Theoretical Theorem Proving

or An L4.verified Roundup

Thomas Sewell

NICTA

24 July 2010

Overview

1. Calculemus!
2. seL4
3. L4.verified artefacts
4. Induction
5. L4.verified proofs
   - Correspondence
   - Invariants
6. Big questions:
   - Is Isabelle/HOL a good functional program?
   - Why was it hard?
   - What's seL4 good for? Who can we sell it to?
   - What's L4.verified good for? Are there still bugs?
   - How would I do it differently?

Customizable!

Calculemus

Calculemus!
Dijkstra used to finish his papers with the injunction “Calculemus!”

Calculemus

Calculemus!
Dijkstra used to finish his papers with the injunction "Calculemus!"

Definition:
"Let us calculate!" This way, the 18th century German philosopher Gottfried Leibniz expressed the hope to provide a method that would allow people to settle their differences by putting their problems in a formal language *lingua universalis* and then finding out who is right by mechanically applying a simple system of formal rules *calculus ratiocinator*.

Calculemus

Calculemus!
Dijkstra used to finish his papers with the injunction "Calculemus!"

Definition:
"Let us calculate!" This way, the 18th century German philosopher Gottfried Leibniz expressed the hope to provide a method that would allow people to settle their differences by putting their problems in a formal language *lingua universalis* and then finding out who is right by mechanically applying a simple system of formal rules *calculus ratiocinator*.

If you strike out the word 'universalis' that's a pretty good definition of theorem proving.

Liebnitz to Dijkstra

Between Liebnitz and Dijkstra:

- Hilbert's 100 problems to be solved in the 20th century was full of problems that were part of this grand design.

Liebnitz to Dijkstra

Between Liebnitz and Dijkstra:

- Hilbert's 100 problems to be solved in the 20th century was full of problems that were part of this grand design.
- Gödel's famous incompleteness theorem scuttled the effort.

Liebnitz to Dijkstra

Between Liebnitz and Dijkstra:

- Hilbert's 100 problems to be solved in the 20th century was full of problems that were part of this grand design.
- Gödel's famous incompleteness theorem scuttled the effort.
- Turing's machines killed it permanently.

Liebnitz to Dijkstra

Between Liebnitz and Dijkstra:

- Hilbert's 100 problems to be solved in the 20th century was full of problems that were part of this grand design.
- Gödel's famous incompleteness theorem scuttled the effort.
- Turing's machines killed it permanently.
- Resurrected in the context of programming in the sixties and seventies.

Liebnitz to Dijkstra

Between Liebnitz and Dijkstra:

- Hilbert's 100 problems to be solved in the 20th century was full of problems that were part of this grand design.
- Gödel's famous incompleteness theorem scuttled the effort.
- Turing's machines killed it permanently.
- Resurrected in the context of programming in the sixties and seventies.
- Never quite gone out of fashion, but never become a big thing either.

Who Cares?

The amended grand effort is to reason logically *about programs*. This aims to solve the grand problem of software falling apart all over the place. Who cares?

Who Cares?

The amended grand effort is to reason logically *about programs*. This aims to solve the grand problem of software falling apart all over the place. Who cares?

- Logic of computation tragics.

Who Cares?

The amended grand effort is to reason logically *about programs*. This aims to solve the grand problem of software falling apart all over the place. Who cares?

- Logic of computation tragics. My boss.

Who Cares?

The amended grand effort is to reason logically *about programs*. This aims to solve the grand problem of software falling apart all over the place. Who cares?

- Logic of computation tragics. My boss.
- Functional programmers of various kinds.

Who Cares?

The amended grand effort is to reason logically *about programs*. This aims to solve the grand problem of software falling apart all over the place. Who cares?

- Logic of computation tragics. My boss.
- Functional programmers of various kinds. Robin Milner

Who Cares?

The amended grand effort is to reason logically *about programs*. This aims to solve the grand problem of software falling apart all over the place. Who cares?

- Logic of computation tragics. My boss.
- Functional programmers of various kinds. Robin Milner, Ben Lippmeier

Who Cares?

The amended grand effort is to reason logically *about programs*. This aims to solve the grand problem of software falling apart all over the place. Who cares?

- Logic of computation tragics. My boss.
- Functional programmers of various kinds. Robin Milner, Ben Lippmeier, Haskell Curry and William Howard

Who Cares?

The amended grand effort is to reason logically *about programs*. This aims to solve the grand problem of software falling apart all over the place. Who cares?

- Logic of computation tragics. My boss.
- Functional programmers of various kinds. Robin Milner, Ben Lippmeier, Haskell Curry and William Howard
- Me.

Who Cares?

The amended grand effort is to reason logically *about programs*. This aims to solve the grand problem of software falling apart all over the place. Who cares?

- Logic of computation tragics. My boss.
- Functional programmers of various kinds. Robin Milner, Ben Lippmeier, Haskell Curry and William Howard
- Me.
- OS groups

Microkernels

The kernel of an OS is the part that runs with the CPU in privileged mode.

Microkernels

The kernel of an OS is the part that runs with the CPU in privileged mode.

Privileged mode is:

1. the only way the kernel can control key hardware, for instance to time switch between applications.

Microkernels

The kernel of an OS is the part that runs with the CPU in privileged mode.

Privileged mode is:

1. the only way the kernel can control key hardware, for instance to time switch between applications.
2. one of many ways that a service can be provided to applications really fast.

Microkernels

The kernel of an OS is the part that runs with the CPU in privileged mode.

Privileged mode is:

1. the only way the kernel can control key hardware, for instance to time switch between applications.
2. one of many ways that a service can be provided to applications really fast.

A new idea, or maybe a very old idea, is to focus on (1), giving us microkernels.

Issues with Microkernels

Things the microkernel community needed to demonstrate:

1. Performance.
2. Quality.
3. Usability.

Issues with Microkernels

Things the microkernel community needed to demonstrate:

1. Performance.
2. Quality.
3. Usability.

Ever tried to build a build system?

Issues with Microkernels

Things the microkernel community needed to demonstrate:

1. Performance.
2. Quality.
3. Usability.

Ever tried to build a build system?

Lots of interest in verification of microkernels: L4.verified, VFiasco, Verisoft, Kit, FLINT.

seL4 Design

seL4 is a "fourth generation" microkernel. Interesting features:

- EROS-style capabilities for more or less everything.
- Capabilities are used to manage allocation of kernel memory.
- CSpace: capabilities to capabilities.
- Guts-out.
- High-speed L4-style messaging.
- Delegation.
- Big, complicated, expensive revoke operation.

seL4 Artefacts

The seL4 kernel was originally implemented as a Haskell prototype.

The Haskell prototype was simplified into an Isabelle/HOL specification.

The Haskell prototype was also hand-translated to produce the C implementation. Once the implementation was working the prototype fell out of common use.

L4.verified artefacts

See another set of slides.

Induction

I wanted to talk about induction for a couple of reasons.

Induction

I wanted to talk about induction for a couple of reasons.

Proof work gets hard when we become both producer and consumer.

L4.verified Proof Overview

See another set of slides.

What was hard?

What was hard?

What was hard?

What was hard?

1. Induction.

What was hard?

What was hard?

1. Induction.
2. Scale of the problem.

What was hard?

What was hard?

1. Induction.
2. Scale of the problem.
3. Complexity of seL4.

Is Isabelle a good functional program?

Is Isabelle a good functional program?

Is Isabelle a good functional program?

Is Isabelle a good functional program?

No

*** ML ***
* Classical tactics use proper Proof.context instead of historic types claset/clasimpset. Old-style declarations like addIs, addEs, addDs operate directly on Proof.context. Raw type claset retains its use as snapshot of the classical context, which can be recovered via (put_claset HOL_cs) etc. Type clasimpset has been discontinued. INCOMPATIBILITY, classical tactics and derived proof methods require proper Proof.context.

Makarius

What is seL4 good for?

What is seL4 good for?

We're not exactly sure.

What is seL4 good for?

We're not exactly sure.

Use in most industrial safety settings (aerospace, automotive, medical ...) would require more than a safe kernel. It would require a software engineering framework for building a larger safe system.

What is seL4 good for?

We're not exactly sure.

Use in most industrial safety settings (aerospace, automotive, medical ...) would require more than a safe kernel. It would require a software engineering framework for building a larger safe system. See the usability problem with microkernels.

What is seL4 good for?

We're not exactly sure.

Use in most industrial safety settings (aerospace, automotive, medical ...) would require more than a safe kernel. It would require a software engineering framework for building a larger safe system. See the usability problem with microkernels.

Use in secrecy applications would only require a safe kernel.

What is seL4 good for?

We're not exactly sure.

Use in most industrial safety settings (aerospace, automotive, medical ...) would require more than a safe kernel. It would require a software engineering framework for building a larger safe system. See the usability problem with microkernels.

Use in secrecy applications would only require a safe kernel. But our proof doesn't say anything about information flow and the hardware would probably shoot us in the foot.

What is seL4 good for?

We're not exactly sure.

Use in most industrial safety settings (aerospace, automotive, medical ...) would require more than a safe kernel. It would require a software engineering framework for building a larger safe system. See the usability problem with microkernels.

Use in secrecy applications would only require a safe kernel. But our proof doesn't say anything about information flow and the hardware would probably shoot us in the foot.

Also CSpaces are hard.

What is the proof good for?

What is the proof good for? Did we find any bugs?

What is the proof good for?

What is the proof good for? Did we find any bugs?

Yes.

How would you do it differently?