# The Prisoner problem
# - Modelling and Verification of Reactive Systems -

Alessandro Balestrucci

## Trace

*A number of prisoners are segregated in solitary cells. There is a central living room with one light bulb. No prisoner can see the light bulb from his or her own cell. Everyday, the warden picks a prisoner equally at random, and that prisoner goes to the central living room. While there, the prisoner can toggle the bulb if he or she wishes. Also, the prisoner has the option of asserting that all prisoners have been to the living room. If this assertion is false (that is, some prisoners still haven't been to the living room), all prisoners will be shot. However, if it is true, all prisoners are set free. Thus, the assertion should only be made when there is 100% certainty of its validity. The prisoners are allowed to get together one night in the courtyard, there they agree on a protocol that will guarantee their freedom.*

It's supposed that the game start after the prisoner meeting, in this way than can establish a "freedom" protocol to apply. It's supposed that all prisoner agree and follow the established protocol and that a prisoner can be picked more then once also if another prisoner has not enter in the living room a first time.

## Tasks

## 1 The "Freedom" protocol

### 1.1 Informal protocol description

The protocol indicates that only one of them(*CounterPrisoner*) can turn off the bulb light every time he enters in living room, counting it, all the others(*CommonPrisoner*) can only turn on the light. Not knowing the initial state of the light in the room, all the *CommonPrisoner*s have to turn on the light twice; this is due to the fact that if the textitCounterPrisoner enters in the living his first time and the light is on, he cannot know:

- IF other *CommonPrisoner*s have been inside before him and how many;
- OR IF he is the first prisoner picked by the warden and so, initially the light is on.

If *CommonPrisoner*s turn on the light twice, the counter can claim that "all prisoner are entered" when he turn off the light $2(\#NormalPrisoners) - 2$ times. (e.g. if there are 100 prisoners in the prison, *CounterPrisoner* can claim a free all after 98 turn off actions)

The behaviour of a *CommonPrisoner* is more complex. As I said before, this kind of prisoner can only turn on the light (if when he enters the light is off, otherwise exits leaving the light on) and he must do this action exactly twice. It's forbidden to refuse to perform the action if he can do it. When a *CommonPrisoner* has performed the turn-on action twice and he is picked again in the living, either the light is on or off, he have only to exit from the living without changing the state of the light.

## 1.2 Formal (CCS) protocol description

Since slides seen during the lectures (*5.CCS*) follows the formal description of the solution:

1. The **Ligth** process:

   $$\begin{gathered}\textbf{Light} = \tau.\textbf{LightON} + \tau.\textbf{LightOFF}\\ \textbf{LightON} = \textit{turnOff}.\textbf{LightOFF}\\ \textbf{LightOFF}= \textit{turnON}.\textbf{LightON}\end{gathered}$$

   Not knowing the inital state of the light inside the room is logic that the initial state is consequence of an *invisible* action *tau*, and after the process **Light** can evolve in the process **LightON** or **LightOFF**. A **LightON** process can evolve in **LigtOFF** performing an action *turnOff*; in the opposite way, the **LightOFF** process can evolve in **LigtON** performing an action *turnON*.

2. The **Room** process:

   $$\textbf{Room} = \textit{roomIN}.\textit{roomOUT}.\textbf{Room}$$

   This process represent only a behaviour regarding the living room where is located the light bulb of aforementioned process; it is possible to enter in the living room performing a *roomIN* action and immediately afterwords to exit performing a *roomOUT* action and so evolving in the same process.

3. The processes of **prisoners**. There are two main processes that can be individuated:

   (a) The **CommonPrisoner**:
   $\textbf{CommonPrisoner} = \overline{roomIN}.(\overline{turnON}.\overline{roomOUT}.\textbf{RepeatingPrisoner} + \overline{turnOFF}.\overline{turnON}.\overline{roomOUT}.\textbf{CommonPrisoner})$

   $\textbf{RepeatingPrisoner} = \overline{roomIN}.(\overline{turnON}.\overline{roomOUT}.\textbf{IdlePrisoner} + \overline{turnOFF}.\overline{turnON}.\overline{roomOUT}.\textbf{RepeatingPrisoner})$

   $\textbf{IdlePrisoner} = \overline{roomIN}.\overline{roomOUT}.\textbf{IdlePrisoner}$

   The **CommonPrisoner** process have to turn on the light twice, after that he has to do nothing the further times is picked in the room. To do this we need to define 2 other processes: **RepeatingPrisoner** and **IdlePrisoner**. The former is needed to indicate that when the prisoner performs the *turnON* action his first time, and the process **CommonPrisoner** evolves in a new process (**RepeatingPrisoner**) where only another *turnON* action is permitted; while the latter is needed to indicate that when the prisoner performs his last *turnON* action, every next time that he will be brought in the room he will not have to do any action with the light bulb but only to exit. We said before that if the **CommonPrisoner** has still to perform a light action, but he can not perform the *turnON* action (due to the fact the when he inside the room the light is already on) he have to exit and so he have to perform directly an *roomOUT* action. This is not possible because due to non-determinism both the **CommonPrisoner** and the **CounterPrisoner**, in dual way, because the operator plus (+) of CCS can perform the *roomOUT* action also when is possible to perform an action with the light bulb.
   For this reason, we concatenate two action *turnOFF.turnON.* to don't change the situation respectiong the condition that when is possible to perform an action with the light bulb it's performed.

   (b) The **CounterPrisoner**:
   $\textbf{CounterPrisoner}_{2n-2} = \overline{roomIN}.(\overline{turnOFF}.\overline{roomOUT}.\ \textbf{CounterPrisoner}_{(2n-2)-1} + \overline{turnON}.\overline{turnOFF}.\overline{roomOUT}.\textbf{CounterPrisoner}_{2n-2})$

   $\vdots$

   $\textbf{CounterPrisoner}_{(2n-2)-k} = \overline{roomIN}.(\overline{turnOFF}.\overline{roomOUT}.\ \textbf{CounterPrisoner}_{(2n-2)-(k+1)} + \overline{turnON}.\overline{turnOFF}.\overline{roomOUT}.\textbf{CounterPrisoner}_{(2n-2)-k})$

   $\vdots$

   $\textbf{CounterPrisoner}_0 = \mathit{freeAll}.\textbf{nil}$

4. The **“Freedom” protocol** system:

## 2 Correctness using behavioural equivalence

## 3 Correctness using Temporal Logic (HML)

## 4 CAAL project

I thought to use CAAL for two reason:

1. Syntax almost equal to CCS;
2. Web-based.

### 4.1 Project availability