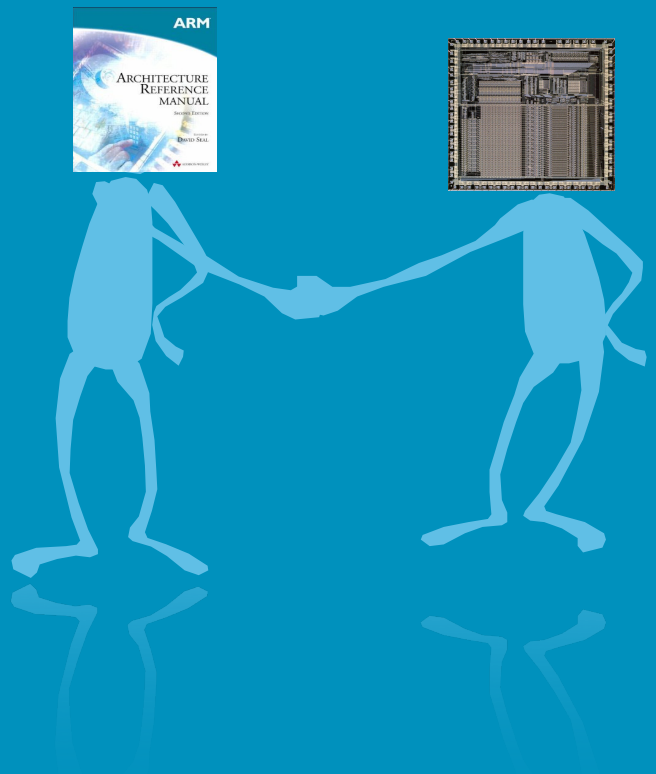
arm



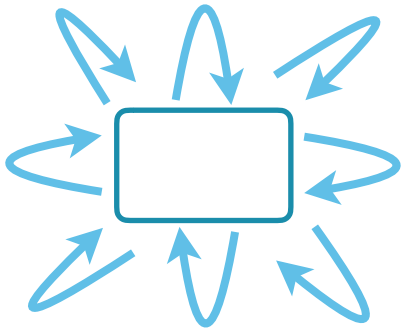
Creating Formal Specifications of Real World Artifacts

Alastair Reid

Arm Research

@alastair_d_reid

Overview



1. What's different about Real World Artifacts?
2. ARM's formal processor specifications
   - Three experiences
   - Lessons learned
3. Conclusions

"Trustworthy Specifications of the ARM v8-A and v8-M architecture," FMCAD 2016
"End to End Verification of ARM processors with ISA Formal," CAV 2016
"Who guards the guards? Formal Validation of ARM v8-M Specifications," OOPSLA 2017
https://alastairreid.github.io/papers/

arm

ARM

Designs processors

Designs architecture

Licenses architecture

16B processors / year

(also GPUs, IoT, …)

arm

Real World Artifacts

Linux Kernel, C compilers, ARM processors, TCP/IP, WiFi, etc.

- Multiple implementations, suppliers, versions, configurations
- Important: commercial, security, …
- Long history, initial spec informal
- Formal spec not 100% welcome
- Backwards compatibility requirements
- Spec must include all quirks of recent versions of major implementations to be useful
- Conformance suites?

Current status of ARM specifications

- Formal specifications of A, R and M-class processor classes exist
- Integrated into ARM's official processor specifications
- Maintained by ARM's architecture team
- Used by multiple teams within ARM
  - Formal validation of ARM processors using Bounded Model Checking
  - Development of test suites
  - Designing architecture extensions
  - …
- Publicly released in machine readable form

Creating trustworthy specifications

arm

The state of most processor specifications

Large (1000s of pages)

Broad (10+ years of implementations, multiple manufacturers)

Complex (exceptions, weak memory, …)

Informal (mostly English prose)

Pseudocode (10000s of lines)

We are all just learning how to (retrospectively) formalize specifications

Unstructured English Prose (A-class spec)

Concurrent modification and execution of instructions

The ARMv8 architecture limits the set of instructions that can be executed by one thread of execution as they are being modified by another thread of execution without requiring explicit synchronization.

Concurrent modification and execution of instructions can lead to the resulting instruction performing any behavior that can be achieved by executing any sequence of instructions that can be executed from the same Exception level, except where each of the instruction before modification and the instruction after modification is one of a `B`, `BL`, `BRK`, `HVC`, `ISB`, `NOP`, `SMC`, or `SVC` instruction.

For the `B`, `BL`, `BRK`, `HVC`, `ISB`, `NOP`, `SMC`, and `SVC` instructions the architecture guarantees that, after modification of the instruction, behavior is consistent with execution of either:

- The instruction originally fetched.
- A fetch of the modified instruction.

If one thread of execution changes a conditional branch instruction, such as `B` or `BL`, to another conditional instruction and the change affects both the condition field and the branch target, execution of the changed instruction by another thread of execution before the change is synchronized can lead to either:

- The old condition being associated with the new target address.
- The new condition being associated with the old target address.

These possibilities apply regardless of whether the condition, either before or after the change to the branch instruction, is the *always* condition.

Semi-structured English prose (M-class spec)

R_{JRJC}

Exit from lockup is by any of the following:

- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority exception.

R_{VGNW}

Entry to lockup from an exception causes:

- Any Fault Status Registers associated with the exception to be updated.
- No update to the exception state, pending or active.
- The PC to be set to `0xEFFFFFFE`.
- EPSR.IT to be become UNKNOWN.

In addition, HFSR.FORCED is not set to 1.

arm

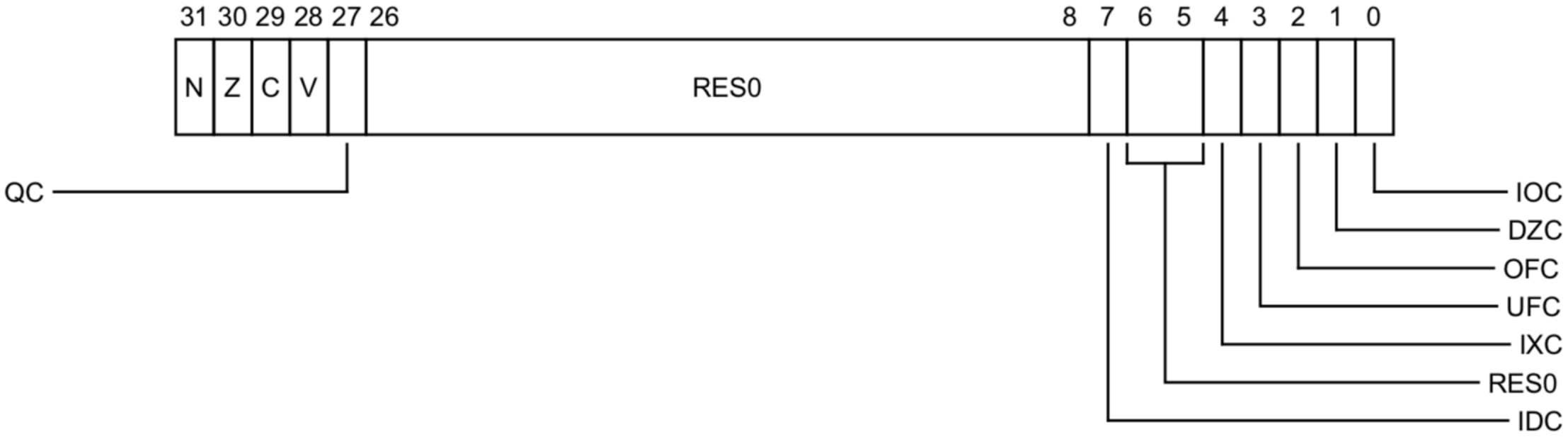
Tables - semistructured, not machine readable

Table B2-1 Encoding of the DMB and DSB <option> parameter

Accesses		Shareability domain			
Before the barrier	**After the barrier**	**Full system**	**Outer Shareable**	**Inner Shareable**	**Non-shareable**
Reads and writes	Reads and writes	SY	OSH	ISH	NSH
Writes	Writes	ST	OSHST	ISHST	NSHST
Reads	Reads and writes	LD	OSHLD	ISHLD	NSHLD

arm

Registers - structured, machine-readable



N, bit [31]

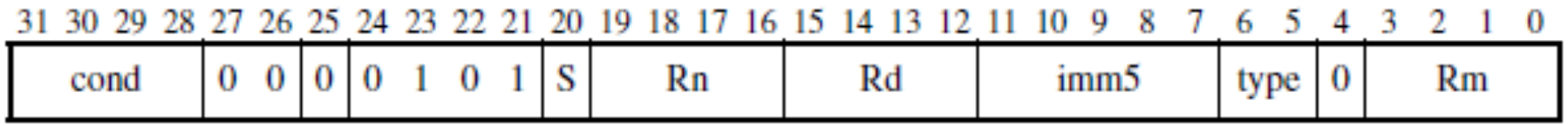
Negative condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.N flag instead.

Z, bit [30]

Zero condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.Z flag instead.

Pseudocode

```
ADC{S}<c> <Rd>,<Rn>,<Rm>{,<shift>}
```



31 30 29 28	27 26	25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8 7	6 5	4	3 2 1 0
cond	0 0	0	0 1 0 1	S	Rn	Rd	imm5	type	0	Rm

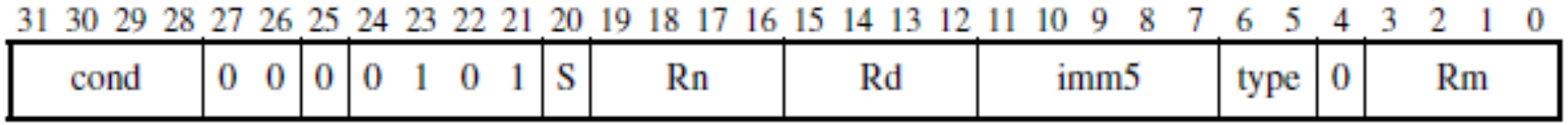
```
if Rd == '1111' && S == '1' then SEE SUBS PC, LR and related instructions;
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  setflags = (S == '1');
(shift_t, shift_n) = DecodeImmShift(type, imm5);

if ConditionPassed() then
    EncodingSpecificOperations();
    shifted = Shift(R[m], shift_t, shift_n, APSR.C);
    (result, carry, overflow) = AddWithCarry(R[n], shifted, APSR.C)
    if d == 15 then          // Can only occur for ARM encoding
        ALUWritePC(result);  // setflags is always FALSE here
    else
        R[d] = result;
        if setflags then
            APSR.N = result<31>;
            APSR.Z = IsZeroBit(result);
            APSR.C = carry;
            APSR.V = overflow;
```


arm

Pseudocode

```
ADC{S}<c> <Rd>,<Rn>,<Rm>{,<shift>}
```



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				0	0	0	0	1	0	1	S	Rn				Rd				imm5					type		0	Rm			

```
if Rd == '1111' && S == '1' then SEE SUBS PC, LR and related instructions;
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  setflags = (S == '1');
(shift_t, shift_n) = DecodeImmShift(type, imm5);

if ConditionPassed() then
    EncodingSpecificOperations();
    shifted = Shift(R[m], shift_t, shift_n, APSR.C);
    (result, carry, overflow) = AddWithCarry(R[n], shifted, APSR.C)
    if d == 15 then          // Can only occur for ARM encoding
        ALUWritePC(result);  // setflags is always FALSE here
    else
        R[d] = result;
        if setflags then
            APSR.N = result<31>;
            APSR.Z = IsZeroBit(result);
            APSR.C = carry;
            APSR.V = overflow;
```

- Type Inference
- Unbounded Integers
- Enumerations
- Bit Vectors
- Indentation-based Syntax
- Dependent Types
- Imperative
- Exceptions

arm

Status at the start

- No tools (parser, type checker)
- Incomplete (around 15% missing)
- “Document by comment”
- Many trivial errors (that confuse tools but not humans)
- Unexecuted, untested
- Scepticism that executing spec is
  - Possible
  - Desirable
  - Would compromise important aspects of specification

Architectural Conformance Suite

Processor architectural compliance sign-off

Large

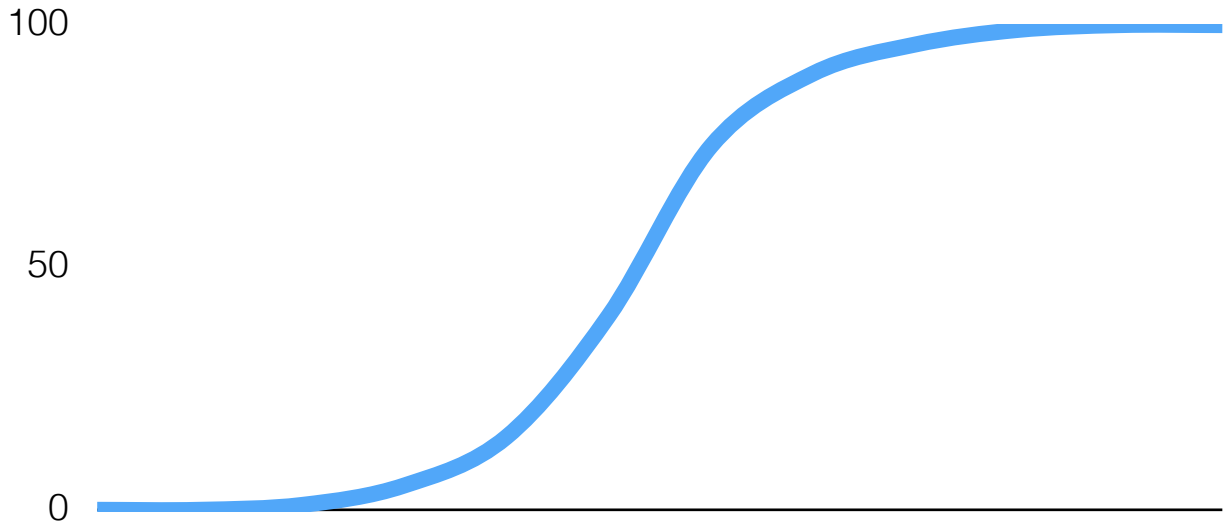
- v8-A 11,000 test programs, > 2 billion instructions
- v8-M 3,500 test programs, > 250 million instructions

Thorough

- Tests dark corners of specification

arm

Progress in testing Arm specification



- Does not parse, does not typecheck
- Can’t get out of reset
- Can’t execute first instruction
- Can’t execute first 100 instructions
- …
- Passes 90% of tests
- Passes 99% of tests
- …

arm

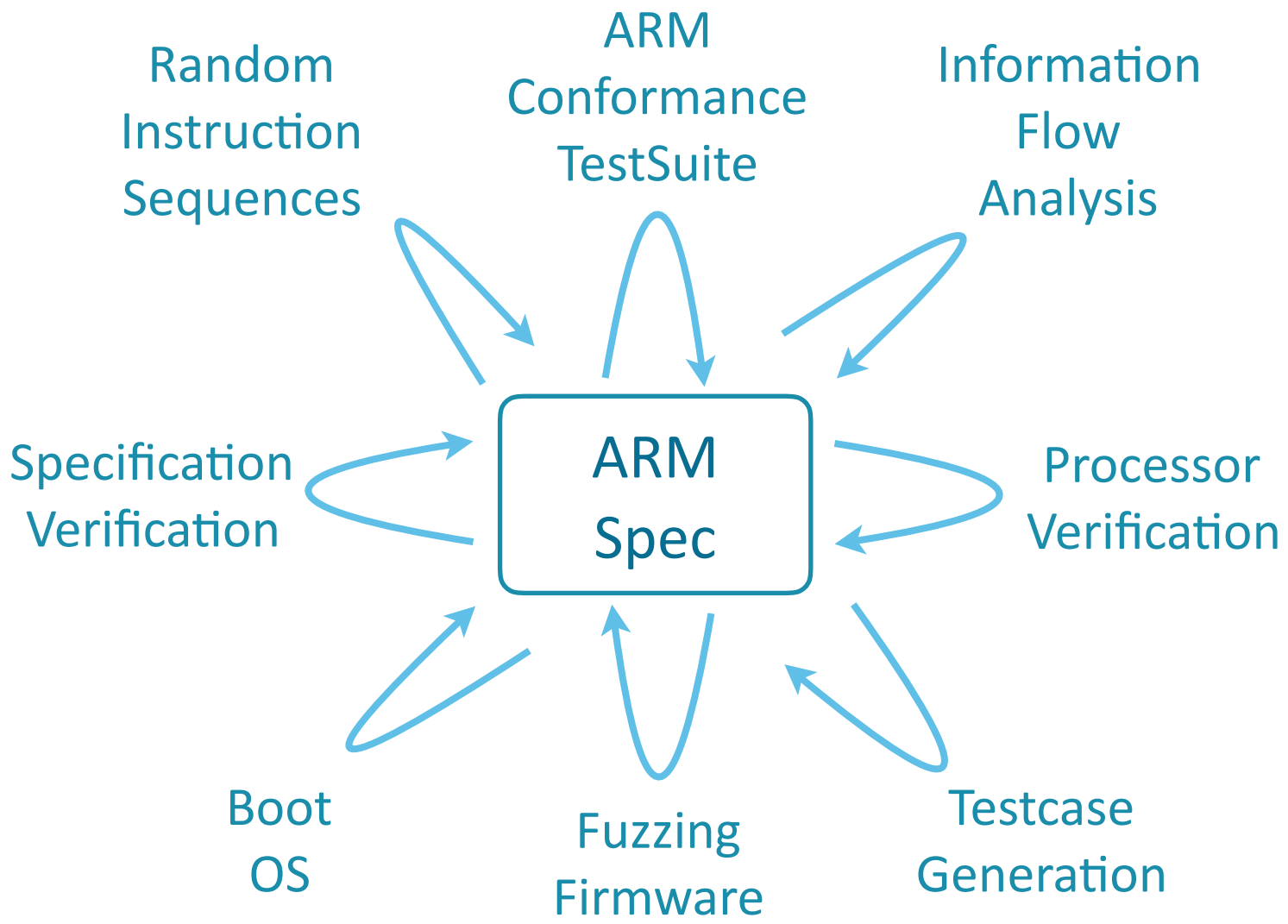
Measuring architecture coverage of tests

Untested: op1*op2 == -3.0, FPCR.RND=-Inf


```
bits(N) FPRSqrtStepFused(bits(N) op1, bits(N) op2)
    assert N IN {32, 64};
    bits(N) result;
    op1 = FPNeg(op1);    // per FMSUB/FMLS
    (type1,sign1,value1) = FPUnpack(op1, FPCR);
    (type2,sign2,value2) = FPUnpack(op2, FPCR);
    (done,result) = FPProcessNaNs(type1, type2, op1, op2, FPCR);
    if !done then
        inf1 = (type1 == FPType_Infinity);
        inf2 = (type2 == FPType_Infinity);
        zero1 = (type1 == FPType_Zero);
        zero2 = (type2 == FPType_Zero);
        if (inf1 && zero2) || (zero1 && inf2) then
            result = FPOnePointFive('0');
        elsif inf1 || inf2 then
            result = FPInfinity(sign1 EOR sign2, N);
        else
            // Fully fused multiply-add and halve
            result_value = (3.0 + (value1 * value2)) / 2.0;
            if result_value == 0.0 then
                // Sign of exact zero result depends on rounding mode
                sign = if FPCRRounding() == FPRounding_NEGINF then '1' else '0';
                result = FPZero(sign, N);
            else
                result = FPRound(result_value, FPCRRounding());
    return result;
```


arm

Creating a Virtuous Cycle



Lessons (Part 1)

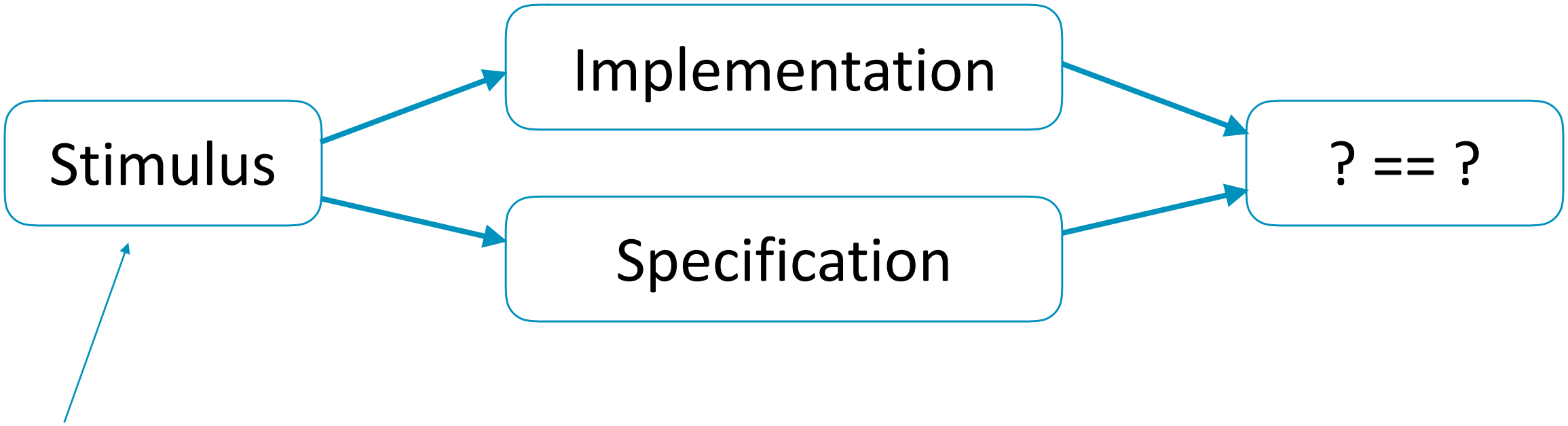
- Specifications contain bugs
- Huge value in being able to run existing test suites
  - Need to balance against benefits of non-executable specs
- Find ways to provide direct benefit to other users of spec
  - They will do some of the testing/debugging for you
  - They will support getting your changes/spec adopted as master spec
  - Creates Virtuous Cycle

"End to End Verification of ARM processors with ISA Formal," CAV 2016

Formal validation of processors

arm

Formal/Testing framework (deterministic specs)



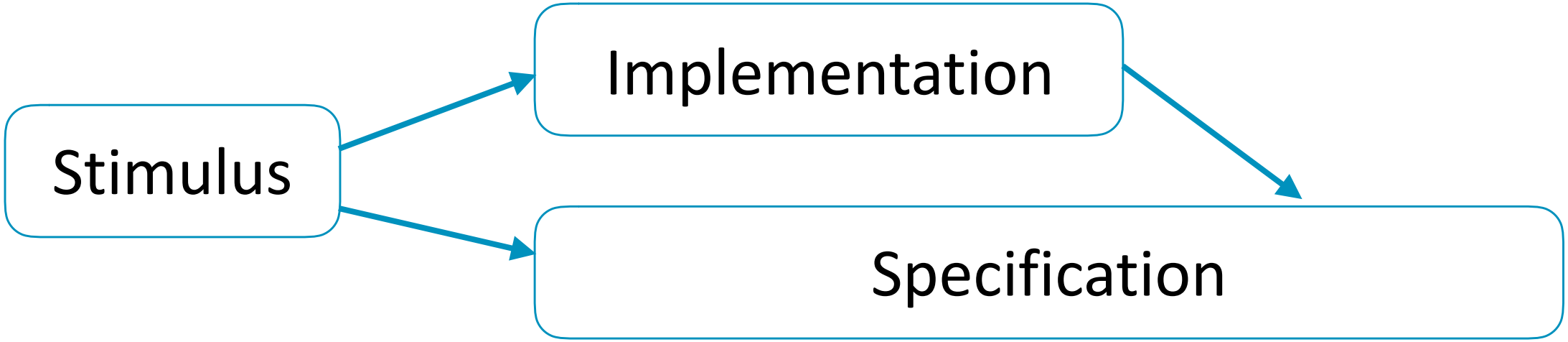
Test vectors

Bounded model checker

…

arm

Formal/Testing framework (non-deterministic specs)

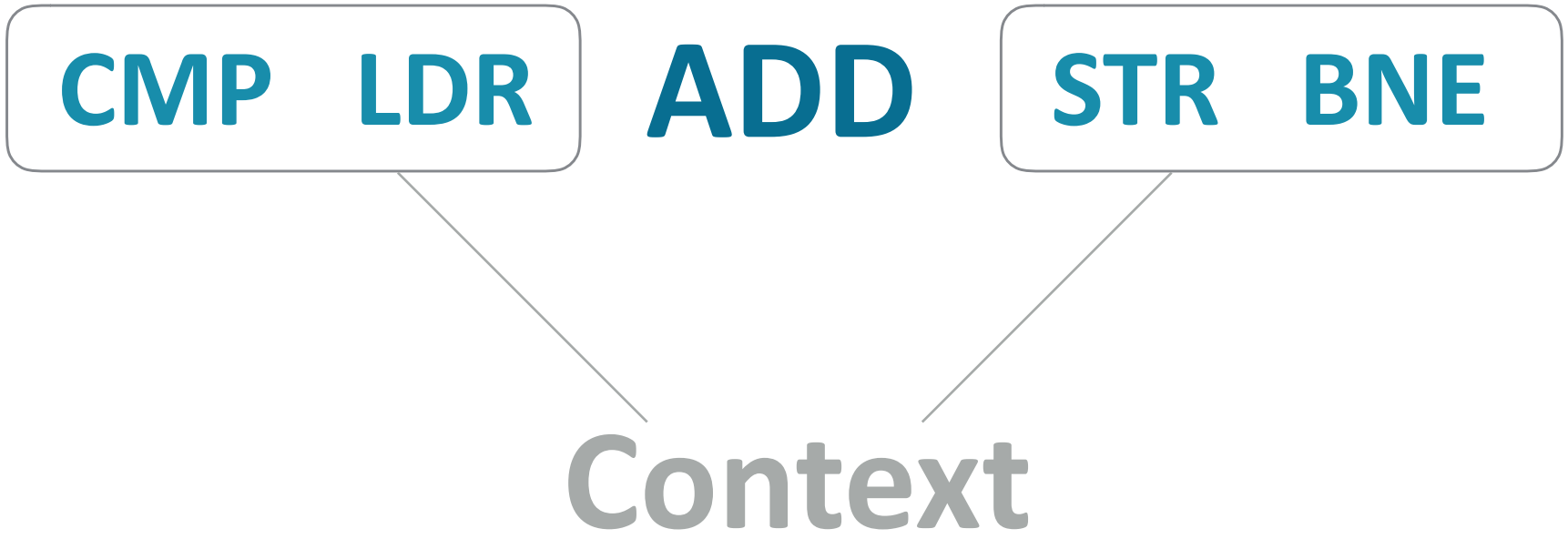


arm

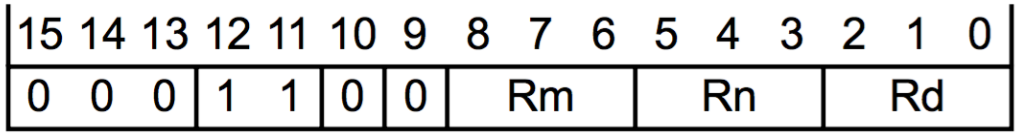
Checking an instruction

ADD

Checking an instruction



Specifying ADD




```
assign ADD_retiring = (pre.opcode & 16'b1111_1110_0000_0000)
                                  == 16'b0001_1000_0000_0000;
assign ADD_result   = pre.R[pre.opcode[8:6]] + pre.R[pre.opcode[5:3]];
assign ADD_Rd       = pre.opcode[2:0];

assert property (@(posedge clk) disable iff (~reset_n)
      ADD_retiring |-> (ADD_result == post.R[ADD_Rd]));
```

Specialize
Monomorphize
Constant Propagation
Width Analysis
Exception Handling
…

Arm CPUs verified with ISA-Formal

A-class	R-class	M-class
Cortex-A53	Cortex-R52	Cortex-M4
Cortex-A32	Next generation	Cortex-M7
Cortex-A35		Cortex-M33
Cortex-A55		Next generation
Next generation		

Cambridge Projects

Rolling out globally to other design centres

Sophia, France - Cortex-A75 (partial)

Austin, USA - TBA

Chandler, USA - TBA

arm

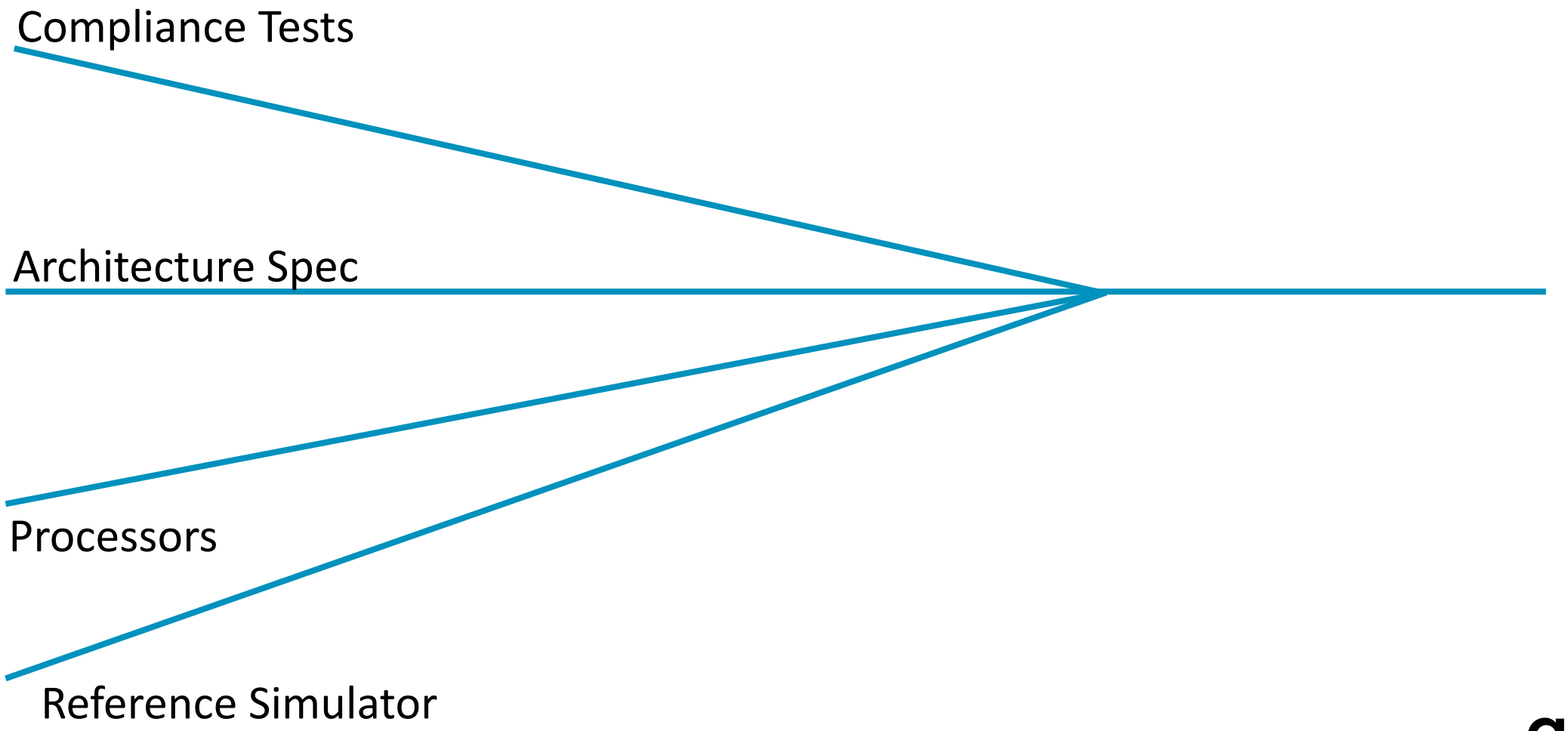
Lessons Learned (part 2)

- Very effective way to find bugs in implementations
- Very effective at finding bugs in spec
  - Try to find most of the bugs in your spec before you start
- Huge value in being able to use spec to validate implementations
  - Helps get formal spec adopted as part of official spec
  - Justifies investment in spec by implementors

Formal validation of specifications

arm

One Specification to rule them all?



arm

One Specification to rule them all?

Pro

- Authoritative

- Easier to maintain

Con

- No redundancy

- Extending specification is harder

Creating a redundant specification

Where to get a list of redundant properties from?

How to formalise this list?

How to formally validate specification against properties?

(This may look familiar from formal specification of software)

Rule JRJC

Exit from lockup is by any of the following:

- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority processor exception.

Rule R

State Change X is by any of the following:

- Event A
- Event B
- State Change C
- Event D

Rule R

State Change X is by any of the following:

- Event A
- Event B
- State Change C
- Event D

And cannot happen any other way

arm

Rule R

State Change X is by any of the following:

- Event A
- Event B
- State Change C
- Event D

And cannot happen any other way

Rule R: $X \rightarrow A \vee B \vee C \vee D$

arm

State Change X	Exit from lockup	Fell(LockedUp)
Event A	A Cold reset	Called(TakeColdReset)
Event B	A Warm reset	Called(TakeReset)
State Change C	Entry to Debug state	Rose(Halted)
Event D	Preemption by a higher priority processor exception	Called(ExceptionEntry)

“Eyeball Closeness”

Rule JRJC

Exit from lockup is by any of the following:

- A Cold reset.
- A Warm reset.
- Entry to Debug state.
- Preemption by a higher priority processor exception.

$$\begin{aligned} \text{Fell(LockedUp)} \rightarrow\ & \text{Called(TakeColdReset)} \\ \vee\ & \text{Called(TakeReset)} \\ \vee\ & \text{Rose(Halted)} \\ \vee\ & \text{Called(ExceptionEntry)} \end{aligned}$$

arm

Rule VGNW

Entry to lockup from an exception causes

- Any Fault Status Registers associated with the exception to be updated.

Out of date
- No update to the exception state, pending or active.

Misleading
- The PC to be set to 0xEFFFFFFE.

Untestable
- EPSR.IT to become UNKNOWN.

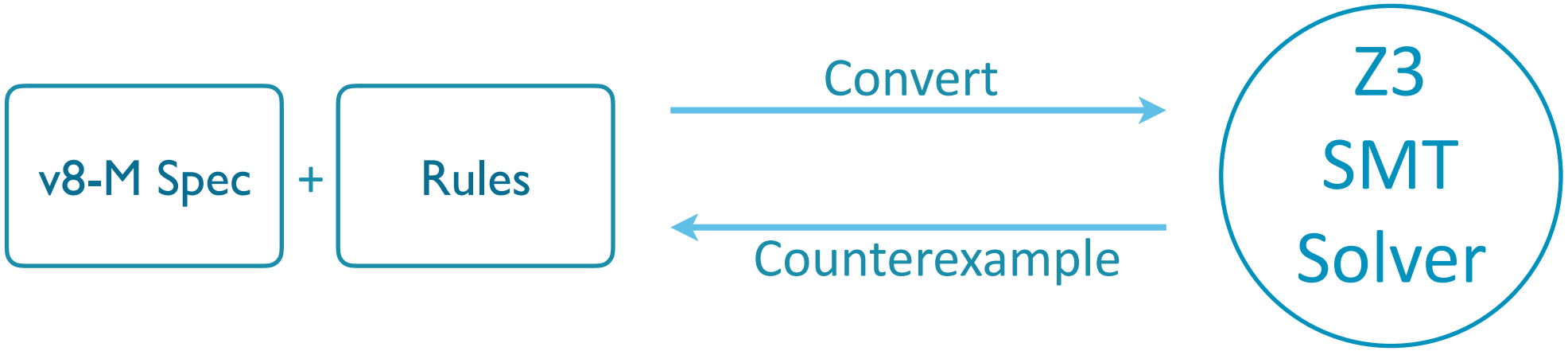
Ambiguous

In addition, HFSR.FORCED is not set to 1.

arm

~10,000 lines

~1,000,000 lines



arm

Results (more in OOPSLA paper)

Most properties proved in under 100 seconds

Found 12 bugs in specification:

- debug, exceptions, system registers, security

Found bugs in English prose:

- ambiguous, imprecise, incorrect, …

Lessons Learned (part 3)

- Redundancy essential for detecting errors
- Need set of ‘orthogonal’ properties
  - Invariants
  - Security properties
  - Reachability properties
  - etc.
- Eyeball closeness

Creating Formal Specifications of Real World Artifacts

Plan for adoption into official specs

Test your specification

Build a virtuous cycle

- What is “killer app” of your spec?

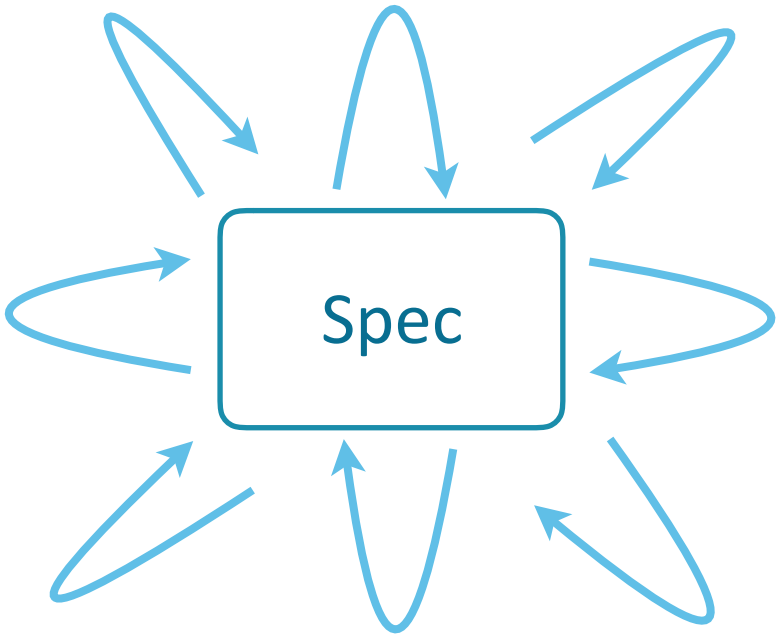
  Formally validation of implementations?

- Look for early adopters

- Ensure specifications have many uses

  Don’t write spec in Coq/HOL/ACL2/…

Create redundant specifications



arm

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!

@alastair_d_reid

arm

“Trustworthy Specifications of the ARM v8-A and v8-M architecture,” FMCAD 2016

“End to End Verification of ARM processors with ISA Formal,” CAV 2016

“Who guards the guards? Formal Validation of ARM v8-M Specifications,” OOPSLA 2017