Almost One Page Informal Description of Manson/Pugh model

October 17, 2003, 11:24am

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

There is a **happens-before** relation $\xrightarrow{hb}$ defined on actions $i \xrightarrow{hb} j$ 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 $i \xrightarrow{hb} k \xrightarrow{hb} j$ 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 $w \xrightarrow{hb} w' \xrightarrow{hb} r$.

An execution that has only allowed reads and respects intra-thread semantics (see Appendix B) 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**.

Any read action must see a write that occurs earlier in the justification order. A volatile read always sees the result of the last volatile write in the justification order.

An action x is **prescient** if there exists an action y that occurs after x in the justification order such that $y \xrightarrow{hb} x$. Each prescient action x in an execution E must be justified by the actions that come before it in the justification order. Let α be the sequence of actions that precedes x in the justification order of E. Let J be the set of all non-forbidden hb-consistent executions whose justification order consists of α followed by non-prescient actions (see Appendix C for an algorithm to generate J). To prove x is justified, we need to show that for each E' in J it must have an action x' such that:

- x' is congruent to x; specifically, either x' and x are the same action, or they are both reads of the same variable and it would be hb-consistent for x' to see the write seen by x, and
- if x is a write, let R' be the set of all writes r' such that r' reads the same variable as x', r' is not in α and $r' \xrightarrow{hb'} x'$. There must be a corresponding congruent set R of reads in E, such that for all reads $r \in R$, r is not in α and $r \xrightarrow{hb} x$.

Prescient Relaxation Consider any execution E with justification order $\alpha xy\beta$ where:

- x and y are not both synchronization actions, and
- x is prescient, y is not.
- x is not a write seen by y.

Given this, the **prescient relaxation** of x in E gives an execution E' that is identical to E, except that the justification order of E' is $\alpha yx\beta$.

Forbidden Executions Justification may involve the use of forbidden executions. Forbidden executions are defined by a set of forbidden justification order prefixes F. Given F, an execution E is forbidden by F if any application of zero or more applications of prescient relaxation to E generates an execution trace whose justification order starts with a forbidden prefix (typically, F is empty and no executions are forbidden).

A set of forbidden prefixes must be valid. To show that a set of forbidden prefixes is valid, we must show that for each prefix $\alpha x \in F$, there exists some non-forbidden execution E with a justification order $\alpha\beta$ such that β contains no prescient actions.

Valid Executions Given these definitions, an hb-consistent execution E is legal if and only if there exists a set of forbidden prefixes F_E such that E is not forbidden by F_E and using F_E as the forbidden prefixes, all of the prescient actions in E are justified.

Appendix

These appendices include clarifications that have been requested.

A Differences with Old Model

Here is a brief rundown on the differences between the new model and the model in the community review draft.

- Consistency is now called *hb-consistency*.
- Previously, we allowed a prescient read action to see a write that occurs later in the justification order.

  Now all reads must see writes that occur earlier in the justification order.
- A write w cannot occur presciently if in the justifying execution there is a conflicting read r such that $r \stackrel{hb}{\rightarrow} w$.
- Forbidden sets are defined in a slightly different way. In particular, they are global, so that in order to justify an action x in an execution E, you may not forbid E.

B 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.

C 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.