Formal Methods and the WebAssembly Specification

Conrad Watt [1] Andreas Rossberg [2] Jean Pichon-Pharabod [1]

[1]University of Cambridge

[2]Dfinity Stiftung

SREPLS'11

REMS

A brief history of JavaScript

- Prototyped in 10 days, in 1995.
- We're stuck with it now.
- Every website relies on it (almost).
- Accumulated technical debt weighs heavy on the spec.

The web's evolution

- We want richer web apps - 3D rendering, physics, 60fps.
- asm.js exists but is limited by being built on top of JavaScript.
- 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)

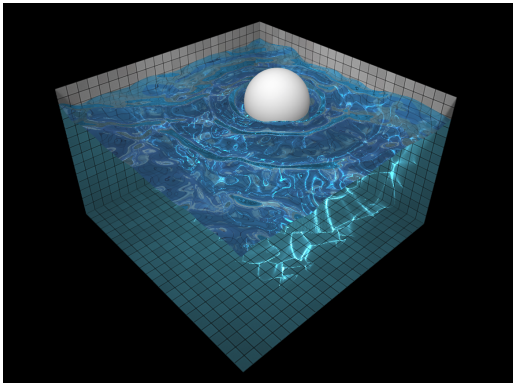
Teaching

- 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

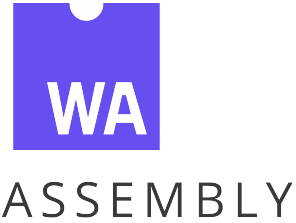
- We want richer web apps - 3D rendering, physics, 60fps.
- asm.js exists but is limited by being built on top of JavaScript.
- We're at the limits of JavaScript - need a purpose-built language.



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

What is WebAssembly?

- A web-friendly bytecode.
- Runs on any browser.
- "Near-native" performance.
- Targetted by LLVM.
- Formally specified! [1]



[1] Andreas Rossberg et al. "Bringing the Web Up to Speed with WebAssembly". In: *Communications of the ACM* 61.12 (Nov. 2018), pp. 107–115. ISSN: 0001-0782. DOI: 10.1145/3282510. URL: http://doi.acm.org/10.1145/3282510.

WebAssembly execution

A small-step 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
```

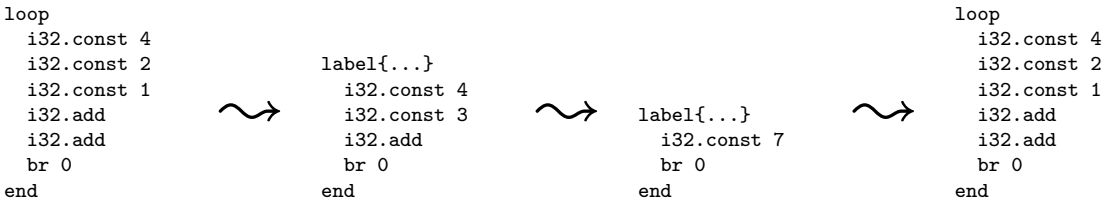
Type: [i32]

```
i32.const 7
```

Type: [i32]

WebAssembly execution

...but allows only structured control flow.



Note

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

WebAssembly type system

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

```
i32.const 4
```

Type:
$[] \to [\text{i32}]$

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

Type:
$[\text{i32}, \text{i32}, \text{i32}] \to [\text{i32}]$

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

Type:
$\bot$

WebAssembly type system

Preservation

If a program P is validated with a type ts, any program obtained by reducing P to P' can also be validated with type ts.

Progress

For any validated program P that has not terminated with a result, there exists P' such that P reduces to P'

These properties together guarantee syntactic type soundness.[2]

[2] A.K. Wright and M. Felleisen. "A Syntactic Approach to Type Soundness". In: *Information and Computation* 115.1 (1994). ISSN: 0890-5401.

Mechanisation

- An unambiguous formal specification and an unambiguous correctness condition.
- Perfect for mechanisation!
- ∼11,000 lines of Isabelle/HOL.[3]
- Found several errors in the draft specification.
- Also included:
  - Verified sound and complete type-checking algorithm.
  - Verified sound run-time interpreter.

[3] Conrad Watt. "WebAssembly". In: *Archive of Formal Proofs* (Apr. 2018). http://isa-afp.org/entries/WebAssembly.html, Formal proof development. ISSN: 2150-914x.

Mechanisation

Two categories of errors were found.

- Trivial "syntactic" errors:
  - typos, obviously malformed constraints
  - missing conditions/cases
- Deeper "semantic" errors:
  - edge-cases where well-typed programs get stuck
  - sound inter-op with JavaScript/the host environment

Mechanisation

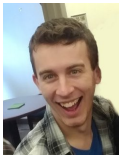
Two categories of errors were found.

- Trivial "syntactic" errors:
  - often discovered because of Isabelle's type-checked metatheory
  - don't need the full power of an interactive theorem prover
- Deeper "semantic" errors:
  - discovered during the soundness proof
  - difficult to find by hand/light-weight specification

Mechanisation

CT-Wasm

Secure information flow type system.



John Renner

Natalie Popescu

Sunjay Cauligi

Deian Stefan

UC San Diego

Wasm Logic

A separation logic for WebAssembly.

Petar Maksimović*

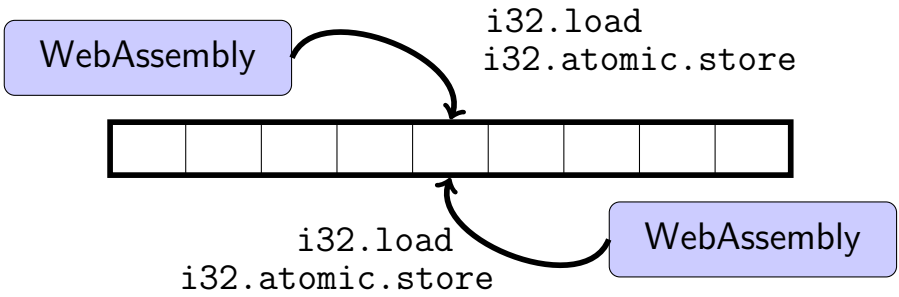
Neel Krishnaswami†

Philippa Gardner*

Imperial College London*/Cambridge†

Relaxed memory

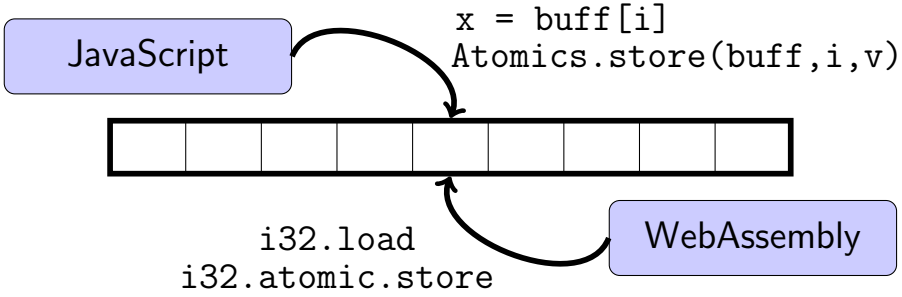
- WebAssembly program can read from and write to a linear buffer of raw bytes.
- Adding threads, these buffers can now be shared.
- Need a relaxed memory model.



Relaxed memory

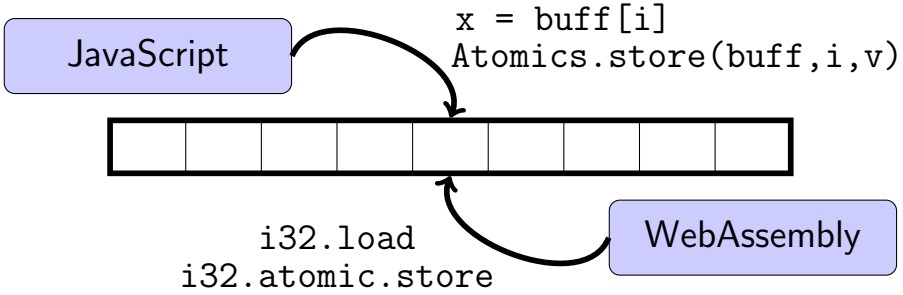
- JavaScript also has threads ("web workers") and shared buffers, even a memory model!

- The WebAssembly memory will be exposed to JavaScript as a shared buffer.



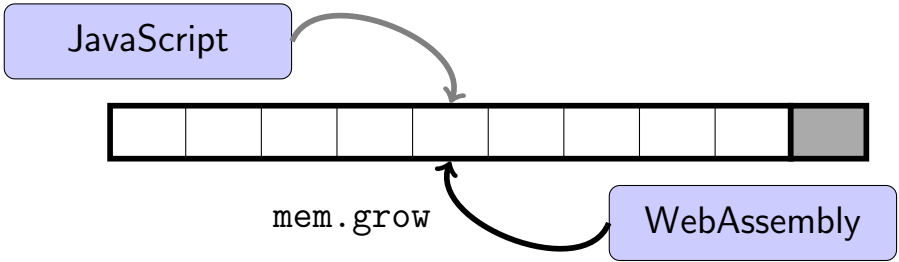
Relaxed memory

- Committee: JS/Wasm interop should "just work".
- So a lot of Wasm consistency behaviour is inherited from JS.



Relaxed memory

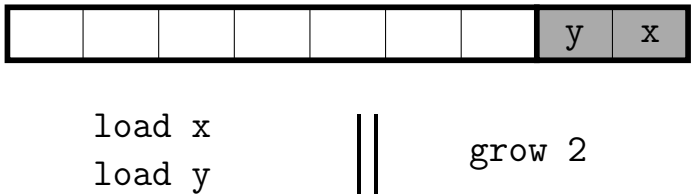
- But Wasm has additional feature - memory growth.
- Now, the size of the memory needs to become part of the axiomatic model.



Relaxed memory

- Implementers don't want to guarantee SC bounds-checking behaviour.

- Updates to memory size can create "data" races.



Relaxed memory

- We said Wasm follows JS.
- What if the JS model is wrong? Ideally, we fix it.
- JS standards body has been *very* welcoming.
- Shu-yu Guo (Bloomberg LP) has been a great point of contact.

Relaxed memory

- Several JS memory model problems discovered.
  - Missing synchronization for wait/wake ops.[4]
  - SC-DRF violation.[5]
  - ARMv8 lda/stl not supported (Stephen Dolan, Cambridge).[6]

[4] 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.

[5] Shu-yu Guo. *Normative: Fix memory model so DRF-SC holds*. Nov. 2018. URL: https://github.com/tc39/ecma262/pull/1362.

[6] Shu-yu Guo. *Memory Model Support for ARMv8 LDA/STL*. Jan. 2019. URL: https://docs.google.com/presentation/d/1qif7z-Y8C-nvJM20UNJQzAKJgLN4wmXS_5NN2Wgipb4/edit?usp=sharing.

Conclusion

- WebAssembly's formal specification hasn't saved it from errors, but at least we can find them more easily.
- Building PL research on top of the WebAssembly semantics works excellently.
- WebAssembly would be widely used even if it was badly designed. It's deserving of research attention!

Thanks for listening!

WEBASSEMBLY