Open problems from system software verification

Peter Sewell
University of Cambridge

Iris Workshop, Zurich 2024-06-03

This work was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (AdG grant 789108 ELVER), the UK Government Industrial Strategy Challenge Fund (ISCF) under the Digital Security by Design (DSbD) Programme, to deliver a DSbDtech enabled digital platform (grant 105694), EPSRC programme grant EP/K008528/1 REMS, Arm Limited, Google, Google DeepMind, Arm iCASE awards, EPSRC IAA KTF funding, the Isaac Newton Trust, and the Gates Cambridge Trust. Approved for public release; distribution is unlimited. This work was supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contracts FA8750-10-C-0237 ("CTSRD"), FA8750-11-C-0249 ("MRC2"), HR0011-18-C-0016 ("ECATS"), FA8650-18-C-7809 ("CIFV"), and HR0011-22-C-0110 ("ETC"), as part of the DARPA CRASH, MRC, and SSITH research programs. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

open postdoc position in C verif

Open problems from system software verification

Peter Sewell
University of Cambridge

Iris Workshop, Zurich 2024-06-03

This work was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (AdG grant 789108 ELVER), the UK Government Industrial Strategy Challenge Fund (ISCF) under the Digital Security by Design (DSbD) Programme, to deliver a DSbDtech enabled digital platform (grant 105694), EPSRC programme grant EP/K008528/1 REMS, Arm Limited, Google, Google DeepMind, Arm iCASE awards, EPSRC IAA KTF funding, the Isaac Newton Trust, and the Gates Cambridge Trust. Approved for public release; distribution is unlimited. This work was supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contracts FA8750-10-C-0237 ("CTSRD"), FA8750-11-C-0249 ("MRC2"), HR0011-18-C-0016 ("ECATS"), FA8650-18-C-7809 ("CIFV"), and HR0011-22-C-0110 ("ETC"), as part of the DARPA CRASH, MRC, and SSITH research programs. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

When is it worth doing semantics?

When is it worth doing semantics?

- for existing pervasive abstractions

When is it worth doing semantics?

- for existing pervasive abstractions
- to explore paths to a better world

When is it worth doing semantics?

- for existing pervasive abstractions
- to explore paths to a better world
- for fun

When is it worth doing semantics?

- for existing pervasive abstractions
- to explore paths to a better world
- for fun

When is it worth it – and possible – to do verification?

When is it worth doing semantics?

- for existing pervasive abstractions
- to explore paths to a better world
- for fun

When is it worth it – and possible – to do verification?

- interactive full verification for small critical software – especially *systems* software

When is it worth doing semantics?

- for existing pervasive abstractions
- to explore paths to a better world
- for fun

When is it worth it – and possible – to do verification?

- interactive full verification for small critical software – especially *systems* software
- more automated methods (bug-finding analysis, type systems, ...) for larger-scale software

When is it worth doing semantics?

- for existing pervasive abstractions
- to explore paths to a better world
- for fun

When is it worth it – and possible – to do verification?

- interactive full verification for small critical software – especially *systems* software
- more automated methods (bug-finding analysis, type systems, ...) for larger-scale software
- to explore paths to a better world

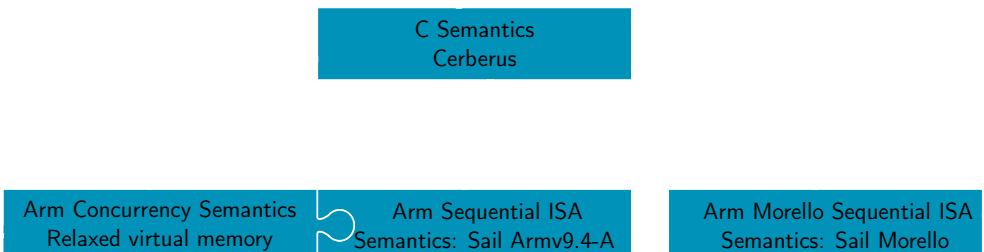
When is it worth doing semantics?

- for existing pervasive abstractions
- to explore paths to a better world
- for fun

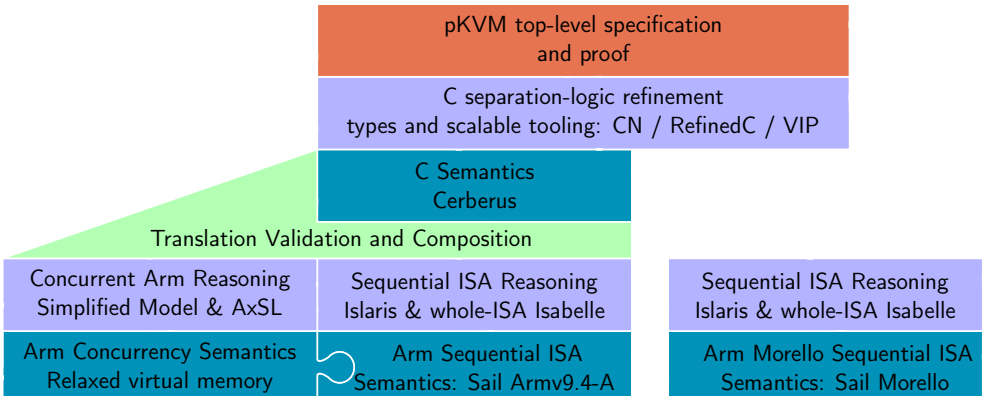
When is it worth it – and possible – to do verification?

- interactive full verification for small critical software – especially *systems* software
- more automated methods (bug-finding analysis, type systems, ...) for larger-scale software
- to explore paths to a better world
- for fun

Can we do semantics for real systems? (Arm, RISC-V, C, Morello)? (WIP!)



Can we do verification for real systems software? (Arm, RISC-V, C, pKVM, Morello)? (WIP!)



pKVM and Morello verification people

Cambridge
Peter Sewell
Neel Krishnaswami
Alasdair Armstrong
Rini Banerjee
Thomas Bauereiss
Thomas Fourier
Angus Hammond
David Kaloper-Meršinjak
Dhruv Makwana
Kayvan Memarian
Thibaut Pérami
Christopher Pulte
Thomas Sewell
Ben Simner
Robert Watson

Aarhus
Lars Birkedal
Jean Pichon-Pharabod
Zongyuan Liu

MPI-SWS
Derek Dreyer
Deepak Garg
Michael Sammler→ETH,ISTA
Rodolphe Lepigre
Laila Elbeheiry
Youngju Song
Paul Zhu

Edinburgh
Ian Stark
Ohad Kammar
Brian Campbell
Ricardo Almeida

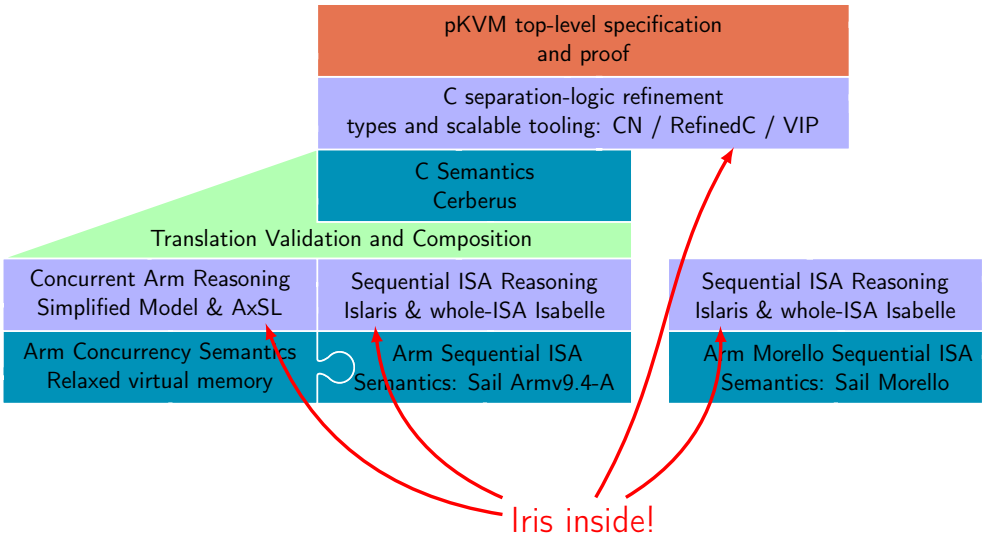
Radboud
Robbert Krebbers
Ike Mulder
Marc Hermes
Pierre Goutagny
Malo Jaffre

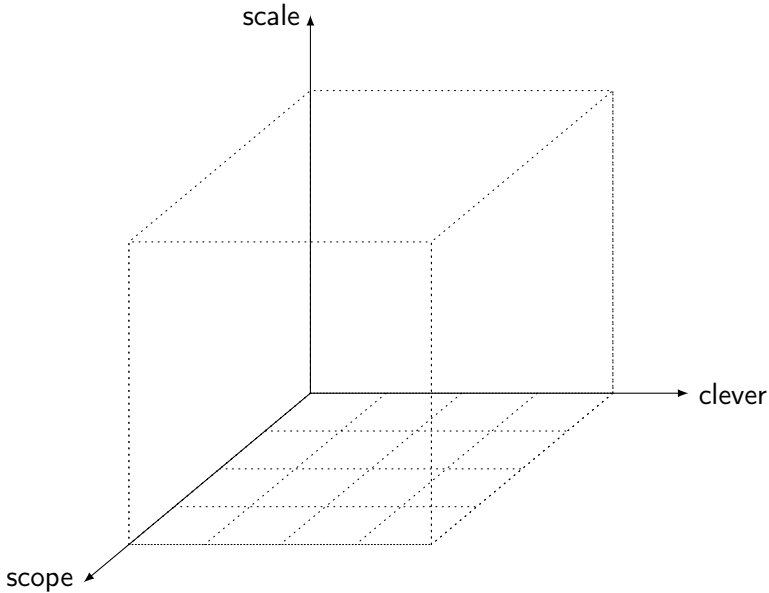
SNU
Chung-Kil (Gil) Hur
Yeji Han
Hyunwoo Lee

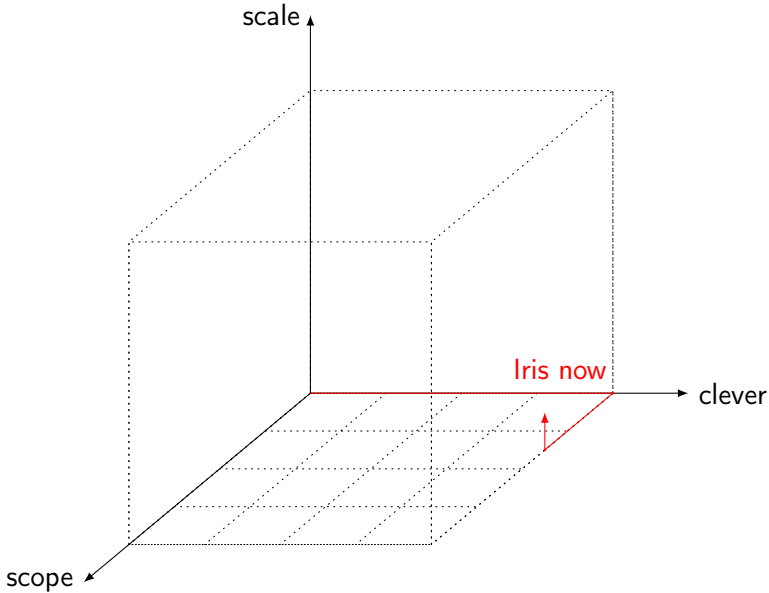
Google Android KVM
Will Deacon
Evi Karakozoglou
David Brazdil
Keir Fraser
Quentin Perret
Andrew Scull
Marc Zyngier
...

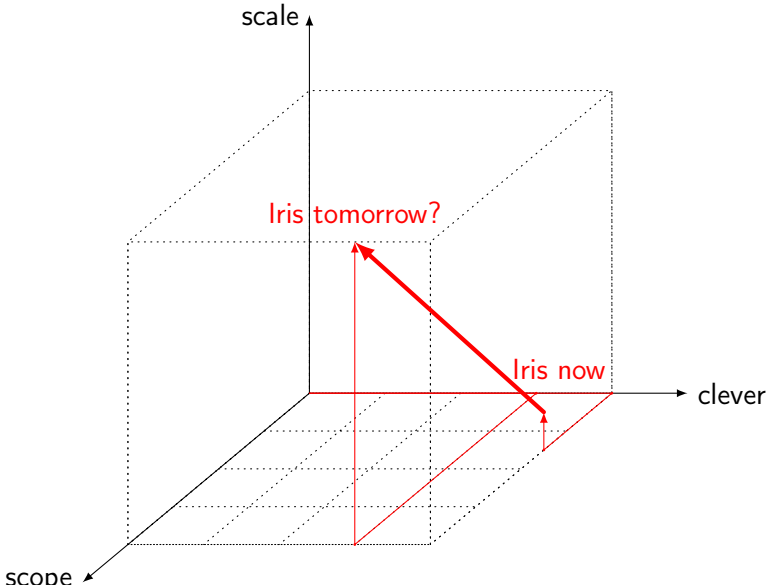
Google Project Oak
Sarah de Haas
Ben Laurie
Hong-Seok Kim
Jieung Kim

Can we do verification for real systems software? (Arm, RISC-V, C, pKVM, Morello)? (WIP!)




scale
clever
scope


scale
Iris now
clever
scope


scale
Iris tomorrow?
Iris now
clever
scope

Bottom-up

Things we like to pretend are true...

...but aren't – at the architecture level of abstraction

Bottom-up

Things we like to pretend are true...

...but aren't – at the architecture level of abstraction

...can we prove that they're true, in some circumstances?

Programs are executed in order – with a structural operational semantics

In the golden age:

$$\text{III}\rightarrow \quad (x_i := e), M \rightarrow \underline{\text{null}}, M[\underline{n}/i] \text{ if } e, M \overset{*}{\rightarrow} \underline{n}, M.$$

$$\text{IV}\rightarrow \quad (p_1; p_2), M \rightarrow (p_1'; p_2), M' \text{ if } p_1, M \rightarrow p', M'$$

$$(\underline{\text{null}}; p), M \rightarrow p, M.$$

[Program Semantics and mechanized proof, Milner 1976]

Semantics as a *transition system* of some kind

Defined more-or-less compositionally in the syntactic structure of the program

And we can build program logics for compositional reasoning above that

But...

Machine-code programs don't have a lot of syntactic structure

```
00006aac:  a9be7bfd  stp  x29, x30, [sp, #-32]!
00006ab0:  f9000bf3  str  x19, [sp, #16]
00006ab4:  910003fd  mov  x29, sp
00006ab8:  900000e8  adrp  x8, 22000 <overflow_stack+0xef0>
00006abc:  900000e9  adrp  x9, 22000 <overflow_stack+0xef0>
00006ac0:  f940b113  ldr  x19, [x8, #352]
00006ac4:  f940b929  ldr  x9, [x9, #368]
00006ac8:  cb130129  sub  x9, x9, x19
00006acc:  f140053f  cmp  x9, #0x1, lsl #12
00006ad0:  54000062  b.cs  6adc <hyp_early_alloc_page+0x30>  // b.hs, b.nlast
00006ad4:  aa1f03f3  mov  x19, xzr <- 00006ad0(b.cc-succ)<fallthrough>
00006ad8:  14000007  b  6af4 <hyp_early_alloc_page+0x48>
>00006adc:  91400669  add  x9, x19, #0x1, lsl #12 <- 00006ad0(b.cc)<hyp_early_alloc_page+0x30>
00006ae0:  aa1303e0  mov  x0, x19
00006ae4:  2a1f03e1  mov  w1, wzr
00006ae8:  52820002  mov  w2, #0x1000                  // #4096
00006aec:  f900b109  str  x9, [x8, #352]
00006af0:  940047b4  bl  189c0 <__memset>
>00006af4:  aa1303e0  mov  x0, x19 <- 00006ad8(b)<hyp_early_alloc_page+0x48>,00006af0(bl-succ)<return>
00006af8:  f9400bf3  ldr  x19, [sp, #16]
00006afc:  a8c27bfd  ldp  x29, x30, [sp], #32
00006b00:  d65f03c0  ret
```

Machine instructions aren't simple atomic primitives

Arm definition of the intra-instruction behaviour, of each instruction in isolation:

400k lines of ASL/Sail

(a first-order program that does uninterpreted register and memory accesses)

Machine instructions aren't simple atomic primitives

Arm definition of the intra-instruction behaviour, of each instruction in isolation:

400k lines of ASL/Sail

(a first-order program that does uninterpreted register and memory accesses)

How can we reason about that? For arbitrary code:

- Proof of "simple" properties about the complete definition, in Isabelle (Arm Morello and Arm-A)

Verified security for the Morello capability-enhanced prototype Arm architecture. ESOP 2022. Bauereiss, Campbell, T.Sewell, Armstrong, Esswood, Stark, Barnes, Watson, Sewell.

Machine instructions aren't simple atomic primitives

Arm definition of the intra-instruction behaviour, of each instruction in isolation:

400k lines of ASL/Sail

(a first-order program that does uninterpreted register and memory accesses)

How can we reason about that? For known code:

- Isla symbolically evaluates the semantics of individual instructions, wrt constraints on the opcode, register values, etc., to tree-structured traces of register/memory events and SMT constraints
- instantiate Iris with an embedding of that (with an opsem) in Rocq
- adapt Lithium separation logic programming automation

Islaris: Verification of Machine Code Against Authoritative ISA Semantics. PLDI 2022. Sammler, Hammond, Lepigre, Campbell, Pichon-Pharabod, Dreyer, Garg, Sewell

Machine instructions aren't simple atomic primitives

Arm definition of the intra-instruction behaviour, of each instruction in isolation:

400k lines of ASL/Sail

(a first-order program that does uninterpreted register and memory accesses)

How do we make such a huge definition manageable?

Find or invent narrow interfaces: ~~instructions~~ register/memory interface events

Build automation: Isabelle / Isla / Islaris proof automation

Machine instructions aren't simple atomic primitives

```
Inductive trace_label : Set :=
| LReadReg (r : sail_name) (al : accessor_list) (v : valu)
| LWriteReg (r : sail_name) (al : accessor_list) (v : valu)
| LReadMem (data : valu) (kind : valu) (addr : valu) (len : N) (tag : tag_value)
| LWriteMem (res : valu) (kind : valu) (addr : valu) (data : valu) (len : N) (tag : tag_value)
| LBranchAddress (v : valu)
| LBranch (c : Z) (desc : string)
| LDone (next : isla_trace)
| LAssert (b : bool)
| LAssume (b : bool)
| LAssumeReg (r : sail_name) (al : accessor_list) (v : valu)
```

(simple version, as in Islaris, for non-systems-architecture things)

Machine instructions aren't simple atomic primitives

Open problems:

- Can we do whole-ISA proof in Rocq, to avoid gluing two provers? (Or Iris in Isabelle...)

Machine instructions aren't simple atomic primitives

Open problems:

- Can we do whole-ISA proof in Rocq, to avoid gluing two provers? (Or Iris in Isabelle...)
- For interactive proof, make Islaris more usable – wrt both understanding Isla traces and the proof automation

Machine instructions aren't simple atomic primitives

Open problems:

- Can we do whole-ISA proof in Rocq, to avoid gluing two provers? (Or Iris in Isabelle...)
- For interactive proof, make Islaris more usable – wrt both understanding Isla traces and the proof automation
- For larger-scale automated proof, to use as part of other verification tooling, make the automation scale

Machine instructions aren't simple atomic primitives

Open problems:

- Can we do whole-ISA proof in Rocq, to avoid gluing two provers? (Or Iris in Isabelle...)
- For interactive proof, make Islaris more usable – wrt both understanding Isla traces and the proof automation
- For larger-scale automated proof, to use as part of other verification tooling, make the automation scale
- For more foundational proof, replicate Isla symbolic evaluation inside Rocq, for the Sail-to-Rocq generated version of the semantics (or a deep-embedding variant thereof)?

Machine instructions aren't simple atomic primitives

In progress (Brian Campbell) – Improving shallow embedding:

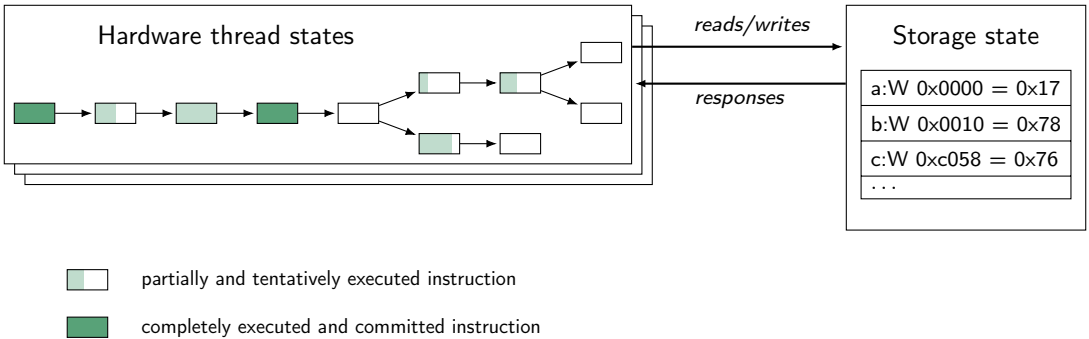
- Addressing efficiency problems (e.g., equality for large enums)
- Simpler, type safe register and sequential register state representations
- Support for coq-record-update
- More stdpp support soon

Deep embedding:

- Dominique Devriese's team experimenting with a backend for their Katamaran tool

Programs don't execute in order

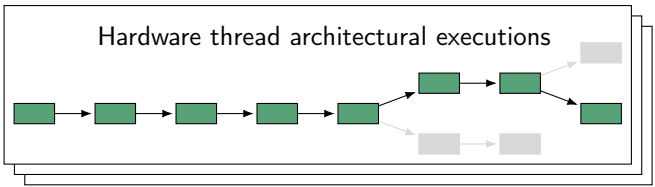
Hardware execution – and abstract-microarchitectural operational models



Transition system – but with explicit out-of-order and speculative execution, with restarts

Defined only for whole programs

After the dust settles – and in axiomatic models



completely executed and committed instruction

discarded instruction

Axiomatic models are predicates on candidate complete execution graphs

Defined only for whole programs and complete executions

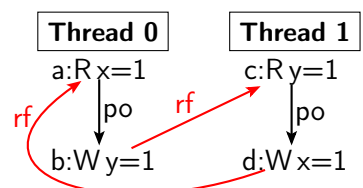
Executions aren't incremental (in any obvious way)

Totally non-compositional: not merely whole-program, but also whole-complete-execution

Intrinsically so: because LB is allowed, a program-order prefix of an allowed execution may not be even a well-formed candidate.

LB AArch64

Initial state: 0:X3=y; 0:X2=1; 0:X1=x; 0:X0=0; 1:X3=x; 1:X2=1; 1:X1=y; 1:X0=0; y=0; x=0;	
Thread 0	**Thread 1**
LDR X0,[X1] r0=x//a **STR** X2,[X3] y =1//b	**LDR** X0,[X1] r0=y//c **STR** X2,[X3] x =1//d
Allowed: 0:X0=1; 1:X0=1;	



Can we reason compositionally about Arm or RISC-V relaxed concurrency?

AxSL Iris program logic for Arm-A user relaxed concurrency, POPL 2024

Angus Hammond[1], Zongyuan Liu[1], Thibaut Pérami, Peter Sewell, Lars Birkedal, Jean Pichon-Pharabod

How, given LB? Instantiate Iris with strange operationalisation of the axiomatic model: guess a complete memory-model-consistent candidate, then check each thread in program order

Can we reason compositionally about Arm or RISC-V relaxed concurrency?

AxSL Iris program logic for Arm-A user relaxed concurrency, POPL 2024

Angus Hammond[1], Zongyuan Liu[1], Thibaut Pérami, Peter Sewell, Lars Birkedal, Jean Pichon-Pharabod

How, given LB? Instantiate Iris with strange operationalisation of the axiomatic model: guess a complete memory-model-consistent candidate, then check each thread in program order

Open problems:

- (Zongyuan in progress) extend to mixed ob+coherence
- extend with other Arm features (mixed-size, ...)
- glue the AxSL logic onto the real Arm instruction-set semantics
- establish *"fiction of sequential consistency"* – local DRF-SC for real Arm
- establish *"fiction of LB freedom"*
- make usable for hand proofs of larger small examples
- make target for automated proofs of real code
- use the same idea for other axiomatic relaxed models
- what does the underlying Iris model look like for this instantiation?

Can we reason compositionally about Arm or RISC-V relaxed concurrency?

How do we make such a subtle definition manageable?

Find or invent narrow interfaces:

- ~~instructions~~ register/memory interface events
- program logic over those
- derived rules for instructions

Can we reason compositionally about Arm or RISC-V relaxed concurrency?

```
Lemma LB_data_thread1 :
  (* Start the program with empty context *)
  (None -{LPo}> ∗ ∅ -{Ctrl}> ∗ None -{Rmw}>) -∗
  (* No local writes to address [x] and [y] *)
  last_local_write tid addr_x None -∗
  last_local_write tid addr_y None -∗
  (** Register points-to of [r1] *)
  (∃ rv, "r1" ↦r rv) -∗
  (** Instruction memory *)
  (* Half of LB-data at [0x1000] *)
  (BV 64 0x1000) ↦i read "r1" addr_x -∗
  (BV 64 0x1004) ↦i write_with_dep "r1" addr_y -∗
  (* Program ends at [0x1008] *)
  (BV 64 0x1008) ↦i - -∗
  (** Weakest precondition *)
  (* Run the program starting from the instruction at [0x1000]
     in thread [1] *)
  WPi (LTSI.Normal, (BV 64 0x1000)) @ 1
    {{ λ lts',
         (** Postcondition *)
         (* Execution terminates at address [0x1008] *)
         ⌜lts' = (LTSI.Done, (BV 64 0x1008))⌝ ∗
         (* Value of [r1] is [0] *)
         ∃ rv, "r1" ↦r rv ∗ ⌜rv.(reg_val) = BV 64 0⌝ }}.
```

Systems semantics

- instruction fetch
- virtual memory
- exceptions

Programs

Programs don't exist

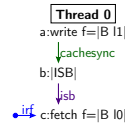
Self-modifying code: dynamic loading, linking, patching, and just-in-time compilation

SM+cachesync-isb AArch64

Initial state: 0:W0="B l1", 0:X1=f

Thread 0	Common
STR W0,[X1] //overwrite f with branch **DC** CVAU,X1 //clean data cache **DSB** ISH **IC** IVAU,X1 //invalidate instruction cache **DSB** ISH **ISB** //flush pipeline BL f **MOV** X0,X10	f: B l0 l1: **MOV** X10,#2 RET l0: **MOV** X10,#1 RET

Forbidden: 1:X0=1



ARMv8-A system semantics: instruction fetch in relaxed architectures.

Simner, Flur, Pulte, Armstrong, Pichon-Pharabod, Maranget, Sewell. ESOP 2020

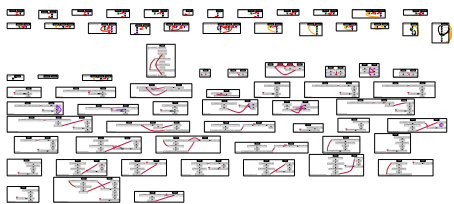
Open problems:

- how can we reason about self-modifying code?
- establish the *"fiction of Harvard architecture"*: for programs that don't write to any address they fetch from, one can ignore ifetch

Memory addresses don't mean what you think
...virtual addresses and address translation

...which again needs careful discipline by system software, with enough synchronisation and TLB invalidation, to avoid unpredictable behaviour and to enforce security

Relaxed virtual memory in Armv8-A. Simner, Armstrong, Pichon-Pharabod, Pulte, Grisenthwaite, Sewell. ESOP 2022



[From Ben Simner draft thesis]

Memory addresses don't mean what you think
...virtual addresses and address translation

Open problems:

- (Ben, Jean, Kayvan, David, ThomasF, Yeji, Peter) Can one test that systems s/w follows that discipline?
- (Thibaut, Brian, Ben, ThomasB) Can one integrate ISA + VM semantics and prove it provides the *"fiction of virtual memory"*: for programs that don't write to the memory holding their address-translation mappings, one can ignore address translation
- (Christopher et al.) Can one reason compositionally about the sequential aspects of the code that manages page tables?
- Can one reason compositionally about the relaxed concurrent semantics of virtual memory management?

Machines don't just execute instructions – there are exceptions

Synchronous exceptions, for system calls, page faults, etc.
Asynchronous exceptions, for timers, devices, inter-processor interrupts

Open problems:

- (Ben, Jean, Ohad, Peter) How do they behave?
- Can we reason about them?
- ...e.g., for the *"fiction of virtual memory with mapping on demand"*, by the OS or hypervisor, on page faults

Almost enough foundations for actual systems software...?

- instruction semantics
- "user" relaxed concurrency
- instruction fetch
- virtual memory
- exceptions
- GIC
- IOMMU (SMMU)
- other SoC aspects

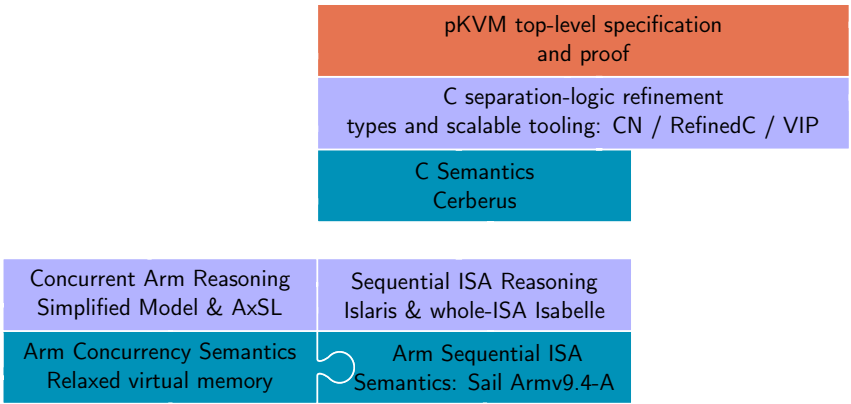
Almost enough foundations for actual systems software...?

- instruction semantics
- "user" relaxed concurrency
- instruction fetch
- virtual memory
- exceptions
- GIC
- IOMMU (SMMU)
- other SoC aspects

Open problems:

- (Thibaut etc.) build integrated semantics
- build integrated program logic, sufficient in principle for reasoning at the machine-code level about everything pKVM does

Top-down: what about systems C code



Ongoing C verification experiments

open postdoc position

CN:

- Verifying systems C code with separation-logic refinement types. POPL 2023.
  Pulte, Makwana, T. Sewell, Memarian, P. Sewell, Krishnaswami.

  Aimed at predictable SMT-based proof automation, above Cerberus semantics.
  Used for pKVM `page_alloc.c` buddy allocator and some `pgtable.c` safety proof

RefinedC:

- RefinedC: Automating the Foundational Verification of C Code with Refined Ownership Types
  Sammler, Lepigre, Krebbers, Memarian, Dreyer, Garg. PLDI2021 (VIP POPL2022, BFF OOPSLA2022).

  In Rocq with Lithium proof automation, using Cerberus front-end and a simplified C semantics.
  Demo'd on small examples.

Ongoing C verification experiments

open postdoc position

CN carefully limits the specification language, for predictable SMT-based proof automation, and for testing, with fall-back to Rocq where needed

Open problems:

- How far can we make CN usable by non-prover-experts?
- Bitvectors vs integers?
- Exercise CN on more big examples
- ...does that limited expressiveness continue to suffice?
- Extend the CN program logic with concurrency and the underlying Arm systems features
- Define an Iris instantiation for CN-exported lemmas about resources
- Better user experience at the CN-automation / Rocq-proof boundary
  (conceivably, embed the same idea in Rocq?)
- Make more foundational by generating (RefinedC-like?) proof scripts
  (needs Cerberus Core in Rocq)

Ongoing C verification experiments

open postdoc position

How do we make such a huge and subtle definition manageable?

Find or invent narrow interfaces: $\in$ Core

...use the semantics directly in the verification tool

Relating C and binary

Open problem:

- translation validation for verified (systems) code

First experiment, from CN to Islaris: 2023 Mete Polat TUM/UCam Masters thesis

...use something like Islaris in an automated flow?

How are programs actually composed?

- With syntactic language constructs

How are programs actually composed?

- With syntactic language constructs
- With static and dynamic linking, and ABIs, and FFIs based on those

The missing link: explaining ELF static linking, semantically, Kell, Mulligan, Sewell. OOPSLA 2016

Open problems:

- Compositional reasoning for real linking

A stack of fictions

- Arm (or RISC-V or x86) architecture semantics

A stack of fictions

- fiction of Harvard architecture
- Arm (or RISC-V or x86) architecture semantics

A stack of fictions

- fiction of virtual memory
- fiction of Harvard architecture
- Arm (or RISC-V or x86) architecture semantics

A stack of fictions

- fiction of virtual memory, with mapping on demand
- fiction of virtual memory
- fiction of Harvard architecture
- Arm (or RISC-V or x86) architecture semantics

A stack of fictions

- fiction of LB freedom
- fiction of virtual memory, with mapping on demand
- fiction of virtual memory
- fiction of Harvard architecture
- Arm (or RISC-V or x86) architecture semantics

A stack of fictions

- fiction of SC for race-free nonatomics
- fiction of LB freedom
- fiction of virtual memory, with mapping on demand
- fiction of virtual memory
- fiction of Harvard architecture
- Arm (or RISC-V or x86) architecture semantics

A stack of fictions

- fiction of composable (linkable) object files
- fiction of SC for race-free nonatomics
- fiction of LB freedom
- fiction of virtual memory, with mapping on demand
- fiction of virtual memory
- fiction of Harvard architecture
- Arm (or RISC-V or x86) architecture semantics

A stack of fictions

- fiction of C (or Rust)
- fiction of composable (linkable) object files
- fiction of SC for race-free nonatomics
- fiction of LB freedom
- fiction of virtual memory, with mapping on demand
- fiction of virtual memory
- fiction of Harvard architecture
- Arm (or RISC-V or x86) architecture semantics

A stack of fictions

- fiction of C (or Rust)
- fiction of composable (linkable) object files
- fiction of SC for race-free nonatomics
- fiction of LB freedom
- fiction of virtual memory, with mapping on demand
- fiction of virtual memory
- fiction of Harvard architecture
- Arm (or RISC-V or x86) architecture semantics

Whole-model theorems, or local results?
Each holds only under some assumptions that have to be established by software...

A stack of fictions

- fiction of C (or Rust)
- fiction of composable (linkable) object files
- fiction of SC for race-free nonatomics
- fiction of LB freedom
- fiction of virtual memory, with mapping on demand
- fiction of virtual memory
- fiction of Harvard architecture
- Arm (or RISC-V or x86) architecture semantics

Whole-model theorems, or local results?
Each holds only under some assumptions that have to be established by software...
Not really a stack...

Conclusion

Lots of interesting Iris-adjacent open problems!

Interesting tensions among scope, scale, and level of automation