‘Easy’ fixes to the C11 memory model

Viktor Vafeiadis
Max Planck Institute for Software Systems (MPI-SWS)

25 September 2014
Dependency cycles

Initially \( x = y = 0 \).

\[
\begin{align*}
\textbf{if} \ (x.\text{load}(rlx) == 1) & \quad \textbf{if} \ (y.\text{load}(rlx) == 1) \\
\quad y.\text{store}(1, rlx); & \quad x.\text{store}(1, rlx);
\end{align*}
\]

C11 allows the outcome \( x = y = 1 \).

**Justification:**

\[
\begin{array}{c}
R_{rlx}(x, 1) \\
\downarrow \quad \leftarrow \quad \downarrow \\
W_{rlx}(y, 1) & \quad W_{rlx}(x, 1)
\end{array}
\]

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.

Viktor Vafeiadis  
‘Easy’ fixes to the C11 memory model
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, $C_1 || C_2 \leadsto C_1 ; C_2$
- Expression evaluation linearisation:
  \[
  x = a + b ; \quad \leadsto \quad t_1 = a ; t_2 = b ; x = t_1 + t_2 ;
  \]
- (Roach motel reorderings)
Obstacles to monotonicity

1. The axiom for non-atomic reads

\[ rf(b) = a \land (\text{isNA}(a) \lor \text{isNA}(b)) \implies \text{hb}(a, b) \]

(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

\[ a = 1; \quad \begin{align*}
&\text{if } (x.\text{load}(rlx) == 1) \\
&\quad \text{if } (a == 1) \\
&\quad \quad y.\text{store}(1, rlx); \\
&\text{if } (y.\text{load}(rlx) == 1) \\
&\quad x.\text{store}(1, rlx); \\
\end{align*} \]

\[ [a = x = y = 0] \]

\[ W_{\text{na}}(a, 1) \quad \quad R_{\text{rlx}}(x, 1) \quad \quad R_{\text{rlx}}(y, 1) \]

\[ R_{\text{na}}(a, 1) \quad \quad W_{\text{rlx}}(x, 1) \quad \quad R_{\text{rlx}}(y, 1) \]

\[ W_{\text{rlx}}(y, 1) \]

\[ \text{rf}(b) = a \land (\text{isNA}(a) \lor \text{isNA}(b)) \implies \text{hb}(a, b) \]
Proposed fix

- Let

\[ rf_{NA}(a, b) \overset{\text{def}}{=} rf(b) = a \land (\text{isNA}(a) \lor \text{isNA}(b)) \]

- Drop the \( rf_{NA} \subseteq hb \) axiom.

- Strengthen the acyclicity axiom:

\[ \text{acyclic}(hb \cup rf_{NA}) \]
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, [...]

\[ rf(b) = c \land \text{isSC}(b) \implies \]
\[ \text{iscr}(c, b) \lor \neg \text{isSC}(c) \land \exists a. \: \text{hb}(c, a) \land \text{iscr}(a, b) \]

where $\text{iscr}(c, b) \overset{\text{def}}{=} \text{scr}(c, b) \land \exists d. \: \text{scr}(c, d) \land \text{scr}(d, b)$

\[ \text{scr}(c, b) \overset{\text{def}}{=} \text{iswrite}_{\text{locs}}(b)(c) \land \text{sc}(c, b) \]
Strengthening is invalid

\[
x.\text{store}(1, \text{rlx}); \quad x.\text{store}(3, \text{rlx}); \quad y.\text{store}(3, \text{sc}); \\
x.\text{store}(2, \text{sc}); \quad y.\text{store}(2, \text{sc}); \quad y.\text{store}(3, \text{sc}); \\
y.\text{store}(1, \text{sc}); \quad r = x.\text{load}(\text{sc}); \\
r = s_1 = t_1 = 1 \land s_2 = t_2 = 2 \land s_3 = t_3 = 3 \quad \text{— Disallowed}
\]
Strengthening is invalid

\[
\begin{align*}
    x.\text{store}(1, rlx); \\
    x.\text{store}(2, sc); \\
    y.\text{store}(1, sc);
\end{align*}
\]

\[
\begin{align*}
    x.\text{store}(3, sc); \\
    y.\text{store}(2, sc); \\
    y.\text{store}(1, sc);
\end{align*}
\]

\[
\begin{align*}
    s_1 &= x.\text{load}(rlx); \\
    s_2 &= x.\text{load}(rlx); \\
    s_3 &= x.\text{load}(rlx);
\end{align*}
\]

\[
\begin{align*}
    r &= x.\text{load}(sc); \\
    t_1 &= y.\text{load}(rlx); \\
    t_2 &= y.\text{load}(rlx); \\
    t_3 &= y.\text{load}(rlx);
\end{align*}
\]

\[
\begin{align*}
    r = s_1 = t_1 = 1 & \land s_2 = t_2 = 2 & \land s_3 = t_3 = 3 & \text{ — Allowed}
\end{align*}
\]
No intra-thread synchronisation

Initially $x = y = 0$.

\[
\begin{align*}
a &= 1; & \text{if} & (x.\text{load}(\text{cons})) & \text{if} & (y.\text{load}(\text{acq})) \\
x.\text{store}(1, \text{rel}); & y.\text{store}(1, \text{rel}); & a &= 2;
\end{align*}
\]

\[
\downarrow \text{sequentialise threads 2 & 3}
\]

\[
\begin{align*}
a &= 1; & \text{if} & (x.\text{load}(\text{cons})) & y.\text{store}(1, \text{rel}); \\
x.\text{store}(1, \text{rel}); & \text{if} & (y.\text{load}(\text{acq})) & a &= 2;
\end{align*}
\]

- 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 \( x = y = 0 \).

\[
\begin{align*}
a &= 1; \\
x.\text{store}(1, \text{rel}); & \quad \text{while} \ (x.\text{load}(\text{acq}) \neq 3); \\
x.\text{store}(3, \text{rlx}); & \quad a = 2;
\end{align*}
\]

This program is not racy.
The acquire synchronizes with the release.

\[
rsElem(a, b) \overset{\text{def}}{=} \text{sameThread}(a, b) \lor \text{isrmw}(b) \\
rseq(a, b) \overset{\text{def}}{=} a = b \lor \\
rsElem(a, b) \land \text{mo}(a, b) \land \\
(\forall c. \text{mo}(a, c) \land \text{mo}(c, b) \Rightarrow rsElem(a, c))
\]
Release sequences are too strong

Initially \( x = y = 0 \).

\[
\begin{align*}
a &= 1; \\
x.\text{store}(1, \text{rel}); \\
x.\text{store}(3, \text{rlx}); \\
\textbf{while} (x.\text{load}(\text{acq}) \neq 3); \\
&\quad a = 2; \\
&\quad x.\text{store}(2, \text{rlx});
\end{align*}
\]

But this one is racy according to C11. The acquire no longer synchronizes with the release.

\[
\begin{align*}
\text{rsElem}(a, b) & \overset{\text{def}}{=} \text{sameThread}(a, b) \lor \text{isrmw}(b) \\
rseq(a, b) & \overset{\text{def}}{=} a = b \lor \\
& \quad \text{rsElem}(a, b) \land \text{mo}(a, b) \land \\
& \quad (\forall c. \text{mo}(a, c) \land \text{mo}(c, b) \Rightarrow \text{rsElem}(a, c))
\end{align*}
\]

Viktor Vafeiadis

‘Easy’ fixes to the C11 memory model
Proposed fix for release sequences

\[ rseq_{new}(a, b) \overset{\text{def}}{=} a = b \]
\[ \lor \ sameThread(a, b) \land mo(a, b) \]
\[ \lor \ isrmw(b) \land rseq_{new}(a, rf(b)) \]