Exercise 3.32.

The procedures to be run during each time segment of the agenda are kept in a queue. Thus, the procedures for each segment are called in the order in which they were added to the agenda (first in, first out). Explain why this order must be used. In particular, trace the behavior of an and-gate whose inputs change from 0,1 to 1,0 in the same segment and say how the behavior would differ if we stored a segment's procedures in an ordinary list, adding and removing procedures only at the front (last in, first out).

Answer.

Keeping the procedures of the same time segment in a queue guarantees us each function box derives its output from the most advanced inputs offered by the user. If we stored a segment's procedures in an ordinary list, the output a function box produces would be computed from the intermediate result and it may possibly be wrong. We shall see the underlying reason in tracing the behavior of an and-gate whose inputs change from 0,1 to 1,0 in the same segment.

Figure 1 shows states of wires connected to an and-gate after initializing with 0,1.

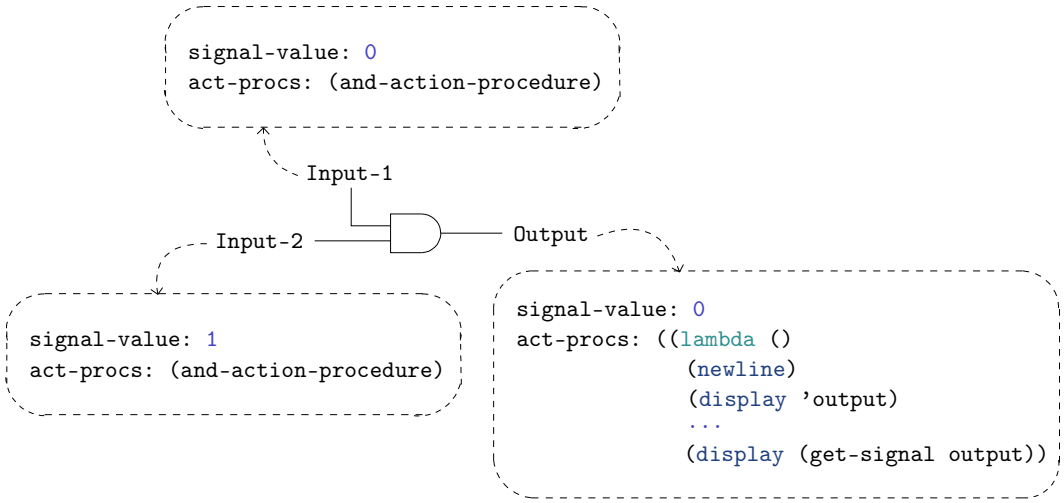

Figure 1. States of wires connected to an and-gate after initializing with inputs 0,1.

```
(define the-agenda (make-agenda))
(define and-gate-delay 3)

(define input-1 (make-wire))
(define input-2 (make-wire))
(define output (make-wire))

(probe 'output output)

output 0  New-value = 0
;Unspecified return value

(and-gate input-1 input-2 output)
;Value: ok

(propagate)
;Value: done

(set-signal! input-1 0)
;Value: done
```

```
(set-signal! input-2 1)
;Value: done

(propagate)
;Value: done

(set-current-time! the-agenda 0)
;Unspecified return value
```

There may be two possible kinds of intermediate inputs the and-gate would go through, say, $0,1 \to 1$, $1 \to 1,0$ or $0,1 \to 0,0 \to 1,0$. This depends on which one of the inputs the user's choose to alter first.

If the inputs to the and-gate were modified through the states of $0,1 \to 1,1 \to 1,0$, that is, the user first changed the signal value on `input-1` to be 1, then revised the signal value on `input-2` to be 0. The intermediate inputs 1,1 would stimulate a procedure to set the signal on `output` to 1 after three time units from the begining and add this procedure into the agenda. Later the switch on inputs from 1,1 to 1,0 arouse another procedure to set the signal on `output` to 0 in the same segment. As procedures to be run during each time segment of the agenda were kept in an ordinary list, the procedure which computed the output signal from the most advanced inputs would be put ahead of the one which derived the output signal from the intermediate inputs in the same segment. Figure 2 presents us the

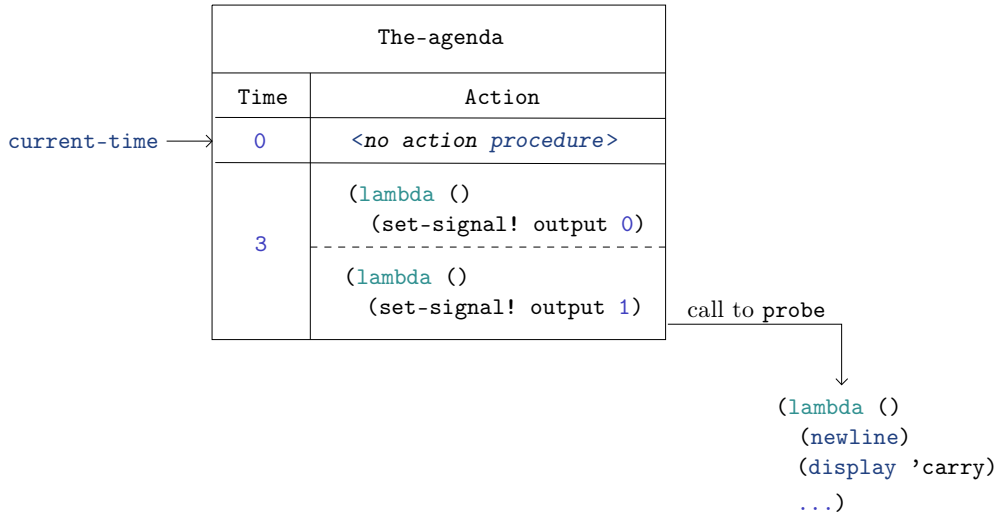

Figure 2. Contents of `the-agenda` after changing the inputs in the sequence of $0,1 \to 1,1 \to 1,0$.

agenda after changing the inputs in the sequence of $0,1 \to 1,1 \to 1,0$. Hence, when the simulator run the actions at the time segment 3, it first set the signal on `output` to 0, then set it to 1. It was this mutation of changing the signal of `output` from 0 to 1 triggered the `probe` to print the new signal value of `output`:

```
(set-signal! input-1 1)
;Value: done

(set-signal! input-2 0)
;Value: done

(propagate)

output 3  New-value = 1
;Value: done
```

However, if the user choose to modified the signal value on `input-2` to be 0 first, in which the states of inputs the and-gate experienced were $0,1 \to 0,0 \to 1,0$. Since none of these inputs would make the and-gate derive its `output` aside from 0, the agenda would remain empty through out the simulation and nothing more than `'done` would be printed when we allow the values to propogate:

```
(set-signal! input-2 0)
;Value: done

(set-signal! input-1 1)
;Value: done

(propagate)
;Value: done
```