Semantics for Verified Software-Hardware Stacks

CS 152

Samuel Gruetter

Reusing/citing slides by Nada Amin
Based on joint work with Joonwon Choi, Andres Erbsen, Clark Wood, Adam Chlipala

Harvard, 21 April 2022
About me

- Undergrad/masters at EPFL (Switzerland), supervised by then-PhD-student Nada Amin
- Internship at Princeton University with Andrew Appel’s group using the Verified Software Toolchain (VST)
- Currently: PhD student at MIT in Adam Chlipala’s group
Styles of Semantics

Operational Semantics
Denotational Semantics
Axiomatic Semantics
Algebraic Semantics

Integration Verification Across Software and Hardware for a Simple Embedded System

Andres Erbsen* Samuel Gruetter* Joonwon Choi Clark Wood Adam Chlipala

Abstract
The interfaces between layers of a system are susceptible to bugs if developers of adjacent layers proceed under subtly different assumptions. Formal verification of two layers of a simple embedded system using the automated theorem prover PVS illustrates the difficulties involved.

1 Introduction
We present the comprehensive and modular verification of functional correctness of a newly realistic but still very simple embedded system, highlighting important challenges that are unique to verification of this type.
Software-Hardware Stacks

- System Behavior Requirements
- Applications
- Library API Specs
- Libraries
- Programming Language Spec
- Compiler
- RISC-V ISA
- Processor
- Hardware Description Language Spec
Need agreement at each layer boundary

- System Behavior Requirements
- Applications
- Library API Specs
- Libraries
- Programming Language Spec
- Compiler
- RISC-V ISA
- Processor
- Hardware Description Language Spec
Problem

No matter
- how the specs are stated
- how often they change
- whether providers and users are the same team or not
Problem

No matter

- how the specs are stated
- how often they change
- whether providers and users are the same team or not

There will always be

- Misunderstandings
- Oversights
Problem

No matter
• how the specs are stated
• how often they change
• whether providers and users are the same team or not

There will always be
• Misunderstandings
• Oversights

Which lead to
• Bugs
• Security vulnerabilities
Problem

No matter how the specs are stated
● how often they change
● whether providers and users are the same team or not

There will always be
● Misunderstandings
● Oversights
Which lead to
● Bugs
● Security vulnerabilities

Can a computer help us solve this problem?

That’s our research
Systems development using machine-checked specs at all levels

- Applications
- Library API Specs
- Libraries
- Programming Language Spec
- Compiler
- RISC-V ISA
- Processor
- Hardware Description Language Spec
Systems development using machine-checked specs at all levels

- System Behavior Requirements
  - Applications
  - Library API Specs
    - Libraries
  - Programming Language Spec
    - Compiler
    - RISC-V ISA
  - Processor
- Hardware Description Language Spec
Systems development using machine-checked specs at all levels

- System Behavior Requirements
- Applications
- Library API Specs
- Libraries
- Programming Language Spec
- Compiler
- RISC-V ISA
- Processor
- Hardware Description Language Spec
Prototype: The IoT lightbulb

- Wrote all digital parts from scratch
- Using bedrock2
  - a low-level programming language with verification support
- Very simple at the moment
- Goal: Should scale to more realistic applications
printf '1' | nc -w0 -u 192.168.1.123 9999
printf '0' | nc -w0 -u 192.168.1.123 9999
Specification using a Log

- Receive1
- TurnOn
- PollNone
- Receive0
- TurnOff
- ReceiveInvalid
- PollNone
- PollNone
How to specify the behavior of the processor
Small-Step Operational Semantics

step from configuration to configuration:

\[ c_0 \rightarrow c_1 \rightarrow \ldots \rightarrow c_n \]
Semantic Rules (1/2)

**LADD**
\[
\frac{< e_1, \sigma > \rightarrow < e_1', \sigma' >}{< e_1 + e_2, \sigma > \rightarrow < e_1' + e_2, \sigma' >}
\]

**RADD**
\[
\frac{< e_2, \sigma > \rightarrow < e_2', \sigma' >}{< n + e_2, \sigma > \rightarrow < n + e_2', \sigma' >}
\]

**ADD**
\[
\frac{< n + m, \sigma > \rightarrow < p, \sigma >}{\text{where } p \text{ is the sum of } n \text{ and } m}
\]

**LMUL**
\[
\frac{< e_1, \sigma > \rightarrow < e_1', \sigma' >}{< e_1 \times e_2, \sigma > \rightarrow < e_1' \times e_2, \sigma' >}
\]

**RMUL**
\[
\frac{< e_2, \sigma > \rightarrow < e_2', \sigma' >}{< n \times e_2, \sigma > \rightarrow < n \times e_2', \sigma' >}
\]

**MUL**
\[
\frac{< n \times m, \sigma > \rightarrow < p, \sigma >}{\text{where } p \text{ is the product of } n \text{ and } m}
\]
Large-step semantics

\[ \Downarrow \subseteq \text{Config} \times \text{FinalConfig} \]

where

\[ \text{Config} = \text{Exp} \times \text{Store} \]

and \( \text{Final Config} = \text{Int} \times \text{Store} \subseteq \text{Config} \).

We write \( < e, \sigma > \Downarrow < n, \sigma' > \) to mean that

\( ( < e, \sigma >, < n, \sigma' > ) \in \Downarrow \)
Semantic Rules (1)

\[
\text{INT}_{\text{LRG}} \quad \frac{< n, \sigma > \downarrow < n, \sigma >}{< x, \sigma > \downarrow < n, \sigma > \quad \text{where } n = \sigma(x)}
\]

\[
\text{VAR}_{\text{LRG}} \quad \frac{< e_1, \sigma > \downarrow < n_1, \sigma'' > \quad < e_2, \sigma'' > \downarrow < n_2, \sigma' >}{< e_1 + e_2, \sigma > \downarrow < n, \sigma' > \quad \text{where } n \text{ is the sum of } n_1 \text{ and } n_2}
\]
Arithmetic expressions

\[ A[n] = \{ (\sigma, n) \} \]
\[ A[x] = \{ (\sigma, \sigma(x)) \} \]
\[ A[a_1 + a_2] = \{ (\sigma, n) \mid (\sigma, n_1) \in A[a_1] \wedge (\sigma, n_2) \in A[a_2] \wedge n = n_1 + n_2 \} \]
\[ A[a_1 \times a_2] = \{ (\sigma, n) \mid (\sigma, n_1) \in A[a_1] \wedge (\sigma, n_2) \in A[a_2] \wedge n = n_1 \times n_2 \} \]
How to specify the behavior of the processor

• Using the hardware description language Kami
• Embedded in Coq
Kami Semantics

Expression  $e ::= c | x | \text{op}(\vec{e}) | [\vec{e}] | \{k = e\} | e[e] | e.k$

Action $a ::= \text{let } x = r \text{ in } a$
  $| r := e ; a$
  $| \text{let } x = f(e) \text{ in } a$
  $| \text{let } x = e \text{ in } a$
  $| \text{if } e \text{ then } a \text{ else } a ; a$
  $| \text{assert } e ; a$
  $| \text{return } e$

Module $m ::= \langle[\langle r, c \rangle], [\langle s, a \rangle], [\langle f, \lambda x : \tau \cdot a \rangle] \rangle$
  $| m + m$

Q: Where do you see/expect small-step/big-step/denotational semantics?

More details on Kami: See ICFP’17 paper

More details on Kami: See ICFP’17 paper

$\ell$:
\begin{align*}
o &\xrightarrow{\ell} (u, v) \\
o &\xrightarrow{\ell} (u, v) \\
[e] &= \nu_r \quad r \notin u \quad o \xrightarrow{\ell} (u, v) \\
[e] &= \nu_a \quad (f(\_ ) = \_ ) \notin \ell \quad o \xrightarrow{\ell} (u, v) \\
o &\xrightarrow{\ell} (u, v) \\
o &\xrightarrow{\ell} (u, v)
\end{align*}
Arithmetic expressions

\[ A[n] = \{(\sigma, n)\} \]
\[ A[x] = \{(\sigma, \sigma(x))\} \]
\[ A[a_1 + a_2] = \{(\sigma, n) \mid (\sigma, n_1) \in A[a_1], \]
\[ \sigma, n_2 \in A[a_2], \]
\[ n = n_1 + n_2 \} \]
\[ A[a_1 \times a_2] = \{(\sigma, n) \mid (\sigma, n_1) \in A[a_1], \]
\[ \sigma, n_2 \in A[a_2], \]
\[ n = n_1 \times n_2 \} \]

Denotational semantics for Kami expressions

\[ [[c]] = c \]
\[ [[\text{op}(e)]] = [[\text{op}]]_{\text{op}}([[e]]) \]
\[ [[e_0, \ldots, e_{2^{n-1}}]] = \lambda i : \text{word}(n). [[e_i]] \]
\[ [[\{k_1 = e_{k_1}, \ldots, k_n = e_{k_n}\}]] = \lambda k : \{k_1, \ldots, k_n\}. [[e_k]] \]
\[ [[e_v[e_i]]] = [[e_v]]([[e_i]]) \]
\[ [[e.k]] = [[e]](k) \]
Inductive Set $\rightarrow^* \text{ (Multi-Step Rel.)}$

\[
\begin{align*}
&< e, \sigma > \rightarrow^*< e, \sigma > \\
&\quad < e, \sigma > \rightarrow < e', \sigma' > \quad < e', \sigma' > \rightarrow^*< e'', \sigma'' > \\
&\quad \quad \quad \quad < e, \sigma > \rightarrow^*< e'', \sigma'' >
\end{align*}
\]

From single step to multistep in Kami

**Inductive** Multistep: RegsT $\rightarrow$ RegsT $\rightarrow$ list LabelT $\rightarrow$ Prop :=
- NilMultistep $o1 \; o2$: $o1 = o2 \rightarrow$ Multistep $o1 \; o2 \; nil$
- Multi $o \; a \; n$ (HMultistep: Multistep $o \; n \; a$)
  - $u \; l$ (HStep: Step $n \; u \; l$):
    Multistep $o$ (M.union $u \; n$) ($l :: a$).
Specification using a Log

- Recv1
- TurnOn
- PollNone
- Recv0
- TurnOff
- RecvInvalid
- PollNone
- PollNone
The lightbulb spec

**lightbulb_spec:**

The log of the blue wires and the red wire respects the following regular expression:

\[
\text{BootSeq } ((\text{Recv1 On}) \mid (\text{Recv0 Off}) \mid \text{RecvInvalid} \mid \text{PollNone})^*
\]
The end-to-end theorem

Using Coq, we developed

- a software image (list of bytes)
- a pipelined processor
- a theorem:

> If you put the image at address 0 and set PC to 0 and run our processor, the log of the blue wires and the red wire respects the following regular expression:

\[ \text{BootSeq } ((\text{Recv1 TurnOn}) \mid (\text{Recv0 TurnOff}) \mid \text{RecvInvalid} \mid \text{PollNone})^* \]
End-to-end theorem

**Theorem** end2end_lightbulb: ∀ mem₀ t,
bytes_at (instrencode lightbulb_insts) 0 mem₀ ∧
Trace (p4mm mem₀) t →
∃ t': list (string * word * word),
KamiRiscv.KamiLabelSeqR t t' ∧
prefix_of t' goodHlTrace.

**Definition** goodHlTrace := BootSeq +++
( (EX b:bool, Recv b +++ LightbulbCmd b)
||| RecvInvalid ||| PollNone
)*.
The end-to-end theorem

- a concise description of the behavior of the whole stack
- high assurance of correctness
The source language
IMP syntax

\[ a ::= \ x \ | \ n \ | \ a_1 + a_2 \ | \ a_1 \times a_2 \]
\[ b ::= \ \text{true} \ | \ \text{false} \ | \ a_1 < a_2 \]
\[ c ::= \ \text{skip} \ | \ x := a \ | \ c_1; c_2 \]
\[ \ | \ \text{if } b \text{ then } c_1 \text{ else } c_2 \]
\[ \ | \ \text{while } b \text{ do } c \]
\[ e ::= \]
\[ v \quad \text{integer literal} \]
\[ x \quad \text{local variable} \]
\[ \text{load}N(e) \quad \text{load } N \text{ bytes from address } e, N = 1, 2, \text{or } 4 \]
\[ \text{load}W(e) \quad \text{load } 4 \text{ or } 8 \text{ bytes, depending on bitwidth} \]
\[ e_1 \ op \ e_2 \quad \text{binary operation, } op = +, -, *, &, \ldots \]

\[ c ::= \]
\[ \text{skip} \quad \text{do nothing} \]
\[ x = e \quad \text{assignment to local variable} \]
\[ \text{store}N(e_1,e_2) \quad \text{store the lower } N \text{ bytes of } e_2 \]
\[ \quad \text{at address } e_1, N = 1, 2, \text{or } 4 \]
\[ \text{store}W(e_1,e_2) \quad \text{store } 4 \text{ or } 8 \text{ bytes, depending} \]
\[ \quad \text{on bitwidth, at address } e_1 \]
\[ \text{if } (e) \ c_1 \ \text{else } c_2 \quad \text{if-then-else} \]
\[ c_1 ; c_2 \quad \text{sequence} \]
\[ \text{while } (e) \ c \quad \text{while loop} \]
\[ x_1, ..., x_n = f(e_1, ..., e_m) \quad \text{call procedure } f \text{ and assign} \]
\[ \quad \text{return values to } x_1, ..., x_n \]
Bedrock2 Semantics

\text{vcgen \ \text{skip} \ \text{t} \ \text{m} \ \text{l} \ \text{P} :=}
\text{P \ t \ m \ l}

\text{vcgen \ (x = e) \ \text{t} \ \text{m} \ \text{l} \ \text{P} :=}
\exists \ v, \ \text{vcexpr} \ \text{m} \ \text{l} \ e \ v \ \land \ \text{P} \ \text{t} \ \text{m} \ \text{l}[x:=v]

\text{vcgen \ (c1; \ c2) \ \text{t} \ \text{m} \ \text{l} \ \text{P} :=}
\text{vcgen \ c1 \ \text{t} \ \text{m} \ \text{l} \ (\lambda \ t' \ m' \ l' \ \Rightarrow \ \text{vcgen} \ c2 \ t' \ m' \ l' \ \text{P})}

\text{vcgen \ (store1(e1, e2)) \ \text{t} \ \text{m} \ \text{l} \ \text{P} :=}
\exists \ \text{addr}, \ \text{vcexpr} \ \text{m} \ \text{l} \ e1 \ \text{addr} \ \land
\exists \ \text{value}, \ \text{vcexpr} \ \text{m} \ \text{l} \ e2 \ \text{value} \ \land
\text{addr} \in \text{dom}(m) \ \land
\text{P} \ \text{t} \ \text{m}[\text{addr}:=\text{value}] \ \text{l}

\text{vcgen \ (x = f_{ext}(e)) \ \text{t} \ \text{m} \ \text{l} \ \text{P} :=}
\exists \ v, \ \text{vcexpr} \ \text{m} \ \text{l} \ e \ v \ \land
\text{vcexpr} \ f_{ext} \ \text{t} \ [v]
\text{(\lambda \ r \ \Rightarrow \ \text{P} \ ((f,[v],[r]):\text{t}) \ \text{m} \ \text{l}[x:=r])}
Partial Correctness and Total Correctness

\{ Pre \} \ c \ \{ Post \}

“If $Pre$ holds before $c$, and $c$ terminates, then $Post$ holds after $c$.”

[ $Pre$ ] $c$ [ $Post$ ]

“If $Pre$ holds before $c$ then $c$ will terminate and $Post$ will hold after $c$.”
Sample program verification

- Let’s step through a simple program and see how it creates an MMIO event in the trace
function attempt_read_byte() returns (uint32_t err, uint32_t v) {
    uint32_t busy = MMIOREAD(SPI_READ_ADDR);
    if (busy >> 31) {
        err = 1;
    } else {
        err = 0;
        v = busy & 0xff;
    }
}

Lemma attempt_read_byte_correct: ∀ t m,
vcgen (body attempt_read_byte) t m empty (λ t’ m’ l’ =>
∃b, t’ = (“MMIOREAD”, SPI_READ_ADDR, b)::t ∧
(0 <= b < 256 ∧ l’[err] = 0 ∧ l’[v] = b) V
(b >> 31 = 1 ∧ l’[err] = 1).
\[ \forall t \, m, \text{vcgen} (\]
\[
\text{uint32\_t busy = MMIOREAD(SPI\_READ\_ADDR);} \\
\text{if (busy >> 31) \{} \\
\text{err = 1;} \\
\text{\} else \{} \\
\text{err = 0;} \\
\text{v = busy \& 0xff;} \\
\text{\} \\
\text{t m empty (\lambda t' \, m' \, l' \mapsto} \\
\exists b, t' = ("MMIOREAD", SPI\_READ\_ADDR, b)::t \land \\
(0 \leq b < 256 \land l'[err] = 0 \land l'[v] = b) \lor \\
(b >> 31 = 1 \land l'[err] = 1) \]
∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)

∀ t m, vcgen ( 
  uint32_t busy = MMIOREAD(SPI_READ_ADDR); 
)
\forall t \ m, \ \text{vcextern} \ \text{MMIOREAD} \ t \ [\text{SPI\_READ\_ADDR}] \ (\lambda \ b =>
(\lambda \ t' \ m' \ l' =>
\text{vcgen} (\n\text{if} \ (\text{busy} >> 31) \ {\}
\text{err} = 1;
}\text{else} {\}
\text{err} = 0;
\text{v} = \text{busy} \ & \ 0xff;
})\n\text{t'} \ m' \ l' \ (\lambda \ t'' \ m'' \ l'' =>
\exists b, \ t'' = ("\text{MMIOREAD}", \text{SPI\_READ\_ADDR}, b)::t \ \land
(0 <= b < 256 \ \land \ l''[err] = 0 \ \land \ l''[v] = b) \ \lor
(b >> 31 = 1 \ \land \ l''[err] = 1).\n)\n("\text{MMIOREAD}", \text{SPI\_READ\_ADDR}, b)::t) \ m \ \text{empty}[\text{busy} := b])
∀ t m, vcextern MMIREAD t [SPI_READ_ADDR] (λ b =>
vcgen
(
    if (busy >> 31) {
        err = 1;
    } else {
        err = 0;
        v = busy & 0xff;
    }
)
((“MMIOREAD”, SPI_READ_ADDR, b)::t)
m
empty[busy:=b]
(λ t'' m'' l'' =>
    ∃b, t'' = (“MMIOREAD”, SPI_READ_ADDR, b)::t ∧
    (0 <= b < 256 ∧ l''[err] = 0 ∧ l''[v] = b) ∨
    (b >> 31 = 1 ∧ l''[err] = 1)))
∀ t m, SPI_READ_ADDR ∈ MMIO_RANGE ∧ ∀ r, (λ b =>
  vcgen
  (
    if (busy >> 31) {
      err = 1;
    } else {
      err = 0;
      v = busy & 0xff;
    }
  )
  m
  empty[busy:=b]
  (λ t'' m'' l'' =>
    ∃ b, t'' = ("MMIOREAD", SPI_READ_ADDR, b)::t ∧
    (0 <= b < 256 ∧ l''[err] = 0 ∧ l''[v] = b) ∨
    (b >> 31 = 1 ∧ l''[err] = 1))
) r
∀ t m, SPI_READ_ADDR ∈ MMIO_RANGE ∧ ∀ r,
vcodegen
  (if (busy >> 31) {
    err = 1;
  } else {
    err = 0;
    v = busy & 0xff;
  })

  ("MMIOREAD", SPI_READ_ADDR, r)::t)

m

empty[busy:=r]

(λ t'' m'' l'' =>
  ∃ b, t''' = ("MMIOREAD", SPI_READ_ADDR, b)::t ∧
  (0 <= b < 256 ∧ l''[err] = 0 ∧ l''[v] = b) ∨
  (b >> 31 = 1 ∧ l''[err] = 1))
∀ t m r,
vcgen
(
    if (busy >> 31) {
        err = 1;
    } else {
        err = 0;
        v = busy & 0xff;
    }
)
(("MMIOREAD", SPI_READ_ADDR, r)::t)
m
empty[busy:=r]
(∀ t' m' l' =>
    ∃b, t'' = ("MMIOREAD", SPI_READ_ADDR, b)::t ∧
    (0 <= b < 256 ∧ l''[err] = 0 ∧ l''[v] = b) ∨
    (b >> 31 = 1 ∧ l''[err] = 1))
\forall t \, m \, r,
\exists a, \text{vcexpr} \, m \, \text{empty}[\text{busy}:=r] \, (\text{busy} \gg 31) \, a
\wedge (a \neq 0 \rightarrow
\text{vcgen} (\text{err} = 1
\, (("MMIOREAD", \text{SPI\_READ\_ADDR}, r)::t) \, m \, \text{empty}[\text{busy}:=r]
(\lambda t'' \, m'' \, l'' \rightarrow
\exists b, t'' = ("MMIOREAD", \text{SPI\_READ\_ADDR}, b)::t \, \wedge
(0 \leq b < 256 \wedge l''[\text{err}] = 0 \wedge l''[v] = b) \lor
(b \gg 31 = 1 \wedge l''[\text{err}] = 1)))\wedge
\wedge (a = 0 \rightarrow
\text{vcgen} (\text{err} = 0;
\, v = \text{busy} \& 0xff;
) \, (("MMIOREAD", \text{SPI\_READ\_ADDR}, r)::t) \, m \, \text{empty}[\text{busy}:=r]
(\lambda t'' \, m'' \, l'' \rightarrow
\exists b, t'' = ("MMIOREAD", \text{SPI\_READ\_ADDR}, b)::t \, \wedge
(0 \leq b < 256 \wedge l''[\text{err}] = 0 \wedge l''[v] = b) \lor
(b \gg 31 = 1 \wedge l''[\text{err}] = 1)))\wedge
∀ t m r,
\exists a, a = r >> 31
\land (a <> 0 ->
\quad \text{vcgen (}
\quad \quad \text{err} = 1
\quad \) (("MMIOREAD", SPI_READ_ADDR, r)::t) m empty[busy:=r]
\quad (\lambda t'' m'' l'' =>
\quad \exists b, t'' = ("MMIOREAD", SPI_READ_ADDR, b)::t \land
\quad \quad (0 <= b < 256 \land l''[err] = 0 \land l''[v] = b) \lor
\quad \quad (b >> 31 = 1 \land l''[err] = 1))
\land (a = 0 ->
\quad \text{vcgen (}
\quad \quad \text{err} = 0;
\quad \quad v = busy \& 0xff;
\quad \) (("MMIOREAD", SPI_READ_ADDR, r)::t) m empty[busy:=r]
\quad (\lambda t'' m'' l'' =>
\quad \exists b, t'' = ("MMIOREAD", SPI_READ_ADDR, b)::t \land
\quad \quad (0 <= b < 256 \land l''[err] = 0 \land l''[v] = b) \lor
\quad \quad (b >> 31 = 1 \land l''[err] = 1))
\forall t \ m \ r,
(r >> 31 <> 0 \rightarrow
\text{vcgen (}
\text{err} = 1
)\ (("MMIOREAD", SPI\_READ\_ADDR, r)::t) \ m \ \text{empty}[\text{busy}:=r]
(\lambda \ t'' \ m'' \ l'' \Rightarrow
\exists b, t'' = ("MMIOREAD", SPI\_READ\_ADDR, b)::t \land
(0 <= b < 256 \land l''[err] = 0 \land l''[v] = b) \lor
(b >> 31 = 1 \land l''[err] = 1))\)
\land (r >> 31 = 0 \rightarrow
\text{vcgen (}
\text{err} = 0;
\text{v} = \text{busy} \land 0xff;
)\ (("MMIOREAD", SPI\_READ\_ADDR, r)::t) \ m \ \text{empty}[\text{busy}:=r]
(\lambda \ t'' \ m'' \ l'' \Rightarrow
\exists b, t'' = ("MMIOREAD", SPI\_READ\_ADDR, b)::t \land
(0 <= b < 256 \land l''[err] = 0 \land l''[v] = b) \lor
(b >> 31 = 1 \land l''[err] = 1)))
∀ t m r,

\( (r \gg 31 \neq 0 \rightarrow \) 
\( \text{vcgen (} 
\quad \text{err} = 1 
\) 
\( ) (\text{"MMIOREAD"}, \text{SPI_READ_ADDR}, r)::t) \ m \ \text{empty}[\text{busy}:=r] 
(\lambda t'' m'' l'' \Rightarrow 
\exists b, t'' = (\text{"MMIOREAD"}, \text{SPI_READ_ADDR}, b)::t \land 
(0 \leq b < 256 \land l''[\text{err}] = 0 \land l''[v] = b) \lor 
(b \gg 31 = 1 \land l''[\text{err}] = 1)) \)
\forall t \, m \, r,
(r \gg 31 < 0 \rightarrow
(\lambda \ t'\, m'\, l'\, =\rightarrow
\exists b, t'\, = ("MMIOREAD", SPI_READ_ADDR, b)::t \land
(0 \leq b < 256 \land l''[err] = 0 \land l''[v] = b) \lor
(b \gg 31 = 1 \land l''[err] = 1)))
(("MMIOREAD", SPI_READ_ADDR, r)::t) \, m \, empty[busy:=r][err:=1]
\(\forall t \in m \mapsto r, r >> 31 <> 0 \rightarrow\)
\(\exists b, ((\text{"MMIOREAD"}, \text{SPI\_READ\_ADDR}, r)::t) = ((\text{"MMIOREAD"}, \text{SPI\_READ\_ADDR}, b)::t \land\)
\((0 <= b < 256 \land \text{empty}[\text{busy}:=r][\text{err}:=1][\text{err}] = 0 \land \text{empty}[\text{busy}:=r][\text{err}:=1][v] = b)\lor\)
\((b >> 31 = 1 \land \text{empty}[\text{busy}:=r][\text{err}:=1][\text{err}] = 1)))\)
∀ t m r, r >> 31 <> 0 ->
   True ∧
   ((0 <= r < 256 ∧ empty[bussy:=r][err:=1][err] = 0 ∧
     empty[bussy:=r][err:=1][v] = r)
   ∨
   (r >> 31 = 1 ∧ empty[bussy:=r][err:=1][err] = 1)))
\( \forall \; t \; m \; r, \; r >> 31 \; <> \; 0 \; \rightarrow \\
\; (r >> 31 = 1 \; \wedge \; empty[busy:=r][err:=1][err] = 1) \)
∀ t m r, r >> 31 <> 0 ->
   r >> 31 = 1 ∧ empty[busy:=r][err:=1][err] = 1
Viewing postconditions as continuations

$$\text{SEQ} \quad \vdash \{P\} \ c_1 \ \{R\} \quad \vdash \{R\} \ c_2 \ \{Q\}$$

$$\vdash \{P\} \ c_1; \ c_2 \ \{Q\}$$

$$\mathcal{T}[\text{let } x = e_1 \ \text{in } e_2] = \lambda k. \mathcal{T}[e_1] \ (\lambda x. \mathcal{T}[e_2] k)$$

k: continuation taking the result value of the expression
P: postcondition taking resulting trace t, memory m, and local vars l

\[ \text{vgen} \ (c_1; \ c_2) \ t \ m \ l \ P := \]

\[ \text{vgen} \ c_1 \ t \ m \ l \ (\lambda \ t' \ m' \ l' \Rightarrow \text{vgen} \ c_2 \ t' \ m' \ l' \ P) \]
Bedrock2 semantics for compiler correctness proof

- vcgen is a function
- But for compiler correctness, need an Inductive representing executions to drive proof by induction over executions of programs

### EVAL-UNOP

\[
\begin{align*}
(y, v_y) & \in \ell & \Rightarrow & \text{evalunop}(op, v_y, v) Q(m, \ell[x := v], \tau) \\
(x = \text{op}(y))/m/\ell/\tau & \Downarrow Q
\end{align*}
\]

### EVAL-INPUT

\[
\begin{align*}
\forall n, Q(m, \ell[x := n], \tau :: \text{IN } n) & \\
(x = \text{input()}/m/\ell/\tau & \Downarrow Q
\end{align*}
\]

### EVAL-SEQ

\[
\begin{align*}
\forall m' \ell' \tau', Q_1(m', \ell', \tau') & \Rightarrow c_2/m'/\ell'/\tau' \Downarrow Q' \\
(c_1; c_2)/m/\ell/\tau & \Downarrow Q
\end{align*}
\]

### EVAL-STORE

\[
\begin{align*}
(x, a) & \in \ell & \Rightarrow & (a + n) \in \text{dom } m \\
(y, v) & \in \ell & Q(m[(a + n) := v], \ell, \tau) \\
(x[n] = y)/m/\ell/\tau & \Downarrow Q
\end{align*}
\]
Large-step semantics

\[ \downarrow \subseteq \text{Config} \times \text{FinalConfig} \]

where

\[ \text{Config} = \text{Exp} \times \text{Store} \]

and \( \text{Final Config} = \text{Int} \times \text{Store} \subseteq \text{Config}. \)

We write \( < e, \sigma > \downarrow < n, \sigma' > \) to mean that

\( ( < e, \sigma >, < n, \sigma' > ) \in \downarrow \)
Compiler correctness

Simple case: Source and target language is the same

\[ \forall t \; m \; l, \; c/m/l/t \downarrow Q \rightarrow compile(c)/m/l/t \downarrow Q \]

“c/m/l/t \downarrow Q” is an Inductive Prop, and we can write proofs by induction on it
Omnisemantics: Smoother Handling of Nondeterminism

ARTHUR CHARGUÉRAUD, Inria & Université de Strasbourg, CNRS, ICube, France
ADAM CHLIPALA, MIT CSAIL, USA
ANDRES ERBSEN, MIT CSAIL, USA
SAMUEL GRUETTER, MIT CSAIL, USA

This paper gives an in-depth presentation of the omni-big-step and omni-small-step styles of semantic judgments. These styles describe operational semantics by relating starting states to sets of outcomes rather than to individual outcomes. A single derivation of these semantics for a particular starting state and program describes all possible nondeterministic executions (hence the name *omni*), whereas in traditional small-step and big-step semantics, each derivation only talks about one single execution. This restructuring allows for straightforward modeling of languages featuring both nondeterminism and undefined behavior. Specifically, omnisemantics inherently assert safety, i.e. they guarantee that none of the execution branches gets stuck, while traditional semantics need either a separate judgment or additional error markers to specify safety in the presence of nondeterminism.

Omnisemantics can be understood as an inductively defined weakest-precondition semantics (or more generally, predicate-transformer semantics) that does not involve invariants for loops and recursion, but instead uses unrolling rules like in traditional small-step and big-step semantics. Omnisemantics have already been used in the past, but we believe that it has been under-appreciated and that it deserves a well-motivated, extensive and pedagogical presentation of its benefits. We also explore several novel aspects associated with these semantics, in particular their use in type-soundness proofs for lambda calculi, partial-correctness reasoning, and forward proofs of compiler correctness for terminating but potentially nondeterministic programs being compiled to nondeterministic target languages. All results in this paper are formalized in Coq.
Apply semantics, PL, verification throughout the whole stack

<table>
<thead>
<tr>
<th>System Behavior Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
</tr>
<tr>
<td>Library API Specs</td>
</tr>
<tr>
<td>Libraries</td>
</tr>
<tr>
<td>Programming Language Spec</td>
</tr>
<tr>
<td>Compiler</td>
</tr>
<tr>
<td>RISC-V ISA</td>
</tr>
<tr>
<td>Processor</td>
</tr>
<tr>
<td>Hardware Description Language Spec</td>
</tr>
</tbody>
</table>
Questions?

References:
- Integration Verification across Software and Hardware for a Simple Embedded System
  https://dl.acm.org/doi/pdf/10.1145/3453483.3454065
- Omnisemantics: Smoother Handling of Nondeterminism
  https://hal.inria.fr/hal-03255472v2/document
- A Multipurpose Formal RISC-V Specification
- Kami: a platform for high-level parametric hardware specification and its modular verification
  https://dl.acm.org/doi/pdf/10.1145/3110268
- Coq Code
  https://github.com/mit-plv/bedrock2

gruetter@mit.edu  @samuelgruetter