Compiler optimisations under C11

— Recall that a program transformation $S \rightsquigarrow T$ is correct iff $\text{Behaviours}_{C11}(T) \subseteq \text{Behaviours}_{C11}(S)$.

Q: What "Behaviours" are we interested in preserving?

→ the (A, lab, po) components of executions?
NO, because then the compiler cannot remove any events.

→ the externally visible events (e.g., print statements)?

→ the final value of the global variables?
(i.e. the maximal-mo events)

→ whether the program terminates or diverges?

all of these are sensible.

For simplicity, let's just fix a set of "externally visible" variables, X, and preserve only the part of the execution corresponding to events on X, i.e.

$$A_X \stackrel{\text{def}}{=} \{ a \in A \mid loc(a) \in X \},$$

$$lab_X \stackrel{\text{def}}{=} \lambda a. \begin{cases} lab(a) & \text{if } a \in A_X \\ \bot & \text{o/w} \end{cases}$$

$$rf_X \stackrel{\text{def}}{=} \lambda a. \begin{cases} rf(a) & \text{if } a \in A_X \\ \bot & \text{o/w} \end{cases}$$

$$mo_X \stackrel{\text{def}}{=} \{ (a,b) \mid mo(a,b) \wedge a \in A_X \wedge b \in A_X \}$$

$$sc_X \stackrel{\text{def}}{=} \{ (a,b) \mid sc(a,b) \wedge a \in A_X \wedge b \in A_X \}.$$

In other words, we should change any of the X-related events, but we can change any of the others

Sequentialization

$$C_1 \,||\, C_2 \rightsquigarrow C_1 ; C_2$$

- Valid under SC
- We have to be a bit careful regarding non-terminating C_1's.

```
x = 0
while (x==0) { }   ||   x = 1
```

may diverge (but diverges only under unfair execution/interleaving)

$\overset{?}{\rightsquigarrow}$

```
x = 0
while(x==0) { }
x = 1
```

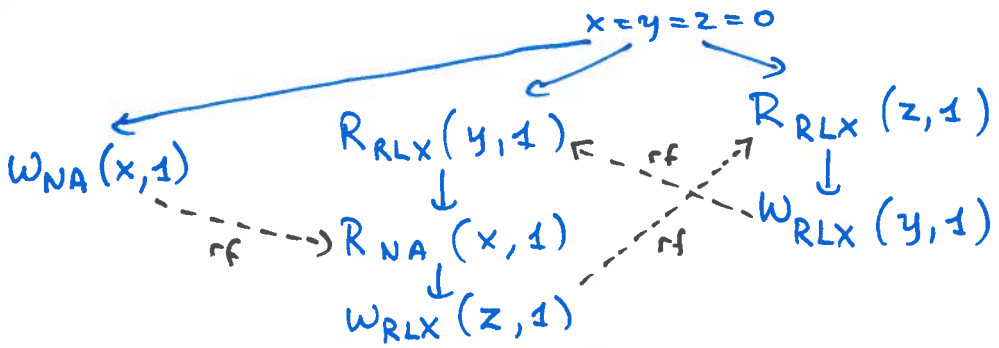
diverges

- So, if we (want to) rule out unfair executions, the transformation is not valid under SC.
- We can regain validity under fair interleaving semantics by requiring C_1 to always terminate.

A simple special case: $C_1 \overset{def}{=} x_{NA} = 1$.
Is the transformation valid? Consider the program:

$$x_{NA} = y_{NA} = z_{NA} = 0$$

$$P \overset{def}{=} \Big(x_{NA} = 1 \;\Big\|\; \text{if } (y_{RLX} == 1) \text{ if } (x_{NA} == 1) \; z_{RLX} = 1 \;\Big\|\; \text{if } (z_{RLX} == 1) \; y_{RLX} = 1 \Big)$$



Execution is inconsistent according to C11

What's the problem?

There are two problems:

- Dependency cycles
- Non-monotonicity of ConsistentRFna axiom.

Solutions?

- Strengthen the model to rule out dependency cycles. (Non-obvious.)
- Strengthen the model to rule out all (hb ∪ rf) cycles. (Has an implementation cost.)
- Weaken the model by dropping the ConsistentRFna axiom. (Then the DRF theorem does not hold.)
- Forbid RLX accesses. (Forbidding RLX writes & strengthening the axioms for RLX reads should also work.)
- Something else? (Open research topic.)

Here, for simplicity, we just rule out RLX accesses.

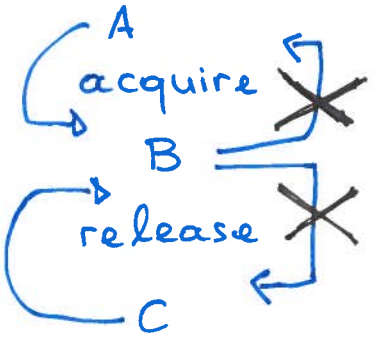
"Roach motel" reorderings

$A ; B \overset{?}{\rightsquigarrow} B ; A$ — when is such a reordering transformation valid.

Some simple cases:

- A & B are non-atomic accesses to different locations
- A or B is skip

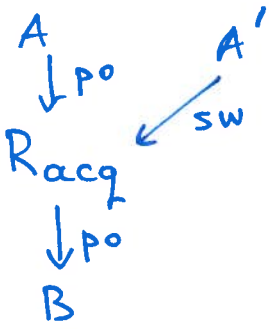
What if A and/or B are atomic?



Think of acquire & release accesses as acq/rel of locks.

You can move commands inside the critical region, but not out of it.

Why?

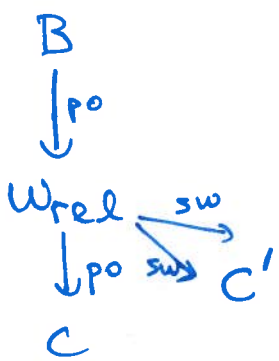


Event B by being after an acquire is guaranteed to see all updates hb the Racq.

If we move B before the Racq it is no longer guaranteed to see the A' updates.

Conversely for releases:

The programmer can know that hb(B,C'), but if we move B after the Wrel, this knowledge is lost.

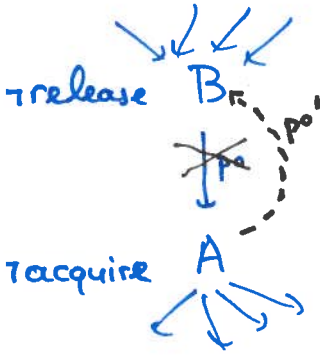


These one-way reorderings are referred to as "roach motel" reorderings. (In a filthy motel, cockroaches check in, but never check out.)

Verifying "roach motel" reorderings

- Assume there are no RLX accesses. (Then, $rf \subseteq hb$.)
- Consider the reordering $A;B \rightsquigarrow B;A$ where $loc(A) \neq loc(B)$, $\neg acquire(A)$, $\neg release(B)$.

To show: If DRF(Src), then for every consistent Tgt execution, there is a consistent Src execution with the same events and $po' := po \setminus \{(B,A)\} \cup \{(A,B)\}$.



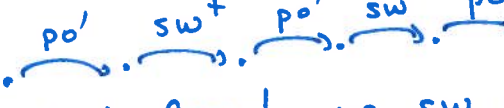
Observe that the only immediate outgoing hb-edge from B is the po edge to A.
Likewise, the only immediate incoming hb-edge to A is the $B \xrightarrow{po} A$ edge.

Key Lemma: $hb' \stackrel{def}{=} (po' \cup sw)^+ \subseteq hb \setminus \{(B,A)\} \cup \{(A,B)\}$

Consider an hb' path

(no sw edges)

or



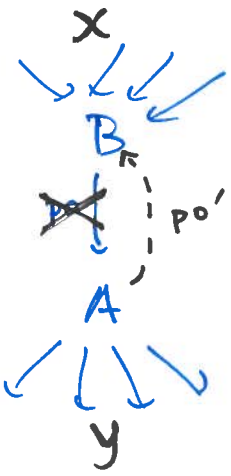
(at least one sw edge)

In the former case, $hb' = po' \subseteq hb \setminus \{(B,A)\} \cup \{(A,B)\}$, trivial
In the latter case, none of the $\xrightarrow{po'}$ edges can be A→B as there are no $\xrightarrow{sw} A$ and $B \xrightarrow{sw}$. Therefore, that is also a valid $(po' \setminus \{(A,B)\} \cup sw)^+$ path, i.e. a valid hb path.
Finally, $(B,A) \notin hb'$ because there are no $\xrightarrow{sw} A$ and $B \xrightarrow{sw}$., and A & B are immediate neighbours in po/po'.

With this lemma, we can validate all the axioms except (ConsistentRFna) where hb appears positively.

Verifying "roach motel" reorderings (ctd.)

Recall the picture and the (Consistent RFna) axiom.



$\forall xy.\ rf(y) = x \wedge \left(NA(x) \vee NA(y)\right) \Rightarrow hb(x,y)$

The case where hb is affected is if

$$hb^*(x, B) \wedge hb^*(A, y).$$

Note that we cannot have both $x = B$ and $y = A$, because $loc(A) \neq loc(B)$.

Consider the earliest y (in hb-order) violating the axiom for hb'.

Then, construct a prefix execution of the program containing up to the event y (in hb-order). Further make y read from some hb-earlier event that writes to the same location (or $\perp$ if no such event exists). That constructed prefix execution is consistent and racy (there is a race between x & y), contradicting our DRF(Src) assumption.

Optimisations on NA-accesses and the power of DRF.

Consider the sequence of "optimisations" shown below:

```
                          t = y_NA                     t = y_NA
if (x_ACQ == 1)     ~>    if (x_ACQ == 1)      ~>      if (x_ACQ == 1)
    print(y_NA)               print(y_NA)                  print(t).
```

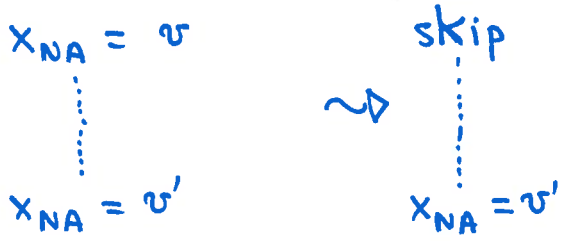
- First, we introduced a redundant load of y.
- Then, we did common subexpression elimination (CSE) over the acquire instruction.
- The net-effect is that we moved the access of y before the x_{ACQ}, which is clearly wrong. (It goes against the "roach motel" principle.)

Therefore one of the two "optimisations" should be forbidden. Which should that be?

- SC, TSO, Coherence, Release-Acq, Power, ARM, PSO, RMO all allow the first transformation, but not the second.
- In C11, compilers writers may want to do the second (and therefore not the first) because only the second makes the program run faster. So, indeed, the model chosen allows the second but not the first. [The first transformation may introduce a race, whereas for the second, the value of y cannot have changed without a race.]

NA - optimisations allowed by DRF models

① Overwritten write elimination:

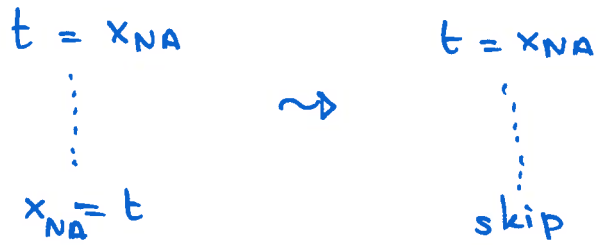


provided that x is not accessed in between & there is no REL-ACQ pair in between.

② Write after write elimination:

provided that there is no REL-ACQ pair in between

③ Write after read elimination



- t unchanged in between.
- t local variable
- No REL-ACQ pair in between

④ Read after read elimination

- t local variable
- t unchanged in between
- No REL-ACQ pair in between

⑤ Read after write elimination

- No REL-ACQ pair in between.

(Argument: Any execution that could read/write x_{NA} in parallel is racy.)