Weak memory consistency

Lecture 2

Viktor Vafeiadis
MPI-SWS

2022-11-21

Axiomatic memory models

An alternative way of defining the semantics

Declarative/axiomatic concurrency semantics

- Define the notion of a program *execution*
  (generalization of an execution trace)
- Map a program to a set of executions
- Define a *consistency* predicate on executions
- Semantics = set of consistent executions of a program

Exception: "catch-fire" semantics

- Existence of at least one "bad" consistent execution implies undefined behavior.

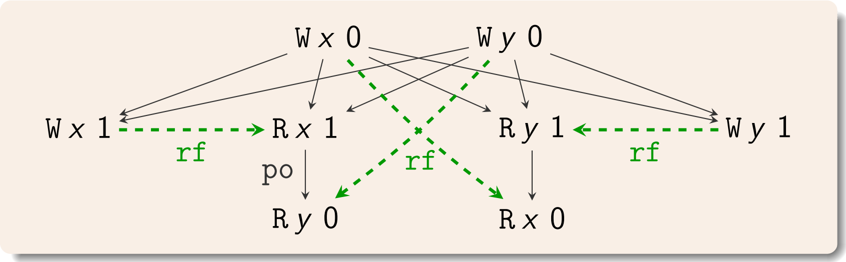
Executions

Events

- Reads, Writes, Updates, Fences

Relations

- Program order, po (also called “sequenced-before”, sb)
- Reads-from, rf



Executions

Definition (Label)

A *label* has one of of the following forms:

$$\mathtt{R}\ x\ \ v_r \qquad \mathtt{W}\ x\ \ v_w \qquad \mathtt{U}\ x\ \ v_r\ \ v_w \qquad \mathtt{F}$$

where $x \in \mathsf{Loc}$ and $v_r, v_w \in \mathsf{Val}$.

Definition (Event)

An *event* is a triple $\langle id, i, l \rangle$ where

- $id \in \mathbb{N}$ is an event identifier,
- $i \in \mathsf{Tid} \cup \{0\}$ is a thread identifier, and
- l is a label.

Executions

Definition (Execution graph)

An *execution graph* is a tuple $\langle E, po, rf \rangle$ where:

- E is a finite set of events
- po ("*program order*") is a partial order on E
- rf ("*reads-from*") is a binary relation on E such that:
  - For every $\langle w, r \rangle \in rf$
    - $\mathtt{typ}(w) \in \{\mathtt{W}, \mathtt{U}\}$
    - $\mathtt{typ}(r) \in \{\mathtt{R}, \mathtt{U}\}$
    - $\mathtt{loc}(w) = \mathtt{loc}(r)$
    - $\mathtt{val_w}(w) = \mathtt{val_r}(r)$
  - rf^{-1} is a function
    (that is: if $\langle w_1, r \rangle, \langle w_2, r \rangle \in rf$ then $w_1 = w_2$)

Some notations

Let $G = \langle E, po, rf \rangle$ be an execution graph.

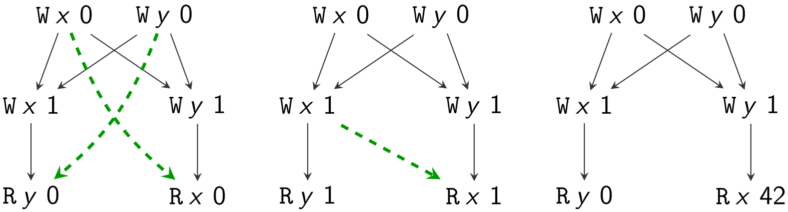
- $G.\mathtt{E} \triangleq E$
- $G.\mathtt{po} \triangleq po$
- $G.\mathtt{rf} \triangleq rf$
- $G.\mathtt{R} \triangleq \{r \in E \mid \mathtt{typ}(r) = \mathtt{R} \vee \mathtt{typ}(r) = \mathtt{U}\}$
- $G.\mathtt{W} \triangleq \{w \in E \mid \mathtt{typ}(w) = \mathtt{W} \vee \mathtt{typ}(w) = \mathtt{U}\}$
- $G.\mathtt{U} \triangleq \{u \in E \mid \mathtt{typ}(u) = \mathtt{U}\}$
- $G.\mathtt{F} \triangleq \{f \in E \mid \mathtt{typ}(f) = \mathtt{F}\}$
- $G.\mathtt{R}_x \triangleq G.\mathtt{R} \cap \{r \in E \mid \mathtt{loc}(r) = x\}$
- ...

Mapping programs to executions: Example

Store buffering (SB)

$$x = y = 0$$

$$\begin{array}{l||l} x := 1 & y := 1 \\ a := y & b := x \end{array}$$



Consistency predicate

Let X be some consistency predicate (on execution graphs)

Definition (Allowed outcome under a declarative model)

An outcome O is *allowed* for a program P under X if there exists an execution graph G such that:

- G is an execution graph of P with outcome O.
- G is X-consistent.

Exception: "catch-fire" semantics

... or if there exists an execution graph G such that:

- G is an execution graph of P.
- G is X-consistent.
- G is "bad".

Completeness

The most basic consistency condition:

Definition (Completeness)

An execution graph G is called *complete* if

$$codom(G.\mathtt{rf}) = G.\mathtt{R}$$

i.e., *every* read reads from *some* write.

Sequential consistency

the result of any execution is the same as if the operations of all the processors were executed in some sequential order, respecting the order specified by the program *[Lamport, 1979]*

Sequential consistency [Lamport]

Definition

Let $\texttt{sc}$ be a total order on $G.\mathtt{E}$. G is called SC-*consistent* wrt $\texttt{sc}$ if the following hold:

- If $\langle a, b \rangle \in G.\mathrm{po}$ then $\langle a, b \rangle \in \texttt{sc}$.
- If $\langle a, b \rangle \in G.\texttt{rf}$ then $\langle a, b \rangle \in \texttt{sc}$ and there does not exist $c \in G.\mathtt{W}_{\mathtt{loc}(b)}$ such that $\langle a, c \rangle \in \texttt{sc}$ and $\langle c, b \rangle \in \texttt{sc}$.

Definition

An execution graph G is called SC-*consistent* if the following hold:

- G is complete.
- G is SC-consistent wrt some total order $\texttt{sc}$ on $G.\mathtt{E}$.

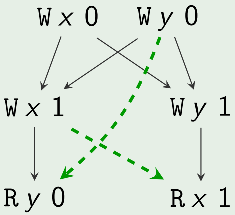
SB example

Store buffering (SB)

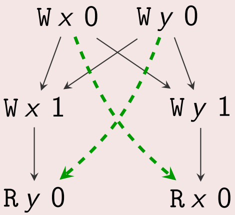
$$x = y = 0$$

$$\begin{array}{l|l} x := 1 & y := 1 \\ a := y & b := x \end{array}$$

Allowed



Forbidden



Sequential consistency (Alternative)

Definition (Modification order (aka coherence order))

$\texttt{mo}$ is called a *modification order* for an execution graph G if $\texttt{mo} = \bigcup_{x \in \mathsf{Loc}} \texttt{mo}_x$ where each $\texttt{mo}_x$ is a total order on $G.\texttt{W}_x$.

Definition (Alternative SC definition)

An execution graph G is called SC-*consistent* if the following hold:

- G is complete
- There exists a modification order $\texttt{mo}$ for G such that $G.\mathrm{po} \cup G.\texttt{rf} \cup \texttt{mo} \cup \texttt{rb}$ is acyclic where:
  - $\texttt{rb} \triangleq G.\texttt{rf}^{-1}; \texttt{mo} \setminus \mathrm{id}$ (from-reads / reads-before)

Equivalence

Theorem

The two SC definitions are equivalent.

Proof (sketch).

Lamport SC $\Rightarrow$ alternative SC:

- Take $\mathtt{mo}_x \triangleq [\mathtt{W}_x]; \mathtt{sc}; [\mathtt{W}_x]$.
- Then, $G.\mathrm{po} \cup G.\mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb} \subseteq \mathtt{sc}$.

Alternative SC $\Rightarrow$ Lamport SC:

- Take $\mathtt{sc}$ to be any total order extending $G.\mathrm{po} \cup G.\mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}$. $\square$

Relaxing sequential consistency

- SC is very expensive to implement in hardware.
- It also forbids various optimizations that are sound for sequential code.

What most hardware guarantee and compilers preserve is "SC-per-location" (aka *coherence*).

Definition

An execution graph G is called *coherent* if the following hold:

- G is complete
- For every location x, there exists a total order $\mathtt{sc}_x$ on all accesses to x such that:
  - If $\langle a, b\rangle \in [\mathtt{RW}_x]; G.\mathtt{po}; [\mathtt{RW}_x]$ then $\langle a, b\rangle \in \mathtt{sc}_x$
  - If $\langle a, b\rangle \in [\mathtt{W}_x]; G.\mathtt{rf}; [\mathtt{R}_x]$ then $\langle a, b\rangle \in \mathtt{sc}_x$ and there does not exist $c \in G.\mathtt{W}_x$ such that $\langle a, c\rangle \in \mathtt{sc}_x$ and $\langle c, b\rangle \in \mathtt{sc}_x$.

Alternative definition of coherence I

SC: $\mathrm{po} \cup \mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic
COH: $\mathrm{po}|_{\mathtt{loc}} \cup \mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic

Definition

Let $\mathtt{mo}$ be a modification order for an execution graph G. G is called *coherent wrt* $\mathtt{mo}$ if $G.\mathrm{po}|_{\mathtt{loc}} \cup G.\mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic (where $\mathtt{rb} \triangleq G.\mathtt{rf}^{-1}; \mathtt{mo} \setminus \mathrm{id}$).

Theorem

An execution graph G is coherent iff the following hold:

- *G is complete*
- *G is coherent wrt some modification order $\mathtt{mo}$ for G.*

“Bad patterns” I

$$\begin{tikzcd} \mathtt{R}_x \arrow[d, "\mathtt{po}"] \\ \mathtt{W}_x \arrow[u, bend left, dashed, "\mathtt{rf}"] \end{tikzcd}$$

no-future-read

$a := x$ // 1
$x := 1$

$$\begin{tikzcd} \mathtt{U}_x \arrow[loop above, dashed, "\mathtt{rf}"] \end{tikzcd}$$

rmw-1

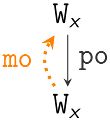
$r :=$ **CAS**$(x, 1, 1)$ // 1

Recall:

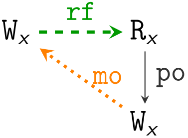
- $\mathtt{W}$ is either a write or an RMW.
- $\mathtt{R}$ is either a read or an RMW.

“Bad patterns” II

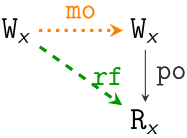
$$
\begin{array}{l||l}
x := 1 & a := x \ \color{red}{/\!/\ 2} \\
x := 2 & a := x \ \color{red}{/\!/\ 1}
\end{array}
$$



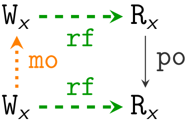
coherence-ww



coherence-rw



coherence-wr



coherence-rr

"Bad patterns" III

$$\begin{tikzcd} \mathtt{W}_x \arrow[r, bend right, dashed, "\mathtt{rf}"'] & \mathtt{U}_x \arrow[l, dotted, "\mathtt{mo}"'] \end{tikzcd}$$

rmw-2

$$\begin{tikzcd} \mathtt{W}_x \arrow[r, dotted, "\mathtt{mo}"] \arrow[rr, bend right, dashed, "\mathtt{rf}"'] & \mathtt{W}_x \arrow[r, dotted, "\mathtt{mo}"] & \mathtt{U}_x \end{tikzcd}$$

atomicity

In coherent executions, an RMW event may only read from its immediate mo-predecessor.

Alternative definition of coherence II

Theorem

Let $\mathtt{mo}$ *be a modification order for an execution graph* G*.*
G *is coherent wrt* $\mathtt{mo}$ *iff the following hold:*

- $\mathtt{rf}; \mathtt{po}$ *is irreflexive.* *(no-future-read)*
- $\mathtt{mo}; \mathtt{po}$ *is irreflexive.* *(coherence-ww)*
- $\mathtt{mo}; \mathtt{rf}; \mathtt{po}$ *is irreflexive.* *(coherence-rw)*
- $\mathtt{rf}^{-1}; \mathtt{mo}; \mathtt{po}$ *is irreflexive.* *(coherence-wr)*
- $\mathtt{rf}^{-1}; \mathtt{mo}; \mathtt{rf}; \mathtt{po}$ *is irreflexive.* *(coherence-rr)*
- $\mathtt{rf}$ *is irreflexive.* *(rmw-1)*
- $\mathtt{mo}; \mathtt{rf}$ *is irreflexive.* *(rmw-2)*
- $\mathtt{rf}^{-1}; \mathtt{mo}; \mathtt{mo}$ *is irreflexive.* *(rmw-atomicity)*

Examples (aka "litmus tests")

Coherence test

$$x = 0$$

$$\begin{array}{l|l} x := 1 & x := 2 \\ a := x \quad /\!/\ 2 & b := x \quad /\!/\ 1 \end{array}$$

Store buffering

$$x = y = 0$$

$$\begin{array}{l|l} x := 1 & y := 1 \\ a := y \quad /\!/\ 0 & b := x \quad /\!/\ 0 \end{array}$$

Atomicity

Parallel increment

$$x = 0$$
$$a := \textbf{FAA}(x, 1) \;\Big\|\; b := \textbf{FAA}(x, 1)$$

Guarantees that $a = 1 \lor b = 1$.

Can we implement locks in this semantics?

Spinlock implementation

$\underline{\textbf{lock}(l) :}$

$r := 0;$
$\textbf{while } \neg r \textbf{ do}$
$\quad r := \textbf{CAS}(l, 0, 1)$

$\underline{\textbf{unlock}(l) :}$

$l := 0$

Implementing locks?

Under COH, the spinlock implementation does not guarantee mutual exclusion.

Spinlock implementation

$\underline{\textbf{lock}(l):}$

$$
\begin{array}{l}
r := 0; \\
\textbf{while } \neg r \textbf{ do} \\
\quad r := \textbf{CAS}(l, 0, 1)
\end{array}
$$

$\underline{\textbf{unlock}(l):}$

$$l := 0$$

Lock example

$$
\begin{array}{l||l}
\textbf{lock}(l) & \textbf{lock}(l) \\
x := 1 & y := 1 \\
a := y \quad /\!/\, 0 & b := x \quad /\!/\, 0 \\
\textbf{unlock}(l) & \textbf{unlock}(l)
\end{array}
$$

Message passing

More generally, COH is often too weak:

$$x = y = 0$$

$$\begin{array}{l} x := 42; \\ y := 1 \end{array} \;\Bigg\|\; \begin{array}{l} a := y; \\ \textbf{while } \neg a \textbf{ do } a := y; \\ b := x \quad /\!/ \mathit{0} \end{array}$$

$$x = y = 0$$

$$\begin{array}{l} x := 42; \\ y := 1 \end{array} \;\Bigg\|\; \begin{array}{l} a := y; \quad /\!/ \mathit{1} \\ b := x \quad /\!/ \mathit{0} \end{array}$$

MP is a common programming idiom.

How can we disallow the weak behavior?

Supporting message passing

$$\begin{tikzcd} \mathtt{W}_x \arrow[r, dotted, "\mathtt{mo}"] \arrow[dr, dashed, "\mathtt{rf}"'] & \mathtt{W}_x \arrow[d, "\mathtt{po}"] \\ & \mathtt{R}_x \end{tikzcd} \quad \rightsquigarrow \quad \begin{tikzcd} \mathtt{W}_x \arrow[r, dotted, "\mathtt{mo}"] \arrow[dr, dashed, "\mathtt{rf}"'] & \mathtt{W}_x \arrow[d, "(\mathtt{po} \cup \mathtt{rf})^+"] \\ & \mathtt{R}_x \end{tikzcd}$$

coherence-wr   coherence-wr

Solution:

- Strengthen the notion of an “observed” write.
- In other words, make rf-edges “synchronizing.”

Release/acquire (RA) memory model

SC: $\mathtt{po} \cup \mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic
COH: $\mathtt{po}|_{\mathtt{loc}} \cup \mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic
RA: $(\mathtt{po} \cup \mathtt{rf})^{+}|_{\mathtt{loc}} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic

Definition

Let $\mathtt{mo}$ be a modification order for an execution graph G.
G is called RA-*consistent wrt* $\mathtt{mo}$ if $(\mathtt{po} \cup \mathtt{rf})^{+}|_{\mathtt{loc}} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic for some modification order $\mathtt{mo}$ for G (where $\mathtt{rb} \triangleq G.\mathtt{rf}^{-1}; \mathtt{mo} \setminus \mathrm{id}$).

Definition

An execution graph G is RA-*consistent* if the following hold:

- G is complete
- G is RA-consistent wrt some modification order $\mathtt{mo}$ for G.

Alternative definition of RA consistency

Theorem

Let $\mathtt{mo}$ be a modification order for an execution graph G. G is RA-consistent wrt $\mathtt{mo}$ iff the following hold:

- $(\mathrm{po} \cup \mathtt{rf})^{+}$ *is irreflexive.* *(no-future-read)*
- $\mathtt{mo}; (\mathrm{po} \cup \mathtt{rf})^{+}$ *is irreflexive.* *(coherence-ww)*
- $\mathtt{rf}^{-1}; \mathtt{mo}; (\mathrm{po} \cup \mathtt{rf})^{+}$ *is irreflexive.* *(coherence-wr)*
- $\mathtt{rf}^{-1}; \mathtt{mo}; \mathtt{mo}$ *is irreflexive.* *(rmw-atomicity)*

The C/C++11 memory model

Mixing the models

$$\mathrm{COH} < \mathrm{RA} < \mathrm{SC}$$

- Revisit the MP idiom:

$$\begin{array}{l} x := 42 \\ y := 1 \end{array} \;\Bigg\|\; \begin{array}{l} a := y \\ \textbf{while } \neg a \textbf{ do } a := y \\ b := x \quad /\!/\, 0 \end{array}$$

- We only need the last read of y to synchronize.
- Idea: introduce *access modes*.

$$\begin{array}{l} x :=_{\mathbf{rlx}} 42 \\ y :=_{\mathbf{rel}} 1 \end{array} \;\Bigg\|\; \begin{array}{l} a := y_{\mathbf{rlx}} \\ \textbf{while } \neg a \textbf{ do } a := y \\ a := y_{\mathbf{acq}} \\ b := x_{\mathbf{rlx}} \quad /\!/\, 0 \end{array}$$

Happens-before

Each memory accesses has a *mode*:

- Reads: **rlx**, **acq**, or **sc**
- Writes: **rlx**, **rel**, or **sc**
- RMWs: **rlx**, **acq**, **rel**, **acq-rel**, or **sc**

“Strength” order $\sqsubset$ is given by:

$$\begin{tikzcd}[row sep=small]
 & \textbf{acq} \arrow[dr] & & \\
\textbf{rlx} \arrow[ur] \arrow[rr] \arrow[dr] & & \textbf{acq-rel} \arrow[r] & \textbf{sc} \\
 & \textbf{rel} \arrow[ur] & &
\end{tikzcd}$$

Synchronization:

$$G.\mathtt{sw} = [\mathtt{W}^{\sqsupseteq \textbf{rel}}];G.\mathtt{rf};[\mathtt{R}^{\sqsupseteq \textbf{acq}}]$$

Happens-before:

$$G.\mathtt{hb} = (G.\mathtt{po} \cup G.\mathtt{sw})^{+}$$

Towards C/C++11 memory model

SC: $\texttt{po} \cup \texttt{rf} \cup \texttt{mo} \cup \texttt{rb}$ is acyclic
COH: $\texttt{po}|_{\texttt{loc}} \cup \texttt{rf} \cup \texttt{mo} \cup \texttt{rb}$ is acyclic
RA: $(\texttt{po} \cup \texttt{rf})^{+}|_{\texttt{loc}} \cup \texttt{mo} \cup \texttt{rb}$ is acyclic
C11: $\texttt{hb}|_{\texttt{loc}} \cup \texttt{rf} \cup \texttt{mo} \cup \texttt{rb}$ is acyclic

Definition

Let $\texttt{mo}$ be a modification order for an execution graph G. G is called C11-*consistent wrt* $\texttt{mo}$ if $\texttt{hb}|_{\texttt{loc}} \cup \texttt{rf} \cup \texttt{mo} \cup \texttt{rb}$ is acyclic (where $\texttt{rb} \triangleq G.\texttt{rf}^{-1}; \texttt{mo} \setminus \mathrm{id}$).

Definition

An execution graph G is C11-consistent if the following hold:

- G is complete
- G is C11-consistent wrt some modification order $\texttt{mo}$ for G.

The C/C++11 memory model

non-atomic $\sqsubset$ relaxed $\sqsubset$ release/acquire $\sqsubset$ sc

The full C/C++11 is more general:

- Non-atomics for non-racy code (the default!)
- Four types of fences for fine grained control
- SC accesses to ensure sequential consistency if needed
- More elaborate definition of `sw` ("release sequences")

C11 model through examples

1

```
        int a = 0;
        int x = 0;
a = 42; || if(x == 1){
x = 1;  ||     print(a);
        || }
```

C11 model through examples

1

```
      int a = 0;
      int x = 0;
a = 42; || if(x == 1){
x = 1;  ||     print(a);
        || }
```

race

C11 model through examples

1

```
      int a = 0;
      int x = 0;
a = 42; || if(x == 1){
x = 1;  ||     print(a);
        || }
```

race

2

```
      int a = 0;
atomic_int x = 0;
a = 42;    || if(x_rlx == 1){
x_rlx = 1; ||     print(a);
           || }
```

1

```
int a = 0;
int x = 0;
a = 42; || if(x == 1){
x = 1;  ||     print(a);
        || }
```

race

2

```
int a = 0;
atomic_int x = 0;
a = 42;     || if(x_rlx == 1){
x_rlx = 1;  ||     print(a);
            || }
```

race

C11 model through examples

1

```
        int a = 0;
        int x = 0;
a = 42; || if(x == 1){
x = 1;  ||     print(a);
        || }
```

race

2

```
        int a = 0;
     atomic_int x = 0;
a = 42;     || if(x_rlx == 1){
x_rlx = 1;  ||     print(a);
            || }
```

race

3

```
        int a = 0;
     atomic_int x = 0;
a = 42;     || if(x_acq == 1){
x_rel = 1;  ||     print(a);
            || }
```

C11 model through examples

1

```
      int a = 0;
      int x = 0;
a = 42; || if(x == 1){
x = 1;  ||     print(a);
        || }
```

race

2

```
      int a = 0;
  atomic_int x = 0;
a = 42;     || if(x_rlx == 1){
x_rlx = 1;  ||     print(a);
            || }
```

race

3

```
      int a = 0;
  atomic_int x = 0;
a = 42;     || if(x_acq == 1){
x_rel = 1;  ||     print(a);
            || }
```

rf

1

```
        int a = 0;
        int x = 0;
a = 42; || if(x == 1){
x = 1;  || race  print(a);
        || }
```

2

```
        int a = 0;
   atomic_int x = 0;
a = 42;     || if(x_rlx == 1){
x_rlx = 1;  || race  print(a);
            || }
```

3

```
        int a = 0;
   atomic_int x = 0;
a = 42;     || if(x_acq == 1){
            rf
x_rel = 1;  || sw  print(a);
            || }
```

C11 model through examples

1

```
      int a = 0;
      int x = 0;
a = 42; || if(x == 1){
x = 1;  ||     print(a);
        || }
```

race

2

```
      int a = 0;
   atomic_int x = 0;
a = 42;     || if(x_rlx == 1){
x_rlx = 1;  ||     print(a);
            || }
```

race

3

```
      int a = 0;
   atomic_int x = 0;
a = 42;     || if(x_acq == 1){
x_rel = 1;  ||     print(a);
            || }
```

rf

sw

4

```
      int a = 0;
   atomic_int x = 0;
a = 42;     || if(x_rlx == 1){
fence_rel;  ||     fence_acq;
x_rlx = 1;  ||     print(a);
            || }
```

C11 model through examples

1

int a = 0;
int x = 0;

a = 42;		if(x == 1){
x = 1;	race	print(a);
		}

2

int a = 0;
atomic_int x = 0;

a = 42;		if(x_{rlx} == 1){
x_{rlx} = 1;	race	print(a);
		}

3

int a = 0;
atomic_int x = 0;

a = 42;		if(x_{acq} == 1){
x_{rel} = 1;	rf, sw	print(a);
		}

4

int a = 0;
atomic_int x = 0;

a = 42;		if(x_{rlx} == 1){
$fence_{rel}$;	rf	$fence_{acq}$;
x_{rlx} = 1;		print(a);
		}

C11 model through examples

1

```
        int a = 0;
        int x = 0;
a = 42;  ||      if(x == 1){
x = 1;   || race     print(a);
         ||      }
```

2

```
        int a = 0;
    atomic_int x = 0;
a = 42;       ||      if(x_rlx == 1){
x_rlx = 1;    || race     print(a);
              ||      }
```

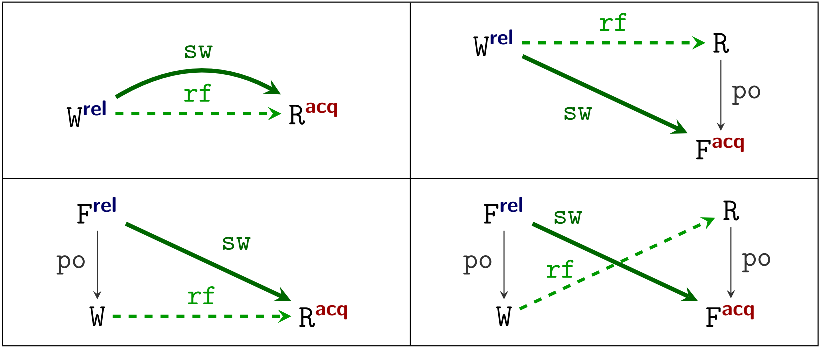
3

```
        int a = 0;
    atomic_int x = 0;
a = 42;       ||    if(x_acq == 1){
              || rf
x_rel = 1;    || sw     print(a);
              ||    }
```

4

```
        int a = 0;
    atomic_int x = 0;
a = 42;        ||    if(x_rlx == 1){
               || rf
fence_rel;     ||      fence_acq;
               || sw
x_rlx = 1;     ||      print(a);
               ||    }
```

The "synchronizes-with" relation



$$\mathtt{sw} \triangleq ([\mathtt{W}^{\sqsupseteq \mathsf{rel}}] \cup [\mathtt{F}^{\sqsupseteq \mathsf{rel}}]; \mathtt{po}); \mathtt{rf}; ([\mathtt{R}^{\sqsupseteq \mathsf{acq}}] \cup \mathtt{po}; [\mathtt{F}^{\sqsupseteq \mathsf{acq}}])$$

Fence modes

$$\begin{tikzcd}
\textbf{acq} \arrow[dr] & & \\
& \textbf{acq-rel} \arrow[r] & \textbf{sc} \\
\textbf{rel} \arrow[ur] & &
\end{tikzcd}$$

Release sequences (RMW's)

$$
\begin{array}{l|l|l}
x_{\mathsf{rlx}} := 42; & & b := y_{\mathsf{acq}}; \quad /\!/ \, 2 \\
 & a := \mathbf{FAI}_{\mathsf{rlx}}(y); \quad /\!/ \, 1 & \\
y_{\mathsf{rel}} := 1 & & c := x_{\mathsf{rlx}}; \quad /\!/ \, 0
\end{array}
$$

$$
\begin{tikzcd}
\mathtt{W}^{\mathsf{rel}} \arrow[r, dashed, "\mathtt{rf}"'] \arrow[rrrr, bend left, "\mathtt{sw}"] & \mathtt{U} \arrow[r, dashed, "\mathtt{rf}"'] & \cdots \arrow[r, dashed, "\mathtt{rf}"'] & \mathtt{U} \arrow[r, dashed, "\mathtt{rf}"'] & \mathtt{R}^{\mathsf{acq}}
\end{tikzcd}
$$

$$
\mathtt{sw} \triangleq ([\mathtt{W}^{\sqsupseteq\mathsf{rel}}] \cup [\mathtt{F}^{\sqsupseteq\mathsf{rel}}]; \mathtt{po}); \mathtt{rf}^{+}; ([\mathtt{R}^{\sqsupseteq\mathsf{acq}}] \cup \mathtt{po}; [\mathtt{F}^{\sqsupseteq\mathsf{acq}}])
$$

“Catch-fire” semantics

Definition (Race in C11)

Given a C11-execution graph G, we say that two events a, b C11-*race* in G if the following hold:

- $a \neq b$
- $\texttt{loc}(a) = \texttt{loc}(b)$
- $\{\texttt{typ}(a), \texttt{typ}(b)\} \cap \{\texttt{W}, \texttt{U}\} \neq \emptyset$
- $\textbf{na} \in \{\texttt{mod}(a), \texttt{mod}(b)\}$
- $\langle a, b\rangle \notin \texttt{hb}$ and $\langle b, a\rangle \notin \texttt{hb}$

G is called C11-*racy* if some a, b C11-race in G.

C11 consistency

Definition

Let $\texttt{mo}$ be a modification order for an execution graph G.
G is called C11-*consistent wrt* $\texttt{mo}$ if:

- $\texttt{hb}|_{\texttt{loc}} \cup \texttt{rf} \cup \texttt{mo} \cup \texttt{rb}$ is acyclic (where $\texttt{rb} \triangleq G.\texttt{rf}^{-1}; \texttt{mo} \setminus \mathrm{id}$).
- ...**sc**... ?

Definition

An execution graph G is C11-consistent if the following hold:

- G is complete
- G is C11-consistent wrt some modification order $\texttt{mo}$ for G.

- The most involved part of the model, due to the possible mixing of different access modes to the same location.
- Changed in C++20
- *If there is no mixing of SC and non-SC accesses*, then additionally require acyclicity of $\mathtt{hb} \cup \mathtt{mo}_{\mathbf{sc}} \cup \mathtt{rb}_{\mathbf{sc}}$.

Further reading:

- Overhauling SC atomics in C11 and OpenCL. Mark Batty, Alastair F. Donaldson, John Wickerson, POPL 2016.
- Repairing sequential consistency in C/C++11. Ori Lahav, Viktor Vafeiadis, Jeehoon Kang, Chung-Kil Hur, Derek Dreyer, PLDI 2017.

Repaired SC condition for fences

$$\mathtt{eco} \triangleq (\mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb})^+ \qquad \text{(extended coherence order)}$$

$$\mathtt{psc_F} \triangleq [\mathtt{F}^{\mathsf{sc}}]; (\mathtt{hb} \cup \mathtt{hb}; \mathtt{eco}; \mathtt{hb}); [\mathtt{F}^{\mathsf{sc}}] \qquad \text{(SC fence order)}$$

Condition on SC fences

$\mathtt{psc_F}$ is acyclic

Example: SB with fences

$$x = y = 0$$

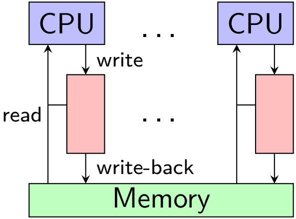
$x_{\mathbf{rlx}} := 1;$	$y_{\mathbf{rlx}} := 1;$
fence(**sc**);	**fence**(**sc**);
$a := y_{\mathbf{rlx}};$ // 0	$b := x_{\mathbf{rlx}};$ // 0

✗ behavior disallowed

Reduction from RA to SC

Reduction to SC (robustness)

For TSO, it suffices to have a fence between every racy write & subsequent racy read.



For RA, we need more fences. Recall the IRIW example:

Independent reads of independent writes (IRIW)

$$x = y = 0$$

$$x := 1 \;\left\|\; \begin{array}{ll} a := x; & /\!/ 1 \\ b := y & /\!/ 0 \end{array} \;\right\|\; \begin{array}{ll} c := y; & /\!/ 1 \\ d := x & /\!/ 0 \end{array} \;\left\|\; y := 1\right.$$

What is the semantics of SC fences?

From C11, we had:

$$\mathtt{eco} \triangleq (\mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb})^{+} \qquad \text{(extended coherence order)}$$

$$\mathtt{psc_F} \triangleq [\mathrm{F}^{\mathsf{sc}}]; (\mathtt{hb} \cup \mathtt{hb}; \mathtt{eco}; \mathtt{hb}); [\mathrm{F}^{\mathsf{sc}}]$$

(partial SC fence order)

and required that $\mathtt{psc_F}$ is acyclic.

That is,

Definition (RA consistency with fences)

An execution graph G is RA-*consistent* iff there exists some modification order $\mathtt{mo}$ for G such that:

- G is complete,
- $(\mathrm{po} \cup \mathtt{rf})^{+}|_{\mathtt{loc}} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic, and
- $\mathtt{psc_F}$ is acyclic.

Alternative definition of RA consistency

Theorem

An execution graph G is RA*-consistent iff there exists a total order* $\mathtt{sc}$ *on* $G.\mathrm{F}^{\mathit{sc}}$ *and a modification order* $\mathtt{mo}$ *for* G *such that:*

- *G is complete,*
- $(\mathrm{po} \cup \mathtt{rf} \cup \mathtt{sc})^{+}$ *is irreflexive, and*
- $(\mathrm{po} \cup \mathtt{rf} \cup \mathtt{sc})^{*}; \mathtt{eco}$ *is irreflexive.*

Simple reduction theorem

Theorem

Let G be an RA*-consistent execution graph. If*

- *For every G-racy events a, b, if $\langle a, b\rangle \in (G.\mathtt{po} \cup G.\mathtt{rf})^+$, then $\langle a, c\rangle, \langle c, b\rangle \in (G.\mathtt{po} \cup G.\mathtt{rf})^+$ for some fence event c.*

Then, G is SC-consistent.

Proof of the simple reduction theorem (1/2)

Recall:

- Recall SC-consistency : $\mathtt{po} \cup \mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic.
- Let $\mathtt{hb} \triangleq (\mathtt{po} \cup \mathtt{rf} \cup \mathtt{sc})^+$ and $K \triangleq (\mathtt{mo} \cup \mathtt{rb}) \setminus \mathtt{hb}$.
- It suffices to prove : $\mathtt{hb} \cup K$ is acyclic.

Consider minimal cycle in $(\mathtt{hb} \cup K)$.

- Cycles with ≤ 1 K-edges disallowed by RA consistency.
- Cycle with two K-edges:

$$\begin{tikzcd}
\mathrm{a} \arrow[r, "K"] & \mathrm{b} \arrow[d, "\mathtt{hb}"] \\
\mathrm{d} \arrow[u, "\mathtt{hb}"] & \mathrm{c} \arrow[l, "K"]
\end{tikzcd}$$

Proof of the simple reduction theorem (1/2)

Recall:

- Recall SC-consistency : $\mathtt{po} \cup \mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic.
- Let $\mathtt{hb} \triangleq (\mathtt{po} \cup \mathtt{rf} \cup \mathtt{sc})^+$ and $K \triangleq (\mathtt{mo} \cup \mathtt{rb}) \setminus \mathtt{hb}$.
- It suffices to prove : $\mathtt{hb} \cup K$ is acyclic.

Consider minimal cycle in $(\mathtt{hb} \cup K)$.

- Cycles with ≤ 1 K-edges disallowed by RA consistency.
- Cycle with two K-edges:

$$\begin{tikzcd}
\mathsf{a} \arrow[rrr, "K"] & & & \mathsf{b} \arrow[dl] \arrow[dd, "\mathtt{hb}"] \\
& f_1 \arrow[ul] & f_2 \arrow[dr] & \\
\mathsf{d} \arrow[uu, "\mathtt{hb}"] \arrow[ur] & & & \mathsf{c} \arrow[lll, "K"]
\end{tikzcd}$$

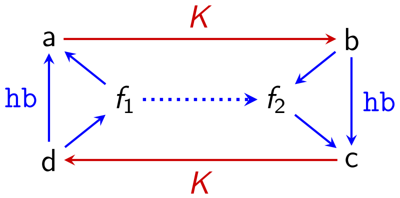
Proof of the simple reduction theorem (1/2)

Recall:

- Recall SC-consistency : $\mathtt{po} \cup \mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}$ is acyclic.
- Let $\mathtt{hb} \triangleq (\mathtt{po} \cup \mathtt{rf} \cup \mathtt{sc})^+$ and $K \triangleq (\mathtt{mo} \cup \mathtt{rb}) \setminus \mathtt{hb}$.
- It suffices to prove : $\mathtt{hb} \cup K$ is acyclic.

Consider minimal cycle in $(\mathtt{hb} \cup K)$.

- Cycles with ≤ 1 K-edges disallowed by RA consistency.
- Cycle with two K-edges:



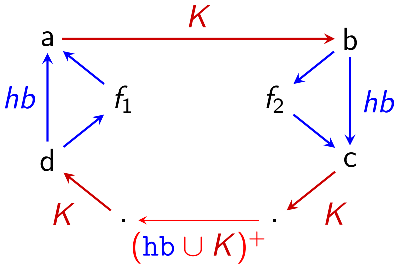
Proof of the simple reduction theorem (2/2)

Finally, consider a cycle with three or more K-edges.

$$\begin{tikzcd}
a \arrow[rrr, "K"] & & & b \arrow[d, "hb"] \\
d \arrow[u, "hb"] & & & c \arrow[dl, "K"] \\
& \cdot \arrow[ul, "K"] & \cdot \arrow[l, "(\mathtt{hb} \cup K)^+"'] &
\end{tikzcd}$$

Proof of the simple reduction theorem (2/2)

Finally, consider a cycle with three or more K-edges.



Proof of the simple reduction theorem (2/2)

Finally, consider a cycle with three or more K-edges.