Distributed Systems HW3

Samuel Alaniz, Juan Neri

Fall 2022

1 Offline Predicate Detection

The intuition behind our proposal is that the predicate $B = (x_1 < x_2) \wedge (x_3 < 20)$ can be turned into a conjunction of local predicates if x_1 is fixed (e.g. $B = (X < x_2) \wedge (x_3 < 20)$). For each value of x_1, the given conjunction can be solved in $O(n)$, resulting in an overall worst case of $O(n^2)$ computations.

- To construct each trace, process P_i should keep a vector clock. Before each send or receive event (or end of computation), it should store the best value of x_i seen in the past state interval alongside its vector clock c_i to the trace.
  - For P_1 the best value of x_1 would be the smallest seen in the interval
  - For P_2 the best value of x_2 would be the largest seen in the interval
  - For P_3 the best value of x_3 would be the smallest seen in the interval
- The checker process will iterate over the values of P_1's trace. For each (x_{1_i}, c_{1_i}) do:
  1. Find the values in P_2 and P_3 that are concurrent with c_{1_i}
  2. Get the first pair of values from P_2 and P_3: (x_{2_0}, c_{2_0}), (x_{3_0}, c_{3_0})
  3. If c_{2_0} and c_{3_0} are not concurrent, advance the trace that is behind (pick the next value from that trace until they are concurrent or you run out of values).
  4. If you run out of values in P_2 or P_3, proceed to the next value of P_1
  5. Evaluate the predicate $(x_{1_i} < x_2) \wedge (x_3 < 20)$. If the predicate is true, you are done, otherwise advance the trace that makes the predicate false
  6. Repeat steps 3, 4, 5 until the predicate either becomes true, or you run out of values in P_1's trace (which would mean that the predicate never became true during the computation.

The time complexity for the checker process is $O(m^2)$ where each process has at most m state intervals. This is the case because there is one value of x_1 for each state interval in P_1 $(O(m))$, and for each of those values we advance through at most each of the values of x_2 and x_2 exactly once $(O(2m))$ resulting in a $O(m^2)$ computation.

2 Centralized Algorithm Theorem

We show that $\exists s_1, ..., s_m : wcp(s_1, ..., s_m) \iff < \exists s'_1, ..., s'_m : wcp(s'_1, ..., s'_m) \land \forall i : 1 \leq i \leq m : first(s'_i) >$

- First we show ($\Leftarrow$)

Assume that $< \exists s'_1, ..., s'_m : wcp(s'_1, ..., s'_m) \land \forall i : 1 \leq i \leq m : first(s'_i) >$. By the definition of logical and we have that $\exists s'_1, ..., s'_m : wcp(s'_1, ..., s'_m)$. By construction each s'_i is a state where the local predicate became true. Therefore this constitutes an example where WCP became true and thus its existance shows that $\exists s_1, ..., s_m : wcp(s_1, ..., s_m)$

- Next we show ($\Rightarrow$)

Assume that $\exists s_1, ..., s_m : wcp(s_1, ..., s_m)$. Let s'_1 be the first state in P_1 where the local predicate became true since the most recent sent message or the beginning of the trace. We know that such s'_1 exists from our hypothesis (since s_1 exists), and we also know by construction that s'_1 satisfies $first(s'_1)$.

Additionally, since $wcp(s_1, ..., s_m)$ then we know that $\forall i : s_1 \not\rightarrow s_i$, and since s'_1 satisfies $first(s'_1)$ these two conditions imply that $\forall i : s'_1 \not\rightarrow s_i$. Similarly since $\forall i : s_i \not\rightarrow s_1$ and $s'_1 \rightarrow s_1$ then $\forall i : s_i \not\rightarrow s'_1$. Thus s'_1 is concurrent with all $s_i : i \geq 2$ and thus $wcp(s'_1, s_2, ..., s_m)$ holds.

The above two claims mean that $\exists s'_1, s_2, ..., s_m : wcp(s'_1, s_2, ..., s_m) \land first(s'_1)$. The same proof is applicable for all other $s_i : i \geq 2$, and thus we get that $< \exists s'_1, ..., s'_m : wcp(s'_1, ..., s'_m) \land \forall i : 1 \leq i \leq m : first(s'_i) >$

3 Leader Election on a Torus

```
Pi ::
 var
     myid : integer;
     awake : boolean initially false;
     leaderid : integer initially null;
     leaderList : list of integers initially empty;

 To initiate election:
     send (election, myid) to east
     awake := true;

 Upon receiving a message (election, j):
     if (j > myid) then
         send (election, j) to east
     else if (j = myid) then
         send (leader, myid) to east;
     else if ((j < myid) ∧ ¬awake) then
         send (election, myid) to east;
     awake := true;

 Upon receiving a (leader, j) from west:
     if (j ≠ myid) then send(leader, j) to east;
     leaderid := max(leaderList, j);
     send(leader, eaderid) to north;

 Upon receiving a (leader, j) from south:
     if (leaderid == null) then
         leaderList.append(leader, j)
     else if (leaderid < j) then
         leaderid := j AND send(leader, j) to north
```

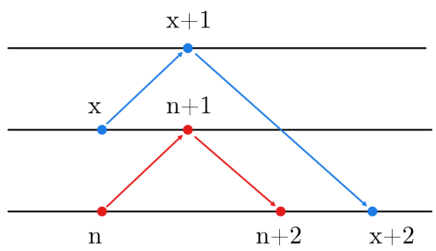
Figure 1: Torus Leader Election Algo

Note:

- This a tweak on the Chang–Roberts Algorithm.
- First tweak, when receiving a leader message, always broadcast that message to a north node.
- Second tweak, when receiving a leader message from a south node, if the id is greater than the current leader id, then make the new greater id the leader. Then pass that new leader id along to a northern node.

4 Causal Order as a Mapping

We show by example that there exists an $(E, \rightarrow)$ that is causally ordered, but it is not possible to build a mapping that satisfies both conditions.



First note that the events presented in the diagram follows causal order.

However it is not possible to generate a mapping that satisfies the requirements. First without loss of generality we assign variable integers to each event such that they satisfy rule (2). However, when we include the constraint (1) we get the following contradiction:

- $x < (n + 1) \implies x \leq n$
- $(x + 2) > (n + 2) \implies x > n$

Clearly both statements cannot be true at the same time, therefore no mapping exists that satisfies both (1) and (2) for the given causally ordered computation.

5 Causal Order Properties

5.1 Proof $C_1 \implies C_2$

Assume condition C_1 holds

- $s_1 \to s_2 \implies \neg(r_2 \to r_1)$

Now let $s_1 \prec s_2$, by definition this implies $s_1 \to s_2$ which by C_1 implies $\neg(r_2 \to r_1)$

5.2 Proof $C_2 \implies C_1$

Assume condition C_2 holds

- $s_1 \prec s_2 \implies \neg(r_2 \to r_1)$

Now let $s_1 \to s_2$. The fact that the events are comparable means that there must exist some s_x in the same process as s_1 such that $s_1 \to s_x$ (and also $s_x \to r_x$ and $r_x \to s_2$)

By C_2 this implies that (1) $\neg(r_x \to r_1)$

By contstruction $r_x \to s_2$ and $s_2 \to r_2$ then we cannot have $r_2 \to r_1$ since that would mean $(r_x \to r_1)$ and go against (1)

Therefore $\neg(r_2 \to r_1)$