A Denotational Semantics for SPARC TSO

Ryan Kavanagh
Stephen Brookes

MFPS XXXIII

Carnegie Mellon University

Funded in part by an NSERC Postgraduate Fellowship

Fact
The Dekker algorithm fails to achieve mutual exclusion on Sun Microsystem's SPARC architecture.

Example
Starting from initial state $\sigma = [x : 0, y : 0, z : 0, w : 0]$,

$$(x := 1; \textbf{if } y = 0 \textbf{ then } z := 1) \,\|\, (y := 1; \textbf{if } x = 0 \textbf{ then } w := 1)$$

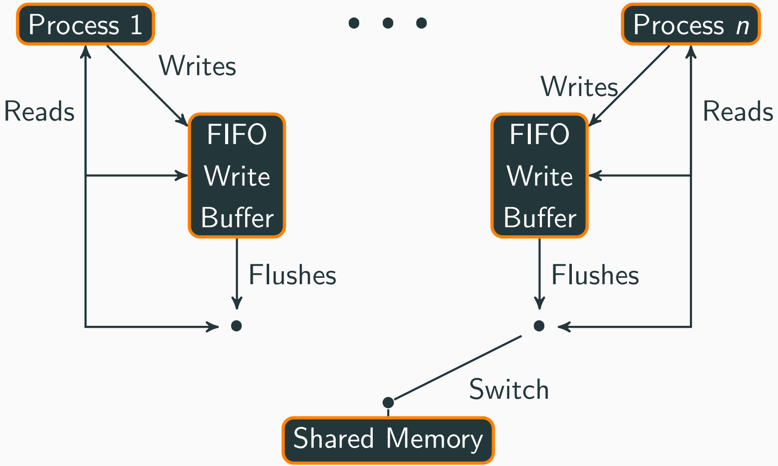
can reach a final state τ with $z = 1$ and $w = 1$.

The Implicit Assumption

The Dekker algorithm implicitly assumes **sequential consistency**, where

> *. . . the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.*
> *— Leslie Lamport, 1979*

SPARC TSO


Process 1
Process n
Writes
Writes
Reads
Reads
FIFO
Write
Buffer
FIFO
Write
Buffer
Flushes
Flushes
Switch
Shared Memory

The Problem

How can we give a compositional semantics that exactly captures the behaviour of the SPARC TSO memory model?

Partially-Ordered Multisets

Definition
A **pomset** on a set L of labels is a triple $\langle P, <, \Phi \rangle$, where

- P is a set,
- $<$ is a (strict) partial order on P, and
- $\Phi : P \to L$ is a labelling function.

The set of pomsets over L is $\mathrm{Pom}(L)$.

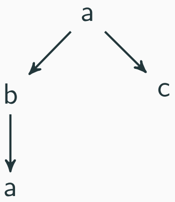
We usually write P instead of $\langle P, <, \Phi \rangle$, and let $<_P$ and Φ_P denote the order and labelling function.

Pomsets As DAGs

We can draw pomsets as labelled directed acyclic graphs. We draw $a \to b$ whenever $\Phi(p) = a$ and $\Phi(q) = b$ for some $p < q$.

Example

- $P = \{0, 1, 2, 3\}$.
- $0 < 1$, $1 < 2$, $0 < 2$, $0 < 3$.
- $\Phi(0) = a$, $\Phi(1) = b$, $\Phi(2) = a$, $\Phi(3) = c$.



Memory Actions

Assume

- a set of locations **Loc** ranged over by $x, y, z, \ldots$.
- a set of values $\mathbf{V} = \mathbb{Z}$ ranged over by v, u.

Read actions are $x = v$ for $x \in \mathbf{Loc}$ and $v \in \mathbf{V}$.
Write actions are $x := v$ for $x \in \mathbf{Loc}$ and $v \in \mathbf{V}$.
The **skip action** is δ.

These collectively form the set $\mathcal{A}_{PO}$ of **program order actions**, ranged over by λ.

Program Orders

Definition
A pomset P satisfies the **(locally) finite height property** if for all $b \in P$, $\{a \in P \mid a < b\}$ is finite.

Definition
A **program order pomset** is a pomset over the set of labels $\mathcal{A}_{PO}$ satisfying the finite height property.

TSO (Axiomatically)

TSO-consistency

Definition
Given a program order pomset P, a strict partial order $<_T$ on P is **TSO-consistent** with P from an initial state $\sigma : \mathbf{Loc} \rightharpoonup_{fin} \mathbf{V}$ if it satisfies six axioms, including:

(O)rdering: $<_T$ is a total order on the write actions of P.

(S)toreStore: for all writes $w, w' \in P$, $w <_P w'$ implies $w <_T w'$

(F)ork: if $\begin{tikzcd}[column sep=small] & \lambda_1 \arrow[dl] \arrow[dr] & \\ \lambda_2 & & \lambda_3 \end{tikzcd}$ in $<_P$, then $\begin{tikzcd}[column sep=small] & \lambda_1 \arrow[dl] \arrow[dr] & \\ \lambda_2 & & \lambda_3 \end{tikzcd}$ in $<_T$.

TSO (Denotationally)

Three Ingredients

1. TSO pomsets
2. Pomset executions
3. Soundness and completeness

A Simple Imperative Language

We restrict our attention to the following simple imperative language:

$$v ::= \ldots, -2, -1, 0, 1, 2, \ldots$$

$$e ::= v \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid \cdots$$

$$b ::= \mathit{true} \mid \mathit{false} \mid \neg b \mid e_1 = e_2 \mid e_1 < e_2 \mid b_1 \vee b_2 \mid b_1 \wedge b_2 \mid \cdots$$

$$c ::= \textbf{skip} \mid x := e \mid c_1 ; c_2 \mid c_1 \parallel c_2 \mid \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \mid \textbf{while } b \textbf{ do } c$$

$$p ::= c$$

TSO Pomsets

Goal
To assign to each expression a set of "TSO pomsets" denoting its possible TSO executions.

Buffers

Introduce a new set $\mathbf{BLoc} = \{\bar{x} \mid x \in \mathbf{Loc}\}$ of **buffer locations**.

A **buffer write action** is an action $\bar{x} := v$.

A write buffer is a list L of write actions $x := v$, with the front of the queue at the head of the list. We let **Ls** be the set of all write buffers.

TSO Pomsets

Definition
A **TSO pomset** is a pomset over the set of labels

$\mathcal{A}_{TSO} = \{\delta, x := v, \bar{x} := v, x = v \mid x \in \mathbf{Loc}, v \in \mathbf{V}\}$

satisfying the finite height property.

Combining Pomsets

The **parallel composition** $P_1 \parallel P_2$ of pomsets $\boxed{P_1}$ and $\boxed{P_2}$ is

$$\boxed{P_1} \qquad \boxed{P_2}.$$

Definition (Formally)
The parallel composition $\langle P_0, <_0, \Phi_0 \rangle \parallel \langle P_1, <_1, \Phi_1 \rangle$ is $\langle \{0\} \times P_0 \cup \{1\} \times P_1, <, \Phi \rangle$, where $(i, p) < (j, q)$ if and only if $i = j$ and $p <_i q$, and $\Phi(i, p) = \Phi_i(p)$.

Combining Pomsets

The **sequential composition** $P_1; P_2$ of pomsets $\boxed{P_1}$ and $\boxed{P_2}$ is

$$\begin{tikzcd} \boxed{P_1} \arrow[d] \\ \boxed{P_2} \end{tikzcd}$$

if $\boxed{P_1}$ is finite, and it is $\boxed{P_1}$ if $\boxed{P_1}$ is infinite.

Definition (Formally)

The sequential composition $\langle P_0, <_0, \Phi_0 \rangle ; \langle P_1, <_1, \Phi_1 \rangle$ when P_0 is finite is $\langle \{0\} \times P_0 \cup \{1\} \times P_1, <, \Phi \rangle$, where $(i, p) < (j, q)$ if and only if $i = j$ and $p <_i q$, or $i = 0$ and $j = 1$, and $\Phi(i, p) = \Phi_i(p)$. When P_0 is infinite, $\langle P_0, <_0, \Phi_0 \rangle ; \langle P_1, <_1, \Phi_1 \rangle$ is $\langle P_0, <_0, \Phi_0 \rangle$.

Three types of semantic clauses, given a write buffer $L \in \mathbf{Ls}$:

Buffer Flushes. $\mathsf{split}(L) \subseteq \mathrm{Pom}(\mathcal{A}_{TSO}) \times \mathbf{Ls}$.

Expressions. Given an expression e,

$$\mathcal{P}_L(e) \subseteq \mathrm{Pom}(\mathcal{A}_{TSO}) \times \mathbf{V} \times \mathbf{Ls}.$$

Commands. Given a command c, $\mathcal{P}_L(c) \subseteq \mathrm{Pom}(\mathcal{A}_{TSO}) \times \mathbf{Ls}$.

Pomsets for Commands

$\mathcal{P}_L(c)$ is recursively defined on the structure of c.

- If $(P_1, v, B_1) \in \mathcal{P}_L(e)$ and $(F_2, B_2) \in \mathsf{split}(B_1; \{x := v\})$, then $(P_1; \{\bar{x} := v\}; F_2, B_2) \in \mathcal{P}_L(x := e)$.
- If $(P_1, B_1) \in \mathcal{P}_L(c_1)$ and $(P_2, B_2) \in \mathcal{P}_{B_1}(c_2)$, then $(P_1; P_2, B_2) \in \mathcal{P}_L(c_1; c_2)$.
- If $(P_i, [\,]) \in \mathcal{P}_{[\,]}(c_i)$ for $i = 1, 2$, then $(L; (P_1 \parallel P_2), [\,]) \in \mathcal{P}_L(c_1 \parallel c_2)$.

Semantic clauses for conditionals, loops, etc., can be found in the paper.

Pomsets for Dekker

Let c be $(x{:=}1; \textbf{if } y{=}0 \textbf{ then } z{:=}1) \parallel (y{:=}1; \textbf{if } x{=}0 \textbf{ then } w{:=}1)$.
The pair $(P, [\,])$ can be found in $\mathcal{P}_{[\,]}(c)$ for each P below and each choice $u, v \in \mathbf{V}$.

$$
\begin{tikzcd}[row sep=small]
\bar{x} := 1 \arrow[d] & \bar{y} := 1 \arrow[d] \\
y = 0 \arrow[d] & x = 0 \arrow[d] \\
\bar{z} := 1 \arrow[d] & \bar{w} := 1 \arrow[d] \\
x := 1 \arrow[d] & y := 1 \arrow[d] \\
z := 1 & w := 1
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}[row sep=small]
\bar{x} := 1 \arrow[d] & \bar{y} := 1 \arrow[d] \\
x := 1 \arrow[d] & y := 1 \arrow[d] \\
y = v & x = u
\end{tikzcd}
$$

Executing Pomsets

$$P \mapsto \{(\sigma, \tau)\}$$

Buffered States

Definition
A **buffered state** is an element of

$$\Sigma = (\mathbf{BLoc} \rightharpoonup_{fin} (\mathbf{V} \times \mathbb{N})/{\approx}) \times (\mathbf{Loc} \rightharpoonup_{fin} \mathbf{V}),$$

where $\approx$ is the least equivalence relation on $\mathbf{V} \times \mathbb{N}$ generated by $(v, 0) \approx (v', 0)$ for all $v, v' \in \mathbf{V}$.

We let σ, τ range over states, and we identify Σ with its inclusion in $(\mathbf{BLoc} \cup \mathbf{Loc}) \rightharpoonup_{fin} (\mathbf{V} \cup (\mathbf{V} \times \mathbb{N})/{\approx})$.

Write v_n for the equivalence class of (v, n) under $\approx$.

Footprints

Definition
Given an action λ, its **TSO footprint** $[\![\lambda]\!] \subseteq \Sigma \times \Sigma$ is:

$$
\begin{aligned}
[\![\bar{x} := v]\!] &= \{([\bar{x} : v'_n], [\bar{x} : v_{n+1}]) \mid v' \in V \wedge n \in \mathbb{N}\} \\
[\![x := v]\!] &= \{([x : v', \bar{x} : v''_{n+1}], [x : v, \bar{x} : v''_n]) \mid v', v'' \in V \wedge n \in \mathbb{N}\} \\
[\![x = v]\!] &= \{([x : v, \bar{x} : v'_0], [\,]), ([\bar{x} : v'_{n+1}], [\,]) \mid v' \in V \wedge n \in \mathbb{N}\} \\
[\![\delta]\!] &= \{([\,], [\,])\}
\end{aligned}
$$

Sequencing Footprints

Definition
Sequencing two sets $S_1, S_2 \subseteq \Sigma \times \Sigma$ is the associative operation

$$S_1 \lhd S_2 = \{(\sigma_1 \cup \sigma_2{\restriction}_{\mathrm{dom}(\sigma_2) \setminus \mathrm{dom}(\tau_1)}, [\tau_1 \mid \tau_2]) \mid (\sigma_i, \tau_i) \in S_i, [\sigma_1 \mid \tau_1] \Uparrow \sigma_2\},$$

where $\sigma \Uparrow \tau$ if $\sigma(x) = \tau(x)$ for all $x \in \mathrm{dom}(\sigma) \cap \mathrm{dom}(\tau)$,
and $[\sigma \mid \tau](x) = \begin{cases} \tau(x) & \text{if } x \in \mathrm{dom}(\tau) \\ \sigma(x) & \text{if } x \in \mathrm{dom}(\sigma) \setminus \mathrm{dom}(\tau). \end{cases}$

“Remote” Global Writes

Definition
The footprint of a sequence of remote global writes $L \in \mathbf{Ls}$ is inductively defined as

$$\llbracket [\,] \rrbracket^* = \{([], [])\}$$
$$\llbracket x := v :: L \rrbracket^* = \{([x : v'], [x : v]) \mid v' \in V\} \triangleleft \llbracket L \rrbracket^*$$

Note that remote global writes don't affect buffers! Contrast this with the footprint of a global write action $\llbracket x := v \rrbracket = \{([x : v', \bar{x} : v''_{n+1}], [x : v, \bar{x} : v''_n]) \mid v', v'' \in V \wedge n \in \mathbb{N}\}$.

Pomset Footprints

Definition
A **remote global-write environment** for a TSO pomset P is a $\Lambda \in \{\mathsf{Lin}(P \parallel L) \mid L \in \mathbf{Ls}\}$.

Definition
The **footprint** $[\![P]\!]_\Lambda$ of a pomset P in the presence of Λ is inductively defined by three rules: (Act) for $P = \{\lambda\}$, (Seq) for $P = P_1; P_2$, and (Par) for $P = P_1 \parallel P_2$.

Pomset Footprints, (Act) and (Seq)

(Act) If $P = \{\lambda\}$ for some action λ, and $\Lambda = \Lambda_1; P; \Lambda_2$ for some $\Lambda_i \in \mathbf{Ls}$, then $[\![P]\!]_\Lambda = [\![\Lambda_1]\!]^* \triangleleft [\![\lambda]\!] \triangleleft [\![\Lambda_2]\!]^*$.

(Seq) If $P = P_1; P_2$ and $\Lambda = \Lambda_1; \Lambda_2$, then $[\![P_1]\!]_{\Lambda_1} \triangleleft [\![P_2]\!]_{\Lambda_2} \subseteq [\![P]\!]_\Lambda$.

Executions

Definition
The set of **executions** of a TSO pomset P is

$$\mathcal{E}(P) = \{(\sigma, [\sigma \mid \tau]) \mid \exists \Lambda \in \mathsf{Lin}(P).(\sigma', \tau) \in [\![P]\!]_\Lambda, \sigma' \subseteq \sigma\}.$$

Sanity Check

Does our semantics truly capture TSO?

Key Observations

Theorem
Every $<_T$ TSO-consistent with a program order P is contained in a total order $\sqsubset$ TSO-consistent with P.

Soundness

We can define sets program order pomsets $\mathcal{P}_{PO}(c) \subseteq \mathrm{Pom}(\mathcal{A}_{PO})$ for a command c in a way analogous to TSO pomsets.

Definition
A function $f : \mathrm{Pom}(\mathcal{A}_{PO}) \to \wp(\mathcal{A}_{PO}$ list) is **sound** when for every program p and finite pomset $P \in \mathcal{P}_{PO}(p)$, if $L \in f(P)$, then L is TSO-consistent with P.

Soundness (continued)

We can define a function $U : \mathrm{Pom}(\mathcal{A}_{TSO}) \to \mathrm{Pom}(\mathcal{A}_{PO})$ taking every TSO pomset to its underlying program order pomset.

Let

$$\mathcal{T}(P) = \bigcup_{P' \in U^{-1}(P)} \{\Lambda\restriction_{\mathcal{A}_{PO}} \mid \Lambda \in \mathsf{Lin}(P') \land [\![P']\!]_\Lambda \neq \varnothing\}.$$

Informally, $\mathcal{T}(P)$ captures the linearisations of TSO pomsets in $U^{-1}(P)$ that give rise to TSO executions.

Theorem
The function $\mathcal{T}$ is sound.

Completeness

Definition
A function $f : \mathrm{Pom}(\mathcal{A}_{PO}) \to \wp(\mathcal{A}_{PO}\ \text{list})$ is **complete** when for every program p and finite pomset $P \in \mathcal{P}_{PO}(p)$, if L is TSO-consistent with P, then $L \in f(P)$.

Theorem
The function $\mathcal{T}$ is complete.

The Takeaway

Our work provides:

1. an **axiomatisation** of SPARC TSO;
2. a **compositional denotational semantics** for SPARC TSO;
3. a notion of **pomset execution** using **buffered states**; and
4. a **precise correspondence** between the axiomatic and denotational accounts.

(Act) example

(Act) If $P = \{\lambda\}$ for some action λ, and $\Lambda = \Lambda_1; P; \Lambda_2$ for some $\Lambda_i \in \mathbf{Ls}$, then $[\![P]\!]_\Lambda = [\![\Lambda_1]\!]^* \triangleleft [\![\lambda]\!] \triangleleft [\![\Lambda_2]\!]^*$.

Example. $\Lambda = [x := 3, \bar{x} := 2]$, $P = \{\bar{x} := 2\}$.

$$\begin{aligned}
[\![P]\!]_\Lambda &= [\![[x := 3]]\!]^* \triangleleft [\![\bar{x} := 2]\!] \triangleleft [\![[\ ]]\!]^* \\
&= [\![[x := 3]]\!]^* \triangleleft [\![\bar{x} := 2]\!] \triangleleft \{([\ ], [\ ])\} \\
&= \{([x : v], [x : 3]) \mid v \in \mathbf{V}\} \triangleleft \{([\bar{x} : v_n], [\bar{x} : 2_{n+1}]) \mid v \in \mathbf{V}, n \in \mathbb{N}\} \\
&= \{([x : v, \bar{x} : v'_n], [x : 3, \bar{x} : 2_{n+1}]) \mid v, v' \in \mathbf{V}, n \in \mathbb{N}\}.
\end{aligned}$$

Pomset Footprints, (Par)

Let $\zeta(\sigma)$ if and only if for all $x \in \mathrm{dom}(\sigma\restriction_{\mathbf{BLoc}})$, $\sigma(x) = v_0$.

(Par) If $P = P_1 \parallel P_2$, Λ_1 is the result of deleting the read and buffer write actions of P_2 from Λ, Λ_2 is the symmetric restriction, $(\sigma_i, \tau_i) \in [\![P_i]\!]_{\Lambda_i}$, $\zeta(\sigma_i)$, $\zeta(\tau_i)$ $(i = 1, 2)$, and $\sigma_1 \Uparrow \sigma_2$, then $(\sigma_1 \cup \sigma_2, \tau_1 \cup \tau_2) \in [\![P]\!]_{\Lambda}$.

Reminder: $\sigma \Uparrow \tau$ iff for all $x \in \mathrm{dom}(\sigma) \cap \mathrm{dom}(\tau)$, $\sigma(x) = \tau(x)$.