From Processor Verification Upwards

Three Research Vignettes
in Memory of Mike Gordon

Oxford, July 2018

Speaker: Magnus Myreen

Covering years: 2005-2014

Meeting Mike for the first time 2005

Also met: Hasan Amjad, Anthony Fox, Juliano Iyoda

Mike: I suggest you start with

A COMPUTATIONAL LOGIC

ACL2

APPLICATIVE COMMON LISP

Later: try proving some crypto-like code, e.g. bignum arithmetic

Tea at 4pm every day

a pot of tea, a box full of biscuits and a tray of small change

Often there:

Mike Gordon, Larry Paulsson, Anthony Fox, Thomas Tuerk, Scott Owens, Aaron Coble, Tjark Weber, Peter Sewell, Joe Hurd, …

but also visitors:

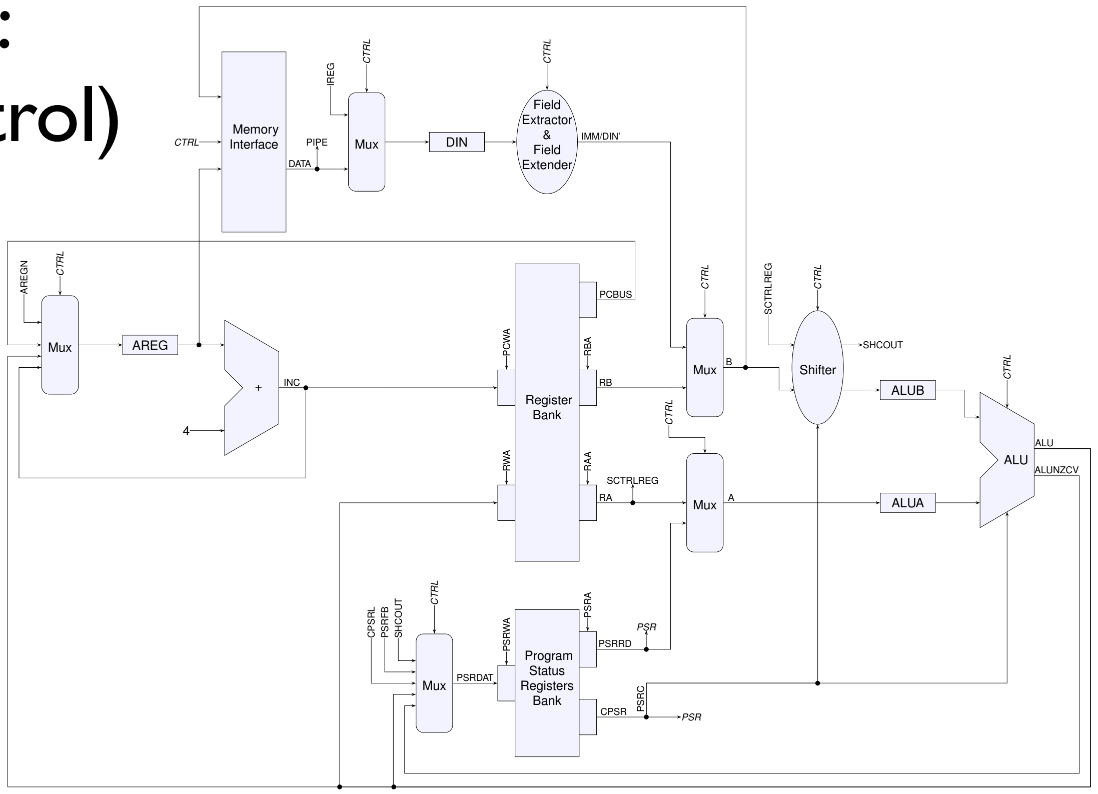
Warren Hunt,
Anna Slobodova,
Kristin Yvonne Rozier, …

ARM6 verification in HOL (Anthony Fox)

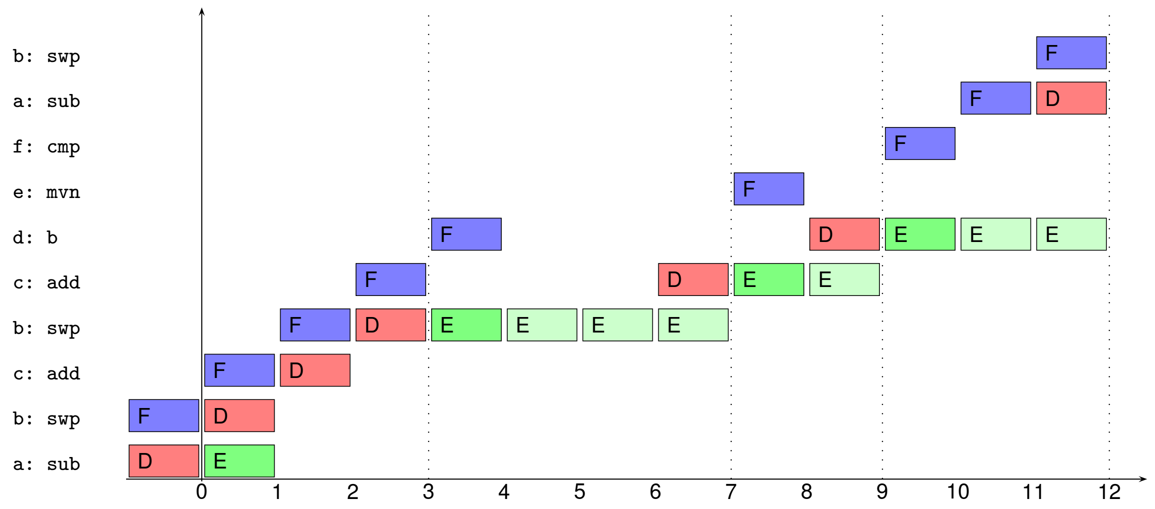
2003: End of the first project. The initial proof was complete but it lacked some features.

Late 2005: End of ARM6 verification work. The final version included features that were omitted in the first proof, e.g. multiplication, block data transfers, co-processor instructions and all interrupts/exceptions.

Datapath:
(not control)



Pipeline illustration:



Can Anthony's ARM model be used?

His tooling produced theorems that describe ARM,
e.g. ARM instruction add r0,r0,r0 is described by:

encoding of
add r0,r0,r0

```
|- (ARM_READ_MEM ((31 >< 2) (ARM_READ_REG 15w state)) state =
    0xE0800000w) ∧ ¬state.undefined ⇒
   (NEXT_ARM_MMU cp state =
     ARM_WRITE_REG 15w (ARM_READ_REG 15w state + 4w)
     (ARM_WRITE_REG 0w
      (ARM_READ_REG 0w state + ARM_READ_REG 0w state) state))
```

My attempt

An ARM program for calculating the factorial of a positive number:

```
      MOV   b, #1      ; b := 1
L:    MUL   b, a, b    ; b := a × b
      SUBS  a, a, #1   ; a := a - 1
      BNE   L          ; jump to L if a ≠ 0
```

A classical Hoare-style specification:

$$\{(a = x) \land (x \neq 0)\}$$
$$\texttt{FACTORIAL}$$
$$\{(a = 0) \land (b = x!)\}$$

Side condition:
The registers associated with a and b are distinct.

What is left unchanged?

Mike's suggestion: try separation logic

Solution based on separation logic worked!

Specification for multiplication and decrement-by-one:

$$\{R\ a\ x * R\ b\ y\}\quad \texttt{MUL}\ \ b\texttt{,}a\texttt{,}b\quad \{R\ a\ x * R\ b\ (x \cdot y)\}^{+1}$$

$$\{R\ a\ x * S\ \_\}\quad \texttt{SUB}\ \ a\texttt{,}a\texttt{,\#1}\quad \{R\ a\ (x{-}1) * S\ (x{-}1{=}0)\}^{+1}$$

proved w.r.t. Anthony's ARM specification

Composition:

$$\{R\ a\ x * R\ b\ y * S\ \_\}\quad \texttt{MUL}\ \ b\texttt{,}a\texttt{,}b\texttt{;}\ \texttt{SUB}\ \ a\texttt{,}a\texttt{,\#1}\quad \{R\ a\ (x{-}1) * R\ b\ (x \cdot y) * S\ (x{-}1{=}0)\}^{+2}$$

Mike's suggestion: try separation logic

Solution based on separation logic worked!

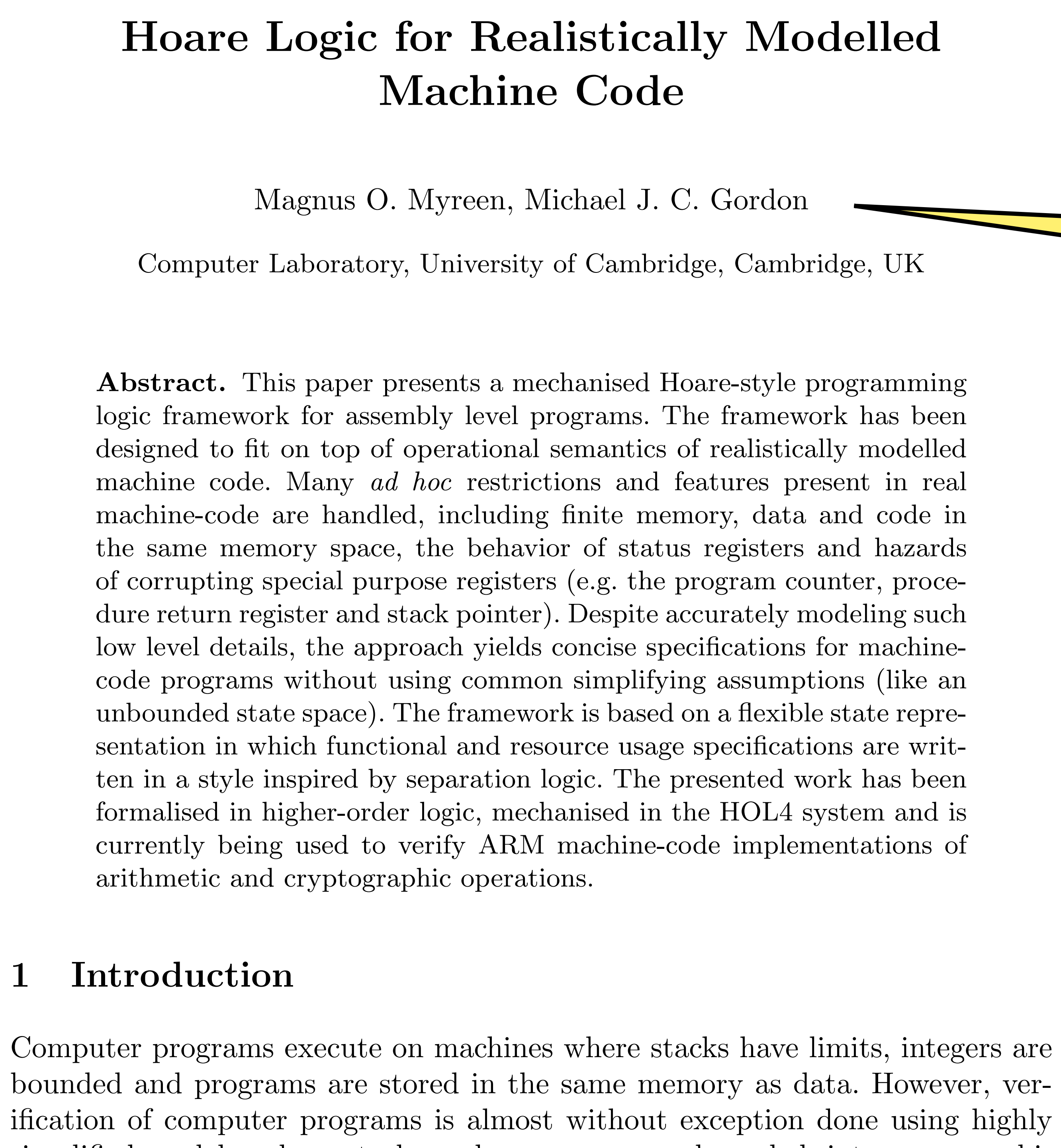
Neat definitions:

The Hoare triple's definition

$$\begin{aligned}\{p\}\,c\,\{q\} \;=\; & \forall r\ s.\ (p * \text{code}\ c * r)\ (\mathit{to\_set}(s)) \Rightarrow \\ & \quad \exists n.\ (q * \text{code}\ c * r)\ (\mathit{to\_set}(\mathit{next}^n(s)))\end{aligned}$$

My first paper during my PhD

TACAS'07



Hoare Logic for Realistically Modelled Machine Code

Magnus O. Myreen, Michael J. C. Gordon

Computer Laboratory, University of Cambridge, Cambridge, UK

Abstract. This paper presents a mechanised Hoare-style programming logic framework for assembly level programs. The framework has been designed to fit on top of operational semantics of realistically modelled machine code. Many *ad hoc* restrictions and features present in real machine-code are handled, including finite memory, data and code in the same memory space, the behavior of status registers and hazards of corrupting special purpose registers (e.g. the program counter, procedure return register and stack pointer). Despite accurately modeling such low level details, the approach yields concise specifications for machine-code programs without using common simplifying assumptions (like an unbounded state space). The framework is based on a flexible state representation in which functional and resource usage specifications are written in a style inspired by separation logic. The presented work has been formalised in higher-order logic, mechanised in the HOL4 system and is currently being used to verify ARM machine-code implementations of arithmetic and cryptographic operations.

1 Introduction

Computer programs execute on machines where stacks have limits, integers are bounded and programs are stored in the same memory as data. However, verification of computer programs is almost without exception done using highly

Mike didn't want to be a co-author (felt I had key ideas and done the work)

I insisted and Mike eventually agreed to be co-author.

Met Konrad Slind.

Konrad had an ESOP paper at the same instance of ETAPS.

Konrad visits Cambridge

Konrad had a PhD student working on proof-producing compilation to ARM code.

I worked on verification of machine code.

Mike advised me to not do verified / proof-producing compilation

… in order to too avoid competing with Konrad's PhD student.

I demoed my tools to Konrad, but he wanted more automation.

My response to Konrad's request

Example: Given some hard-to-read (ARM) machine code,

```
 0: E3A00000       mov r0, #0
 4: E3510000    L: cmp r1, #0
 8: 12800001       addne r0, r0, #1
12: 15911000       ldrne r1, [r1]
16: 1AFFFFFB       bne L
```

The decompiler produces a readable HOL4 function:

$$
\begin{aligned}
f(r_0, r_1, m) &= \text{let } r_0 = 0 \text{ in } g(r_0, r_1, m) \\
g(r_0, r_1, m) &= \text{if } r_1 = 0 \text{ then } (r_0, r_1, m) \text{ else} \\
&\qquad \text{let } r_0 = r_0{+}1 \text{ in} \\
&\qquad \text{let } r_1 = m(r_1) \text{ in} \\
&\qquad\quad g(r_0, r_1, m)
\end{aligned}
$$

My response to Konrad's request (cont.)

Decompiler automatically proves a certificate, which states that f describes the effect of the ARM code:

$$f_{pre}(r_0, r_1, m) \Rightarrow$$

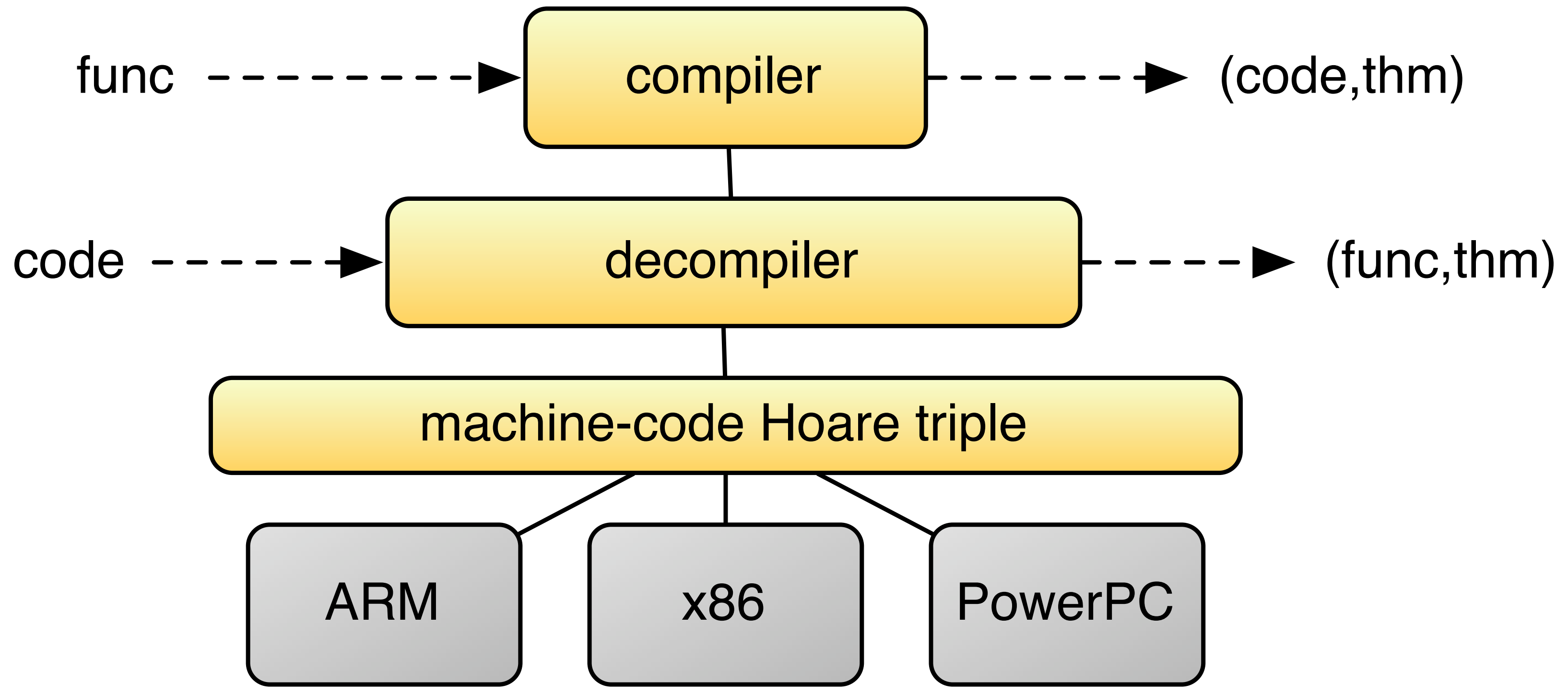
$$\{\, (\mathrm{R0}, \mathrm{R1}, \mathrm{M}) \text{ is } (r_0, r_1, m) * \mathrm{PC}\ p * \mathrm{S} \,\}$$

p : `E3A00000 E3510000 12800001 15911000 1AFFFFFB`

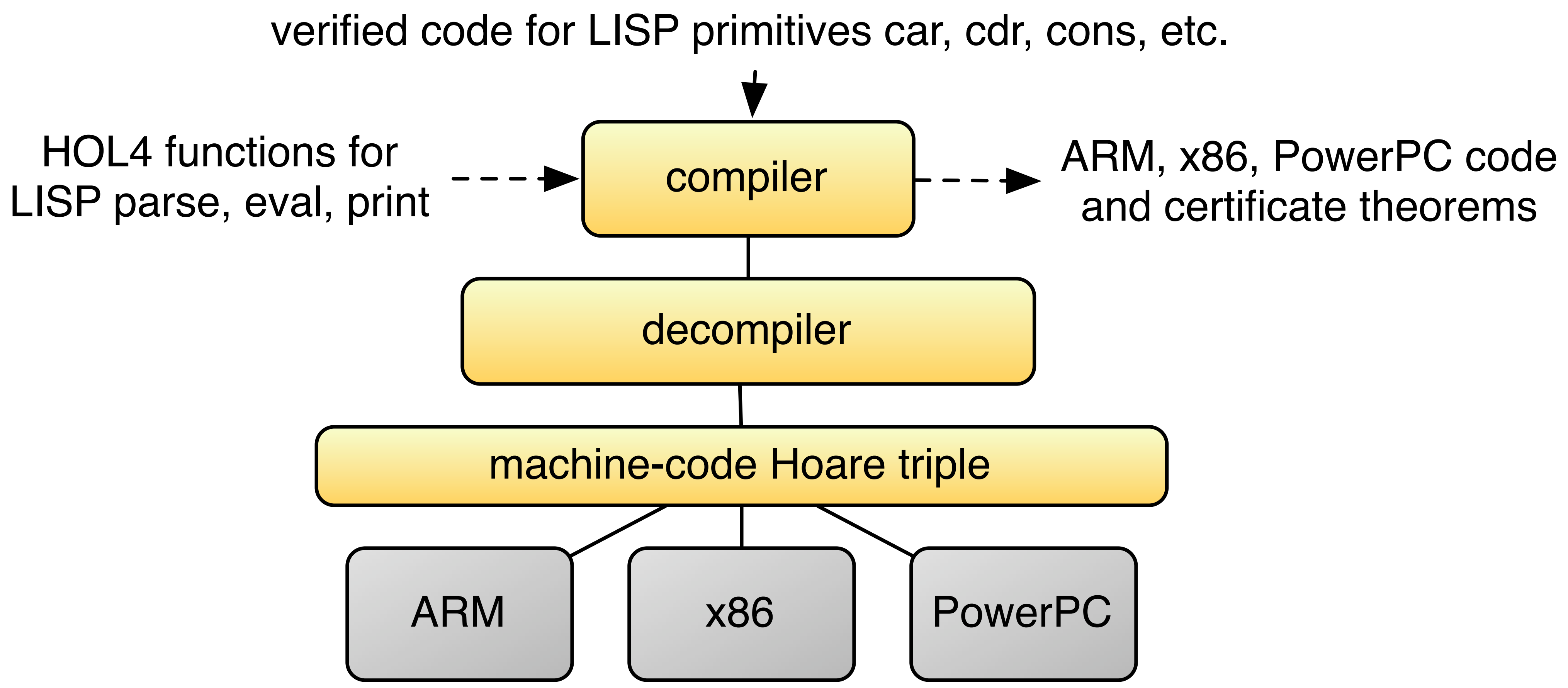
$$\{\, (\mathrm{R0}, \mathrm{R1}, \mathrm{M}) \text{ is } f(r_0, r_1, m) * \mathrm{PC}\ (p + 20) * \mathrm{S} \,\}$$

My thesis work

During my PhD, I developed the following infrastructure:



My work turns to Lisp


verified code for LISP primitives car, cdr, cons, etc.
HOL4 functions for LISP parse, eval, print
compiler
ARM, x86, PowerPC code and certificate theorems
decompiler
machine-code Hoare triple
ARM
x86
PowerPC

The final case study in my PhD thesis echos something of Mike's PhD thesis (which was about Lisp).

It was a lot of fun

Example: paper gives a definition of `pascal-triangle`, for which:

```
(pascal-triangle '((1)) '6)
```

returns:

```
((1 6 15 20 15 6 1)
 (1 5 10 10 5 1)
 (1 4 6 4 1)
 (1 3 3 1)
 (1 2 1)
 (1 1)
 (1))
```

The verified code was run on several platforms:

Nintendo DS lite (ARM)

MacBook (x86)

old MacMini (PowerPC)

EPSRC proposal

Mike and I wrote an EPSRC proposal. Mike claimed that I wrote the proposal myself, but Mike edited significantly.

Proposal accepted!

4 years of freedom

Mike was very hands off by now, but suggested I apply ideas from my thesis

- Single-author POPL paper on self-modifying code / JIT
- Collaboration with seL4 team at NICTA
- Joint work with Jared Davis on Milawa prover (Lisp)

a reflective ACL2-like prover with a novel minimal trusted kernel

More about Mike's influence

certicom™
securing innovation

Mike arranged for me to visit
a Canadian crypto company
(accompanied by Peter Homeier)

Mike managed to get Xavier Leroy
to be the examiner of my PhD
thesis in 2008 (viva 2009).

(timely due to CompCert POPL'06)

Approach: *create collaboration instead of competition*

Mike’s other PhD students 2005-2014

Juliano Iyoda

James Reynolds

Alexey Gotsman

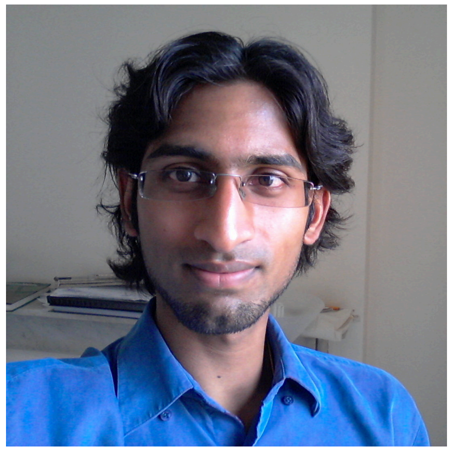
Thomas Tuerk

Eric Koskinen

Matko Botincan

Ramana Kumar

Mike’s last PhD student: Ramana Kumar



Started his PhD in the autumn of 2011.

Strong drive to do collaborative work that would produce results that last.

Context:

Around this time, Scott Owens and I published an ICFP paper on *Proof-Producing Synthesis of ML from HOL*

Also: Freek Wiedijk had asked me at ITP’11:
“Can you do for HOL Light what you did for Milawa?”

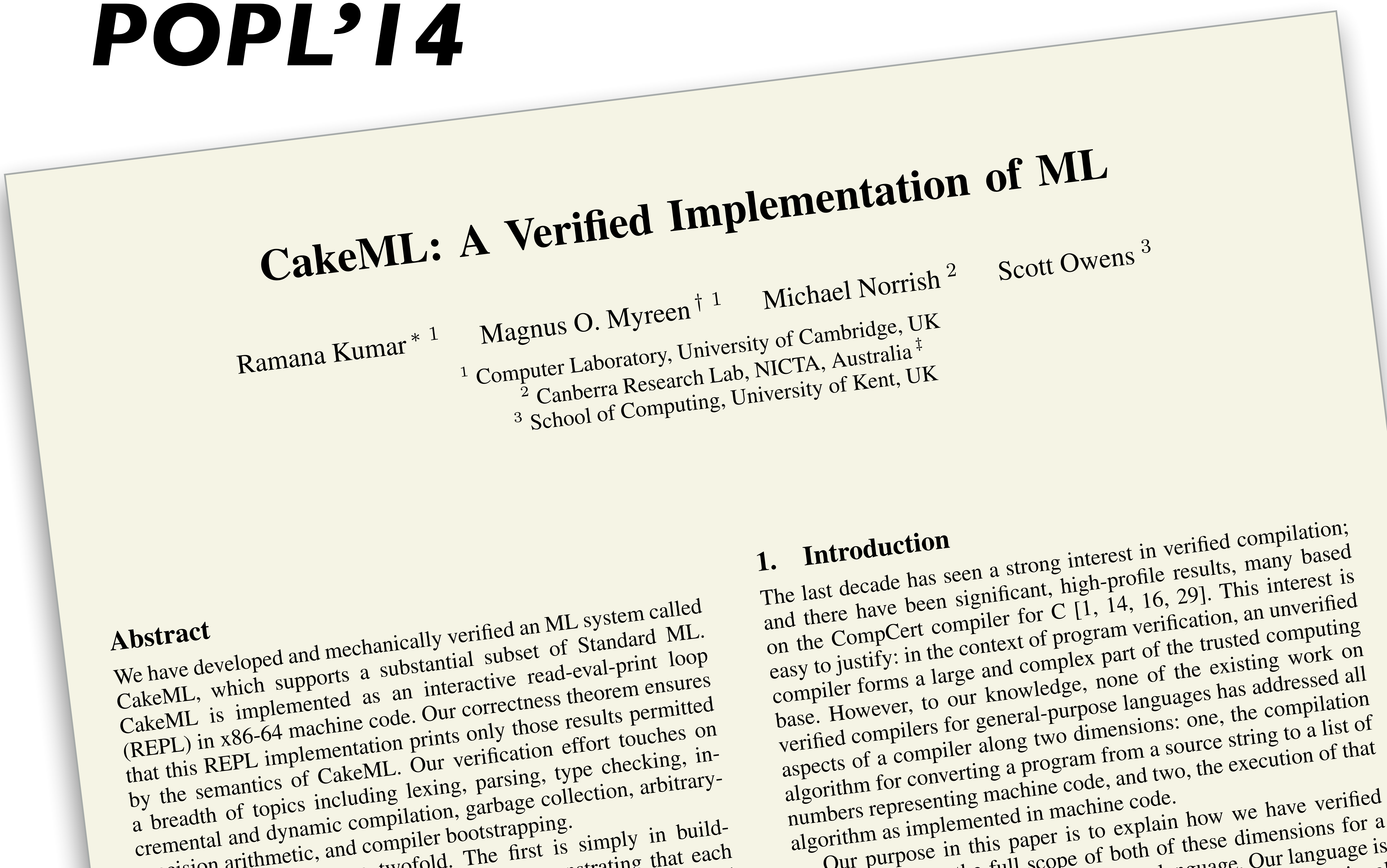
Michael had recent work on verified parsing.

$\longrightarrow$ The CakeML project started.

CakeML's first major result

(Mike liked this result.)

POPL'14



CakeML: A Verified Implementation of ML

Ramana Kumar [*] [1] Magnus O. Myreen [†] [1] Michael Norrish [2] Scott Owens [3]

[1] Computer Laboratory, University of Cambridge, UK
[2] Canberra Research Lab, NICTA, Australia [‡]
[3] School of Computing, University of Kent, UK

Abstract

We have developed and mechanically verified an ML system called CakeML, which supports a substantial subset of Standard ML. CakeML is implemented as an interactive read-eval-print loop (REPL) in x86-64 machine code. Our correctness theorem ensures that this REPL implementation prints only those results permitted by the semantics of CakeML. Our verification effort touches on a breadth of topics including lexing, parsing, type checking, incremental and dynamic compilation, garbage collection, arbitrary-precision arithmetic, and compiler bootstrapping.

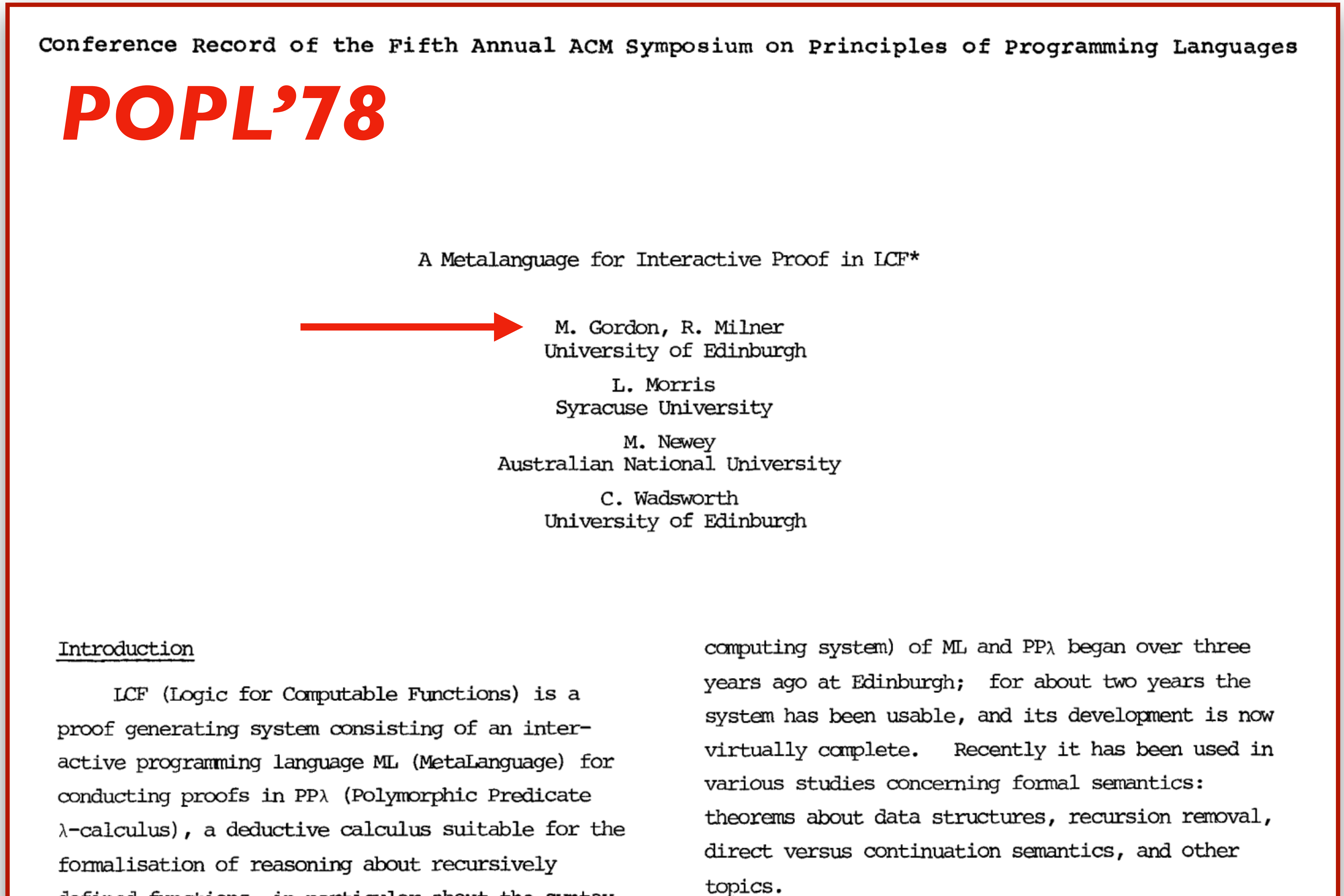
twofold. The first is simply in build-
nstrating that each

1. Introduction

The last decade has seen a strong interest in verified compilation; and there have been significant, high-profile results, many based on the CompCert compiler for C [1, 14, 16, 29]. This interest is easy to justify: in the context of program verification, an unverified compiler forms a large and complex part of the trusted computing base. However, to our knowledge, none of the existing work on verified compilers for general-purpose languages has addressed all aspects of a compiler along two dimensions: one, the compilation algorithm for converting a program from a source string to a list of numbers representing machine code, and two, the execution of that algorithm as implemented in machine code.

Our purpose in this paper is to explain how we have verified
full scope of both of these dimensions for a
language. Our language is

… connection to the original paper on ML:



Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages

POPL'78

A Metalanguage for Interactive Proof in LCF*

M. Gordon, R. Milner
University of Edinburgh

L. Morris
Syracuse University

M. Newey
Australian National University

C. Wadsworth
University of Edinburgh

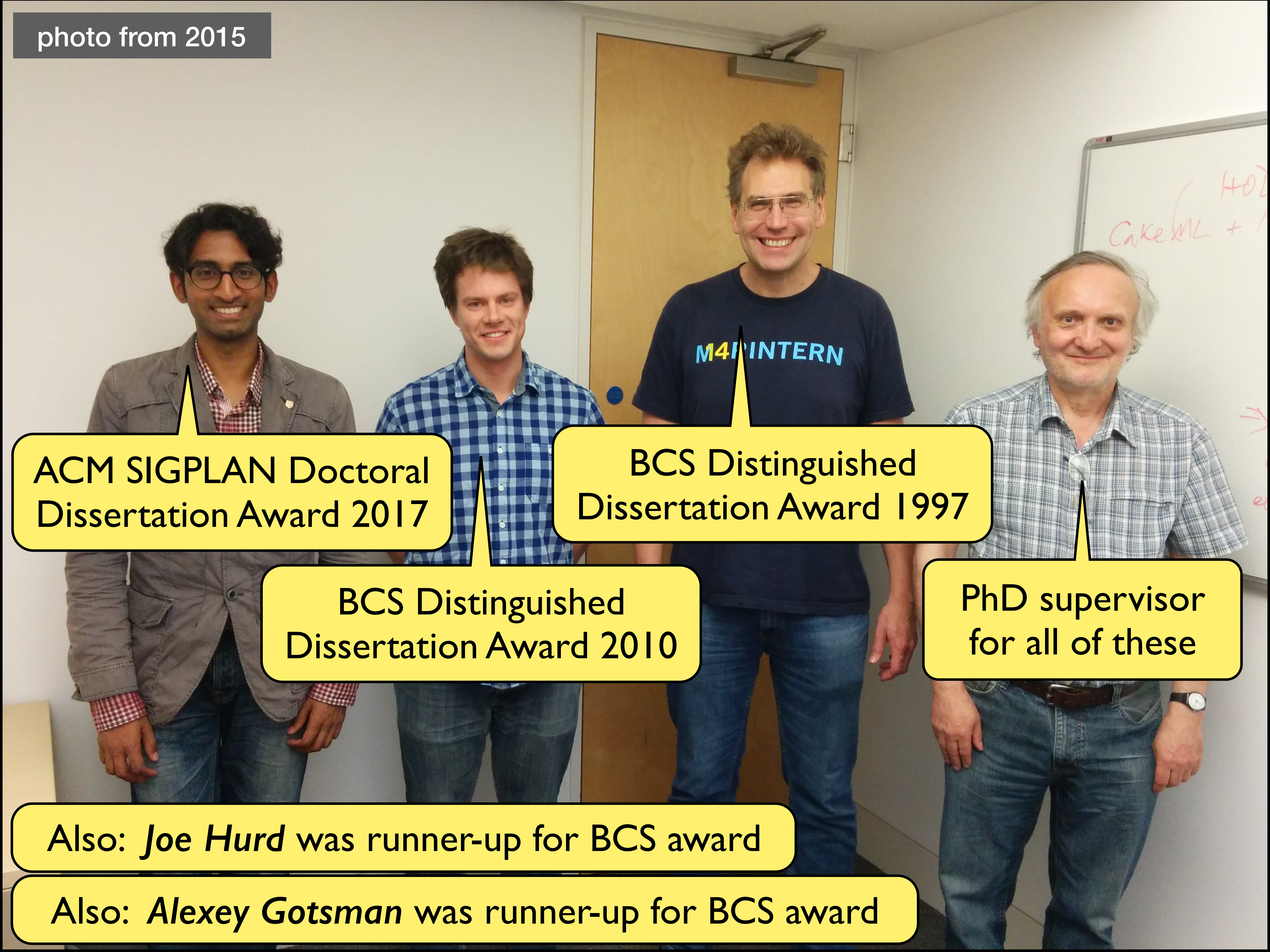
Introduction

LCF (Logic for Computable Functions) is a proof generating system consisting of an interactive programming language ML (MetaLanguage) for conducting proofs in PPλ (Polymorphic Predicate λ-calculus), a deductive calculus suitable for the formalisation of reasoning about recursively

computing system) of ML and PPλ began over three years ago at Edinburgh; for about two years the system has been usable, and its development is now virtually complete. Recently it has been used in various studies concerning formal semantics: theorems about data structures, recursion removal, direct versus continuation semantics, and other topics.

photo from 2015

photo from 2015


ACM SIGPLAN Doctoral Dissertation Award 2017
BCS Distinguished Dissertation Award 2010
BCS Distinguished Dissertation Award 1997
PhD supervisor for all of these

Also: *Joe Hurd* was runner-up for BCS award

Also: *Alexey Gotsman* was runner-up for BCS award