The State of Sail

Alasdair Armstrong
Department of Computer Science and Technology, University of Cambridge, UK

Thomas Bauereiss
Department of Computer Science and Technology, University of Cambridge, UK

Brian Campbell
School of Infomatics, University of Edinburgh, UK

Alastair Reid
ARM Ltd., Cambridge, UK

Kathryn E. Gray
Department of Computer Science and Technology, University of Cambridge (Formerly), UK

Robert M. Norton
Department of Computer Science and Technology, University of Cambridge, UK

Prashanth Mundkur
SRI International, Menlo Park, US

Mark Wassell
Department of Computer Science and Technology, University of Cambridge, UK

Jon French
Department of Computer Science and Technology, University of Cambridge, UK

Christopher Pulte
Department of Computer Science and Technology, University of Cambridge, UK

Shaked Flur
Department of Computer Science and Technology, University of Cambridge, UK

Ian Stark
School of Infomatics, University of Edinburgh, UK

Neel Krishnaswami
Department of Computer Science and Technology, University of Cambridge, UK

Peter Sewell
Department of Computer Science and Technology, University of Cambridge, UK

Abstract

Sail is a custom domain-specific language for ISA semantics, in which we have developed formal models for ARMv8-A, RISC-V, and MIPS, as well as CHERI-based capability extensions for both RISC-V and MIPS. In particular, our model of ARMv8-A is automatically translated from ARM-internal definitions and tested against the ARM Architecture validation suite. All the above models contain enough system-level features to boot various operating systems, including Linux and FreeBSD, but also various smaller microkernels and hypervisors.

In this short paper, we present the ways in which Sail enables us to bridge the gap between our various ISA models and the myriad use cases for such models. By using Sail, we are able to generate emulators for testing and validation, generate theorem prover definitions across multiple major tools (Isabelle, HOL4, and Coq), translate Sail to SMT for automatic verification, and integrate with both operational models for relaxed-memory concurrency via our RMEM tool.

We will also present our current work to extend Sail to support axiomatic concurrency models, in the style of Alglave and Maranget's herd7 tool, with the intent being to explore the behaviour of concurrent litmus tests that span the full behaviour of the architecture. As an illustrative example, one could consider how instruction cache maintenance instructions interact with self-modifying code in an axiomatic setting, or other interesting cases that are not well-covered by existing tools.

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Theory of computation → Semantics and reasoning; Computer systems organization → Architectures; Software and its engineering → Assembly languages

Keywords and phrases Instruction Set Architectures, Semantics, Theorem Proving

Digital Object Identifier 10.4230/LIPIcs...

1 Overview

Sail is a custom pseudocode-like language for specifying the semantics of instruction set architectures (ISAs). It is a first-order imperative language with a semantics that is as straightforward as possible. We have aimed to strike a balance in designing a language that is both expressive enough to idiomatically express multiple instruction set architectures, which simultaneously being as inexpressive as possible to allow translation to each desired target. This balance is partially achieved with a type system that allows dependent types for bitvector widths and integer ranges, the typing information from which can be exploited by various generic rewrites and backend-specific optimisations e.g. for monomorphisation. Figure 1 gives an overview of our currently supported instruction set architectures and target uses, updated with changes since our previous paper [6].

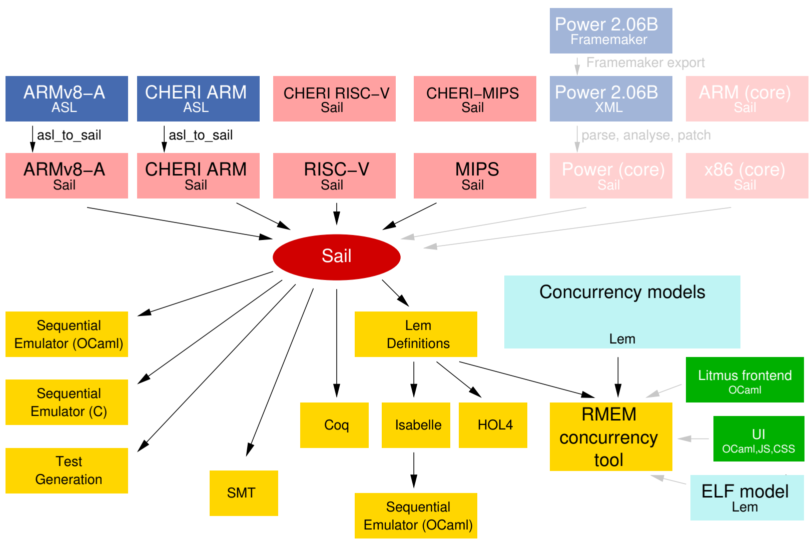

Figure 1 Sail ISA semantics (top) and target use cases (bottom). The greyed out ISAs are from previous work we are not actively working on

This short paper describes the current state of each of our Sail models, and describes ongoing work to enhance Sail with automatic verification and axiomatic concurrency support via a translation from Sail into SMTLIB definitions for the Z3 and CVC4 SMT solvers. The below table summarises the state of most of our models including our CHERI extensions.

	source	KLoS	provers	boots
ARMv8.5-A	ASL	125	Isa, HOL4, Coq*	Linux, Hafnium
MIPS	hand	2	Isa, HOL4, Coq	FreeBSD
CHERI MIPS	hand	+2	Isa, HOL4, Coq	FreeBSD, CheriBSD
RISC-V	hand	5	Isa, HOL4, Coq	Linux, FreeBSD, FreeRTOS, Hafnium
CHERI RISC-V	hand	+2	Isa, HOL4, Coq	
CHERI ARM	ASL		Isa	

ARMv8.5-A Our ARMv8-A model is translated from ARM's internal architecture specification language ASL. Our translation has been validated by running our translation against the ARM internal architecture validation suite, as previously discussed in [6]. Recently we have been improving Sail's translation into Coq. We have continued to work on emulation performance for ARM, which was previously slower than our other models due to the size of the specification, and it now boots Linux in just under 2 minutes (on a Ryzen 5 2600X), corresponding to approximately 200 000 instructions per second, roughly a four-fold improvement over [6]. We have also been working on developing infrastructure for formally proving properties of a CHERI ARM specification, but this work is in early stages.

(CHERI) RISC-V We have extended our RISC-V model with CHERI capability support. We have ensured our RISC-V model is extensible, so the CHERI extension (and other extensions) are able to exist as a separate repository which builds upon the base model. We have further validated the RISC-V spec by running FreeRTOS and a port of the Hafnium hypervisor atop the RISC-V model, in addition to Linux, FreeBSD, and seL4.

(CHERI) MIPS Our CHERI-MIPS model continues to be extended with new CHERI instructions, and is now an official part of the CHERI ISAv7 [10] architecture manual.

2 Automatic property verification with Sail-SMT

In addition to translating Sail to interactive theorem provers, we have more recently implemented a translation from Sail into SMT. This enables QuickCheck-like properties to be stated and verified in Sail itself (provided they are free of loops, in which case they are checked up to some iteration bound). For example, Figure 2 shows a property from our CHERI RISC-V specification, which verifies that if `setCapBounds` claims to have set `c`'s bounds to base and top exactly, then `getCapBounds` will return the same bounds as were set. While this property seems simple, capability bounds are stored using a fairly intricate floating-point-like compressed format, and there are additional subtle edge cases at the top and bottom of the address space. Our SMT translation was able to discover bugs in our implementations of such capability manipulation functions which had not been found via random testing.

A major advantage we have found in this style of lightweight verification with SMT solvers is that it can be used by hardware-designers developing ISA extensions who have no experience with interactive theorem proving tools. Another use for hardware-designers is to write a Sail version of a function that closely mimics a Bluespec (or other HDL) implementation that is complicated due to e.g. timing requirements, and automatically prove it equivalent to a simple Sail implementation.

```
function set_bounds_exact(c : Capability, base : bits(64), top : bits(65)) -> bool = {
  let (exact, c') = setCapBounds(c, base, top);
  let (base', top') = getCapBounds(c');
  ~(exact) | (unsigned(base) >= unsigned(top))
  | (base' == unsigned(base) & top' == unsigned(top))
}
```

Figure 2 An automatically verified property from CHERI RISC-V

The basic approach is similar to that used by existing model-checking tools such as CBMC [1], and the approach used for ARM's ASL language [9]. We first translate the Sail source into an intermediate representation (IR) which is shared by the C backend, this is then converted into a SSA based control-flow graph, which is then turned into a sequence of SMTLIB definitions which can be used with either Z3 or CVC4.

3 Axiomatic relaxed-memory concurrency with Sail

Previous work on concurrent behaviours of instruction set architectures using Sail was based on our RMEM tool [8] which provides operational-semantics for various memory models. However, many architectures, such as RISC-V specify their memory model in an axiomatic-style, where the memory model is described in terms of axioms that restrict the set of possible candidate executions. Alglave et al's diy7 [3, 4] tool suite, in particular the herd7 [5] tool, already provides a framework for evaluating the relaxed-memory behaviour for small assembly programs (litmus tests) over several architectures, using a language called *cat* [2]. However the ISA semantics used by herd is hard-coded in OCaml for each supported architecture within the tool, plus additional architecture specific infrastructure for e.g. assembly parsing.

By combining our Sail to SMT translation with the existing infrastructure for litmus tests and cat files provided by the diy7 tools, we aim to produce a tool similar to herd7, except using the Sail instruction semantics and assembly parsing infrastructure (which can also be specified within Sail). This would give us a architecture-agnostic tool that can combine an arbitrary memory model specified in cat, with an ISA specified in Sail. While our implementation is still very experimental, initial results are promising, and prior work such as Lau et al's Cerberus-BMC [7] for C11 concurrency demonstrate that the use of a SMT solver in this area is practicable.

References

1. CBMC: Bounded Model Checking for Software, 2017. `http://www.cprover.org/cbmc/`.
2. Jade Alglave, Patrick Cousot, and Luc Maranget. Syntax and semantics of the weak consistency model specification language cat. *CoRR*, abs/1608.07531, 2016. URL: `http://arxiv.org/abs/1608.07531`, `arXiv:1608.07531`.
3. Jade Alglave and Luc Maranget. The `diy7` tool. `http://diy.inria.fr/`, 2017.
4. Jade Alglave, Luc Maranget, Susmit Sarkar, and Peter Sewell. Fences in weak memory models. In *Proceedings of CAV 2010: the 22nd International Conference on Computer Aided Verification, LNCS 6174*, 2010. `doi:10.1007/978-3-642-14295-6_25`.
5. Jade Alglave, Luc Maranget, and Michael Tautschnig. Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory. *ACM TOPLAS*, 36(2):7:1–7:74, July 2014.
6. Alasdair Armstrong, Thomas Bauereiss, Brian Campbell, Alastair Reid, Kathryn E. Gray, Robert M. Norton, Prashanth Mundkur, Mark Wassell, Jon French, Christopher Pulte, Shaked Flur, Ian Stark, Neel Krishnaswami, and Peter Sewell. ISA semantics for armv8-a, risc-v, and CHERI-MIPS. *PACMPL*, 3(POPL):71:1–71:31, 2019. `doi:10.1145/3290384`.
7. Stella Lau, Victor B. F. Gomes, Kayvan Memarian, Jean Pichon-Pharabod, and Peter Sewell. Cerberus-BMC: a principled reference semantics and exploration tool for concurrent and sequential C. In *CAV 2019*, July 2019. (to appear).
8. Christopher Pulte, Shaked Flur, Will Deacon, Jon French, Susmit Sarkar, and Peter Sewell. Simplifying ARM Concurrency: Multicopy-atomic Axiomatic and Operational Models for ARMv8. In *POPL 2018*, July 2018. `doi:10.1145/3158107`.
9. Alastair Reid. Who guards the guards? formal validation of the arm v8-m architecture specification. *Proc. ACM Program. Lang.*, 1(OOPSLA):88:1–88:24, October 2017. `doi:10.1145/3133912`.
10. Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff, Michael Roe, Hesham Almatary, Jonathan Anderson, John Baldwin, David Chisnall, Brooks Davis, Nathaniel Wesley Filardo, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, Simon W. Moore, Steven J. Murdoch, Kyndylan Nienhuis, Robert Norton, Alex Richardson, Peter Rugg, Peter Sewell, Stacey Son, and Hongyan Xia. Capability Hardware Enhanced RISC Instructions: CHERI instruction-set architecture (version 7). Technical report, Computer Laboratory, June 2019.