Midterm Exam

CSc72010

Each question counts 20 points.

1. Given an automaton with an output "parent(x:NodeID)" and a network graph connecting the automata. For each possible execution of the network of automata, consider another graph whose nodes correspond to process automata and whose edges are pairs (m, n) where node m outputs parent(n) at some point in the execution. You can assume that each automaton has a distinct node ID.

   What is the safety property that says that this graph is a tree? State it formally, as a trace property. Does this property hold for the attached BFS algorithm? If so, state and prove an invariant assertion on the states of a network of BFS automata, from which you can prove the safety property.

   Safety property:
   Informally, there are no cycles in the graph described by the parent edges.
   Formally, the set of traces such that the set of parent messages in the trace does not contain any subset of the form { $parent_{m1}(m2)$, $parent_{m2}(m3)$, ..., $parent_{mn}(m1)$ }. Note that this subsequence implies a cycle in the graph.

   Invariant assertion:
   Denote the states by s_1, s_2, ..., s_k, ... Let myparent(k,m) = $s_k.myparent_m$ denote the value of myparent in state s_k of automaton m.
   Then in every state s_k, $s_k.myparent_m$ has either 0 or 1 members; if $s_k.myparent$!= $s_{k-1}.myparent$, then $s_{k-1}.myparent$ ={}; and all sequences of nodes n_1, n_2, ... such that $n_{i+1} \in myparent(k, n_i)$ terminate (i.e., there are no cycles created by parent edges).

   Proof: By induction on k.
   Induction base: State s_0 is the start state. Since $myparent_m$ is initialized to {}, the assertion holds.
   Induction hypothesis: Suppose that the assertion holds for state s_{k-1}.
   Induction step: The last action in the execution before state s_k must be $receive_m(d, n, j)$, $send_m(d, j, n)$, or $parent_m(n)$ for some node j.
   Case 1: $receive_m(d, n, j)$:
   If $myparent_j$ is non-empty, nothing happens. Thus the invariant must still hold.
   If $myparent_j$ is empty, then $myparent_j$ becomes { n }. Thus the size of $myparent_j$ is 1 (first condition); since $myparent_i$ was empty in the previous state, it's ok for it to change (second condition); and finally, consider any sequence of nodes of the form n_1, n_2, ... such that $n_{i+1} \in myparent(k, n_i)$. If j is a node in this sequence, then it must be at the beginning, because *it hasn't yet sent any search messages* and so no other node can have j in its myparent set. So the only new sequences that we get when in state s_k are those formed by adding j to the beginning of one of the sequences from state s_{k-1} that begins with n (which is in $myparent_j$).
   Case 2: $send_m(d, j, n)$: There is no change to myparent.
   Case 3: $parent_m(n)$: There is no change to myparent.

   There's a problem, though, because BFS doesn't have this property; in fact, if the root node receives a search message, it will set a parent and there will be a loop. The mistake in the algorithm is not providing some test for root. The mistake in the proof is

the assumption that if a node has an empty myparent set when it receives a search message, then it hasn't sent any messages yet. That's not true for the root node.

(Not required by the question): To prove the safety property, using the above assertion, just note that the invariant doesn't allow any cycles to form.

2. Does the liveness property "every send is eventually followed by a receive" hold for:
    a. Reliable FIFO channel with tasks { { receive(m) : m ∈ Message } }.
    b. Channel with reordering and duplication but no lost messages, with tasks { { receive(m) : m ∈ Message } }.
    c. Channel with reordering and duplication but no lost messages, with tasks { { receive(m) }: m ∈ Message }.

    In each case, explain informally why or why not.

    a. Yes. In a reliable FIFO channel, the head of the queue is enabled until it has been received, so it must eventually be received in each fair execution.
    b. No. In the unreliable channel, there is only one task, which is enabled until some message is received. If a message M has been sent (put into the channel), the task is enabled until all messages have been received; but other messages may be sent and subsequently received, without a receive(M), since messages can be duplicated or reordered. This satisfies the requirement that if a task is enabled forever, then it occurs infinitely often in each fair execution, without receive(M) ever appearing in the execution.
    c. Yes. Since there is a task for each message, the message receive is enabled once the message arrives in the channel, so in a fair execution it must eventually be received.

3. Describe a single node implementing a sender for a stop-and-wait algorithm that is robust over an unreliable channel (messages lost, reordered, and dropped). Use English, pseudo-code, or any programming language you like, but the node can only see its own state and the messages that it receives.
    a. What messages does it receive (describe any fields)?
    Messages with an ack and a parity bit to distinguish successive acks in case an ack is reordered.
    b. What messages does it send (describe any fields)?
    Messages with data and a parity field (0/1) to distinguish successive messages in case an ack is dropped.
    c. What are its states (define these in terms of state variables) and how are they initialized?
    The state contains a FIFO queue of messages to be sent and a parity variable to determine the value of the parity field in the next message. The FIFO queue can be initialized to an infinite collection of messages or (if there is another automaton providing messages and a write action) to empty. The parity variable can be initialized to 0 (or 1), it doesn't matter.
    d. How does it change state when it receives a message?
    The message will be an ack, so it will check that the ack parity field matches the current value of the parity field in the state and if it does, it removes the message at the head of the queue and reverses the parity field.
    e. When does it send a message, what message does it send, and how does the state change when a message is sent?
    Any time the queue is non-empty, it sends the message at the head of the queue with the message parity field set to the same value as the parity field in the

current state. It doesn't change state.

4. What is the number of messages required for synchronous leader election to output a leader in a unidirectional ring using LCR? The time? Give an informal explanation for each of your answers.

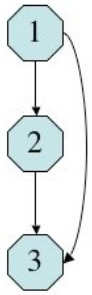
   Let n be the number of nodes in the ring.

   Number of messages: Best case 2n-1, **worst case n(n+1)/2**
   Best case (not required): If the node IDs are in increasing order going in the direction of the ring, then n-1 nodes send a message containing a UID smaller than the UID of the receiving node. These nodes do not forward the message; this is n-1 messages. All nodes forward the maximum UID; this is n messages.
   Worst case (required): If the node IDs are in decreasing order going in the direction of the ring, then n nodes send the largest ID, n-1 nodes send the next largest ID, etc. The sum $1 + 2 + ... + n = n(n+1)/2$

   Time: n
   It takes exactly n rounds for the maximum ID to get all the way around the ring (moving one node each round).

5. Construct counter-examples to two of the following statements:
   a. In an asynchronous network, the breadth first search algorithm computes a minimum-weight spanning tree

      In the following diagram, if messages get from node 1 to node 2 and then from node 2 to node 3 before any message gets from node 1 to node 3, the tree edges will be <1,2> and <2,3>, but the edges for a breadth-first tree would be <1,2> and <1,3>. Suppose that the weight on <1,2> is 100 and the weights on the other edges is 1, then the tree is not minimum weight.



   b. The LCR algorithm for leader election needs at least n^2 messages to find a leader.

      See the best case described above.

   c. Using the learning bridge algorithm guarantees that messages follow the shortest path from source to destination.

The learning bridge algorithm uses a mac-address-table that relates the source of a message to the port that it arrived on. When a message arrives, it looks up the destination address and sends the message out the port in the table (if the table has such an entry) or out all ports (otherwise).

Consider bridges connected as in the diagram in part a, assuming that all paths are bidirectional. Then if the first message is from node 1 and it travels more rapidly through node 2 to node 3 than it does going directly to 3, and if 3 receives a message to send to 1 before it receives the message along the direct link from 1 to 3, then 3 will send the message back to node 2 to get it to 1. This is not the shortest path.

d. If β is a trace of B then there is an trace γ of $A \times B$ such that $\beta = \gamma|B$.

Let A an input b and an output a. Let B have an output b and an input a. traces(B) = {a,b}*. traces(A) = { α | $\alpha \in$ {a, b}* and no subsequence of α is bb }. Then traces(A×B) = traces(A), because the external actions of A are the same as the external actions of A×B and thus the restriction of any trace $\gamma \in$ traces(A×B) to external actions of A is just γ.
But traces(B) is all traces over {a,b}*, so pick any trace β of B containing two successive b actions. By the same argument as we used above, the restriction of any trace $\gamma \in$ traces(A×B) to external actions of B is just γ. Since there can be no $\gamma \in$ traces(A×B) with two b actions in succession, there is no γ such that $\beta = \gamma|B$.

e. If a finite execution α is a fair execution of A and β is any execution of B, then $\alpha \cdot \beta$ is a fair execution of $A \times B$.

Let β be any execution of B that is not fair. If β is not a fair execution, $\alpha \cdot \beta$ cannot be a fair execution because the same actions will be enabled at the end of β.