Concurrent Object-Oriented Languages

The Java Memory Model

wiki.eecs.yorku.ca/course/6490A

Sources

- Jeremy Manson and Brian Goetz. JSR 133 (Java Memory Model) FAQ. February 2004.
- Jeremy Manson, William Pugh and Sarita V. Adve. The Java Memory Model. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* , pages 378–391, Long Beach, CA, USA, January 2005. ACM.
- James Gosling, Bill Joy, Guy Steele and Gilad Bracha. *The Java Language Specification.* Chapter 17. Addison-Wesley, Reading, MA, USA, 2nd edition, 2000.

What is a Memory Model?

A memory model defines necessary and sufficient conditions for knowing that writes to memory by other processors are visible to the current processor, and writes by the current processor are visible to other processors.

Why do we Need a Memory Model?

Local memory (caches, registers, and other hardware and compiler optimizations) can improve performance tremendously, but it presents a host of new challenges. What, for example, happens when two processors examine the same memory location at the same time? Under what conditions will they see the same value? A memory model provides answers to these questions.

Synchronized Methods/Blocks

After we exit a synchronized block, we release the monitor, which has the effect of flushing the cache to main memory, so that writes made by this thread can be visible to other threads.

Before we can enter a synchronized block, we acquire the monitor, which has the effect of invalidating the local processor cache so that variables will be reloaded from main memory. We will then be able to see all of the writes made visible by the previous release.

Volatile

Each read or write of a volatile field acts like "half" a synchronization, for purposes of visibility.

Each read of a volatile will see the last write to that volatile by any thread; in effect, they are designated by the programmer as fields for which it is never acceptable to see a "stale" value as a result of caching or reordering.

What is a Partial Order?

Definition

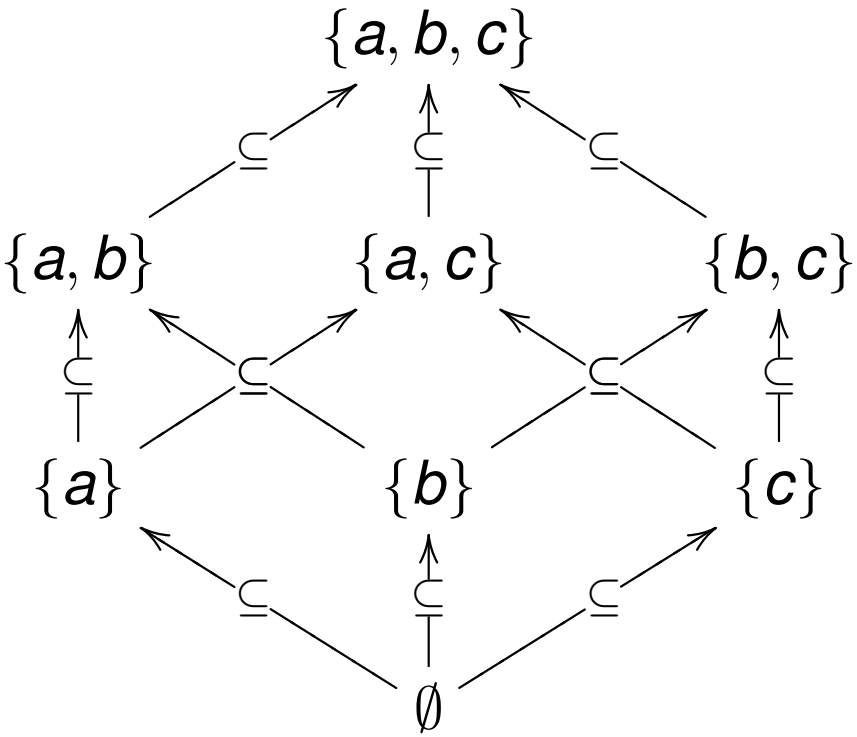
Let X be a set. A binary relation $\sqsubseteq$ on X is a partial order if for all x, y and $z \in X$,

- $x \sqsubseteq x$,
- if $x \sqsubseteq y$ and $y \sqsubseteq x$ then $x = y$, and
- if $x \sqsubseteq y$ and $y \sqsubseteq z$ then $x \sqsubseteq z$.

Partial Orders

- The standard less-than-or-equal relation $\leq$ on the real numbers.
- The relation $\cdot$divides$\cdot$ on the natural numbers.
- The inclusion relation $\subseteq$ on the powerset of a given set.

Partial Orders



What is a Total Order?

Definition

Let X be a set. A binary relation $\sqsubseteq$ on X is a total order if for all x, y and $z \in X$,

- if $x \sqsubseteq y$ and $y \sqsubseteq x$ then $x = y$,
- if $x \sqsubseteq y$ and $y \sqsubseteq z$ then $x \sqsubseteq z$, and
- $x \sqsubseteq y$ or $y \sqsubseteq x$.

Total Orders

- The standard less-than relation $<$ on the real numbers.
- The lexicographic order on words.

Formal Specification of the JMM

An action is described by a tuple $\langle t, k, v \rangle$ where

- t is the thread performing the action,
- k is the kind of action:
  - volatile read;
  - volatile write;
  - non-volatile read;
  - non-volatile write;
  - lock;
  - unlock;
  - special synchronization actions;
  - thread divergence actions;
  - external actions,
- v is the variable or monitor involved in the action.

Examples

- $\langle t, \text{volatile read}, x \rangle$
- $\langle t, \text{volatile write}, x \rangle$
- $\langle t, \text{non-volatile read}, x \rangle$
- $\langle t, \text{non-volatile write}, x \rangle$
- $\langle t, \text{lock}, x \rangle$
- $\langle t, \text{unlock}, x \rangle$

are actions.

Formal Specification of the JMM

Synchronization actions include

- locks,
- unlocks,
- reads of volatile variables, and
- writes to volatile variables.

Formal Specification of the JMM

An execution is described by a tuple $\langle P, A, \xrightarrow{po}, \xrightarrow{so}, W, V \rangle$ where

- P is the program,
- A is the set of actions,
- $\xrightarrow{po}$ is the program order, which for each thread t, is a total order over all actions performed by t in A,
- $\xrightarrow{so}$ is the synchronization order, which is a total order over all synchronization actions in A,
- W is the write-seen function, which for each read r in A, gives $W(r)$, the write action seen by r in the execution,
- V is the value-written function, which for each write w in A, gives $V(w)$, the value written by w in the execution.

The Synchronizes-With Order

The synchronizes-with order is defined in terms of the synchronization order.

For each unlock action u and lock action ℓ, if $u.v = \ell.v$ and $u \xrightarrow{so} \ell$ then $u \xrightarrow{sw} \ell$.

For each volatile read r and volatile write w, if $r.v = w.v$ and $w \xrightarrow{so} r$ then $w \xrightarrow{sw} r$.

Recall that $a.v$ is the variable or monitor involved in the action a.

Transitive Closure

Definition

The reflexive and transitive closure closure(R) of a binary relation R on X is the smallest binary relation on X such that

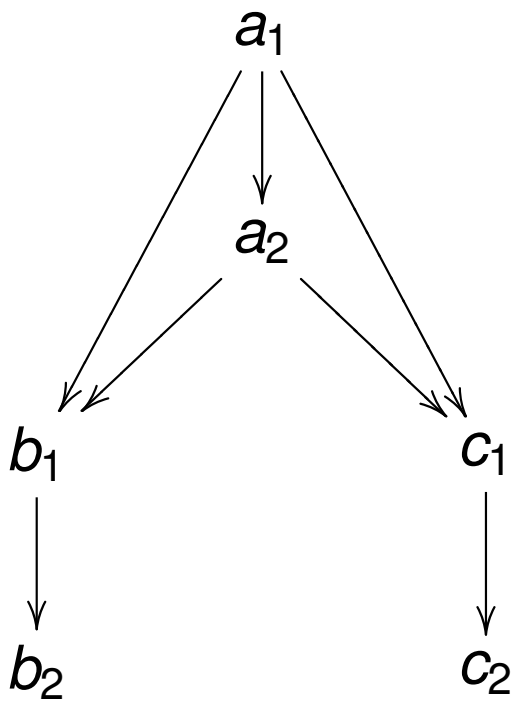
- closure(R) contains R,
  for all $x, y \in X$, if $x\, R\, y$ then x closure(R) y,
- closure(R) is reflexive
  for all $x \in X$, x closure(R) x,
- closure(R) is transitive
  for all $x, y\ z \in X$, if x closure(R) y and y closure(R) z then x closure(R) z.

The Happens-Before Order

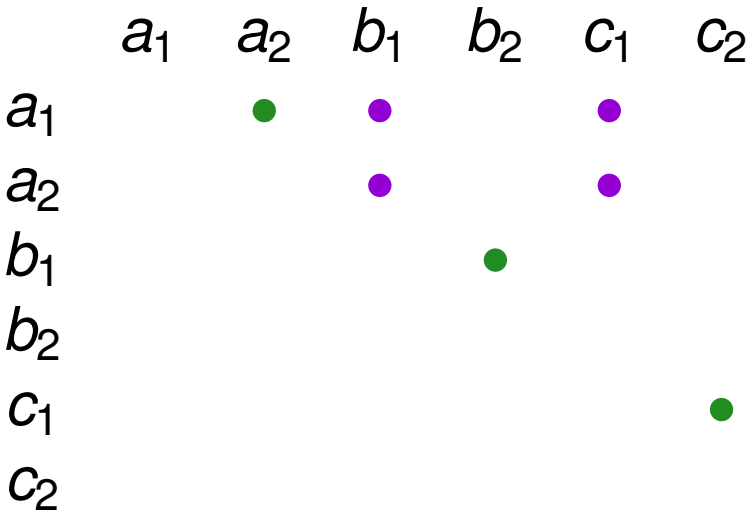
The happens-before order is defined in terms of the program order and the synchronizes-with order.

$$\xrightarrow{hb} = \text{closure}(\xrightarrow{po} \cup \xrightarrow{sw}).$$

Example



Example



Example