ISSN 2186-7437

NII Shonan Meeting Report

No. 2015-13

Semantics and Verification of Object-Oriented Languages

Atsushi Igarashi
Andrzej Murawski
Nikos Tzevelekos

September 21–25, 2015

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

Semantics and Verification of Object-Oriented Languages

Organizers:
Atsushi Igarashi (Kyoto University)
Andrzej Murawski (University of Warwick)
Nikos Tzevelekos (Queen Mary University of London)

September 21–25, 2015

Software applications are increasingly often desired, or even required, to come with guarantees as to their performance, correctness and reliability. Such guarantees have typically been of focal importance to safety-critical applications such as avionics, automotive systems and nuclear reactor controllers. Nowadays, with software being pervasive and having a key role in our daily lifestyles, the focus of the game has greatly broadened. Verified code has become a desideratum of software development in general: it concerns widespread applications written in mainstream languages. The verification problem for such languages poses new challenges as high-level programming features (references, objects, classes, etc.) need to be handled accurately. Moreover, realistic applications can expand to million lines of code, thus demanding accurate but scalable methods. Taking these parameters into account, this seminar is devoted to modelling and verifying object-oriented code. The aim of the meeting is to take stock of the present situation, to foster communication between researchers pursuing diverse approaches and to propose challenges for the future.

The object-oriented paradigm has been embraced by mainstream software companies as well as academic curricula. It has also been gaining ground in verification. Although, traditionally, verification has been conducted for constrained but technically robust programming languages, time has shown that there is a need to loosen the constraints in order to make comprehensive impact. Everyday software development primarily concerns mainstream languages, which provide a range of expressive features, greatly enhancing the development and maintenance of code. Moreover, they also attract a good supply of skilful developers. On the object-oriented front, Java has become widespread and the trend is complemented by the development of C# and Scala. Each of them offers safety compile-time and run-time checks, which are not present in other competitors like C/C++. They emerge as the natural choices to target with modelling and verification techniques, if one is interested in the analysis of safe object-oriented programs. Many efforts have been dedicated in recent years to provide techniques to address the issue. The vision of dependable/verified software has become the driving force behind numerous research projects in the UK and inspired one of the official UK CRC Grand Challenges for the next decade. The formal verification of object-oriented software has also been the

subject of international initiatives, such as the European Concerted Research Action IC0701.

Object-oriented code presents a whole variety of interesting technical challenges, stemming from the fact that it involves a unique combination of more general computational concepts, such as recursive types, higher-order references, self-reference, state encapsulation, polymorphism etc. Each of them is a challenge in its own right and over the years dedicated research programmes have been developed to understand the features in isolation. This seminar provides a forum in which such neighbouring fields can be discussed from the point of view of object-oriented languages. We hope to identify opportunities for transfers of techniques with a view to specializing them to the specific setting of object-oriented code. As well as assessing the more established lines of work in the area, we also aim to predict new fruitful directions, emerging from the constant evolution of OO-languages, the addition of new features, capabilities and structuring constructs.

Overview of Talks

The Viper Project: Verification Infrastructure for Permission-based Reasoning

Alex Summers, ETH Zürich

Modern verification techniques are becoming ever-more powerful and sophisticated, and building tools to implement them is a time-consuming and difficult task. Writing a new verifier to validate each on-paper approach is impractical; for this reason intermediate verification languages such as Boogie and Why3 have become popular over the last decade. However, verification approaches geared around complex program logics (such as separation logic) have typically been implemented in specialised tools, since the reasoning is hard to map down to first-order automated reasoning. In practice, this means that a rich variety of modern techniques have no corresponding tool support.

In this talk, I will present the new Silver intermediate verification language, which has been designed to facilitate the lightweight implementation of a variety of modern methodologies for program verification. In contrast to lower-level verification languages, Silver provides native support for heap reasoning; modes of reasoning such as concurrent separation logic, dynamic frames and rely-guarantee/invariants can be simply encoded.

Silver has been developed as part of the Viper project, which provides two automated back-end verifiers for Silver programs. Since releasing our software in September last year, it has been used for (internal and external) projects to build tools for Java verification, non-blocking concurrency reasoning, flow-sensitive typing and reasoning about GPU and Linux kernel code.

Compositional Reachablility in Petri Nets

Julian Rathke, University of Southampton

We consider a compositional approach to deciding reachability for 1-bounded Petri Nets. We introduce a novel notion of Petri Nets with boundaries in order to

capture the requirements on interactions between the composed subnets. This approach allows for a new divide and conquer style algorithm for the problem. We also describe a tool based implementation of the approach and compare its performance against other state-of-the-art reachability checkers.

Traits and dynamic frames

Rustan Leino, Microsoft Research

Dafny is a verification-aware programming language [1]. It includes instantiable classes that can be reasoned about using the *dynamic frames* specification idiom. Previously, there was no inheritance among classes, but a recent addition to the language added *traits* [2], which are abstract subperclasses with fields, functions, and methods. In this talk, I describe the traits in Dafny. I then give a tutorial on using dynamic frames and show how these are used with traits. I also mention some future work and bring up some questions I have about the design.

[1] K. Rustan M. Leino. Dafny: An automatic program verier for functional correctness. In LPAR-16, volume 6355 of LNAI, pages 348370. Springer, April 2010.

[2] Reza Ahmadi, K. Rustan M. Leino, Jyrki Nummenmaa. Automatic verification of Dafny programs with traits. FTfJP@ECOOP 2015: 4:1-4:5. July 2015.

Wyvern Formalisation

Alex Potanin, Victoria University of wellington

Wyvern is a new object-oriented programming language developed at CMU and VUW targeting web and mobile app developers concerned about security, safety, and productivity. In this talk, I will go over our current state of formalisation of different aspects of Wyvern starting with a simple core (extended Lambda Calculus) and moving through more complex stages adding objects, classes, and modules that translate back to the simpler cores. Finally, I will address the current major issues we are experiencing formalising Wyvern module system and security guarantees it can provide as well as formalising type members (as found in Scala or Beta) as we add them to Wyvern.

Reasoning about Class Behavior using the Bisimulation Technique

Vasileios Koutavas, Trinity College Dublin

In this talk we will discuss a proof technique for reasoning about contextual equivalence of classes in an imperative subset of Java. The technique is based on environmental bisimulations and is sound and complete with respect to contextual equivalence. We will see how inheritance and downcasting affect the technique, and explore example equivalent and inequivalent classes.

Deciding contextual equivalence for IMJ*

Steven Ramsay, University of Warwick

I will describe joint work with Andrzej Murawski and Nikos Tzevelekos in which we consider the problem of deciding observational equivalence for IMJ*. IMJ* is a kind of maximal fragment of the language Interface Middleweight Java (IMJ) for which the problem is decidable. Given two, possibly open (containing free identiers), terms of IMJ*, the contextual equivalence problem asks if the terms can be distinguished by any possible IMJ context. Our decision procedure reduces this problem to the emptiness problem for fresh-register pushdown automata. I will demonstrate our implementation of the procedure in the tool Coneqct.

Game Semantics for Interface Middleweight Java

Nikos Tzevelekos, Queen Mary University of London

In this talk we present a denotational/operational model that captures the behaviour of programs written in an object calculus called IMJ and in which open terms interact with the environment through interfaces. The calculus is intended to represent the essence of contextual interactions of Middleweight Java code. The model is based on game semantics, a technique which models interactions between a program and its environment as formal games. Working with games, we build fully abstract models for the induced notions of contextual approximation and equivalence.

This is joint work with A. Murawski.

Type Systems for Dynamic Layer Composition

Atsushi Igarashi, Kyoto University

Key features of context-oriented programming (COP) are *layers* – modules to describe context-dependent behavioral variations of a software system – and their *dynamic activation*, which can modify the behavior of multiple objects that have already been instantiated. Typechecking programs written in a COP language is difficult because the activation of a layer can even change objects interfaces.

In this talk we present a few type systems to deal with such dynamic layer compositions. Starting with a very simple but restrictive type system, we gradually add language features and extend the type system accordingly.

This is a joint work with Hiroaki Inoue, Robert Hirschfeld, and Hidehiko Masuhara.

A semantics for context-oriented programming languages with multiple layer activation mechanisms

Tomoyuki Aotani, Tokyo Tech

Context-oriented programming (COP) is an approach to modularize behavioral variations of programs into layers from the viewpoint of contexts that

change dynamically during the execution of a program. Changing behavior with respect to changes of contexts is achieved by activating or deactivating layers and dispatching methods according to not only the dynamic type of the receiver object but also the set of active layers. Several layer-activation mechanisms have been proposed separately for COP languages. In this talk we propose new semantics using monads and comonads for COP languages supporting multiple layer-activation mechanisms.

Type soundness proofs with big-step operational semantics of object-oriented languages

Davide Ancona, University of Genoa

Proofs of soundness of type systems are typically based on small-step operational semantics, because of the inability of big-step operational semantics to distinguish stuck from non-terminating computations.

In this talk I will present two techniques to prove type soundness with big-step semantics.

The former is based on the notion of coinductive big-step semantics, obtained by interpreting the semantics rules coinductively, and by considering non-well-founded values. The corresponding proof of soundness is rather involved, since it requires showing that the set of proof trees defining the semantic judgment forms a complete metric space when equipped with a specific distance function.

A simpler approach consists in approximating at arbitrary depth the possibly infinite derivations of the coinductive semantics with infinite sequences of finite derivations; in this way, type soundness can be proved by standard mathematical induction.

Finally, I will briefly discuss interesting research directions to overcome some of the limitations of the presented techniques.

Predicate Refinement Heuristics in Program Verification with CEGAR

Tachio Terauchi, JAIST

Many of the modern program verifiers are based on the predicate abstraction with CEGAR method, The method looks for a sufficient inductive invariant to prove the given property of the given program by iteratively accumulating predicates that are obtained by analyzing spurious counterexamples, and predicate refinement heuristics like Craig interpolation are used for this purpose. In this talk, we overview our recent work on predicate refinement heuristics in CEGAR. Specifically, we show that a certain strategy for choosing predicates can guarantee the convergence of CEGAR at a modest cost. We also show that choosing small refinements guarantees fast convergence under certain conditions.

State machine learning: from testing to formal specifications

Erik Poll, Radboud University Nijmegen

Finite state machines are a great specification formalism: they are intuitive

and simple to understand, and can be used to describe all sorts of systems, or properties of systems, that we want to verify.

Coming up with accurate state machine models can be a lot of work. However, state machine inference provides a great technique to automatically reverse-engineer application using just black-box testing. This can provide formal models of existing code for free, as a useful first step to rigorous specification and then verification of this code. We will show this using case studies of bank cards, internet banking tokens, and TLS implementations.

Behavioral Subtyping, Specification Inheritance, and Modular Reasoning

Gary Leavens, University of Central Florida

This talk aims to characterize modular reasoning for object-oriented (OO) programs that use subtyping and dynamic dispatch. It gives a semantic characterization of supertype abstraction using two different semantics for a Java-like OO language. The first semantics is an accurate semantics of the languages behavior. The second semantics mimics supertype abstraction, using static type information and method specifications to determine what a program should do. The validity of supertype abstraction is defined as having this second semantics accurately predict the behavior of programs with the dynamic semantics.

This talk is based on joint work with David A. Naumann of Stevens Institute of Technology, and is based on our paper "Behavioral Subtyping, Specification Inheritance, and Modular Reasoning" published in ACM TOPLAS, 37(4):13:1-13:88, Aug 2015. The work of Gary Leavens was supported by NSF grants CNS-0808913, CCF-0916350, CCF-0916715, CCF-1017262, and CNS-1228695 David Naumanns work was supported in part by NSF grants CCF-0915611 and CNS-1228930.

Cause Im Strong Enough: Reasoning about Consistency Choices in Distributed Systems

Hongseok Yang, University of Oxford

Large-scale distributed systems often rely on replicated databases that allow a programmer to request different data consistency guarantees for different operations, and thereby control their performance. Using such databases is far from trivial: requesting stronger consistency in too many places may hurt performance, and requesting it in too few places may violate correctness. To help programmers in this task, we propose the first proof rule for establishing that a particular choice of consistency guarantees for various operations on a replicated database is enough to ensure the preservation of a given data integrity invariant. Our rule is modular: it allows reasoning about the behaviour of every operation separately under some assumption on the behaviour of other operations. This leads to simple reasoning, which we have automated in an SMT-based tool. We present a nontrivial proof of soundness of our rule and illustrate its use on several examples.

This is joint work with Alexey Gotsman (IMDEA, Spain), Carla Ferreira (Universidade Nova Lisboa), Mahsa Najafzadeh (UPMC & INRIA), and Marc Shapiro (UPMC & INRIA).

Analyzing JavaScript Web Applications in the Wild (Mostly) Statically

Sukyoung Ryu, KAIST

Analyzing real-world JavaScript web applications is a challenging task. On top of understanding the semantics of JavaScript, it requires modeling of web documents, platform objects, and interactions between them. Not only JavaScript itself but also its usage patterns are extremely dynamic. Most of web applications load JavaScript code dynamically, which makes pure static analysis approaches inapplicable. We present our attempts to analyze JavaScript web applications in the wild mostly statically using various approaches to analyze libraries.

A Fully Verified Container Library

Nadia Polikarpova, MIT

The comprehensive functionality and flexible design of general-purpose container libraries pose nontrivial challenges to formal verification. In this talk I will present our experience using the AutoProof program verifier to prove full functional correctness of a realistic container library. I will discuss the key ingredients of AutoProofs methodology, which made the verification possible with moderate annotation overhead and good performance.

Tutorial: Preventing Errors Before They Happen

Werner Dietl, University of Waterloo

Are you tired of null pointer exceptions, unintended side effects, SQL injections, concurrency errors, mistaken equality tests, and other runtime errors? Are your users tired of them in your code? This presentation shows you how to guarantee, at compile time, that these runtime exceptions cannot occur. You have nothing to lose but your bugs!

Automata-Based Abstraction Refinement for muHORS Model Checking

Xin Li, University of Tokyo

The model checking of higher-order recursion schemes (HORS), aka. higher-order model checking, is the problem of checking whether the tree generated by a given HORS satisfies a given property. It has recently been studied actively and applied to automated verification of higher-order programs. Kobayashi and Igarashi studied an extension of higher-order model checking called HORS model checking, where HORS has been extended with recursive types, so that a

wider range of programs, including object-oriented programs and multi-threaded programs, can be precisely modeled and verified. Although the HORS model checking is undecidable in general, they developed a sound but incomplete procedure for HORS model checking. Unfortunately, however, their procedure was not scalable enough. Inspired by recent progress of (ordinary) HORS model checking, we propose a new procedure for HORS model checking, based on automata-based abstraction refinement. We have implemented the new procedure and confirmed that it often outperforms the previous procedure.

Verification of Featherweight Java Programs via Transformation to Higher-order Functional Programs with Recursive Data Types

Hiroshi Unno, University of Tsukuba

In this talk, I will present a verification method for object-oriented programs in Featherweight Java (FJ) based on a transformation to functional programs in ML. The transformation encodes objects and dynamic method dispatching in FJ using higher-order functions and recursive data types in ML. The transformed programs are then verified by using the state-of-the-art techniques for ML programs such as refinement types, Horn clause solving, and higher-order model checking. In the talk, I will explain one of the techniques called refinement type optimization which we have proposed in SAS 2015.

Automating Separation Logic using SMT

Thomas Wies, NYU

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.

Collaborative Verification of the Information Flow for a High-Assurance App Store

Werner Dietl, University of Waterloo

Malware is a serious problem on mobile devices. The vision of this sessions speakers was a verified app store in which each application has been formally proven to be free of (certain) defects and exploits. They have built such a system and successfully applied it to dozens of challenge applications created by hostile red teams. The session describes their type system for information flow along with support for implicit invocation (intents and reflection), varieties of polymorphism, and other challenges that have arisen. Based on my CCS 2014 and ASE 2015 papers.

List of Participants

- Davide Ancona, University of Genoa
- Tomoyuki Aotani, Tokyo Tech
- Werner Dietl, University of Waterloo
- Atsushi Igarashi, Kyoto University
- Vasileios Koutavas, Trinity College Dublin
- Gary T. Leavens, University of Central Florida
- Rustan Leino, Microsoft Research
- Xin Li, University of Tokyo
- Nadia Polikarpova, MIT
- Erik Poll, Radboud University Nijmegen
- Alex Potanin, Victoria University of wellington
- Steven Ramsay, University of Warwick
- Julian Rathke, University of Southampton
- Sukyoung Ryu, KAIST
- Alex Summers, ETH Zürich
- Tachio Terauchi, JAIST
- Nikos Tzevelekos, Queen Mary University of London
- Hiroshi Unno, University of Tsukuba
- Thomas Wies, NYU
- Hongseok Yang, University of Oxford

Meeting Schedule

20th September, 2015

- 19:00-21:00: Welcome Reception

21st September, 2015

- 7:30-9:00: Breakfast
- 9:00-10:30: Technical Session

  Self-introduction by participants
- 10:30-11:00: Break
- 11:00-11:45: Technical Session

  Werner Dietl: Collaborative Verification of the Information Flow for a High-Assurance App Store
- 12:00-13:30: Lunch
- 13:30-14:00: Group Photo Shooting
- 14:00-15:30: Technical Session

  Thomas Wies: Automating Separation Logic using SMT
- 15:00-15:30: Break
- 15:30-17:00: Technical Session

  Hiroshi Unno: Verification of Featherweight Java Programs via Transformation to Higher-order Functional Programs with Recursive Data Types

  Xin Li: Automata-Based Abstraction Refinement for muHORS Model
- 18:00-19:30: Dinner

22nd September, 2015

- 7:30-9:00: Breakfast
- 9:00-10:00: Technical Session

  Werner Dietl: Tutorial on the Checker framework
- 10:00-10:30: Break
- 10:30-12:00: Technical Session

  Nadia Polikarpova: A Fully Verified Container Library

  Sukyoung Ryu: Analyzing JavaScript Web Applications in the Wild (Mostly) Statically
- 12:00-13:30: Lunch

- 13:30-15:00: Technical Session

  Hongseok Yang: Cause Im Strong Enough': Reasoning about Consistency Choices in Distributed Systems

  Gary Leavens: Modular Reasoning and a Definition of Supertype Abstraction

- 15:00-15:30: Break

- 15:30-17:00: Technical Session

  Erik Poll: State machine learning: from testing to formal specifications

  Tachio Terauchi: On predicate refinement heuristics used in CEGAR

- 18:00-19:30: Dinner

23rd September, 2015

- 7:30-9:00: Breakfast

- 9:00-10:00: Technical Session

  Davide Ancona: Type soundness proofs with big-step operational semantics of object-oriented languages

- 10:00-10:30: Break

- 10:30-12:00: Technical Session

  Atsushi Igarashi: Type Systems for Dynamic Layer Composition

  Tomoyuki Aotani: A semantics for context-oriented programming languages with multiple layer activation mechanisms

- 12:00-13:30: Lunch

- 13:30-15:00: Technical Session

  Nikos Tzevelekos: Game Semantics for Interface Middleweight Java

  Steven Ramsay: Deciding contextual equivalence for IMJ*

- 15:00-15:30: Break

- 15:30-16:15: Technical Session Vassilis Koutavas: Reasoning about Class Behavior using the Bisimulation Technique

- 18:00-19:30: Dinner

24th September, 2015

- 7:30-9:00: Breakfast

- 9:00-10:00: Technical Session

  Alex Summers: The Viper Project: Verification Infrastructure for Permission-based Reasoning

- 10:00-10:30: Break

- 10:30-12:00: Technical Session

  Alex Potanin: Wyvern Formalisation: Objects, Classes, Modules, and Type Members

  Rustan Leino: Traits and dynamic frames

- 12:00-13:30: Lunch

- 13:30- Excursion to Kamakura and Banquet

25th September, 2015

- 7:30-9:00: Breakfast

- 9:00-10:00: Technical Session

  Julian Rathke: Compositional Reachablility in Petri Nets

- 10:00-10:30: Break

- 10:30-12:00: Technical Session

  Alex Summers, Thomas Wies, Nadia Polikarpova, and Rustan Leino: Methods for Verifying the Heap

- 12:00-13:30: Lunch