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

Ben Selfridge, Galois Inc.
September 13, 2019

Overview

- RISC-V & the RISC-V Formal Specification Task Group
- GRIFT walkthrough and demo
- Questions

RISC-V

- Open source instruction set architecture developed at UC Berkeley
- Attempts to avoid cruft and stagnation of proprietary ISAs
  - Base instruction set (“I”) is very small (~40 instructions)
  - Other instructions available via *extensions* (M, A, F, D, C, …)
- Overall design emphasizes *configurability* across various parameters (register width, available extensions)
  - Example configurations: RV32I, RV64IMAFDC, etc.
- Evolution of RISC-V standards and artifacts is stewarded by the RISC-V Foundation via a number of “task groups”

RISC-V Formal Specification Task Group

- Goal: Develop a formal specification of the RISC-V instruction set architecture that is:
  - Precise/unambiguous
  - Readable (as text)
  - Executable
  - Useful for hardware engineers and formal methods engineers

GRIFT

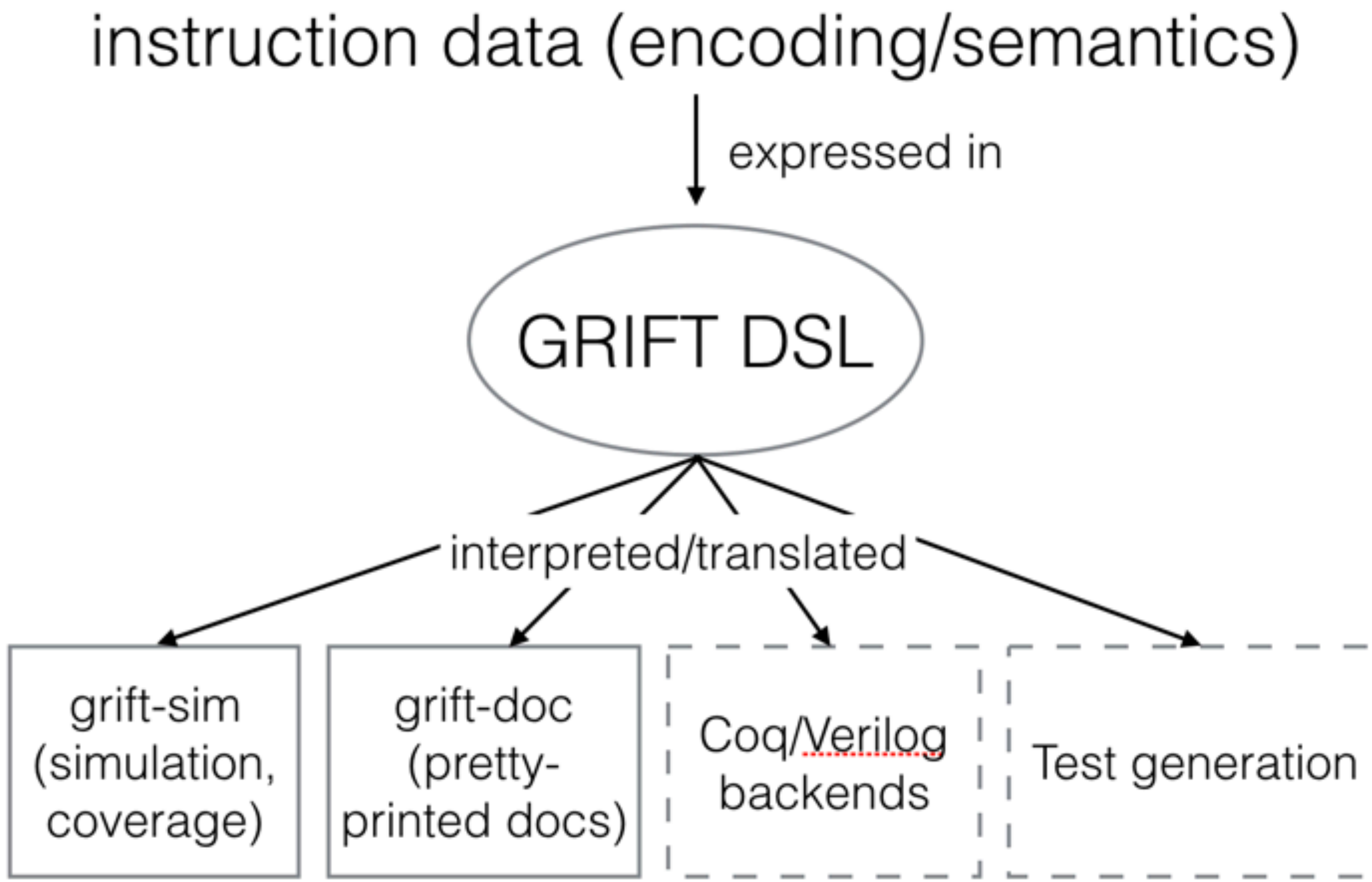
- “Galois RISC-V Formal Tools” (GRIFT) is our contribution to the RISC-V Formal Spec Group

- Formalizes instruction encoding and semantics in *embedded domain-specific language* (eDSL) within Haskell, as a **library**

- Includes **command-line tools** for simulation, coverage analysis, and documentation/pretty printing

GRIFT: Design goals

- Express RISC-V configuration in Haskell’s type system (compile-time guarantees)
- Express encoding and semantics in an embedded DSL to allow translation to other languages and environments
- Represent core ISA as data, rather than Haskell functions, so it can be manipulated directly and translated into other environments


instruction data (encoding/semantics)
expressed in
GRIFT DSL
interpreted/translated
grift-sim (simulation, coverage)
grift-doc (pretty-printed docs)
Coq/Verilog backends
Test generation

Type-level RISC-V Configuration

- RISC-V configurability:
  - 32-bit/64-bit
  - Base ISA + extensions (M, A, F/D, C, …)
- We capture the configuration of a RISC-V system as a type parameter for our core data types
- Instructions in a particular extension can only be used if it is known that the configuration supports that extension

Type-level RISC-V Configuration

```
data RV = RVConfig (BaseArch, Extensions)
```

```
data BaseArch = RV32
              | RV64
              | RV128
```

Type-level RISC-V Configuration

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

```haskell
Add   :: Opcode rv R
Addw    :: 64 <= RVWidth rv => Opcode rv R
Mul     :: MExt << rv => Opcode rv R
```

GRIFT semantics DSL

- Instruction semantics represented in an embedded DSL, with AST nodes for:
  - Arithmetic and bitvector operations
  - Register/memory accesses
  - Reading from instruction operands
- Dependently typed, using type-level naturals in GHC to track bitvector widths
- “Shape” of instruction (number and size of operands) captured as a type parameter

GRIFT semantics DSL

```
Mul     :: MExt << rv => Opcode rv R
```

```
Pair Mul $ instSemantics (Rd :< Rs1 :< Rs2 :< Nil) $ do
  comment "Multiplies x[rs1] by x[rs2] and writes the prod to x[rd]."
  comment "Arithmetic overflow is ignored."

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

  let x_rs1 = readGPR rs1
  let x_rs2 = readGPR rs2

  assignGPR rd (x_rs1 `mulE` x_rs2)
  incrPC
```

GRIFT's encoding representation

- Instructions are parameterized by "format", which determines operand number and width
- Format determines mapping between operands and their locations in the instruction
- The *fixed bits* of a particular instruction must also be defined to perform encoding and decoding

GRIFT's encoding representation

```
Mul      :: MExt << rv => Opcode rv R
```

```
Pair Mul     (OpBits RRepr (0b0110011 :< 0b000 :< 0b0000001 :< Nil))
```

GRIFT simulator

- ~40,000 instructions per second
- Disassembles ELF binaries compiled by gcc
- Interprets semantics DSL code for each instruction against a concrete machine state
- Dumps output to terminal as directed (register file, section of memory)

GRIFT simulator — coverage analysis

- Bonus feature of simulator (available via command-line options)

- Tracks coverage of individual instructions based on the branching structure of their semantics as expressed in semantics DSL

- Discover coverage holes in RISC-V compliance suites

- Notion of coverage is limited, but could be refined for particular needs

Other RISC-V Formal Specification Efforts

- SAIL RISC-V (Cambridge)
- riscv-semantics (MIT)
- Forvis (Bluespec)
- Kami RISC-V (MIT/Si-Five)

	Forvis	Grift	Sail	riscv-plv	Kami
Author/Group	Bluespec	Galois	SRI/Cambridge	MIT	SiFive
Licence	MIT	GPL3	BSD	MIT	Apache 2.0
Metalanguage	Haskell	embedded DSL in Haskell	Sail	Haskell	Kami/Coq
Functional coverage - Base ISA and extensions	RV32/64IMAFDC	RV32/64GC	RV32/RV64IMAC	RV32/64IMAF	RV32 IMAFC
Functional coverage - Privilege levels	MUS,Sv32,39,48	M	MUS,Sv32,39,48	Sv39	no
Specification of assembly syntax and encoding	no	pp	yes	no	no
Concurrency	no	no	yes	no	no

	Forvis	Grift	Sail	riscv-plv	Kami
Floating-point	via Softfloat	via Softfloat	no	via Softfloat	Native implementation of IEEE 754-2008
Emulation	Haskell	Haskell	generated C or OCaml	Haskell	Verilator
...emulation speed	??? IPS (40min Linux boot)	40K IPS on Intel Xeon E312	300K IPS on Intel i7-7700 (4min Linux boot)	100K IPS on 6700HQ (Linux boot)	Not measured
Use as test oracle in tandem verification	yes	no	yes	yes	yes
Use for software coverage analysis	???	yes	???	???	???
Theorem-prover definitions	via hs-to-coq?	no	Coq,Isa,HOL4	hs-to-coq	Coq
Use in documentation	to LaTeX	to text	to LaTeX in RISC-V ISA	no	no
Use in test generation	(at UPenn?)	no	yes	no	no

	Forvis	Grift	Sail	riscv-plv	Kami
Use for concurrency-model litmus test evaluation	no	no	yes	no	no
Test coverage - riscv-tests suite	???	yes	yes	yes	yes
Test coverage - RISC-V compliance tests	all	almost all	yes	yes	yes
Test coverage - OS boots	Linux,FreeRTOS	no	Linux,FreeBSD,seL4	Linux	no
Test coverage - Concurrency litmus tests	no	no	yes	no	no

Conclusion

- GRIFT: A Haskell library comprising a formal RISC-V specification
  - RISC-V configuration expressed via Haskell types
  - Instruction encoding/semantics expressed in an embedded DSL
- Future work:
  - Applications: binary analysis, hardware/software verification
  - Other backends (Coq, ACL2, Verilog, PDF manuals)
  - Automated test generation
  - Concurrency?

[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/.