Recap

We considered a simplified C11 model:

- Each memory accesses has a *mode*:
  - Reads: rlx or acq
  - Writes: rlx or rel
  - RMWs: rlx, acq, rel or acq-rel

- Synchronization:
  \[
  G.\text{sw} = [W^{\text{rel}}]; \ G.\text{rf}; [R^{\text{acq}}]
  \]

- Happens-before:
  \[
  G.\text{hb} = (G.\text{po} \cup G.\text{sw})^+ \]

- C11-consistent wrt mo:
  \[
  \text{hb}|_{\text{loc}} \cup \text{rf} \cup \text{mo} \cup \text{rb} \text{ is acyclic}
  \]

- C11-consistent:
  complete & C11-consistent wrt some mo
The C/C++11 memory model

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 \texttt{sw} (“release sequences”)
C11 model through examples

1. int a = 0;
   int x = 0;
   a = 42;  \( \text{if}(x == 1) \{ \)
   x = 1;  \( \text{print}(a); \)
   \}

2. int a = 0;
   \text{atomic\_int} x = 0;
   a = 42;  \( \text{if}(x_{\text{rlx}} == 1) \{ \)
   x_{\text{rlx}} = 1;  \( \text{print}(a); \)
   \}

3. int a = 0;
   \text{atomic\_int} x = 0;
   a = 42;  \( \text{if}(x_{\text{acq}} == 1) \{ \)
   x_{\text{rel}} = 1;  \( \text{print}(a); \)
   \}

4. int a = 0;
   \text{atomic\_int} x = 0;
   a = 42;  \( \text{if}(x_{\text{rlx}} == 1) \{ \)
   \text{fence}_{\text{rel}};
   \text{fence}_{\text{acq}};
   x_{\text{rlx}} = 1;  \( \text{print}(a); \)
   \}
The “synchronizes-with” relation

\[
sw \triangleq ([W^{\text{rel}}] \cup [F^{\text{rel}}]; po); rf; ([R^{\text{acq}}] \cup po; [F^{\text{acq}}])
\]

Fence modes

- `acq` → `acq-rel` → `sc`
- `rel`
Release sequences (RMW’s)

\[
\begin{align*}
    x_{rlx} & := 42; \\
    y_{rel} & := 1 \\
    a & := FAI_{rlx}(y); \quad \text{// 1} \\
    b & := y_{acq}; \quad \text{// 2} \\
    c & := x_{rlx}; \quad \text{// 0}
\end{align*}
\]

\[
sw \triangleq \left( [W^{\supseteq \text{rel}}] \cup [F^{\supseteq \text{rel}}]; po \right) \cup \left( [R^{\supseteq \text{acq}}] \cup po; [F^{\supseteq \text{acq}}] \right)
\]
Release sequences (thread internal)

\[
\begin{align*}
x_{rlx} & := 42; \\
y_{rel} & := 1; \\
y_{rlx} & := 2; \\
\end{align*}
\]

\[
a := y_{acq} \parallel 2 \\
b := x_{rlx} \parallel 0
\]

\[
sw \triangleq ([W \sqsupseteq \text{rel}]; po|_{\text{loc}} \cup [F \sqsupseteq \text{rel}]; po); rf^+; ([R \sqsupseteq \text{acq}] \cup po; [F \sqsupseteq \text{acq}])
\]
C11 “synchronizes-with” relation

**Read modes**

\[ \text{na} \rightarrow \text{rlx} \rightarrow \text{acq} \rightarrow \text{sc} \]

**Write modes**

\[ \text{na} \rightarrow \text{rlx} \rightarrow \text{rel} \rightarrow \text{sc} \]

**RMW modes**

\[ \text{rlx} \rightarrow \text{acq-rel} \rightarrow \text{sc} \]

\[ \text{rel} \rightarrow \text{acq-rel} \rightarrow \text{sc} \]

**Fence modes**

\[ \text{acq} \rightarrow \text{acq-rel} \rightarrow \text{sc} \]

\[ \text{rel} \rightarrow \text{acq-rel} \rightarrow \text{sc} \]

\[ \text{sw} \triangleq ([W^{\sqsubseteq \text{rel}}]; \text{po}|_{\text{loc}} \cup [F^{\sqsubseteq \text{rel}}]; \text{po}); \text{rf}^\dagger; ([R^{\sqsubseteq \text{acq}}] \cup \text{po}; [F^{\sqsubseteq \text{acq}}]) \]

\[ \text{hb} \triangleq (\text{po} \cup \text{sw})^\dagger \]
“Catch-fire” semantics

Definition (Race in C11)

Given a C11-execution graph \( G \), we say that two events \( a, b \) are \( \text{C11-race} \) in \( G \) if the following hold:

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

\( G \) is called \( \text{C11-racy} \) if some \( a, b \) C11-race in \( G \).

Definition (Allowed outcome under C11)

An outcome \( O \) is \textit{allowed} for a program \( P \) under C11 if there exists an execution graph \( G \) such that:

- \( G \) is an execution graph of \( P \)
- \( G \) is C11-consistent.
- \( G \) has outcome \( O \) or \( G \) is C11-racy.
C11 consistency

Definition

Let \( mo \) be a modification order for an execution graph \( G \).
\( G \) is called C11-consistent \(^\bigstar\) wrt \( mo \) if:

\[ \text{hb}\mid_{loc} \cup rf \cup mo \cup rb \text{ is acyclic (where } rb \triangleq G.rf^{-1}; mo \setminus \text{id}). \]

\[ \ldots \text{sc} \ldots ? \]

Definition

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

\[ G \text{ is complete} \]

\[ G \text{ is C11-consistent wrt some modification order } mo \text{ for } G. \]
SC conditions

- The most involved part of the model, due to the possible mixing of different access modes to the same location.
- Currently (August 2017) under revision.
- If there is no mixing of SC and non-SC accesses, then additionally require acyclicity of $\text{hb} \cup \text{mo}_{\text{sc}} \cup \text{rb}_{\text{sc}}$.

Further reading:

- Overhauling SC atomics in C11 and OpenCL. Mark Batty, Alastair F. Donaldson, John Wickerson, POPL 2016.
(Repaired) SC condition for fences

\[ \text{eco} \triangleq (\text{rf} \cup \text{mo} \cup \text{rb})^+ \]  
(extended coherence order)

\[ \text{psc}_F \triangleq [F^{sc}]; (\text{hb} \cup \text{hb}; \text{eco}; \text{hb}); [F^{sc}] \]  
(partial SC order on fences)

**Condition on SC fences**

\[ \text{psc}_F \text{ is acyclic} \]

**Example: SB with fences**

\[ x = y = 0 \]
\[ x_{rlx} := 1; \quad y_{rlx} := 1; \]
\[ \text{fence}(\text{sc}); \quad \text{fence}(\text{sc}); \]
\[ a := y_{rlx}; \quad // 0 \quad b := x_{rlx}; \quad // 0 \]
\[ X \text{ behavior disallowed} \]
Exercise: ARC

\[
a = \text{new}(v)
\]

\[
y = \text{read}(a)
\]

\[
\text{clone}(a)
\]

\[
\text{drop}(a)
\]

\[
\text{new}(v)\{
    a = \text{alloc}();
    a.data = v;
    a.count = 1;
    \text{return } a;
\}
\]

\[
\text{clone}(a)\{
    \text{FADD}(a.count, +1);
\}
\]

\[
\text{drop}(a)\{
    t = \text{FADD}(a.count, -1);
    \text{if}(t == 1)\{
        \text{free}(a);
    \}
\}
\]

\[
\text{FADD} = \text{fetch\_and\_add}
\]
writer(v1,v2) {
    local a,b;
    do {
        a = s;
        if (a % 2 == 1)
            continue;
        b = CAS(s,a,a+1);
    } while (¬b);
    x1 = v1;
    x2 = v2;
    s = a + 2;
}

casual reader(t1,t2) {
    local a,b;
    while (1) {
        a = s;
        if (a % 2 == 1)
            continue;
        t1 = x1;
        t2 = x2;
        b = s;
        if (a==b) return;
    }
}

Exercise: seqlock