A Denotational Semantics for SPARC TSO

Ryan Kavanagh  
Stephen Brookes  
MFPS XXXIII  

Carnegie Mellon University  

Funded in part by an NSERC Postgraduate Fellowship
The Dekker Algorithm

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

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

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

can reach a final state $\tau$ with $z = 1$ and $w = 1$. 
The Dekker algorithm implicitly assumes **sequential consistency**, where

\[ \text{the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.} \]

— Leslie Lamport, 1979
SPARC TSO

Process 1

FIFO Write Buffer

Reads

FIFO Write Buffer

Reads

Switch

Shared Memory
The Problem

How can we give a compositional semantics that exactly captures the behaviour of the SPARC TSO memory model?
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
- $\Phi : P \rightarrow L$ is a labelling function.

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

We usually write $P$ instead of $\langle P, <, \Phi \rangle$, and let $<_P$ and $\Phi_P$ denote the order and labelling function.
We can draw pomsets as labelled directed acyclic graphs. We draw \( a \rightarrow b \) 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 \( \text{Loc} \) 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 \( x \in \text{Loc} \) and \( v \in \mathbf{V} \).

**Write actions** are \( x := v \) for \( x \in \text{Loc} \) and \( v \in \mathbf{V} \).

The **skip action** is \( \delta \).

These collectively form the set \( A_{PO} \) of **program order actions**, ranged over by \( \lambda \).
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 $A_{PO}$ 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 $\sigma : \text{Loc} \to_{\text{fin}} \text{V}$ 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 $\lambda_2 \xrightarrow{\lambda_1} \lambda_3$ in $<_P$, then $\lambda_2 \xrightarrow{\lambda_1} \lambda_3$ in $<_T$. 
TSO (Denotationally)
Three Ingredients

1. TSO pomsets
2. Pomset executions
3. Soundness and completeness
We restrict our attention to the following simple imperative language:

\[ v ::= \ldots, -2, -1, 0, 1, 2, \ldots \]

\[ e ::= v \mid x \mid e_1 + e_2 \mid e_1 \ast e_2 \mid \cdots \]

\[ b ::= \text{true} \mid \text{false} \mid \neg b \mid e_1 = e_2 \mid e_1 < e_2 \mid b_1 \lor b_2 \mid b_1 \land b_2 \mid \cdots \]

\[ c ::= \text{skip} \mid x := e \mid c_1 ; c_2 \mid c_1 || c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c \]

\[ p ::= c \]
**Goal**
To assign to each expression a set of “TSO pomsets” denoting its possible TSO executions.
Introduce a new set $B\text{Loc} = \{\overline{x} \mid x \in \text{Loc}\}$ of buffer locations.

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

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 $\text{Ls}$ be the set of all write buffers.
Definition

A TSO pomset is a pomset over the set of labels
\[ A_{TSO} = \{ \delta, x := \nu, \bar{x} := \nu, x = \nu \mid x \in \text{Loc}, \nu \in V \} \]
satisfying the finite height property.
The parallel composition $P_1 \parallel P_2$ of pomsets $P_1$ and $P_2$ is

$P_1 \cup P_2$.

**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 $p <; q$, and $\Phi(i, p) = \Phi_i(p)$. 
Combining Pomsets

The **sequential composition** $P_1; P_2$ of pomsets $P_1$ and $P_2$ is

if $P_1$ is finite, and it is $P_1$ if $P_1$ is infinite.

**Definition (Formally)**
The sequential composition $\langle P_0, \langle 0, \Phi_0 \rangle; \langle P_1, \langle 1, \Phi_1 \rangle \rangle$ 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, $\langle P_0, \langle 0, \Phi_0 \rangle; \langle P_1, \langle 1, \Phi_1 \rangle \rangle$ is $\langle P_0, \langle 0, \Phi_0 \rangle \rangle$. 

16
Generating Sets of Pomsets

Three types of semantic clauses, given a write buffer \( L \in Ls \):

**Buffer Flushes.** \( \text{split}(L) \subseteq \text{Pom} \left( A_{TSO} \right) \times Ls \).

**Expressions.** Given an expression \( e \),

\[ \mathcal{P}_L(e) \subseteq \text{Pom} \left( A_{TSO} \right) \times V \times Ls \]

**Commands.** Given a command \( c \), \( \mathcal{P}_L(c) \subseteq \text{Pom} \left( A_{TSO} \right) \times Ls \).
\( P_L(c) \) is recursively defined on the structure of \( c \).

- If \((P_1, v, B_1) \in P_L(e)\) and \((F_2, B_2) \in \text{split}(B_1; \{x := v\})\), then \((P_1; \{\bar{x} := v\}; F_2, B_2) \in P_L(x := e)\).

- If \((P_1, B_1) \in P_L(c_1)\) and \((P_2, B_2) \in P_{B_1}(c_2)\), then \((P_1; P_2, B_2) \in P_L(c_1; c_2)\).

- If \((P_i, [ ]) \in P_{[ ]}(c_i)\) for \(i = 1, 2\), then \((L; (P_1 \parallel P_2), [ ]) \in P_L(c_1 \parallel c_2)\).

Semantic clauses for conditionals, loops, etc., can be found in the paper.
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 $u, v \in V$. 

\[
\begin{array}{ccc}
\bar{x} := 1 & \bar{y} := 1 \\
\downarrow & \downarrow \\
y = 0 & x = 0 \\
\downarrow & \downarrow \\
\bar{z} := 1 & \bar{w} := 1 \\
\downarrow & \downarrow \\
x := 1 & y := 1 \\
\downarrow & \downarrow \\
z := 1 & w := 1
\end{array}
\]

\[
\begin{array}{ccc}
\bar{x} := 1 & \bar{y} := 1 \\
\downarrow & \downarrow \\
x := 1 & y := 1 \\
\downarrow & \downarrow \\
y = v & x = u
\end{array}
\]
Executing Pomsets

\[ P \mapsto \{ (\sigma, \tau) \} \]
Buffered States

Definition
A buffered state is an element of

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

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

We let $\sigma, \tau$ range over states, and we identify $\Sigma$ with its inclusion in $(BLoc \cup Loc) \xrightarrow{fin} (V \cup (V \times \mathbb{N})/\approx)$.

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

**Definition**
Given an action \( \lambda \), its **TSO footprint** \([\lambda] \subseteq \Sigma \times \Sigma\) is:

\[
\begin{align*}
[x := v] &= \{ ([x : v'], [\bar{x} : v_{n+1}]) \mid v' \in V \land n \in \mathbb{N} \} \\
[x := v] &= \{ ([x : v', \bar{x} : v''_{n+1}], [x : v, \bar{x} : v'_{n}]) \mid v', v'' \in V \lor n \in \mathbb{N} \} \\
[x = v] &= \{ ([x : v, \bar{x} : v'_{n}], [], ([\bar{x} : v'_{n+1}], [])) \mid v' \in V \lor n \in \mathbb{N} \} \\
[\delta] &= \{ ([], []) \}
\end{align*}
\]
Sequencing Footprints

Definition

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

$$S_1 \triangleleft S_2 = \{(\sigma_1 \cup \sigma_2 \upharpoonright_{\text{dom}(\sigma_2) \setminus \text{dom}(\tau_1)}, [\tau_1 \| \tau_2]) \mid (\sigma_i, \tau_i) \in S_i, [\sigma_1 \| \tau_1] \uparrow \sigma_2\},$$

where $\sigma \uparrow \tau$ if $\sigma(x) = \tau(x)$ for all $x \in \text{dom}(\sigma) \cap \text{dom}(\tau)$,

and $[\sigma \| \tau](x) = \begin{cases} 
\tau(x) & \text{if } x \in \text{dom}(\tau) \\
\sigma(x) & \text{if } x \in \text{dom}(\sigma) \setminus \text{dom}(\tau). 
\end{cases}$
"Remote" Global Writes

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

\[
[[\mathbf{[]}}]^{*} = \{([], [])\}
\]

\[
[[x := v :: L]]^{*} = \{([x : v'], [x : v]) | v' \in V\} \triangleleft [[L]]^{*}
\]

Note that remote global writes don’t affect buffers! Contrast this with the footprint of a global write action

\[
[[x := v]] = \{([x : v', \bar{x} : v''_{n+1}], [x : v, \bar{x} : v''_{n}]) | v', v'' \in V \land n \in \mathbb{N}\}.
\]
Definition
A remote global-write environment for a TSO pomset $P$ is a
$\Lambda \in \{\text{Lin}(P \parallel L) \mid L \in Ls\}$.

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

$$\mathcal{E}(P) = \{ (\sigma, [\sigma \mid \tau]) \mid \exists \Lambda \in \text{Lin}(P). (\sigma', \tau) \in [P]_\Lambda, \sigma' \subseteq \sigma \}.$$
Does our semantics truly capture TSO?
Key Observations

**Theorem**
Every $<_T$ TSO-consistent with a program order $P$ is contained in a total order $\sqsubseteq$ TSO-consistent with $P$. 
We can define sets program order pomsets $\mathcal{P}_{PO}(c) \subseteq \text{Pom}(A_{PO})$ for a command $c$ in a way analogous to TSO pomsets.

**Definition**

A function $f : \text{Pom}(A_{PO}) \rightarrow \wp(A_{PO} \text{ list})$ 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 $P$. 
We can define a function $U : \text{Pom}(A_{TSO}) \rightarrow \text{Pom}(A_{PO})$ taking every TSO pomset to its underlying program order pomset.

Let

$$\mathcal{T}(P) = \bigcup_{P' \in U^{-1}(P)} \{\Lambda | A_{PO} | \Lambda \in \text{Lin}(P') \land \lfloor P' \rfloor_\Lambda \neq \emptyset \}.$$ 

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.*
Definition
A function $f : \text{Pom}(A_{PO}) \to \wp(\text{PO list})$ 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 $T$ 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) If $P = \{\lambda\}$ for some action $\lambda$, and $\Lambda = \Lambda_1; P; \Lambda_2$ for some $\Lambda_i \in Ls$, then $\llbracket P \rrbracket_\Lambda = \llbracket \Lambda_1 \rrbracket^* \llbracket \lambda \rrbracket \llbracket \Lambda_2 \rrbracket^*$.

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

$$
\llbracket P \rrbracket_\Lambda = \llbracket \llbracket x := 3 \rrbracket \rrbracket^* \llbracket \overline{x} := 2 \rrbracket \llbracket \llbracket \rrbracket \rrbracket^*
$$

$$=
\llbracket \llbracket x := 3 \rrbracket \rrbracket^* \llbracket \overline{x} := 2 \rrbracket \llbracket \llbracket \rrbracket \rrbracket
$$

$$=
\llbracket \llbracket x := 3 \rrbracket \rrbracket^* \llbracket \overline{x} := 2 \rrbracket \llbracket \llbracket \rrbracket \rrbracket
$$

$$=
\{([x : v], [x : 3]) | v \in V \} \llbracket \{([\overline{x} : v_n], [\overline{x} : 2_{n+1}]) | v \in V, n \in \mathbb{N} \}
$$

$$=
\{([x : v, \overline{x} : v'_n], [x : 3, \overline{x} : 2_{n+1}]) | v, v' \in V, n \in \mathbb{N} \}.\]
Let $\zeta(\sigma)$ if and only if for all $x \in \text{dom}(\sigma|_{\text{BLoc}})$, $\sigma(x) = v_0$.

(Par) If $P = P_1 \parallel P_2$, $\Lambda_1$ is the result of deleting the read and buffer write actions of $P_2$ from $\Lambda$, $\Lambda_2$ is the symmetric restriction, $(\sigma_i, \tau_i) \in \llbracket P_i \rrbracket_{\Lambda_i}$, $\zeta(\sigma_i)$, $\zeta(\tau_i)$ ($i = 1, 2$), and $\sigma_1 \upharpoonright \sigma_2$, then $(\sigma_1 \cup \sigma_2, \tau_1 \cup \tau_2) \in \llbracket P \rrbracket_{\Lambda}$.

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