Correspondence between operational and declarative concurrency semantics

Ori Lahav    Viktor Vafeiadis

29 August 2017
Two alternative definitions of SC

Definition (Operational SC)
An outcome $O$ is allowed for a program $P$ under SC if there exists $M$ such that $P, S_0, M_0 \Rightarrow^* \text{skip} \parallel \ldots \parallel \text{skip}, O, M$.

Definition (Declarative SC)
An outcome $O$ is allowed for a program $P$ under SC if there exists an SC-consistent execution graph of $P$ with outcome $O$.

How do we show that the two definitions are equivalent?
Operational version of SC declarative semantics

State:

\[ \langle P, S, G, sc \rangle \in \text{Program} \times (\text{Tid} \rightarrow \text{Store}) \times \text{ExecutionGraph} \times \mathcal{P}(\text{Event} \times \text{Event}) \]

- Initial stores: \( S_0 \triangleq \lambda i. s_0 \)
- Initial execution: \( G_0 \) consisting only of the initialization events
- Initial \( sc \)-relation: \( sc_0 \) is an arbitrary total order on \( G_0.E \)
- Initial \( sc \)-relation: \( sc_0 \) is an arbitrary total order on \( G_0.E \)

\[
\begin{align*}
\text{NON-SILENT} \\
P, S \xrightarrow{\text{tid}(a) : \text{lab}(a)} P', S' \\
G' \in \text{Add}(G, a) \\
sc' = sc \cup (G.E \times \{a\}) \\
G' \text{ is SC-consistent wrt } sc'
\end{align*}
\]

\[
\begin{align*}
\text{SILENT} \\
P, S \xrightarrow{i : \varepsilon} P', S' \\
P, S, G, sc \Rightarrow P', S', G, sc
\end{align*}
\]

where \( \text{Add}(G, a) \) is the set of all complete graphs \( G' \) satisfying:

- \( G'.E = G.E \cup \{a\} \)
- \( G'.po = G.po \cup ((E_0 \cup G.E^{\text{tid}(a)}) \times \{a\}) \)
- \( G.rf \subseteq G'.rf \)
Operational version of SC declarative semantics

**Definition (Operational-declarative SC)**

An outcome $O$ is allowed for a program $P$ under SC if there exist $G, sc$ such that $P, S_0, G_0, sc_0 \Rightarrow^* \text{skip} \parallel \ldots \parallel \text{skip}, O, G, sc$.

Establish correspondence between operational SC and declarative SC in two steps:

1. operational SC $=$ intermediate SC
2. declarative SC $=$ intermediate SC
We will use forward weak *simulation*. Consider two labeled state transition systems $M_1 = \langle Q_1, q_1^0, \rightarrow_1 \rangle$ and $M_2 = \langle Q_2, q_2^0, \rightarrow_2 \rangle$.

- $\mathcal{R} \subseteq Q_1 \times Q_2$ is a *simulation relation* from $M_1$ to $M_2$ if:
  - $q_1^0 \mathcal{R} q_2^0$, and
  - whenever $q_1 \mathcal{R} q_2$ and $q_1 \rightarrow_1 q'_1$, then there exists some $q'_2 \in Q_2$ such that $q_2 \rightarrow_2^* q'_2$ and $q'_1 \mathcal{R} q'_2$.

- $\mathcal{R} \subseteq Q_1 \times Q_2$ is called a *bisimulation relation* if it is a simulation relation from $M_1$ to $M_2$ and $\mathcal{R}^{-1}$ is a simulation relation from $M_2$ to $M_1$.

**Lemma**

*If a simulation relation exists then for every state $q_1 \in Q_1$ that is reachable from $q_1^0$ in $M_1$, there exists some $q_2 \in Q_2$ that is reachable from $q_2^0$ in $M_2$ and satisfies $q_1 \mathcal{R} q_2$.***
Our bisimulation relation:
\[ \langle P, S, M \rangle \sim \langle P', S', G, \text{sc} \rangle \text{ if the following hold:} \]
- \( P = P' \)
- \( S = S' \)
- \( M = \lambda x. \text{val}_w(\max_{\text{sc}} G.W_x) \)
- \( G \) is complete and SC-consistent wrt \( \text{sc} \).

- Show that \( \sim \) is a bisimulation relation.
- Deduce that operational SC and intermediate SC have the same outcomes for any given program.
Declarative $SC = \text{intermediate } SC$

Two directions:

$\subseteq$ Every outcome allowed for $P$ according to declarative $SC$ is allowed according to intermediate $SC$

$\supseteq$ Every outcome allowed for $P$ according to intermediate $SC$ is allowed according to declarative $SC$

Reminders:

**Definition**

$G$ is an execution graph of a program $P$ with an outcome $O$ if $G^i$ is an execution of $P(i)$ with final store $O(i)$ for every $i \in \text{Tid}$.

**Definition (Declarative SC)**

An outcome $O$ is allowed for a program $P$ under SC if there exists an SC-consistent execution graph of $P$ with outcome $O$. 

Lemma (Execution generation)

Let $G$ be an execution of a program $P_0$ with outcome $O$. Let $a_1, \ldots, a_n$ be an enumeration of $G.E \setminus E_0$ that respects $G.po$. Then, there exist $\langle P_1, S_1 \rangle, \ldots, \langle P_n, S_n \rangle$ such that:

- $P_n = \text{skip} \parallel \ldots \parallel \text{skip}$ and $S_n = O$
- For every $1 \leq j \leq n$, we have:

$$
P_{j-1}, S_{j-1} \xrightarrow{\text{tid}(a_j) : \varepsilon}^* \xrightarrow{\text{tid}(a_j) : \text{lab}(a_j)} \xrightarrow{\text{tid}(a_j) : \varepsilon}^* P_j, S_j
$$
Let $G$ be an SC-consistent execution graph of $P_0$ with outcome $O$.
Let $sc$ be a total order on $G.E$ such that $G$ is SC-consistent wrt $sc$.
We show that $P, S_0, G_0, sc_0 \Rightarrow^* \text{skip} \parallel \ldots \parallel \text{skip}, O, G, sc$.
Let $a_1, \ldots, a_n$ be an enumeration of $G.E \setminus E_0$ following $sc$.
Since $G.po \subseteq sc$, by the previous lemma, there exist $\langle P_1, S_1 \rangle, \ldots, \langle P_n, S_n \rangle$ such that:
- $P_n = \text{skip} \parallel \ldots \parallel \text{skip}$ and $S_n = O$
- For every $1 \leq j \leq n$, we have:
  $$P_{j-1}, S_{j-1} \xrightarrow{\text{tid}(a_j):\varepsilon}^* \xrightarrow{\text{tid}(a_j):\text{lab}(a_j)} \xrightarrow{\text{tid}(a_j):\varepsilon}^* P_j, S_j$$
- For every $0 \leq j \leq n$, let
  - $G_j$ - the restriction of $G$ to $E_0 \cup \{a_1, \ldots, a_j\}$
  - $sc_j$ - the restriction of $sc$ to $E_0 \cup \{a_1, \ldots, a_j\}$
- Then, for every $1 \leq j \leq n$, we have:
  $$P_{j-1}, S_{j-1}, G_{j-1}, sc_{j-1} \Rightarrow^* P_j, S_j, G_j, sc_j$$
Operational-declarative SC ⊆ declarative SC

- Suppose that $P_0, S_0, G_0, sc_0 \Rightarrow^* skip \parallel \ldots \parallel skip, O, G, sc$.
- By definition, $G$ is SC-consistent. It remains to show that each $G^i$ is an execution of $P_0(i)$ with final store $O(i)$.
- We know: $P_0, S_0, G_0, sc_0 \Rightarrow P_1, S_1, G_1, sc_1 \Rightarrow \ldots \Rightarrow P_n, S_n, G_n, sc_n$ where $P_n, S_n, G_n, sc_n = skip \parallel \ldots \parallel skip, O, G, sc$.
- The sequence above induces the following sequence of transitions:

\[
P_0, S_0 \xrightarrow{i_1:l_1} P_1, S_1 \xrightarrow{i_2:l_2} P_2, S_2 \xrightarrow{i_3:l_3} \ldots \xrightarrow{i_n:l_n} P_n, S_n
\]

- In turn, by filtering only the transitions of thread $i$ we obtain:

\[
P_0(i), S_0(i) \xrightarrow{l_{k_1}} P_{k_1}(i), S_{k_1}(i) \xrightarrow{l_{k_2}} \ldots \xrightarrow{l_{k_{n_i}}} P_{k_{n_i}}(i), S_{k_{n_i}}(i) = skip, O(i)
\]

- It follows that $P_0(i), s_0, G_\emptyset \Rightarrow^* skip, O(i), G^i$, and so $G^i$ is an execution of $P_0(i)$ with final store $O(i)$. 
Operational semantics for COH
Recall the following litmus tests:

<table>
<thead>
<tr>
<th>Store buffering</th>
<th>Coherence test</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x = y = 0$</td>
<td>$x = 0$</td>
</tr>
<tr>
<td>$x := 1$</td>
<td>$x := 1$</td>
</tr>
<tr>
<td>$y := 1$</td>
<td>$x := 2$</td>
</tr>
<tr>
<td>$a := y$</td>
<td>$a := x$</td>
</tr>
<tr>
<td>$b := x$</td>
<td>$b := x$</td>
</tr>
<tr>
<td>$// 0$</td>
<td>$// 2$</td>
</tr>
<tr>
<td>$// 0$</td>
<td>$// 1$</td>
</tr>
</tbody>
</table>

Two approaches:

- Out-of-order execution with SC memory.
- In-order execution with non-standard memory:
  - Allow threads to observe different subsets of writes.
  - Use timestamps to order writes to the same location.
Operational semantics for coherence

**Store buffering**

\[
\begin{align*}
x &= y = 0 \\
x &:= 1; \quad \parallel \quad y := 1; \\
a &:= y; \quad \parallel \quad b := x; \quad \parallel \quad 0
\end{align*}
\]
Operational semantics for coherence

### Store buffering

\[
x = y = 0
\]

\[
\begin{align*}
\blacktriangleright & \quad x := 1; \\
\blacktriangleright & \quad a := y; \quad // 0 \\
\blacktriangleright & \quad y := 1; \\
\blacktriangleright & \quad b := x; \quad // 0
\end{align*}
\]

#### Memory

\[
\begin{array}{c}
\langle x : 0@0 \rangle \\
\langle y : 0@0 \rangle
\end{array}
\]

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$'s view</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_2$'s view</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>


- Global memory is a pool of messages of the form

\[
\langle location : value @ timestamp \rangle
\]

- Each thread maintains a **thread-local view** recording the last observed timestamp for every location
Global memory is a pool of messages of the form

\[ \langle \text{location : value} @ \text{timestamp} \rangle \]

Each thread maintains a \textit{thread-local view} recording the last observed timestamp for every location
Global memory is a pool of messages of the form

$$\langle \text{location} : \text{value} @ \text{timestamp} \rangle$$

Each thread maintains a *thread-local view* recording the last observed timestamp for every location.
Global memory is a pool of messages of the form

\[ \langle \text{location} : \text{value} \circledast \text{timestamp} \rangle \]

Each thread maintains a \textit{thread-local view} recording the last observed timestamp for every location.
Operational semantics for coherence

Store buffering

\[ \begin{align*}
&x = y = 0 \\
x &:= 1; \\
a &:= y; \quad // 0 \\
y &:= 1; \\
b &:= x; \quad // 0
\end{align*} \]

Memory

\[ \begin{array}{c@{\quad}c@{\quad}c}
\langle x : 0 @ 0 \rangle & \langle y : 0 @ 0 \rangle & \\
\langle x : 1 @ 1 \rangle & \langle y : 1 @ 1 \rangle
\end{array} \]

\[ \begin{array}{c|c|c|c|c|c|c}
\hline
\multicolumn{7}{c}{T_1's view} \\
\hline
x & y & \\
\hline
0 & 0 & \\
\hline
0 & x & 1
\end{array} \]

\[ \begin{array}{c|c|c|c|c|c|c}
\hline
\multicolumn{7}{c}{T_2's view} \\
\hline
x & y & \\
\hline
0 & 0 & \\
\hline
0 & x & 1
\end{array} \]

- Global memory is a pool of messages of the form
  \[ \langle \text{location} : \text{value} @ \text{timestamp} \rangle \]

- Each thread maintains a *thread-local view* recording the last observed timestamp for every location
Operational semantics for coherence

**Store buffering**

\[
x = y = 0 \\
x := 1; \\
a := y; \quad // 0 \\
\]

\[
\begin{array}{c}
x := 1; \\
a := y; \quad // 0 \\
\end{array}
\]

**Memory**

\[
\begin{array}{c|c}
T_1's \ view \\
x & y \\
\hline
\langle x:0@0 \rangle & 0 \\
\langle y:0@0 \rangle & 0 \\
\langle x:1@1 \rangle & 1 \\
\langle y:1@1 \rangle & 1 \\
\end{array}
\]

\[
\begin{array}{c|c}
T_2's \ view \\
x & y \\
\hline
0 & \times \\
1 & \times \\
\end{array}
\]

**Coherence test**

\[
x = 0 \\
\]

\[
\begin{array}{c}
x := 1; \\
a := x; \quad // 2 \\
\end{array}
\]

\[
\begin{array}{c}
x := 2; \\
b := x; \quad // 1 \\
\end{array}
\]
Operational semantics for coherence

**Store buffering**

\[ x = y = 0 \]
\[ x := 1; \] \hspace{1cm} \[ y := 1; \]
\[ a := y; \quad \text{// 0} \] \hspace{1cm} \[ b := x; \quad \text{// 0} \]

**Memory**

\[ \langle x : 0 @ 0 \rangle \]
\[ \langle y : 0 @ 0 \rangle \]
\[ \langle x : 1 @ 1 \rangle \]
\[ \langle y : 1 @ 1 \rangle \]

**T\(_1\)’s view**

\[
\begin{array}{cc}
  x & y \\
  0 & 0 \\
  1 & 1
\end{array}
\]

**T\(_2\)’s view**

\[
\begin{array}{cc}
  x & y \\
  0 & 0 \\
  1 & 1
\end{array}
\]

**Coherence test**

\[ x = 0 \]
\[ x := 1; \] \hspace{1cm} \[ x := 2; \]
\[ a := x; \quad \text{// 2} \] \hspace{1cm} \[ b := x; \quad \text{// 1} \]

**Memory**

\[ \langle x : 0 @ 0 \rangle \]

**T\(_1\)’s view**

\[
\begin{array}{c}
  x \\
  0 \\
  0
\end{array}
\]

**T\(_2\)’s view**

\[
\begin{array}{c}
  x \\
  0 \\
  0
\end{array}
\]
Operational semantics for coherence

**Store buffering**

\[ x = y = 0 \]

\[ x := 1; \]
\[ a := y; \quad \text{// 0} \]
\[ y := 1; \]
\[ b := x; \quad \text{// 0} \]

**Memory**

\[ \langle x \!: 0@0 \rangle \]
\[ \langle y \!: 0@0 \rangle \]
\[ \langle x \!: 1@1 \rangle \]
\[ \langle y \!: 1@1 \rangle \]

\[ T_1 \text{'s view} \]
\[ x \quad y \]
\[ \begin{array}{c|c}
0 & 0 \\
\end{array} \]
\[ \begin{array}{c|c}
1 & 1 \\
\end{array} \]

\[ T_2 \text{'s view} \]
\[ x \quad y \]
\[ \begin{array}{c|c}
0 & 0 \\
\end{array} \]
\[ \begin{array}{c|c}
1 & 1 \\
\end{array} \]

**Coherence test**

\[ x = 0 \]

\[ x := 1; \]
\[ a := x; \quad \text{// 2} \]
\[ x := 2; \]
\[ b := x; \quad \text{// 1} \]

**Memory**

\[ \langle x \!: 0@0 \rangle \]
\[ \langle x \!: 1@1 \rangle \]

\[ T_1 \text{'s view} \]
\[ x \]
\[ \begin{array}{c|c}
0 & 0 \\
\end{array} \]
\[ \begin{array}{c|c}
1 & 1 \\
\end{array} \]

\[ T_2 \text{'s view} \]
\[ x \]
\[ \begin{array}{c|c}
0 & 0 \\
\end{array} \]
Operational semantics for coherence

**Store buffering**

\[ x = y = 0 \]
\[ x := 1; \]
\[ a := y; \quad \text{// 0} \]
\[ y := 1; \]
\[ b := x; \quad \text{// 0} \]

**Memory**

\[ T_1 \text{’s view} \]
\[ x \quad y \]
\[ \begin{array}{cc}
0 & 0 \\
1 & 1 
\end{array} \]

\[ T_2 \text{’s view} \]
\[ x \quad y \]
\[ \begin{array}{cc}
0 & x \\
1 & 1 
\end{array} \]

**Coherence test**

\[ x = 0 \]
\[ x := 1; \]
\[ a := x; \quad \text{// 2} \]
\[ x := 2; \]
\[ b := x; \quad \text{// 1} \]

**Memory**

\[ T_1 \text{’s view} \]
\[ x \quad x \]
\[ \begin{array}{cc}
0 & 0 \\
1 & x 
\end{array} \]

\[ T_2 \text{’s view} \]
\[ x \quad x \]
\[ \begin{array}{cc}
0 & 1 \\
2 & 2 
\end{array} \]
**Operational semantics for coherence**

### Store buffering

\[
x = y = 0
\]
\[
x := 1;\quad y := 1;\quad a := y;\quad b := x;
\]

\[
\langle x : 0 \rangle @0\quad \langle y : 0 \rangle @0\quad \langle x : 1 \rangle @1\quad \langle y : 1 \rangle @1
\]

\[
T_1's\ view\quad T_2's\ view
\]
\[
\begin{array}{c|c}
x & y \\
\hline
0 & 0 \\
1 & 1
\end{array}
\]

### Coherence test

\[
x = 0
\]
\[
x := 1;\quad x := 2;\quad a := x;\quad b := x;
\]

\[
\langle x : 0 \rangle @0\quad \langle x : 1 \rangle @1\quad \langle x : 2 \rangle @2
\]

\[
T_1's\ view\quad T_2's\ view
\]
\[
\begin{array}{c|c|c}
x & 0 & x \\
\hline
0 & 0 & 0 \\
1 & 1 & 1
\end{array}
\]

\[
\begin{array}{c|c}
x & 2 \\
\hline
x & 2
\end{array}
\]
Operational semantics for coherence

**Store buffering**

\[ x = y = 0 \]

\[ x := 1; \]
\[ a := y; \quad \text{// 0} \]
\[ y := 1; \]
\[ b := x; \quad \text{// 0} \]

**Memory**

\[ \langle x : 0 \rangle \]
\[ \langle y : 0 \rangle \]
\[ \langle x : 1 \rangle \]
\[ \langle y : 1 \rangle \]

**T1’s view**

\[ x \quad y \]
\[ 0 \quad 1 \]

**T2’s view**

\[ x \quad y \]
\[ 0 \quad 1 \]

---

**Coherence test**

\[ x = 0 \]

\[ x := 1; \]
\[ a := x; \quad \text{// 2} \]
\[ x := 2; \]
\[ b := x; \quad \text{// 1} \]

**Memory**

\[ \langle x : 0 \rangle \]
\[ \langle x : 1 \rangle \]
\[ \langle x : 2 \rangle \]

**T1’s view**

\[ x \]
\[ 0 \]
\[ 1 \]

**T2’s view**

\[ x \]
\[ 2 \]
Supporting write-write reordering

\begin{align*}
2+2W
\quad x &= y = 0 \\
\quad x &:= 1; \quad \| \quad y &:= 1; \\
\quad y &:= 2; \quad \| \quad x &:= 2; \\
\quad a &:= y \quad \| \quad b &:= x
\end{align*}

\textit{T}_1\textquote{’s view} \quad x \quad y \\
\textit{T}_2\textquote{’s view} \quad x \quad y

\textit{\textbf{\textit{\textbf{Wits choose timestamp}} greater than the thread’s view, not necessarily the globally greatest one.}}
Supporting write-write reordering

2+2W

\[ x = y = 0 \]

\[ x := 1; \quad y := 1; \]
\[ y := 2; \quad x := 2; \]
\[ a := y \quad // 1 \]
\[ b := x \quad // 1 \]

Memory

\[ \langle x : 0@0 \rangle \]
\[ \langle y : 0@0 \rangle \]

\[ \begin{array}{cc}
T_1's \ view & x & y \\
0 & 0 \\
\end{array} \]

\[ \begin{array}{cc}
T_2's \ view & x & y \\
0 & 0 \\
\end{array} \]

Writes choose timestamp greater than the thread's view, not necessarily the globally greatest one.
Supporting write-write reordering

2+2W

\[ x = y = 0 \]

\[
\begin{align*}
x &:= 1; \\
\triangleright y &:= 2; \\
a &:= y \quad \text{// 1} \quad \triangleright y &:= 1; \\
\end{align*}
\]

\[ x := 2; \quad b := x \quad \text{// 1} \]

\[
\begin{array}{|c|c|}
\hline
\text{Memory} & \langle x : 0@0 \rangle & \langle y : 0@0 \rangle & \langle x : 1@1 \rangle \\
\hline
\text{T}_1 \text{'s view} & \begin{pmatrix} x & y \end{pmatrix} & \begin{pmatrix} 0 & 0 \\
0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 \end{pmatrix} \\
\text{T}_2 \text{'s view} & \begin{pmatrix} x & y \end{pmatrix} & \begin{pmatrix} 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \end{pmatrix} \\
\hline
\end{array}
\]
Supporting write-write reordering

\[ 2+2W \]

\[ x = y = 0 \]
\[ x := 1; \quad y := 1; \]
\[ y := 2; \quad x := 2; \]
\[ a := y \quad // 1 \quad b := x \quad // 1 \]

Memory

\[ \langle x : 0@0 \rangle \]
\[ \langle y : 0@0 \rangle \]
\[ \langle x : 1@1 \rangle \]
\[ \langle y : 2@1 \rangle \]

\[ T_1 \text{'s view} \]

\[ \begin{array}{cc}
  x & y \\
  0 & 0 \\
  1 & 1 \\
\end{array} \]

\[ T_2 \text{'s view} \]

\[ \begin{array}{cc}
  x & y \\
  0 & 0 \\
\end{array} \]

Writes choose timestamp greater than the thread's view, not necessarily the globally greatest one.
Supporting write-write reordering

\[
\begin{align*}
x &= y = 0 \\
x &:= 1; & y &:= 1; \\
y &:= 2; & \triangleright x &:= 2; \\
\triangleright a &:= y \quad \text{// 1} & b &:= x \quad \text{// 1}
\end{align*}
\]

Memory
\[
\begin{array}{c}
\langle x : 0 @ 0 \rangle \\
\langle y : 0 @ 0 \rangle \\
\langle x : 1 @ 1 \rangle \\
\langle y : 2 @ 1 \rangle \\
\langle y : 1 @ 2 \rangle \\
\end{array}
\]

\[
T_1's \text{ view} \\
\begin{array}{cc}
x & y \\
0 & 0 \\
1 & 1 \\
\end{array}
\]

\[
T_2's \text{ view} \\
\begin{array}{cc}
x & y \\
0 & 0 \\
1 & 2 \\
\end{array}
\]

\[\text{Writes choose timestamp greater than the thread's view, not necessarily the globally greatest one.}\]
Supporting write-write reordering

2+2W

\[ x = y = 0 \]

\[ x := 1; \quad y := 1; \]
\[ y := 2; \quad x := 2; \]
\[ a := y \quad \# 1 \quad b := x \quad \# 1 \]

Memory

- \( \langle x : 0@0 \rangle \)
- \( \langle y : 0@0 \rangle \)
- \( \langle x : 1@1 \rangle \)
- \( \langle y : 2@1 \rangle \)
- \( \langle y : 1@2 \rangle \)
- \( \langle x : 2@0.5 \rangle \)

\( T_1 \)'s view

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( T_2 \)'s view

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

- Writes choose timestamp *greater than the thread’s view*, not necessarily the globally greatest one.
Load buffering

COH allows this outcome.

But, the suggested operational semantics disallows it!

We will see later an approach to fix this mismatch (using out-of-order execution).

For now, we will strengthen the declarative semantics.
Declarative semantics for strong coherence

**Definition (Strong coherence)**

An execution $G$ is *strongly coherent* if the following hold:

- $G$ is complete.
- $G$ is coherent wrt some modification order $mo$ for $G$.
- $G.po \cup G.rf$ is acyclic.

**A note about the implementability of StrongCOH**

Some hardware implementations (e.g., ARM) allow $po \cup rf$ cycles involving only plain loads and stores. To implement StrongCOH on those architectures, a syntactic dependency or a fence has to be introduced between every load and subsequent store.
The state consists of

- a program $P$
- a store function $S$
- a memory $M$
- a thread view function $V$

Initial state $\langle P, S_0, M_0, V_0 \rangle$ where

- $S_0 = \lambda i. s_0 = \lambda i. \lambda r. 0$
- $M_0 = \{ \langle x : 0@0 \rangle \mid x \in \text{Loc} \}$
- $V = \lambda i. \text{view}_0 = \lambda i. \lambda x. 0$. 
Machine transitions

SILENT-THREAD

\[ P, S \xrightarrow{i: \varepsilon} P', S' \]

\[ P, S, M, V \Rightarrow P', S', M, V \]

READ

\[ P, S \xrightarrow{i:l} P', S' \]
\[ l = R(x, v) \]
\[ \langle x : v@t \rangle \in M \]
\[ V(i)(x) \leq t \]
\[ \text{view}' = V(i)[x \mapsto t] \]

\[ P, S, M, V \Rightarrow P', S', M, V[i \mapsto \text{view}'] \]

WRITE

\[ P, S \xrightarrow{i:l} P', S' \]
\[ l = W(x, v) \]
\[ V(i)(x) < t \]
\[ \forall v'. \langle x : v'@t \rangle \notin M \]
\[ M' = M \cup \{\langle x : v@t \rangle\} \]
\[ \text{view}' = V(i)[x \mapsto t] \]

\[ P, S, M, V \Rightarrow P', S', M', V[i \mapsto \text{view}'] \]

Definition (Operational StrongCOH)

An outcome \( O \) is allowed for a program \( P \) under StrongCOH if there exist \( M, V \) such that \( P, S_0, M_0, V_0 \Rightarrow^* \text{skip} \parallel \ldots \parallel \text{skip}, O, M, V \).
Correspondence proof

As for SC, we will introduce an “intermediate” semantics for StrongCOH.

Establish correspondence between operational StrongCOH and declarative StrongCOH in two steps:

1. operational StrongCOH = intermediate StrongCOH
2. declarative StrongCOH = intermediate StrongCOH
Operational version of StrongCOH declarative semantics

**State** \( \langle P, S, G, mo \rangle \) where \( P \in \text{Program}, \ S \in (\text{Tid} \rightarrow \text{Store}), \ G \in \text{ExecutionGraph}, \ mo \subseteq G.E \times G.E. \)

- Initial stores: \( S_0 \triangleq \lambda i. \ s_0 \)
- Initial execution: \( G_0 \) consisting only of the initialization events
- Initial modification order: \( mo_0 = \emptyset \)

\[
\begin{align*}
\text{SILENT} & \\
P, S & \xrightarrow{i:\varepsilon} P', S' \\
\hline
P, S, G, mo & \Rightarrow P', S', G, mo
\end{align*}
\]

\[
\begin{align*}
\text{NON-SILENT} & \\
P, S & \xrightarrow{i:l} P', S' \quad l \neq \varepsilon \\
G' & \in \text{Add}(G, \langle n, i, l \rangle, i) \quad \text{mo} \subseteq \text{mo}' \\
\text{mo}' & \text{is a modification order for } G' \\
G' & \text{is COH-consistent wrt } \text{mo}' \\
\hline
P, S, G, mo & \Rightarrow P', S', G', \text{mo}'
\end{align*}
\]

**Definition (Operational-declarative StrongCOH)**

An outcome \( O \) is allowed for a program \( P \) under StrongCOH if there exist \( G, mo \) such that \( P, S_0, G_0, mo_0 \Rightarrow^{\star} \text{skip} \parallel \ldots \parallel \text{skip}, O, G, mo. \)
Our bisimulation relation:

\[ P, S, M, V \sim P', S', G, \text{mo} \] if the following hold:

- \( P = P' \)
- \( S = S' \)
- there exists a function \( ts : G.W \to \text{Time} \) such that:
  - \( ts(w_1) < ts(w_2) \) whenever \( \langle w_1, w_2 \rangle \in \text{mo} \)
  - \( M = \{ \langle \text{loc}(w) : \text{val}_w(w)@ts(w) \rangle \mid w \in G.W \} \)
  - \( V = \lambda i \ x. \max \{ ts(w) \mid w \in \text{dom}(G.W_x; G.rf^?; [G.E^i]) \} \)
- \( G \) is strongly coherent (wrt \( \text{mo} \)).

Exercise

- Show that \( \sim \) is a bisimulation relation.
- Hence deduce that the operational StrongCOH model and the intermediate StrongCOH model have the same outcomes for any given program.
Declarative StrongCOH $\subseteq$ intermediate StrongCOH

- Let $G$ be an StrongCOH-consistent execution graph of $P_0$ with outcome $O$.
- Let $mo$ be a modification order for $G$ such that $G$ is COH-consistent wrt $mo$.
- We show that $P_0, S_0, G_0, mo_0 \Rightarrow^* \text{skip} \parallel \ldots \parallel \text{skip}, O, G, mo$.
- Let $a_1, \ldots, a_n$ be an enumeration of $G.E \setminus E_0$ following $G.po \cup G.rf$.
- By the “execution generation” lemma, there exist $\langle P_1, S_1 \rangle, \ldots, \langle P_n, S_n \rangle$ such that:
  - $P_n = \text{skip} \parallel \ldots \parallel \text{skip}$ and $S_n = O$
  - For every $1 \leq j \leq n$, we have:
    
    $P_{j-1}, S_{j-1} \xrightarrow{\text{tid}(a_j) : \varepsilon}^* \xrightarrow{\text{tid}(a_j) : \text{lab}(a_j)} \xrightarrow{\text{tid}(a_j) : \varepsilon}^* P_j, S_j$

- For every $0 \leq j \leq n$, let
  - $G_j$ - the restriction of $G$ to $E_0 \cup \{a_1, \ldots, a_j\}$
  - $mo_j$ - the restriction of $mo$ to $E_0 \cup \{a_1, \ldots, a_j\}$
- Then, for every $1 \leq j \leq n$, we have: (why?)
  
  $P_{j-1}, S_{j-1}, G_{j-1}, mo_{j-1} \Rightarrow^* P_j, S_j, G_j, mo_j$
Operational-declarative StrongCOH ⊆ declarative StrongCOH

- Suppose that $P_0, S_0, G_0, \text{mo}_0 \Rightarrow^* \text{skip} \parallel \ldots \parallel \text{skip}, O, G, \text{mo}$.
- We show that $G$ is a StrongCOH-consistent execution graph of $P$ with outcome $O$.
- We know:

  $P_0, S_0, G_0, \text{mo}_0 \Rightarrow P_1, S_1, G_1, \text{mo}_1 \Rightarrow \ldots \Rightarrow P_n, S_n, G_n, \text{mo}_n$

  where $P_n, S_n, G_n, \text{mo}_n = \text{skip} \parallel \ldots \parallel \text{skip}, O, G, \text{mo}$.

- By definition, $G$ is COH-consistent.
- Using induction on the length of the sequence, we also have that $G.\text{po} \cup G.\text{rf}$ is acyclic.
- It remains to show that each $G^i$ is an execution of $P_0(i)$ with final store $O(i)$. (This is done exactly as for SC.)
Operational semantics for RA
Can we extend the operational semantics to support message passing (i.e., release-acquire synchronization)?

**Message passing (MP)**

\[
\begin{align*}
x &= y = 0 \\
x &:= 42; \quad a := y; \quad \text{// 1} \\
y &:= 1 \quad b := x \quad \text{// 0}
\end{align*}
\]

**Double message passing**

\[
\begin{align*}
x &= y = 0 \\
x &:= 42; \quad a := y; \quad \text{// 1} \quad b := z; \quad \text{// 1} \\
y &:= 1 \quad z := 1 \quad c := x \quad \text{// 0}
\end{align*}
\]
Message views

Desired semantics

When reading a message the thread becomes aware of all messages that the writer of the message was aware of when the message was written.

We implement this using *message views*:

- Each message $m$ will carry a view: the view of the thread who wrote $m$ when $m$ was written.
- When reading a message $m$, the thread will update its view to include at least the view contained in $m$. 
Operational semantics for RA

- A message is a tuple $\langle x : v @ t \text{ view} \rangle$ where $x \in \text{Loc}$, $v \in \text{Val}$, $t \in \text{Time}$ and $\text{view} : \text{Loc} \rightarrow \text{Time}$
- Initially, $M_0 \triangleq \{ \langle x : 0 @ 0 \bot \rangle \mid x \in \text{Loc} \}$
- Bottom view: $\bot \triangleq \lambda x. \ 0$
- Joining views: $\text{view}_1 \sqcup \text{view}_2 \triangleq \lambda x. \ \text{max}\{\text{view}_1(x), \text{view}_2(x)\}$

**READ**

\[
P, S \xrightarrow{i : l} P', S' \quad I = \text{R}(x, v) \\
\langle x : v @ t \text{ view} \rangle \in M \quad V(i)(x) \leq t \\
\text{view}' = V(i) \sqcup \text{view} \\
P, S, M, V \Rightarrow P', S', M, V[i \mapsto \text{view}']
\]

**WRITE**

\[
P, S \xrightarrow{i : l} P', S' \quad I = \text{W}(x, v) \\
V(i)(x) < t \quad \forall v', \text{view}. \langle x : v' @ t \text{ view} \rangle \notin M \\
\text{view}' = V(i)[x \mapsto t] \\
M' = M \cup \{ \langle x : v @ t \text{ view}' \rangle \} \\
P, S, M, V \Rightarrow P', S', M', V[i \mapsto \text{view}']
\]
Definition (Operational RA)

An outcome $O$ is allowed for a program $P$ under RA if there exist $M, V$ such that $P, S_0, M_0, V_0 \Rightarrow^* \textbf{skip} \parallel \ldots \parallel \textbf{skip}, O, M, V$.

Exercise

Prove the correspondence between the declarative and the operational definitions of RA.
Suppose we change the write step in the operational semantics of RA as follows:

$$P, S \xrightarrow{i:l} P', S' \quad l = W(x, v)$$

$$\forall t', v', \text{view}. \langle x : v' \odot t' \text{ view} \rangle \in M \Rightarrow t' < t$$

$$\text{view}' = \text{V}(i)[x \mapsto t] \quad M' = M \cup \{\langle x : v \odot t \text{ view}' \rangle\}$$

$$P, S, M, V \Rightarrow P', S', M', V[i \mapsto \text{view}']$$

Here, when writing a message, the thread may only choose a timestamp larger than all timestamps that were used for the given location.

- Show an example which differentiates this model from RA.
- What will be the corresponding declarative semantics?
Recall the following alternative definition of coherence:

Let $mo$ be a modification order for an execution graph $G$. $G$ is coherent wrt $mo$ iff the following hold:

- $rf; po$ is irreflexive. (no-future-read)
- $mo; po$ is irreflexive. (coherence-ww)
- $mo; rf; po$ is irreflexive. (coherence-rw)
- $rf^{-1}; mo; po$ is irreflexive. (coherence-wr)
- $rf^{-1}; mo; rf; po$ is irreflexive. (coherence-rr)

Plain accesses in the Java memory model do not provide full coherence. In particular, they do not ensure “coherence-rr”.

Adapt the StrongCOH timestamp machine to match this weaker variant.
Further reading

- Taming release-acquire consistency. Ori Lahav, Nick Giannarakis, Viktor Vafeiadis. POPL 2016: 649-662