EE 382C/361C: Multicore Computing **Fall 2016**

Lecture 11: September 29

Lecturer: Vijay Garg *Scribe: William Mills*

11.1 Exam 1 Study Tips

Exam 1 will only cover up-through Lecture 11 (today) There will be a problem from the seminar Some sample problems have been posted on Canvas. It is advised that you follow this order in your studying:

1. Attempt the problems on your own
2. Compare your solutions with your peers
3. Solutions will be posted next week to check for correctness

Additionally, Homework 3 has been posted, and some of its questions will be good preparation (but some of it has not been covered yet) The material to best-prepare you, in descending order:

1. Material covered in both Lecture and Handouts
2. Material covered in Lecture only
3. Material covered in Handouts only
4. Past Assignments

11.2 Concurrent History Equivalence

Recall that a concurrent History $(H, <_H)$ is a sequential history $(S, <_S)$ if $<_S$ is a total order. Two histories are *equivalent* if they are composed of the same operations.

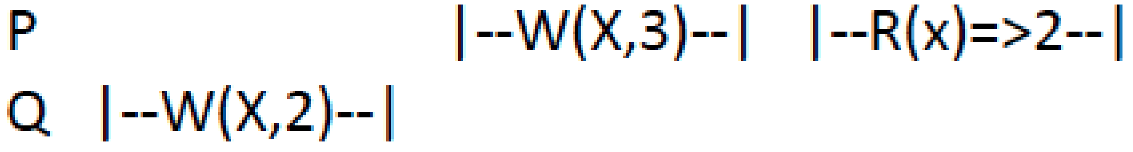

Figure 11.1: H1

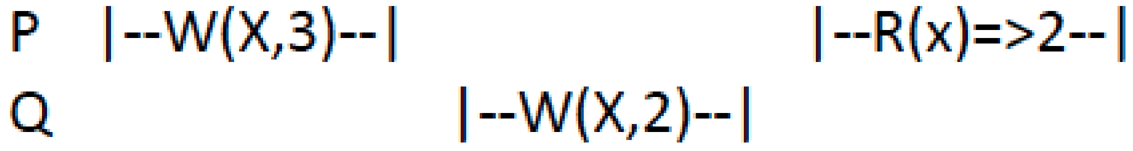

Figure 11.2: H2

Observe that `H2` is simply a permutation of the operations in `H1`, but `H1` is not identical to `H2`. These two histories are equivalent.

11.3 Sequential Consistency

A history $(H, <_H)$ is *sequentially consistent* if there exists a legal sequential history `S` such that:

1. `S` is equivalent to `H`
2. `S` preserves the process order in $(H, <_H)$

Both `H1` and `H2` from the previous section are sequentially consistent. A legal sequential history for either of them would be

$P : W(X, 3), Q : W(X, 2), P : R(X) => 2$

Now consider History `H3`:

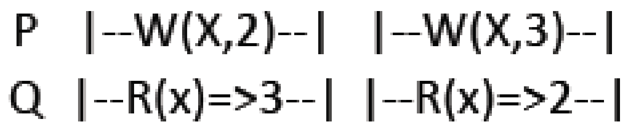

Figure 11.3: H3

H3 is NOT sequentially consistent. This can be shown by adding arrows that represent the *happened-before* relationship. When a cycle is identified, we know the history is not legal and thus cannot be sequentially consistent. When we place these edges in H3, we can see that a cycle exists when we follow the *happened-before* relationship:

$P : W(X, 2), \underline{Q : R(X) => 2, PW(X, 3), Q : R(X) => 3, Q : R(X) => 2}$

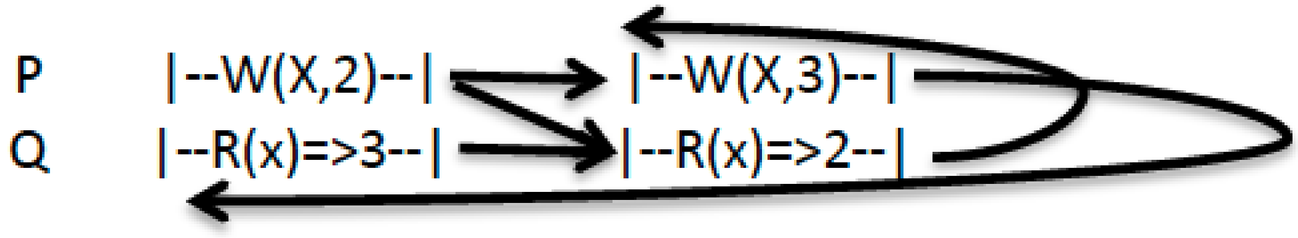

Figure 11.4: H3

11.4 Atomic Consistency (Linearization)

A history $(H, <_H)$ is linearizable/atomically consistent if there exists a legal sequential history S such that

1. S is equivalent to H
2. S preserves $<_H$
   - That is, responses occurring before invocations in H must preserve that relationship in S

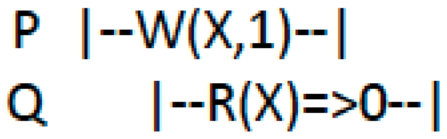

Figure 11.5: H4

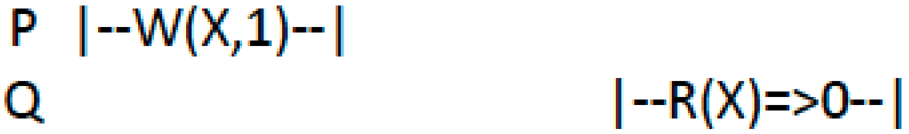

Figure 11.6: H5

Thus, H4 is linearizable, while H5 is not.

11.4.1 Linearization Points

To help determine if a history is linearizable, We can think of each operation having a single moment where its effect takes place, and try to find an order for them. We call such moments *linearization points.* Theorem: `H` is linearizable if there exist linearization points for all operations such that the resulting sequence is legal.

11.5 Composability (Locality Property)

Linearization is arguably a better tool for software applications, because it has the *locality property*, which says the concurrent history is legal when multiple objects are considered.

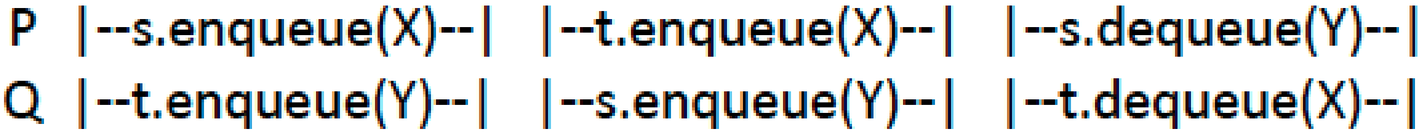
P |--s.enqueue(X)--| |--t.enqueue(X)--| |--s.dequeue(Y)--|
Q |--t.enqueue(Y)--| |--s.enqueue(Y)--| |--t.dequeue(X)--|

Figure 11.7: H7

Note that `H7` is linearizable when all of object t's operations are ignored. The same holds when all of s's operations are ignored. However, when the two are combined, we see that there is a cycle the same way we did before.

$P : s.enqueue(X), t.enqueue(X), t.enqueue(Y), s.enqueue(Y), s.enqueue(X)$

When a history `H` is projected on an object x, we write the sub-history as $H|x$. Note that $H|s$ and $H|t$ are both sequentially consistent, but `H` is not, because there is a cycle in its $<_H$ trace.

11.5.1 Linearization and the Locality Property

By definition, a consistency condition `CONSISTENCY` satisfies composability if:

$\forall x, H|xsatisfies$`CONSISTENCY` $\Leftrightarrow Hsatisfies$`CONSISTENCY`

Linearizability can be shown to satisfy composability from this definition. It is trivial to show that $Hislinearizable \rightarrow \forall x, H|xislinearizable$, so we will focus on the proof for the other direction.

As before, we will place arrows on our history to represent the happened-before relation. The claim is that adding these arrows will maintain an acyclic representation of the history. These arrows can be drawn for two reasons:

1. $<_H$ enforces it
2. $<_x \forall x$

We know arrows from `1` cannot form a cycle because they are enforced by the poset they come from. It remains to be shown that traces including both types of arrows cannot introduce a cycle. We assert that such subpaths including `1` $\rightarrow$ `2` $\rightarrow$ `1` can be re-written as a single arrow of type `1`.

$e <_H f$ – response(e) happened before invocation(f). $f <_x g$ – response(e) happened before invocation(f). $g <_H h$ – response(e) happened before invocation(f).

Transitively, we know that $c <_H h$ holds.

11.6 All Concurrent Histories

There are many other types of histories looser than sequential consistency that we did not discuss. Realize that the weaker your consistency rules are, the better performance you can achieve with the same hardware.

For this class, we will stick to atomic consistency

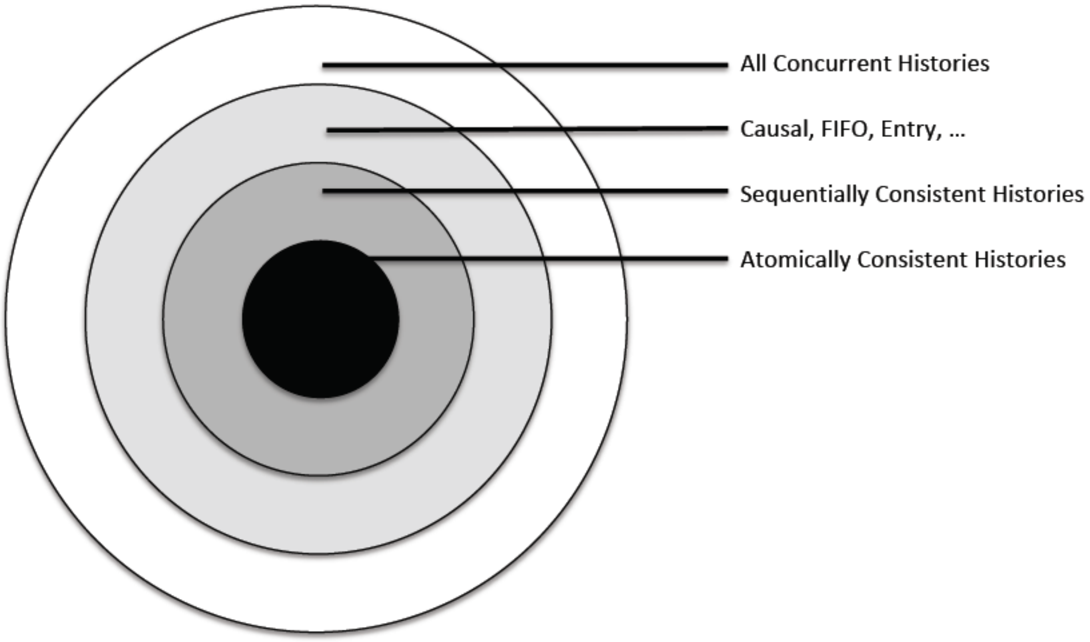
All Concurrent Histories
Causal, FIFO, Entry, ...
Sequentially Consistent Histories
Atomically Consistent Histories

Figure 11.7: Concurrent History Strengths

References

[1] V. K. Garg, Introduction to Multicore Computing