GRIFT: A richly-typed, deeply-embedded RISC-V semantics written in Haskell

Benjamin Selfridge
Galois, Inc.
benselfridge@galois.com

We introduce the Galois RISC-V Formal Tools (GRIFT) [1], a formal specification of the RISC-V instruction set architecture written in the Haskell programming language. It consists of a description of the encoding and semantics of each RISC-V instruction in an embedded domain-specific language (DSL) specialized for instruction set specification, as well as command-line tools implemented atop this specification for sequential simulation, coverage analysis, and documentation. We compare and contrast our approach to ISA specification with others developed in the RISC-V Formal Specification task group—two of which are also written in Haskell—and discuss future applications and directions.

1 Introduction

RISC-V is an open source instruction set architecture developed at UC Berkeley [15]. The base instruction set is very limited, at around 50 instructions, and more functionality is offered via a number of standard extensions (*M* for multiplication, *F* and *D* for single- and double-precision floating-point, etc.). Hardware designers can pick and choose which bit widths and extensions they want to support, and can also develop their own custom extensions that, if widely adopted by the community, can be standardized by the RISC-V Foundation.

As a new foundation for hardware platforms, RISC-V has provided an opportunity to design an instruction set, along with its associated ecosystem of compilers, hardware designs, and formal methodologies, in a more unified and intentional manner than has been possible for proprietary ISAs. To this end, the RISC-V Foundation has established a number of task groups to address the growth of the RISC-V instruction set and its accompanying infrastructure. The Formal Specification task group was organized with the following stated intent:

> This group will produce a Formal Specification for the RISC-V ISA. This is a specification of the ISA in a formal language, for precision, unambiguity, consistency and completeness. It should be readable and understandable as a canonical reference by practising CPU architects and compiler writers. It should [sic] executable and machine-manipulable for use in formal tools for establishing correctness and transformations in both compilers and CPU designs. [8]

We suggest that such a specification ought to contain a maximally *general* and *precise* description of the behavior of every RISC-V instruction. It should also be highly *configurable*, because RISC-V has a large number of customizable implementation choices, including register width and available extensions.

We further suggest that a canonical specification, capable of being useful to hardware designers, compiler writers, and formal methods experts, should be designed as a kind of Rosetta Stone for RISC-V . The ISA should be easily translated from the specification language to any other target language. One needs to be able to use the specification to validate a large number of hardware and software artifacts, and such validations will likely involve multiple external tools, each with a unique interface. It is therefore important to ensure that backends for these tools are easy to build as new needs arise.

Galois RISC-V Formal Tools (GRIFT) [1], the system described in this paper, was designed with these goals in mind. It uses Haskell as the modeling language, which was a natural choice given its facility for describing domain-specific languages (DSLs), as well as the place it occupies at the intersection of practical programming languages and formal methods. By explicitly expressing the RISC-V *feature model* within the Haskell type system via a number of advanced extensions to the Glasgow Haskell Compiler (GHC), GRIFT puts the customizability of the architecture front-and-center. Furthermore, rather than modeling state transitions as Haskell functions, GRIFT uses an embedded DSL to encode those state transitions, the syntax of which can be easily inspected, analyzed, and translated into other forms (see figure 1). This makes GRIFT a logical candidate as an all-purpose RISC-V specification that can be leveraged for a wide variety of applications.

GRIFT is configurable for both 32-bit and 64-bit register widths, and also supports all the standard extensions: *I* (base), *M* (multiplication), *A* (atomic memory operations), *F* (single-precision floating point), *D* (double-precision float-

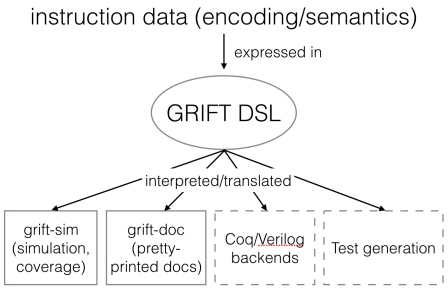

Fig. 1. GRIFT's DSL-based design. Dotted lines indicate future work (see section 5).

ing point), and *C* (compressed instructions). It does not yet have full support for user-level and supervisor-level privilege modes, but such support is planned.

2 The GRIFT Haskell Library

GRIFT provides the Haskell programmer with a number of artifacts, including a type-level representation of the RISC-V feature model that pervades all aspects of the specification, a declarative mechanism for expressing instruction encodings, and an embedded DSL for encoding RISC-V state transitions.

2.1 The RISC-V Feature Model as a Type

One aspect of RISC-V that should be very precisely captured by a canonical specification is its parameterizability. Every implementation must contain the base (RV32I) instruction set, which consists of the core instructions that are always present regardless of register width and supported extensions. It can then choose whether to support 32- or 64-bit register widths, which of a selection of various standard (and potentially non-standard) extensions to include, and other, finer-grained decisions (such as whether to support misaligned memory accesses in hardware). The set of all possible parameterizations of a complex system like this is best represented by a *feature model*. Feature models are well-studied in a specialized subset of the formal methods and programming languages communities. [12] [13] The full RISC-V feature model is quite large, due to the plethora of minute decisions that are left to the implementer. GRIFT's feature model currently accounts for the register width and extensions, but does not yet incorporate other implementation decisions such as misaligned accesses and the many acknowledged ambiguities in the RISC-V privileged architecture.

We chose to encode the RISC-V feature model directly in the type system as a parameter to all of the central GRIFT data types. A particular instance of RISC-V (characterized by its register width and implemented extensions) is represented as a *data kind*, or type-level data structure, holding all the information specific to that configuration. All of GRIFT's core data types are parameterized over all possible configurations, and we can then impose constraints on the constructors of those types to guarantee that they cannot be used in a RISC-V instance that forbids them. As a simple example, the declaration of the `Opcode` data type is as follows:

```
data Opcode :: RV -> Format -> * where

  Add  :: Opcode rv R
  Sub  :: Opcode rv R
  <...>
  Mul  :: MExt << rv => Opcode rv R
  <...>
```

The `Opcode` data type has two type parameters, an `RV` and a `Format`. The `RV` parameter is a particular instance of the RISC-V feature model. Since `Add` and `Sub` are valid for any RISC-V instance, no constraint is placed on the `rv` type variable. However, `Mul` is only valid in a context where the *M* (multiply/divide) extension is present. Accordingly, we add the constraint `MExt << rv` to that constructor, ensuring that `Mul` is only used in such scenarios.

The choice to enforce the RISC-V feature model so rigorously was not without its drawbacks. GHC's support for dependent types is somewhat ad hoc and experimental, making it difficult to use and understand for someone not familiar with the more cutting-edge GHC extensions. However, we believe that the consistency and structure of the specification benefits greatly from this level of rigor, in spite of some of the resulting awkwardness.

2.2 Instruction Encoding

A significant design goal for GRIFT was to avoid directly encoding aspects of the ISA via Haskell functions, because the syntax of Haskell cannot be directly inspected in a reflective manner. Instead, we chose a more concrete representation for the core aspects of the ISA. Instead of simply writing a `decode` function that converts a 32-bit word to an `Opcode`, we chose instead to represent an instruction's encoding as a piece of data that can be inspected, analyzed, and rendered by the user however he pleases.

One consequence of this design decision is that we have effectively created a bidirectional mapping between concrete instructions in binary form and their abstract representations. This means that the `decode` and `encode` functions can both reference the same encoding data to perform their respective tasks, and are automatically inverses of each other. Thus, GRIFT can be used for *generating* binaries as well as analyzing and simulating them.

2.3 The GRIFT State Transition DSL

GRIFT uses a *deep embedding* of the semantics of each instruction. Instead of modeling the effect that each instruction has on the machine state as a Haskell function, we instead developed an abstract syntax tree (AST) to encode these transitions syntactically. Listed below are the semantics of the `add` instruction:

```
defInst Add $ instSemantics
  (Rd :< Rs1 :< Rs2 :< Nil) $ do
```

```
rd :< rs1 :< rs2 :< Nil <- operandEs

let x_rs1 = readGPR rs1
let x_rs2 = readGPR rs2
let res   = x_rs1 `addE` x_rs2

assignGPR rd res
incrPC
```

This code does not modify any state directly; instead, it builds abstract syntax in our embedded DSL. This representation can then be executed in simulation (similar to other Haskell specifications, like *Forvis* [2] and *riscv-semantics* [3]), and it can also be leveraged for other purposes that deal directly with the semantics as data.

The GRIFT DSL also makes strategic use of dependent types to track the compile-time width of all the symbolic bit vectors within the semantics. We have found through experience that many ISA semantics bugs are directly attributable to an imprecise notion of vector width; GRIFT addresses this issue by always demanding that the width of any bit vector is known at compile time. This leads to a wide class of semantics bugs being eliminated before the code is ever run.

3 Command-Line Tools

To demonstrate the GRIFT library, we developed a simulator, `grift-sim`, that executes ELF files containing RISC-V programs compiled for any valid RISC-V configuration. This simulator also provides two different flavors of post-simulation *coverage analysis*: 1) instruction set coverage (that is, how many different kinds of instructions were executed), and 2) intra-instruction branching coverage (so, if an instruction includes semantic branching, the user can easily discover which of those branches were actually executed and which were not). This capability was used to discover several coverage holes in the RISC-V compliance suite [4], involving all of the branching and memory access instructions. Because the semantics of GRIFT are represented as syntax rather than Haskell functions, we are able to directly inspect and track the structure of the state transitions, rendering this analysis straightforward.

To use GRIFT to analyze overall instruction coverage of the compliance suite for RV32I, we invoke the following command:

```
$ grift-sim --arch=RV32I
  --inst-coverage=all rv32i/*.elf
```

This lists every available instruction, along with how many of the possible branches were actually executed during simulation. To discover which branches of a particular instruction were missed, we change the keyword `all` to the instruction's opcode (e.g. `add`), and the tool prints out the branching structure of the instruction, using different terminal colors to indicate which branches were executed and which were not.

GRIFT also comes with a proof-of-concept documentation tool, `grift-doc`, which displays both the binary encoding and the semantics of individual instructions on the command line. The textual descriptions generated by this tool are extracted directly from the semantics. This tool makes GRIFT useful for those wishing to understand how specific instructions operate, without having to understand the source code itself, and we plan to expand it to produce LaTeXas well as simple text.

4 Comparison with other RISC-V Formal Specification Task Group Efforts

GRIFT is one of five different specification efforts under development as part of the RISC-V Formal Specification Task Group. The others are the Haskell-based *Forvis* [2] (Bluespec) and *riscv-semantics* [3] (MIT), the RISC-V model written in the Sail ISA description language [5] [10] (University of Cambridge), and the RISC-V model written in Kami [6] (MIT and SiFive). The Kami system is itself embedded in the Coq proof assistant.

GRIFT's main distinguishing feature in relation to the other two Haskell specifications is its use of a deeply-embedded DSL to express instruction encoding and semantics. This means that the state transition induced by an instruction is an object that can be analyzed and manipulated from within the host language, rather than as a function, which cannot be inspected so directly. Both *Forvis* and *riscv-semantics* model instruction semantics as Haskell functions. This approach has the advantage of being more accessible to a human reader of the specification. However, this representation can be somewhat cumbersome to work with for other purposes; as an example, both of these specifications have been translated to Coq and Verilog, but the translation must be performed by analyzing the Haskell source with external tools (*hs-to-coq* [14] and *CLASH* [11], respectively). With GRIFT, backends for Coq, Verilog, etc. could be developed within Haskell, with code that translates the syntax of the DSL into the destination language.

Despite being written in the same language as *Forvis* and *riscv-semantics*, GRIFT is closer in principle to the RISC-V specification written in the Sail ISA description language [9]. Sail is a DSL designed specifically for describing the semantics of instruction set architectures, and has been used to develop formal specifications of x86, ARM, PowerPC, and now RISC-V. Sail also has language constructs and supporting infrastructure for describing the semantics of concurrent memory accesses by multiple processors; GRIFT currently has no such support. However, given GRIFT's DSL-based rendering of the semantics, entirely divorced in principle from a sequential environment, we can imagine expanding GRIFT to express concurrency, either declaratively or operationally. The question of how best to accomplish this is a complex problem; it leads to intractable non-determinism at simulation time, and even if a memory model is very precisely specified (as the RISC-V Memory Model Task Group has made it), it is no simple task to express it adequately in a useful way.

A notable shortcoming of GRIFT is that it does not currently support the full privileged specification with accom-

panying user and supervisor modes, as the other models do. It will be straightforward to incorporate these features into GRIFT, and we plan to do so in the near future.

5 Future Work and Reflections

Aside from making GRIFT more feature-complete with respect to the RISC-V privileged architecture, we plan to explore the applicability of GRIFT to software and hardware verification. Recently, we began development of a RISC-V backend for *macaw*, a Haskell library for binary code discovery also developed at Galois [7]. We also plan to develop backends that translate the GRIFT DSL to other systems, including hardware description languages like Verilog and theorem provers like Coq and ACL2. This would enable use of the specification in those environments for equivalence checking against hardware designs, and for binary-level software correctness proofs. All of these applications can be accomplished by writing more Haskell libraries that import GRIFT as a submodule; no other external tooling is required, in principle.

We have also experimented with using GRIFT to generate test cases that are complete with respect to various notions of coverage. This could be incorporated into the RISC-V Compliance Task Group's work, generating randomized variants of the tests they have developed.

GRIFT was designed to be an all-purpose formal specification for RISC-V . It puts the RISC-V feature model at the center of its design, which provides clarity about how the various extensions and register widths interact. It also represents the core aspects of the ISA (instruction encodings and semantics) as concrete data, which makes it straightforward to use in a wide variety of contexts. We plan to continue to develop and use GRIFT as a Rosetta Stone for RISC-V .

Acknowledgment

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-18-C-0013. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).

This document was cleared by DARPA on July 23, 2019. All copies should carry the Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). If you have any questions, please contact the Public Release Center.

References

[1] https://github.com/GaloisInc/grift.

[2] https://github.com/rsnikhil/Forvis_RISCV-ISA-Spec.

[3] https://github.com/mit-plv/riscv-semantics.

[4] https://github.com/riscv/riscv-compliance.

[5] https://github.com/rems-project/sail-riscv.

[6] https://github.com/sifive/RiscvSpecFormal.

[7] https://github.com/GaloisInc/macaw.

[8] Formal specification task group. https://lists.riscv.org/g/tech-formalspec.

[9] A. Armstrong, T. Bauereiss, B. Campbell, S. Flur, K. E. Gray, P. Mundkur, R. M. Norton, C. Pulte, A. Reid, and P. Sewell. Detailed models of instruction set architectures: From pseudocode to formal semantics. *Automated Reasoning Workshop*, 2018.

[10] A. Armstrong, T. Bauereiss, B. Campbell, A. Reid, K. E. Gray, R. M. Norton, P. Mundkur, M. Wassell, J. French, C. Pulte, S. Flur, I. Stark, N. Krishnaswami, and P. Sewell. Isa semantics for armv8-a, risc-v, and cheri-mips. *Proc. ACM Program. Lang.*, 3(POPL):71:1–71:31, Jan. 2019.

[11] C. Baaij, M. Kooijman, J. Kuper, W. Boeijink, and M. Gerards. Cash: Structural descriptions of synchronous hardware using haskell. In *Proceedings of the 13th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools*, pages 714–721, United States, 9 2010. IEEE Computer Society. eemcs-eprint-18376.

[12] M. Janota and G. Botterweck. Formal approach to integrating feature and architecture models. In J. L. Fiadeiro and P. Inverardi, editors, *Fundamental Approaches to Software Engineering*, pages 31–45, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[13] K. C. Kang, J. Lee, and P. Donohoe. Feature-oriented project line engineering. *IEEE Softw.*, 19(4):58–65, July 2002.

[14] A. Spector-Zabusky, J. Breitner, C. Rizkallah, and S. Weirich. Total haskell is reasonable coq. *CoRR*, abs/1711.09286, 2017.

[15] K. Waterman, Andrew; Asanovic. The risc-v instruction set manual, volume 1: Unprivileged isa, June 2019. https://riscv.org/specifications/.