

'Easy' fixes to the C11 memory model

Viktor Vafeiadis

Max Planck Institute for Software Systems (MPI-SWS)

25 September 2014

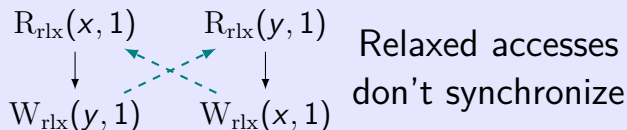
Dependency cycles

Initially $x = y = 0$.

```
if (x.load(rlx) == 1) || if (y.load(rlx) == 1)
    y.store(1, rlx);      x.store(1, rlx);
```

C11 allows the outcome $x = y = 1$.

Justification:



No easy fix?

Given a memory model definition

1. Check that the model is *mathematically sane*.
 - ▶ For example, it is monotone.
2. Check that it is *not too weak*.
 - ▶ Provides useful reasoning principles.
3. Check that it is *not too strong*.
 - ▶ Can be implemented efficiently.
4. Check that it is *actually useful*.
 - ▶ Admits the intended program optimisations.

How does the C11 definition rate?

1. Check that the model is *mathematically sane*.

✗ No, it is not monotone.

2. Check that it is *not too weak*.

✗ No, due to dependency cycles.

3. Check that the model is *not too strong*.

✓ Yes, prior work.

4. Check that it is *actually useful*.

✗ No, it disallows intended program transformations.

“Adding synchronisation should not introduce new behaviours”

Examples:

- ▶ Adding a memory fence
- ▶ Strengthening the access mode of an operation
- ▶ Reducing parallelism, $C_1 \parallel C_2 \rightsquigarrow C_1 ; C_2$
- ▶ Expression evaluation linearisation:

$$x = a + b; \rightsquigarrow t_1 = a; t_2 = b; x = t_1 + t_2;$$

- ▶ (Roach motel reorderings)

1. The axiom for non-atomic reads

$$\text{rf}(b) = a \wedge (\text{isNA}(a) \vee \text{isNA}(b)) \implies \text{hb}(a, b)$$

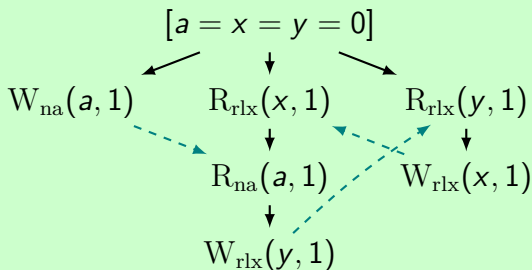
(in combination with dependency cycles)

2. The axiom for SC reads

3. Lack of intra-thread synchronisation
(in the presence of consumes)

Sequentialisation is invalid

$a = 1; \parallel \text{if } (x.\text{load}(r/x) == 1) \parallel \text{if } (y.\text{load}(r/x) == 1)$
 $\parallel \text{if } (a == 1) \parallel x.\text{store}(1, r/x);$
 $y.\text{store}(1, r/x);$



$$rf(b) = a \wedge (\text{isNA}(a) \vee \text{isNA}(b)) \implies hb(a, b)$$

- ▶ Let

$$\text{rf}_{\text{NA}}(a, b) \stackrel{\text{def}}{=} \text{rf}(b) = a \wedge (\text{isNA}(a) \vee \text{isNA}(b))$$

- ▶ Drop the $\text{rf}_{\text{NA}} \subseteq \text{hb}$ axiom.
- ▶ Strengthen the acyclicity axiom:

$$\text{acyclic}(\text{hb} \cup \text{rf}_{\text{NA}})$$

SC read restriction

There shall be a single total order S on all `seq_cst` operations [...] such that each `seq_cst` operation B that loads a value from an atomic object M observes one of the following values:

- ▶ the result of the last modification A of M that precedes B in S , if it exists, or
- ▶ if A exists, the result of some modification of M in the visible sequence of side effects with respect to B that is not `seq_cst` and that does not happen before A , or
- ▶ if A does not exist, [...]

[N1570, §7.17.3.6]

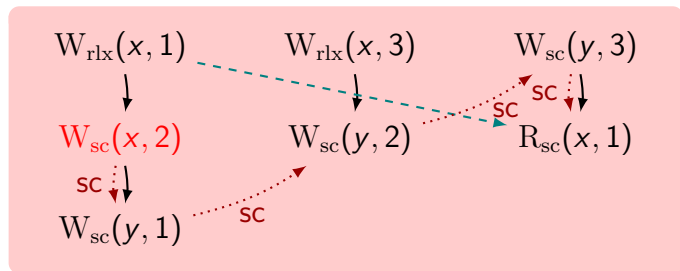
$$\text{rf}(b) = c \wedge \text{isSC}(b) \implies \text{isrc}(c, b) \vee \neg \text{isSC}(c) \wedge \nexists a. \text{hb}(c, a) \wedge \text{isrc}(a, b)$$

$$\begin{aligned} \text{where } \text{isrc}(c, b) &\stackrel{\text{def}}{=} \text{scr}(c, b) \wedge \nexists d. \text{scr}(c, d) \wedge \text{scr}(d, b) \\ \text{scr}(c, b) &\stackrel{\text{def}}{=} \text{iswrite}_{\text{locs}(b)}(c) \wedge \text{sc}(c, b) \end{aligned}$$

Strengthening is invalid

$x.\text{store}(1, r/x);$ $x.\text{store}(2, sc);$ $y.\text{store}(1, sc);$	$x.\text{store}(3, r/x);$ $y.\text{store}(2, sc);$	$y.\text{store}(3, sc);$ $r = x.\text{load}(sc);$	$s_1 = x.\text{load}(r/x);$ $s_2 = x.\text{load}(r/x);$ $s_3 = x.\text{load}(r/x);$ $t_1 = y.\text{load}(r/x);$ $t_2 = y.\text{load}(r/x);$ $t_3 = y.\text{load}(r/x);$
---	---	--	--

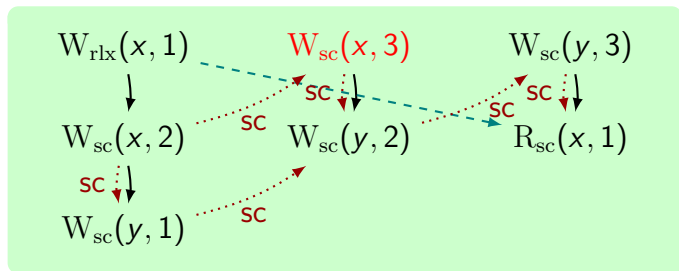
$r = s_1 = t_1 = 1 \wedge s_2 = t_2 = 2 \wedge s_3 = t_3 = 3$ — Disallowed



Strengthening is invalid

$\left\ \begin{array}{l} x.\text{store}(1, r/x); \\ x.\text{store}(2, sc); \\ y.\text{store}(1, sc); \end{array} \right\ $	$\left\ \begin{array}{l} x.\text{store}(3, \textcolor{red}{sc}); \\ y.\text{store}(2, sc); \end{array} \right\ $	$\left\ \begin{array}{l} y.\text{store}(3, sc); \\ r = x.\text{load}(sc); \end{array} \right\ $	$\left\ \begin{array}{l} s_1 = x.\text{load}(r/x); \\ s_2 = x.\text{load}(r/x); \\ s_3 = x.\text{load}(r/x); \\ t_1 = y.\text{load}(r/x); \\ t_2 = y.\text{load}(r/x); \\ t_3 = y.\text{load}(r/x); \end{array} \right\ $
---	---	--	--

$r = s_1 = t_1 = 1 \wedge s_2 = t_2 = 2 \wedge s_3 = t_3 = 3$ — Allowed



No intra-thread synchronisation

Initially $x = y = 0$.

$a = 1;$ $\parallel$ **if** ($x.load(cons)$) $\parallel$ **if** ($y.load(acq)$)
 $x.store(1, rel);$ $y.store(1, rel);$ $a = 2;$

↓ sequentialise threads 2 & 3

$a = 1;$ $\parallel$ **if** ($x.load(cons)$) $y.store(1, rel);$
 $x.store(1, rel);$ $\parallel$ **if** ($y.load(acq)$) $a = 2;$

- ▶ Only different thread rel-acq synchronize.
- ▶ *Proposed fix*: drop this restriction

“Removing $(hb \cup rf)$ -maximal events should preserve consistency”

- ▶ Maximal events should not affect other events
- ▶ Does not hold because of release sequences

Release sequences are too strong

Initially $x = y = 0$.

```
a = 1;
x.store(1, rel);
x.store(3, rlx);
```

 $\parallel$

```
while (x.load(acq)  $\neq$  3);
a = 2;
```

 $\parallel$

This program is not racy.

The acquire synchronizes with the release.

$$rsElem(a, b) \stackrel{\text{def}}{=} sameThread(a, b) \vee isrmw(b)$$

$$rseq(a, b) \stackrel{\text{def}}{=} a = b \vee$$

$$rsElem(a, b) \wedge mo(a, b) \wedge$$

$$(\forall c. mo(a, c) \wedge mo(c, b) \Rightarrow rsElem(a, c))$$

Release sequences are too strong

Initially $x = y = 0$.

$a = 1;$ $x.\text{store}(1, \text{rel});$ $x.\text{store}(3, \text{rlx});$	$\parallel$	while ($x.\text{load}(\text{acq}) \neq 3$); $a = 2;$	$\parallel$	$x.\text{store}(2, \text{rlx});$
--	-------------	--	-------------	----------------------------------

But this one is racy according to C11.

The acquire no longer synchronizes with the release.

$$rsElem(a, b) \stackrel{\text{def}}{=} sameThread(a, b) \vee isrmw(b)$$
$$rseq(a, b) \stackrel{\text{def}}{=} a = b \vee$$
$$rsElem(a, b) \wedge mo(a, b) \wedge$$
$$(\forall c. mo(a, c) \wedge mo(c, b) \Rightarrow rsElem(a, c))$$

Proposed fix for release sequences

$$\begin{aligned} rseq_{new}(a, b) \stackrel{\text{def}}{=} & a = b \\ & \vee \text{sameThread}(a, b) \wedge mo(a, b) \\ & \vee isrmw(b) \wedge rseq_{new}(a, rf(b)) \end{aligned}$$