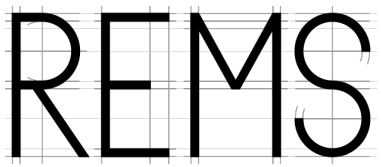
Adventures in Mechanising and Verifying WebAssembly

Conrad Watt

University of Cambridge, UK

Formal Methods Meets JavaScript, VeTSS



The web's evolution

- We want richer web apps - 3D rendering, physics, 60fps.
- Asm.js exists but is too slow and janky.
- We're at the limits of JavaScript - need a purpose-built language.

Peter Sewell

Professor of Computer Science, Computer Laboratory, University of Cambridge
Member of the Cambridge Programming, Logic, and Semantics Group
Fellow of Wolfson college

Here are my contact details, a photo, short bio, and CV

PhD students, RAs, and Co-authors Meetings Funding Papers (by date) Papers (by topic)

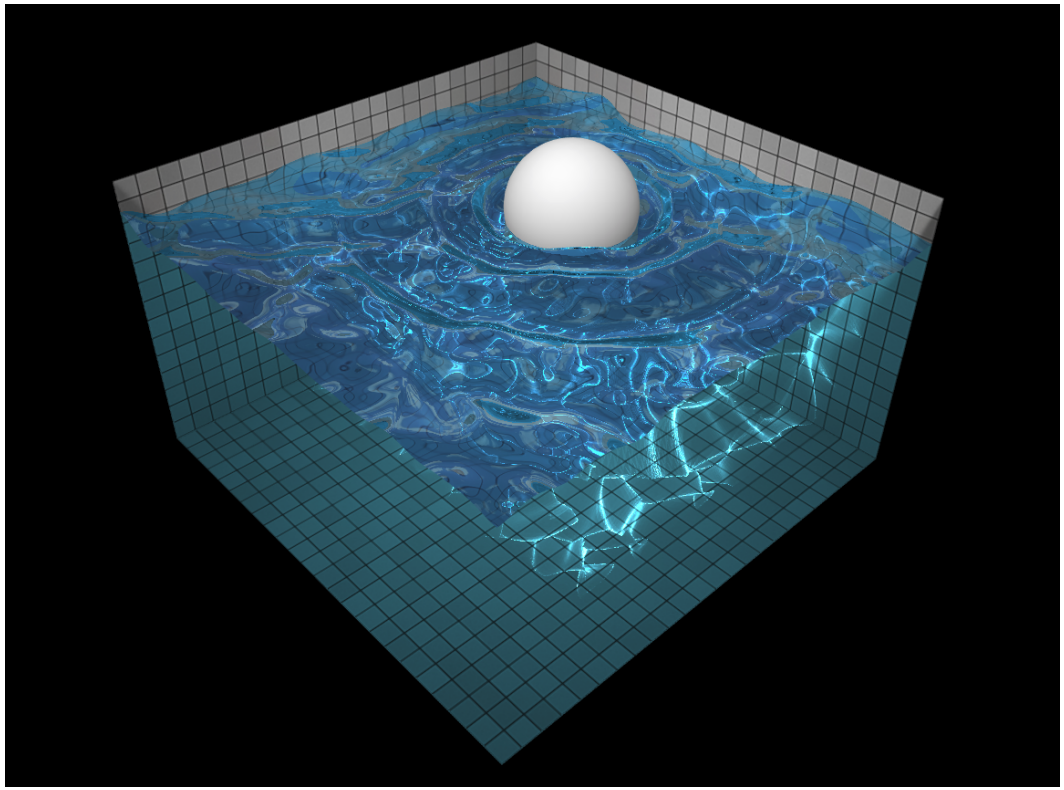
Teaching

- The 2017-18 Part 1B *Semantics of Programming Languages* course.
- The 2017-18 *Multicore Semantics and Programming (R204)* ACS MPhil module
- ...previous teaching

http://www.cl.cam.ac.uk/~pes20/

The web's evolution

- We want richer web apps - 3D rendering, physics, 60fps.
- Asm.js exists but is too slow and janky.
- We're at the limits of JavaScript - need a purpose-built language.



https://github.com/evanw/webgl-water

What is WebAssembly?

- A web-friendly bytecode.
- Runs on any browser.
- “Near-native” performance.
- Targetted by LLVM.

WebAssembly is weird

A stack reduction semantics...

```
i32.const 4
i32.const 2
i32.const 1
i32.add
i32.add
```

Type: [i32]

```
i32.const 4
i32.const 3
i32.add
```

Type: [i32]

```
i32.const 7
```

Type: [i32]

WebAssembly is weird

...but allows only structured control flow.

```
loop
  i32.const 4
  i32.const 2
  i32.const 1
  i32.add
  i32.add
  br 0
end
```

```
label{...}
  i32.const 4
  i32.const 3
  i32.add
  br 0
end
```

```
label{...}
  i32.const 7
  br 0
end
```

```
loop
  i32.const 4
  i32.const 2
  i32.const 1
  i32.add
  i32.add
  br 0
end
```

Note

label is an "administrative" operation. It represents the loop unrolled once, keeping track of the continuation (abbreviated).

The WebAssembly type system

- All WebAssembly programs must be validated (typed) before execution.
- WebAssembly instruction types have the form t* $\rightarrow$ t*

```
i32.const 4
```

Type:
[] $\rightarrow$ [i32]

```
i32.add
i32.add
```

Type:
[i32, i32, i32] $\rightarrow$ [i32]

```
f32.const 0
i32.const 4
i32.add
```

Type:
$\bot$

The WebAssembly type system

Preservation

If a program P is validated with a type ts, the program obtained by running P one step to P' can also be validated with type ts.

Progress

For any validated program P that is not a list of constant values or a bare trap result, there exists P' such that P reduces to P'

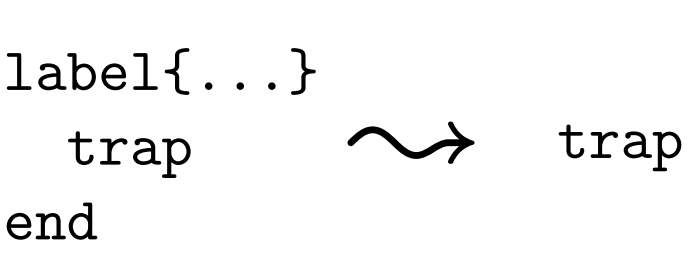
Initial mechanisation and soundness proof

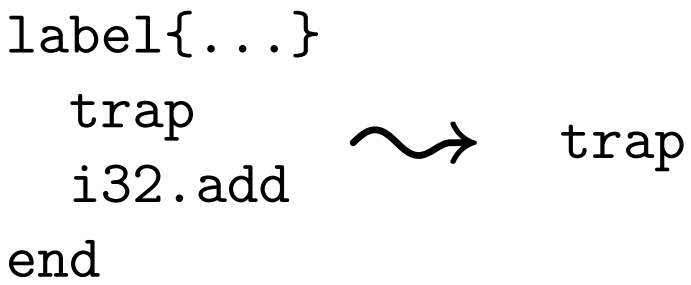
- Initially based on an accepted draft of the WASM group's PLDI paper[1] combined with the draft specification.
- Definitions and proofs in Isabelle.
- Type soundness properties: preservation and progress.
- Progress property as stated in the draft had a trivial counterexample.

[1] Andreas Haas et al. "Bringing the Web Up to Speed with WebAssembly". In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2017. New York, NY, USA: ACM, 2017, pp. 185–200.

Problems found - administrative instructions

- Exceptions did not properly propagate through administrative instructions.
- Malformed, irreducible nestings of administrative instructions containing a `return` opcode could be well-typed.
- Our suggested fixes were incorporated into the specification.





Problems found

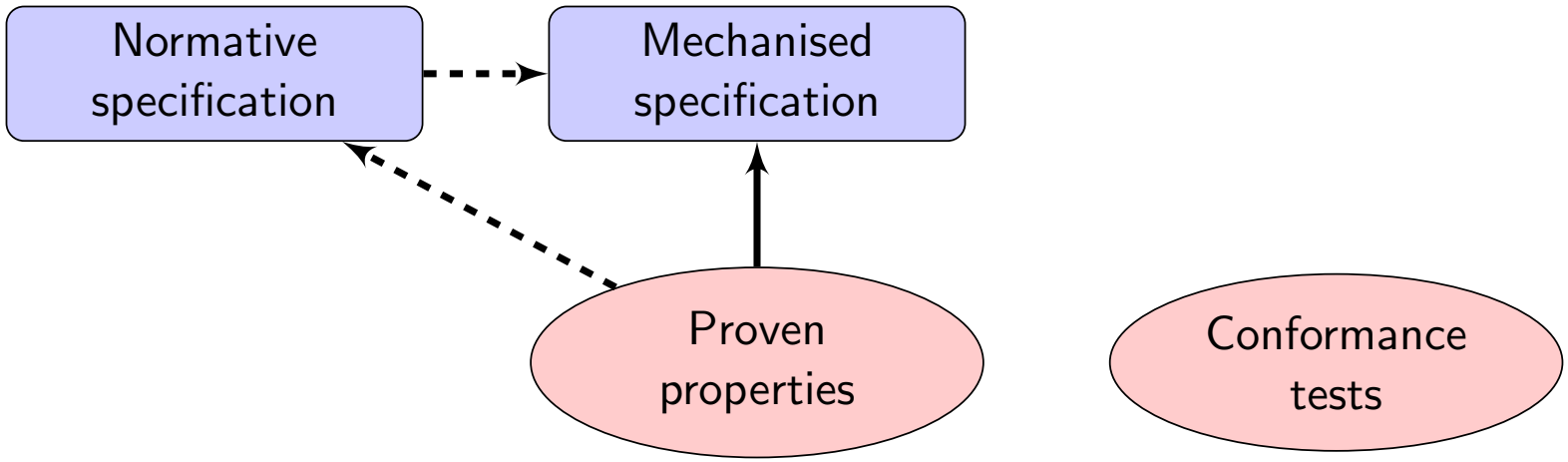
- Various trivial mistakes in the constraints of casting instructions.
- Big one - host function interface was unsound.[2]
- After these changes, managed to get a fully mechanised proof of soundness! (∼5000 LOC)

[2] Andreas Rossberg. *[spec] Fix and clean up invariants for host functions*. Sept. 2017. URL: https://github.com/WebAssembly/spec/pull/563.

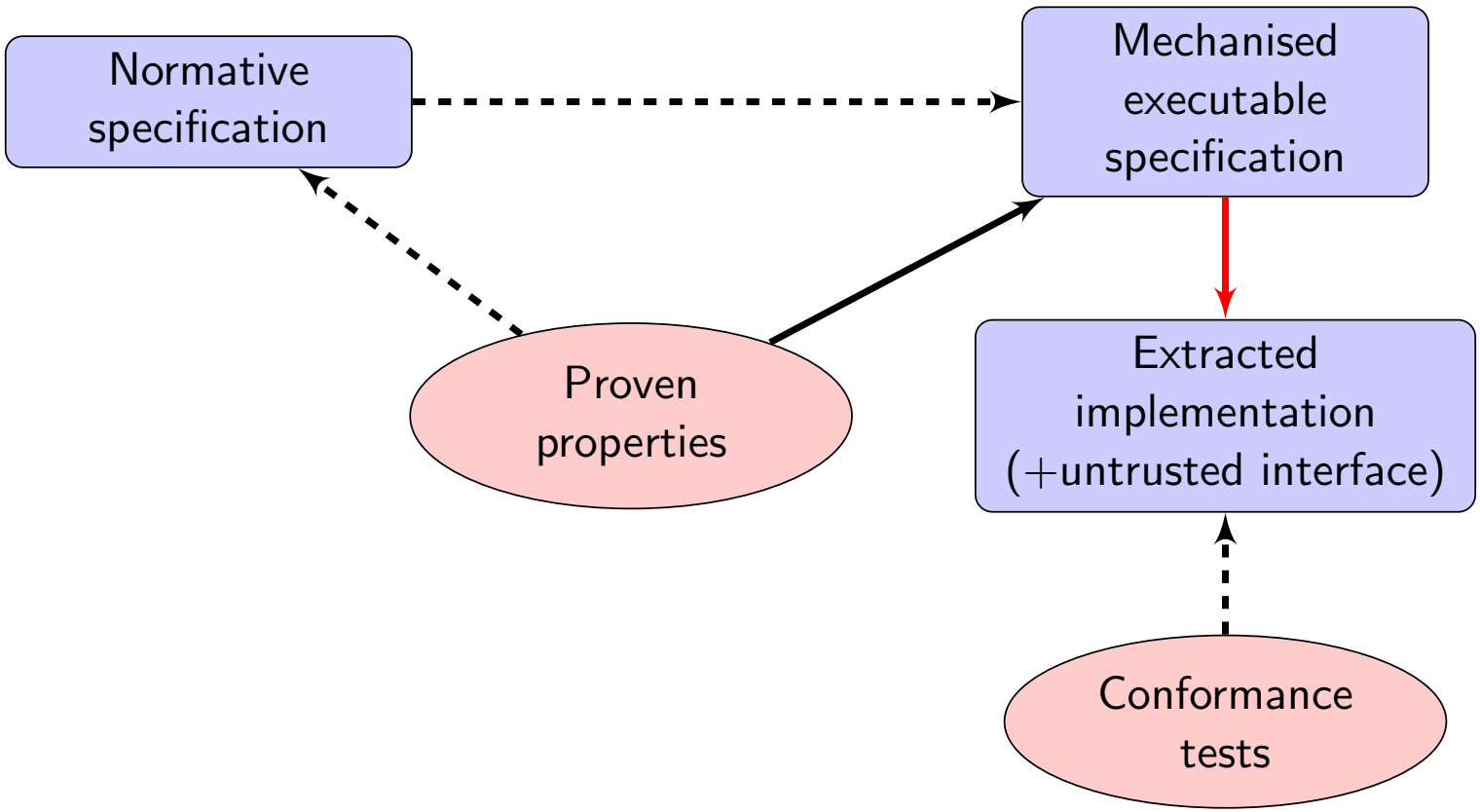
Verified reference interpreter

- Directly animating the mechanised specification was infeasible.
- For the reduction relation - exception propagation is non-deterministic (but confluent), and the specification leans heavily on recursively defined evaluation contexts.
- For the typing judgement - there is a weakening rule with no upper bound, and the rules for typing dead code(!) involve a high degree of polymorphism - not syntax-directed.
- Some of these problems are solvable by re-formulating the mechanisation, but wanted eyeball-closeness with the official specification.

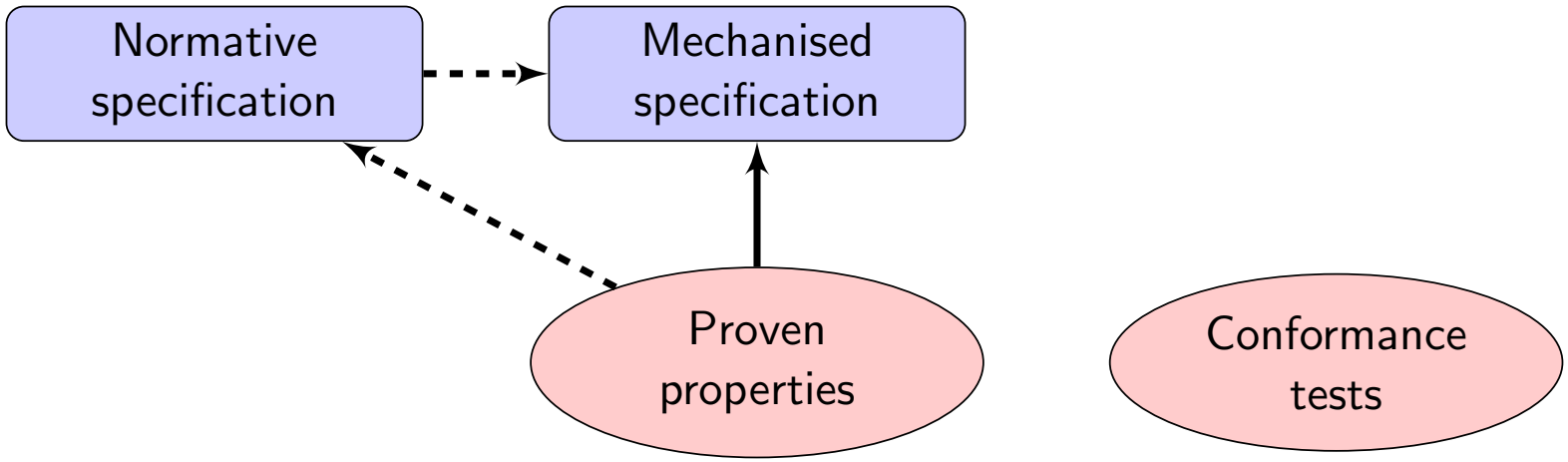
The flow of trust



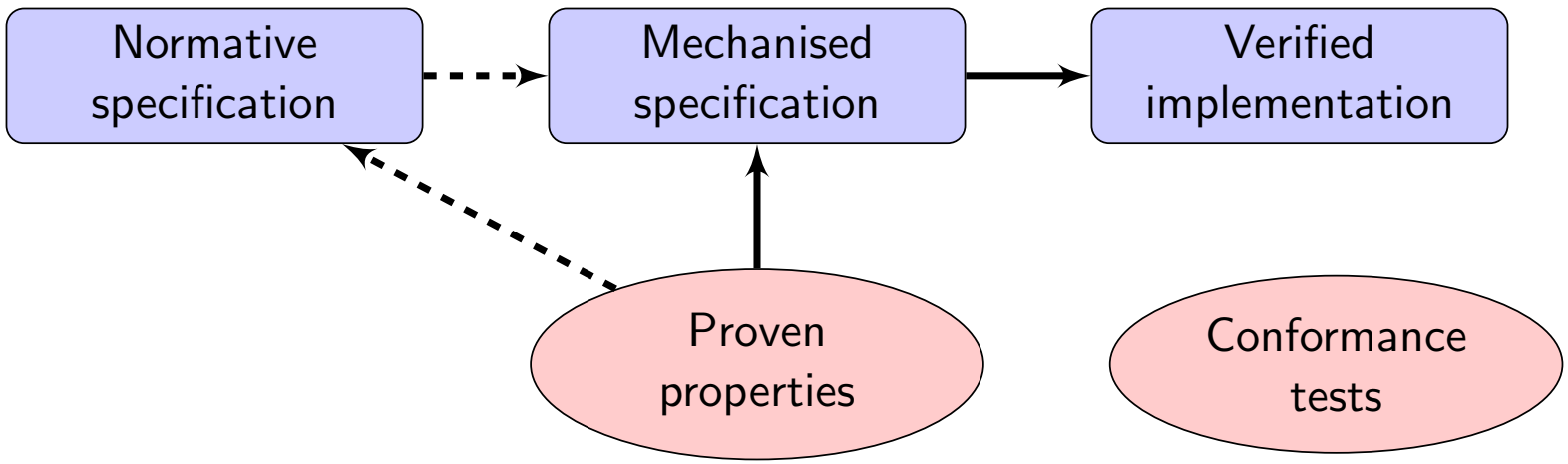
The flow of trust



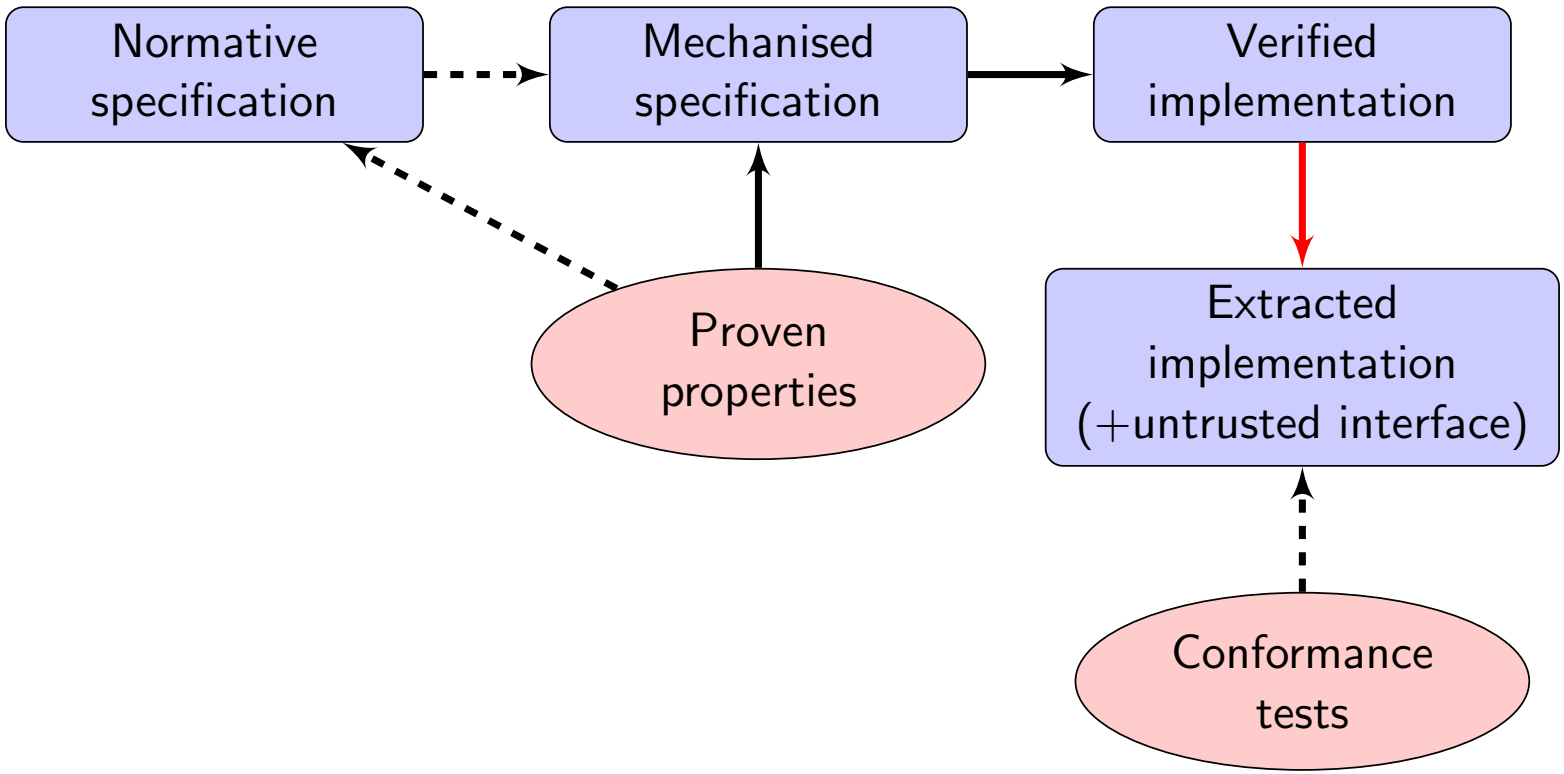
The flow of trust



The flow of trust



The flow of trust



Solution

- A separate reference interpreter, and typechecker.
- Proof of correctness between the inductive rules of the model, and the executable definitions of the interpreter and typechecker.
- Attempted fuzzing using interpreter as a test oracle - only found crash bugs in industry tools unfortunately.

Next steps

- The threads proposal!
- We've already seen that specifying interop between JS and WebAssembly isn't trivial, but this is on another level.
- Need a compatible axiomatic weak memory model.
- But more complicated than JS: WASM memory can change size, but (until now) SharedArrayBuffers cannot.

Next steps

- Already finding bugs in the JS memory model.[3]

```
Atomics.wait(tA, 0, 0)          ||  Atomics.store(tA, 0, 1)
var x = Atomics.load(tA, 0)     ||  Atomics.wake(tA, 0, 1)
```

- Full formal spec for WebAssembly threading is being drafted.
- Mechanisation? Not impossible, but meaningful proofs could be a lot of work.

[3] Conrad Watt. *Normative: Strengthen Atomics.wait/wake synchronization to the level of other Atomics operations*. Mar. 2018. URL: https://github.com/tc39/ecma262/pull/1127.

Future work

- Continue looking at SharedArrayBuffer, WASM threads.
- Verifying ct-wasm (watch this space!).
- Model module instantiation.
- Look at Ethereum's EVM2.0 (?)