For Dynamic Protocol Analysis

Richard Trefler (University of Waterloo, CA)

Joint work of Zarrin Langari; Kedar Namjoshi; Richard Trefler

We consider the problem of analyzing programs of several processes that operate over an underlying network. Model checking and other program analysis engines applied to such programs may suffer from the state explosion problem, in which the number of global states to be analyzed is exponential in the number of component processes that compose the system. Compositional reasoning techniques decompose the problem of reasoning about the global system states to a local problem that reasons about the component processes one by one. Symmetry reduction techniques allow many similar processes to be considered all at once by choosing a single representative process as an instance of any one of the symmetric individual processes. We show how to tailor the notion of local process symmetry to apply in the context of compositional analysis. This allows local symmetry reduction techniques to be applied in cases where notions of global symmetry are not applicable. Utilizing abstractions on the local processes we can then apply local symmetry and compositional analysis techniques to locally symmetric protocols; parametrized families of locally symmetric protocols; and dynamic protocols, where the number of processes and their underlying connection network is not fixed during program execution.

References

1 Kedar S. Namjoshi, Richard J. Trefler: Loop Freedom in AODVv2. FORTE 2015: 98–112
2 Kedar S. Namjoshi, Richard J. Trefler: Analysis of Dynamic Process Networks. TACAS 2015: 164–178
3 Kedar S. Namjoshi, Richard J. Trefler: Uncovering Symmetries in Irregular Process Networks. VMCAI 2013: 496–514
4 Kedar S. Namjoshi, Richard J. Trefler: Local Symmetry and Compositional Verification. VMCAI 2012: 348–362
5 Zarrin Langari, Richard J. Trefler: Symmetry for the Analysis of Dynamic Systems. NASA Formal Methods 2011: 252–266
6 Zarrin Langari, Richard J. Trefler: Formal Modeling of Communication Protocols by Graph Transformation. FM 2006: 348–363

3.28 Separation Logic (Tutorial)

Viktor Vafeiadis

Separation logic is an extension of Hoare logic introduced by Reynolds, O'Hearn and others [1], that is suitable for reasoning about programs manipulating heap-allocated data structures. The tutorial gave an overview of separation logic. It covered both sequential and concurrent separation logic [2, 11], and presented an opinionated view of the pros and cons of separation logic.

References

1 Reynolds, J., Separation logic: A logic for shared mutable data structures. In *LICS 2002* (2002), pp. 55–74
2 Brookes, S., *A semantics for concurrent separation logic*, Theor. Comput. Sci. **375** (2007), pp. 227–270.
3 O'Hearn, P. W., *Resources, concurrency and local reasoning*, Theor. Comput. Sci. **375** (2007), pp. 271–307.

3.29 Shape Analysis via Symbolic Memory Graphs and Its Application for Conversion of Pointer Programs to Container Programs (Tutorial)

Tomas Vojnar (Brno University of Technology, CZ)

Joint work of Kamil Dudka; Lukas Holik; Petr Peringer; Marek Trtik; Tomas Vojnar
Main reference K. Dudka, P. Peringer, T. Vojnar, "Byte-Precise Verification of Low-Level List Manipulation," in Proc. of the 20th Int'l Symposium on Static Analysis (SAS'13), LNCS, Vol. 7935, pp. 215–237, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-38856-9_13
Main reference K. Dudka, L. Holik, P. Peringer, M. Trtik, T. Vojnar, "From Low-Level Pointers to High-Level Containers," in Proc. of 17th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'16), LNCS, Vol. 9583, pp. 431–452, Springer, 2016.
URL http://dx.doi.org/10.1007/978-3-662-49122-5_21

In the talk, we briefly overview the main principles of shape analysis based on symbolic memory graphs (SMGs) as implemented in the Predator analyser for C programs with low-level pointer operations (such as pointer arithmetic, address alignment, or block operations). Subsequently, we present a novel application of shape analysis for converting pointer programs to programs using high-level (list) containers.

3.30 Automating Separation Logic Using SMT

Thomas Wies (New York University, US)

Joint work of Ruzica Piskac; Thomas Wies; Damien Zufferey

Separation logic (SL) has gained widespread popularity as a formal foundation of tools that analyze and verify heap-manipulating programs. Its great asset lies in its assertion language, which can succinctly express how data structures are laid out in memory, and its discipline of local reasoning, which mimics human intuition about how to prove heap programs correct.

While the succinctness of separation logic makes it attractive for developers of program analysis tools, it also poses a challenge to automation: separation logic is a nonclassical logic that requires specialized theorem provers for discharging the generated proof obligations. SL-based tools therefore implement their own tailor-made theorem provers for this task. However, these theorem provers are not robust under extensions, e.g., involving reasoning about the data stored in heap structures.

I will present an approach that enables complete combinations of decidable separation logic fragments with other theories in an elegant way. The approach works by reducing

SL assertions to first-order logic. The target of this reduction is a decidable fragment of first-order logic that fits well into the SMT framework. That is, reasoning in separation logic is handled entirely by an SMT solver. We have implemented our approach in the GRASShopper tool and used it successfully to verify interesting data structures.

3.31 Shape and Content

Florian Zuleger (TU Wien, AT)

Joint work of Diego Calvanese; Tomer Kotek; Mantas Simkus; Helmut Veith; Florian Zuleger

Pointers in programs serve two different purposes: (1) Storage of information; pointers are used to build data structures. (2) Semantic information; pointers relate different data items to each other. While the program analysis community has spent considerable effort on analyzing the shape of pointer structures, much less effort has been spent on the analysis of data structure content and the relationship between data items. In this talk I will argue that two-variable logic with counting (C2) is an interesting choice for content anlaysis as it can describe UML-like properties, model pointers and express weakest preconditions of pointer programs [1]. I will discuss extensions of C2 that can express data structures such as lists and trees [2]. Further I will present a combination of C2 with MSO over graphs with bounded tree-width; the resulting logic allows to describe complex data structures and is still decidable [1].

References

1 Tomer Kotek, Helmut Veith, Florian Zuleger: Monadic second order finite satisfiability and unbounded tree-width. arXiv preprint arXiv:1505.06622 (2015)
2 Tomer Kotek, Mantas Simkus, Helmut Veith, Florian Zuleger: Extending ALCQIO with Trees. LICS 2015: 511-522
3 Diego Calvanese, Tomer Kotek, Mantas Simkus, Helmut Veith, Florian Zuleger: Shape and Content – A Database-Theoretic Perspective on the Analysis of Data Structures. IFM 2014: 3-17

4 Summary of Working Groups

4.1 Working Group: Benchmarks and Application Domains

The starting point of the workgroup was a quick introduction to exemplary benchmarks in the different domains of the workgroup participants, e.g. a car platooning case study and a particular routing protocol for mobile ad-hoc networks (AODV routing).

Soon the group agreed that a benchmark collection is of great importance for progress (on tools and approaches) in the area of graph-based analysis and verification. The library SMT-LIB substantiates the above claim and was brought up as an example. Here the problem statement is provided by this widely accepted library and it is common that tool developers in the SMT community provide parsers to automatically transfer the SMT-LIB input problems into a form of input their own tools accept. However, the field of graph-based analysis and verification is a lot more diverse.

Consequently, it was discussed if fixing a language in which problems can be stated in our setting can be realised in a reasonable manner at all. On the one hand restricting to an input language would increase the efficiency for reusing benchmarks drastically, while manual translation is laborious and error-prone. On the other hand in this diverse setting there will always be features that a fixed language does not cover.

Stating the problem of a benchmark and the checked properties, e.g. by providing reference models, was accepted as a solution/compromise.

As outcome of the discussion the benchmarks workgroup decided to take action in setting up an (initial) spreadsheet to collect benchmarks. During the session it was filled by benchmarks that participants suggested to serve as exemplary entries.

Benchmarks Spreadsheet

The following points are identified to be the goal of the benchmarks collection:

- state problems, not solutions
- obstacles to access and edit should be very low
- for now a more or less unsorted collection is targeted, categorisation is provided by tags users can enlist

Concept Discussion

After first setting up an initial spreadsheet for the collection, details on its contents are discussed in the second session.

The group agreed on collecting information on the benchmark's name and a (very) short description. Moreover, a category with tags like 'distributed' or 'seq. heap' and the underlying graph dynamics (e.g. adding/removing edges/nodes) as well as difficulties of the benchmark and interesting properties are requested. Optionally, it is possible to provide the originator and a reference model. For additional information, such as example models for the benchmark or solutions/papers that tackle it, a column 'References' is added. Here one can provide a link to a subfolder where the information (e.g. in the form other further spreadsheets, plain text, ...) is found.

To guide new users, explaining the purpose of the benchmarks collection and the policy regarding the quality of the entries, an additional Readme-document is made available.

The second session was completed by a short demo on a reference model provided as a graph transformation system (in a file format that is accepted by the GROOVE tool).

Access

The benchmarks spreadsheet is made available to the public via a google spreadsheet accessible at

http://tiny.cc/egbd.

The spreadsheet was presented to all participants of the seminar with the appeal to spread the word.

4.2 Working Group: Specification Languages for Graphs

The working group was formed in order to compare, discuss and assess various specification formalism for graphs. Specifically, it discussed the following questions: (i) Which logics

and/or specification languages are used by the different communities to specify (possibly infinite) sets of graphs? (ii) How do they compare with respect to their expressiveness? (iii) What are possible applications? (iv) What are desirable properties of such specification languages?

In order to answer questions (i) and (ii) the working group compiled the following list of specification languages and discussed their expressiveness:

- Recursively enumerable graph languages
- Context-free graph languages, generated by hyperedge-replacement grammars
- Monadic second-order logic, with strong ties to recognizable graph languages
- First-order logic, equivalent to nested graph conditions
- Separation logic
- Forest automata
- Type graphs
- DATALOG
- Propositional Dynamic Logic
- Spatial Logic for Closure Spaces

Afterwards the group talked about item (iii) and discussed applications in verification such as reachability analysis, bounded model-checking and counterexample-guided abstraction refinement.

Finally the working group turned its attention to point (iv) and came up with the following list of important and/or desirable properties of such specification languages (with respect to decidability and complexity):

- Membership test
- Satisfiability
- Entailment
- Computation of post-/preconditions (for certain transformations)
- Invariant checking/closure under rewriting
- Computation of interpolants and interpolant-like notions
- Expressiveness (which properties can be expressed?)
- Ease of use (i.e., a specification language should specify a set of graphs that is expressible in the underlying formalism in a way that is both modular and easy to read/manipulate)
- Closure properties (wrt. negation, disjunction, conjunction, concatenation, ...)
- Framed inference ($F\star? \models G\star?$)
- Widening/approximation

In the end the main open question that arose is to find the best balance between expressiveness and complexity (for problems such as satisfiability, entailment and computation of post-/preconditions). Naturally, this has to be tailored according to the the application. The central open problem that has emerged during the discussion is the need of a better classification/overview of available specification languages with respect to their relative expressiveness and their properties.

4.3 Working Group: Ownership

The working group discussed the various techniques that are used to reason about how the ownership of various resources is distributed to different components of a software system.

Ownership disciplines became very important for programming languages, such as C and C++, where memory is explicitly managed. In such setting, it is very important that unused memory cells are deallocated exactly once. Deallocating a given memory cell multiple times may corrupt the contents of memory, while not deallocating a memory cell constitutes a space leak. The ownership of a memory cell is a very useful concept, because it defines the component responsible for deallocating the cell. Moreover, since ownership of a memory cell is required for a function to access the cell, ownership systems ensure memory safety; that is, absence of memory errors, such as accessing a deallocated memory cell.

Ownership is also particularly useful in the concurrent setting, where each thread requires ownership of a memory cell in order to access it. By doing so, one can ensure that a program does not have any data races. This means that one does not need to worry about any weak memory effects of the hardware, because weak memory models typically ensure that race-free programs have interleaving semantics.

An important ownership technique that the group discussed at length was *separation logic* [13]. In separation logics, the unit of ownership is a memory cell, but small pieces of ownership can be combined together and abstracted away using *abstract predicates* [12]. Extensions of separation logic with fractional and counting *permissions* [2, 1] enable also partial ownership of a memory cell, which is sufficient for reading, but not writing to it.

Concurrent separation logic (CSL) [11] is an extension of separation logic that handles interleaving concurrency. CSL allows a component to increase its ownership capabilities by acquiring a lock, and decrease them by releasing a lock. Carefully chosen invariants for locks enable even *ownership transfer* from one thread to another via a lock synchronization. This idea has also been used in the weak memory setting using *relaxed separation logic* [16]. Further, by careful ownership transfer of fractional permissions, one can encode various other ownership disciplines, such as a single writer with multiple atomic readers. Ownership is a very important component of almost all modern concurrent program logics such as RGSep [17], LRG [6], CAP [5], CaReSL [15], TADA [4], IRIS [8].

Besides separation logic, there are other ways of ensuring an ownership discipline. There is a rich literature on *ownership type systems*, which are very useful for describing well-structured forms of ownership. A nice survey about ownership types is given by Clarke et al. [3]. Unfortunately, there is no corresponding survey for separation logic techniques. Another more recent idea is that of *implicit dynamic frames* [14]. Moreover, there are several tools based on these ideas. Examples are Dafny [9], VeriFast [7], and Viper [10].

References

1. R. Bornat, C. Calcagno, P. W. O'Hearn, and M. J. Parkinson. Permission accounting in separation logic. In *POPL*, pages 259–270. ACM, 2005.
2. J. Boyland. Checking interference with fractional permissions. In *10th SAS*, volume 2694 of *LNCS*, pages 55–72. Springer, 2003.
3. D. Clarke, J. Östlund, I. Sergey, and T. Wrigstad. Ownership types: A survey. In *Aliasing in Object-Oriented Programming*, volume 7850 of *LNCS*, pages 15–58. Springer, 2013.
4. P. da Rocha Pinto, T. Dinsdale-Young, and P. Gardner. Tada: A logic for time and data abstraction. In *ECOOP*, volume 8586 of *LNCS*, pages 207–231. Springer, 2014.
5. T. Dinsdale-Young, M. Dodds, M. P. Philippa Gardner, and V. Vafeiadis. Concurrent abstract predicates. In *ECOOP'10*, Lecture Notes in Computer Science. Springer, 2010.
6. X. Feng. Local rely-guarantee reasoning. In *POPL*, pages 315–327, 2009.
7. B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens. VeriFast: A powerful, sound, predictable, fast verifier for C and Java. In *NASA Formal Methods*, volume 6617 of *LNCS*, pages 41–55. Springer, 2011.

8 R. Jung, D. Swasey, F. Sieczkowski, K. Svendsen, A. Turon, L. Birkedal, and D. Dreyer. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In *POPL*, pages 637–650. ACM, 2015.

9 K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *LPAR (Dakar)*, volume 6355 of *LNCS*, pages 348–370. Springer, 2010.

10 P. Müller, M. Schwerhoff, and A. J. Summers. Viper: A verification infrastructure for permission-based reasoning. In *VMCAI*, volume 9583 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2016.

11 P. W. O'Hearn. Resources, concurrency and local reasoning. *Theoretical Computer Science*, 375(1-3):271–307, 2007.

12 M. J. Parkinson and G. M. Bierman. Separation logic and abstraction. In *POPL*, pages 247–258. ACM, 2005.

13 J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.

14 J. Smans, B. Jacobs, and F. Piessens. Implicit dynamic frames. *ACM Trans. Program. Lang. Syst.*, 34(1):2, 2012.

15 A. Turon, D. Dreyer, and L. Birkedal. Unifying refinement and Hoare-style reasoning in a logic for higher-order concurrency. In *ICFP*. ACM, 2013.

16 V. Vafeiadis and C. Narayan. Relaxed separation logic: A program logic for C11 concurrency. In *OOPSLA 2013*, pages 867–884. ACM, 2013.

17 V. Vafeiadis and M. Parkinson. A marriage of rely/guarantee and separation logic. In L. Caires and V. T. Vasconcelos, editors, *CONCUR*, volume 4703 of *LNCS*, pages 256–271. Springer, 2007.

4.4 Working Group: Graph Rewriting for Verification

Graph rewriting is a rule-based formalism for the transformation of graphs. Rules are local and replace a left-hand side with a right-hand side, taking also some embedding information into account. In his invited talk the day before Arend Rensink distinguished between three types of graph transformation systems: (i) grammars (in which case we are interested in the language, or set, of terminal graphs for a fixed start graph); (ii) production systems (in which case we are interested in the relation between start and terminal graphs); and (iii) behavioural specifications (in which case we are interested in the transition system as a whole).

The aim of the working group was to discuss applications of graph rewriting in verification. In principle, two possible applications came to mind: (1) using graph rewriting as an auxiliary technique to support other verification methods; (2) developing methods in order to specifically analyze and verify graph transformation systems.

As a first case study (presented by Christoph Haase) the group discussed a production system to support entailment checks in separation logic. In separation logic it is not possible to do entailment checks directly, but only after having reduced the formulas to normal form. The group studied a reduction rule that collapses two nodes in case there are certain distinct paths.

Such rules have a non-local flavour and can not be straightforwardly expressed in classical graph rewriting. The group discussed possible extensions that would be desirable in order to be able to express such rules.

The second case study (presented by Barbara König) was a behavioural specification (a termination detection protocol), where the aim is to verify whether an invalid configuration (termination has been declared, but there are still active processes) can be reached. For this specific task it is for instance possible to use a backward analysis technique based on well-structured transition systems. For finite systems also classical model-checking can be used, but many interesting systems do not satisfy finiteness.

A third case study (presented by Richard Trefler) involved the verification of telephone communication. In this case we have a graph that is evolving and de-evolving and models the communication structure, and the protocol can be described by local rule-based transformations. Verification is concerned with problems such as loop-checking, involving compositional reasoning (in the sense that the description of the global behaviour is given by means of the description of the local ones).

The working group concluded by discussing the question of which concepts in graph rewriting could be helpful in the design of verification techniques. One answer was the concept of morphisms that allow us to precisely describe the relation between two graphs or the occurrence of a graph within a host graph.

Figure 1 The seminar group during the excursion on Wednesday afternoon.

Participants

- Mohamed-Faouzi Atig
Uppsala University, SE
- Parosh Aziz Abdulla
Uppsala University, SE
- Peter Backes
Universität des Saarlandes, DE
- Paolo Baldan
University of Padova, IT
- Ahmed Bouajjani
University of Paris VII, FR
- Andrea Corradini
University of Pisa, IT
- Aiswarya Cyriac
Uppsala University, SE
- Giorgio Delzanno
University of Genova, IT
- Cezara Dragoi
IST Austria – Klosterneuburg, AT
- Constantin Enea
University of Paris VII, FR
- Javier Esparza
TU München, DE
- Fabio Gadducci
University of Pisa, IT
- Silvio Ghilardi
University of Milan, IT
- Holger Giese
Hasso-Plattner-Institut – Potsdam, DE
- Christoph Haase
ENS – Cachan, FR
- Annegret Habel
Universität Oldenburg, DE
- Reiko Heckel
University of Leicester, GB
- Alexander Heußner
Universität Bamberg, DE
- Lukas Holik
Brno Univ. of Technology, CZ
- David Janin
University of Bordeaux, FR
- Christina Jansen
RWTH Aachen University, DE
- Bengt Jonsson
Uppsala University, SE
- Joost-Pieter Katoen
RWTH Aachen, DE
- Barbara König
Universität Duisburg-Essen, DE
- Tomer Kotek
TU Wien, AT
- Narayan Kumar Krishnan
Chennai Mathematical Inst., IN
- Leen Lambers
Hasso-Plattner-Institut – Potsdam, DE
- Michele Loreti
University of Firenze, IT
- Roland Meyer
TU Kaiserslautern, DE
- Thomas Noll
RWTH Aachen, DE
- Fernando Orejas
UPC – Barcelona, ES
- Eugenio Orlandelli
CIS, IT
- Oded Padon
Tel Aviv University, IL
- Detlef Plump
University of York, GB
- Chris Poskitt
ETH Zürich, CH
- Arend Rensink
University of Twente, NL
- Ahmed Rezine
Linköping University, SE
- Leila Ribeiro
Federal University of Rio Grande do Sul, BR
- Xavier Rival
ENS – Paris, FR
- Arnaud Sangnier
University of Paris VII, FR
- Richard Trefler
University of Waterloo, CA
- Viktor Vafeiadis
MPI-SWS – Kaiserslautern, DE
- Tomas Vojnar
Brno Univ. of Technology, CZ
- Thomas Wies
New York University, US
- Florian Zuleger
TU Wien, AT