11. Robustness against Partial Store Ordering

Goal:
- Solve robustness against partial store ordering (PSO), a memory model weaker than TSO.
- PSO does not exist in practice
  - ↳ PGAS-clusters at Fraunhofer ITWM behave similarly (partitioned global address space)
  - ↳ The approach to show decidability carries over to Power.

Problem: PSO is not local

Approach:
- Show a weaker normal form result
- Develop an automaton model stronger than SC reachability that is able to detect such normal form computations.

11.1 Partial Store Ordering and Robustness

Syntax of programs: As before.

PSO-Semantics of programs:

Idea:
- There is a buffer per address (per thread)
- Hence, stores to different addresses can overtake each other.
- Loads read their values early from the corresponding buffer
- A fence flushes all buffers (heavy-weight synchronization)

Technically:
- Configurations take the form (pc, mem, buf)

with

$pc : TID \to LOC$

$mem : Var \to DOM$. Here $VAR := DOM \cup \bigcup_{t \in TID} t^*$

$buf : TID \times DOM \to DOM^*$.

- The set of actions that serve as transition labels is

$$ACT := TID \times (\{Local, mfence\} \cup \{isu, ist\} \times DOM)$$

- The definition of the (labelled) transition relation for PSO is left as homework.

- We use

$$C_{PSO}(P) := \{ \tilde{c} \in ACT^* \mid s_0 \xrightarrow{\tilde{c}}_{PSO} s \text{ with } buf(t,a) = \varepsilon \text{ for all } t \in TID, a \in DOM \}$$

for the set of all PSO computations.

- As in the case of TSO, we associate with a computation $\tilde{c} \in C_{PSO}(P)$ a trace $Tr(\tilde{c})$ that only keeps

$\to_{po}$, $\to_{st}$, and $\to_{src}$.

Moreover, $\to_{cf}$ is again a derived relation.

We use

$$Tr_{PSO}(P) := Tr(C_{PSO}(P))$$

for the set of all PSO traces.

- The robustness problem against PSO is defined as follows

Given: A parallel program P.

Question: Does $Tr_{PSO}(P) \subseteq Tr_{SC}(P)$ hold?

- To find a way to handle PSO robustness.
  We again phrase robustness in terms of (absence of) happens-before cycles.
- Recall that $\rightarrow_{src}$ and $\rightarrow_{st}$ define $\rightarrow_{ff}$ and that the <u>happens-before relation</u> of $T_r(\bar{\tau})$ is

$$\rightarrow_{hb} := \rightarrow_{po} \cup \rightarrow_{src} \cup \rightarrow_{st} \cup \rightarrow_{ff}.$$

Now the Shasha & Snir lemma still holds for PSO.

<u>Lemma (Shasha & Snir 1988):</u>
Consider $T_r(\bar{\tau}) \in Tr_{PSO}(P)$.
Then $T_r(\bar{\tau}) \in Tr_{SC}(P)$ iff $\rightarrow_{hb}$ is acyclic.

Again, we call a computation $\bar{\tau} \in C_{PSO}(P)$ <u>violating</u> if $T_r(\bar{\tau})$ is cyclic (in terms of happens-before).

11.2 Non-Locality and Normal-Form Violations

<u>Goal:</u> Show that we can assume a particular shape of violating computations.

<u>Proposition (conjecture):</u>
PSO is not local.

<u>Proof:</u>
- In a simplified programming model, the following program disproves locality:

$$\begin{array}{l|l|l} w(z,1); & r(x,1); & w(z,1); \\ w(x,1); & r(y,1); & w(y,1); \\ & r(z,0); & \end{array}$$

- When translating this program to our assembly model, it no longer serves as a counterexample to locality:

l_1: mem[z] ← 1 goto l_2;	l_1': r ← mem[x] goto l_2';	l_1'': mem[z] ← 1 goto l_2'';
l_2: mem[x] ← 1 goto l_3;	l_2': assert r = 1 goto l_3';	l_2'': mem[y] ← 1 goto l_3'';
	l_3': r ← mem[y] goto l_4';	
	l_4': assert r = 1 goto l_5';	
	l_5': r ← mem[z] goto l_6';	
	l_6': assert r = 0 goto l_7';	

Problem: · The assert at l_6' is not needed to find a cycle.
· If the assert is missing, there is no need to reorder l_1'' and l_2''.

Lesson learned:
↳ Commands $r(x,v)$ are too coarse an abstraction.
↳ The combination of load and assert is stronger than what can be found in practice.

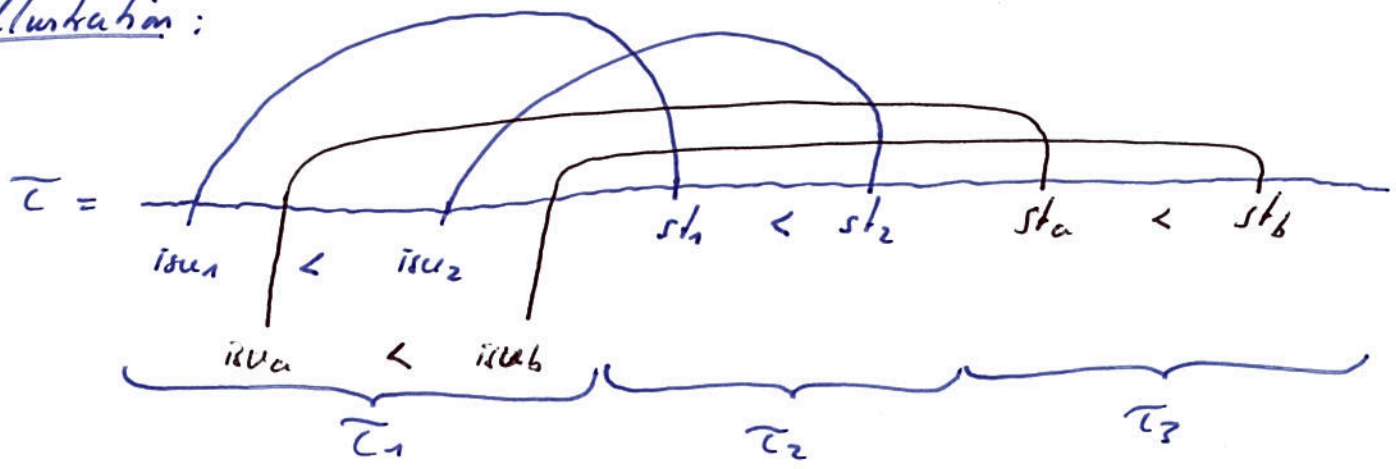
Whether PSO is local is open.

Definition (Normal-Form Computation):
A computation $\tau \in C_{PSO}(P)$ is in normal form if it consists of at most three parts

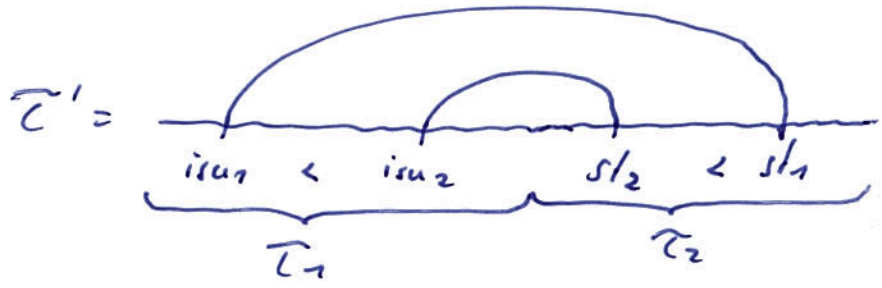
$$\tau = \tau_1 . \tau_2 . \tau_3$$

so that
(1) τ_2, τ_3 only contain store actions that were delayed, (without issues).
(2) there are no delays within τ_1, and
(3) for all $isu_1, isu_2 \in \tau_1$ with associated $st_1, st_2 \in \tau_i$, $i = 2,3$, we have $isu_1 < isu_2$ implies $st_1 < st_2$.

Illustration:



↳ $\bar{\tau}$ is in normal form.
↳ The following computation $\bar{\tau}'$ is not in normal form:



What is cool about this normal form?

If $\bar{\tau}_1$, $\bar{\tau}_2$, and $\bar{\tau}_3$ are processed simultaneously,
they can be accepted in streaming-fashion,
without need for unbounded storage.

Theorem (Normal-Form Result, Čulin, Derevenetz, Majumdar, M. '13)

Program P is robust iff
there is no violation $\tau \in C_{PSO}(P)$ in normal form.

The direction from left to right is by Sharma & Snir.
For the reverse implication, we need combinatorics:
↳ For TSO we studied a violation with fewest delays.
↳ For PSO we now concentrate on a shortest violation.

Need an interesting property of PSO.

Lemma (Cancellation):

Let $E \neq \tau \in C_{PSO}(P)$.

There is an action $a \in \tau$ so that

$\tau \setminus a \in C_{PSO}(P)$.

Here,

$\tau \setminus a :=$ τ with a removed.

If a is an issue, we also remove the corresponding store action.

Proof (Cancellation):

Consider the last action a in τ that is not a store.

Then
- a is the program-order last action of a thread and
- the following actions are executable unconditionally (stores that leave buffers).

Hence, a can be removed while preserving feasibility, which means

$\tau \setminus a \in C_{PSO}(P)$. □

Proof (Normal-Form Result):

$\Leftarrow$: Consider a shortest violation $\tau \in C_{PSO}(P)$.

With the cancellation lemma, τ contains an action a that can be removed.

There are two cases:

$a \neq isu$:

$\tau = \tau_1 . a . \tau_2$ and $\tau \setminus a := \tau_1 . \tau_2$

$a = isu$:

$\tau = \tau_1 . isu . \tau_2 . sd . \tau_3$ and $\tau \setminus a := \tau_1 . \tau_2 . \tau_3$

We focus on the latter case.

We have

$$\bar{\tau} \setminus a := \bar{\tau}_1 . \bar{\tau}_2 . \bar{\tau}_3$$

is shorter than $\bar{\tau}$.

By minimality of $\bar{\tau}$,

$\bar{\tau} \setminus a$ cannot be violating.

Hence, there is an SC computation $\sigma \in C_{sc}(P)$ with

$$Tr(\bar{\tau} \setminus a) = Tr(\sigma).$$

We project σ to $\bar{\tau}_1$, $\bar{\tau}_2$, and $\bar{\tau}_3$, and reinsert a:

$$\bar{\tau}' := (\sigma \downarrow \bar{\tau}_1) . isu . (\sigma \downarrow \bar{\tau}_2) . sl . (\sigma \downarrow \bar{\tau}_3).$$

It can be checked that

$\bar{\tau}' \in C_{pso}(P)$,

$Tr(\bar{\tau}') = Tr(\bar{\tau})$, and

$\bar{\tau}'$ is in normal form.

□