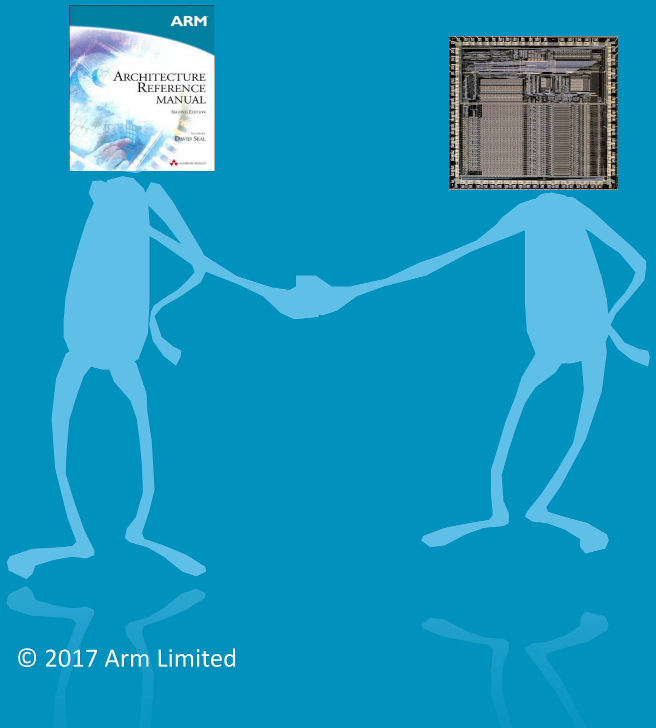
Who guards the guards?

Formal validation of the Arm v8-M Architecture Specification



Alastair Reid

Arm Research

@alastair_d_reid

OOPLSA 2017

Uses of formal processor specifications

Writing compilers, operating systems, …

Formally verifying compilers, operating systems, …

Program synthesis

Security analysis

Malware analysis

Formally verifying processor implementations

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)

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

Arm Processor Specifications

A-class (phones, tablets, servers, …)

6,000 pages
40,000 line formal specification

Instructions (32/64-bit)
Exceptions / Interrupts
Memory protection
Page tables
Multiple privilege levels
System control registers
Debug / trace

M-class (microcontrollers, IoT)

1,200 pages
15,000 line formal specification

Instructions (32-bit)
Exceptions / Interrupts
Memory protection
~~Page tables~~
Multiple privilege levels
System control registers
Debug / trace

Is my specification correct?

Is my specification correct?

Testing

Is my specification correct?

Formal Validation of CPUs

Testing

Is my specification correct?

Formal Validation of CPUs

Multiple Users

Testing

Executable Specification

Defines what *is* allowed

Animation	→	Check spec matches expectation
Testable	→	Compare spec against implementation

Executable Specification

Defines what *is* allowed

Animation → Check spec matches expectation

Testable → Compare spec against implementation

Does *not* define what is *not* allowed

e.g., Impossible states, impossible actions/transitions, security properties

No redundancy

Problem when extending specification

Creating a specification of disallowed behaviour

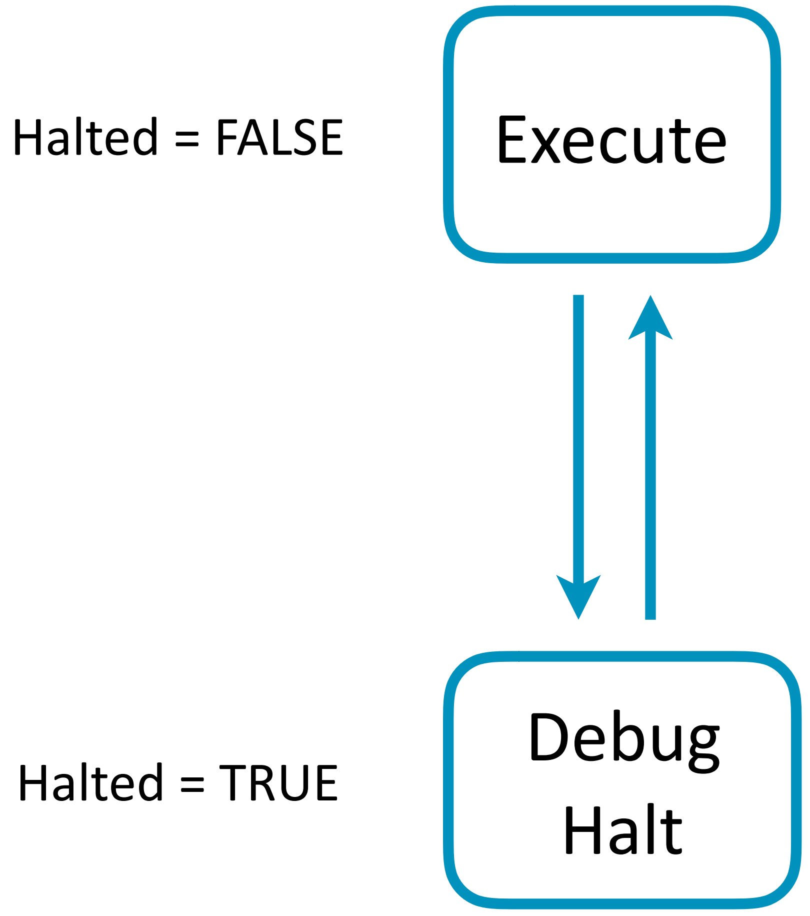
Where to get a list of disallowed behaviour?

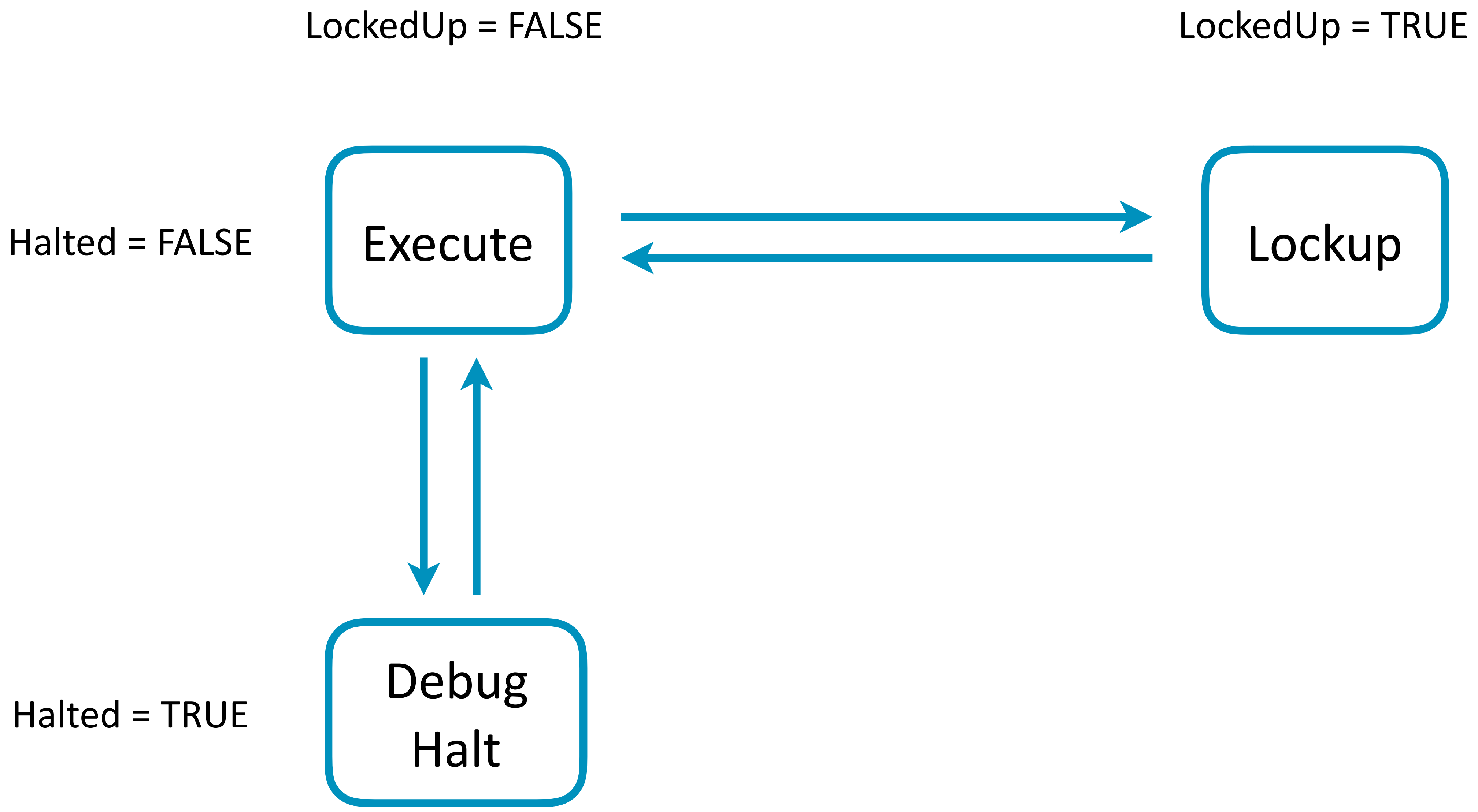
How to formalise this list?

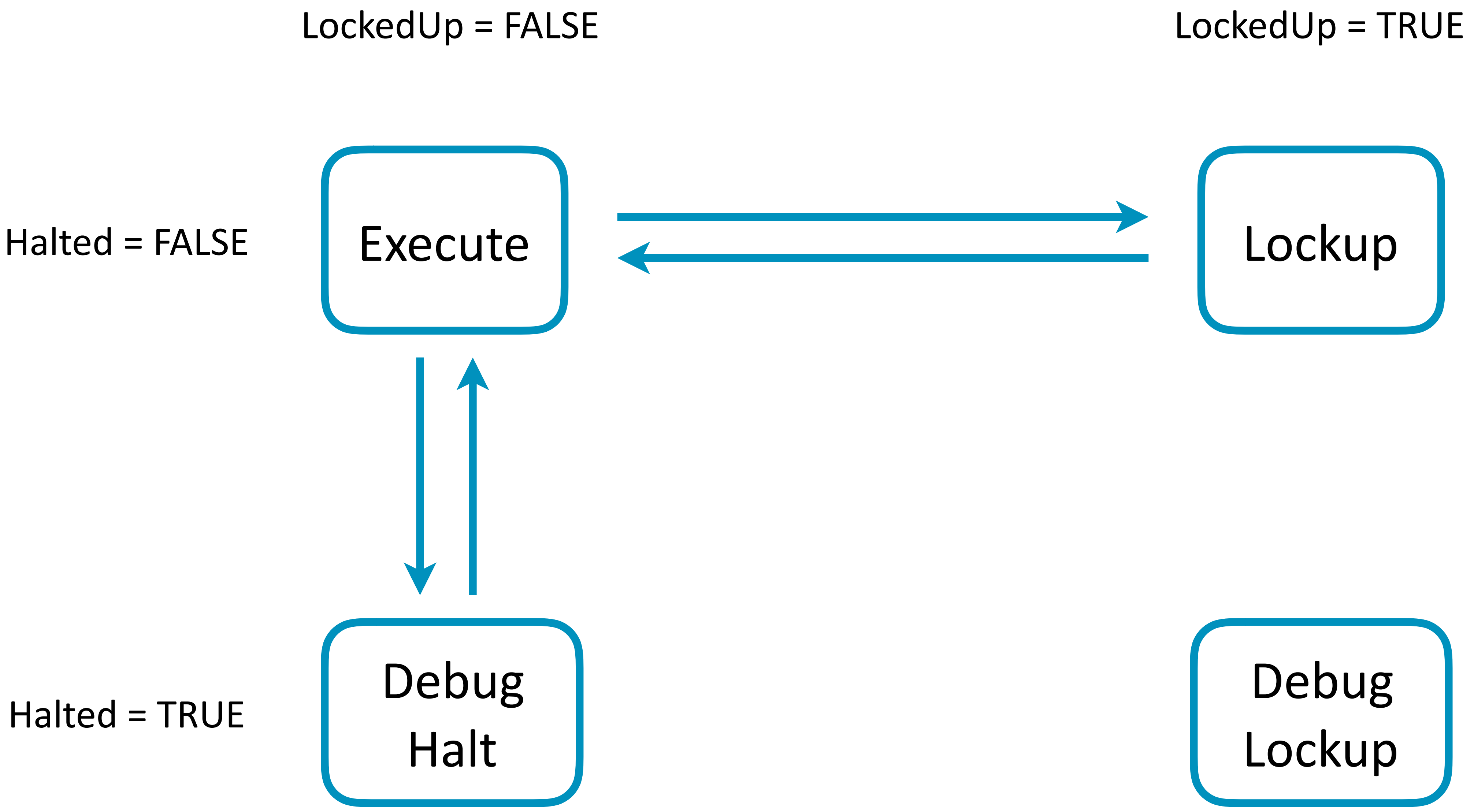
How to formally validate specification against spec of disallowed behaviour?

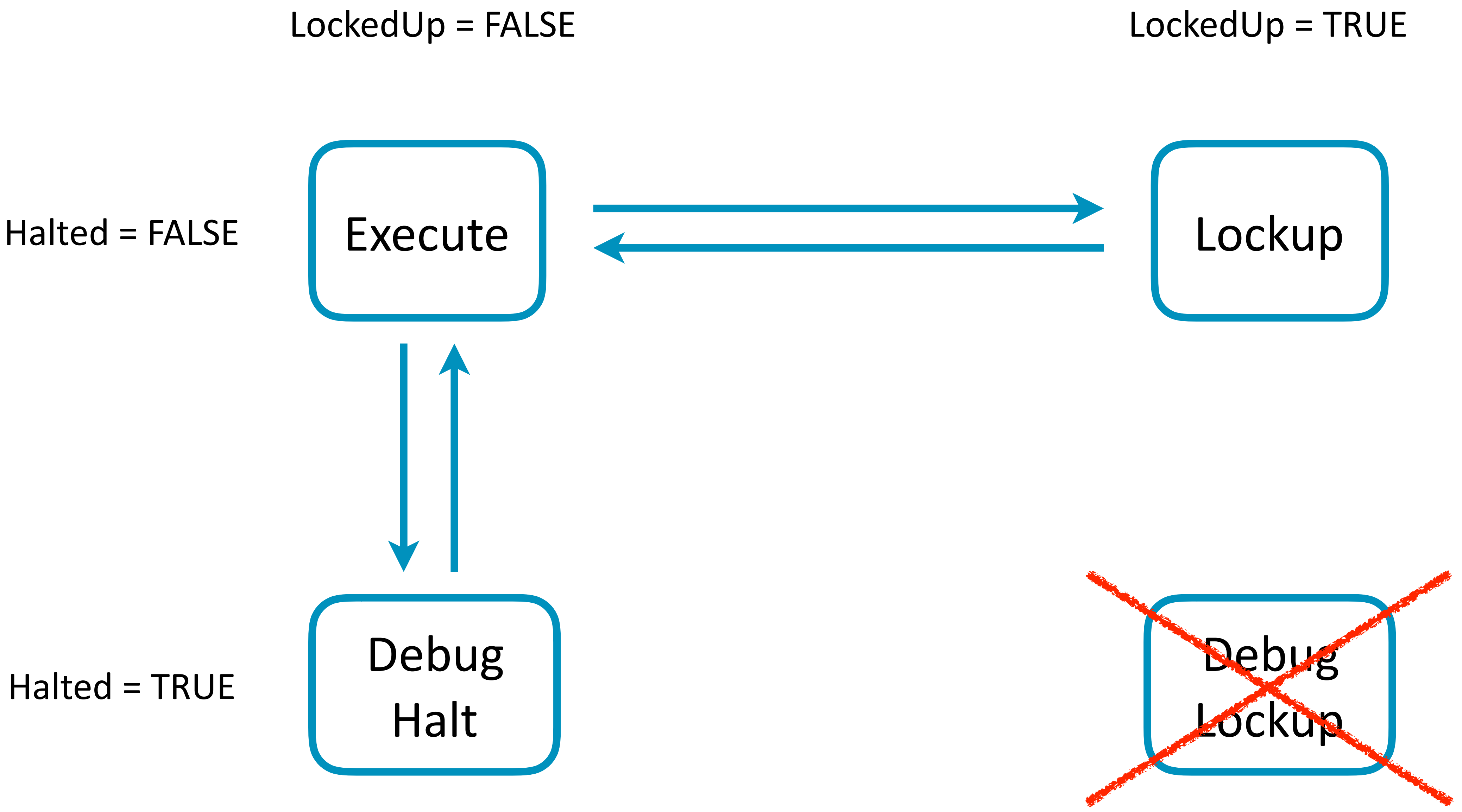
(This may look familiar from formal specification of software)

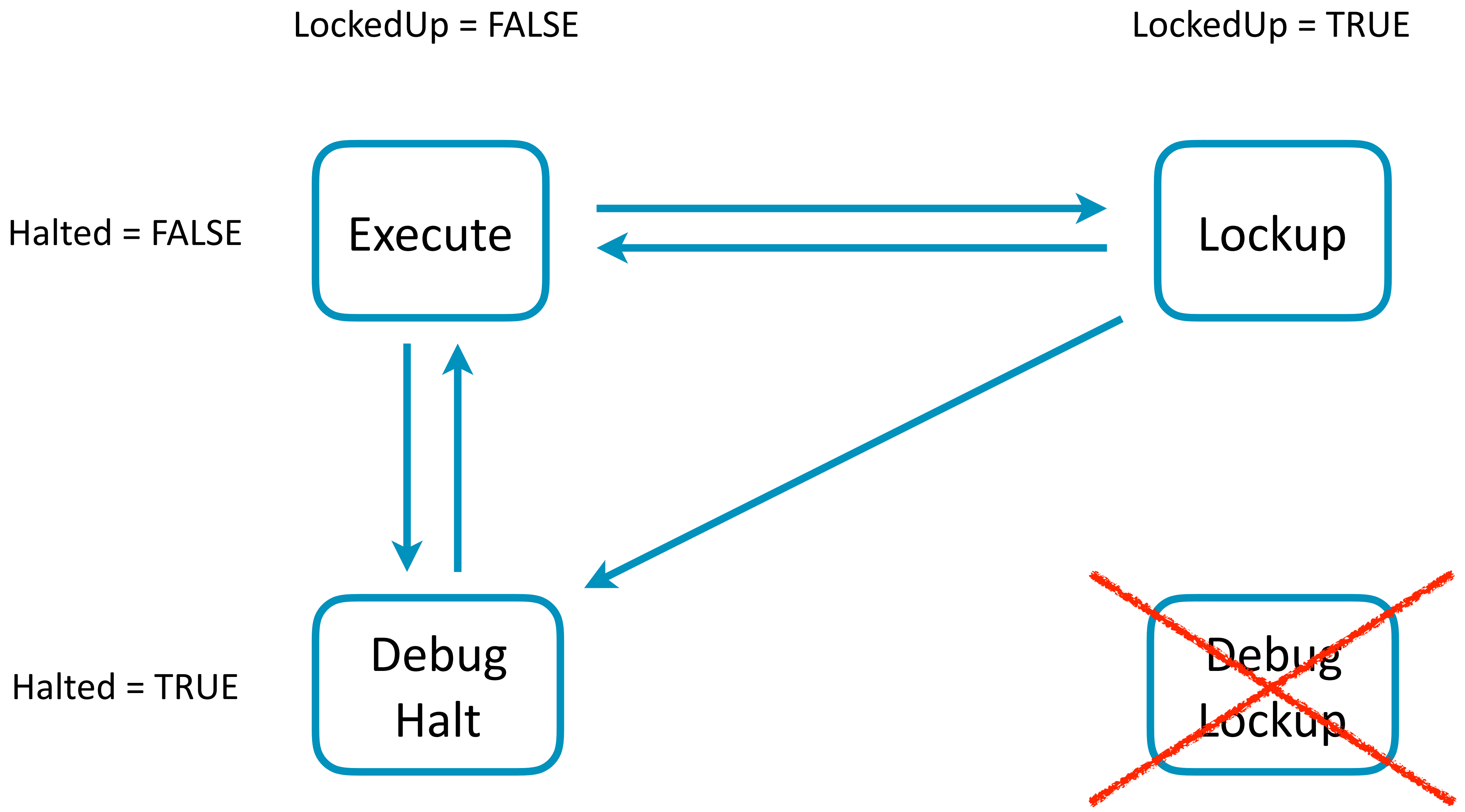
Execute


Halted = FALSE
Execute
Halted = TRUE
Debug
Halt


LockedUp = FALSE
LockedUp = TRUE
Halted = FALSE
Execute
Lockup
Halted = TRUE
Debug Halt


LockedUp = FALSE
LockedUp = TRUE
Halted = FALSE
Halted = TRUE
Execute
Lockup
Debug Halt
Debug Lockup


LockedUp = FALSE
LockedUp = TRUE
Halted = FALSE
Halted = TRUE
Execute
Lockup
Debug Halt
~~Debug Lockup~~


LockedUp = FALSE
LockedUp = TRUE
Halted = FALSE
Execute
Lockup
Halted = TRUE
Debug Halt
~~Debug Lockup~~

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

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

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)

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

Rule 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 become UNKNOWN.

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

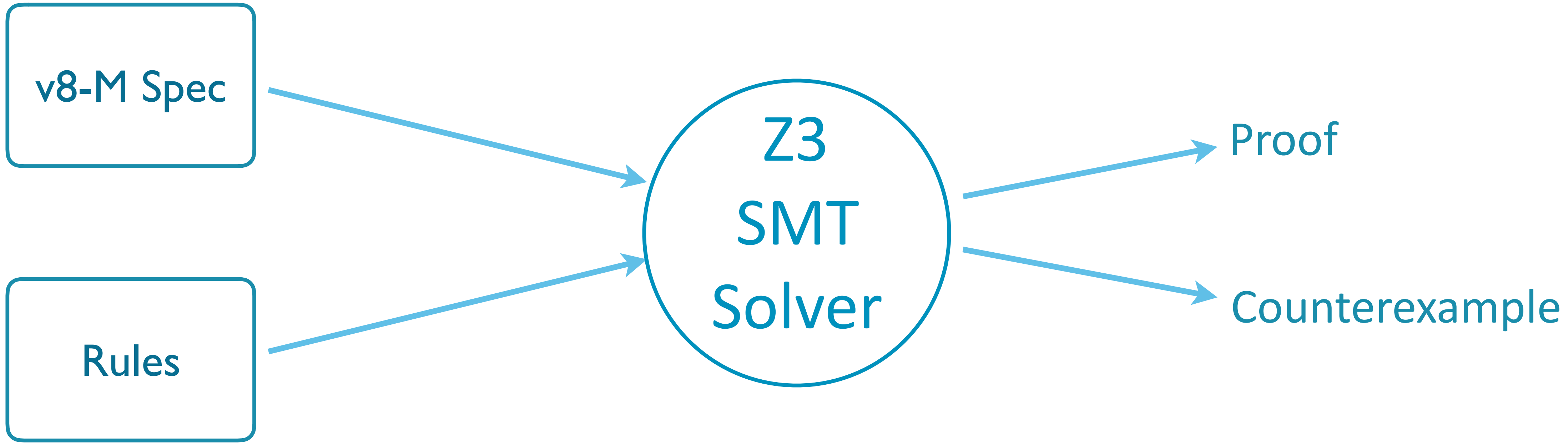
Out of date

Misleading

Untestable

Ambiguous

arm


v8-M Spec
Rules
Z3
SMT
Solver
Proof
Counterexample

Temporal Operators

$$\boxed{\text{Fell(LockedUp)}} \rightarrow \text{Called(TakeColdReset)} \\ \lor \text{Called(TakeReset)} \\ \lor \boxed{\text{Rose(Halted)}} \\ \lor \text{Called(ExceptionEntry)}$$

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

Event Operators

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

Temporal Operators

$$\begin{tikzcd}
\text{Fell(e)} \arrow[d] & \text{Stable(e)} \arrow[d] & \text{Rose(e)} \arrow[d] \\
\text{Past(e)} > \text{e} & \text{Past(e)} = \text{e} & \text{Past(e)} < \text{e}
\end{tikzcd}$$

Temporal Operators

Fell(LockedUp)

↓

```
__Past_LockedUp = LockedUp;

FunctionUnderTest();

… __Past_LockedUp > LockedUp …
```

Event Operators

Called(TakeReset)

$$\downarrow$$

```
TakeReset()
{
        __Called_TakeReset = TRUE;
        …
}
```

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.

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

```
__Called_TakeColdReset      = FALSE;    assert((__Past_LockedUp > LockedUp)
__Called_TakeReset          = FALSE;           ==>
__Called_TakeExceptionEntry = FALSE;           (   __Called_TakeColdReset
__Past_LockedUp = LockedUp;                    || __Called_TakeReset
__Past_Halted   = Halted;                      || __Past_Halted < Halted
                                               || __Called_ExceptionEntry));
```

Arm Specification Language $\longrightarrow$ SMT

Arithmetic operations	Arithmetic operations
Boolean operations	Boolean operations
Bit Vectors	Bit Vectors
Arrays	Arrays
Functions	~~Functions~~
Local Variables	~~Local Variables~~
Statements	~~Statements~~
Assignments	~~Assignments~~
If-statements	~~If-statements~~
Loops	~~Loops~~
Exceptions	~~Exceptions~~

Results (more in 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, …

Summary

Formalization of large, complex specifications

Executable specifications have a fatal flaw

Need specification of disallowed behaviour

Manually formalized structured English prose

Used SMT checker to find bugs in both spec and prose

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

@alastair_d_reid

arm

See also:
“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