1 Labelled Graphs

a) We give two algorithms, an iterative and a recursive one, that calculate whether the given LTS $\mathcal{L}$ accepts the word $\omega = w_1 \ldots w_n$.

**Algorithm 1 AcceptIterative($\mathcal{L}, \omega$)**

**Input:** LTS $\mathcal{L} = (S, S_0, Act, E)$, $\omega = w_1 w_2 \ldots w_n$

1. $states \leftarrow S_0$ \hspace{1cm} \triangleright contains the states reachable by $w_1 \ldots w_{i-1}$
2. for $i \leftarrow 1$ to $n$
   1. newStates $\leftarrow \emptyset$ \hspace{1cm} \triangleright contains the new states reachable by $w_1 \ldots w_i$
      1. for all $v \in states$
         1. for all $c \in PostSetNodes(v)$
            1. if $Act((v, c)) = w_i$ then
               1. newStates $\leftarrow$ newStates $\cup \{c.target()\}$ \hspace{1cm} \triangleright If the label matches...
               1. if newStates $= \emptyset$ then return false \hspace{1cm} \triangleright If no edge with label $w_i$ exists...
3. $states \leftarrow$ newStates
4. return true

**Algorithm 2 AcceptRecursive($\mathcal{L}, states, \omega$)**

**Input:** LTS $\mathcal{L} = (S, S_0, Act, E)$, $states$: set of states, $\omega = w_1 w_2 \ldots w_n$

1. newStates $\leftarrow \emptyset$
2. if $\omega = \emptyset$ then return true \hspace{1cm} \triangleright Every letter of the word has been matched to a path.
3. else if $states = \emptyset$ then return false \hspace{1cm} \triangleright No state was reachable by the last letter.
4. for all $v \in states$
   1. for all $c \in PostSetNodes(v)$
      1. if $Act((v, c)) = w_1$ then
         1. newStates $\leftarrow$ newStates $\cup \{c\}$ \hspace{1cm} \triangleright If the label matches...
5. $\omega \leftarrow w_2, \ldots, w_n$ \hspace{1cm} \triangleright Remove first letter of $\omega$
6. return AcceptRecursive($\mathcal{L},$ newStates, $\omega$) \hspace{1cm} \triangleright Recursive call for the remaining word

The initial call is AcceptRecursive($\mathcal{L}, S_0, \omega$).
2 Structural Properties of Petri Nets and Token Game

a) The pre and post sets of a transition are defined as follows:
   - pre set: \( \text{pre: } t := \{ p \mid (p, t) \in C \} \)
   - post set: \( \text{post: } t := \{ p \mid (t, p) \in C \} \),
the pre and post sets of a place are defined analogously.

For the petri net \( N_1 \) we obtain the following sets:
   - \( t_5 = \{ p_5, p_9 \} \), \( t_5^\bullet = \{ p_6 \} \)
   - \( t_8 = \{ p_8 \} \), \( t_8^\bullet = \{ p_{10}, p_5 \} \)
   - \( p_3 = \{ t_2 \} \), \( p_3^\bullet = \{ t_3 \} \)

b) A transition is enabled if all places in its pre set contain enough tokens. In the case of \( N_1 \), which has only unweighted edges, one token per place suffices. When \( t_2 \) fires, it consumes one token out of each place in the pre set of \( t_2 \) and produces one token on each place in the post set of \( t_2 \). Hence, the firing of \( t_2 \) produces one token on place \( p_3 \) and \( p_9 \) each, the one on \( p_2 \) is consumed. After this, \( t_5 \) is enabled because both \( p_9 \) and \( p_5 \) hold one token. However, \( t_3 \) is not enabled because \( p_3 \) contains a token but \( p_{10} \) does not.

c) Before \( t_2 \) fires there are two tokens in \( N_1 \), one on \( p_2 \) and \( p_5 \) each. Directly afterwards, there are tokens on places \( p_3 \), \( p_9 \) und \( p_5 \).

d) A token traverses the upper cycle until \( t_2 \) fires. Then one token remains on \( p_3 \) and waits, and another one is produced in \( p_9 \), which enables transition \( t_5 \). When \( t_5 \) consumes the tokens on \( p_9 \) and \( p_5 \) and produces a token on \( p_6 \), this one can traverse the lower cycle until \( t_8 \) is enabled. One token now remains on \( p_5 \) and waits, another one enables \( t_3 \), because there is still one token on \( p_3 \). Now one token traverses the upper cycle again until \( t_2 \) is enabled, and so on.

Hence, this petri net models two processes which always appear alternately.

The reachability graph \( RG(P, s_0) \) of a petri net \( P \) is a quadruple \((S, S_0, Act, E)\) such that
   - \( S \) is the set of reachable states of \( P \) starting from \( s_0 \)
   - \( S_0 := \{ s_0 \} \) is the start state of \( P \)
   - \( Act \) is the set of transition labels
   - \( E \subseteq S \times Act \times S \) is the set of edges such that \( E = \{ (s, t, \delta(s, t)) \mid s \in S \land t \in T \land \delta \triangleright t \} \)

Usually the states of the petri net are denoted by vectors such that the \( i \)-th position in the vector indicates the number of tokens on place \( p_i \) of the petri net. So, for example, the starting state \( s_0 \) of \( N_1 \), in which the places \( p_1 \) and \( p_5 \) hold one token each, is denoted by
\( s_0 = (1, 0, 0, 0, 1, 0, 0, 0, 0, 0) \). Hence, the reachability graph looks as follows:

\[
S = \{ (1, 0, 0, 0, 1, 0, 0, 0, 0, 0), (0, 1, 0, 0, 1, 0, 0, 0, 0, 0), (0, 0, 1, 0, 0, 1, 0, 0, 0, 0), (0, 0, 1, 0, 0, 0, 1, 0, 0, 0), (0, 0, 1, 0, 0, 0, 0, 1, 0, 0), (0, 0, 1, 0, 0, 0, 0, 0, 1, 0) \},
\]

\( S_0 = \{ (1, 0, 0, 0, 1, 0, 0, 0, 0, 0) \} \),

\( Act = \{ t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10} \} \),

\( E = \{ ((1, 0, 0, 0, 1, 0, 0, 0, 0, 0), t_1, (0, 1, 0, 0, 1, 0, 0, 0, 0, 0)), ((0, 1, 0, 0, 1, 0, 0, 0, 0, 0), t_2, (0, 0, 1, 0, 1, 0, 0, 0, 1, 0)), ((0, 0, 1, 0, 1, 0, 0, 0, 1, 0), t_5, (0, 0, 1, 0, 0, 1, 0, 0, 0, 0)), ((0, 0, 1, 0, 0, 1, 0, 0, 0, 0), t_6, (0, 0, 1, 0, 0, 0, 1, 0, 0, 0)), ((0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0), t_7, (0, 0, 0, 1, 0, 0, 0, 0, 1, 0)), ((0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0), t_8, (0, 0, 1, 0, 1, 0, 0, 0, 0, 1)), ((0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1), t_3, (0, 0, 0, 1, 1, 0, 0, 0, 0, 0)), ((0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0), t_4, (1, 0, 0, 0, 1, 0, 0, 0, 0, 0)) \} \).

For better legibility we denote the states in such a way that the index contains the places that hold a token in this state, for example \( s_0 = (1, 0, 0, 0, 1, 0, 0, 0, 0, 0) = s_{1.5} \).

Then the reachability graph can also be specified as follows:

3 Basic Properties of Petri Nets
See exercise sheet 13.

4 Reachability Analysis for Petri Nets
See exercise sheet 13.

5 Mutual Exclusion
For each process we introduce two places \( p_1, p_2, p_3 \) und \( p_4 \) representing the process within the normal program execution \( p_1, p_2 \) as well as in the critical section \( p_3, p_4 \). For each process, we have a token indicating which section of the program currently is executed. Additionally, we introduce a place \( p_0 \) representing the mutex variable. If the mutex variable is 0, then we have a token at \( p_0 \). We have to make sure that a process can only enter its critical section if there is a token at the mutex place. The resulting petri net looks as follows.
Assume that initially, both processes are in an uncritical section. A process can only enter its critical section if there is a token at $p_0$. In this case, the token is consumed when entering the critical section. A new mutex token at $p_0$ is not created until the process leaves its critical section. Hence, both processes exclude each other mutually from the concurrent access to the critical section.