Proposal #1: A JIT compiler for Sail

Context

For formal reasoning about low-level systems code, formal semantics for CPU instruction set architectures (e.g. x86, ARMv8, RISC-V) are required. Sail[1] is a simple imperative language for describing instruction set architectures in a way that can generate definitions usable for interactive proof tools. Currently we have formal models for ARMv8, RISC-V and MIPS that are complete enough to boot various operating systems - such complete formal models have not been previously available for mainstream production architectures.

However, as instruction set architectures are large and complex (the ARMv8 reference manual is 7900 pages long at the time of writing), it is desirable to validate their specifications of by compiling the formal semantics into an emulator that is performant enough to perform common tasks like booting operating systems and hypervisors, as well as run existing architectural validation suites.

There is currently a large performance gap between existing tools for fast emulation/simulation of processors, such as QEMU, GenSim[2] and gem5[3], and tools for specifying the semantics of instruction set architectures.

Sail currently generates an emulator which is performant enough to boot Linux in several minutes by compiling the high level Sail definitions into a small and simple intermediate language (IR), which is then directly compiled to C functions. However, this up-front approach is limited when compared to a JIT approach as it cannot use run-time information to optimise the code, which because of the formal nature of the Sail, is forced to perform expensive tasks like e.g. executing the entire definition of address translation on every memory access.

Project

This project would involve (1) Designing and writing a JIT compiler and runtime for the Sail IR, likely using the LLVM compiler infrastructure project. (2) Identifying optimisations that can be performed using runtime information that would be particularly beneficial for this particular use case (ISA simulation) by profiling real world operating systems and instruction set test suites.

Requirements

The student should have a good grasp of the following topics:

- Compilers
- Programming in a systems programming language e.g. Rust, C, C++
- Computer architecture

Project supervisors

Alasdair Armstrong alasdair.armstrong@cl.cam.ac.uk
Peter Sewell Peter.Sewell@cl.cam.ac.uk

References

[1]: ISA semantics for ARMv8-a, RISC-v, and CHERI-MIPS (https://alastairreid.github.io/papers/popl19-isasemantics.pdf)
[2]: https://gensim.org/home
[3]: http://gem5.org/Main_Page
[4]: https://llvm.org/

Proposal 2: x86 semantics in Sail

Context

The x86 instruction set architecture is used by the majority non-mobile devices today. Therefore having a formal semantics for x86 is of key importance for the verification of critical low-level systems code.

There have been various attempts to create formal specifications of the x86 instruction set architecture, such as [2], and [3], with the
ACL2 model in [3] being arguably one of the most complete. Sail[1] is a language for specifying instruction set semantics, which supports several use cases such as integration with concurrency semantics, emulator generation, automatic analysis via SMT solvers, and translation into multiple interactive proof tools. We currently have formal models for ARMv8, RISC-V and MIPS which are complete enough to boot various operating systems, including Linux and FreeBSD.

Project

This project would involve (1) Translating the ACL2 x86 model into Sail. We expect that this step might involve some collaboration with the original authors of the ACL2 model. (2) Validating the translated Sail by e.g. running Linux or another operating system on a generated emulator. (3) Potentially applying the generated semantics to some interesting problem domain, such as binary analysis, translation validation, or concurrency semantics.

Requirements

The student should have a good grasp of the following topics:

- Functional programming
- Computer architecture

Project supervisors

Alasdair Armstrong alasdair.armstrong@cl.cam.ac.uk
Peter Sewell Peter.Sewell@cl.cam.ac.uk

References

[1]: ISA semantics for ARMv8-a, RISC-v, and CHERI-MIPS (https://alastairreid.github.io/papers/popl19-isasemantics.pdf)
[2]: x86-64 semantics in K -
https://github.com/kframework/X86-64-semantics