Informal Description of the new Manson/Pugh model

February 24, 2004, 2:46pm

Note: the issue of what it means for an action to occur in more than one execution is elided.

Each execution has a **synchronization order**, which is a total order over all synchronization actions in the execution. There is a **happens-before** relation $\stackrel{hb}{\rightarrow}$ defined on actions $i \stackrel{hb}{\rightarrow} j$ if i is before j in program order, if i is an unlock or volatile write and j is a matching lock or volatile read that comes after it in the total order over synchronization actions, or if $i \stackrel{hb}{\rightarrow} k \stackrel{hb}{\rightarrow} j$ for some k.

A read r is **allowed** to see a write w to the same variable v if r does not happen-before w and if there is no other write w' to v such that $w \stackrel{hb}{\rightarrow} w' \stackrel{hb}{\rightarrow} r$. A volatile read always sees the result of the most recent volatile write of the same variable in the synchronization order.

An execution that has only allowed reads and respects intra-thread semantics (see Appendix A) is a **happens-before consistent** execution, or **hb-consistent** for short.

For every execution, there is a total order over actions, consistent with the synchronization order, called the **justification order**. An execution is **weakly justified** if each read action sees a write that occurs earlier in the justification order.

A set of reads R is **directly responsible** for an action x occurring within a set of executions J if, for all executions $E \in J$, if $R \subseteq E$, x occurs in E and $R \stackrel{jo}{\rightarrow} x$. If x is a read, we do not require that R ensure that in all executions in J containing R, x sees the same value. Rather, we simply require that it be happens-before consistent for x to see the same value (possibly from a write that occurs later in the justification order). There may be more than one such set of reads, in which case any such set of reads may be used. Note that if x occurs in every execution in J, the empty set may be used as the set of actions responsible for x in J. For purposes of responsibility, a lock can be treated as a volatile read, seeing the previous matching unlock.

The actions **transitively responsible** for an action x occurring an execution E with respect to a set of executions J is a set of actions in E given by:

- The reads R directly responsible for x in J.
- The set W of writes seen by R in E.
- The actions transitively responsible for R and W in E with respect to J.

An action x is **prescient** if there exists an action y that occurs after x in the justification order such that $y \stackrel{hb}{\rightarrow} x$. Each prescient action x in an execution E must be justified. Let α be the sequence of actions that precedes x in the justification order of E. Let J be the set of all weakly justified executions whose justification order consists of α followed by non-prescient actions (see Appendix B for an algorithm to generate J). To justify that x can occur presciently after α, we must show an execution $E' \in J$ containing x. Furthermore, E and E' must be consistent according to three properties:

- **Responsible Actions Rule:** Let R be the set of actions transitively responsible for x in E' in the context of J. All of these actions must occur in E. While the reads responsible for x in E' do not have to see the same write or value as in E, it must be happens-before consistent for each read in R to see in E the same write it saw in E'.
- **Inertia Rule:** Each read y in E must occur identically in E' unless
  - It sees a write or value it could not see in E', or
  - There is an action z in E that does not occur identically, and z is before y in program order.
- **Prescient Write Rule:** If x is a prescient write, then for each thread t, let c be the number of reads in E' performed by t that conflict with x and happen-before x. At least c reads that conflict with x and happen-before x must be performed by t in E.

An execution is legal according to the Java Memory Model if it is weakly justified and each prescient action is justified.

Appendix

These appendices include clarifications that have been requested.

A Intra-thread Semantics

Given an execution where each read sees a write that it is *allowed* to see by the happens-before constraint, we verify that the execution respects intra-thread semantics as follows. For each thread t, we go through the actions of that thread in program order. For each non-read action x, we verify that the behavior of that action is what would follow from the previous actions in that thread according to the JLS/JVMS. For a read action, we only verify that the variable read is the one that is determined by the previous actions in the thread according to the JLS; the value seen by the read is determined by the memory model.

B Generating Non-prescient Extensions

Say we have a program P, and a partial justification order α. We can compute the set of all non-prescient extensions to α as follows.

- Let S be a set of partial and complete justification orders, initialized to be the singleton set containing α.
- Let W be a worklist of justification orders to be explored, initialized to S.
- While W is non-empty, choose and remove a justification order β from W
  - For each thread t in P, select the first statement in program order whose execution is not in β.
    - If that statement is not a read, then evaluate that statement in the thread-local context of β, generating action x, and add βx to both S and W.
    - If that statement is a read, determine, in the thread-local context of β, which variable v will be read. For each write $w \in \beta$ of v that could be seen by the read, generate the action r corresponding to that read seeing w, and add βr to both S and W.
- When W is empty, the complete justification orders in S corresponding to hb-consistent executions are the non-prescient extensions to α.