1 Different Behaviors for the Holes

We build an OA serving as a prototype for connection-oriented communication protocols. Let N be the number of data units to be sent, whose index ranges from 0 to $N-1$. There are two holes, namely Sender and Receiver. The Sender invokes actions `send`, `receive` and `process`, corresponding to sending a data unit, receiving an acknowledgement and doing an internal work, such as waiting until timeout. The Receiver also invokes actions `send`, `receive` and `process`, but here it sends an acknowledgement and receives a data unit. A data unit or an acknowledgement may also be lost during transmission. From the environment's perspective, a data unit can be in one of the following situation

- it needs to be sent,
- it is on fly i.e. it was sent by the Sender but not arrives at the Receiver yet,
- it has been received by the Receiver, or
- it has been sent and acknowledged successfully.

Hence we use four variables of type lists D_{tosend}, D_{onfly}, D_{received} and D_{done} to store the indices of the data units in these situations. For an acknowledgement, it can also by on fly, which is store in A_{onfly}. We do not need to store received acknowledgements because the Sender processes an acknowledgement immediately when receiving it. The initial values are $D_{\text{tosend}} = [0, \ldots, N-1]$ and $D_{\text{onfly}} = D_{\text{received}} = D_{\text{done}} = A_{\text{onfly}} = []$. We use two operation on a list L

- L.push(i): appending i to the list, and
- L.pop(i): removing i in its first appearance from the list or return the old list if there is no i

and a function $\min(L)$ to find the minimal element in the list. The transitions are given in the following table.

N.O.	Source	Target	Action	Hole actions	Guard	Reassignment
1	q_0	q_0	sendData(i)	Sender $\mapsto$ `send`(i)	$i = \min(D_{\text{tosend}})$	D_{onfly}.push(i)
2	q_0	q_0	receiveAck(i)	Sender $\mapsto$ `receive`(i)	$i \in A_{\text{onfly}}$	A_{onfly}.pop(i) D_{tosend}.pop(i)
3	q_0	q_0	τ	Sender $\mapsto$ `process`($opts$)	True	
4	q_0	q_0	receiveData(i)	Receiver $\mapsto$ `receive`(i)	$i \in D_{\text{onfly}}$	D_{onfly}.pop(i) D_{received}.push(i)
5	q_0	q_0	sendAck(i)	Receiver $\mapsto$ `send`(i)	$i \in D_{\text{received}}$	D_{received}.pop(i)
6	q_0	q_0	τ	Sender $\mapsto$ `process`($opts$)	True	
7	q_0	q_0	loseData(i)		$i \in D_{\text{onfly}}$	D_{onfly}.pop(i)
8	q_0	q_0	loseAck(i)		$i \in A_{\text{onfly}}$	A_{onfly}.pop(i)

Table 1: Transition Table for Protocol OA

An protocol implementing the Sender and Receiver is correct if given the initial assignments of the lists, the composed OA always arrives at a terminal state with $D_{\text{done}} = [1, \ldots, N]$.

Consider and example. Let $D_{\text{tosend}} = [0, 1, 2]$ and $D_{\text{onfly}} = D_{\text{received}} = D_{\text{done}} = A_{\text{onfly}} = []$. Denote by $(D_{\text{tosend}}, D_{\text{onfly}}, D_{\text{received}}, A_{\text{onfly}})$ each configuration of the OA. One possible run is

$$([0, 1, 2], [], [], [], []) \xrightarrow{\text{sendData}(0)} ([1, 2], [0], [], [], [])$$