Labelled Transition Systems for Agents' Behaviour

Multi-Agent Systems Project

Roberta Lamberti [SM3800007], Tanja Derin [SM3800013]

Data Science and Artificial Intelligence

Supervised by Prof. Padoan Tommaso

September 2025

1 Introduction and Setting

Three prisoners are kept in separate cells. From time to time, a fair (but unknown) scheduler selects one prisoner to visit a room containing a single light switch. Initially the light is off and is not visible from outside. At any time, if some prisoner correctly declares *all three have visited the room at least once*, they are released. Beforehand they agree on a protocol. We model such a protocol as LTS for the agents and environment, and we prove that, under fair scheduling, the protocol guarantees eventual correct termination. It's possible to find an implementation of the project at **this repository** [1]. The whole system is the synchronous parallel composition

$$\mathsf{Sys} \;=\; \mathsf{Room} \parallel \mathsf{Scheduler} \parallel \mathsf{Counter}(2) \parallel \mathsf{Leader} \parallel \mathsf{Sig}_2 \parallel \mathsf{Sig}_3.$$

The components are:

- Room: the room/light; synchronises on enter, exit, and on on/off (only one of on/off is enabled according to the current light state).
- Scheduler: a fair scheduler; synchronises on cell:1, cell:2, cell:3.
- Counter(2): a counter with $c_0 \xrightarrow{\mathsf{add}} c_1 \xrightarrow{\mathsf{add}} c_2$; self-loops cont in c_0, c_1 and end in c_2.
- Leader: the leader/counter prisoner (LTS in Fig. 1).
- $\mathsf{Sig}_2, \mathsf{Sig}_3$: the two identical signaler prisoners.

The initial state is from r_0 (light OFF) for Room, t_0 for Scheduler, c_0 for Counter(2), ℓ_0 for Leader, and s_0 for each signaler.

Protocol. In order to solve the puzzle, we suggest the following solution: one of the prisoner is the leader, denoted by l, who will act as the counter, and the left prisoners are the signalers. In this protocol, the leader and the signalers behave as follows:

- **Leader** l: whenever he enters the room, and the light is on, he switches it off, add to the counter the time he has switched off the light and exit the room. If the light is off when they enter the room, do nothing and exit.
- **Signalers** s**:** whenever they enter the room and the light is off, they switch it on. Indeed, when they enter the room and the light is on, they do nothing and exit.

The LTS for the Leader prisoner, the one who is the counter, has the following states and actions:

$$S = \{\ell_0,\ \ell_0^{in,0},\ \ell_0^{in,1},\ \ell_1^{in,0},\ \ell_1,\ \ell_1^{in,1},\ \ell_2^{in,0},\ \ell_{done}\}$$
$$A = \{\text{cell:1},\ enter,\ exit,\ cont,\ off,\ add,\ end,\ done\}$$

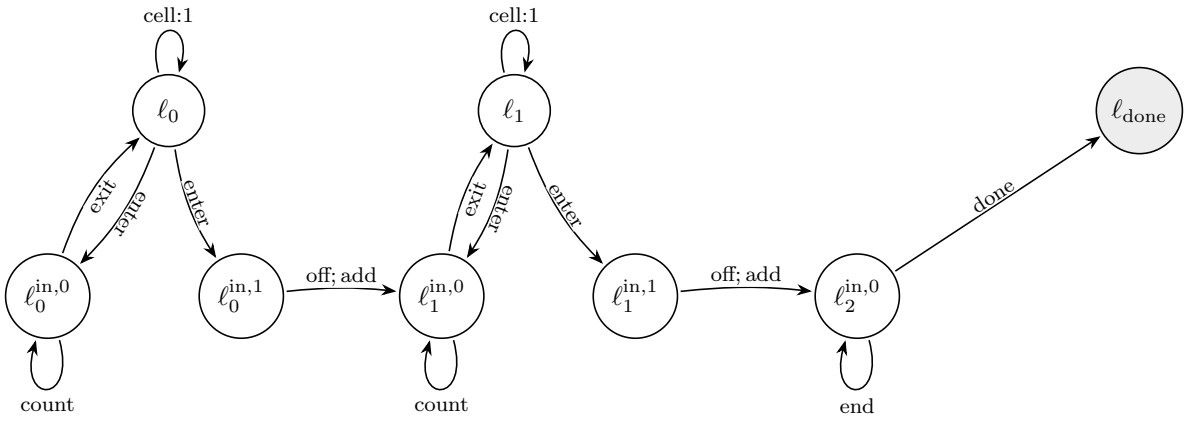

Figure 1: Leader (counter) LTS with all *in*-states aligned.

The states l_0 and l_1 are a representation of the leader prisoner being in the cell. Note that $\ell_k^{c,j}$, where c represent the room inside which the prisoner has just entered, j is for the light (0 when it's switched off, 1 when it's switched on), and k represent the count.

We start from the state l_0, that will synchronize with the scheduler $cell : 1$, to start his turn and perform the action *enter* to reach the state $\ell_0^{in,0}$, which denotes the state where the leader prisoner is at state count 0, and the light is off.
So once he inside the room, he can *count* or *exit*. As the light is already off, he will exit the room. When instead he enters the room again, going from state l_0 to the state $\ell_0^{in,1}$, the leader prisoner finds the light switched on. So he switches it off through the action off and then *add* it to the counter. Now being in the state $\ell_1^{in,0}$, the light is off, so he exits the room and reach state l_1. He repeats the same actions as before, to then reach the state $\ell_2^{in,0}$, to end the protocol and reach the state l_{done} through the action *done*.

Instead, the LTS for the signalers prisoners is represented by:

$$S = \{s_0,\ s_0^{in,1},\ s_0^{in,0},\ s_1^{in,0},\ s_1,\ s_1^{in,1}\}$$
$$A = \{\text{cell:2},\ enter,\ exit,\ on,\ add\}$$

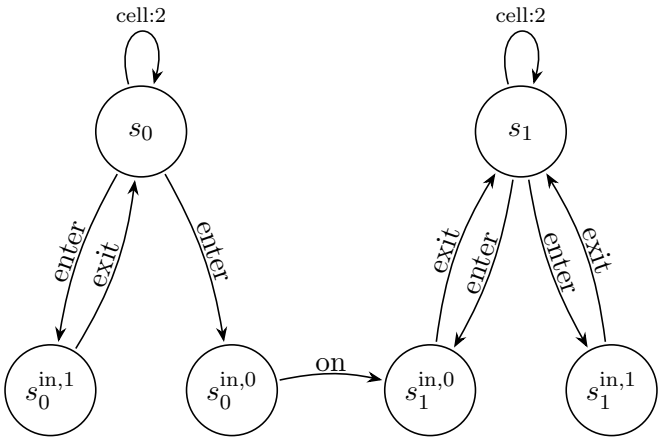

Figure 2: Signaler LTS (for prisoner 2 or 3).

From s_0 (where nobody has signalled yet), the first time the signaler enters and finds the light *OFF*, he performs *on; exit* and moves to s_1. If he finds it *ON*, he simply *exit*s and remains in s_0. Once in s_1 (where somebody has already signalled), he always *exit*s without touching the switch, regardless of the light. Replace `cell:2` with `cell:3` for the other signaler.

2 Temporal Property

We express "eventually the prisoners give the correct solution" by the recursive HML formula

$$X \;=_{\min} \langle \mathsf{end} \rangle \langle \mathsf{done} \rangle \top \;\vee\; \big(\Diamond \top \wedge \Box X \big).$$

The left disjunct says that the sequence end;done is already executable. The right disjunct requires that at least one transition is currently possible ($\Diamond\top$) and that, no matter which enabled action occurs, the successor state still belongs to X ($\Box X$). Taking the *min* fixpoint ensures reachability in finitely many steps.
Let

$$\mathsf{Sys} \;=\; \mathsf{Room} \parallel \mathsf{Scheduler} \parallel \mathsf{Counter}(2) \parallel \mathsf{Leader} \parallel \mathsf{Sig}_2 \parallel \mathsf{Sig}_3$$

and let the initial global state be $s_{init} = (r_0, t_0, c_0, \ell_0, s_0, s_0)$. Then $s_{init} \in X$.

This holds because for any fair execution of synchronous composition, the scheduler is fair, so each prisoner is chosen infinitely often. By the signalers' LTS, a signaler in s_0 performs *on; exit* the first time he enters when the light is off, then moves to s_1 and never touches the switch again so in any fair run each signaler produces exactly one on. Whenever the leader enters and finds the light on, his LTS forces the synchronisation *off; add*, restoring the light to off and increasing the counter. Therefore the counter evolves monotonically $c_0 \to c_1 \to c_2$ in finitely many steps; in c_2 action *end* is enabled while the leader is at a state where *done* is offered. Thus a state satisfying $\langle \text{end} \rangle \langle \text{done} \rangle \top$ is reached after a finite prefix. Moreover, before that point the system is never deadlocked and whichever enabled action occurs the successor state still satisfies X, because the protocol LTSs forbid deviations (signalers cannot create extra *on* events and the leader cannot skip *off; add*).

3 Unknown Initial Light

We now assume the initial light state is unknown so the room may start in r_0 or r_1. To remove the ambiguity of a possible initial on that was not produced by a signaler, we modify only the counter bound and the prisoners' protocols as follows, leaving room and scheduler unchanged.

We take Counter(4) (states $c_0 \to c_1 \to c_2 \to c_3 \to c_4$ with the usual self-loops *cont* in $c_0, \ldots, c_3$ and *end* in c_4). Each signaler now signals twice and after never touches the switch again. The leader's policy is unchanged, whenever he enters and finds the light on, he turns it off and increases the counter by one; if he finds it off, he exits. When the counter is at 4 he performs *end* and then *done*.

To be sure this works we can consider two cases. (i) If the light initially is *off*, the only on events are produced by the signalers. Since each can produce at most two, exactly four on events will eventually occur in a fair run. The leader consumes each (off; add), so the counter reaches 4 and he announces done. (ii) If the light initially is *on*, there is one extra on event before any signaler acts. The leader will consume that one as well, and then continue to consume on events produced by the signalers. Because each signaler can contribute at most two on events, reaching a total of 4 necessarily uses contributions from *both* signalers (initial on $+2+1$), which implies both have entered at least once. In either case, once the counter hits 4 the leader executes *end; done*.

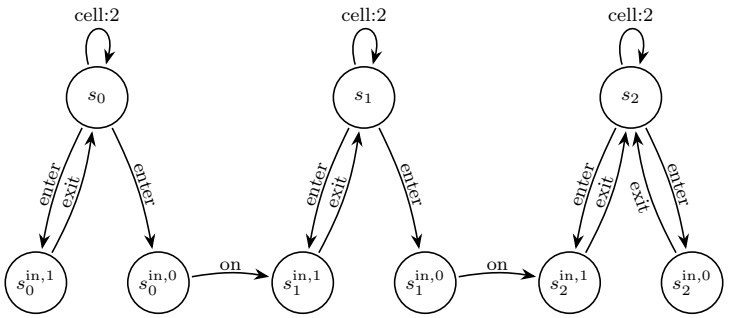

Figure 3: Signaler LTS for the unknown initial light case (scaled down).

References

[1] Hans Ditmarsch, Jan van Eijck, and William Wu. “One Hundred Prisoners and a Lightbulb - Logic and Computation.” In: Jan. 2010.