One Page Informal Description of Manson/Pugh model

Note: we are going to gloss-over the issue of what it means for a action x in one execution to also occur in another 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 $w \stackrel{hb}{\rightarrow} w' \stackrel{hb}{\rightarrow} r$. An execution that has only allowed reads and respects intra-thread semantics (see Appendix A) is a *consistent* execution.

For every execution, there is a total order over actions called the *causal order*. An action x is *prescient* if either:

- x is a read that sees a write that occurs later in the causal order, or
- there exists an action y that occurs after x in the causal order and $y \stackrel{hb}{\rightarrow} x$.

Each prescient action x in an execution E must be justified. Let α be the sequence of actions that precedes x in the causal order of E. Let J be the set of all consistent executions whose causal order consists of α followed by non-prescient actions (see Appendix B for an algorithm to generate J). To prove x is justified, we need demonstrate a set of prohibited executions $P \subset J$ such that the following is true (P may be, and often is, empty):

- If x is not a read, then we need to show that x occurs in each $E' \in J - P$.
- If x is a read, then we need to show that x occurs in each $E' \in J - P$, with the caveat that x may see different values in E and E'. If different values are seen by x in E and E', there must be a write $w \in E'$ such that w writes the value seen by x in E and x would be allowed to see w in E' by the happens before constraint.
- If the set of prohibited executions P is non-empty, we must show that P is valid. For each prohibited execution $E' \in P$, we must identify a read r in E', but not in α, that is allowed by the happens before constraint to see multiple writes. Assume that in E' the read r sees a write w. Then there must exist a different write w' that could be seen by r in E' and an execution $E'' \in J - P$ such that
  - r and w' occur in E'', and in E'', the read r sees w'.
  - E'' includes all of the actions that come before r in the causal order of E'.

  Note that since r represents a choice at which E' and E'' diverge, the remainder of their executions can be substantially different.

Justification of prescient actions is needed in order to prevent unacceptable behaviors that are instances of causal loops. Prohibited executions are needed because compiler optimizations or processor memory model limitations can restrict possible executions, allowing other behaviors to be guaranteed and therefore performed presciently.

Appendix

These appendices include clarifications that were requested by Victor Luchangco. More to come as people request them.

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 causal 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 causal orders, initialized to be the singleton set containing α.
- Let W be a worklist of causal orders to be explored, initialized to S.
- While W is non-empty, choose and remove a causal 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 causal orders in S corresponding to consistent executions are the non-prescient extensions to α.