The Java Memory Model

March 30, 2011

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 Partial Order?

Definition

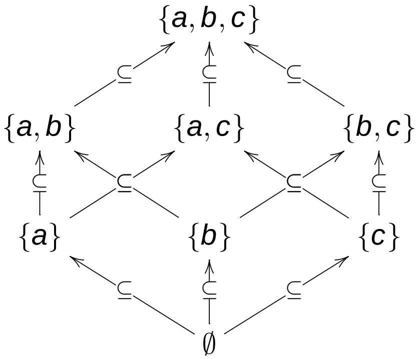
Let X be a set. A binary relation $\sqsubseteq$ on X is a partial order if for all x, y and $z \in X$,

- $x \sqsubseteq x$,
- 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 threads 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

- $\langle T_0, \text{non-volatile write}, x \rangle$
- $\langle T_0, \text{non-volatile write}, y \rangle$
- $\langle T_1, \text{non-volatile read}, x \rangle$
- $\langle T_1, \text{non-volatile write}, y \rangle$
- $\langle T_2, \text{non-volatile read}, y \rangle$
- $\langle T_2, \text{non-volatile write}, x \rangle$

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 $\langle P, A, \stackrel{po}{\rightarrow}, \stackrel{so}{\rightarrow}, W, V \rangle$ where

- P is the program,
- A is the set of actions,
- $\stackrel{po}{\rightarrow}$ is the program order, which for each thread t, is a total order over all actions performed by t in A,
- $\stackrel{so}{\rightarrow}$ 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 $u \stackrel{so}{\rightarrow} \ell$ then $u \stackrel{sw}{\rightarrow} \ell$.

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

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

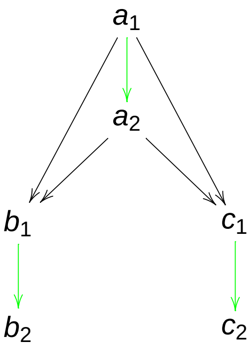
- closure(R) contains R,
  for all $x, y \in X$, if $x\ R\ y$ then x closure(R) y,
- closure(R) is reflexive
  for all $x \in X$, x closure(R) x,
- closure(R) is transitive
  for all $x, y\ z \in X$, if x closure(R) y and y closure(R) z then x closure(R) z.

The Happens-Before Order

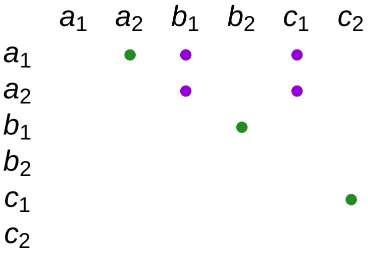
The happens-before order is defined in terms of the program order and the synchronizes-with order.

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

Example



Example



Example

	a_1	a_2	b_1	b_2	c_1	c_2
a_1	●	●	●	●	●	●
a_2		●	●	●	●	●
b_1			●	●		
b_2				●		
c_1					●	●
c_2						●

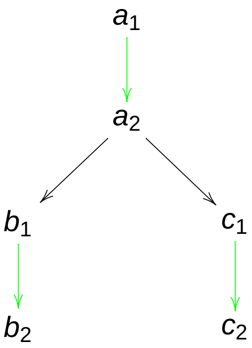
The Sufficient Synchronization Order

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

The sufficient synchronization order $\stackrel{ssw}{\rightarrow}$ is the smallest subset R of $\stackrel{sw}{\rightarrow}$ such that

$$\stackrel{hb}{\rightarrow} = \text{closure}(\stackrel{po}{\rightarrow} \cup R).$$

Example



Formal Specification of the JMM

An execution $\langle P, A, \stackrel{po}{\rightarrow}, \stackrel{so}{\rightarrow}, W, V \rangle$ is well-formed if the following conditions are true:

- Each read of a variable x sees a write to x. All reads and writes of volatile variables are volatile actions.
- The synchronization order is consistent with the program order.
- The execution obeys intra-thread consistency.
- The execution obeys synchronization-order consistency.
- The execution obeys happens-before consistency.

Formal Specification of the JMM

Each read of a variable x sees a write to x. All reads and writes of volatile variables are volatile actions.

For all reads $r \in A$, we have that $W(r) \in A$ and $W(r).v = r.v$. The variable $r.v$ is volatile if and only if r is a volatile read, and the variable $W(r).v$ is volatile if and only if $W(r)$ is a volatile write.

Recall that 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. Also recall that $a.v$ is the variable involved in action a.

Formal Specification of the JMM

The synchronization order is consistent with the program order.

For all actions $a_1\ a_2 \in A$, it is not the case that $a_1 \stackrel{po}{\rightarrow} a_2$ and $a_2 \stackrel{so}{\rightarrow} a_1$.

Formal Specification of the JMM

The execution obeys intra-thread consistency.

“For each thread t, the actions performed by t in A are the same as would be generated by that thread in program-order in isolation, with each write w writing the value $V(w)$, given that each read r sees/returns the value $V(W(r))$. Values seen by each read are determined by the memory model.”

Formal Specification of the JMM

The execution obeys synchronization-order consistency.

For each volatile read $r \in A$, it is not the case that $r \stackrel{so}{\to} W(r)$ and there does not exist a write w such that $w.v = r.v$ and $W(r) \stackrel{so}{\to} w \stackrel{so}{\to} r$.

Recall that 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. Also recall that $a.v$ is the variable involved in action a.

Formal Specification of the JMM

The execution obeys happens-before consistency.

For each read $r \in A$, it is not the case that $r \stackrel{hb}{\rightarrow} W(r)$ and there does not exist a write w such that $w.v = r.v$ and $W(r) \stackrel{hb}{\rightarrow} w \stackrel{hb}{\rightarrow} r$.

Recall that 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. Also recall that $a.v$ is the variable involved in action a.

Formal Specification of the JMM

An execution $\langle P, A, \stackrel{po}{\rightarrow}, \stackrel{so}{\rightarrow}, W, V \rangle$ satisfies the causality requirements if there exist

- sets of actions $C_0, C_1, \ldots$ such that
  - $C_0 = \emptyset$,
  - $C_i \subset C_{i+1}$,
  - $A = \bigcup_i C_i$,
- well-formed executions $\langle P_i, A_i, \stackrel{po_i}{\rightarrow}, \stackrel{so_i}{\rightarrow}, W, V \rangle$

such that ...

Formal Specification of the JMM

...

- $C_i \subseteq A_i$,
- $\stackrel{hb_i}{\rightarrow}$ and $\stackrel{hb}{\rightarrow}$ agree on C_i,
- $\stackrel{so_i}{\rightarrow}$ and $\stackrel{so}{\rightarrow}$ agree on C_i,
- V_i and V agree on C_i,
- W_i and W agree on C_i,
- for each read $r \in A_i \setminus C_{i-1}$, we have that $W_i(r) \stackrel{hb_i}{\rightarrow} r$,
- for each read $r \in C_i \setminus C_{i-1}$, we have that $W_i(r) \in C_{i-1}$ and $W(r) \in C_{i-1}$,
- for all actions $x, y, z \in A_i$, if $x \stackrel{ssw_i}{\rightarrow} y \stackrel{hb_i}{\rightarrow} z$ and $z \in C_i \setminus C_{i-1}$ then $x \stackrel{sw_j}{\rightarrow} y$ for all $j \geq i$,
- for all actions $x, y \in A_i$, if $y \in C_i$, x is an external action and $x \stackrel{hb_i}{\rightarrow} y$ then $x \in C_i$.