

# Basic operational semantics for concurrency

Ori Lahav

Viktor Vafeiadis

28 August 2017

# A simple concurrent programming language

## Basic domains:

- $r \in \text{Reg}$  – Registers (local variables)
- $x \in \text{Loc}$  – Locations
- $v \in \text{Val}$  – Values including 0
- $i \in \text{Tid} = \{1, \dots, N\}$  – Thread identifiers

## Expressions and commands:

- $e ::= r \mid v \mid e + e \mid \dots$
- $c ::= \text{skip} \mid \text{if } e \text{ then } c \text{ else } c \mid \text{while } e \text{ do } c \mid$   
 $c ; c \mid r := e \mid r := x \mid x := e \mid$   
 $r := \mathbf{FAA}(x, e) \mid r := \mathbf{CAS}(x, e, e) \mid \mathbf{fence}$

**Programs**,  $P : \text{Tid} \rightarrow \text{Cmd}$ , written as  $P = c_1 \parallel \dots \parallel c_N$

## Thread subsystem

- ▶ Thread-local steps:  $c, s \xrightarrow{l} c', s'$ .
- ▶ Interpret sequential programs.
- ▶ Lift them to program steps:  $P, S \xrightarrow{i:l} P', S'$ .

## Storage subsystem (defined by the memory model)

- ▶ Describe the effect of memory accesses and fences.
- ▶  $M \xrightarrow{i:l} M'$  where  $M$  is the state of the storage subsystem.

## Linking the two

- ▶ Either the thread or the storage subsystem make an internal step,  $\varepsilon$ ; or they make matching  $i:l$  steps.
- ▶  $P, S, M \Rightarrow P', S', M'$ .

# The thread subsystem

**Store:**  $s : \text{Reg} \rightarrow \text{Val}$       (Initial store:  $s_0 \triangleq \lambda r. 0$ )

**State:**  $\langle c, s \rangle \in \text{Command} \times \text{Store}$

**Transitions:**

$$\frac{}{\text{skip}; c, s \xrightarrow{\varepsilon} c, s} \qquad \frac{c_1, s \xrightarrow{l} c'_1, s'}{c_1; c_2, s \xrightarrow{l} c'_1; c_2, s'} \qquad \frac{s' = s[r \mapsto s(e)]}{r := e, s \xrightarrow{\varepsilon} \text{skip}, s'}$$

$$\frac{l = R(x, v)}{r := x, s \xrightarrow{l} \text{skip}, s[r \mapsto v]} \qquad \frac{l = W(x, s(e))}{x := e, s \xrightarrow{l} \text{skip}, s}$$

$$\frac{s(e) \neq 0}{\text{if } e \text{ then } c_1 \text{ else } c_2, s \xrightarrow{\varepsilon} c_1, s} \qquad \frac{s(e) = 0}{\text{if } e \text{ then } c_1 \text{ else } c_2, s \xrightarrow{\varepsilon} c_2, s}$$

$$\frac{}{\text{while } e \text{ do } c, s \xrightarrow{\varepsilon} \text{if } e \text{ then } (c; \text{while } e \text{ do } c) \text{ else skip}, s}$$

# The thread subsystem: RMW and fence commands

## Fetch-and-add:

$$\frac{l = U(x, v, v + s(e))}{r := \mathbf{FAA}(x, e), s \xrightarrow{l} \mathbf{skip}, s[r \mapsto v]}$$

## Compare-and-swap:

$$\frac{l = R(x, v) \quad v \neq s(e_r)}{r := \mathbf{CAS}(x, e_r, e_w), s \xrightarrow{l} \mathbf{skip}, s[r \mapsto 0]}$$

$$\frac{l = U(x, s(e_r), s(e_w))}{r := \mathbf{CAS}(x, e_r, e_w), s \xrightarrow{l} \mathbf{skip}, s[r \mapsto 1]}$$

## Fence:

$$\frac{}{\mathbf{fence}, s \xrightarrow{F} \mathbf{skip}, s}$$

# Lifting to concurrent programs

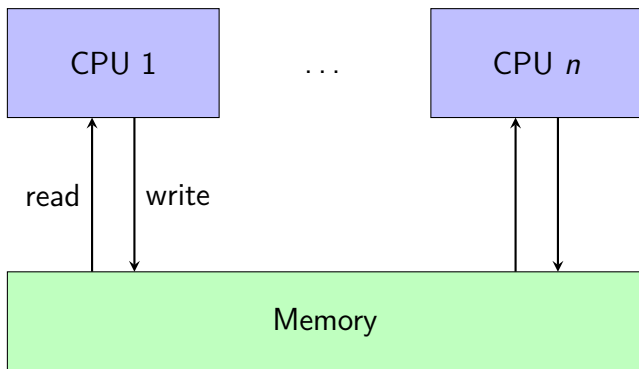
**State:**  $\langle P, S \rangle \in \text{Program} \times (\text{Tid} \rightarrow \text{Store})$

- ▶ Initial stores:  $S_0 \triangleq \lambda i. s_0$
- ▶ Initial state:  $\langle P, S_0 \rangle$

**Transition:**

$$\frac{P(i), S(i) \xrightarrow{l} c, s}{P, S \xrightarrow{i:l} P[i \mapsto c], S[i \mapsto s]}$$

# SC storage subsystem



# SC storage subsystem

**Machine state:**  $M : \text{Loc} \rightarrow \text{Val}$

- ▶ Maps each location to its value.
- ▶ Initial state:  $M_0 \triangleq \lambda x. 0$   
(i.e., the memory that maps every location to 0)

**Transitions:**

$$\frac{I = W(x, v)}{M \xrightarrow{i:l} M[x \mapsto v]}$$

$$\frac{I = R(x, v) \quad M(x) = v}{M \xrightarrow{i:l} M}$$

$$\frac{I = U(x, v_r, v_w) \quad M(x) = v_r}{M \xrightarrow{i:l} M[x \mapsto v_w]}$$

$$\frac{I = F}{M \xrightarrow{i:l} M}$$



## SC: Linking the thread and storage subsystems

SILENT

$$\frac{P, S \xrightarrow{i:\varepsilon} P', S'}{P, S, M \Rightarrow P', S', M}$$

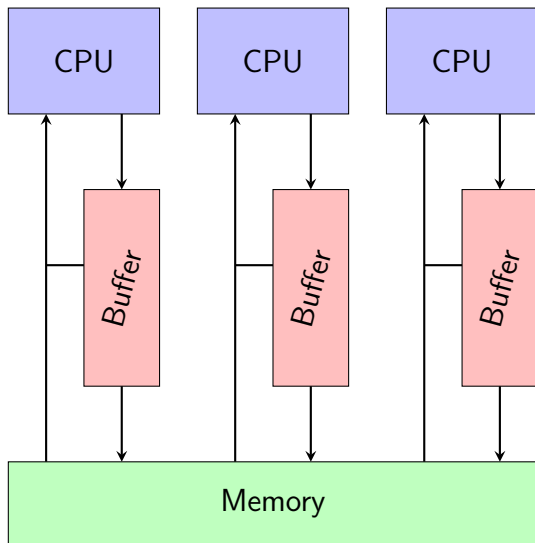
NON-SILENT

$$\frac{P, S \xrightarrow{i:l} P', S' \quad M \xrightarrow{i:l} M'}{P, S, M \Rightarrow P', S', M'}$$

### Definition (Allowed outcome)

- ▶ An *outcome* is a function  $O : \text{Tid} \rightarrow \text{Store}$ .
- ▶ An outcome  $O$  is *allowed* for a program  $P$  under SC if there exists  $M$  such that
$$P, S_0, M_0 \Rightarrow^* \text{skip} \parallel \dots \parallel \text{skip}, O, M.$$

# TSO storage subsystem



## The state consists of:

- ▶ A memory  $M : \text{Loc} \rightarrow \text{Val}$
- ▶ A function  $B : \text{Tid} \rightarrow (\text{Loc} \times \text{Val})^*$   
assigning a *store buffer* to every thread.

## Initial state: $\langle M_0, B_0 \rangle$ where

- ▶  $M_0 = \lambda x. 0$  (the memory maps 0 to every location)
- ▶  $B_0 = \lambda i. \epsilon$  (all store buffers are empty)

# TSO storage subsystem transitions

WRITE

$$\frac{l = W(x, v)}{M, B \xrightarrow{i:l} M, B[i \mapsto \langle x, v \rangle \cdot B(i)]}$$

PROPAGATE

$$\frac{B(i) = b \cdot \langle x, v \rangle}{M, B \xrightarrow{i:\epsilon} M[x \mapsto v], B[i \mapsto b]}$$

READ

$$\frac{\begin{array}{l} l = R(x, v) \\ B(i) = \langle x_n, v_n \rangle \cdot \dots \cdot \langle x_2, v_2 \rangle \cdot \langle x_1, v_1 \rangle \\ M[x_1 \mapsto v_1][x_2 \mapsto v_2] \dots [x_n \mapsto v_n](x) = v \end{array}}{M, B \xrightarrow{i:l} M, B}$$

RMW

$$\frac{l = U(x, v_r, v_w) \quad B(i) = \epsilon \quad M(x) = v_r}{M, B \xrightarrow{i:l} M[x \mapsto v_w], B}$$

FENCE

$$\frac{l = F \quad B(i) = \epsilon}{M, B \xrightarrow{i:l} M, B}$$

# TSO: linking thread and storage subsystems

SILENT-THREAD

$$\frac{P, S \xrightarrow{i:\varepsilon} P', S'}{P, S, M, B \Rightarrow P', S', M, B}$$

SILENT-STORAGE

$$\frac{M, B \xrightarrow{i:\varepsilon} M', B'}{P, S, M, B \Rightarrow P, S, M', B'}$$

NON-SILENT

$$\frac{P, S \xrightarrow{i:l} P', S' \quad M, B \xrightarrow{i:l} M', B'}{P, S, M, B \Rightarrow P', S', M', B'}$$

## Definition (Allowed outcome)

An outcome  $O$  is *allowed* for a program  $P$  under TSO if there exists  $M$  such that  $P, S_0, M_0, B_0 \Rightarrow^* \mathbf{skip} \parallel \dots \parallel \mathbf{skip}, O, M, B_0$ .

## Exercise: PSO storage subsystem

*Partial Store Ordering (PSO)* is a WMM similar to TSO, but it does not guarantee that stores to *different* locations propagate to the main memory in the order they were issued. In particular, it allows the following weak behavior:

$$\begin{array}{l} x := 1; \\ y := 1 \end{array} \parallel \begin{array}{l} a := y; \text{ //1} \\ b := x \text{ //0} \end{array}$$

1. Provide operational semantics for PSO.
2. Extend the semantics with a *store-store fence*, whose placement between two stores ensures that the stores propagate to the main memory in their issue order.
3. (Optional) Show that programs containing store-store fences between every two writes have the same outcomes under TSO and PSO.