Data races & Data Race Freedom (DRF)

→ Two operations are conflicting if they both access the same location and at least one is a write

→ Two operations are concurrent if they are not ordered by happens-before.

→ A data race (or simply, a race) occurs whenever we have two concurrent conflicting operations, such that at least one is non-atomic.

$$\begin{aligned} race(a,b) \stackrel{def}{=} \quad & a \neq b \wedge \neg hb(a,b) \wedge \neg hb(b,a) \\ & \wedge (isWrite(a) \vee isWrite(b)) \\ & \wedge (isNA(a) \vee isNA(b)) \\ & \wedge (\exists \ell .\ loc(a) = loc(b) = \ell) \end{aligned}$$

→ A program is DRF (data race free) if every consistent execution of the program has no races.

→ The semantics of C11 programs is defined to be:
- either the set of all its consistent executions if they are DRF,
- or "undefined" if the program has a consistent execution with a data race.

→ This style of defining weak memory models is also known as the "DRF" style.

→ Motivation for giving undefined semantics to racy programs comes from optimising compilers.
A program transformation $S \to T$ is deemed correct if $\boxed{Behaviours(T) \subseteq Behaviours(S)}$.

Examples:

Sequential:

(1) `int x; print(x)` $\rightsquigarrow$ `print(0)`

(can print arbitrary value)

(2) `*NULL = 3` $\rightsquigarrow$ skip [MSVC does this]

(undefined/crash)

SC

(3) $x = 1 \parallel y = 2 \rightsquigarrow x = 1 ; y = 2$

(4)

Common subexpression elimination (CSE)

$$\begin{array}{c} x=0;\ p=\&x; \\ \begin{array}{l\|l} a = x; & \\ b = *p; & x = 1 \\ c = x; & \\ \text{print}(a,b,c); & \end{array} \end{array} \quad \not\rightsquigarrow \quad \begin{array}{c} x=0;\ p=\&x \\ \begin{array}{l\|l} a = x; & \\ b = *p; & x = 1 \\ c = a; & \\ \text{print}(a,b,c); & \end{array} \end{array}$$

<u>cannot</u> print 0,1,0 (Coherence) — can print 0,1,0.

C11

(4) The LHS is undefined. The compiler can do anything it likes!

→ But race freedom is a global property, and makes it impossible to reason locally against a malicious client. Consider the program:

$$\text{secret} = 7 \parallel \text{Client}$$

where the Client doesn't know the secret (and does not mention the "secret" variable). How can we ensure that it never finds it out?

[We cannot. If the client has a race, the compiler is allowed to leak the secret.]

The DRF theorem

⇒ If under SC a program has no races, then:
Behaviours$_{\text{WeakMem}}$(Program) = Behaviours$_{\text{SC}}$(Program)

⇒ Considered as a sanity check for memory models.

⇒ Allows reasoning under SC for DRF programs
both for (a) showing that the program is DRF,
and (b) proving some property of the reachable program states.

....

Q: What's a race under SC?

A: Two concurrent, conflicting memory accesses where at least one is non-atomic. (as before)

w.r.t. the happens-before order induced by the program order & synchronization (i.e. whenever we have a $\xrightarrow{rf}$ edge)

again, we assume some operations are marked as "atomic"/"volatile" and are used to synchronize between threads.
the others are "non-atomic" and concurrent accesses to them result in a race.

Q: What are Behaviours$_{\text{WeakMem}}$ / Behaviours$_{\text{SC}}$?

A: The set of consistent executions of the program according to the appropriate memory model. (And in case of a DRF model such as C11, Behaviours$_{\text{DRFmodel}}$ is the universe set if the program has a consistent racy execution.)

Does the DRF theorem hold for C11?

→ Not as stated. Example:

$$x = y = 0$$

$x_{REL} = 1$	$y_{REL} = 1$
y_{ACQ}	x_{ACQ}

Has no races. Yet, the two threads can read 0 & 0, which is not possible under SC.

→ OK. So how about the following:

" If under SC the program P has no concurrent, conflicting accesses such that at least one of them is a non-SC access, then

$$\text{Behaviours}_{C11}(P) = \text{Behaviours}_{SC}(P).$$ "

→ Again, doesn't hold because of the dependency cycles:

$$x = y = 0$$

if $(x_{RLX} == 1)$	if $(y_{RLX} == 1)$
$y_{RLX} = 1$	$x_{RLX} = 1$

rf rf

→ What if we remove dependency cycles either by

(a) Strengthening the model by adding the axiom: acyclic $(hb \cup rf \cup sc)$

or (b) assume that there are no relaxed accesses in the program? (in this case $rf \subseteq hb$.)

→ Claim: the theorem now holds.

[Batty et al. POPL'12] prove a slightly simplified version with no REL/ACQ.

For simplicity, we will deal with case (b)

To show:

($\supseteq$) easy ⊛ $\mathrm{Consistent}_{SC}(A, lab, po, rf, sc) \Rightarrow$
$\exists mo', sc'.\ \mathrm{Consistent}_{C11}(A, lab, po, rf, mo', sc')$

[Pick $mo' := \{(a,b) \mid sc(a,b) \wedge \exists \ell.\ \mathrm{isWrite}_\ell(a) \wedge \mathrm{isWrite}_\ell(b)\}$
$sc' := \{(a,b) \mid sc(a,b) \wedge \mathrm{isSC}(a) \wedge \mathrm{isSC}(b)\}$]

⊛ $\mathrm{DRF}_{SC}(A, lab, po, rf, sc) \Rightarrow$
$\mathrm{DRF}_{C11}(A, lab, po, rf, mo', sc')$

[the happens-before relation has not changed.]

($\subseteq$)

Lemma : $\mathrm{DRF}_{C11}(A, lab, po, rf, \cancel{mo}\ sc) \wedge \mathrm{Consistent}_{C11}(A, lab, po, rf, mo, sc)$
[first attempt] $\Rightarrow \exists sc'.\ \mathrm{Consistent}_{SC}(A, lab, po, rf, sc')$

Recall DRF_{C11} in this context : "all conflicting concurrent accesses must be SC-accesses"

conflicting: in $(mo \cup rf \cup fr)^+$
concurrent: not in $hb \cup hb^{-1} \cup (=)$

So:
$$\begin{pmatrix} \mathrm{Non\text{-}SC} \times \mathrm{Non\text{-}SC} \\ \cup\ \mathrm{SC} \times \mathrm{Non\text{-}SC} \\ \cup\ \mathrm{Non\text{-}SC} \times \mathrm{SC} \end{pmatrix} \cap (mo \cup rf \cup fr)^+ \subseteq hb \cup \cancel{hb^{-1}} \cup \cancel{(=)}$$

ruled out by coherence.

SC-accesses are ordered by the sc order, and we know that $hb|_{sc} \subseteq sc$. (ConsistentSChb)

Therefore, acyclic $(hb \cup mo \cup rf \cup fr)$.

Pick sc' to be any total irreflexive order extending $(hb \cup mo \cup rf \cup fr)$.

Oops, we proved a weaker result, i.e. that race-free C11 executions correspond to SC ones. What about racy C11 executions?

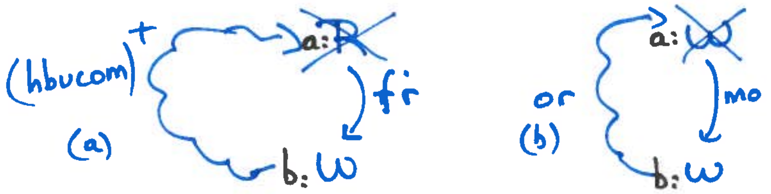
- Well, if acyclic $(hb \cup \underbrace{mo \cup rf \cup fr}_{com})$, then we can find an SC execution.
- So what if there is a $hb \cup com$ cycle?

Dealing with ($hb \cup com$) cycles

Observe that:

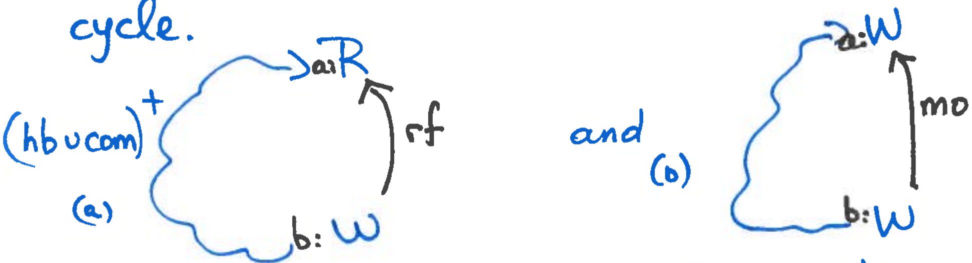
> Given a program P and a consistent execution X of P, we can remove any $(hb \cup rf_{USC})$-maximal event from X and get another consistent execution of P.

- Now consider an execution with a (minimal) ($hb \cup com$) cycle.
- Remove a maximal $(hb \cup rf_{USC})$ event. Either the cycle remains, or it vanishes. In the former case, we continue removing events. (As the graph is finite, we will eventually reach the second case.)
- In the latter case, there were two possibilities:



(The other two $\cancel{\to}$ hb and $\cancel{\to}$ rf are not $(hb \cup rf)$-maximal.)

- In both cases, we can construct a cycle-free execution of the execution with the memory accesses of the cycle that preserves the cycle.



- Since the execution is ($hb \cup com$)-acyclic, we can pick sc' to be any strict total order extension of ($hb \cup com$), thereby ensuring that the execution is also consistent under SC.
- Finally, since we haven't changed the events of the cycle, nor the hb relation (as in subcase (a), the read is a non-SC access), the race persists in the consistent SC execution, contradicting our assumption that all consistent SC executio[ns] of P are race-free.