ISSN 2186-7437

NII Shonan Meeting Report

No. 2017-12

Memory Abstraction, Emerging Techniques and Applications

Bor-Yuh Evan Chang, Xavier Rival and Sukyoung Ryu

September 11–14, 2017

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

Memory Abstraction,
Emerging Techniques and Applications

Organizers:
Bor-Yuh Evan Chang (CU Boulder),
Xavier Rival (CNRS, ENS, INRIA and PSL* Research University) and
Sukyoung Ryu (KAIST)

September 11–14, 2017

1 Overview of the Meeting

Static analysis aims at computing semantic properties of programs, so as to verify properties such as absence of runtime errors, functional correctness, termination, security, and more. Since these properties are generally undecidable, automatic program analysis tools usually need to be conservative so as to attempt to prove properties of interest. In that process, they need to reason about all components of the semantics of programs (numeric and symbolic computations, parallelism, etc). The notion of abstraction is central in static analysis: an abstraction defines a set of logical properties that a program analysis tool may use, their meaning, and the supporting algorithms. Numeric abstractions made remarkable progresses early in the development of static analysis. However, to reason about complex languages (be it C, Java, ML, or JavaScript), analysis tools also need to use a careful abstraction of the structures stored in memory.

Many kinds of memory abstractions have been introduced in the last thirty years. Initially, memory abstractions mainly consisted of pointer abstractions based on aliasing graphs or points-to relations. Such abstractions cannot cope precisely with data-structures of unbounded size, thus shape analysis techniques were introduced in the 1990s. Shape analysis techniques rely on more complex mathematical objects to describe unbounded memory graphs by summarizing inductive patterns [2]. In the last decade, a large number of novel techniques and applications for memory abstractions have emerged. In the same time, novel abstractions for array data-structure have been developed, and many of these are able to summarize array regions of static or dynamic size. More recently, object structures have become common in dynamic languages and several abstractions have been proposed, that can deal with unbounded structures indexed over unordered keys.

Even though the development of memory abstractions still seems to be lagging compared to, e.g., numeric abstraction, recent progresses seem to make this field more mature. Structure-specific abstractions have been developed and implemented for most common memory data-structures. For most kinds of data-structures, we can now choose among several abstractions that provide different levels of performance in terms of precision and cost. Moreover, client analyses have been identified, that utilize memory abstractions to infer other kinds of properties (safety of sequential or parallel programs, termination, resource consumption).

Still, the memory abstraction research field has several open questions such as:

- the standardization of the concrete semantic models of programming languages regarding to memory;
- the unification of APIs between abstract domains;
- the scalability of summarizing abstractions;
- the cooperation between memory abstractions;
- the effective design of abstractions for data-structures found in dynamic languages;
- the emerging applications, e.g., to liveness, and security.

The purpose of the meeting is to bring together experts in the design and in the use of memory abstractions, so as to leverage on the recent advances in this field, allow for the development of new fundamental principles and tools, and ease the use of memory abstractions in emerging applications. In the next two sections we elaborate on the two main aspects of the envisioned seminar, namely the memory abstraction techniques, and their applications. For each aspect, we explain why we think such a meeting would be timely, and outline the expected benefits.

Overview of Talks

Verivita: Lifestate Verification of Event-Driven Apps

Bor-Yuh Evan Chang, CU Boulder

Bugs in mobile applications are particularly difficult to diagnose and fix because applications are structured as a loosely-coupled set of callbacks responding to events and interacting through a shared heap. And thus crashes are often triggered by unexpected event orderings.

In this talk, I present some of our efforts in developing tools and techniques for finding and fixing bugs in event-driven mobile applications. In particular, we identify a central challenge to analyzing programs developed against event-driven software frameworks like Android is that the possible callbacks that may be invoked by the framework is not static for all applications but can be dynamically updated by the app through its interaction with the framework. From this observation, we develop a predictive testing technique that takes as input a trace of execution of an app to either produce an alternative trace that possibly witnesses violating the protocol or a proof that no such alternative trace is realizable. Finally, I discuss how this approach fits into on-going work in mining and understanding framework specifications from execution traces.

Unified Map Abstractions

Arlen Cox, Institute for Defense Analysis)

In abstract interpretation we find that we are frequently abstracting maps in some form or another. For example, the heap is a map from addresses to values and arrays are maps from indexes to values. This talk looks at unifying some of these abstractions with the goal of borrowing ideas from one area of research to apply in another. We focus on two abstractions, the Heap with Open Objects abstraction by Cox, Chang, and Rival, and the Parametric Segmentation Functor abstraction by Cousot, Cousot, and Logozzo.

JavaScript Static Analysis with WALA

Julian Dolby, IBM

As JavaScript has become ubiquitous through rich client-side Web applications, it has become ever more important to analyze JavaScript code; it has not, however, become any easier. Indeed, the proliferation of frameworks such as jQuery has made static analysis ever more difficult. In this talk, I shall discuss JavaScript static analysis with WALA, describing our series of attempts to deal with the ever-growing complexity of JavaScript applications. In keeping with the theme of this meeting, I shall focus on two key memory abstractions that driven our analysis.

Partially synchronous programming abstractions for fault-tolerant distributed algorithms

Cezara Drăgoi, CNRS, ENS, INRIA and PSL* Research University

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.

One fundamental obstacle in having correct fault-tolerant distributed algorithms is the lack of abstractions when reasoning about their behaviors. In this talk we discuss the impact of partially synchronous programming abstractions in increasing the confidence we have in fault-tolerant systems. We will focus on partially synchronous models that view asynchronous faulty systems as synchronous ones with an adversarial environment that simulates asynchrony and faults by dropping messages. This view simplifies the proof arguments making systems amendable to automated verification. We apply partial synchrony to algorithms that solve agreement problems, such as consensus and state machine replication.

Technically, we take a programming language perspective and define a domain specific language which has a high-level partially synchronous semantics and compiles into efficient asynchronous code. We validate our technique by defining partially synchronous implementations of algorithms like Paxos, whose verification becomes now automated, and which compile into efficient asynchronous code, that preserves the properties verified under the partially synchronous semantics.

Memory Analysis in the Parfait static analysis tool

Nathan Keynes, Oracle

Parfait is an industrial-strength static analysis tool developed in Oracle Labs (formerly Sun Labs) that runs daily over 100s of millions of lines of code inside Oracle. Parfait focuses on finding implementation defects in C/C++ code (buffer overflow, memory leak, etc) and security vulnerabilities in the Java platform (e.g. unguarded caller sensitive methods), with a low false positive rate. In this talk we present a brief overview of the Parfait architecture, characterise some of our reference workloads, and describe the approaches currently taken in Parfait for scalable analysis of memory references. Parfaits analysis is modular, context-sensitive, flow-sensitive and path-sensitive, using a bottom-up approach along similar lines to access-path.

Weakly Sensitive Analysis for Unbounded Iteration over JavaScript Objects

Yoonseok Ko, KAIST

JavaScript framework libraries such as jQuery are widely used, but complicate program analyses. Indeed, they encode high-level constructions such as class inheritance via dynamic object copies and transformations that are harder to reason about. One common pattern used in them consists of loops that copy or transform part or all of the fields of an object. Such loops are challenging to precisely analyze due to weak updates and as unrolling techniques do not always apply. We observe that precise field correspondence relations are required for client analyses (e.g. for call-graph construction). In this talk, I present an effective design of abstractions for unbounded iterations over JavaScript objects to precisely reason about program behaviors in such loops. Our abstractions, we call them a composite abstraction, consist of three layers: the abstraction for iterations, the object abstraction, and the string abstraction. The composite abstraction allows to reason separately about the effect of distinct iterations without resorting to full unrolling and the analysis result shows better precision and scalability than that of any other JavaScript analyses that computes over-approximations.

An Array Content Static Analysis Based on Non Contiguous Partitions

Jiangchao Liu, CNRS, ENS, INRIA and PSL* Research University

Abstract: Conventional array partitioning analyses split arrays into contiguous partitions to infer properties of sets of cells. Such analyses cannot group together non contiguous cells, even when they have similar properties. In this paper, we propose an abstract domain which utilizes semantic properties to split array cells into groups. Cells with similar properties will be packed into groups and abstracted together. Additionally, groups are not necessarily contiguous. This abstract domain allows to infer complex array invariants in a fully automatic way. Experiments on examples from the Minix 1.1 memory management and a tiny industrial operating system demonstrate the effectiveness of the analysis.

Systematic Approaches for Increasing Soundness and Precision of Static Analyzers

Anders Moeller, Aarhus University (Joint work with Espen Sparre Andreasen and Benjamin Barslev Nielsen.)

Building static analyzers for modern programming languages is difficult. Often soundness is a requirement, perhaps with some well-defined exceptions, and precision must be adequate for producing useful results on realistic input programs. Formally proving such properties of a complex static analysis implementation is rarely an option in practice, which raises the challenge of how to identify causes and importance of soundness and precision problems.

This talk presents our experience with semi-automated methods based on delta debugging and dynamic analysis for increasing soundness and precision of a static analyzer for JavaScript. The individual methods are well known, but to our knowledge rarely used systematically and in combination.

Data-Driven Program Analysis

Hakjoo Oh, Korea University

In this talk, I will present our on-going project on data-driven program analysis. An ideal program analysis should be able to adapt to a given analysis task automatically, and avoid using techniques that unnecessarily improve precision and increase analysis cost. However, building a cost-effective program analysis tool for real-world programs is currently an art; designing adaptation heuristics is done by trials and error, requiring a huge amount of manual effort and expertise. Furthermore, such hand-tuned heuristics are suboptimal and brittle. Our approach to overcome this shortcoming is to combine program analysis and machine learning, where the analysis heuristics are automatically learned from codebases without reliance on analysis designers. Our approach aims to be powerful and stable; the automatically generated heuristics consistently outperform traditional rule-based heuristics. Toward this goal, we are developing machine learning models, efficient learning algorithms, and automated feature-engineering techniques appropriate for the program analysis application. I will talk about the overall approach, current achievements, and remaining challenges.

Revisiting Recency Abstraction for JavaScript: Towards an Intuitive, Compositional, and Efficient Heap Abstraction

Jihyeok Park, KAIST

JavaScript is one of the most widely used programming languages. To understand the behaviors of JavaScript programs and to detect possible errors in them, researchers have developed several static analyzers based on the abstract interpretation framework. However, JavaScript provides various language features that are difficult to analyze statically and precisely such as dynamic addition and removal of object properties, first-class property names, and higher-order functions. To alleviate the problem, JavaScript static analyzers often use recency abstraction, which refines address abstraction by distinguishing recent objects from summaries of old objects. We observed that while recency abstraction enables more precise analysis results by allowing strong updates on recent objects, it is not monotone in the sense that it does not preserve the precision relationship between the underlying address abstraction techniques: for an address abstraction A and a more precise abstraction B, recency abstraction on B may not be more precise than recency abstraction on A. Such an unintuitive semantics of recency abstraction makes its composition with various analysis sensitivity techniques also unintuitive. Thus, we want to propose a new heap abstraction materialized by another criteria instead of the time of object allocations. As an intermediate result, we propose a new singleton abstraction technique, which distinguishes singleton objects to allow strong updates on them without changing a given address abstraction. We formally define recency and singleton abstractions, and explain the unintuitive behaviors of recency abstraction. Our preliminary experiments show promising results for singleton abstraction.

Towards a library of abstract domains to describe memory properties

Xavier Rival, CNRS, ENS, INRIA, PSL* Research University

In this talk, we will present MemCAD, an abstract interpreter that integrates a library of abstract domains to represent properties of memory states. It can describes structures, dynamic structures, and arrays. It can also interact with abstract domains that represent numeric properties and constraints over set symbols. It relies on interfaces that allow adding novel abstract domains, and to define combination operators, to build up advanced abstract domains from basic ones. We will present the structure of the analyzer and highlight the main abstract domains that it implements. We will also highlight the core analysis algorithms, and show a few examples of static analyses using these domains.

Static Analysis of Android Applications for Bug Finding

Sukyoung Ryu, KAIST, organizer

Mobile applications have become prevalent and they introduce new kinds of problems compared to traditional applications. We present a series of our efforts in statically analyzing Android applications to find bugs and vulnerabilities in them. We first describe how the powerful Android Debug Bridge (ADB), a command line tool to communicate with Android devices for debugging purposes, can open a gate to adversaries. To protect Android devices from various attacks using ADB, we present several mitigation mechanisms including a static analysis tool that analyzes Android applications to detect possible attacks using ADB capabilities. Then, we present HybriDroid, a static analysis framework for Android hybrid apps. We investigate the semantics of Android hybrid apps especially for the interoperation mechanism of Android Java and JavaScript. Then, we design and implement a static analysis framework that analyzes inter-communication between Android Java and JavaScript. As example analyses supported by HybriDroid, we implement a bug detector that identifies programmer errors due to the hybrid semantics, and a taint analyzer that finds information leaks cross language boundaries. Our empirical evaluation shows that the tools are practically usable in that they found previously uncovered bugs in real-world Android hybrid apps and possible information leaks via a widely-used advertising platform. Finally, we demonstrate Android activity injection attacks with a simple malware, and formally specify the activity activation mechanism using operational semantics. Based on the operational semantics, we develop a static analysis tool, which analyzes Android apps to detect activity injection attacks. Our tool is fast enough to analyze real-world Android apps in 6 seconds on average, and our experiments found that 1,761 apps out of 129,756 real-world Android apps inject their activities into other apps tasks.

Optimistic JavaScript AOT Compilation

Manuel Serrano, INRIA

JavaScript has escaped web pages. It is now also used for programming web servers, compilers, and other general purpose tasks. There is even a growing trend for using it for programming embedded devices. In this context, JIT compilation is ineffective because it is too memory demanding, and interpretation is too slow for anything else but simplistic tasks. Static compilation, //a.k.a.//, ahead-of-time (AOT) compilation, is an alternative approach that can combine the good speed of JIT compilers and the lightweight memory footprint of interpreters.

We have designed an AOT compiler for full-fledged JavaScript. It relies on a genuine type analysis called //hint typing//. Contrary to most approaches, hint typing does not infer types according to the data structures the program manipulates but according to the best code the compiler is able to generate. In this presentation, we will present this analysis and the overall architecture of the compiler.

Declarative Static Program Analysis: An Intelligent System over Programs

Yannis Smaragdakis, University of Athens

It's the dream of most every programmer: a smart system that "knows more about my code than I do". How do we go about building it? I will argue for the benefits of using logic-based declarative languages as a means to specify static program analysis algorithms. Every aspect of complex program behavior (e.g., regular language features, reflection, exceptions, code generation) is captured by separate logical rules that cooperate to produce a model of what the code does. The result is "holistic" analysis: although every sub-analysis has its own concerns, everything is connected. Concretely, the focus will be on the Doop framework for analysis of Java programs, and especially on its latest developments. Doop encodes multiple analysis algorithms for Java declaratively, using Datalog: a logic-based language for defining (recursive) relations. With an aggressive optimization methodology, Doop also achieves very high performance–often an order of magnitude faster than comparable frameworks.

An Abstract Interpretation Framework for Input Data Usage

Caterina Urban, ETHZ

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.

Effect Summaries for Thread-Modular Analysis (Sound Analysis despite an Unsound Heuristic)

Thomáš Vojnar, Brno University

We propose a novel guess-and-check principle to increase the efficiency of thread-modular verification of lock-free data structures. We build on a heuristic that guesses candidates for stateless effect summaries of programs by searching the code for instances of a copy-and-check programming idiom common in lock-free data structures. These candidate summaries are used to compute the interference among threads in linear time. Since a candidate summary need not be a sound effect summary, we show how to fully automatically check whether the precision of candidate summaries is sufficient. We can thus perform sound verification despite relying on an unsound heuristic. We have implemented our approach and found it up to two orders of magnitude faster than existing ones. The result is a joint work with Lukas Holik (FIT BUT), Roland Meyer (TU Braunschweig), and Sebastian Wolff (TU Braunschweig and Fraunhofer ITWM, Kaiserslautern).

Scalable Global Static Analysis, Automation, and Secrecy

Kwangkeun Yi, Seoul National University

I will talk about three techniques towards our goal of making scalable, semantic-based global static analysis easily available to non-expert software developers. Though static analysis is widely deployed in practice (verification, bug-finding, maintenance, optimizations, and etc.), it is still of limited use. Developing an impactful static analyzer is difficult. Depending on its deployment models, every static analysis needs to strike a different balance between its soundness, scalability, and precision. Our position is that sound and scalable analyzers whose precision is open as a parameter can be automatically available at least for C-like languages. I will first present our general sparse analysis framework to achieve sound, scalable, semanti-based global analysis (to globally analyze million-line C programs in about 10 hours). Given a static analysis definition as a fixpoint computation of an approximate semantics of the input program, the sparse framework guides you how to make it scalable without compromising the analysis precision. Then I will introduce our ZooBerry system to automatically implement this sparse techniques inside static analyzers. From a high-level approximate semantics definition of a C-like language and its soundness Coq proof, ZooBerry automatically generates a sparse static analyzer and its verified validator. Lastly, I will discuss static analysis of encrypted programs, to help sw developers enjoy static analysis service in clouds.

Shape and Content

Florian Zuleger, Technische Universität Wien

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. I will discuss extensions of C2 that can express data structures such as lists and trees. 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.

List of Participants

- Bor-Yuh Evan Chang (CU Boulder, organizer)
- Arlen Cox (Institute for Defense Analysis)
- Julian Dolby (IBM)
- Cezara Drăgoi (CNRS, ENS, INRIA and PSL* Research University)
- Nathan Keynes (Oracle)
- Yoonseok Ko (KAIST)
- Jiangchao Liu (CNRS, ENS, INRIA and PSL* Research University)
- Anders Moeller (Aarhus University)
- Hakjoo Oh (Korea University)
- Jihyeok Park (KAIST)
- Xavier Rival (CNRS, ENS, INRIA and PSL* Research University, organizer)
- Sukyoung Ryu (KAIST, organizer)
- Manuel Serrano (INRIA)
- Yannis Smaragdakis (University of Athens)
- Caterina Urban (ETHZ)
- Thomáš Vojnar (Brno University)
- Kwangkeun Yi (Seoul National University)
- Florian Zuleger (Technische Universität Wien)

Meeting Schedule

Check-in Day: September 10 (Sun)

- 15:00–19:00 Hotel Check In
- 19:00–21:00 Welcome Reception

Day1: September 11 (Mon)

- 7:30–9:00 Breakfast (Cafeteria ”Oak”)
- 9:00–9:10 Introduction Movie of NII Shonan Meetings
- 9:10–9:45 Seminar introduction, and presentation of the members
- 9:45–10:30 Talk: Arlen Cox
- 10:30–11:00 Break
- 11:00–12:00 Talks: Anders Mller / Manuel Serrano
- 12:00–13:30 Lunch (Cafeteria ”Oak”)
- 13:30–14:00 Shooting of the group photo
- 14:00–15:30 Talks: Xavier Rival / Yannis Smaragdakis
- 15:30–16:00 Break
- 16:00–18:00 Talks: Florian Zuleger / Jihyeok Park / Sukyoung Ryu / Bor-Yuh Evan Chang
- 18:00–19:30 Dinner (Cafeteria ”Oak”)

Day2: September 12 (Tue)

- 7:30–9:00 Breakfast (Cafeteria ”Oak”)
- 9:00–10:30 Talks: Kwangkeun Yi / Nathan Keynes
- 10:30–11:00 Break
- 11:00–12:00 Talks: Yoonseok Ko / Jiangchao Liu
- 12:00–13:30 Lunch (Cafeteria ”Oak”)
- 13:30–15:00 Talks: Thomáš Vojnar / Julian Dolby
- 15:00–15:30 Break
- 15:30–17:15 Talks: Caterina Urban / Hakjoo Oh / Cezara Drăgoi
- 17:15–18:00 Discussion: Plan for the Week
- 18:00–19:30 Dinner (Cafeteria ”Oak”)

Day3: September 13 (Wed)

- 7:30–9:00 Breakfast (Cafeteria ”Oak”)

- 9:00–10:30 Talks
- 10:30–11:00 Break
- 11:00–12:00 Talks
- 12:00–13:30 Lunch (Cafeteria ”Oak”)
- 13:30–18:15 Excursion
- 18:15–19:30 Main Banquet

Day4: September 14 (Thu)

- 7:30–9:00 Breakfast (Cafeteria ”Oak”)
- 9:00–10:30 Talks
- 10:30–11:00 Break
- 11:00–12:00 Talks
- 12:00–13:30 Lunch (Cafeteria ”Oak”)
- 13:30 End of the Seminar