Proof sketch that Manson/Pugh allows reordering

September 3, 2003, 8:16am

Consider a program P and the program P' that is obtained from P by reordering two adjacent statements x and y. Let x be the statement that comes before y in P, and after y in P'. The statements x and y may be any two statements such that

- reordering x and y doesn't eliminate any transitive happens-before edges (it will reverse the direct happens-before edge between x and y),
- x and y are not conflicting accesses to the same variable,
- x and y are not both synchronization actions, and
- the intra-thread semantics of x and y allow reordering (e.g., x doesn't store into a register that is read by y).

Assume that we have a valid execution E' of program P'. To show that the transformation of P into P' is legal, we need to show that there is a valid execution E of P that has the same observable behavior as E'.

Assume $E' = \langle S, so, hb', co\rangle$. We are going to show that $E = \langle S, so, hb, co\rangle$ is also a valid execution of P. Let a_x and a_y denote the actions corresponding to the statements x and y. Because of the reordering the happens-before ordering may be different but we know that $hb - \{a_x \rightarrow a_y\} \subseteq hb' - \{a_y \rightarrow a_x\}$. Clearly, if E' is consistent then E is consistent, so we only need to worry about showing that co is justified as the causal order of E.

- Assume that $co = \alpha a_y \beta a_x \gamma$.

  We don't need to worry about any actions that were prescient in E'. The justification of those prescient actions in E' will also justify them in E.

  The only action that could be prescient in E but not E' is a_y. If a_y is not prescient in E', we know a_x is the only action that comes after a_y in the causal order such that $a_x \xrightarrow{hb} a_y$. We also know that if a_y is a read, it sees a value in α. In order to justify the prescient action a_y in E, we need to show that there is an execution of P whose causal order starts with α in which the action a_y is allowed to occur and, if a_y is a read, that a_y can see a write of the value seen by a_y in E.

  We know intra-thread semantics will cause a_y to occur, since all actions other than a_x that occur before a_y in program order are in α, and we have assumed as a condition for the reordering that a_x does not affect the intra-thread semantics of a_y.

  If a_y is a read that sees a write w in α, we know it can see w in any execution with a causal order that starts with α; there is therefore some execution in which it will see w.

  If a_y is a write, then all read actions that conflict with it and happen before it must be in α. Since a_y does not induce any interthread happens-before orderings, the only way for an action b to happen before a_y is if b happens-before an action that occurs earlier than a_y in program order. The action b cannot be a_x, because a_x and a_y do not conflict. The action b can also not be in β, because then the reversal of a_x and a_y would have removed the transitive happens-before edge between them. The action b must, therefore, still be in α.

- Alternatively, assume that $co = \alpha a_x \beta a_y \gamma$. Then any action in E that is prescient is also prescient in E', and the justifications used to show that those actions are justified in E' will also show that those actions are justified in E.

Proof Sketch that Model Allows Unrolling / Merging

Compiler transformations can take place that take code that executes along one control path, and split that path so it executes along multiple control paths that are equivalent to the original. Conversely, it can take code that executes along multiple control paths, and merge these paths so it only executes along one control path.

Consider a program P, and a program P' that is generated by splitting or merging. Is it the case that every execution E' of P' has a corresponding execution E in P?

All forms of splitting and merging control paths must preserve intra-thread semantics. It is therefore only the inter-thread actions that may be affected by splitting and merging. Because such actions do not need to correspond to program statements, other than that they must obey the intra-thread semantics of the program, any execution E' of P' will have the same actions of an execution E of P.

Correctly Synchronized Programs exhibit only SC Behaviors

We say an execution has *sequentially consistent* (SC) results if its results are the same as if the actions of all the thread were executed in some sequential order, and the actions of each individual thread appear in this sequence in program order.

Two memory accesses are *conflicting* if they access the same variable and one or both of them are writes. A program is defined to be *correctly synchronized* (CS) if in all sequentially consistent executions, any two conflicting accesses are ordered by a happens-before path.

A causal order will return a non-SC result if a read returns a value of a write that does not happen before that read.

Lemma 1 *If an execution E has a non-prescient causal order and all conflicting accesses are ordered by happens-before edges, E has sequentially consistent behavior.*

Proof: Assume we have an execution E with a non-prescient causal order *co*. Since *co* is non-prescient, the ordering of the actions in *co* is an valid sequentially consistent execution order. The only way the execution could not be sequentially consistent is if a read of a variable v, rather than seeing the most recent write to v, sees an earlier write to v. But all of those accesses are ordered by happens before edges, so only the most recent write to v is visible. So only sequentially consistent behaviors are allowed. $\square$

Definition 1 *A program is* **correctly synchronized** *if and only if in all sequentially consistent executions, all conflicting accesses are ordered by happens-before edges.*

Lemma 2 *In all non-prescient executions of correctly synchronized programs, all conflicting accesses are ordered by happens-before edges.*

Proof: By contradiction. Assume there is a non-prescient execution with a causal order $\alpha x \beta$, where x is the first action on the causal order that is not correctly synchronized with respect to conflict actions that occur before it in the causal order.

Let x' be an action congruent to x; if x is a read of a variable v, then x' sees the last write to v in α.

By executing additional actions β' according to sequentially consistent semantics, we arrive at a sequentially consistent execution with a causal order $\alpha x' \beta'$ that has a data race. So this program must not have been correctly synchronized.

Lemma 3 *All non-prescient executions of correctly synchronized programs exhibit sequentially consistent behavior.*

Proof: Follows directly from Lemmas 2 and 1.

Definition 2 *Let E be an execution with a causal order $\alpha x y \beta$ such that x is prescient, y is not prescient, and y is not a read that sees x and x and y are not both synchronization actions. Then $\alpha y x \beta$ is the causal order of an execution E' that is the* **prescient relaxation** *of x in E. Note that E and E' have the same actions, behavior, synchronization order and happens-before edges.*

Definition 3 *Given any execution E, the* **full prescient relaxation** *of E is an execution E' that is obtained by applying the reordering described in Definition 2 repeatedly until no more reorderings are possible.*

Lemma 4 *For any prefix α of a causal order of an execution of a correctly synchronized program, full prescient relaxation of any non-prescient extension of α gives an equivalent execution with no prescient actions.*

Proof: Our proof is by induction; the inductive hypothesis is that for every execution E that is a non-prescient extension of α (i.e., $E \in NPE(\alpha)$), full prescient relaxation of E gives an execution with no prescient actions.

Base Case In the base case, $\alpha = \emptyset$. All non-prescient extensions of $\emptyset$ are non-prescient. □

Inductive Case Given the inductive hypothesis, we must show that for every execution $E \in NPE(\alpha x)$, the full prescient relaxation of E is entirely non-prescient.

Every $E \in NPE(\alpha x)$ has a causal order $\alpha x \beta$, where β is non-prescient. Repeatedly apply prescient relaxation to x in E.

- If this makes x non-prescient, then the resulting execution has a causal order that is a non-prescient extensions of α, and our inductive hypothesis tells us that full prescient relaxation of E gives an entirely non-prescient execution.
- Alternatively, after relaxing x zero or more times, we have a causal order $\alpha\beta' xy\gamma$, where x is prescient and cannot be further relaxed, and y, β' and γ are non-prescient.
  - It cannot be the case that both x and y are synchronization actions, since then the causal order of E wouldn't respect the synchronization order of E.
  - Then it must be the case that y is a read r that sees the write x.

    If $x \xrightarrow{hb} r$, then r would also be prescient. So we know there is no happens-before edge from x to r.

    There must be a non-forbidden $E' \in NPE(\alpha\beta')$ that includes x; otherwise, it would not be possible to include it as the next action in the causal order.

    The action x does not influence whether r occurs or the variable it reads; it only influences the value it sees. All non-prescient extensions of $\alpha\beta'$ must therefore contain a read corresponding to r, reading the same variable, but possibly seeing a different value. Let r' be this corresponding read in E', and let E'' be the full prescient relaxation of E'.

    Because β' is non-prescient, we know that E' is a non-prescient extension of α. Therefore, by the inductive hypothesis, E'' must be entirely non-prescient. By Lemma 2, all conflicting accesses are ordered by happens-before edges in both E'' and E'.

    Because E' is correctly synchronized, r' and x must be ordered by happens-before edges in E':

- $x \xrightarrow{hb} r'$: Since r' is in all non-prescient extensions of $\alpha\beta'$, and can occur directly after $\alpha\beta'$, the only actions that have happen before edges leading to r' are in $\alpha\beta'$. Since x is not in $\alpha\beta'$, we cannot have $x \xrightarrow{hb} r'$.
- $r' \xrightarrow{hb} x$: Since a write cannot occur presciently if there exists a non-forbidden NPE in which a conflicting read happens before the write to be performed presciently, x would not be allowed to occur presciently in E. Since it did, it must not be the case that $r' \xrightarrow{hb} x$.

Therefore, our inductive hypothesis holds.

Theorem 5 *Correctly synchronized programs have sequentially consistent semantics.*

Proof: All executions of correctly synchronized programs are equivalent to non-prescient executions of the same program because of Lemma 4. Therefore, by Lemma 3, all executions of correctly synchronized programs are sequentially consistent. □