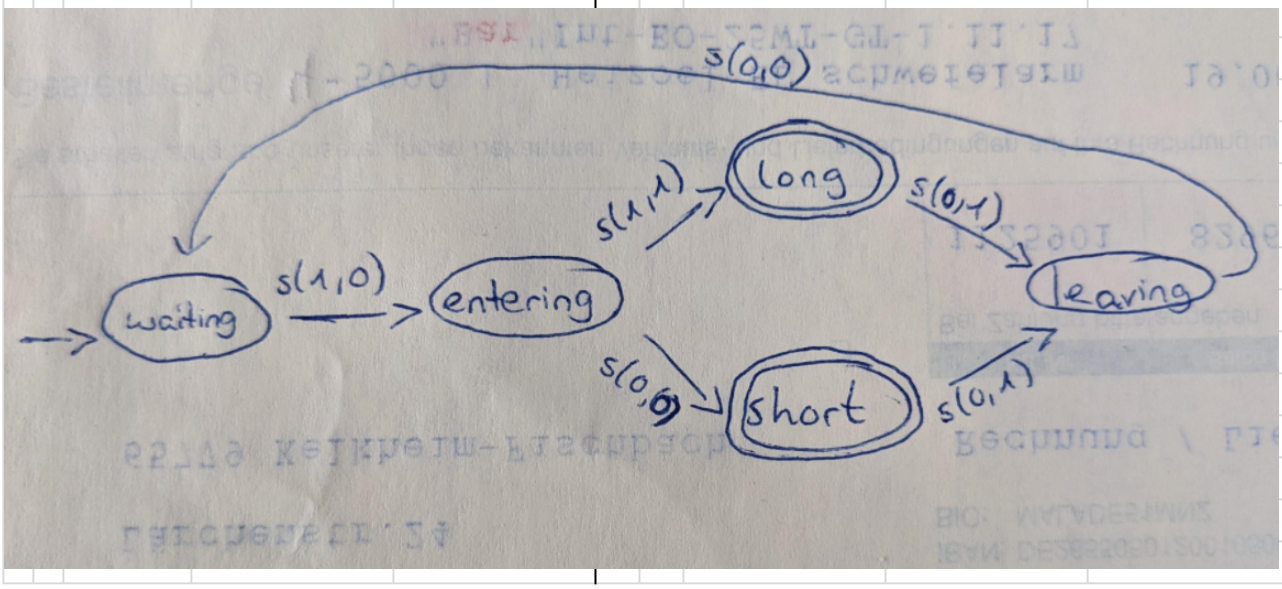
Differences highlighted yellow

L1					L2				
s0	**s1**	**causative action**	**packet position**	**FSM state**	**s0**	**s1**	**causative action**	**packet position**	**FSM state**
0	0		ahead of both sensors	**waiting** for approaching packet	0	0		ahead of both sensors	**waiting** for approaching packet
1	0	enter s1	under s1 only	packet **entering**	1	0	enter s1	under s1 only	packet **entering**
1	1	enter s2	under capture of both sensors	from here on, you already know it's a **long** packet	0	0	leave s1	between both sensors	from here on, you already know it's a **short** packet
0	1	leave s1	under s2 only	packet **leaving**	0	1	enter s2	under s2 only	packet **leaving**
0	0	leave s2	behind both sensors	packet handled	0	0	leave s2	behind both sensors	packet handled



Task: Differentiation if a packet is $>$ or $<$ compared to a given L_s (length of a short packet).
(the case of being equal is not requested in the task, hence not covered here)

Sensor states:
0 = No object / empty
1 = Object / blocked

Data-packet whenever any sensor flips its state. Consists of tuple (s_1, s_2)

“Life cycle”:
When entering a **final state**, the machine identified the packet type and can send this information to other processes in need of this information for further handling.
This should be done concurrently to avoid that delays in the transmission of the result may cause the machine to miss out on sensor information about the next package, leaving the FSM in a wrong state.

Looking at the picture:
$L2 < L_s < L1$
Since $d > L_s$, you can assure that between two packets, the state $(0, 0)$ will always occur
$\rightarrow$ packet separation

Walkthrough of packets L1 and L2, derivation of FSM idea and description in textual form:
See attachment

FSM:
See attachment

Further thoughts:
Transition occurs whenever a new data-packet is received (which the task specifies to only occur on sensor change).
Even though not shown in the graph, the FSM may (for formal correctness) include a “catch state” to catch all not yet depicted transitions. This state might throw an application exception since the sensor data doesn’t fit up (unexpected sensor state transitions).

As initial state, “waiting” is expected, although it’s possible to have any state except for $s(0, 1)$ as starting state, as the length of the current packet can still be determined. This can be seen in the graph.

Therefore, in practice, the machine should only require the packet to be moved back out of the machine when it
(1) was started in $s(0, 1)$
and
(2) does not remember the previous state (as is currently not implemented)