JFP **26**, e8, 7 pages, 2016. © Cambridge University Press 2016
doi:10.1017/S0956796816000125

PhD Abstracts

GRAHAM HUTTON
University of Nottingham, UK
(*e-mail:* graham.hutton@nottingham.ac.uk)

Many students complete PhDs in functional programming each year. As a service to the community, the Journal of Functional Programming publishes the abstracts from PhD dissertations completed during the previous year.

The abstracts are made freely available on the JFP website, i.e. not behind any paywall. They do not require any transfer of copyright, merely a license from the author. A dissertation is eligible for inclusion if parts of it have or could have appeared in JFP, that is, if it is in the general area of functional programming. The abstracts are not reviewed.

We are delighted to publish 6 abstracts from 2015/16 in this round and hope that JFP readers will find many interesting dissertations in this collection that they may not otherwise have seen. If a student or advisor would like to submit a dissertation abstract for publication in this series, please contact the series editor for further details.

Graham Hutton
PhD Abstract Editor

Verification of Message Passing Concurrent Systems

EMANUELE D'OSUALDO
University of Oxford, UK

Date: September 2015; Advisor: C.-H. Luke Ong
URL: http://tinyurl.com/zuzdbva

This dissertation is concerned with the development of fully-automatic methods of verification, for message-passing based concurrent systems.

In the first part of the thesis we focus on Erlang, a dynamically typed, higher-order functional language with pattern-matching algebraic data types extended with asynchronous message-passing. We define a sound parametric control-flow analysis for Erlang, which we use to bootstrap the construction of an abstract model that we call *Actor Communicating System* (ACS). ACS are given semantics by means of *Vector Addition Systems* (VAS), which have rich decidable properties. We exploit VAS model checking algorithms to prove properties of Erlang programs such as unreachability of error states, mutual exclusion, or bounds on mailboxes. To assess the approach empirically, we constructed Soter, a prototype implementation of the verification method, thereby obtaining the first fully-automatic, infinite-state model checker for a core concurrent fragment of Erlang.

The second part of the thesis addresses one of the major sources of imprecision in the ACS abstraction: process identities. To study the problem of algorithmically verifying models where process identities are accurately represented we turn to the π-calculus, a process algebra based around the notion of *name* and *mobility*. The full π-calculus is Turing-powerful so we focus on the *depth-bounded* fragment introduced by Roland Meyer, which enjoys decidability of some verification problems. The main obstacle in using depth-bounded terms as a target abstract model, is that depth-boundedness of arbitrary π-terms is undecidable. We therefore consider the problem of identifying a fragment of depth-bounded π-calculus for which membership is decidable. We define the first such fragment by means of a novel type system for the π-calculus. Typable terms are ensured to be depth-bounded. Both type-checking and type inference are shown to be decidable. The constructions are based on the novel notion of T-compatibility, which imposes a hierarchy between names. The type system's main goal is proving that this hierarchy is preserved under reduction, even in the presence of unbounded name creation and mobility.

The Modular Compilation of Effects

LAURENCE E. DAY
University of Nottingham, UK

Date: October 2015; Advisor: Graham Hutton
URL: http://tinyurl.com/gtv62c3

The introduction of new features to a programming language often requires that its compiler goes to the effort of ensuring they are introduced in a manner that does not interfere with the existing code base. Engineers frequently find themselves changing code that has already been designed, implemented and (ideally) proved correct, which is bad practice from a software engineering point of view. This thesis addresses the issue of constructing a compiler for a source language that is modular in the computational features that it supports. Utilising a minimal language that allows us to demonstrate the underlying techniques, we go on to introduce a significant range of effectful features in a modular manner, showing that their syntax can be compiled independently, and that source languages containing multiple features can be compiled by making use of a fold. In the event that new features necessitate changes in the underlying representation of either the source language or that of the compiler, we show that our framework is capable of incorporating these changes with minimal disruption. Finally, we show how the framework we have developed can be used to define both modular evaluators and modular virtual machines.

Profiling of Parallel Programs in a Non-Strict Functional Language

HENRIQUE FERREIRO
University of A Coruña, Spain

Date: January 2016; Advisor: Laura Castro and Kevin Hammond
URL: http://tinyurl.com/zcp9bz6

Purely functional programming languages offer many benefits to parallel programming. The absence of side effects and the provision for higher-level abstractions eases the programming effort. In particular, non-strict functional languages allow further separation of concerns and provide more parallel facilities in the form of semi-implicit parallelism. On the other hand, because the low-level details of the execution are hidden, usually in a runtime system, the process of debugging the performance of parallel applications becomes harder. Currently available parallel profiling tools allow programmers to obtain some information about the execution; however, this information is usually not detailed enough to precisely pinpoint the cause of some performance problems. Often, this is because the cost of obtaining that information would be prohibitive for a complete program execution.

In this thesis, we design and implement a parallel profiling framework based on *execution replay*. This debugging technique makes it possible to simulate recorded executions of a program, ensuring that their behaviour remains unchanged. The novelty of our approach is to adapt this technique to the context of parallel profiling and to take advantage of the characteristics of non-strict purely functional semantics to guarantee minimal overhead in the recording process. Our work allows to build more powerful profiling tools that do not affect the parallel behaviour of the program in a meaningful way. We demonstrate our claims through a series of benchmarks and the study of two use cases.

Program Synthesis with Types

PETER-MICHAEL OSERA
University of Pennsylvania, USA

Date: August 2015; Advisor: Steve Zdancewic
URL: http://tinyurl.com/za7oxdr

Program synthesis, the automatic generation of programs from specification, promises to fundamentally change the way that we build software. By using synthesis tools, we can greatly speed up the time it takes to build complex software artifacts as well as construct programs that are automatically correct by virtue of the synthesis process. Studied since the 70s, researchers have applied techniques from many different sub-fields of computer science to solve the program synthesis problem in a variety of domains and contexts. However, one domain that has been less explored than others is the domain of typed, functional programs. This is unfortunate because programs in richly-typed languages like OCaml and Haskell are known for writing themselves once the programmer gets the types correct. In light of this observation, can we use type theory to build more expressive and efficient type-directed synthesis systems for this domain of programs? This dissertation answers this question in the affirmative by building novel type-theoretic foundations for program synthesis. By using type theory as the basis of study for program synthesis, we are able to build core synthesis calculi for typed, functional programs, analyze the calculis meta-theoretic properties, and extend these calculi to handle increasingly richer types and language features. In addition to these foundations, we also present an implementation of these synthesis systems, Myth, that demonstrates the effectiveness of program synthesis with types on real-world code.

Which Types Have a Unique Inhabitant? Focusing on Pure Program Equivalence

GABRIEL SCHERER
Université Paris-Diderot, France

Date: March 2016; Advisor: Didier Rémy
URL: http://tinyurl.com/gsvj45s

Some programming language features (coercions, type-classes, implicits) rely on inferring a part of the code that is determined by its usage context. In order to better understand the theoretical underpinnings of this mechanism, we ask: when is it the case that there is a *unique* program that could have been guessed, or in other words that all possible guesses result in equivalent program fragments? Which types have a unique inhabitant?

To approach the question of unicity, we build on work in proof theory on more canonical representation of proofs. Using the proofs-as-programs correspondence, we can adapt the logical technique of focusing to obtain more canonical program representations.

In the setting of simply-typed lambda-calculus with sums and the empty type, equipped with the strong beta-eta-equivalence, we show that uniqueness is decidable. We present a saturating focused logic that introduces irreducible cuts on positive types "as soon as possible". Goal-directed proof search in this logic gives an effective algorithm that returns either zero, one or two distinct inhabitants for any given type.

A Framework for Relating, Implementing and Verifying Argumentation Models and Their Translations

BAS VAN GIJZEL
University of Nottingham, UK

Date: October 2015; Advisor: Henrik Nilsson
URL: http://tinyurl.com/h25uowl

Computational argumentation theory deals with the formalisation of argument structure, conflict between arguments and domain-specific constructs, such as proof standards, epistemic probabilities or argument schemes. However, despite these practical components, there is a lack of implementations and implementation methods available for most structured models of argumentation and translations between them.

This thesis addresses this problem by constructing a general framework for relating, implementing and formally verifying argumentation models and translations between them, drawing from dependent type theory and the Curry-Howard correspondence. The framework provides mathematical tools and programming methodologies to implement argumentation models, allowing programmers and argumentation theorists to construct implementations that are closely related to the mathematical definitions.

It furthermore provides tools for construction of counter-examples to desired properties. The construction is almost automatic: very little effort is required on behalf of the programmer. Finally, the dissertation provides methodologies that aid with proving formal correctness of the implementation in a theorem prover.

The thesis consists of various use cases that demonstrate the general approach of the framework. The Carneades argumentation model, Dung's abstract argumentation frameworks and a translation between them, are implemented in the functional programming language Haskell. Implementations of formal properties of the translation are provided together with a formalisation of AFs in the theorem prover, Agda. The result is a verified pipeline, from the structured model Carneades into existing efficient SAT-based implementations of Dung's AFs. Finally, the ASPIC+ model for argumentation is generalised to incorporate content orderings, weight propagation and argument accrual. The framework is applied to provide a translation from this new model into Dung's AFs, together with a complete implementation.