‘Easy’ fixes to the C11 memory model

Viktor Vafeiadis

Max Planck Institute for Software Systems (MPI-SWS)

25 September 2014

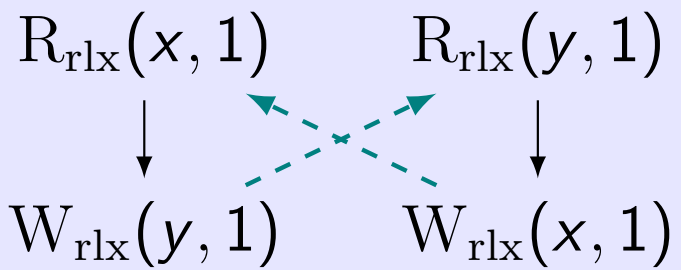
Dependency cycles

Initially $x = y = 0$.

$$\textbf{if } (x.load(rlx) == 1) \quad \| \quad \textbf{if } (y.load(rlx) == 1)$$
$$\quad y.store(1, rlx); \quad \| \quad \quad x.store(1, rlx);$$

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

Justification:



Relaxed accesses don't synchronize

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.

Monotonicity

"Adding synchronisation should not introduce new behaviours"

Examples:

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

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

- (Roach motel reorderings)

Obstacles to monotonicity

1. The axiom for non-atomic reads

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

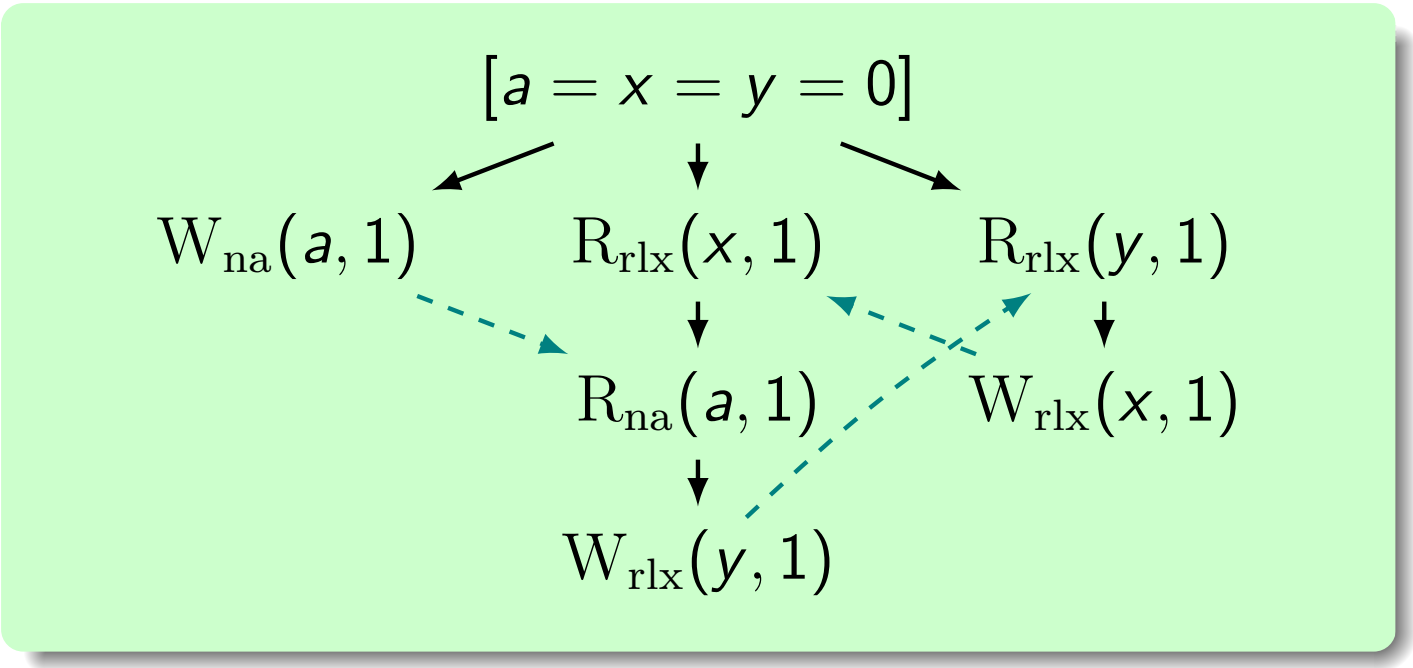
   (in combination with dependency cycles)

2. The axiom for SC reads

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

Sequentialisation is invalid

$$a = 1; \;\Big\|\; \textbf{if}\ (x.\mathsf{load}(\mathit{rlx}) == 1)\ \textbf{if}\ (a == 1)\ y.\mathsf{store}(1, \mathit{rlx}); \;\Big\|\; \textbf{if}\ (y.\mathsf{load}(\mathit{rlx}) == 1)\ x.\mathsf{store}(1, \mathit{rlx});$$



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

Proposed fix

- Let

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

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

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

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]

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

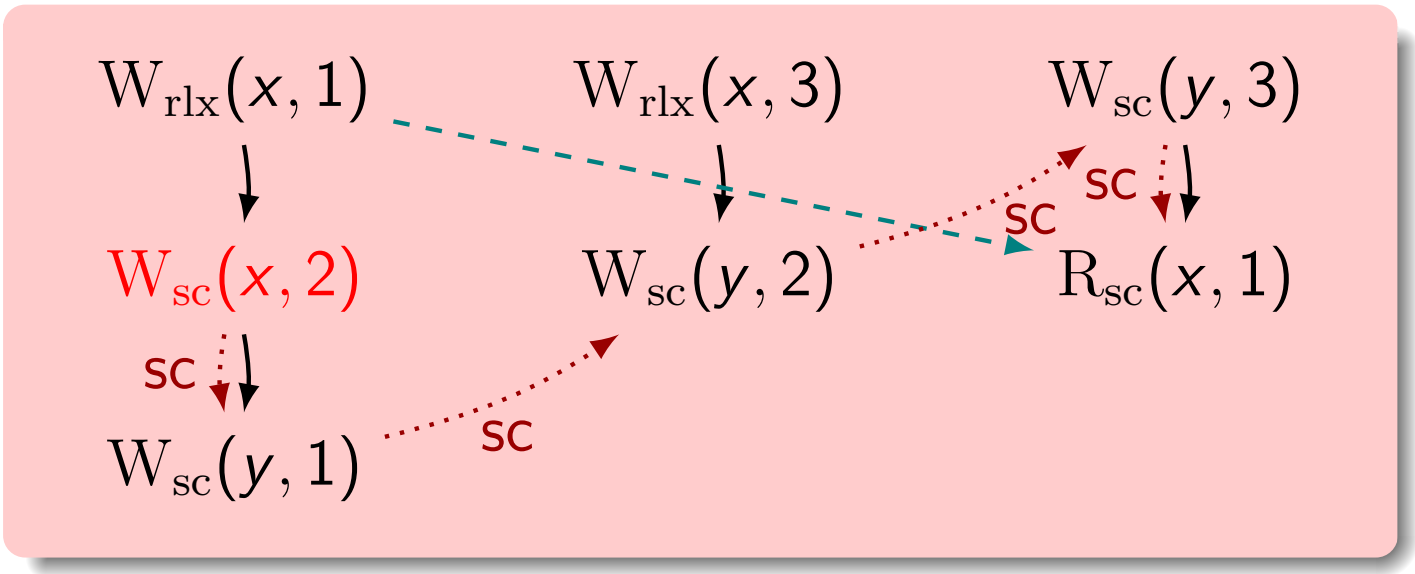
where $\mathsf{iscr}(c, b) \stackrel{\text{def}}{=} \mathsf{scr}(c, b) \wedge \nexists d.\ \mathsf{scr}(c, d) \wedge \mathsf{scr}(d, b)$

$$\mathsf{scr}(c, b) \stackrel{\text{def}}{=} \mathsf{iswrite}_{\mathsf{locs}(b)}(c) \wedge \mathsf{sc}(c, b)$$

Strengthening is invalid

$$
\begin{array}{l}
x.\mathsf{store}(1, rlx); \\
x.\mathsf{store}(2, sc); \\
y.\mathsf{store}(1, sc);
\end{array}
\;\Big\|\;
\begin{array}{l}
x.\mathsf{store}(3, rlx); \\
y.\mathsf{store}(2, sc);
\end{array}
\;\Big\|\;
\begin{array}{l}
y.\mathsf{store}(3, sc); \\
r = x.\mathsf{load}(sc);
\end{array}
\;\Big\|\;
\begin{array}{l}
s_1 = x.\mathsf{load}(rlx); \\
s_2 = x.\mathsf{load}(rlx); \\
s_3 = x.\mathsf{load}(rlx); \\
t_1 = y.\mathsf{load}(rlx); \\
t_2 = y.\mathsf{load}(rlx); \\
t_3 = y.\mathsf{load}(rlx);
\end{array}
$$

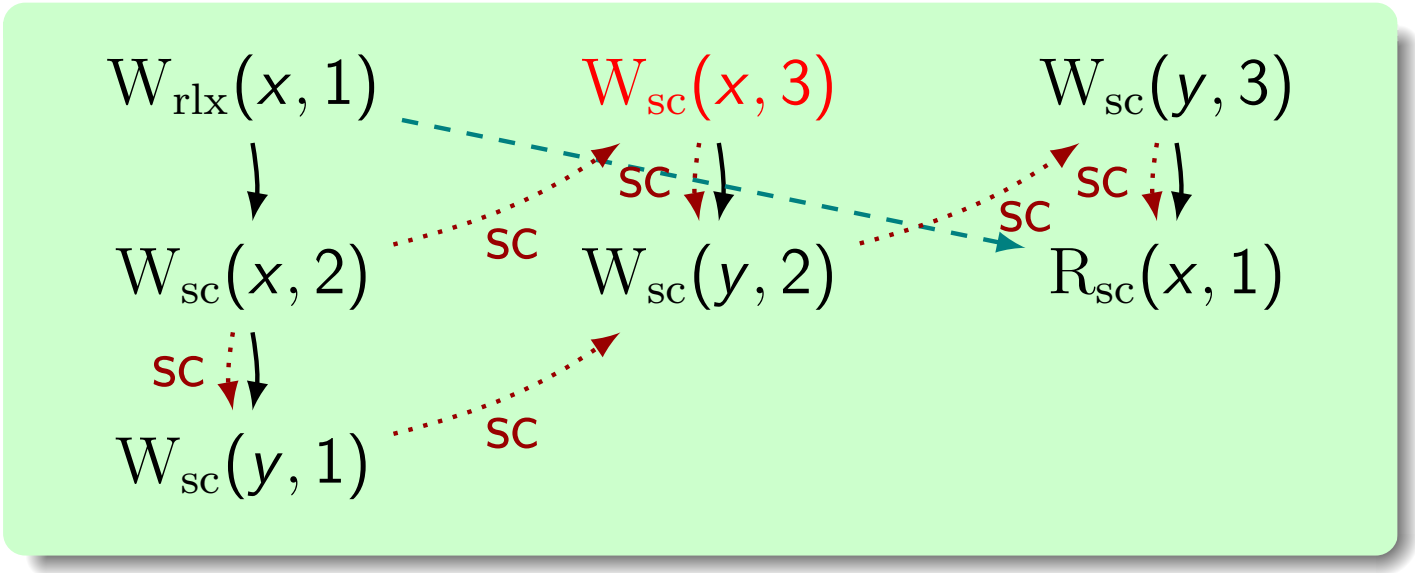
$r = s_1 = t_1 = 1 \wedge s_2 = t_2 = 2 \wedge s_3 = t_3 = 3$ — Disallowed



Strengthening is invalid

$$
\begin{array}{l||l||l||l}
x.\mathsf{store}(1, \mathit{rlx}); & x.\mathsf{store}(3, \mathit{sc}); & y.\mathsf{store}(3, \mathit{sc}); & s_1 = x.\mathsf{load}(\mathit{rlx}); \\
x.\mathsf{store}(2, \mathit{sc}); & y.\mathsf{store}(2, \mathit{sc}); & r = x.\mathsf{load}(\mathit{sc}); & s_2 = x.\mathsf{load}(\mathit{rlx}); \\
y.\mathsf{store}(1, \mathit{sc}); & & & s_3 = x.\mathsf{load}(\mathit{rlx}); \\
 & & & t_1 = y.\mathsf{load}(\mathit{rlx}); \\
 & & & t_2 = y.\mathsf{load}(\mathit{rlx}); \\
 & & & t_3 = y.\mathsf{load}(\mathit{rlx});
\end{array}
$$

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



No intra-thread synchronisation

Initially $x = y = 0$.

$$\begin{array}{l|l|l} a = 1; & \textbf{if } (x.load(cons)) & \textbf{if } (y.load(acq)) \\ x.store(1, rel); & \quad y.store(1, rel); & \quad a = 2; \end{array}$$

$$\downarrow \text{ sequentialise threads 2 \& 3}$$

$$\begin{array}{l|l} a = 1; & \textbf{if } (x.load(cons))\ y.store(1, rel); \\ x.store(1, rel); & \textbf{if } (y.load(acq))\ a = 2; \end{array}$$

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

Prefix closure

"Removing $(\mathsf{hb} \cup \mathsf{rf})$-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 $x = y = 0$.

$$\begin{array}{l} a = 1; \\ x.\text{store}(1, \mathit{rel}); \\ x.\text{store}(3, \mathit{rlx}); \end{array} \;\Big\|\; \begin{array}{l} \textbf{while}\ (x.\text{load}(\mathit{acq}) \neq 3); \\ a = 2; \end{array}$$

This program is not racy.

The acquire synchronizes with the release.

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

Release sequences are too strong

Initially $x = y = 0$.

$$
\begin{array}{l}
a = 1; \\
x.\text{store}(1, \mathit{rel}); \\
x.\text{store}(3, \mathit{rlx});
\end{array}
\;\Big\|\;
\begin{array}{l}
\textbf{while}\ (x.\text{load}(\mathit{acq}) \neq 3); \\
a = 2;
\end{array}
\;\Big\|\;
x.\text{store}(2, \mathit{rlx});
$$

But this one is racy according to C11.

The acquire no longer synchronizes with the release.

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

Proposed fix for release sequences

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