

Proof that SC- guarantees SC to data-race-free programs

1/7/2004 3:15 AM

Theorem: Consider a program P that is data-race-free. Then an execution E of program P that obeys SC- must be E SC.

Proof: We say an access X is before (respectively after) an access Y in an execution if X is ordered before (respectively after) Y by the serialization order of the execution. Similarly, we use “last” and “first” to implicitly refer to the serialization order.

We first prove some simple lemmas.

Lemma 1. A synchronization read in E returns the value of the last conflicting write ordered before it in E.

Proof: A validated synchronization read must return the value of a write that is race-consistent for it in E. By definition, this is the last conflicting write ordered before the read in E.

Lemma 2. Let E_v be an execution that is used to validate some read in E. Then a synchronization read in E_v returns the value of the last conflicting write ordered before it in E_v .

Proof: Only data reads can return non-SC values in E_v . Therefore, by definition, a synchronization read must return the value of the last conflicting write ordered before it in E_v .

Lemma 3. Let E_v be an execution that is used to validate some read in E. Then if $X \text{ hb } Y$ in E_v , then X is before Y in E_v . Similarly, if $X \text{ hb } Y$ in E, then X is before Y in E.

Proof: Recall that $I \text{ hb } J$ if (1) I is before J by program order, or (2) I is a synchronization write, J is a synchronization read, J returns the value written by I, or (3) $I \text{ hb } K \text{ hb } J$ for some K.

We refer to the first type of edge as a **po edge** and the second type as a **synch edge** (for synchronization order).

Thus, if $X \text{ hb } Y$ in E_v due to a po edge, then X is before Y in E_v since the serialization order of E_v is consistent with program order by definition. If $X \text{ hb } Y$ in E_v due to a synch edge, then again X is before Y in E_v by Lemma 2. If $X \text{ hb } Y$ due to the transitive closure of po and synch edges, then transitively applying the above observations, again X is before Y in E_v .

A similar argument using Lemma 1 shows that if $X \text{ hb } Y$ in E, then X is before Y in E.

Lemma 4. Let E_v be an execution that is used to validate some read in E. Let R' be a read that was previously validated or is validated by E_v . Then if R' is a synchronization read, then R' returns the same value in E and E_v .

Proof: The write whose value R' returns in E must be race-consistent for E_v . Thus, this write must be the last conflicting write before R' in E_v . It follows from Lemma 2 that R' returns the value of this write in E_v .

Let E_{vl} be the execution used to validate the last read in E. E_{vl} contains all reads of E. There are two cases.

Case 1. E_{vl} does not have a non-SC read.

Then E_{vl} is an SC execution and cannot have data races. If all reads in E_{vl} return the same value as in E, then E is SC and we are done.

So assume for a contradiction that all reads in E_{vl} do not return the same value as in E. Then there must be some read in E_{vl} that occurs in both E and E_{vl} and returns a different value. Let R be the first such read in E_{vl} (according to its serialization order).

By Lemma 4, R must be a data read. Let R return the value of W in E. Since W is hb-consistent for R in E_{vl} , it follows that we cannot have $R \text{ hb } W$ in E_{vl} . Since E_{vl} does not have data races, it must be that $W \text{ hb } R$ in E_{vl} . By Lemma 3, W must be before R in E_{vl} .

Let W_v be the write whose value R returns in E_v . Then since E_v does not have non-SC reads, W_v is the last conflicting write before R in E_v and since E_v does not have data races, $W_v \text{ hb } R$. So it must be that W is before W_v (since W is before R) in E_v and since there are no data races in E_v , $W \text{ hb } W_v$ in E_v . But then we have $W \text{ hb } W_v \text{ hb } R$ in E_v . This contradicts the requirement that W is hb-consistent for R in E_v .

Case 2. E_v has a non-SC read.

Consider the first execution used for validation that has a non-SC read. Call it E_v . Consider the first non-SC read R in E_v . R returns the value of the same write W in E_v and E . Consider the execution E_{vr} that was used to validate R . E_{vr} cannot have any non-SC reads (because E_v is the first such execution used for validation). So E_{vr} is SC and so has no data races. So $W \text{ hb } R$ in E_{vr} . So $W \text{ hb } R$ in E . So $W \text{ hb } R$ in E_v as well. By the previous lemma, W is before R in E_v .

Since R is non-SC in E_v , there must be a conflicting write W' between W and R in E_v . Either W and W' form a data race in E_v or W' and R form a data race in E_v , since W is hb-consistent for R . Since there are no non-SC reads before R , we can take the execution E_v until R , make R read the value of the last conflicting write before it, and continue the execution without any SC reads. This is an SC execution of program P that has a data race (between W and W' or between W' and R). This is a contradiction.