- **Regan Robertson** Mod • 14 days ago
  Good Afternoon,
  The video will start on this page at 15:00pm, and you may have to hit play or unmute. As a reminder this site works best in a Chrome Browser. You can write in comments through this feature to continue the conversation or ask questions. Please login or sign up for a Disqus account to participate in the discussion boards.
  - Reply
  - Share ›

Matthew Brecknell • 14 days ago
Good morning from Sydney! (And good afternoon/evening in other places!)
- 2
- Reply
- Share ›

Renato Levy Matthew Brecknell • 14 days ago
Good Afternoon Matthew
- Reply
- Share ›

Curtis Millar Matthew Brecknell • 14 days ago

Morning

- Reply
- Share ›

Matthew Brecknell • 14 days ago

If folks are struggling with the image quality in the live version, I've posted a copy here:

https://youtu.be/AdakDMYu4lM

Nathaniel Husted Matthew Brecknell • 14 days ago

Awesome! I appreciate the ability to go back and watch again as well.

- 2
- Reply
- Share ›

Kyle Tillotson Matthew Brecknell • 14 days ago

Thank you

Reply

Share ›

Nathaniel Husted • 14 days ago

As someone who's experience with theorem provers is hanging out with formal methods folks, I find it really fascinating how syntactically there's a lot of commonality with modern program languages (e.g., Rust) that makes a lot of the concepts easier to follow. I wonder if this is following one of the themes from this week that strong type systems have become well integrated with modern languages?

Reply

Share ›

Raymond Richards Nathaniel Husted • 14 days ago • edited

Once you have completed formal verification, which often means you have completed proofs with respect some model of the system, you are left with convincing people that your model and proofs are relevant. By this I mean, that they correspond with reality. The NSA call the activity of validating that the model is a faithful abstraction of the system a "code to spec" review. I have spent weeks comparing a formal model to an implementation with a conference room of NSA evaluators. Having a syntax that easily relates to an implementation is useful.

1

Reply

Share ›

Matthew Brecknell Raymond Richards • 14 days ago

Of course, we side-step this question by proving that the implementation is related to the spec, via refinement. :-)

There's still the question of whether the specification defines the thing that you actually wanted. Doing lots of proofs about the specification (e.g. invariants) is one way to get that confidence.

Reply

Share ›

Raymond Richards Matthew Brecknell • 14 days ago

Yes, the proofs have to show something meaningful about the system, or what's the point. One approach to this is to use the properties you are proving as axioms in some higher level system, demonstrating that your properties providing value to the larger system.

Reply

Share ›

Renato Levy Nathaniel Husted • 14 days ago • edited

I agree, leaves the feeling that if you program correctly, it should be easier to prove assumptions about your code....

- 1
- Reply
- Share ›

Lennart Beringer Nathaniel Husted • 14 days ago

Yes, it's no coincidence that modern proof assistants have strong relationships to modern programming languages: they embed functional specification languages (ML/Haskell-based) and are themselves implemented in such languages. Proof assistants differ a bit in the degree to which type concepts such as linearity, capabilities, dependent types,.. are built into the ambient logic (and not all of them are equally useful in all use cases).

- 1
- Reply
- Share ›

Bo Gan • 14 days ago

Hi Matthew, how do we deal with concurrency? E.g., in some TOCTTOU cases, the object/value we are checking could change when reached from multiple code paths. Does l4v Isabelle/HOL proof

take such into account? Forgive me if this is silly, as I don't have much knowledge about formal methods.

Reply

Share ›

Matthew Brecknell Bo Gan • 14 days ago

Good question! The verification currently only covers single-core seL4, so we effectively assume that the only changes we will see are the ones we performed ourselves.

We are, however, working on the verification of multi-core seL4. This is quite a major change to the verification, so it will take some time to complete. In particular, it requires changing the underlying model of execution. We replace the state monad with a "trace" monad that allows for interleaved executions:

https://github.com/seL4/l4v...

At the same time, we want to reuse as many of the existing proofs as possible, so I believe we have some tricks that give us the same proof rules for the trace monad that we had for the state monad, if we know that we're under a lock that gives us exclusive access to the relevant state. I'm not so familiar with that part of the multi-core verification, though.

Reply

Share ›

Axel Heider • 14 days ago

Great presentation, thanks!

Reply

Share ›

Lennart Beringer Axel Heider • 14 days ago

+1

Reply

Share ›

Olin Sibert • 14 days ago

This was a great presentation, but as someone for whom the practice of formal methods is but a distant memory, I'm afraid the Isabelle/HOL quick introduction was way too quick for me. Can you provide a list of documents, books, lectures, exercised etc., that can help provide some grounding in the tools used here?

Reply

Share ›

Matthew Brecknell Olin Sibert • 14 days ago

Hi Olin, thanks for the feedback! It's certainly tough to get from no recent formal methods to reading seL4 proofs in the space of an hour.

The book that I used to learn Isabelle/HOL (and that we give to new proof engineers in the Trustworthy Systems group) is Concrete Semantics by Tobias Nipkow and Gerwin Klein:

http://www.concrete-semanti...
The first five chapters are enough for a basic grounding in Isabelle/HOL. More chapters are good for practising on various applications. Chapter 12, in particular, covers Hoare logic, although the treatment is a bit different to what we do in the l4v proofs. Beware that the book doesn't give every detail of how to use Isabelle/HOL, so you do have to use the Isabelle/Isar reference manual a bit to fill in those details:

https://isabelle.in.tum.de/...
Or just ping me at https://twitter.com/mbrcknl and I'll see if I can help.
Also beware that it takes a while to get an intuition of which proof automation methods are most likely to work in a given situation.

We often find that people who have some experience with functional programming in a language like Haskell have an easier time following the way we construct the specification. If you've mainly been writing C code, then it might be more challenging.

- Reply
- Share ›