SharedArrayBuffer and Atomics
Stage 2.95 to Stage 3

Shu-yu Guo   Lars Hansen

Mozilla

November 29, 2016

What We Have Consensus On

TC39 agreed on Stage 2.95, July 2016

- Agents
- API (frozen)

What We Have Consensus On

TC39 agreed on Stage 2.95, July 2016

- Agents
- API (frozen)

Memory model had fatal bug

Outline

Memory Model

1. Motivation
2. Intuition
3. What the Model Does

Should We Allow This Optimization?

```
let x = U8[0];
if (x)
  print(x);
```

$\Rightarrow$

```
if (U8[0])
  print(U8[0]);
```

What About This One?

```
while (U8[0] == 42) ;
```

$\Rightarrow$

```
let c = U8[0] == 42;
while (c) ;
```

Or This One?

```
let A = Atomics;                           let A = Atomics;
                                    ⇒      let c = A.load(U8,0) == 42;
while (A.load(U8,0) == 42) ;               while (c) ;
```

What Can Be Printed Here?

`U8[0] = 1;`	`U8[1] = 1;`	`print(U8[0]);` `print(U8[1]);`	`print(U8[1]);` `print(U8[0]);`

What's a Memory Model Good For?

- Arbitrates optimization affordance
- Captures hardware reality

Memory Model Design Space

1. No model
2. Undefined behavior/values for data races
3. **Fully defined; races have meaning**

Why

Because we're the web.

- Interoperability
- Security

Why

Because we're the web.

- Interoperability
- Security
- WebAssembly

What

The model prescribes the set of values that can be read by SAB operations.

Intuition

Strong enough for programmers to reason about programs

Weak enough for hardware and compiler reality

Programmers' Intuition

Sequential Consistency for Data Race Free Programs

Sequential consistency just means interleaving.

Data race freedom means no concurrent, non-atomic memory accesses where one's a write.

Implementors' Intuition: Codegen

Obvious code generation

- Non-atomics compiled to bare stores and loads
- Atomics to atomic instructions or with fences

Implementors' Intuition: Optimizations

- Atomics are carved in stone
- Reads must be stable (e.g. no read rematerialization)
- Writes must be stable (i.e. can't make observable changes to writes)
- Don't completely remove writes (i.e. can coalesce adjacent writes but not remove them completely)

What We Talk About When We Talk About Atomicity

Access atomicity

Indivisible action

What We Talk About When We Talk About Atomicity

Access atomicity

Indivisible action

Copy atomicity

Ordering: what memory accesses become visible to what cores when

What We Talk About When We Talk About Atomicity

The memory model orders shared memory events and prescribes what values can be read by them.

Ordering Analogies: Atomics

- C++ `memory_order_seq_cst`
- LLVM SequentiallyConsistent

Ordering Analogies: Non-Atomics

- Between C++ non-atomics and `memory_order_relaxed`
- Between LLVM non-atomics and Unordered

Details with all the math in the spec.

THIS SLIDE INTENTIONALLY LEFT BLANK

Model Overview

- Axiomatic memory model
- Interfacing with ES evaluation semantics

Axiomatic Model

Ordering is done by an axiomatic model.

Input is a candidate execution—a set of memory events and a set of relations ordering them.

Output is a decision whether the candidate execution is valid.

The meaning of a program is the set of all valid executions.

Axiomatic Model

Ordering is done by an axiomatic model.

Input is a candidate execution—a set of memory events and a set of relations ordering them.

Output is a decision whether the candidate execution is valid.

The meaning of a program is the set of all valid executions.

Not operational!

Events

- Read (atomic and non-atomic)
- Write (atomic and non-atomic)
- ReadModifyWrite (atomic)
- Host-specific events (e.g. `postMessage`)

Candidate Execution

A candidate execution is

- A set of events
- agent-order
- reads-from
- synchronizes-with
- happens-before

agent-order

The union of evaluation orders of all agents.

If E occurred before D in some agent, E is agent-order before D.

reads-from

Maps Read and ReadModifyWrite events to Write and ReadModifyWrite events.

If R $\mathrm{reads\text{-}from}$ W, then R reads one or more bytes written by W.

synchronizes-with

A subset of reads-from that relates synchronizing atomic Read and ReadModifyWrite events to atomic Write and ReadModifyWrite events.

An atomic Read R synchronizes-with an atomic Write W when R reads every byte from W.

happens-before

- agent-order relates intra-agent events
- agent-order relates inter-agent events
- happens-before connects the two

$$(\text{agent-order} \cup \text{synchronizes-with})^{+}$$

Valid Executions

A candidate execution is valid when it has...

- ...coherent reads
- ...tear free reads
- ...sequentially consistent atomics
- ...no out of thin air reads (if we have time)

Coherent Reads

A read of some byte is coherent if it reads the most happens-before recent write to that byte.

$$R \text{ reads-from } W \Rightarrow \nexists W'.W \text{ happens-before } W'$$

Tear Free Reads

Aligned accesses are well-behaved.

Sequentially Consistent Atomics

- All synchronizes-with atomic events exist in a strict total order consistent with happens-before.
- An atomic write becomes visible to atomic reads in finite time.

Data Race Redux

E is in a data race with D iff

- E and D aren't related by $\mathrm{happens\text{-}before}$
- E or D is a Write or ReadModifyWrite event
- E and D aren't synchronized atomics

Event Semantics

- A read event reads a value composed of bytes from write events it reads-from in a valid execution.
- Even racy reads have well-defined values!

Interface with Evaluation Semantics

Where do events come from?

Interface with Evaluation Semantics

Where do events come from?

- Evaluation semantics introduces events

Interface with Evaluation Semantics

Where do events come from?

- Evaluation semantics introduces events
- Value of read events is any possible byte value

Interface with Evaluation Semantics

Without SAB the evaluation semantics constructs a correct execution directly.

With SAB the evaluation semantics constructs many candidate executions nondeterministically and the memory-model decides which ones are valid.

Out of Thin Air

Artifact of axiomatic models
(If we have time)