2nd Seminar – State-Based Modelling – Solutions

1 Traffic light

We are designing a controller mechanism for a traffic light.

a. Create a state space for a simple red–yellow–green traffic light that is detailed enough for controlling the traffic light. Make sure that the state space is mutually exclusive and complete.
Solution
{ *red, yellow, green, red-yellow, blinking yellow, off* }, in brief: $S = \{r, y, g, ry, \hat{y}, \circ\}$. This is mutually exclusive and complete. The initial state: r.

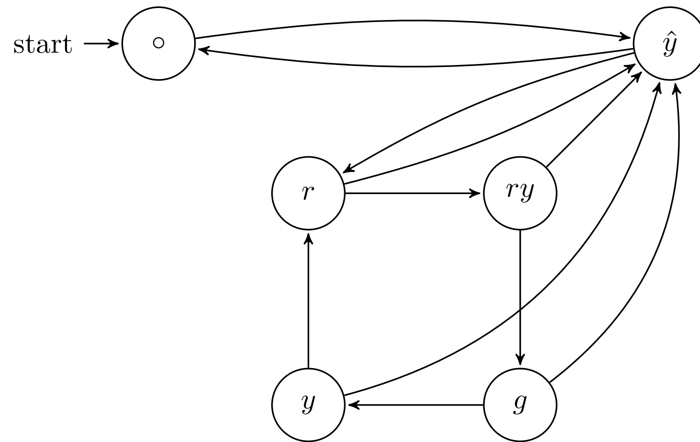
b. What are the state spaces of the individual bulbs? What is the abstraction relation between the state space of the traffic light and the state spaces of the individual bulbs? How does the state space of the traffic light relate to the Cartesian product of the state spaces of the individual bulbs?
Solution
$S_r = \{r, \circ\}$, $S_y = \{y, \hat{y}, \circ\}$, $S_g = \{g, \circ\}$.
What is the abstraction relation between the state space of the traffic light and the state spaces of the individual bulbs? The relation is *projection*. (Graphically: e.g. if we put the traffic light into such a box that we can only see the red bulb, then we see the S_r state space.) A state space of a component is always a valid state space for the whole system (since the states of the other components can be „omitted", i.e., gathered into a single state).
Cartesian product: $S_r \times S_y \times S_g$ yields $2 \times 3 \times 2 = 12$ states. The elements of the Cartesian product are 3-dimensional vectors (in other name: 3-tuples), e.g. $\langle r, y, \circ \rangle$, which we will abbreviate now as ry (omitting the state $\circ$).
Of course not every state actually occurs from the Cartesian product, so $S \subset S_r \times S_y \times S_g$.
The $S_r \times S_y \times S_g$ Cartesian product in tabular form:

		S_g	
S_r	S_y	g	$\circ$
r	y	ryg	ry^*
	$\hat{y}$	$r\hat{y}g$	$r\hat{y}$
	$\circ$	rg	r^*
$\circ$	y	yg	y^*
	$\hat{y}$	$\hat{y}g$	$\hat{y}^*$
	$\circ$	g^*	$\circ^*$

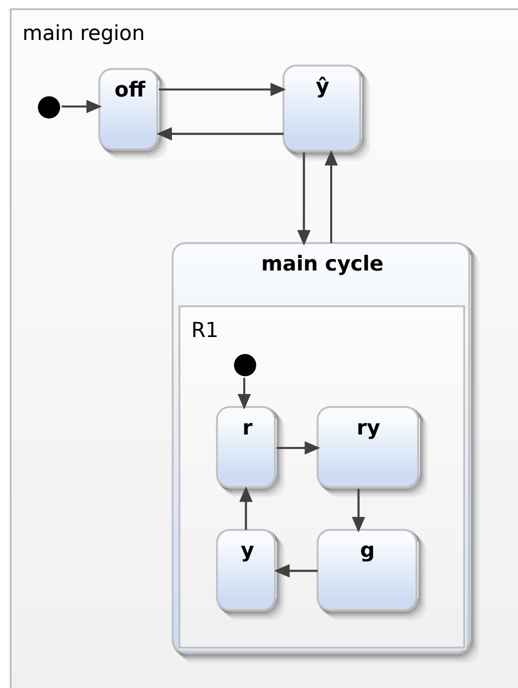
States marked with * are present in the original state space.

c. What are the valid state transition rules? Create the (simple) state graph!
Solution
The goal is to determine the valid transition rules between the six states. We face many design decisions for which we gave some answers between parentheses:
- What is the initial state after powering on the traffic light? (blinking yellow)
- Are we modelling only the correct operation (omitting faulty behaviour)? (yes)
- Is it a faulty behaviour if the power goes out? (yes, let's assume that there are only planned shutdowns)
- Can it change from red to red? (we omit this since it's not observable)
- Can it change directly from red-yellow to red (e.g. in case of a traffic accident)? (no)

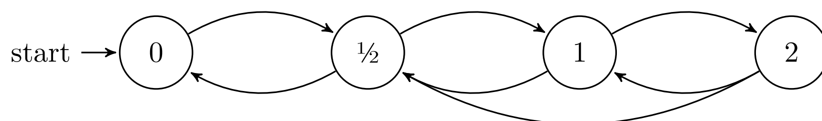
There is a state machine-like figure in the Hungarian Highway Code book, however, it only contains the red, red-yellow, green and yellow states; the blinking-yellow and off states are omitted.
The state machine can transition from anywhere to blinking-yellow state ($\hat{y}$). A possible solution:



d. How can we represent the same behaviour in a more compact way with hierarchical states?
Solution
The traffic light can be switched to blinking yellow from both r, ry, y, g – so we can group the state set $\{r, ry, y, g\}$ into the hierarchical state *main cycle* (which will be the other state of *operational* beside $\hat{y}$) in order to use only 1 transition rule (instead of 4) to express that the traffic light can go to blinking yellow from any state in the main cycle.



e. When measuring the electric energy consumption of the traffic light we are only interested in the number of bulbs emitting light at a given time. Use abstraction on the current state machine to create a new one where the states differ only in their energy consumption!
Solution
Let's construct a state space with 4 states based on the energy consumption of the bulbs: 0, ½, 1, 2 (all three cannot emit light at the same time). We could add self-loop transitions to state 1 (e.g. in case of the green → yellow → red sequence only one bulb emits light) but we won't do this since we can't observe these transitions at *this level of abstraction*. Parallel transitions with the same input/output are omitted.



f. At the end of the red signal there is a period when the green bulb of the perpendicular pedestrian crossing is blinking. Refine the state graph (from before the abstraction) in a way so that this state is presented separately!

Solution
Let's split the red state into two new states: for cars it's red and for pedestrians it's green (r_g); for cars it's red and for pedestrians it's blinking green (r_{bg}). Let's modify the state space graph accordingly: there is a $r_g \rightarrow r_{bg}$ transition between the two new states. The rest of the graph is unchanged.
We could consider whether we need a state where it's red for both cars and pedestrians. If we previously decided that the traffic light must show red after powering it on then we should also add this to the new graph. In this case we split the red state into three new states: r_r, r_g, r_{bg}.

g. There are 10 traffic lights along a road with 4 states each. How many states can the whole system have at the most? Do we have to handle every state?
Solution
$4^{10} = 2^{20} = (2^{10})^2 \approx 10^6$. (Every combination of the 4 states of the 10 traffic lights.)
Of course it's possible that not all of these states are reachable since the state transitions of different traffic lights are probably not independent due to the different synchronization constraints. For example the system of 10 traffic lights has a "rhythm" due to the green wave setting.
Instead of examining the whole system it can often be beneficial to merge multiple road junctions into one segment and examine whether there can be cars on a segment or not (e.g. during traffic control). In this case we need to know the structure of the system in order to determine segment boundaries, i.e. we need the structural model of the system.

2 Three-tier architecture

We would like to model an IT system which is built as a *three-tiered architecture* in the following manner:

Are the following sets valid state spaces for modelling the behaviour of our system?

Solution
Solution notes: We have to check for every *set* below whether its states are *complete* and *mutually exclusive* (thus a state space). Used concepts:

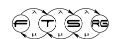
- **Database server**: stores data for a long time
- **Application server**: runs the business logic
- **Web server**: responsible for generating and presenting HTML pages.

a. { Web server in use, Application server in use, Database server in use }
Solution
No. Multiple server can be in use at the same time.

b. { Powered off, Idle, In use }
Solution
Yes. We can have additional states of course (like *hibernated* by refining *Powered off*).

c. $\mathbb{N}$ (as the number of requests currently under processing)
Solution
Yes. Although it's important to note that this results in an infinite state space – in some cases we need a model like this about the system (see the Performance modelling topic). Beside being mutually exclusive we have to check whether it's complete or not. There can't be 3,5 or -9 requests in the system, so it's complete. Although with this representation we can't measure the distribution of the requests.

d. { Processing of the request has not yet started, The servers are working on the request, The request is processed }
Solution
No. This is a valid state space for one request (e.g. John Doe's request at 10:03 AM, Monday) if it's clear what we mean by "request" (and omitting the other request), because the system has to process many requests over time. If we are talking about the life cycle of a single uniquely identified request then this is a valid state space for that request.

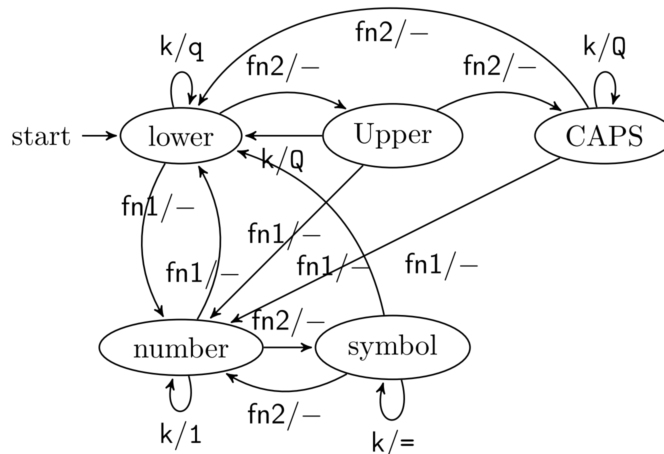


e. { True } ∗
Solution
Yes, every state space is a refinement of this state space. The *true* keyword is used in many formalisms to denote the state that *always* characterizes a system. Because of this the above set is complete. And since it only contains one state, it is mutually exclusive.

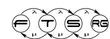
3 Touchscreen keyboard

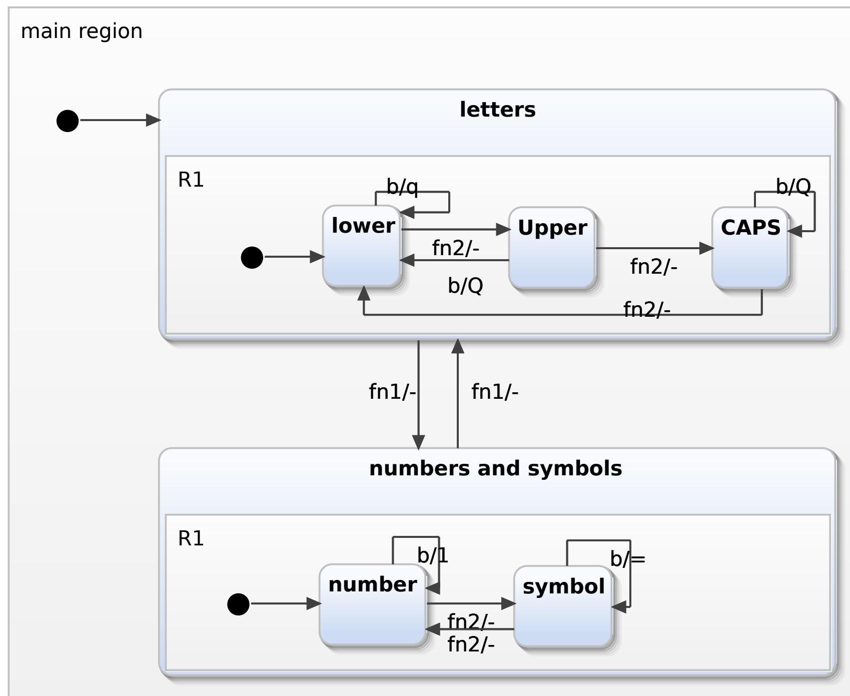
Model a touchscreen keyboard (designed for e.g. smartphones) with a state machine as presented during the lecture! The keyboard displays either lower case letters or capital letters or numbers and important symbols or rare symbols. The primary operation mode button switches between letters and numbers/symbols, the secondary operation mode button switches between the subcategories of the previous main categories. Moreover there is a capital letter state which changes automatically to lower case letters after a letter is pressed. Consider only the top left button (`q/Q/1/=`) and the two operation mode switching buttons as inputs, and consider the typed characters as outputs!

Solution
It's important that we display only one button on the state machine to help readability. Let `k` (for `key`) denote the upper left button, let `fn1` denote the primary operation mode button and let `fn2` denote the secondary operation mode button. A possible solution:



A hierarchical version of the solution:

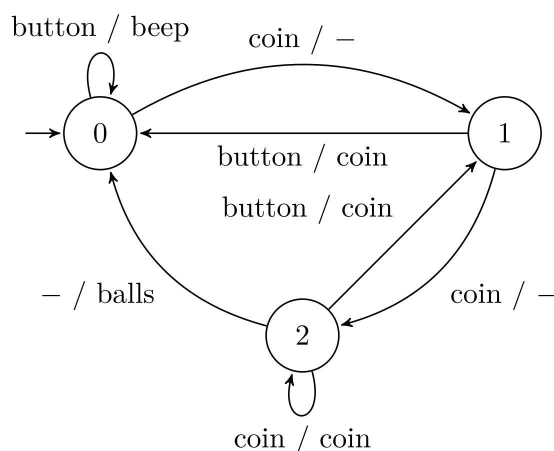
 



According to the current solution the state machine always transitions to the state lower when pressing fn in states number and symbol. However, in reality, advanced implementations can save the previous state of the casing mode (lower, Upper or CAPS) so they know the state to continue from.
This can be realized in this model by refining the number and symbol states (numbers with lower, numbers with Upper, numbers with CAPS, ...), or we use state charts for modelling (like Yakindu does) which permit the usage of history states that can remember the previous state in that region automatically.

4 „What is the output?"

Let's look at state machine M.



Note: the „–" symbol denotes a spontaneous transition when used as an input symbol, and a missing output when used as an output symbol. It's not the *don't care* symbol known from the digital design courses.

a. What real life system could be represented with this state machine? How does it work?

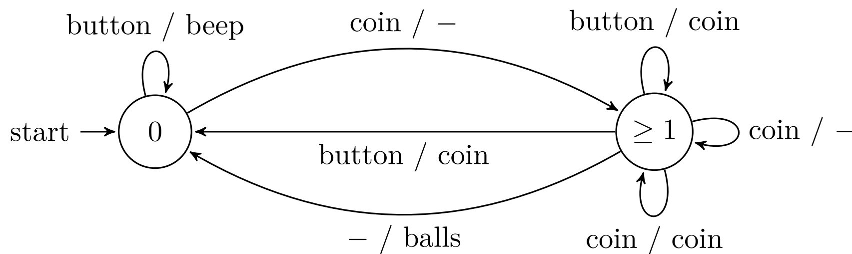
Solution

The state machine models the part of a coin-operated pool table where people insert the coins ("coin mechanism"). After inserting two coins (e.g. two dimes) the mechanism releases the balls. Releasing balls and inserting coins are modelled as atomic events (this is a simplification to make modelling easier). The states of the state machine represent the number of coins currently in the table. When pressing the button the table gives back the coin in it, or beeps if there is no coin in it. The system doesn't allow to have more than two coins in it. Once there are two coins in

the machine they are moved to the coin collector part of the table after a short time due to a *spontaneous transition* while at the same time the table releases the balls for the players. Before the release of the balls the players can quickly press the button to get back a coin if they have changed their minds.

b. Is the behaviour model deterministic? Can we add or remove a single transition rule to change this?
Solution
A state machine is deterministic if it has at most one initial state, and in every state for every event at most one transition can fire. We have to check every state for conflicting (i.e. for the same event two different transitions could fire) and spontaneous transitions. Due to the spontaneous transition from state 2 to state 0 the state machine is *non-deterministic*. Consider the following: we insert two coins then after 5 seconds we push the button; do we get back a coins or hear a beep? This depends on whether the system made the spontaneous transition (processed the coins) or not.
We can conclude that the model doesn't contain the information to decide which of the above possible outcomes will happen. It doesn't determine the behaviour completely. One reason for this could be that it determines the timeout randomly; an other reason could be that we lost the information about an exact timer while using abstraction.
Can we make the state machine deterministic? Yes, by deleting the $2 \to 0$ transition or making it explicit by introducing a second button as an event (button2)

c. Use abstraction on state machine M according to the state partitioning $\{\{0\}, \{1, 2\}\}$!
Solution
The solution is state machine M' with the following state graph:



State machine M' is created from state machine M by merging the states 1 and 2.
The resulting state machine is an abstraction of the original one thus it permits execution sequences (sequence of events) that the original one didn't. Some of these could be the behaviour of real systems: the abstract system is the common abstraction of tables operating with n number of coins ($n > 1$). However, it also permits event sequences that are invalid for real tables.

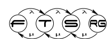
d. What kind of non-deterministic behaviours can be observed on the abstract model and where?
Solution
Aside from the spontaneous transition other sources of non-determinism are present: the button and coin events in state ≥ 1 can trigger more than one transitions.
These new sources were introduced because we used abstraction: we omitted the knowledge which would have helped us to make decisions in a deterministic way.

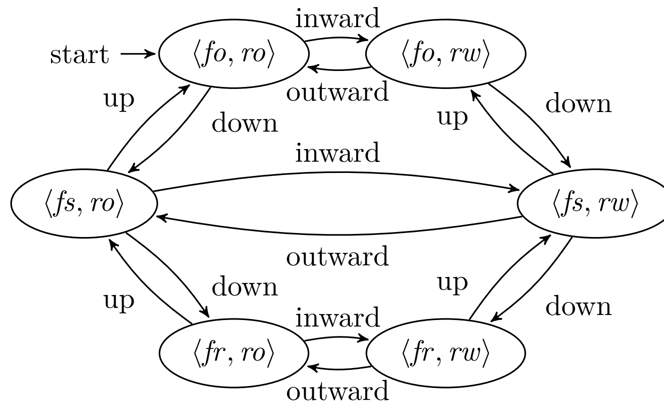
5 Windshield wiper

The front windshield wiper in a car has three states (*front off*, *front wipes slowly*, *front wipes rapidly*), the rear wiper has two states (*rear off*, *rear wipes*). The behaviour of the wipers are represented by the state machines M_1 and M_2, respectively.

a. Construct the asynchronous product of M_1 and M_2!
Solution
Notation:
- the "switch" part is omitted from the events.
- state names are abbreviated using the first letters of their words (except the word "wipes", e.g. front wipes rapidly $\to$ fr)

The asynchronous product:





Both state machines are explicitly present in the asynchronous product as many times as the number of states the other state machine has. Intuitively: we can put one state machine in an arbitrary state then we can "play" every behaviour of the other state machine and vice versa.

b. How many states and transitions does the resulting model have?

Solution

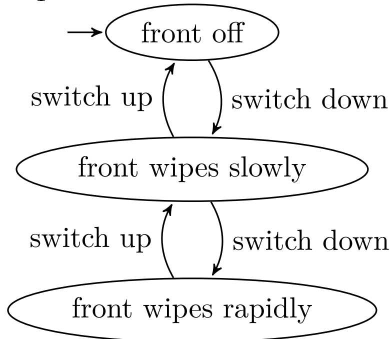
The number of states in the resulting model is the product of the number of states in the components. The number of transitions is $|S(M_1)| \cdot |T(M_2)| + |S(M_2)| \cdot |T(M_1)|$, where $S(M_1)$ is the set of states in M_1 and $T(M_1)$ is the set of transitions in M_1.

c. (Extra task) Can we represent the following behaviour in the product state machine, or in the state machines of the components: the rear wiper can only be switched on if the front wiper is on?

Solution

The product state machine can be easily modified accordingly – we simply omit the forbidden transitions. Using only the component-level models poses a problem since the behaviour of the two state machine is no longer independent. We can use a *guard condition* on the appropriate transition to express the following: the transition of M_2 with the "switch inward" event can be fired only if M_1 is not in the state "front off".

M_1 state machine:



M_2 state machine: