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 $P, S_0, M_0 \Rightarrow^* \textbf{skip} \| \ldots \| \textbf{skip}, O, M$.

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:

$\langle P, S, G, \mathtt{sc}\rangle \in \text{Program} \times (\text{Tid} \to \text{Store}) \times \text{ExecutionGraph} \times \mathcal{P}(\text{Event} \times \text{Event})$

- Initial stores: $S_0 \triangleq \lambda i.\ s_0$
- Initial execution: G_0 consisting only of the initialization events
- Initial $\mathtt{sc}$-relation: $\mathtt{sc}_0$ is an arbitrary total order on $G_0.\mathrm{E}$

SILENT

$$\frac{P, S \xrightarrow{i:\varepsilon} P', S'}{P, S, G, \mathtt{sc} \Rightarrow P', S', G, \mathtt{sc}}$$

NON-SILENT

$$\frac{\begin{array}{c} P, S \xrightarrow{\mathtt{tid}(a):\mathtt{lab}(a)} P', S' \\ G' \in \text{Add}(G, a) \\ \mathtt{sc}' = \mathtt{sc} \cup (G.\mathrm{E} \times \{a\}) \\ G' \text{ is SC-consistent wrt } \mathtt{sc}' \end{array}}{P, S, G, \mathtt{sc} \Rightarrow P', S', G', \mathtt{sc}'}$$

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

- $G'.\mathrm{E} = G.\mathrm{E} \uplus \{a\}$
- $G'.\mathrm{po} = G.\mathrm{po} \cup ((\mathrm{E}_0 \cup G.\mathrm{E}^{\mathtt{tid}(a)}) \times \{a\})$
- $G.\mathtt{rf} \subseteq G'.\mathtt{rf}$

Operational version of SC declarative semantics

Definition (Operational-declarative SC)

An outcome O is allowed for a program P under SC if there exist $G, \mathtt{sc}$ such that $P, S_0, G_0, \mathtt{sc}_0 \Rightarrow^* \textbf{skip} \| \ldots \| \textbf{skip}, O, G, \mathtt{sc}$.

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

1. operational SC = intermediate SC
2. declarative SC = intermediate SC

Operational 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 $q_1 \mathcal{R} q_2$.

Operational SC = intermediate SC

Our bisimulation relation:

$\langle P, S, M\rangle \sim \langle P', S', G, \mathtt{sc}\rangle$ if the following hold:

- $P = P'$
- $S = S'$
- $M = \lambda x.\ \mathtt{val}_{\mathtt{w}}(\max_{\mathtt{sc}} G.\mathtt{W}_x)$
- G is complete and SC-consistent wrt $\mathtt{sc}$.

- Show that $\sim$ is a bisimulation relation.
- Deduce that operational SC and intermediate SC have the same outcomes for any given program.

Declarative SC = 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 $i \in \mathsf{Tid}$.

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.

Declarative SC $\subseteq$ intermediate SC

Lemma (Execution generation)

Let G be an execution of a program P_0 with outcome O. Let $a_1, \dots, a_n$ be an enumeration of $G.\mathtt{E} \setminus \mathtt{E}_0$ that respects $G.\mathtt{po}$. Then, there exist $\langle P_1, S_1 \rangle, \dots, \langle P_n, S_n \rangle$ such that:

- *$P_n = \textbf{\textit{skip}} \| \dots \| \textbf{\textit{skip}}$ and $S_n = O$*
- *For every $1 \leq j \leq n$, we have:*

$$P_{j-1}, S_{j-1} \xrightarrow{\mathtt{tid}(a_j):\varepsilon}{}^{*} \xrightarrow{\mathtt{tid}(a_j):\mathtt{lab}(a_j)} \xrightarrow{\mathtt{tid}(a_j):\varepsilon}{}^{*} P_j, S_j$$

Declarative SC $\subseteq$ intermediate SC

- Let G be an SC-consistent execution graph of P_0 with outcome O.
- Let $\texttt{sc}$ be a total order on $G.\mathrm{E}$ such that G is SC-consistent wrt $\texttt{sc}$.
- We show that $P, S_0, G_0, \texttt{sc}_0 \Rightarrow^* \textbf{skip} \| \dots \| \textbf{skip}, O, G, \texttt{sc}$.
- Let $a_1, \dots, a_n$ be an enumeration of $G.\mathrm{E} \setminus \mathrm{E}_0$ following $\texttt{sc}$.
- Since $G.\mathrm{po} \subseteq \texttt{sc}$, by the previous lemma, there exist $\langle P_1, S_1 \rangle, \dots, \langle P_n, S_n \rangle$ such that:
  - $P_n = \textbf{skip} \| \dots \| \textbf{skip}$ and $S_n = O$
  - For every $1 \leq j \leq n$, we have:

$$P_{j-1}, S_{j-1} \xrightarrow{\texttt{tid}(a_j):\varepsilon}{}^{*} \xrightarrow{\texttt{tid}(a_j):\texttt{lab}(a_j)} \xrightarrow{\texttt{tid}(a_j):\varepsilon}{}^{*} P_j, S_j$$

- For every $0 \leq j \leq n$, let
  - G_j - the restriction of G to $\mathrm{E}_0 \cup \{a_1, \dots, a_j\}$
  - $\texttt{sc}_j$ - the restriction of $\texttt{sc}$ to $\mathrm{E}_0 \cup \{a_1, \dots, a_j\}$
- Then, for every $1 \leq j \leq n$, we have:

$$P_{j-1}, S_{j-1}, G_{j-1}, \texttt{sc}_{j-1} \Rightarrow^* P_j, S_j, G_j, \texttt{sc}_j$$

Operational-declarative SC $\subseteq$ declarative SC

- Suppose that $P_0, S_0, G_0, \texttt{sc}_0 \Rightarrow^* \textbf{skip} \| \dots \| \textbf{skip}, O, G, \texttt{sc}$.
- 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: $P_0, S_0, G_0, \texttt{sc}_0 \Rightarrow P_1, S_1, G_1, \texttt{sc}_1 \Rightarrow \dots \Rightarrow P_n, S_n, G_n, \texttt{sc}_n$ where $P_n, S_n, G_n, \texttt{sc}_n = \textbf{skip} \| \dots \| \textbf{skip}, O, G, \texttt{sc}$.
- The sequence above induces the following sequence of transitions:

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

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

$$P_0(i), S_0(i) \xrightarrow{l_{k_1}} P_{k_1}(i), S_{k_1}(i) \xrightarrow{l_{k_2}} \dots \xrightarrow{l_{k_{n_i}}} P_{k_{n_i}}(i), S_{k_{n_i}}(i) = \textbf{skip}, O(i)$$

- It follows that $P_0(i), s_0, G_\emptyset \Rightarrow^* \textbf{skip}, O(i), G^i$, and so G^i is an execution of $P_0(i)$ with final store $O(i)$.

Operational semantics for COH

Operational semantics for coherence

Recall the following litmus tests:

Store buffering

$$x = y = 0$$

$$\begin{array}{l l || l l} x := 1 & & y := 1 & \\ a := y & /\!/ \, 0 & b := x & /\!/ \, 0 \end{array}$$

Coherence test

$$x = 0$$

$$\begin{array}{l l || l l} x := 1 & & x := 2 & \\ a := x & /\!/ \, 2 & b := x & /\!/ \, 1 \end{array}$$

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

$$x = y = 0$$

$$\begin{array}{ll|ll} x := 1; & & y := 1; & \\ a := y; & \textit{// 0} & b := x; & \textit{// 0} \end{array}$$

Operational semantics for coherence

Store buffering

$$x = y = 0$$

▶ $x := 1;$	‖	▶ $y := 1;$
$a := y;$ *// 0*	‖	$b := x;$ *// 0*

Memory

$\langle x : 0@0 \rangle$

$\langle y : 0@0 \rangle$

T_1's view

x	y
0	0

T_2's view

x	y
0	0

- Global memory is a pool of messages of the form

$$\langle \textit{location} \;:\; \textit{value} \; @ \; \textit{timestamp} \rangle$$

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

Operational semantics for coherence

Store buffering

$$x = y = 0$$

$x := 1;$	∥	▶ $y := 1;$
▶ $a := y;$ // 0	∥	$b := x;$ // 0

Memory

$\langle x : 0@0 \rangle$
$\langle y : 0@0 \rangle$
$\langle x : 1@1 \rangle$

T_1's view

x	y
$\cancel{0}$	0
1	

T_2's view

x	y
0	0

- Global memory is a pool of messages of the form

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

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

Operational semantics for coherence

Store buffering

$$x = y = 0$$

$$\begin{array}{l||l} x := 1; & y := 1; \\ \blacktriangleright\ a := y; \quad /\!/\, 0 & \blacktriangleright\ b := x; \quad /\!/\, 0 \end{array}$$

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

x	y
$\cancel{0}$	0
1	

T_2's view

x	y
0	$\cancel{0}$
	1

- Global memory is a pool of messages of the form

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

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

Operational semantics for coherence

Store buffering

$$x = y = 0$$

$x := 1;$	$\parallel$	$y := 1;$
$a := y;$ // 0	$\parallel$	▶ $b := x;$ // 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

x	y
$\cancel{0}$	0
1	

T_2's view

x	y
0	$\cancel{0}$
	1

- Global memory is a pool of messages of the form

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

- 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;            ||   y := 1;
a := y;   // 0     ||   b := x;   // 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**

x	y
$\cancel{0}$	0
1	

T_2**'s view**

x	y
0	$\cancel{0}$
	1

- Global memory is a pool of messages of the form

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

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

Operational semantics for coherence

Store buffering

$$x = y = 0$$

$x := 1;$	$\parallel$	$y := 1;$
$a := y;$ // 0	$\parallel$	$b := x;$ // 0
▶	$\parallel$	▶

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

x	y
$\cancel{0}$	0
1	

T_2's view

x	y
0	$\cancel{0}$
	1

Coherence test

$$x = 0$$

$x := 1;$	$\parallel$	$x := 2;$
$a := x;$ // 2	$\parallel$	$b := x;$ // 1

Operational semantics for coherence

Store buffering

$$x = y = 0$$

$$\begin{array}{ll||ll} x := 1; & & y := 1; & \\ a := y; & /\!/\, 0 & b := x; & /\!/\, 0 \\ \blacktriangleright & & \blacktriangleright & \end{array}$$

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

x	y
~~0~~	0
1	

T_2's view

x	y
0	~~0~~
	1

Coherence test

$$x = 0$$

$$\begin{array}{ll||ll} \blacktriangleright\ x := 1; & & \blacktriangleright\ x := 2; & \\ a := x; & /\!/\, 2 & b := x; & /\!/\, 1 \end{array}$$

Memory

$\langle x : 0@0 \rangle$

T_1's view

x
0

T_2's view

x
0

Operational semantics for coherence

Store buffering

$x = y = 0$

$x := 1;$	$y := 1;$
$a := y;$ // 0	$b := x;$ // 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

x	y
$\cancel{0}$	0
1	

T_2's view

x	y
0	$\cancel{0}$
	1

Coherence test

$x = 0$

$x := 1;$	▶ $x := 2;$
▶ $a := x;$ // 2	$b := x;$ // 1

Memory
$\langle x : 0@0 \rangle$
$\langle x : 1@1 \rangle$

T_1's view

x
$\cancel{0}$
1

T_2's view

x
0

Operational semantics for coherence

Store buffering

$$x = y = 0$$

$$\begin{array}{l||l} x := 1; & y := 1; \\ a := y; \quad /\!/\, 0 & b := x; \quad /\!/\, 0 \\ \blacktriangleright & \blacktriangleright \end{array}$$

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

x	y
$\cancel{0}$	0
1	

T_2's view

x	y
0	$\cancel{0}$
	1

Coherence test

$$x = 0$$

$$\begin{array}{l||l} x := 1; & x := 2; \\ \blacktriangleright\ a := x; \quad /\!/\, 2 & \blacktriangleright\ b := x; \quad /\!/\, 1 \end{array}$$

Memory

$\langle x : 0@0 \rangle$
$\langle x : 1@1 \rangle$
$\langle x : 2@2 \rangle$

T_1's view

x
$\cancel{0}$
1

T_2's view

x
$\cancel{0}$
2

Operational semantics for coherence

Store buffering

$$x = y = 0$$

$x := 1;$	$\parallel$	$y := 1;$
$a := y;$ // 0	$\parallel$	$b := x;$ // 0
▶	$\parallel$	▶

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

x	y
~~0~~	0
1	

T_2's view

x	y
0	~~0~~
	1

Coherence test

$$x = 0$$

$x := 1;$	$\parallel$	$x := 2;$
$a := x;$ // 2	$\parallel$	▶ $b := x;$ // 1
▶	$\parallel$	

Memory

$\langle x : 0@0 \rangle$
$\langle x : 1@1 \rangle$
$\langle x : 2@2 \rangle$

T_1's view

x
~~0~~
~~1~~
2

T_2's view

x
~~0~~
2

Operational semantics for coherence

Store buffering

$$x = y = 0$$

$$\begin{array}{l||l} x := 1; & y := 1; \\ a := y; \quad /\!/\, 0 & b := x; \quad /\!/\, 0 \\ \blacktriangleright & \blacktriangleright \end{array}$$

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

x	y
$\cancel{0}$	0
1	

T_2's view

x	y
0	$\cancel{0}$
	1

Coherence test

$$x = 0$$

$$\begin{array}{l||l} x := 1; & x := 2; \\ a := x; \quad /\!/\, 2 & b := x; \quad /\!/\, 1 \\ \blacktriangleright & \blacktriangleright \end{array}$$

Memory

$\langle x : 0@0 \rangle$
$\langle x : 1@1 \rangle$
$\langle x : 2@2 \rangle$

T_1's view

x
$\cancel{0}$
$\cancel{1}$
2

T_2's view

x
$\cancel{0}$
2

Supporting write-write reordering

2+2W

$$x = y = 0$$

$$\begin{array}{l||l} x := 1; & y := 1; \\ y := 2; & x := 2; \\ a := y \quad /\!/ \mathit{1} & b := x \quad /\!/ \mathit{1} \end{array}$$

Supporting write-write reordering

2+2W

$$x = y = 0$$

▶ $x := 1;$	‖	▶ $y := 1;$
$y := 2;$	‖	$x := 2;$
$a := y$ *// 1*	‖	$b := x$ *// 1*

Memory

$\langle x : 0@0 \rangle$

$\langle y : 0@0 \rangle$

T_1's view

x	y
0	0

T_2's view

x	y
0	0

Supporting write-write reordering

2+2W

$x = y = 0$

$x := 1;$	▶ $y := 1;$
▶ $y := 2;$	$x := 2;$
$a := y$ // *1*	$b := x$ // *1*

Memory

$\langle x : 0@0 \rangle$

$\langle y : 0@0 \rangle$

$\langle x : 1@1 \rangle$

T_1's view

x	y
$\cancel{0}$	0
1	

T_2's view

x	y
0	0

Supporting write-write reordering

2+2W

$$x = y = 0$$

$x := 1;$	$\parallel$	$\blacktriangleright\ y := 1;$
$y := 2;$		$x := 2;$
$\blacktriangleright\ a := y \quad /\!/ 1$		$b := x \quad /\!/ 1$

Memory

$\langle x : 0@0 \rangle$
$\langle y : 0@0 \rangle$
$\langle x : 1@1 \rangle$
$\langle y : 2@1 \rangle$

T_1**'s view**

x	y
$\cancel{0}$	$\cancel{0}$
1	1

T_2**'s view**

x	y
0	0

Supporting write-write reordering

2+2W

$x = y = 0$

$x := 1;$	$y := 1;$
$y := 2;$	$\blacktriangleright\ x := 2;$
$\blacktriangleright\ a := y$ *// 1*	$b := x$ *// 1*

Memory

$\langle x : 0@0 \rangle$
$\langle y : 0@0 \rangle$
$\langle x : 1@1 \rangle$
$\langle y : 2@1 \rangle$
$\langle y : 1@2 \rangle$

T_1's view

x	y
$\cancel{0}$	$\cancel{0}$
1	1

T_2's view

x	y
0	$\cancel{0}$
	2

Supporting write-write reordering

```
2+2W
              x = y = 0
    x := 1;            ||     y := 1;
    y := 2;            ||     x := 2;
  ▶ a := y   // 1      ||   ▶ b := x   // 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

x	y
$\cancel{0}$	$\cancel{0}$
1	1

T_2's view

x	y
$\cancel{0}$	$\cancel{0}$
0.5	2

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

Load buffering

Load buffering (LB)

$$x = y = 0$$

$$\begin{array}{ll|l} a := x & \textit{// 1} & b := y \quad \textit{// 1} \\ y := 1 & & x := 1 \end{array}$$

- 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 $\mathtt{mo}$ for G.
- $G.\mathrm{po} \cup G.\mathtt{rf}$ is acyclic.

A note about the implementability of StrongCOH

Some hardware implementations (*e.g.*, ARM) allow $\mathrm{po} \cup \mathtt{rf}$ 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.

Operational semantics for strong coherence

- Time $\triangleq \{t \in \mathbb{Q} \mid t \geq 0\}$ is the set of timestamps.
- A message is a triple $\langle x : v\texttt{@}t \rangle$ where $x \in \mathsf{Loc}$, $v \in \mathsf{Val}$, and $t \in \mathsf{Time}$.
- A memory is a finite set of messages.
- A view is a function $\mathit{view} : \mathsf{Loc} \to \mathsf{Time}$.
- A thread view function is a function $V : \mathsf{Tid} \to (\mathsf{Loc} \to \mathsf{Time})$ assigning a view to every thread.

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

- $S_0 = \lambda i.\, s_0 = \lambda i.\, \lambda r.\, 0$
- $M_0 = \{\langle x : 0\texttt{@}0 \rangle \mid x \in \mathsf{Loc}\}$
- $V = \lambda i.\, \mathit{view}_0 = \lambda i.\, \lambda x.\, 0.$

Machine transitions

$$\frac{P, S \xrightarrow{i:\varepsilon} P', S'}{P, S, M, V \Rightarrow P', S', M, V}\ \textsc{Silent-Thread}$$

$$\frac{P, S \xrightarrow{i:l} P', S' \qquad l = \mathtt{R}(x, v) \\ \langle x : v@t \rangle \in M \qquad V(i)(x) \leq t \\ view' = V(i)[x \mapsto t]}{P, S, M, V \Rightarrow P', S', M, V[i \mapsto view']}\ \textsc{Read}$$

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

Definition (Operational StrongCOH)

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

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

- Initial stores: $S_0 \triangleq \lambda i.\ s_0$
- Initial execution: G_0 consisting only of the initialization events
- Initial modification order: $\texttt{mo}_0 = \emptyset$

SILENT

$$\frac{P, S \xrightarrow{i:\varepsilon} P', S'}{P, S, G, \texttt{mo} \Rightarrow P', S', G, \texttt{mo}}$$

NON-SILENT

$$\frac{\begin{array}{c} P, S \xrightarrow{i:l} P', S' \qquad l \neq \varepsilon \\ G' \in \text{Add}(G, \langle n, i, l\rangle, i) \qquad \texttt{mo} \subseteq \texttt{mo}' \\ \texttt{mo}' \text{ is a modification order for } G' \\ G' \text{ is COH-consistent wrt } \texttt{mo}' \end{array}}{P, S, G, \texttt{mo} \Rightarrow P', S', G', \texttt{mo}'}$$

Definition (Operational-declarative StrongCOH)

An outcome O is allowed for a program P under StrongCOH if there exist $G, \texttt{mo}$ such that $P, S_0, G_0, \texttt{mo}_0 \Rightarrow^* \textbf{skip} \| \ldots \| \textbf{skip}, O, G, \texttt{mo}$.

Operational StrongCOH = intermediate StrongCOH

Our bisimulation relation:

$P, S, M, V \sim P', S', G, \texttt{mo}$ if the following hold:

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

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 $\texttt{mo}$ be a modification order for G such that G is COH-consistent wrt $\texttt{mo}$.
- We show that $P_0, S_0, G_0, \texttt{mo}_0 \Rightarrow^* \textbf{skip} \| \dots \| \textbf{skip}, O, G, \texttt{mo}$.
- Let $a_1, \dots, a_n$ be an enumeration of $G.\texttt{E} \setminus \texttt{E}_0$ following $G.\texttt{po} \cup G.\texttt{rf}$.
- By the "execution generation" lemma, there exist $\langle P_1, S_1 \rangle, \dots, \langle P_n, S_n \rangle$ such that:
  - $P_n = \textbf{skip} \| \dots \| \textbf{skip}$ and $S_n = O$
  - For every $1 \leq j \leq n$, we have:

$$P_{j-1}, S_{j-1} \xrightarrow{\texttt{tid}(a_j):\varepsilon}{}^{*} \xrightarrow{\texttt{tid}(a_j):\texttt{lab}(a_j)} \xrightarrow{\texttt{tid}(a_j):\varepsilon}{}^{*} P_j, S_j$$

- For every $0 \leq j \leq n$, let
  - G_j - the restriction of G to $\texttt{E}_0 \cup \{a_1, \dots, a_j\}$
  - $\texttt{mo}_j$ - the restriction of $\texttt{mo}$ to $\texttt{E}_0 \cup \{a_1, \dots, a_j\}$
- Then, for every $1 \leq j \leq n$, we have: (why?)

$$P_{j-1}, S_{j-1}, G_{j-1}, \texttt{mo}_{j-1} \Rightarrow^* P_j, S_j, G_j, \texttt{mo}_j$$

Operational-declarative StrongCOH $\subseteq$ declarative StrongCOH

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

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

  where $P_n, S_n, G_n, \mathtt{mo}_n = \textbf{skip} \| \dots \| \textbf{skip}, O, G, \mathtt{mo}$.
- By definition, G is COH-consistent.
- Using induction on the length of the sequence, we also have that $G.\mathrm{po} \cup G.\mathtt{rf}$ 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

Release/acquire synchronization

Can we extend the operational semantics to support message passing (*i.e.*, release-acquire synchronization)?

Message passing (MP)

$$x = y = 0$$

$$\begin{array}{l} x := 42; \\ y := 1 \end{array} \;\Bigg\|\; \begin{array}{l} a := y; \;\; /\!/ \, 1 \\ b := x \;\; /\!/ \, 0 \end{array}$$

Double message passing

$$x = y = 0$$

$$\begin{array}{l} x := 42; \\ y := 1 \end{array} \;\Bigg\|\; \begin{array}{l} a := y; \;\; /\!/ \, 1 \\ z := 1 \end{array} \;\Bigg\|\; \begin{array}{l} b := z; \;\; /\!/ \, 1 \\ c := x \;\; /\!/ \, 0 \end{array}$$

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 $\langle x : v@t \;\; \mathit{view}\rangle$ where $x \in \mathsf{Loc}$, $v \in \mathsf{Val}$, $t \in \mathsf{Time}$ and $\mathit{view} : \mathsf{Loc} \to \mathsf{Time}$
- Initially, $M_0 \triangleq \{\langle x : 0@0 \;\; \bot\rangle \mid x \in \mathsf{Loc}\}$
- Bottom view: $\bot \triangleq \lambda x.\ 0$
- Joining views: $\mathit{view}_1 \sqcup \mathit{view}_2 \triangleq \lambda x.\ \max\{\mathit{view}_1(x), \mathit{view}_2(x)\}$

$$\textsc{Silent-Thread}\quad \frac{P, S \xrightarrow{i:\varepsilon} P', S'}{P, S, M, V \Rightarrow P', S', M, V}$$

$$\textsc{Read}\quad \frac{\begin{array}{c} P, S \xrightarrow{i:l} P', S' \qquad l = \mathtt{R}(x, v) \\ \langle x : v@t \;\; \mathit{view}\rangle \in M \qquad V(i)(x) \le t \\ \mathit{view}' = V(i) \sqcup \mathit{view} \end{array}}{P, S, M, V \Rightarrow P', S', M, V[i \mapsto \mathit{view}']}$$

$$\textsc{Write}\quad \frac{\begin{array}{c} P, S \xrightarrow{i:l} P', S' \qquad l = \mathtt{W}(x, v) \\ V(i)(x) < t \qquad \forall v', \mathit{view}.\ \langle x : v'@t \;\; \mathit{view}\rangle \notin M \\ \mathit{view}' = V(i)[x \mapsto t] \qquad M' = M \cup \{\langle x : v@t \;\; \mathit{view}'\rangle\} \end{array}}{P, S, M, V \Rightarrow P', S', M', V[i \mapsto \mathit{view}']}$$

Exercise: RA operational semantics

Definition (Operational RA)

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

Exercise

Prove the correspondence between the declarative and the operational definitions of RA.

Exercise: Strong release/acquire semantics

Suppose we change the write step in the operational semantics of RA as follows:

$$\textsc{write}\quad \frac{\begin{array}{c} P, S \xrightarrow{i:l} P', S' \qquad l = \mathtt{W}(x, v) \\ \forall t', v', view.\ \langle x : v'@t' \quad view \rangle \in M \Rightarrow t' < t \\ view' = V(i)[x \mapsto t] \qquad M' = M \cup \{\langle x : v@t \quad view' \rangle\} \end{array}}{P, S, M, V \Rightarrow P', S', M', V[i \mapsto view']}$$

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?

Exercise: Plain accesses in Java

Recall the following alternative definition of coherence:

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

- $\texttt{rf}; \mathrm{po}$ is irreflexive. (no-future-read)
- $\texttt{mo}; \mathrm{po}$ is irreflexive. (coherence-ww)
- $\texttt{mo}; \texttt{rf}; \mathrm{po}$ is irreflexive. (coherence-rw)
- $\texttt{rf}^{-1}; \texttt{mo}; \mathrm{po}$ is irreflexive. (coherence-wr)
- $\texttt{rf}^{-1}; \texttt{mo}; \texttt{rf}; \mathrm{po}$ 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
- A promising semantics for relaxed-memory concurrency. Jeehoon Kang, Chung-Kil Hur, Ori Lahav, Viktor Vafeiadis, Derek Dreyer. POPL 2017: 175-189