ISSN 2186-7437

NII Shonan Meeting Report

No. 2017–14

Analysis and Verification of Pointer Programs

Marieke Huisman
Thomas Noll
Makoto Tatsuta

October 2–5, 2017

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Analysis and Verification of Pointer Programs

Organizers:
Marieke Huisman (University of Twente, The Netherlands)
Thomas Noll (RWTH Aachen University, Germany)
Makoto Tatsuta (National Institute of Informatics, Japan)

October 2–5, 2017

Computer software is ubiquitous in today's information society, and ensuring its correctness is of great importance. This is particularly true for *safety-critical systems*, which occur in transportation, communication, healthcare, and many other application domains. Consequently, software failures can have severe consequences. To tackle the problem of correctness and reliability of such software, *formal methods* are increasingly being employed. They enable the exhaustive and mathematically founded analysis of all possible behaviours of a computer program and the verification of properties such as functional correctness. They also allow to reduce the effort and, thus, the cost of testing activities. Due to their benefits, they are increasingly becoming an integral part of the development cycle of safety-critical systems.

Many software bugs can be traced back to the erroneous use of *pointers*, i.e., references to memory addresses. They constitute an essential concept in modern programming languages, and are used for realising (dynamic) data structures like lists, trees etc., which are organised in the computer's memory as a so-called *heap*. Pointers are also abundantly present in object-oriented software such as Java collections, albeit in the somewhat implicit form of object references. Pointer-handling operations occur in device drivers, operating systems, and all kinds of application codes including those implementing safety-critical systems. The analysis of such software is a highly demanding and important task, as memory leaks, dereferencing null pointers or the accidental invalidation of data structures can cause great damage especially when software reliability is at stake. Moreover, the increasing presence of *concurrency* in modern computing raises additional problems such as the non-synchronised access to memory areas, which can entail so-called data races. Even worse, the formal analysis of concurrent software poses additional challenges caused by the non-deterministic execution order (interleaving) between different strands of concurrent activities.

In consequence, the complexity of state spaces arising from dynamic data structures, recursive method or procedure calls, and dynamic creation of and interleaving between concurrent threads imposes challenges which cannot be handled by standard verification algorithms such as finite-state model checking. This problem has been a topic of continuous research interest since the early 1970s. A common approach are *abstraction techniques* that employ symbolic representations of sets of program states using suitable formalisms. Various such formalisms have been considered for this purpose, which can be distinguished

with regard to their expressiveness, precision, efficiency, and automatability. The most important ones are sketched in the following.

Automata-based approaches. *Regular model checking* is a generic technique for algorithmic verification of infinite-state systems which uses (finite) word, tree, or forest automata to finitely represent potentially infinite sets of reachable configurations. As the related problems are typically undecidable, research activities in this area have focused on developing semi-algorithms, decidability results for restricted cases, and systematic over-approximations of the state space by means of acceleration techniques.

Logic-based approaches. *Shape analysis* is a form of pointer analysis that attempts to characterize collections of heap cells reachable from particular pointers, for example, to determine whether the cells form a list or a tree. In particular, three-valued logic can be used as a framework for designing such analyses. It is instantiated by supplying both predicates that capture different relationships between cells and the functions that determine how the predicates are updated by particular program operations. Another logic-based technique is *separation logic*, which extends classical Hoare logic by operators that enable modular reasoning about heap structures. While the logic was originally designed as a calculus for manually verifying low-level pointer manipulating programs, it has subsequently become the basis for several abstract interpretation based verifiers. Since then, this topic has been actively studied with regard to both its theoretical foundations and its practical applications. Its importance is witnessed by the 2016 Gödel Prize, which was awarded to Stephen Brookes and Peter W. O'Hearn for their invention of Concurrent Separation Logic.

Graph-based approaches. As heap data structures can be formalised by graphs, it is quite natural to employ *graph transformation* techniques for both specifying symbolic execution of pointer programs and abstraction mappings on heap structures. With respect to the former, methods for verifying graph transformation systems based on model checking and similar techniques are being studied. With respect to abstraction methods, *graph grammars* such as regular tree grammars and hyperedge replacement grammars are employed for describing the shape of complex data structures.

Extensions for concurrency. As described earlier, another source of infinite state spaces is unbounded concurrency, i.e., dealing with a parametric number or with unbounded dynamic creation of threads. In order to meet this challenge, for all of the approaches mentioned above, extensions for concurrency have been developed. They are ranging from adaptations of regular model checking over specific kinds of abstraction and automated induction based on network invariants to thread-modular reasoning. An especially challenging task is to verify programs that involve both unbounded concurrency as well as unbounded data structures. In fact, the survey on software model checking by R. Jhala and R. Majumdar states that "the scalable verification of heap-based data structures is perhaps the least understood direction in software model checking" and that "scaling verification techniques in the presence of expressive heap abstractions and concurrent interactions remain outstanding open problems."

Aims of the meeting. Although many of these approaches address similar concerns, their formulations and formalisms often seem dissimilar and sometimes even unrelated. Thus, the insights and results gained in one description of heap abstraction may not directly carry over to some other descriptions. The purpose of this Shonan meeting was to bring together both theoreticians and practitioners working on different techniques for heap abstraction and pointer program analysis. It aimed to provide a broad understanding of the various techniques to support the exploitation of their commonalities such that they can benefit from each other.

Overview of Talks

A Logical System for Modular Information Flow Verification

Mahmudul Faisal Al Ameen, National University of Singapore, Singapore

Information Flow Control (IFC) is important to ensure secure program where secret data does not influence any public data. The pervasive standard that IFC aims is non-interference. Current IFC systems are separated into dynamic IFC, static IFC, and the hybrid between static and dynamic. Dynamic IFC suffers from high overhead and limited ability to prevent implicit flows due to the paths not taken, we propose a novel modular static IFC systems. To the best of our knowledge, this is the first modular static IFC systems. Unlike type-based static IFC systems, ours is separation logic-based. The limitation of type-based IFC systems is in the inviolability of static security label declarations for fields. As such, they suffer from transient leaks on fields. Our proposed system verifies each function independently with the help of separation logic. Furthermore, we provide the proof of correctness for our novel IFC systems with respect to termination and timing insensitive non-interference.

Foundational Program Verification Using VST

Lennart Beringer, Princeton University, USA

The Verified Software Toolchain is a verification framework for C, implemented in the Coq proof assistant and proven sound w.r.t. the operational semantics of CompCert's Clight language. Its key components are a higher-order impredicative concurrent separation logic and a library of proof automation tactics. The talk will give a high-level overview and summarize some recent verification examples. Meeting participants are invited to download and install VST on their own machine (see `http://vst.cs.princeton.edu`) and experiment with the system in their spare time.

Relational Models and Program Logics: Logical Relations in Iris

Lars Birkedal, Aarhus University, Denmark

In this talk I present a formalization of a logical relations model of an ML-like type system for a call-by-value higher-order language with impredicative polymorphism, recursive types, general references, and concurrency. The logical relation interpretation is defined in Iris, a state-of-the-art higher-order concurrent separation logic, which in turn is formalized in Coq. The proof effort is made simpler by the use of the novel interactive proof mode for Iris Proof Mode.

Starling: Lightweight Reasoning with Separation

Mike Dodds, University of York, UK

Starling is a lightweight, automated tool for verifying racy concurrent algorithms. Starling proofs are are written in an abstracted Hoare-logic style, and converted into terms discharged by a sequential solver (for example, Z3). Starling is built on the Views framework, an abstract form of separation logic. In this talk I'll describe how we specialise the Views framework into a simple, generic verification tool, and how we can apply this approach to verify complex pointer programs.

Automatic Local Reasoning of Recursive Data Structures

Joxan Jaffar, National University of Singapore, Singapore

We consider the problem of verifying programs which manipulate recursive data structures. A main challenge here is how to perform local reasoning so that the verification of subprograms, which operate on different components or frames of the data structure, can be combined. Separation Logic (SL) was a significant advance in program verification of data structures. It used a "separating" conjoin operator in data structure specifications to construct heaps from disjoint subheaps, and a "frame rule" to very elegantly realize local reasoning. Subsequently, the methods of Dynamic Frames (DF) and Implicit Dynamic Frames (IDF) provided expressive ways to specify the frames of methods.

Our method begins with a domain of discourse of explicit subheaps with recursive definitions. The resulting specification language can describe arbitrary data structures, and arbitrary sharing therein, thus enabling a very precise specification of frames. The main contribution then is a program verification method which combines strongest postcondition reasoning in the form of symbolic execution, and unfolding recursive definitions of the data structure in question. Conceptually, this makes our method relatively complete in the sense of Hoare Logic. Finally, we present an implementation of our verifier, which essentially reduces to an implementation of unfolding recursive definitions. We then demonstrate automation on a number of representative programs.

Finally, to demonstrate that a new level of automatic verification has been achieved, we present the first automatic proof of a classic graph marking algorithm, paving the way for dealing with a class of programs which traverse complex data structures.

(Joint Work with Duc-Hiep Chu)

The Attestor Tool: Graph-Based Abstract Interpretation in Practice

Christina Jansen, RWTH Aachen University, Germany

The automated analysis and verification of pointer-manipulating programs operating on a heap is a challenging task. It requires abstraction techniques for dealing with complex program behaviour and unbounded state spaces that arise from both dynamic data structures and recursive procedures. In this talk I am going to briefly present the theoretical underpinnings of a static analysis for pointer programs, which fits into the standard abstract interpretation framework. Its abstraction and (local) concretisation functions are defined by graph grammar application. We will have a close look into the prototypic analysis tool Attestor, which analyses Java Bytecode for garbage-freedom, null pointer dereferences, pointer reachability, shape preservance as well as complex functional properties. In detail, we consider case studies including singly- and doubly-linked list reversal, the Deutsch-Schorr-Waite tree traversal algorithm and diverse operations on AVL trees.

Decision Procedure for Entailment of Symbolic Heaps with Arrays

Daisuke Kimura, Toho University, Japan

This talk gives a decision procedure for the validity of entailment of symbolic heaps in separation logic with Presburger arithmetic and arrays. The correctness of the decision procedure is proved under the condition that sizes of arrays in the succedent are not existentially bound. This condition is independent of the condition proposed by the CADE-2017 paper by Brotherston et al, namely, one of them does not imply the other. For improving efficiency of the decision procedure, some techniques are also presented. The main idea of the decision procedure is a novel translation of an entailment of symbolic heaps into a formula in Presburger arithmetic, and to combine it with an external SMT solver.

Iris: A Framework for Higher-Order Concurrent Separation Logic in Coq

Robbert Krebbers, Technical University Delft, The Netherlands

Iris is a framework for higher-order concurrent separation logic, implemented in the Coq proof assistant. In collaboration with a growing network of collaborators, Iris has been deployed in a wide variety of verification projects such verification of fine-grained concurrent data structures, a safety proof of the Rust

type system, logical relations for proving program refinements, and program logics for relaxed memory models.

In this talk I will give an overview of Iris. Firstly, I will introduce the basic building blocks of Iris and show how these can be used to verify classic concurrent programs. Secondly, I will demonstrate the formalization of Iris in Coq.

Enhancing Symbolic Execution of Heap-based Programs with Separation Logic for Test Input Generation

Quang Loc Le, Teesside University, Middlesbrough, UK

Symbolic execution is a well-established method for test input generation. By taking inputs as symbolic values and solving constraints encoding path conditions, it helps achieve a better test coverage. Despite of having achieved tremendous success over numeric domains, existing symbolic execution techniques for heap-based programs (e.g., linked lists and trees) are limited due to the lack of a succinct and precise description for symbolic values over unbounded heaps.

In this work, we present a new symbolic execution method for heap-based programs using separation logic. The essence of our proposal is the use of existential quantifiers to precisely represent symbolic heaps. In order to solve path-condition-constraints, we first present a satisfiability solver in a fragment of separation logic with inductive predicates and arithmetic. Furthermore, we identify conditions for a decidable subfragment. Next, we propose a context-sensitive lazy initialization, a novel approach for efficient test input generation. In particular, we describe a least fixed point analysis to compute a valid set of symbolic values initialized to reference fields during the symbolic execution. We have implemented our proposal into a prototype system, called Java StarFinder and S2SAT solver, and evaluated it on a set of programs with complex heap inputs. The results show that our approach significantly reduces the number of invalid test inputs and improves the test coverage.

Automated Detection of Dynamic Data Structures in C Programs and Binary Code

Gerald Lüttgen, University of Bamberg, Germany

This talk presents the key results of the DFG-funded research project "Learning Data Structure Behaviour from Executions of Pointer Programs" (DSI), in which dynamic analysis techniques have been developed to identify dynamic data structures in C programs and x86 binary code. DSI's analysis utilizes a novel memory abstraction that allows for a compact description of pointer-based data structures such as linked lists and binary trees, and their interconnections such as parent-child nesting. On top of this abstraction, an evidence-collecting approach calculates a natural language description of the observed data structures with the help of a systematic taxonomy. The inferred data structure information is not only helpful for program comprehension but can also be utilized in the contexts of software verification and visualization.

Heap Automata for Pointer Programs and Separation Logic

Christoph Matheja, RWTH Aachen University, Germany

We introduce heap automata, a formalism for automatic reasoning about robustness properties of the symbolic heap fragment of separation logic with user-defined inductive predicates. Robustness properties, such as satisfiability, reachability, and acyclicity, are important for a wide range of reasoning tasks in automated program analysis and verification based on separation logic. Previously, such properties have appeared in many places in the separation logic literature, but have not been studied in a systematic manner.

In this talk, we develop an algorithmic framework based on heap automata that allows us to derive asymptotically optimal decision procedures for a wide range of robustness properties in a uniform way. We implemented a prototype of our framework and obtained promising results for all of the aforementioned robustness properties. Further, we demonstrate the applicability of heap automata beyond robustness properties. We apply our algorithmic framework to the model checking and the entailment problem for symbolic-heap separation logic.

Complete Cyclic-Proof System for Separation Logic with General Inductive Predicates

Koji Nakazawa, Nagoya University, Japan

A new proof system, called $CSLID^{\omega}$, is introduced for entailment checking in separation logic with general inductive predicates. $CSLID^{\omega}$ is based on Brotherston's cyclic proof system, and accompanied by an unfold-match proof-search procedure. We expect that this procedure proves decidability and completeness of $CSLID^{\omega}$.

Graph-Based Abstract Interpretation of Pointer Programs

Thomas Noll, RWTH Aachen University, Germany

In this talk we introduce an abstraction framework for analysing pointer programs featuring dynamic data structures, recursive procedures, and concurrent threads. It uses a graph-based symbolic representation of sets of heaps and employs so-called hyperedge replacement grammars to describe both abstraction and concretisation operations on symbolic heaps. Modular reasoning is supported in the form of contracts with graphical pre- and postconditions that capture the net effect of a procedure's and thread's execution. In the latter case, contracts are enriched by permissions that represent access rights to (parts of) the heap, which allows to check for race conditions and other concurrency issues.

Automated Reasoning about Separation Logic using SMT Solvers

Ruzica Piskac, Yale University, New Haven, USA

Separation logic (SL) has gained widespread popularity as a formal foun-

dation 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.

A Decidable Logic for Tree Data-Structures with Measurements

Xiaokang Qiu, Purdue University, West Lafayette, USA

We present Dryad_dec, a decidable logic that allows reasoning about tree data-structures with measurements. This logic supports user-defined recursive measure functions based on height or size, and measure-related recursive predicates such as AVL trees or red-black trees. We prove the logic's satisfiability is decidable. The crux of the decidability proof is a small model property which allows us to reduce the satisfiability of Dryad_dec to quantifier-free linear arithmetic theory which can be solved efficiently using SMT solvers. We also show that Dryad_dec can encode natural proof verification conditions for functional correctness of recursive tree-manipulating programs, as well as synthesis conditions for conditional linear-integer arithmetic functions. We developed the decision procedure and successfully solved 100+ Dryad_dec formulas raised from various scenarios, including verifying full correctness of programs manipulating AVL trees, red-black trees and treaps, checking size lower bounds for AVL trees and red black trees, and synthesizing candidate solutions from a specification and a set of counterexamples. To our knowledge, this is the first decidable logic that can express these measure-related properties for trees.

Abstract Graphs and their Transformation

Arend Rensink, University of Twente, Enschede, The Netherlands

The data structures built up in pointer programs can for many purposes be viewed as graphs, where the nodes are records and the edges are pointers. The manipulation of that data by a program then corresponds to the transformation of such a graph.

By regarding portions of a data graph that are "similar enough" (in a sense to be defined precisely but dependent on the application) as identical, and merely recording how many of each such portion there are rather than their individual interconnections, we can arrive at a finite model that captures the essential characteristics of pointer data. The transformations can then be lifted from concrete graphs to this abstract level, giving rise to an over- or under-approximation of the reachable states that allows a partial prediction of the behaviour of the original pointer program.

In this presentation I give an overview of graph abstraction techniques that have been studied for this purpose, and identify the most promising approaches.

Forest Automata for Verification of Heap Manipulation

Adam Rogalewicz, Brno University of Technology, Czech Republic

We consider verification of programs manipulating dynamic linked data structures such as various forms of singly and doubly-linked lists or trees. We consider important properties for this kind of systems like no null-pointer dereferences, absence of garbage, shape properties, etc. We develop a verification method based on a novel use of tree automata to represent heap configurations. A heap is split into several "separated" parts such that each of them can be represented by a tree automaton. The automata can refer to each other allowing the different parts of the heaps to mutually refer to their boundaries. Moreover, we allow for a hierarchical representation of heaps by allowing alphabets of the tree automata to contain other, nested tree automata. Program instructions can be easily encoded as operations on our representation structure. This allows verification of programs based on a symbolic state-space exploration together with refinable abstraction within the so-called abstract regular tree model checking. A motivation for the approach is to combine advantages of automata-based approaches (higher generality and flexibility of the abstraction) with some advantages of separation-logic-based approaches (efficiency). We have implemented our approach and tested it successfully on multiple non-trivial case studies.

The Tree Width of Separation Logic with Recursive Definitions

Adam Rogalewicz, Brno University of Technology, Czech Republic

Separation Logic is a widely used formalism for describing dynamically allocated linked data structures, such as lists, trees, etc. The decidability status of various fragments of the logic constitutes a long standing open problem. Current results report on techniques to decide satisfiability and validity of entailments for Separation Logic(s) over lists (possibly with data). In this paper we establish a more general decidability result. We prove that any Separation Logic formula using rather general recursively defined predicates is decidable for satisfiability, and moreover, entailments between such formulae are decidable for validity. These predicates are general enough to define (doubly-) linked lists, trees, and structures more general than trees, such as trees whose leaves are chained in a list. The decidability proofs are by reduction to decidability of Monadic Second Order Logic on graphs with bounded tree width.

Hybrid Program Analyses for Pointwise Permission Inference

Alexander J. Summers, ETH Zürich, Switzerland

Ownership and permissions are concepts commonly employed to aid reasoning about programs with mutable state and concurrency, e.g. in custom program logics such as separation logic. Permissions can be used to specify the potential side-effects of code, guaranteeing which facts can be preserved across changes to the program state. One way to generalise permissions to unbounded data is to support them under quantifiers; e.g. specifying access to a graph structures by ranging over its sets of nodes, or to array segments by ranging over integer indices.

In this talk, I will describe ongoing work to automatically infer such quantified permission specifications for a variety of heap-based data structures. Permission-based program logics extend first-order logic with powerful additional connectives, but I will show that the constraints arising from our inference problem can nonetheless be summarised within first-order arithmetic. Using this idea, we define a precise analysis for straight-line code, which can e.g. summarise the permissions needed to execute a single loop iteration. We then generate loop invariants using several novel techniques for projecting such expressions out over all loop iterations, leveraging complementary static analyses and quantifier elimination algorithms.

The talk will include demonstrations using the Viper verification infrastructure, which I will introduce along the way.

Program Analysis and Verification by Separation Logic

Makoto Tatsuta, National Institute of Informatics, Japan

Our team is currently working on a separation-logic-based program analyzer/verifier. Our plan is to start at the point O'Hearn's group reached and to achieve more precise and faster systems. Our concrete target is to verify OpenSSL. Our current topics are: an entailment checker for separation logic with arithmetic and arrays, a loop invariant generator for Hoare triples by abstract interpretations, and completeness of cyclic proofs in symbolic heaps with general inductive predicates.

An Abstract Interpretation Framework for Input Data Usage

Caterina Urban, ETH Zürich, Switzerland

Nowadays, data science software plays an increasingly important role in critical decision making in fields ranging from economy and finance to biology and medicine. As we rely more and more on data science for making decisions, we become increasingly vulnerable to programming errors. Errors that do not cause failures can have serious consequences, since they give no indication that something went wrong.

In this talk, we focus on programming errors related to input data usage. Specifically, we propose an abstract interpretation framework to automatically detect unused input data. We systematically derive static analyses for data usage by abstraction of the program operational trace semantics. We propose a new abstract domain to detect single unused input data stored in scalar variables, and we lift this abstraction by building upon an existing domain for the analysis of compound data structures such as array and lists to detect unused chunks of the data. Finally, we show that existing static analyses for seemingly different problems can be cast into our framework. In particular, we show that a form of live variable analysis and secure information flow analyses can be used for input data usage, with varying degrees of precision.

Flow Interfaces -– Compositional Abstractions for Concurrent Data Structures

Thomas Wies, New York University, USA

Concurrent separation logics have helped to significantly simplify correctness proofs for concurrent data structures. However, a recurring problem in such proofs remains: data structure abstractions based on inductive predicates, which work well in the sequential setting, are much harder to reason about in a concurrent setting. To solve this problem, we propose a novel approach to abstracting regions in the heap by encoding the data structure invariant into a local property that can be checked on individual nodes. These properties may depend on a quantity of each node that is computed as a fixpoint over the entire heap graph. We refer to this quantity as a flow. Flows can encode both structural properties of the heap (e.g., the reachable nodes from the root form a tree) as well as data invariants that are relevant for proving functional correctness. We then introduce the notion of a flow interface, which expresses the relies and guarantees that a heap region imposes on its context to maintain the global flow invariant.

Our main technical result is that flow interfaces provide a new semantic model for separation logic assertions that admits general implementation-agnostic proof rules for reasoning about concurrent data structures. These include rules that allow a heap region to be split into arbitrary chunks which can be modified and recomposed to form a new region, while maintaining the global data structure invariant. We have used our new approach to obtain simple correctness proofs for non-trivial concurrent dictionary implementations based on B+ trees and non-blocking lists.

Summary of Discussions

The workshop was closed with a plenary discussion. Before and during the workshop, we had collected topics for discussion, which were: abstraction, proof automation, how to combine methodologies and tools, common challenges, and the spread of techniques for pointer analysis.

Most work on pointer analysis seems to be using separation logic. We started the discussion with the question if there are specific reasons for that, but no real conclusions were made. As somebody pointed out: “it’s all the same.”

Moreover, we only gathered people from the verification community; for example in the compiler community still many different techniques are used.

During the workshop we realised that there was a wide interest in common challenges. During the first day of the workshop, Joxan Jaffar in his presentation introduced the mark-graph challenge, and several people later used this example in their own presentations to illustrate their techniques. We agreed that it is a good idea to collect challenges, and make them publicly available for the whole community. We distinguish two kinds of challenges that are relevant for this community: program verification challenges, and separation logic entailment challenges. For both kinds of challenges, there already exists a competition: VerifyThis for program verification, SLComp for separation logic entailment. It was agreed that we should connect with these two initiatives for the collection of challenges.

The discussion then continued to consider the possibility of combining techniques and tools, in order to allow reuse of results. This idea should not be restricted to the use of tools and techniques discussed during the workshop; also the idea to plug in existing static analyses in program verification tools seems to be a promising approach. To combine existing tools and techniques would require some common interchange format for properties (and competitions such as SLComp can be good means to define this). It was noted that a wide variety of different separation logic variants is currently in use. It could be useful to have some kind of feature model, which allows one to compare the capacities of these logics (in addition to the hierarchy that is often shown currently).

Finally, we also discussed where we should concentrate efforts. Should we continue developing deep theory, or focus on scaling, and work on the verification of industrially relevant examples or libraries? The latter also requires substantial engineering work, which can be difficult to realise in an academic environment. However, we also see that there is a growing interest in these kinds of techniques by companies such as Amazon, Facebook and Galois, so we concluded that in addition to further developing the theory, we should also keep on considering how to put these theories into practice.

List of Participants

- Mahmudul Faisal Al Ameen, National University of Singapore, Singapore
- Lennart Beringer, Princeton University, USA
- Lars Birkedal, Aarhus University, Denmark
- Mike Dodds, University of York, UK
- Marieke Huisman, University of Twente, Enschede, Netherlands
- Joxan Jaffar, National University of Singapore, Singapore
- Christina Jansen, RWTH Aachen University, Germany
- Daisuke Kimura, Toho University, Japan
- Robbert Krebbers, Technical University Delft, The Netherlands

- Quang Loc Le, Teesside University, Middlesbrough, UK
- Gerald Lüttgen, University of Bamberg, Germany
- Christoph Matheja, RWTH Aachen University, Germany
- Koji Nakazawa, Nagoya University, Japan
- Chin Wei Ngan, National University of Singapore, Singapore
- Thomas Noll, RWTH Aachen University, Germany
- Ruzica Piskac, Yale University, New Haven, USA
- Xiaokang Qiu, Purdue University, West Lafayette, USA
- Arend Rensink, University of Twente, Enschede, The Netherlands
- Adam Rogalewicz, Brno University of Technology, Czech Republic
- Alexander J. Summers, ETH Zürich, Switzerland
- Makoto Tatsuta, National Institute of Informatics, Japan
- Caterina Urban, ETH Zürich, Switzerland
- Thomas Wies, New York University, USA

Schedule of Meeting

Check-in Day: October 1st (Sun)

- Welcome Banquet

Day1: October 2nd (Mon)

- Introduction
  - Opening by organisers
  - Self-introduction by participants
- General Verification Approaches using Separation Logic
  - Makoto Tatsuta: *Program Analysis and Verification by Separation Logic*
  - Joxan Jaffar: *Automatic Local Reasoning of Recursive Data Structures*
- Group photo shooting
- Separation Logic and Concurrency I
  - Lennart Beringer: *Foundational Program Verification Using VST*
  - Mike Dodds: *Starling: Lightweight Reasoning with Separation*
- Runtime Verification and Input Analysis

- Quang Loc Le: *Enhancing Symbolic Execution of Heap-based Programs with Separation Logic for Test Input Generation*
- Gerald Lüttgen: *Automated Detection of Dynamic Data Structures in C Programs and Binary Code*
- Caterina Urban: *An Abstract Interpretation Framework for Input Data Usage*

Day2: October 3rd (Tue)

- Separation Logic and Concurrency II
  - Robbert Krebbers: *Iris: A Framework for Higher-Order Concurrent Separation Logic in Coq*
  - Lars Birkedal: *Relational Models and Program Logics: Logical Relations in Iris*
- Automata-Based Approaches to Separation Logic
  - Christoph Matheja: *Heap Automata for Pointer Programs and Separation Logic*
  - Adam Rogalewicz: *Forest Automata for Verification of Heap Manipulation*
- Graph Transformation
  - Arend Rensink: *Abstract Graphs and their Transformation*
  - Thomas Noll: *Graph-Based Abstract Interpretation of Pointer Programs*
  - Christina Jansen: *The Attestor Tool: Graph-Based Abstract Interpretation in Practice*
- Decision Problems in Separation Logic
  - Koji Nakazawa: *Complete Cyclic-Proof System for Separation Logic with General Inductive Predicates*
  - Ruzica Piskac: *Automated Reasoning about Separation Logic using SMT Solvers*
  - Adam Rogalewicz: *The Tree Width of Separation Logic with Recursive Definitions*

Day3: October 4th (Wed)

- Separation Logic and Concurrency III
  - Alexander J. Summers: *Hybrid Program Analyses for Pointwise Permission Inference*
  - Thomas Wies: *Flow Interfaces – Compositional Abstractions for Concurrent Data Structures*
- Extensions of Separation Logic I

  - Chin Wei Ngan: *Multi-Party Session Logic*
  - Xiaokang Qiu: *A Decidable Logic for Tree Data-Structures with Measurements*

- Excursion and Main Banquet

Day4: October 5th (Thu)

- Extensions of Separation Logic II
  - Mahmudul Faisal Al Ameen: *A Logical System for Modular Information Flow Verification*
  - Daisuke Kimura: *Decision Procedure for Entailment of Symbolic Heaps with Arrays*

- Plenary Discussion and Closing