

## The Full Formal SC- Memory Model for Java

(see <http://www.cs.uiuc.edu/~sadve/jmm/sc-.pdf> for detailed motivation)

**Program:** When we refer to a program, we also include an input set (i.e., the same code with a different input is a different program).

**Execution E of a Java program P has the following properties:**

- E specifies a set of read and write memory accesses in a total order, called the **serialization order**. Each write specifies a thread, an address, and a value. Each read specifies a thread, an address, and a write in the execution that is to the same address and whose value the read returns. The execution order restricted to accesses of the same thread is called the **program order**.
- The memory accesses and program order of an execution must represent a correct uniprocessor. That is, a correct uniprocessor running the program P for thread T would generate the accesses of thread T in E with the same program order as E, with the assumption that the reads in the uniprocessor return the same values as in E (e.g., from a memory system that magically generates these values).
- E may specify some addresses as volatiles or monitors. All accesses to these addresses are called **synchronization accesses** and others are called **data**.<sup>1</sup>
- E contains a special (*hypothetical*) **initialization** thread that writes initial values to all addresses, followed by a write to a unique volatile address. For all other threads, the first access (by program order) is a (*hypothetical*) read to this unique address that returns the value of the above write.

**Conflicting accesses:** Two accesses to the same address where at least one of them is a write.

**Non-SC read:** A read is non-SC in an execution E if it does not return the value of the conflicting write ordered last before it by the serialization order of E.

**Happens-before relation (hb),** defined on memory accesses of an execution:  $I \text{ hb } J$  if (1) I is before J by program order, or (2) I is a synchronization write, J is a synchronization read, and J returns the value of I, or (3)  $I \text{ hb } K \text{ hb } J$  for some K.

**Data race:** Two conflicting memory accesses form a data race if they are not ordered by hb.

**Hb-consistency:** A write W is hb-consistent for a conflicting read R in execution E if R is not before W by hb in E and if there is no conflicting write  $W'$  such that  $W \text{ hb } W' \text{ hb } R$  in E.

**Race-consistency:** A write W is race-consistent for a conflicting read R in E if (1) W is hb-consistent for R in E and (2) if R is a synchronization access, then W is the last conflicting write before R in E's serialization order.

**An execution E obeys SC-** if all its reads can be validated in some total order. A read  $R'$  is validated for E if there exists an execution  $E'$  such that the following holds:

- All non-SC reads in  $E'$  are data reads that occur in E, have been previously validated for E, and return the value of the same write in  $E'$  and E.
- Let R be  $R'$  or a previously validated read in E. Let R return the value of W in E. Then
  - R and W occur in  $E'$  and W is race-consistent for R in E and  $E'$ .
  - Either W and R form a data race in both E and  $E'$  or  $W \text{ hb } R$  in both E and  $E'$ .

Above, when considering two executions  $E_1$  and  $E_2$  of program P, we say an **access  $A_1$  in  $E_1$  occurs in  $E_2$  or is the same as access  $A_2$  in  $E_2$**  if  $A_1$  is matched with  $A_2$  as follows. We can match two reads or two writes with each other if the set of all matched accesses (for  $E_1$  or  $E_2$ ) is a *maximal* set that obeys the following properties:

- Two accesses matched with each other must be from the same thread, access the same address, and write the same value (if they are writes).
- The sets of matched accesses should have the same program order relation in  $E_1$  and  $E_2$ ; i.e., if  $A_1$  and  $B_1$  in  $E_1$  are respectively matched with  $A_2$  and  $B_2$  in  $E_2$ , then  $A_1 \text{ po } B_1$  in  $E_1$  implies that  $A_2 \text{ po } B_2$  in  $E_2$ .

---

<sup>1</sup> There are constraints on monitor accesses outside of those specified in the memory model (e.g., a lock must be held before it is freed, etc.). These can be integrated here as well.