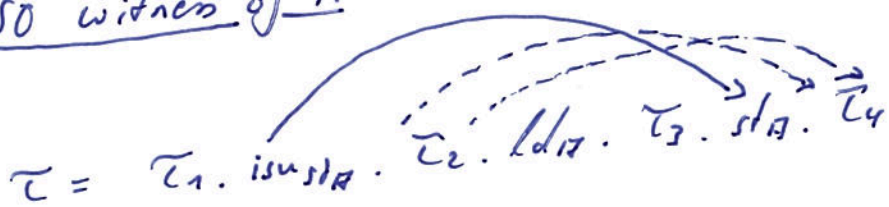
10.3 Attacks on Robustness

Goal: Rephrase robustness in terms of a simple problem:
absence of feasible attacks.

Definition (Attacks):

- An attack is a triple $A = (t_A, stinst, ldinst)$
  (consisting of
  - ↳ a thread $t_A \in \overline{TID}$, called attacker,
  - ↳ a store instruction $stinst$ in t_A, and
  - ↳ a load instruction $ldinst$ in t_A.
- A TSO witness of A is a computation of the form



so that
(W1) Only the attacker delays stores
(W2) Store st_A is an instance of $stinst$.
It is the first store of the attacker
that is delayed.
Load ld_A is an instance of $ldinst$.
It is the last action of the attacker
overtaken by st_A.
  - ↳ So τ_2 contains loads, assignments, asserts, and issues,
    but no fences and stores of the attacker.
  - ↳ It may contain arbitrary actions from the other threads,
    called helpers.

(W3) For all actions act in $ld_A\ \tau_3\ st_A$ we have $ld \rightarrow_{hb}^{*} act$.
Here, an isu + st of a helper is counted as one action act.

(W4) Sequence τ_4 only consists of stores of the attacker that were issued before ld_Π and have been delayed

(W5) All these stores st satisfy
$addr(st) \neq addr(ld_\Pi)$,
which means ld_Π has not read its value early.

If a TSO witness for Π exists, the attack is called <u>feasible</u>.

<u>Example (Dekker = SB)</u>:

l_0: mem[x] ← 1 <u>goto</u> l_1; ‖ l_0': mem[y] ← 1 <u>goto</u> l_1';
l_1: r_1 ← mem[y] <u>goto</u> l_2; ‖ l_1': r_2 ← mem[x] <u>goto</u> l_2';

There is an attack $\Pi = (t_1, st_{inst}, ld_{inst})$
with
st_{inst} = the store at l_0
ld_{inst} = the load at l_1.
A TSO witness of the attack is

$\tau = (t_1, isu).(t_1, ld, y, 0).\underbrace{(t_2, isu).(t_2, st, y, 1).(t_2, ld, x, 0)}_{\tau_3}.(t_1, st, x, 1)$
isu_{st_Π} ld_Π st_Π

So attack Π is feasible.
The program contains a symmetric attack Π'
with t_2 as the attacker.

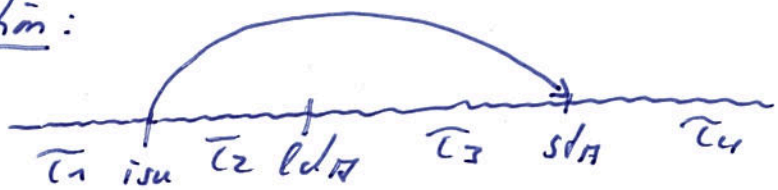
<u>Theorem (Characterisation of robustness with attacks)</u>:
Program P is robust iff no attack is feasible.

<u>Proof</u>:
$\Rightarrow$: A TSO witness comes with a happens-before cycle
$st_\Pi \rightarrow_{po}^+ ld_\Pi \rightarrow_{hb}^+ st_\Pi$.

$\Leftarrow$ Show that if P is not robust,
then there is a feasible attack.

- Among the violating computations,
we select $\tau \in C_{TSO}(P)$ with $\#(\tau)$ minimal.
By locality, only one thread t_A uses its buffer.
Hence, (W1) holds.
- Initially, the attacker t_A executes under SC,
so stores immediately follow their issues.
↳ This computation is embedded into τ_1.
Eventually, the attacker starts delaying stores.
↳ Let st_A be the first store that is delayed.
↳ It gets delayed past several loads,
the last being ld_A. Together, we get (W2).

Illustration:



- For the helpers, note that $ld_A \xrightarrow{+}_{hb} st_A$ through τ_3
by dichotomy.
Consider the actions act with
$ld_A \xrightarrow{+}_{hb}$ act through the intermediary computation.
Then act can be moved before ld_A using dichotomy.
Hence, (W3)
- With cycle $st_A \xrightarrow{*}_{po} ld_A \xrightarrow{+}_{hb} st_A$,
τ_4 only needs to contain stores of the attacker
that have been delayed past ld_A.

These stores are non-blocking, so the helpers can stop with the last action of τ_3.
We can moreover assume ld_T to be the program-order last action of the attacker.
(W4) holds.

- The early read could be avoided by moving ld_T to τ_4.
  This would save a delay and hence contradict minimality.

Together, we have a TSO witness of the attack

$$(d_T, st_{inst}, ld_{inst})$$

where st_{inst} is the instruction of st_T and
ld_{inst} is the instruction of ld_T. □

Note:
- The number of attacks is quadratic in the size of the program.
- Enumerate them and check each for a TSO witness.
- How to? Instrumentation – next.