10351 Abstracts Collection
Modelling, Controlling and Reasoning About State
— Dagstuhl Seminar —

Amal Ahmed[1], Nick Benton[2], Lars Birkedal[3] and Martin Hofmann[4]

[1] Indiana University - Bloomington, US
amal@cs.indiana.edu
[2] Microsoft Research UK - Cambridge, GB
nick@microsoft.com
[3] IT University of Copenhagen, DK
birkedal@itu.dk
[4] LMU München, DE
mhofmann@informatik.uni-muenchen.de

Abstract. From 29 August 2010 to 3 September 2010, the Dagstuhl Seminar 10351 "Modelling, Controlling and Reasoning About State " was held in Schloss Dagstuhl – Leibniz Center for Informatics. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. Links to extended abstracts or full papers are provided, if available.

Keywords. Mutable State, Program Logics, Semantics, Type Systems, Verification

10351 Executive Summary

Amal Ahmed (Indiana University, US), Nick Benton (Microsoft Research, GB), Lars Birkedal (IT University of Copenhagen, DK) and Martin Hofmann (LMU München, DE)

From 29 August 2010 to 3 September 2010, the Dagstuhl Seminar 10351 "Modelling, Controlling and Reasoning About State " was held in Schloss Dagstuhl – Leibniz Center for Informatics. 44 researchers, with interests and expertise in many different aspects of modelling and reasoning about mutable state, met to present their current work and discuss ongoing projects and open problems. This summary provides a general overview of the goals of the seminar and of the topics discussed.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2810

Representing Binding Using Parametricity

Robert Atkey (The University of Strathclyde - Glasgow, GB)

I will talk about using parametric polymorphism with Kripke logical relations to represent binding. This seems to be related to the type based method used to make the Haskell ST monad for isolated local state safe.

Full Paper:
http://personal.cis.strath.ac.uk/~raa/parametricity/syntaxforfree.pdf

Step-Indexing: The Good, the Bad and the Ugly

Nick Benton (Microsoft Research UK - Cambridge, GB)

Over the last decade, step-indices have been widely used for the construction of operationally-based logical relations in the presence of various kinds of recursion. We first give an argument that step-indices, or something like them, seem to be required for defining realizability relations between high-level source languages and low-level targets, in the case that the low-level allows egregiously intensional operations such as reflection or comparison of code pointers. We then show how, much to our annoyance, step-indices also seem to prevent us from exploiting such operations as aggressively as we would like in proving program transformations.

Keywords: Step-Indexing, Logical Relations, Low-Level Languages, Compiler Correctness

Joint work of: Benton, Nick; Hur, Chung-Kil

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2808

Kripke Models over Recursive Worlds (The Joy of Ultrametric Spaces)

Lars Birkedal (IT University of Copenhagen, DK)

Over the last decade, there has been extensive research on modelling challenging features in programming languages and program logics, such as higher-order store and storable resource invariants. A recent line of work has identified a common solution to some of these challenges:

Kripke models over worlds that are recursively defined in a category of metric spaces. In this talk I give an overview of this approach, which applies both to denotational and operational models of programming languages.

Keywords: Semantics of Higher-Order Store, Kripke Models, Recursive Worlds

Verification of Imperative Programs Through Characteristic Formulae

Arthur Chargueraud (INRIA - Le Chesnay, FR)

I have developed and implemented a new approach to reasoning on imperative programs, based on characteristic formulae. Given a Caml program, I generate a Coq formula that accurately describes the behaviour of that program. This formula is stated only in terms of the values and of the basic connectives from higher-order logic; it does not refer to the syntax of Caml source code (like a deep embedding would do) and it does not rely on Coq functions to represent Caml functions (like a shallow embedding would do). The specification of programs is done through statement of Coq lemmas, in which heaps are descibed using Separation Logic style predicates similar to those used in Ynot. The verification of a program is conducted through an interactive proof following the structure of the characteristic formula. One key feature of characteristic formulae is that they are of linear size and that they can be pretty-printed just like the source code they describe. In this talk, I will explain how characteristic formulae are generated and I will describe the verification of imperative functions such as in-place list reversal, map function on lists, CPS list append, and Landin's knot.

Cost-Effective and Foundational Verification of Low-Level Code

Adam Chlipala (Harvard University, US)

Several recent projects have involved machine-checked proofs of correctness for programs in assembly language or other not-much-higher abstraction levels. "Machine-checked proofs" can mean many things.

Projects that rely on automated theorem-provers typically lead to the least programmer effort per unit of assurance. However, automated tools tend not to do well with higher-order goals, such as reasoning about first-class code pointers or proofs of type system soundness.

The more the automated prover can do, the more reason there is to worry that prover has a critical soundness bug. In contrast, other recent work has used interactive proof assistants, such that program verifications may be trusted without believing in the correctness of much more than a small, generic proof checker and some basic formalization of language semantics. The programmer overhead of these systems has been shown to be rather higher than with automated tools, sometimes at the level of hundreds of lines of proof per program instruction.

The Bedrock framework is a new library for Coq that combines some of the characteristic advantages of the two prior approaches.

Higher-order reasoning is readily supported, via Coq's usual mechanisms; and the trusted code base for a program verification depends only on the Coq core and simple operational semantics for machine languages. At the same time, structured programming is supported, in the style of traditional reasoning with axiomatic semantics; and proofs can be automated about as effectively as in recent (non-foundational) work in that style. We have verified a number of example programs so far, including one (list append in CPS with explicit closures) that requires orders of magnitude less proof code than in past work which posed it as a challenge problem.

Keywords: Verification, Coq

Oracular Environments for Compiler Correctness

Robert Dockins (Princeton University, US)

The CompCert verified compier is a well-known project which formally verifies the correctness of a compiler for a C-like language. Correctness is proved with respect to the observable behavior of programs, which is defined to be the sequence of external function calls made be the programs; such calls are restricted to have arguments and return values of atomic types. However, this model of observables is too weak to reason about realistic environments (e.g., POSIX system calls) where communication with the environment may occur through memory.

Here we present preliminary work on "oracular environments", a way to make richer observations of the runtime behavior of a program while still retaining the overall operational structure of the CompCert proofs. They allow us to reason about more sophisticated environments and, with a little ingenuity, make few changes to the existing CompCert proofs.

Keywords: Compiler Correctness, Oracles

The Art of the State in Logical Relations

Derek Dreyer (MPI for Software Systems - Saarbrücken, DE)

Kripke logical relations are a powerful method for reasoning about observational equivalence of higher-order state-manipulating programs.

In this talk, I will present an overview of some of our recent work on developing Kripke logical relations for realistic "ML-like" languages—languages that support recursive types, abstract types, higher-order state, continuations and/or exceptions. One of the central ideas in our work is that, when reasoning about local state, establishing *invariants* on the local state is not enough; rather, one must be able to establish properties about local state that *change* (in a controlled way) over time. Thus, the possible worlds that we use to index our Kripke relations are essentially state transition systems (STS's), with each state corresponding potentially to a different relation on heaps. I will explain how the use of state transition systems sheds light on the interaction of local state with various other features, such as higher-order functions, abstract types, and control operators.

Joint work of: Ahmed, Amal; Birkedal, Lars; Dreyer, Derek; Neis, Georg; Rossberg, Andreas

Abstraction and Refinement in Local Reasoning

Philippa Gardner (Imperial College London, GB)

Local reasoning has become a well-established technique in program verification, which has been shown to be useful at many different levels of abstraction. In separation logic, we use a low-level abstraction that is close to how the machine sees the program state. In context logic, we work with high-level abstractions that are close to how the clients of modules see the program state. We apply program refinement to local reasoning, demonstrating that high-level local reasoning is sound for module implementations. We consider two approaches: one that preserves the high-level locality at the low level; and one that breaks the high-level 'fiction' of locality.

Joint work of: Dinsdale-Young, Thomas; Gardner, Philippa; Wheelhouse, Mark

A Theory of Termination via Indirection

Aquinas Hobor (National University of Singapore, SG)

Step-indexed models provide approximations to a class of domain equations and can prove type safety, partial correctness, and program equivalence; however, a common misconception is that they are inapplicable to liveness problems. We disprove this by applying step-indexing to develop the first Hoare logic of total correctness for a language with function pointers and semantic assertions. In fact, from a liveness perspective, our logic is stronger: we verify explicit time resource bounds. We apply our logic to examples containing nontrivial "higher-order" uses of function pointers and we prove soundness with respect to a standard operational semantics. Our core technique is very compact and may be applicable to other liveness problems. Our results are machine checked in Coq.

Joint work of: Dockins, Robert; Hobor, Aquinas

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2805

A Kripke Logical Relation Between ML and Assembly

Chung-Kil Hur (University Paris-Diderot, FR)

There has recently been great progress in proving the correctness of compilers for increasingly realistic languages with increasingly realistic runtime systems.

Most work on this problem has focused on proving the correctness of a particular compiler, leaving open the question of how to verify the correctness of assembly code that is hand-optimized or linked together from the output of multiple compilers. This has led Benton and other researchers to propose more abstract, compositional notions of when a low-level program correctly realizes a high-level one. However, the state of the art in so-called "compositional compiler correctness" has only considered relatively simple high-level and low-level languages.

In this paper, we propose a novel, extensional, compiler-independent notion of equivalence between high-level programs in an expressive, impure ML-like λ-calculus and low-level programs in an (only slightly) idealized assembly language. We define this equivalence by means of a biorthogonal, step-indexed, Kripke logical relation, which enables us to reason quite flexibly about assembly code that uses local state in a different manner than the high-level code it implements (e.g. self-modifying code). In contrast to prior work, we factor our relation in a symmetric, language-generic fashion, which helps to simplify and clarify the formal presentation, and we also show how to account for the presence of a garbage collector. Our approach relies on recent developments in Kripke logical relations for ML-like languages, in particular the idea of possible worlds as state transition systems.

Keywords: Logical Relation, Compositional Compiler Correctness

Analysing Call-By-Need

Steffen Jost (University of St Andrews, GB)

We want to automatically and statically analyse the worst-case resource usage of a higher-order functional language with explicit deallocation primitives, that is evaluated under a call-by-need strategy. Thus, expressions are first stored in memory and only evaluated to weak-head-normal-form when needed, but the memory is constantly updated with the obtained weak-head-normal-form, thereby avoiding the cost to reevaluate an aliased thunk.

We employ our tried and tested amortised analysis, that abstracts the machine state to a single number, against which costs must be amortised. Thunks are simply treated as use-once functions without arguments.

This talk is about a small problem that arises around thunks which return more resources than they had asked for.

Keywords: Amortised Analysis, Lazy Evaluation, Functional Programming, Call By Need, Program Analysis

Domain-Theoretic Semantics in Coq

Andrew Kennedy (Microsoft Research UK - Cambridge, GB)

We've recently developed a Coq library for doing domain-theoretic semantics, based on earlier work by Paulin-Mohring. Using this library we have formalized the denotational semantics of a typed PCF-like language and of the untyped lambda calculus, proving soundness and adequacy theorems. The mechanization of domains is purely constructive, with lubs represented by Coq functions, and with lifting making use of a coinductive stream type and a cunning corecursive "search" for the lub.

Joint work of: Kennedy, Andrew; Benton, Nick; Varming, Carsten

Full Paper:
http://research.microsoft.com/en-us/um/people/akenn/coq/Domains.pdf

Communicating Transactions

Vasileios Koutavas (Trinity College Dublin, IE)

In this talk I will present a new language construct called communicating transactions, obtained by dropping the isolation requirement from classical transactions, which can be used to model automatic error recovery in distributed systems. I will show a simple reduction semantics of TransCCS, an extension of CCS with this construct, and examine the properties of simple systems. I will also discuss a behavioural theory of traces for TransCCS that is sound and complete with respect to the may-testing preorder, where the atomicity property of communicating transactions is encoded as non-prefix-closure of trace sets in the theory.

Keywords: Concurrency, CCS, Transactions, Non-isolated Transactions

Joint work of: de Vries, Edsko; Koutavas, Vasilios; Hennessy, Matthew

Full Paper:
http://www.scss.tcd.ie/Vasileios.Koutavas/publications/transccs-concur10.pdf

See also: Communicating Transactions, CONCUR 2010

Limitations of Applicative Bisimulation (Preliminary Report)

Vasileios Koutavas (Trinity College Dublin, IE) and Paul Blain Levy (University of Birmingham, UK) and Eijiro Sumii (Tohoku University, JP)

We present a series of examples that illuminate an important aspect of the semantics of higher-order functions with local state.

Namely that certain behaviour of such functions can only be observed by providing them with arguments that contain the functions themselves. This provides evidence for the necessity of complex conditions for functions in modern semantics for state, such as logical relations and Kripke-like bisimulations, where related functions are applied to related arguments (that may contain the functions). It also suggests that simpler semantics, such as those based on applicative bisimulations where functions are applied to identical arguments, would not scale to higher-order languages with local state.

Keywords: Imperative Languages, Higher-Order Functions, Local State

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2807

Proving GUIs correct

Neel Krishnaswami (Microsoft Research UK - Cambridge, GB)

GUIs combine some of the nastiest problems for program verification, for two reasons. On the one hand, the programs make heavy use of state, higher-order code, and concurrent execution. On the other, even if these features were somehow rendered easy to deal with, we can't get started on correctness proof without answering the question of what the specification is!

However, if were able to answer these questions, we could write cool demos.

Motivated by this prospect, we have studied the question of verifying interactive programs, and (a) discovered a beautiful abstract semantics for them in terms of ultrametric spaces, and (b) proved the correctness of a realistic implementation in terms of callbacks and event loops. Our proof calls for nearly the whole toolkit of modern semantics: type theory, denotational models, logical relations, step-indexing, rely-guarantee and separation logic all play a role.

We cannot claim to have a simple proof, but we can claim that the technology exists to verify even very sophisticated programs.

Keywords: Verification, Semantics, Ultrametric Spaces, Separation Logic

Joint work of: Krishnaswami, Neel; Benton, Nick

Operational game semantics

Paul Blain Levy (University of Birmingham, GB)

In many presentations of game semantics, it is explained informally, using examples, that the strategy denoted by a term describes its behaviour. Yet this intuition is not captured in the formal definition - except for ground terms, where an adequacy result says that the operational and denotational semantics agree.

In this talk we show how to give a transition system on terms, defined using the operational semantics. Then the behaviour of each term is its trace set. The compositionality of this semantics is then a theorem, rather than a definition.

Keywords: Transition System, Game Semantics, Continuations, Compositionality

Joint work of: Laird, Jim; Lassen, Soren; Levy, Paul Blain

An algebraic investigation of local stores

Paul-Andre Mellies (University Paris-Diderot, FR)

A few years ago, Gordon Plotkin and John Power described a local state monad on the category of presheaves over finite sets and injections, and provided a nice and intuitive algebraic presentation for it. In this talk, I will explain (1) how to recast this work in the diagrammatic language of string diagrams and (2) how to think of this local state monad as a model of a particular notion of nominal theory — this providing a natural extension of the notion of Lawvere theory for a finitary monad.

Keywords: Local State Monad, String Siagrams, Algebraic Theories

Call-by-value Games and Automated Equivalence Checking

Andrzej Murawski (University of Oxford, GB)

After a brief introduction to call-by-value game semantics, I will show how one can prove some "awkward" equivalences discussed in previous talks using game models.

This will be followed by a survey of decidability/undecidability results for program equivalence in a finitary ML-like language with ground-type references.

Verifying Liveness Properties of Higher-Order Programs

Chih-Hao Luke Ong (University of Oxford, GB)

We consider the problem of verifying liveness properties of higher-order recursive programs that manipulate stateful objects such as files, memory cells and locks.

The desired correctness properties are described in temporal logic (e.g. CTL or CTL*) or appropriate automata. Building on and extending Kobayashi's work, we show that the problem of verifying all modal mu-calculus properties is sound, complete and automatic for a simply-typed lambda calculus with recursion and primitives for dynamic resource creation and access; this is achieved by reduction to the model checking problem for higher-order recursion schemes (HORS). Time permitting, we will briefly describe a type-based model checking algorithm and implementation of a prototype CTL (equivalently alternating weak tree automata) model checker for HORS.

Keywords: Verification of Liveness Properties, Model Checking, Resource Usage Analysis, Functional Programs, Modal mu-calculus, Higher-order Recursion Schemes

Semantics of Scope

Andrew M. Pitts (University of Cambridge, GB)

This talk concerns the operational semantics of programming constructs involving locally scoped names. Typically such semantics involves stateful *dynamic allocation*: a set of currently-used names forms part of the state and upon entering a scope the set is augmented by a new name bound to the the scoped identifier. (Deallocation of names upon leaving a scope is sometimes appropriate too.) More abstractly, one can see this as a transformation of local scopes by expanding them outward to a(n implicit) top-level.

By contrast, in a neglected paper from POPL 1994, Martin Odersky gave a stateless lambda calculus with locally scoped names whose dynamics contracts scopes inward. The properties of "Odersky-style" local names are quite different from dynamically allocated ones. The former enjoy much nicer equational properties up to contextual equivalence than the latter. However, it is not so clear, until now, what is the expressive power of Odersky's system.

Here we show that Odersky-style local names provide a direct semantics of scope from which the dynamic-allocation semantics of scope can be obtained by continuation-passing translation. More precisely, we show that there is a cps translation of typed lambda calculus with dynamically allocated names (the Pitts-Stark nu-calculus) into Odersky's $\lambda\nu$-calculus that is computationally adequate with respect to contextual equivalence in the two calculi.

Keywords: Local Names, Dynamic Allocation, Continuations, Contextual Equivalence

Joint work of: Pitts, Andrew M.; Loesch, Steffen

Step-Indexed Biorthogonality: a Tutorial Example

Andrew M. Pitts (University of Cambridge, GB)

The purpose of this note is to illustrate the use of step-indexing combined with biorthogonality to construct syntactical logical relations. It walks through the details of a syntactically simple, yet non-trivial example: a proof of the "CIU Theorem" for contextual equivalence in the untyped call-by-value λ-calculus with recursively defined functions.

Keywords: Biorthogonality, Logical Relations, Operational Semantics, Step-Indexing

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2806

Proving the Concurrent Garbage Collector

Uday Reddy (University of Birmingham, GB)

One of the first challenging concurrent programs whose correctness proof was attempted is that of a concurrent garbage collector (called "on-the-fly" garbage collector). The algorithm as well as its correctness proof was formulated by Dijkstra et al. in the period 1975-78, and a simpler proof using the Owicki-Gries technique was given by Gries. In this talk, we revisit the problem using the tools of Concurrent Separation Logic with Permissions. The challenges in formulating the proof in this setting as well as the mileage obtained by the new logical tools will be discussed.

Keywords: Concurrent Separation Logic

Full Paper:
http://www.cs.bham.ac.uk/~udr/papers/gc.pdf

Reasoning about Stored Procedures

Bernhard Reus (University of Sussex - Brighton, GB)

This informal talk will address various (incoherent) issues of verification of programs that use stored procedures and recursion through the store. There may also be a demo of an early prototype of a (Smallfoot inspired) tool developed by us supporting such reasoning.

Keywords: Separation Logic for Stored Procedures, Recursion Through the Store, Program Verification

Joint work of: Charlton, B.; Horsfall, B.; Haberland, R; Reus, B.

Verifying Linearizability with Hindsight

Noam Rinetzky (University of London, GB)

We present a proof of safety and linearizability of a highly-concurrent optimistic set algorithm. The key step in our proof is the Hindsight Lemma, which allows a thread to infer the existence of a global state in which its operation can be linearized based on limited local atomic observations about the shared state. The Hindsight Lemma allows us to avoid one of the most complex and non-intuitive steps in reasoning about highly concurrent algorithms: considering the linearization point of an operation to be in a different thread than the one executing it.

The Hindsight Lemma assumes that the algorithm maintains certain simple invariants which are resilient to interference, and which can themselves be verified using purely thread-local proofs. As a consequence, the lemma allows us to

unlock a perhaps-surprising intuition: a high degree of interference makes non-trivial highly-concurrent algorithms in some cases much easier to verify than less concurrent ones.

Keywords: Linearizability, Hindsight, Concurrency

Joint work of: O'Hearn, Peter; Rinetzky, Noam; Vehev, Martin; Yahav, Eran; Yorsh, Greta

Full Paper:
http://www.eecs.qmul.ac.uk/~maon/pubs/PODC10-hindsight.pdf

Type inference for RAJA - A Static Heap Space Analysis of OO-programs.

Dulma Rodriguez (LMU München, DE)

RAJA is a type system for a compile-time analysis of heap-space requirements for Java style object-oriented programs. The system was first described in ESOP 2006 by Hofmann and Jost, where the soundness of the analysis has been proven. Later, in CSL 2009, Hofmann and Rodriguez described efficient type checking for an annotated version of the system. Our next goal is to infer the typing annotations automatically in order to make the system feasible for programming.

In this talk we will present a type inference algorithm consisting of two main parts:

1. A sound and complete constraint-generation system.
2. An algorithm for solving the subtyping constraints by eliminating view variables which is inspired by the Fourier-Motzkin algorithm for eliminating variables from a system of linear inequalities.

Joint work of: Hofmann, Martin; Rodriguez, Dulma

A Step-indexed Kripke Model of Hidden State

Jan Schwinghammer (Universität des Saarlandes, DE)

Frame and anti-frame rules have been proposed as proof rules for modular reasoning about programs. Frame rules allow one to hide irrelevant parts of the state during verification, whereas the anti-frame rule allows one to hide local state from the context.

I describe a possible worlds semantics for Chargueraud and Pottier's type and capability system including frame and anti-frame rules, based on the operational semantics and step-indexed heap relations.

Reasoning about Optimistic Concurrency Using a Program Logic for History

Zhong Shao (Yale University, US)

Optimistic concurrency algorithms provide good performance for parallel programs but they are extremely hard to reason about. Program logics such as concurrent separation logic and rely-guarantee reasoning can be used to verify these algorithms, but they make heavy uses of history variables which may obscure the high-level intuition underlying the design of these algorithms. In this paper, we propose a novel program logic that uses invariants on history traces to reason about optimistic concurrency algorithms. We use past tense temporal operators in our assertions to specify execution histories. Our logic supports modular program specifications with history information by providing separation over both space (program states) and time. We prove Michael's non-blocking stack algorithm and show that the intuition behind such algorithm can be naturally captured using trace invariants.

Keywords: Program Verification, Optimistic Concurrency, Temporal History Invariants, Rely-Guarantee Reasoning

Full Paper:
http://flint.cs.yale.edu/flint/publications/roch.html

See also: To appear in CONCUR'10.

Full Abstraction in a Metalanguage for State

Sam Staton (University Paris-Diderot, FR)

Programming languages only allow the programmer to interact with the machine state through specified operations, such as operations for reading and writing cells. In particular one cannot write programs that copy or discard the state – an observation that goes back at least to Scott and Strachey almost 40 years ago. O'Hearn and Reynolds suggested using a linear state monad translation for Idealized Algol because it preserves more program equivalences than the usual state monad does. In this talk we propose using the Enriched Effect Calculus (EEC), a calculus for linear usage of effects introduced by Egger, Møgelberg and Simpson, as a metalanguage for reasoning about state effects. To support our claim that this is a language capturing stateful computation, we show that the linear state monad translation from a call-by-value language with store into EEC is fully complete in the sense that any term in EEC of a translated type corresponds to a unique program via the translation. The result is not specific to store, but can be applied to any computational effect expressible using algebraic operations in the sense of Plotkin and Power, even to effects that are not usually thought of as stateful. Effect operations are translated using "state access

operations" with linear type. The theory of global store supports state access operations for reading and writing each cell.

As a detailed example we treat local store, i.e., store in which fresh cells can be allocated. In this context, our intuition is that the state keeps track of the cells that have been allocated, as well as what the store contains. We axiomatize a notion of state object for local store, extending the operations for global store above with an operation for allocating cells.

Joint work of: Møgelberg, Rasmus; Staton, Sam

Parametricity via Bisimilarity

Eijiro Sumii (Tohoku University, JP)

I will discuss through examples how to prove parametricity properties by using Sumii et al.'s environmental bisimulations instead of logical relations.

Krivine's Realisability for Low Level Languages

Nicolas Tabareau (Ecole des Mines de Nantes, FR)

I will discuss how to use the notion of Krivine's realisability to develop a semantics for low level languages such as assembly code. This approach offers the possibility to reason modularly about low level programs. In this way, we can prove a compositional version of correctness of a compiler. This technics also enables to show that some low level code coming from the compilation of a high level imperative language (like C) can be connected safely to another low level code coming from a high level functional language (like ML).

Keywords: Krivine's Realisability, Semantics of Low Level Languages, HTT

Joint work of: Jaber, Guilhem; Tabareau, Nicolas

Full Paper:
http://hal.archives-ouvertes.fr/hal-00475210/fr/

See also: Krivine realizability for compiler correctness Jaber G., Tabareau N. LOLA 2010 (LiCS workshop)

Combining Recursive Types and General References: An Unfortunate Observation

Jacob Thamsborg (IT University of Copenhagen, DK)

We observe that a certain naive, possible-world interpretation of general reference types and recursive types does not exists, i.e., we derive a contradiction from assuming the existence.

Joint work of: Birkedal, Lars; Støvring, Kristian; Thamsborg, Jacob

Program Equivalence in a Simple Language with Names

Nikos Tzevelekos (University of Oxford, GB)

The nu-calculus of Pitts and Stark was introduced as a paradigmatic functional language with a very basic local-state effect: references of unit type. These were called *names*, and the motto of the new language went as follows:

> "Names are created with local scope, can be tested for equality, and are passed around via function application, but that is all."

Because of this limited framework, the hope was that fully abstract models and complete proof techniques could be obtained. However, it was soon realised that the behaviour of nu-calculus programs is quite intricate, and program equivalence in particular is surprisingly difficult to capture. In this talk we shall focus on the following "hard" equivalence:

$$\texttt{new}\ x, y\ \texttt{in}\ \lambda f.(fx = fy) \ \ == \ \ \lambda f.\texttt{true}$$

We shall examine attempts and proofs of the above, explain the disadvantages of the proof methods and discuss why program equivalence in this simple language remains to date a mystery.

Keywords: Nu-calculus, Local State, Logical Relations, Game Semantics

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2809

Yarra: A Lightweight Extension to C with Data Integrity and Local Reasoning

David Walker (Princeton University, US)

In this work-in-progress talk, I will discuss the development of Yarra, a lightweight extension of C that allows programmers to specify data integrity constraints. More specifically, Yarra allows programmers to declare new, protected data types. When raw memory is cast to one of these protected types, the only way to access the data is through the use of a pointer with the correct static type — other attempted accesses are dynamically detected and disallowed. This protection mechanism can be used to thwart an important class of non-control data attacks. More broadly, data integrity constraints can be used to improve the robustness and reliability of general-purpose programs. Yarra also comes with a novel program logic that supports sound local reasoning and powerful frame rules, even in the presence of calls to unknown, unverified C libraries.

Keywords: Data Integrity, Write Integrity Testing, Frame Rule, Local Reasoning, Safe C, Array Bounds Checking

Joint work of: Pattabiraman, Karthik; Schlesinger, Cole; Swamy, Nikhil; Walker, David; Zorn, Ben

Automatic Verifiers Based on Separation Logic

Hongseok Yang (University of London, GB)

In the past few years, quite a few automatic verification tools based on separation logic have been developed. In this talk, I will give a brief overview of these tools, and then describe two tools, SpaceInvader and Abductor, in detail. Both SpaceInvader and Abductor are fully automatic verifiers based on separation logic, and they have been developed by me, Cristiano Calcagno and Dino Distefano. The focus of my talk will be the main design decisions that enabled us to come up with and to easily implement efficient algorithms inside those tools.

Keywords: Automatic Verification, Separation Logic, Abstract Interpretation, Static Analysis, Software Verification

Joint work of: Calcagno, Cristiano; Distefano, Dino; Yang, Hongseok