08061 Abstracts Collection
Types, Logics and Semantics for State
— Dagstuhl Seminar —

Amal Ahmed[1], Nick Benton[2], Martin Hofmann[3] and Greg Morrisett[4]

[1] Toyota Technological Inst. - Chicago, USA
amal@tti-c.org
[2] Microsoft Research, GB
nick@microsoft.com
[3] Universität München, D
mhofmann@informatik.uni-muenchen.de
[4] Harvard University, USA
greg@eecs.harvard.edu

Abstract. From 3 February to 8 February 2008, the Dagstuhl Seminar 08061 "Types, Logics and Semantics for State" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. 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. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

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

08061 Executive Summary – Types, Logics and Semantics for State

Amal Ahmed (Toyota Technological Inst. - Chicago, USA), Nick Benton (Microsoft Research, GB), Martin Hofmann (Universität München, D) and Greg Morrisett (Harvard University, USA)

From 3 February to 8 February 2008, the Dagstuhl Seminar 08061 "Types, Logics and Semantics for State" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. 45 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.

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2008/1426

Compiling Self-Adjusting Programs

Umut Acar (Toyota Technological Institute, USA)

Self-adjusting programs respond automatically and efficiently to input changes by tracking the dynamic data dependences of the computation and incrementally updating the output as needed. In this talk, I give an overview of language-based techniques for compiling self-adjusting programs from ordinary programs.

The main idea is to use continuation-passing style (cps) transformation to automatically infer a conservative approximation of the dynamic data dependences. We ensure the efficient propagation of input changes by generating memoizing versions of cps functions that can reuse previous work even when they are invoked with different continuations.

Keywords: Self-adjusting computation, compilations, continuation-passing-style

Step-Indexed Logical Relations

Amal Ahmed (Toyota Technological Institute - Chicago, USA)

Logical relations are a powerful proof technique used to establish many important properties of typed languages, including type safety and contextual equivalence of terms. While logical relations for simple type systems are straightforward, the addition of recursive types or mutable references significantly complicates matters. The key problem is that logical relations, normally defined by induction on the structure of types, are no longer well founded in the presence of recursive types, impredicative polymorphism, or ML- and Java-style references.

In this talk, I will give an introduction to step-indexed logical relations which are indexed not just by types, but also by the number of steps available for future evaluation. This stratification has proved to be an effective means of uniformly handling various circularities from recursive functions, to recursive types, to impredicative polymorphism, to the cyclic stores that arise in the presence of mutable references. In particular, I will present a step-indexed model for mutable references aimed at proving type safety, as well as a step-indexed logical relation for proving contextual equivalence for the polymorphic lambda calculus with recursive types.

A relational proof system for non-interference of unstructured bytecode

Lennart Beringer (LMU München, D)

We present a relational proof system for checking that a bytecode program satisfies non-interference, a basic security property restricting the flow of information in programs.

The system admits assignments to low variables and return instructions to occur in high branches and applies to programs with structured as well as unstructured control flow. The technical contribution consists of a novel kind of relational invariants called relational shape descriptions. Relational shape descriptions approximate the identity of values within a single state as well as (modulo the information flow indistinguishability) across a pair of states. The proof system employs relational shape descriptions in a type-like fashion to promote relations between two initial states to the corresponding final states and the return values. In addition to syntax-directed rules, we present rules for code fragments that are related by simple peephole transformations, preparing for the verification of code resulting from optimising compilers.

Keywords: Non-interference, relational reasoning, program transformations, type systems

Introduction to FM-domain theory and its application to modeling dynamic allocation

Lars Birkedal (IT University of Copenhagen, DK)

In this talk I give an introduction to FM-domain theory (domain theory in the category of FM sets, sets with a permutation action) and outline how FM-domain theory can be used to give denotational semantics of higher-order programming languages with dynamic allocation as in recent papers by Benton-Leperchey (TLCA'05), Bohr-Birkedal (APLAS'06), Birkedal-Yang (FOSSACS'07).

Keywords: FM-domains, dynamic allocation

Formal Reasoning on Imperative ML Programs

Arthur Charguéraud (INRIA Paris-Rocquencourt, F)

I will present a technique for certifying imperative ML programs using a standard HOL theorem prover. The approach is 3-step:

1) type the imperative program in a type system that extends System F with static notions of regions and capabilities,

2) translate the program into a purely functional language, using a type-directed translation (which preserves both the meaning and the global structure of the source),

3) generate a higher-logic formula from the generated functional program, and use it to prove properties about the program, including safety and termination.

Note that steps (1) and (2) extend some earlier work that appears in "Functional Translation of a Calculus of Capabilities" (joint work with François Pottier). Step (3) is work in progress.

In this talk, I will briefly describe each of the three steps, and then illustrate the global process on pratical examples.

Joint work of: Charguéraud, Arthur; Pottier, François

A Unified Framework for Verification Techniques for Object Invariants

Sophia Drossopoulou (Imperial College London, GB)

Object invariants define the consistency of objects. They have subtle semantics, mainly because of call-backs, multi-object invariants, and subclassing.

Several verification techniques for object invariants have been proposed.

It is difficult to compare these techniques, and to ascertain their soundness, because of their differences in restrictions on programs and invariants, in the use of advanced type systems (e.g., ownership types), in the meaning of invariants, and in proof obligations.

We develop a unified framework for such techniques. We distil seven parameters that characterise a verification technique, and identify sufficient conditions on these parameters which guarantee soundness. We instantiate our framework with three verification techniques from the literature, and use it to assess soundness and compare expressiveness.

Keywords: Object invariants, visible states semantics, verification, sound

Joint work of: Drossopoulou, Sophia; Francalanza, Adrian; Müller, P.; Summers, Alexander J.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1427

Specifications of OO programs are not OO

Sophia Drossopoulou (Imperial College London, GB)

We argue that current techniques for the specification of OO programs do not follow the OO paradigm, in the sense that they talk about the state/properties of objects before and after messages, rather than talk about the behaviour of objects in terms of later messages.

We suggest that a possible way to make specifications more OO, would be to express the behaviour of an object in terms of messages sent earlier to relevant other objects.

Keywords: Specification, object oriented, message, trace

Joint work of: Drossopoulou, Sophia; Noble, James

A Step-Indexed Model of Substructural State

Matthew Fluet (Toyota Technological Institute - Chicago, USA)

The concept of a 'unique' object arises in many emerging programming languages such as Clean, CQual, Cyclone, TAL, and Vault. In each of these systems, unique objects make it possible to perform operations that would otherwise be prohibited (e.g., deallocating an object) or to ensure that some obligation will be met (e.g., an opened file will be closed). However, different languages provide different interpretations of 'uniqueness' and have different rules regarding how unique objects interact with the rest of the language.

Our goal is to establish a common model that supports each of these languages, by allowing us to encode and study the interactions of the different forms of uniqueness. The model we provide is based on a substructural variant of the polymorphic λ-calculus, augmented with four kinds of mutable references: unrestricted, relevant, affine, and linear. The language has a natural operational semantics that supports deallocation of references, strong (type-varying) updates, and storage of unique objects in shared references. We establish the strong soundness of the type system by constructing a novel, semantic interpretation of the types.

Joint work of: Fluet, Matthew; Ahmed, Amal; Morrisett, Greg

Pure Pointer Programs with Universal Iteration

Martin Hofmann (LMU München, D)

We introduce a formal class of pure pointer programs (PURPLE) and study them on locally ordered graphs. Existing classes of pointer algorithms, such as Jumping Automata on Graphs (JAGS) or Deterministic Transitive Closure (DTC) logic, often exclude simple programs. PURPLE subsumes these classes and allows algorithms whose pseudocode uses a constant number of variables to be defined in a natural way. It does so by providing a primitive for iterating an algorithm over all nodes of the input graph in an unspecified order.

Since pointers are given as an abstract data type rather than as binary digits we expect that logarithmic-size worktapes cannot be encoded using pointers as is done, e.g. in totally-ordered DTC logic. We show that this is indeed the case by proving that the property "the number of nodes is a power of two" which is in LOGSPACE is not representable in PURPLE.

Keywords: Pointer algorithms, semantics, abstract data type, computational complexity

Joint work of: Hofmann, Martin; Schöpp, Ulrich

Ghost variables, resources, and object invariants in program logics

Martin Hofmann (LMU München, D)

Ghost variables are assignable variables that appear in program annotations but do not correspond to physical entities. They are used to facilitate specification and verification, e.g., by using a ghost variable to count the number of iterations of a loop, to express extra-functional behaviours, like resource usage, and also to delineate validity regions of invariants.

I will discuss the use of ghost variables in the context of proof carrying code and argue that they should be avoided in certificates but have their legitimate use during proof generation.

These points will be substantiated by a formal model of ghost variables in ordinary program logic and an automatic procedure allowing one to eliminate them from specifications and proofs in a compositional way.

Keywords: Program logic, specification, proof-carrying code

Joint work of: Hofmann, Martin; Pavlova, Mariela

Reasoning about stateful interactions using a typed Hennessy-Milner logic

Kohei Honda (Queen Mary College - London, GB)

The pi-calculus can represent a wide range of behaviour, including sequential or concurrent, synchronous or asynchronous, shared variable or message passing, functional or imperative, stateful or stateless, and first-order or higher-order. These different classes of behaviours arise as types for processes — each type discipline yields a universe of typed processes which precisely capture a given notion of behaviour. Thus the pi-calculus offers a uniform framework with which we can study these diverse behaviours, for example logics for programming languages with significant features such as higher-order procedures, local state and different kinds of concurrency.

In the past years Martin Berger, Nobuko Yoshida and I have been carrying out a series of studies for developing logics for programming languages based on analysis of such a typed universe. These include logics for aliasing, higher-order functions, and local state. All of these logics are equipped with a strong completeness property. What has been missing is the precise link between these program logics on the one hand and process logics for the corresponding typed pi-calculi on the other. Such a link will be necessary not only for capturing concurrency but also for having a uniform framework to reason about combination of diverse behaviours, which is how real computing systems are built.

The aim of this talk is to present this missing link: a process logic for typed processes which can be uniformly adapted for many kinds of type disciplines for

the pi-calculus including those representing programming languages, based on the works by Stirling, Dam and Amadio as well as a recent study on logics for higher-order control by the first author. Rather than showing a general theory, we shall focus on the theme of this workshop — how we can reason about state, especially in the context of concurrent communicating processes — and argue for the merit of understanding, specifying and validating properties of stateful concurrent programs using the combination of types (which give a weak safety guarantee only ensuring lack of constructor error but at the same time articulate the foundations of dynamics of computing) and logics (which give arbitrarily detailed specifications based on the articulation of typed program syntax). After exploring examples, we shall also outline several completeness and other theoretical results we have obtained recently, and discuss their significance for practice.

Keywords: Types, Logic, Pi-calculus, Hennessy-Milner Logic, state, session types

Joint work of: Berger, Martin; Yoshida, Nobuko; Honda, Kohei

Carbon credits for resource bounded computation

Steffen Jost (University of St Andrews, GB)

We will explore a compile-time program analysis to determine the worst-case heap-space usage of a given program as presented by M.Hofmann and S.Jost. After an intuitive overview over the general principle, we will discuss the current state of research on this method and discuss future extensions. The 20min presentation was followed by a quick demonstration of the analysis on an implementation of the Quick-sort algorithm.

The analysis applies a basic principle from amortised complexity analysis to construct a linear programming problem while traversing a standard typing derivation.

Any existing solution to these constraints obtained by a standard LP-solver gives rise to a simple arithmetic expression, which linearly depends on the program's input sizes (e.g. $ax + by + c$ for a program that takes two lists of length x and y as its input). We have proved for both a standard functional language and a subset of Java (including inheritance, downcast, update and aliasing) that these expressions present a strict upper bound on a program's heap-space consumption. In both languages it is possible to account for deallocation primitives, whose safety relies on other researchers' work orthogonal to our approach.

We have implemented the analysis for a functional language and expanded it to bound execution time and stack-space usage as well as a part of ongoing research and will conclude the talk with a short demonstration.

Keywords: Type systems, Program Analysis, Resource bounded computation, Amortised Analysis

Reading, Writing and Relations: Towards Extensional Semantics for Effect Analyses

Andrew Kennedy (Microsoft Research UK - Cambridge, GB)

We give an elementary semantics to an effect system, tracking read and write effects by using relations over a standard extensional semantics for the original language. The semantics establishes the soundness of both the analysis and its use in effect-based program transformations.

Joint work of: Benton, Nick; Kennedy, Andrew; Hofmann Martin; Beringer, Lennart

Full Paper:
http://research.microsoft.com/~akenn/effects/rwraplas.pdf

Deriving Proof Techniques for Equivalence

Vasileios Koutavas (Northeastern Univ. - Boston, USA)

Contextual equivalence, i.e. the property that two expressions are indistinguishable inside any program context, is a fundamental property of program expressions. I will present a framework for deriving techniques for proving contextual equivalence, which are sound and complete, but also useful, in a variety of languages. The advantages of the derived proof methods are that they successfully deal with imperative features as well as higher-order features (callbacks, higher-order functions).

We have used this framework to derive sound and complete methods for proving contextual equivalence for a variety of languages, including an untyped imperative lambda calculus, an imperative object calculus, an imperative core of Java, and the nu-calculus.

Keywords: Contextual equivalence, bisimulations, imperative languages, higher-order languages

Verifying the Subject-Observer Pattern with Higher-Order Separation Logic

Neel Krishnaswami (CMU - Pittsburgh, USA)

The subject-observer design pattern is a very common idiom in object-oriented systems; for example, it is an essential part of the model-view-controller pattern for programming graphical user inter faces. We give a modular proof technique using separation logic to verify this pattern. This proof method is modular in the sense that subjects and observers can be verified independently, and both can be verified independently of client code that calls both.

Keywords: Separation Logic, Design Patterns, Subject-Observer

Context Lemmas and Bisimulation for Lambda-Calculi with References

Soeren Lassen (Google - Mountain View, USA)

In this talk I explain, characterize, and contrast some operationally-based methods for reasoning about contextual equivalence of general higher-order functions and dynamically-allocated references: 1. Context lemmas, including the CIU theorem. 2. Applicative bisimulation. 3. Logical bisimulation and environmental bisimulation. 4. Normal form bisimulation, a.k.a. open (applicative) bisimulation.

They all employ elementary syntactic, inductive and co-inductive definitions and proofs to develop useful, operationally-based proof methods to establish non-trivial program equivalences, which I illustrate in an untyped ML fragment with references.

Keywords: Lambda-calculus, references, program equivalence, bisimulation

The State of State in JML

Gary T. Leavens (Univ. of Central Florida - Orlando, USA)

The Java Modeling Language (JML) is used to specify, check, and verify detailed designs for Java classes and interfaces. JML is an open, international, collaborative effort among some 20 research groups and projects. This talk briefly gives an overview of JML, focusing on JML's features for dealing with state. In particular, I describe extensions to JML proposed or implemented for dealing with prevention of representation exposure, and for framing and invariants. I show how these interact to support modular reasoning about layered abstractions

This work was supported in part by US NSF grant CNS 08-08913

Keywords: JML, state, datagroup, representation function, assignable clause, modifies clause

See also:
http://jmlspecs.org

Modular Verification of Higher-Order Methods with Mandatory Calls Specified by Model Programs

Gary T. Leavens (Univ. of Central Florida - Orlando, USA)

What we call a "higher-order method" (HOM) is a method that makes mandatory calls to other dynamically-dispatched methods. Examples include template methods as in the Template method design pattern and notify methods in the Observer pattern.

HOMs are particularly difficult to reason about, because standard pre- and postcondition specifications cannot describe the mandatory calls. For reasoning about such methods, existing approaches use either higher-order logic or traces, but both are unintuitive and verbose.

We describe a simple, intuitive, and modular approach to specifying HOMs We show how to verify calls to HOMs and their code using first-order verification conditions, in a sound and modular way. Verification of client code that calls HOMs can take advantage of the client's knowledge about the mandatory calls to make strong conclusions. Our verification technique validates and explains traditional documentation practice for HOMs, which typically shows their code. However, specifications do not have to expose all of the code to clients, but only enough to determine how the HOM makes its mandatory calls.

Keywords: Model program, verification, specification languages, grey-box approach, higher order method, mandatory call, Hoare logic, refinement calculus

Joint work of: Shaner, Steve; Leavens, Gary T.; Naumann, David A.

Full Paper:
http://doi.acm.org/10.1145/1297027.1297053

See also: Steve M. Shaner, Gary T. Leavens, David A. Naumann, Modular Verification of Higher-Order Methods with Mandatory Calls Specified by Model Programs. In International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA), Montreal, Canada, October 2007, pages 351-367.

Heap Decomposition with Applications to Concurrency Analysis

Roman Manevich (Tel Aviv University, IL)

We demonstrate shape analyses that can achieve a state space reduction exponential in the number of threads compared to the state-of-the-art analyses, while retaining sufficient precision to verify sophisticated properties such as linearizability. The key idea is to abstract the global heap by decomposing it into (not necessarily disjoint) subheaps, abstracting away some correlations between them.

These new shape analyses are instances of an analysis framework based on heap decomposition. This framework allows rapid prototyping of complex static analyses by providing efficient abstract transformers given user-specified decomposition schemes. Initial experiments confirm the value of heap decomposition in scaling concurrent shape analyses.

Keywords: Concurrency, Shape Analysis, Cartesian Abstraction

Program Verification with Hoare Type Theory

Aleksander Nanevski (Microsoft Research UK - Cambridge, GB)

In this talk, I will describe Hoare Type Theory (HTT) which combines dependent type theory like Coq with features which for specification and verification in the style of Hoare Logic.

This combination is desirable for many reasons. On the type-theoretic side, it makes it possible to integrate stateful behaviour into languages and logics that have so far been limited to be purely functional. On the Hoare Logic side, it makes is possible to use the higher-order data abstraction and information hiding mechanisms, which are essential for scaling any kind of verification effort. On the functional programming side, the language may roughly be considered as a dependently typed extension of core Haskell.

Finally, from the technical standpoint, it is interesting that the design of HTT relates in an essential way some of the most venerable ideas from language theory like Dijkstra's predicate transformers, Curry-Howard isomorphism, monads, as well as the more recent idea of Separation Logic, which have not been connected before.

I will discuss the implementation of HTT (called Y-not) which is currently under way, as well as the possibilities for scaling HTT to support further programming features.

Keywords: Type theory, hoare logic, separation logic

An admissible second order frame rule in regional logic

David Naumann (Stevens Institute of Technology, USA)

Higher order frame rules in separation logic provide a way to understand disciplines such as ownership for information hiding in object based programs.

Recent of Banerjee, Naumann, and Rosenberg uses explicit regions to express, in classical first-order assertions, the read-footprints of predicates and write-footprints of commands, supporting an ordinary frame rule.

On this basis, I give a second order frame rule, show its admissibility, and describe its use in encoding disciplines like ownership.

An admissible second order frame rule in region logic (extended abstract)

David Naumann (Stevens Institute of Technology, USA)

Shared mutable objects and reentrant callacks can subvert encapsulation in object-based programs.

For modular reasoning, verifiers rely on methodologies. These combine special annotations or types with instrumentation (ghost state) and syntactic restrictions on programs and specifications, which poses challenges for proving soundness and for comparing/combining methodologies. This paper formalizes a second order frame rule, similar to that of separation logic but for a logic with explicit regions. The rule captures proof obligations of invariant methodologies such as ownership for dynamically instantiable abstractions.

Soundness with respect to a standard semantics is proved by admissibility argument.

Assignment, Substitution and Abstraction

Peter O'Hearn (Queen Mary College - London, GB)

Since the beginning of program logic substitution has been used to describe the semantics of assignment, as in the definition where P[e/x] as the weakest precondition of x:=e and postcondition P. Since before program logic, it has been known that conflating the store and the environment is inconsistent, except in simple situations. Some authors, including this author, have persisted with the substitution semantics in theoretical work, for simplicity. For the most part, this works, until one reaches data abstraction (or, of course, a powerful procedure mechanism). Following the work of Reynolds-Mitchell-Plotkin on types, and then Parkinson and others on program logic, when treating abstraction (as in classes, modules, etc) the most natural thing in the world is to use predicate variables. But then consider the wp P[e/x] if P is a predicate variable. We are in trouble if the predicate depends on x. This is a known problem, and there are possible several routes to solution. My main question is, is this the final straw, forcing us (or, me!) to finally give up on the punning of locations as variables?

Class invariants the end of the road

Matthew Parkinson (Cambridge University, GB)

Class invariants have formed the foundation of most modern object-oriented verification systems. Unfortunately, this causes difficulties as invariants are not always the correct way to structure proofs of programs. In this talk, I demonstrate a different way using predicates from separation logic, which does suffer the same problems.

Keywords: Class invariants, verification

Towards Light-Weight OO-Components with Fully Abstract Semantics

Arnd Poetzsch-Heffter (TU Kaiserslautern, D)

Behavioral semantics for components abstract from implementation details and describe the components' behavior independent of the components' implementations. Such semantics provides an important foundation for behavioral substitutability and interface specifications. In this talk, I investigate a component model for class-based object-oriented languages with aliasing, subclassing, and dynamic dispatch. The code of a component consists of a creator class and possibly several other classes. Like objects, components are instantiable at runtime. A component instance is realized by a dynamically evolving set of objects with a clear boundary to the environment. Based on a small step operational semantics, I develop a behavioral semantics that is expressed in terms of the messages crossing the component boundary.

Keywords: Object-orientation, program components, operational semantics, full abstractness

Hiding local state in direct style: a higher-order anti-frame rule

Francois Pottier (INRIA Paris-Rocquencourt, F)

Separation logic involves two dual forms of modularity: local reasoning makes part of the store invisible within a static scope, whereas hiding local state makes part of the store invisible outside a static scope. In the recent literature, both idioms are explained in terms of a higher-order frame rule. I point out that this approach to hiding local state imposes continuation-passing style, which is impractical. Instead, I introduce a higher-order anti-frame rule, which permits hiding local state in direct style. I formalize this rule in the setting of a type system, equipped with linear capabilities, for an ML-like programming language, and prove type soundness via a syntactic argument. Several applications illustrate the expressive power of the new rule.

A System for Generating Static Analyzers for Machine Instructions

Thomas Reps (University of Wisconsin - Madison, USA)

This paper describes the design and implementation of a language for specifying the semantics of an instruction set, along with a run-time system to support the static analysis of executables written in that instruction set. The work advances the state of the art by creating multiple analysis phases from a specification of the concrete operational semantics of the language to be analyzed.

Keywords: Machine-code analysis, instruction-set semantics, tool generator

Joint work of: Lim, Junghee; Reps, Thomas

Separation Logic for Higher-order Store

Bernhard Reus (University of Sussex - Brighton, GB)

Separation logic is a Hoare-style logic for reasoning about pointer-manipulating programs. Its core ideas have recently been extended from low-level to richer, high-level languages.

In this paper we develop a new semantics of the logic for a simple programming language where code can be stored (i.e., with a higher-order store).

The main improvement upon previous work is the simplicity of the model. As a consequence, several restrictions imposed by the semantics are removed, leading to a considerably more natural assertion language with a powerful specification logic.

Keywords: Separation logic, program logics, Hoare logic, higher-order store, code pointers

Joint work of: Birkedal, Lars; Reus, Bernhard; Schwinghammer, Jan; Yang, Hongseok

The Current State of Grainless Semantics

John Reynolds (CMU - Pittsburgh, USA)

Conventional semantics for shared-variable concurrency suffers from the "grain of time" problem, i.e., the necessity of specifying a default level of atomicity. We propose a semantics that avoids such a choice by regarding all interference that is not controlled by explicit synchronization as catastrophic. It is based on three principles:

- Operations have duration and can overlap one another during execution.
- If two overlapping operations touch the same location, the meaning of the program execution is “wrong”.
- If, from a given starting state, execution of a program can give "wrong", then no other possibilities need be considered.

We show a small-step grainless semantics due to Reynolds and a large-step version due to Brookes. As a first step in relating these approaches, and in seeking a still more abstract semantics, we develop a theory of grainless semantics in the absence of synchronization primitives.

Keywords: Grainless semantics, shared-variable concurrency, grain of time

Modular Shape Analysis for View-Serializable Libraries

Noam Rinetzky (Tel Aviv University, IL)

We present novel modular static shape analysis algorithms for concurrent libraries.

Our analyses conservatively verify the absence of certain memory and concurrency errors, verify a certain class of program assertions, and infer shape (heap) module invariants. The key idea is to focus on a class of concurrent programs that follow certain standard locking policies which ensure view-serializability. This allows our analyses not to consider interleaving between the low-level instructions implementing high-level operations on thread-shared data structures.

Technically, we employ existing sequential shape analysis algorithms to perform interprocedural and modular analyses that can establish sufficient conditions for view-serializability for concurrent libraries.

Keywords: Modularity, serializability, shape analysis, program analysis

Joint work of: Rinetzky, Noam; Bouajjani, Ahmed; Ramalingam, Ganesan; Sagiv, Mooly; Yahav, Eran

Modular Verification with Shared Abstractions

Noam Rinetzky (Tel Aviv University, IL)

Modular verification of shared data structures is a challenging problem:

Side-effects in one module that are observable in another module make it hard to analyze each module separately. We present a novel approach for modular verification of shared data structures. Our main idea is to verify that the inter-module sharing is restricted to a user-provided specification which also enables the analysis to handle side-effects. For our approach, we constructed a novel modular static analysis and implemented a proof-of-concept analyzer.

Using the analyzer, we verified some shared data structures which cannot be verified modularly by current tools.

Keywords: Modular, Verification, Sharing, ADT, model fields

Joint work of: Juhasz, Uri; Rinetzky, Noam; Poetzsch-Heffter, Arnd; Sagiv, Mooly; Yahav, Eran

Thread Quantification for Concurrent Shape Analysis

Mooly Sagiv (Tel Aviv University, IL)

We present new algorithms for automatically verifying properties of programs with an unbounded number of threads.

Our algorithms are based on a new abstract domain whose elements represent thread-quantified invariants: i.e., invariants satisfied by all threads. We exploit existing abstractions to represent the invariants. Thus, our technique lifts existing abstractions by wrapping universal quantification around elements of the base abstract domain.

Such abstractions are effective because they are thread-modular: e.g., they can capture correlations between the local variables of the same thread as well as correlations between the local variables of a thread and global variables, but forget correlations between the states of distinct threads. (The exact nature of the abstraction, of course, depends on the base abstraction lifted in this style.)

We present techniques for computing sound transformers for the new abstraction by using transformers of the base abstract domain. We illustrate our technique in this paper by instantiating it to the Boolean Heap abstraction, producing a Quantified Boolean Heap abstraction. We have implemented an instantiation of our technique with Canonical Abstraction as the base abstraction and used it to successfully verify linearizability of data-structures in the presence of an unbounded number of threads.

Joint work of: Berdine, Josh; Lev-Ami, Tal; Manevich, Roman; Ramalingam, Ganesan; Sagiv, Mooly

An Overview on Shape Analysis

Mooly Sagiv (Tel Aviv University, IL)

Shape analysis concerns the problem of automatically inferring shape invariants for programs that perform destructive updating on dynamically allocated storage.

This was an overview on shape analysis.

Building a Verifying Compiler for C

Wolfram Schulte (Microsoft Corp. - Redmond, USA)

The Microsoft Verifying C Compiler (VCC) is a static analysis tool that uses automatic first order theorem proving to show formally that a given sequential C program, compiled for the Intel x86-32 or x86-64 processors, does what is stated in its specification, expressed as function pre- and post conditions.

VCC uses three formally related semantics. The base model represents values as bit vectors and accesses memory as individual bytes, the second represents values as mathematical integers and accesses memory in word sizes, the third uses the C type system to rule out many pointer aliases.

For modular reasoning VCC introduces a 'region-based' memory management using pure functions and abstract frameing to guarantee that functions only write/read/allocate/free certain locations.

We show how VCC is used to specify, implement and verify the functional correctness of a queue and we sketch the design, implementation and proof for a small hypervisor.

Keywords: Automatic deductive verification; extended static checking; C; design by contract; region-based memory management; abstract frameing; hypervisor

Joint work of: Schulte, Wolfram; Moskal, Michal; Venter, Herman

Step-indexed Semantics of Imperative Objects

Jan Schwinghammer (Saarland University, D)

Step-indexed semantic models of types were proposed as an alternative to purely syntactic proofs of type safety using subject reduction. In joint work with Catalin Hritcu, and building on work by Ahmed, Appel and others, we have constructed a step-indexed model for the imperative object calculus of Abadi and Cardelli. Providing a semantic account of this calculus using more 'traditional', domain-theoretic approaches has proved challenging due to the combination of dynamically allocated objects, higher-order store, and an expressive type system. Here I'll show that the step-indexed model can interpret a rich type discipline with object types, subtyping, recursive and bounded quantified types in the presence of state.

Joint work of: Hritcu, Catalin; Schwinghammer, Jan

Modular Development of System Software: An Overview

Zhong Shao (Yale University, USA)

Certified software consists of a machine executable program plus a rigorous proof (checkable by computer) that the software is free of bugs with respect to a particular specification. Both the proof and the specification are written using a general-purpose mathematical logic, the same logic which all of us use (in reasoning) in our daily life. The logic is also a programming language: everything written in logic, including proofs and specifications, is developed using software known as a proof assistant; they can be mechanically checked for correctness by a small program known as a proof checker. As long as the logic we use is consistent, and the specification describes what the user wants, we can be sure that the underlying software is free of bugs with respect to the specification.

The conventional wisdom is that certified software will never be practical because any real software must also rely on the underlying operating system which is too low-level and complex to be verifiable.

In recent years, however, there have been many advances in the theory and engineering of mechanized proof systems applied to verification of low-level code,

including proof-carrying code, certified assembly programming, logic-based type system, and certified or certifying compilation. In this talk, I will give an overview of this exciting new area, focusing on key insights and high-level ideas that make the work on certified software differ from traditional style program verification. I will also describe several recent work—done by my group at Yale—on building certified garbage collectors, OS bootloader, thread implementation, and stack-based control libraries.

What if Hoare Logic were not hypothetical?

Ian Stark (University of Edinburgh, GB)

Preconditions in classic Hoare Logic are 'hypothetical': there is no requirement that a precondition should hold before executing code, but if it does then on completion the postcondition is true. However, several applications of Hoare Logic use a stricter interpretation where a precondition must hold before code is executed. This is the 'contract' approach of Meyer, and also appears in the JML 'requires' clause. The distinction shows up in variations of the Hoare rule for procedure call, and its corresponding weakest precondition. Strict preconditions are also appropriate when encoding rich type systems into program logics, where function application demands that arguments are of a given type.

However, the conventional semantics of Hoare triples as statements about state relations does not support this strict interpretation. We propose a refined semantics for Hoare logic, based on existing notions of resource algebras, that captures strict preconditions. A key novelty is that the validity of triples becomes relative to procedure specifications, even — recursively — for those procedures themselves. In joint work with Alberto Momigliano and Randy Pollack, we are adapting Nipkow's shallow encoding of Hoare logic in Isabelle to account for strict preconditions.

A Logic for Reasoning about Faulty Programs

David Walker (Princeton University, USA)

In this talk, we discuss the many contexts in which it is necessary to consider the ramifications and results of running software on unreliable hardware. We propose a new program logic that can be used to verify such software despite the possible presence of faults.

Keywords: Hoare Logic, Transient Faults, Verification, Fault Tolerance

Scalable Shape Analysis For Systems Code

Hongseok Yang (Queen Mary College - London, GB)

Pointer safety faults in device drivers are the number one cause of crashes in operating systems code. In principle, shape analysis tools can be used to prove the absence of this type of error. In practice, however, shape analysis is not used due to the unacceptable mixture of scalability and precision provided by existing tools. In this talk, I will describe a new join operation for the separation domain which aggressively abstracts information for scalability yet does not lead to false error reports. The operator is a critical piece of a new shape analysis tool that provides an acceptable mixture of scalability and precision for industrial application.

Experiments with our tool on whole Windows and Linux device drivers (firewire, pci-driver, cdrom, md, etc.) represent the first working application of shape analysis to whole industrial programs—and the beginning of the end for the largest problem plaguing the correctness of systems code.

Keywords: Shape Analysis, Program Verification, Abstract Interpretation, Separation Logic

Joint work of: Yang, Hongseok; Lee, Oukseh; Berdine, Josh; Calcagno, Cristiano; Cook, Byron; Distefano, Dino; O'Hearn, Peter

Multiparty Asynchronous Session Types

Nobuko Yoshida (Imperial College London, GB)

A session takes place between two parties; after establishing a connection, each party interleaves local computations and states with communications (sending or receiving) with the other party. Session types characterise such behaviour in terms of the types of values communicated and the shape of protocols. They have been developed for the pi-calculus, Ambients, multi-threaded functional languages, Web Description languages, F♯, CORBA interfaces and concurrent and distributed Java.

In this talk, I introduce an extension of the session types to multiparty, asynchronous interactions, which often arise in practical communication-centred applications with mutable states. The theory introduces a new notion of types in which interactions involving multiple peers are directly abstracted as a global scenario. A global type plays the role of "a shared agreement" among communication peers, and is used as a basis of efficient type checking through its projection onto individual peers. The fundamental properties of the session type discipline such as communication safety, progress and session fidelity are established for general n-party asynchronous interactions.

Keywords: Session Types, Multiparty Interactions, Conversation, State, Concurrency, Mobile Processes and Communication

Joint work of: Carbone, Marco; Honda, Kohei; Yoshida, Nobuko

Full Paper:
http://wwwhomes.doc.ic.ac.uk/∼yoshida/multiparty