Individual Systems Practical

November 6, 2016

1 Swapping events

Currently, the design reflects the following concepts. A process is an ordered collection of events, each event happening at a certain time value. If two events happen at the same process, then they have a different time value. There are three event types: internal, send and receive event types. The events within a process are grouped together: a receive event is a leader and all other non-receive events are joined to the group of the preceding event. Initiator events are send or internal events that happen at the start of a process, and the first event that happens is the leader of the group. Only event groups can be dragged around and rearranged, not individual events, in the timeline window.

Now consider the gesture to reorder events. This is essential, since that allows students to explore the consequences of a different order of receive events. Initially, it seemed a good idea to let students drag these event groups, and whenever one group is dragged over the other group, the events in the group swap. This solution is problematic: since the existence of an event depends on all previous events at some process.

Suppose we have for some process the receive events p, q, r interleaved by either internal or send events a, b, c, d:

$$p, a, b, q, c, r, d$$

The grouping is as follows: $(p, a, b), (q, c), (r, d)$, i.e. each receive event is a leader and the process is not an initiator (since it starts with a receive event). Now consider that the student wishes to reorder the groups (q, c) and (r, d), to explore what happens when r is received before q. There are several subtleties:

1. Suppose that c is a send event. The receive event r may depend on the event c, e.g. the event c sends to some other process that directly sends back a message which is received by r, and therefore the two groups can not be exchanged. We thus have two different categories of *happens before*: the happens before as defined by program order, and the necessary happens before as an indirect consequence of previous send events. Clearly, (r, d) can not happen before (q, c) since c happens before r. Ideally, the conflicting path should be indicated to the student that prevents the reordering.

2. Suppose that we can reorder events that does not result in the conflict as mentioned above. Now suppose that after the events (p, a, b) the process has some state σ. Then, depending on the message received, the process has a new state σ'. In the case q was received, the new state results in the internal or send event c and thereafter waits for the receival of r. However, if (r, d) was reordered before (q, c), then (r, d) might have a completely different state σ'' than we assumed it had after c. Even more subtle, (q, c) may completely disappear, where the process drops any further messages received. Therefore, we can not assume that the reordering of groups maintains the identity of the groups at all.

   (a) If some group (r, d) is dragged backward over (q, c), the group that is dragged may change itself, and the consequent events may also change. Thus, it may make sense that only receive events may be dragged, since the other events within the dragged group could change or disappear.

   (b) If some group (q, c) is dragged forward over (r, d), then the dragged group itself may completely disappear, since now r determines the future events of the process. In this case, it also makes sense that only receive events may be dragged, but whenever the dragged event disappears the gesture should end.

The path forward is to ensure that a reorder gesture clearly indicates to the student which of these scenarios is the case. In the first scenario, a red highlight of the indirect conflicting consequence should indicate that the reorder is not possible. In the second scenario, the student will have to end the gesture in the case that the event that was dragged disappears. However, every gesture should be reversible by dragging back to the original position, thereby recreating the original execution.