SharedArrayBuffer and Atomics
Stage 2.95 to Stage 3

Shu-yu Guo Lars Hansen

Mozilla

November 28, 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. The memory model proper

Motivation

Why?

- Need implementation-independent semantics obvs
- No undefined behavior
- Races have meaning

Motivation

Strong enough for programmers to reason about programs

Weak enough for hardware and compiler reality

Programmers' Intuition

SC-DRF

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

Allowed optimizations

- On non-atomics, not as loose as undefined behavior (e.g. no "quantum garbage").
- On atomics, optimizations for sequentially consistent atomics.

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

What memory accesses become visible to what cores when.
Ordering of memory events.

What We Talk About When We Talk About Atomicity

The memory model orders shared memory events and describes what values can be read by the.

Model Overview

- Axiomatic memory model
- Interfacing with ES evaluation semantics

Axiomatic Model

Ordering is done by an axiomatic model.

Input is a candidate execution—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—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 thread, E is $\mathrm{agent\text{-}order}$ before D.

reads-from

Maps Read and ReadModifyWrite events to Write and ReadModifyWrite events.

If R reads-from W, then R reads one or more bytes written by W.

synchronizes-with

A subset of $\mathrm{reads\text{-}from}$ that relates synchronizing atomic Read and ReadModifyWrite events to atomic Write and ReadModifyWrite events.

An atomic Read R $\mathrm{synchronizes\text{-}with}$ an atomic Write W when R reads every byte from W.

happens-before

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

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 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)