Discrete Event Systems
Sample Solution to Exercise 8

1 Safety Properties und Invarianten

1. A Petri-Net is deadlock-free if there is always at least one enabled transition in the Petri net. To show this, we can model the system as a labelled transition system (LTS) and then show that all states have an outgoing transition.

2. We have a deadlock if all philosophers are waiting for their forks. In our Petri Net, philosopher \( i \) waits when he is not eating (i.e., there is no token in \( eat_i \)) and there are not enough tokens to fire the transition \( takeFork_i \). Let us denote this situation by the proposition \( \text{wait}_i \). Hence the invariant we want to prove is \( \Psi = \neg (\text{wait}_1 \wedge \text{wait}_2 \wedge \text{wait}_3) \).

3. We transform the Petri net into an LTS via state space exploration. We label a state by \( d_1d_2d_3d_4d_5d_6 \) where each \( d_i \) represents one place in the Petri net. The value of \( d_i \) shall indicate the number of tokens that are currently in the place corresponding to \( d_i \). In particular, we represent by \( d_1-d_3 \) the places \( \text{Fork}_1-\text{Fork}_3 \) and by \( d_4-d_6 \) the places \( \text{eat}_1-\text{eat}_3 \). E.g., \( s_0 = 111000 \) represents the initial state where all \( \text{Fork} \) places have one token. To find the next states in our LTS, we explore all possible combinations of transitions that can be fired from \( s_0 \). All transitions \( \text{takeFork}_i \) are enabled, however, only one of them can fire. This gives three possible LTS transitions \( T_1-T_3 \) and leads to three new states \( s_1 = 001100, s_2 = 100010 \) and \( s_3 = 010001 \). \( T_i \) can be interpreted as philosopher \( i \) taking the two forks. From each of these states, there is only one possible transition. If we are in \( s_2 \), e.g., there is a token in \( \text{eat}_2 \) and \( \text{Fork}_1 \). Thus \( \text{putForkDown}_2 \) is the only enabled transition in our Petri net. If we fire this transition, we get to the initial state again. In fact, all states \( s_1-s_3 \) have an LTS transition back to \( s_0 \). You can interpret a transition \( T_{-i} \) as philosopher \( i \) putting down his forks.

![LTS Diagram]

To see that our model is deadlock-free, we just have to realize, that each state in the constructed LTS has at least one outgoing transition. To proof the invariant \( \Psi \), we can translate
all states in the LTS to propositions. In the initial state $s_0$, no philosopher is waiting. In states $s_1$-$s_3$, one philosopher eats and the two others are waiting. In any state, $\Psi$ is true. The model is deadlock-free.

2 LTL-Properties

1. Please note that there was one inhibitor arc from state yellow to the transition $switch_1$ missing in the exercise sheet. This prevents $switch_1$ from firing when there are tokens in yellow.

- **Transform Petri net into LTS.** We again do a state exploration starting from the initial state where there is one token in red. We denote the LTS states by $d_1d_2d_3$ where $d_1$ corresponds to the number of tokens in red, $d_2$ to the tokens in yellow and $d_3$ to the tokens in green.

The initial state $s_0$ is 100 and there is one transition, $switch_1$ enabled. We fire $switch_1$ and get one token in red and one token in yellow, thus $s_1 = 110$. Now $switch_2$ is the only transition enabled. We fire it and get one token in green, $s_2 = 001$. Firing $switch_3$ yields one token in yellow, $s_3 = 010$. Firing $switch_4$ gets us back to the initial state $s_0$.

- **Negate property.** The LTL formulas we want to check are $\boxdot green$ (read: “always eventually green”) meaning that at any point in time, we can be sure that the lights will turn green, and $\Diamond \boxdot green$ (read “eventually always green”) meaning that after some initial time period, the light will always stay green. We build the inverse using the LTL equivalence rules:

$$
\neg \varphi_1 = \neg (\boxdot green) = \Diamond \neg green
$$

$$
\neg \varphi_2 = \neg (\Diamond \boxdot green) = \Box \Diamond \neg green
$$

- **Construct NBA.** We construct the non-deterministic Buechi automata (NBA) for $\neg \varphi_1$ and $\neg \varphi_2$.

- **Construct product automaton.** We construct the Cartesian product of the LTS and the NBA.

In Figure 1, the automaton on the left depicts the Cartesian product for $\neg \varphi_1$ and the automaton on the right the one for $\neg \varphi_2$. On the vertical axis we have the states of the LTS and on the horizontal we have the states of the NBA. Notice that not all power states are reachable. We construct the product by combining the proposition that holds in the current LTS state with the possible transitions in the NBA. E.g., in the initial state $(s_0, q_0)$, $\neg green$ holds thus we can take the NBA transitions $\neg green$ and true.
(since true is always true $\mathbf{\bigcirc}$). In the Cartesian product, this results in a transition from state $(s_0, q_0)$ to $(s_1, q_1)$ and to $(s_1, q_0)$ respectively.

Once we have the Cartesian product automaton, we check whether there exists a cycle which contains an accepting state. If this is the case, we can conclude that the LTL-formula we are checking does not hold. If no cycle contains an accepting state, we may derive that the LTL-formula holds. In the Cartesian product for the formula $\neg \varphi_1$, there are two cycles (colored blue in the illustration). Both do not contain any accepting state, hence we know that $\varphi_1$ holds. In the Cartesian product for the formula $\neg \varphi_2$, there is one cycle containing accepting states, hence we know that $\varphi_2$ is false.

2. In order to check the LTL-formula $\Phi$, we have to first compute the inverse of $\Phi$.

$$
\neg \Phi = \neg \Box (\mathit{notSent} \Rightarrow \bigcirc \mathit{sent}) \\
= \bigcirc \neg (\mathit{notSent} \Rightarrow \bigcirc \mathit{sent}) \\
= \bigcirc (\mathit{notSent} \land \neg \Box \mathit{sent}) \\
= \bigcirc (\mathit{notSent} \land \neg \mathit{sent})
$$

For the third equality, we used the logic equivalence rule $\neg(a \Rightarrow b) = a \land \neg b$. Informally, $\neg \Phi$ means that at some point in time, both $\mathit{notSent}$ and $\neg \mathit{sent}$ hold and after that, $\mathit{notSent}$ may become false again, but $\mathit{sent}$ must stay false forever. We now represent $\neg \Phi$ by an NBA.

NBA of $\neg \Phi$:

The Cartesian product of the LTS and the NBA looks as follows:
The product contains one cycle with an accepting state, namely the self loop of \((s_1, q_1)\). This provides us with an infinite run that fulfills \(\neg \Phi\). Hence we have found a counter-example which proves that \(\Phi\) does not hold. In the LTS, starting in \(s_1\) and taking the transition \(\text{doSomething}\) infinitely often, we will never establish \(\text{sent}\). This disproves \(\Phi\) which says that it is always the case that when “a message” is not sent, it will sooner or later be sent.

3 Taskaktivierungsmuster und Timed Automata

See LoesungAufgabe3.pdf

4 Scheduling und Timed Automata

See UppaalModel_Aufg4.xml