Concurrent Object-Oriented Languages

The Java Memory Model

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

Actions

We restrict our attention to interthread actions, that is, those you would see if you were standing at the interface between processor and memory.

Formal specification of actions

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, and
- u is an arbitrary unique identifier for the action.

Synchronization actions

Synchronization actions include

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

Executions

An executions consists of

- its actions
- for each thread, the program order of the actions, and
- the synchronization order of the synchronization actions.

Program order

The program order is a total order of the actions of a thread in an execution that reflects the order in which these actions in the code (I think).

Synchronization order

The synchronization order is a total order of the synchronization actions of an execution.

For the synchronization actions of a thread, the program and synchronization orders coincide.

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.

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.

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

Formal specification of executions

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,
- $\xrightarrow{sw}$ is the synchronizes-with order, a partial order over synchronization actions, and
- $\xrightarrow{hb}$ is the happens-before order.

Well-formed executions

An execution is well-formed if

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

Well-formed executions

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.

Well-formed executions

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.”

Well-formed executions

The execution obeys synchronization-order consistency.

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

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.

Well-formed executions

The execution obeys happens-before consistency.

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

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.

Causality requirements

An execution $\langle P, A, \xrightarrow{po}, \xrightarrow{so}, W, V \rangle$ 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 $\langle P_i, A_i, \xrightarrow{po_i}, \xrightarrow{so_i}, W, V \rangle$

such that ...

Causality requirements

...

- $C_i \subseteq A_i$,
- $\xrightarrow{hb_i}$ and $\xrightarrow{hb}$ agree on C_i,
- $\xrightarrow{so_i}$ and $\xrightarrow{so}$ 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 $W_i(r) \xrightarrow{hb_i} r$,
- 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 $x \xrightarrow{ssw_i} y \xrightarrow{hb_i} z$ and $z \in C_i \setminus C_{i-1}$ then $x \xrightarrow{sw_j} y$ for all $j \geq i$,
- for all actions $x, y \in A_i$, if $y \in C_i$, x is an external action and $x \xrightarrow{hb_i} y$ then $x \in C_i$.

Volatile

```
class Volatile
{
   private int value;
   private volatile boolean initialized;

   public void write(int value)
   {
      this.value = value;
      this.initialized = true;
   }

   public void use()
   {
      if (this.initialized)
      {
         // write has been invoked
      ...
   }
}
```