T-79.186 Reactive Systems: Finite State Automata, part II

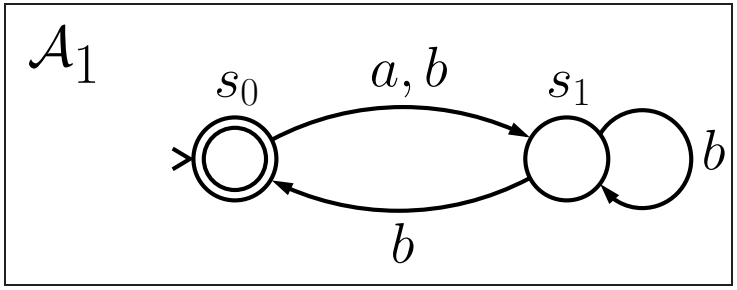
Spring 2005, Lecture 2

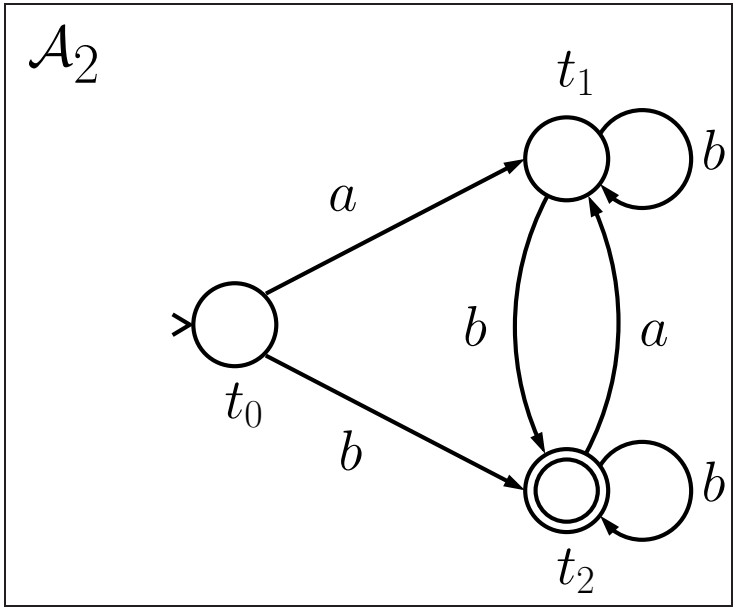
Keijo Heljanko

January 31, 2005

Example: Operations on Automata

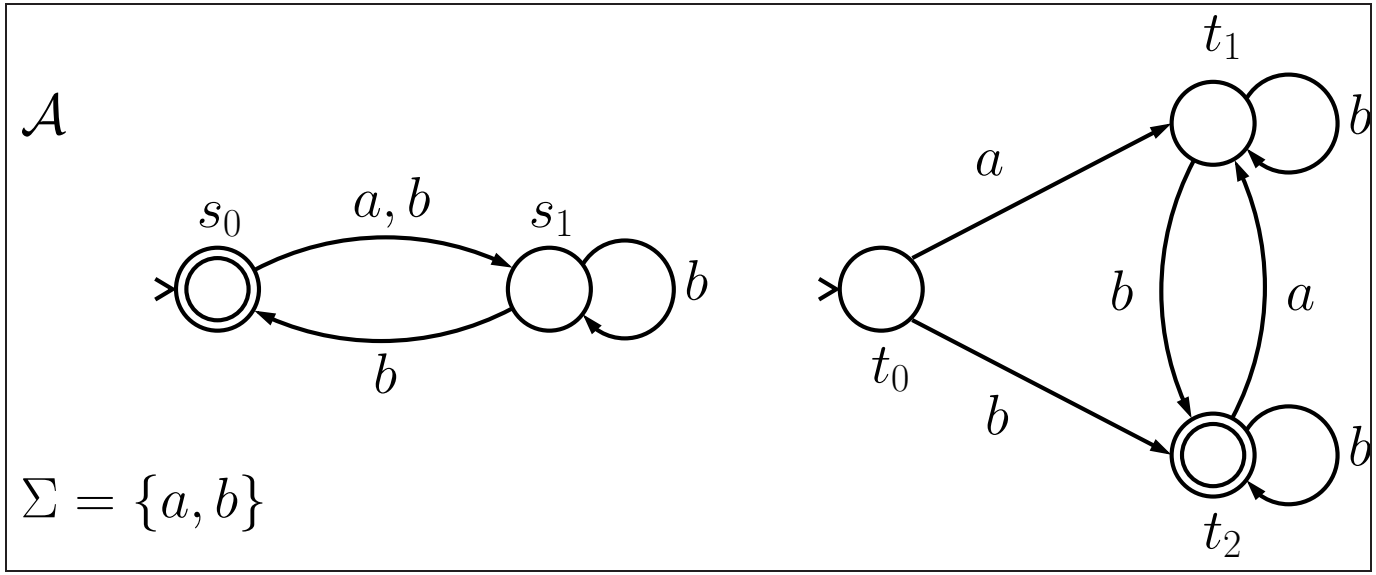
Consider the following automata $\mathcal{A}_1$ and $\mathcal{A}_2$, both over the alphabet $\Sigma = \{a, b\}$.





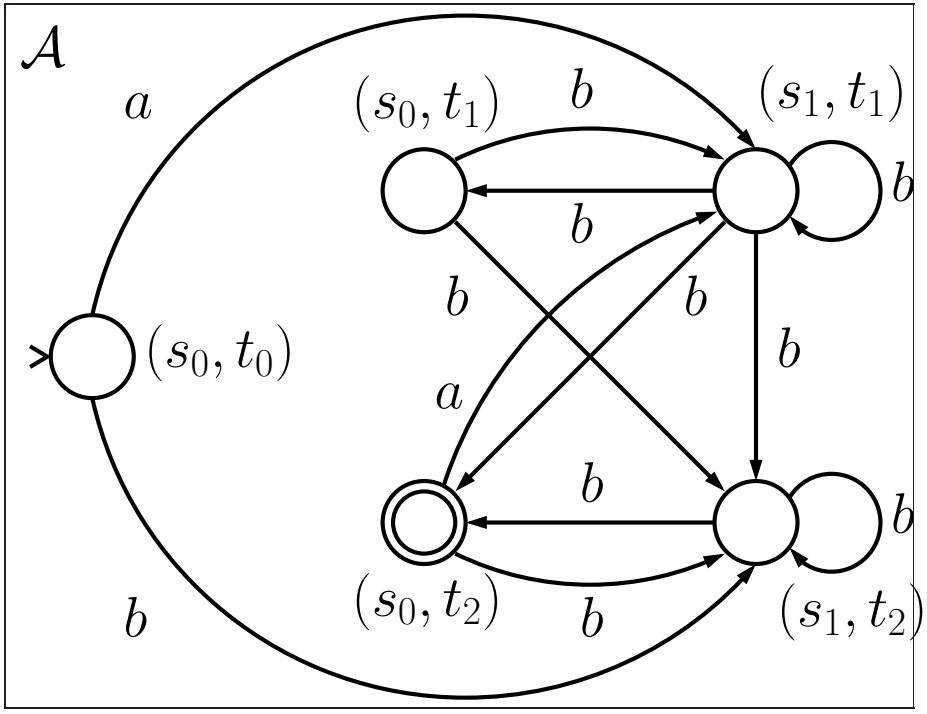
Example: Union of Automata

The following automaton $\mathcal{A}$ is their union, in other words $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$.



Example: Intersection of Automata

The following automaton $\mathcal{A}$ is their intersection, in other words $\mathcal{A} = \mathcal{A}_1 \cap \mathcal{A}_2$.



Complementing Nondeterministic Automata

The operations we have defined for finite state automata so far have resulted in automata whose size is polynomial in the sizes of input automata.

The most straightforward way of implementing complementation of a non-deterministic automaton is to first determinize it, and after this to complement the corresponding deterministic automaton.

Unfortunately determinization yields an exponential blow up. (A worst-case exponential blow-up is in fact unavoidable in complementing non-deterministic automata.)

Determinization of finite state automata can be done as follows:

Definition 1 Let $\mathcal{A}_1 = (\Sigma, S_1, S_1^0, \Delta_1, F_1)$ be a non-deterministic automaton. We define a deterministic automaton $\mathcal{A} = (\Sigma, S, S^0, \Delta, F)$, where

- $S = 2^{S_1}$, the set of all sets of states in S_1,
- $S^0 = \{S_1^0\}$, a single state containing all the initial states of $\mathcal{A}_1$,
- $(Q, a, Q') \in \Delta$ iff
  $Q \in S, a \in \Sigma$, and $Q' = \{s' \in S_1 \mid$ there is $(s, a, s') \in \Delta_1$ such that $s \in Q\}$; and
- $F = \{s \in S \mid S \cap F_1 \neq \emptyset\}$, those states in S which contain at least one accepting state of $\mathcal{A}_1$.

The intuition behind the construction is that it combines all possible runs on given input word into one run, where we keep track of all the possible states we can currently be in by using the "state label". (The automaton state consists of the set of states in which the automaton can be in after reading the input so far.)

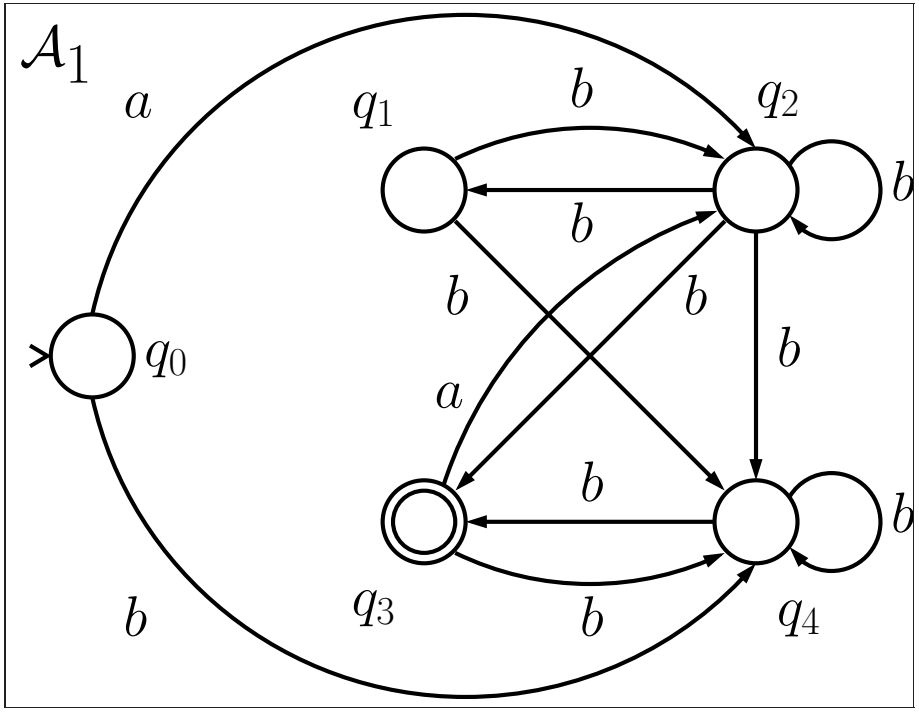
We denote the construction of the previous slide with $\mathcal{A} = det(\mathcal{A}_1)$ Note that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1)$, and $\mathcal{A}$ is deterministic. If $\mathcal{A}_1$ has n states, the automaton $\mathcal{A}$ will contain 2^n states.

Note also that the determinization construction gives an automaton $\mathcal{A}$ with a completely specified transition relation as output. Thus to complement an automaton $\mathcal{A}_1$, we can use the procedure $\mathcal{A} = det(\mathcal{A}_1)$, $\mathcal{A}' = \overline{\mathcal{A}}$, and we get that $\mathcal{L}(\mathcal{A}') = \Sigma^* \setminus \mathcal{L}(\mathcal{A}) = \Sigma^* \setminus \mathcal{L}(\mathcal{A}_1) = \overline{\mathcal{L}(\mathcal{A}_1)}$.

To optimize the construction slightly, usually only those states of $\mathcal{A}$ which are reachable from the initial state are added to set of states set of $\mathcal{A}$.

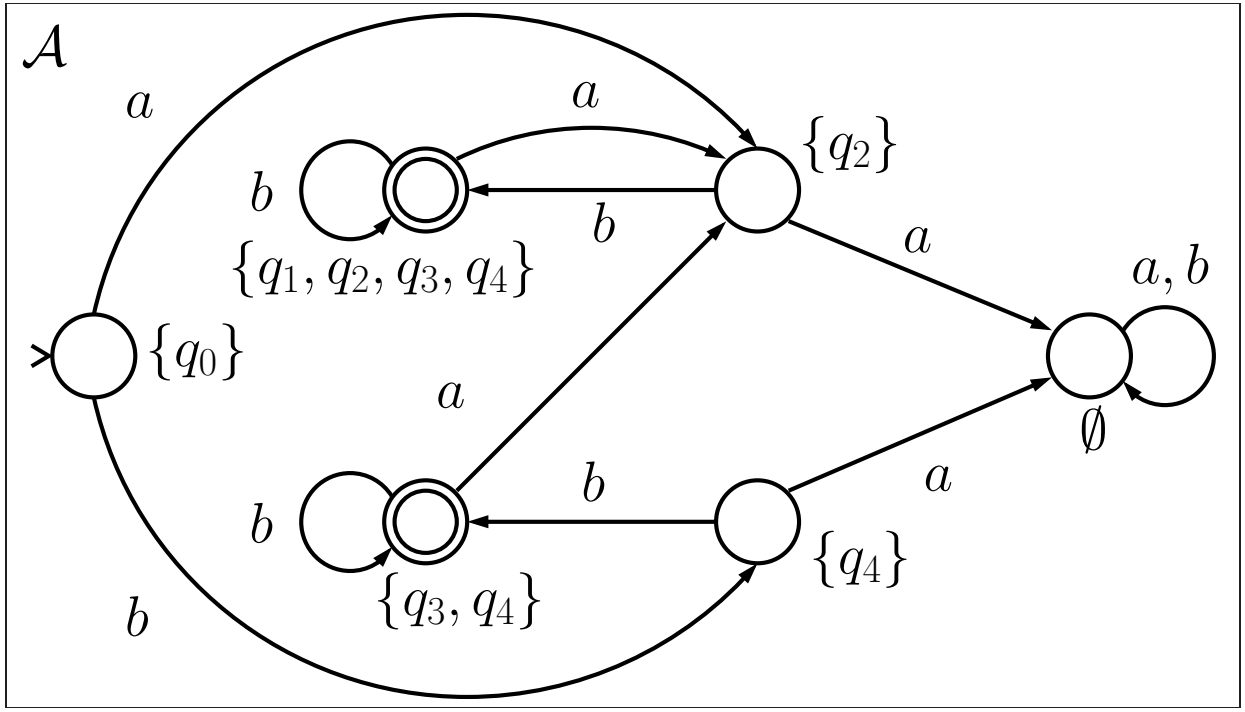
Example: Determinization of Automata

We want to determinize the following automaton $\mathcal{A}_1$ over the alphabet $\Sigma = \{a, b\}$.



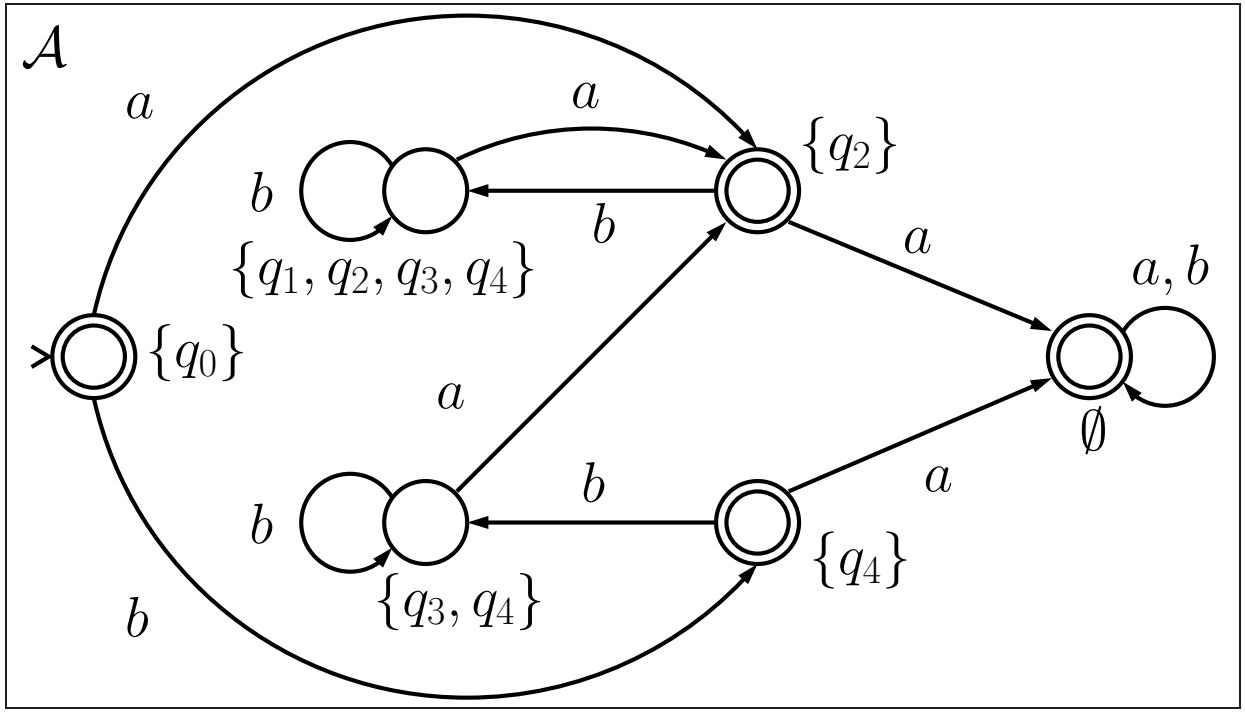
Example: Determinization Result

As a result we obtain the automaton $\mathcal{A}$ below. (Only the reachable part shown!)



Example: Complementation

Let's call the result of the previous slide $\mathcal{A}_1$, and complement the result. We get:



We have now shown that finite state automata are closed under all Boolean operations, as with $\cup$, $\cap$, and $\overline{\mathcal{A}}$ all other Boolean operations can be done.

All operations except for determinization (which is also used to complement nondeterministic automata!) created a polynomial size output in the size of the inputs.

Note, however, that even if $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$ have k states each, the automaton $\mathcal{A}'_4 = \mathcal{A}_1 \cap \mathcal{A}_2 \cap \mathcal{A}_3 \cap \mathcal{A}_4$ (alternatively denoted by $\mathcal{A}'_4 = \mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3 \times \mathcal{A}_4$) can have k^4 states, and thus in the general $\mathcal{A}'_i$ will have k^i states. Therefore even if if a single use of $\cap$ is polynomial, repeated applications often will result in a state explosion problem.

In fact, the use of $\times$ as demonstrated above (or some slight variation of the definition) is the main way of composing a reactive system out of its components when using an automata based modeling formalism.

Checking Safety Properties with FSA

A safety property be informally described as a property stating that "nothing bad should happen". (We will come back to the formal definition later in the course.)

When checking safety properties, the behavior of a system can be described by a finite state automaton, call it $\mathcal{A}$.

Also in most cases the allowed behaviors of the system can be specified by another automaton, call it the specification automaton $\mathcal{S}$.

Assume that the specification specifies all legal behaviors of the system. In other words a system is incorrect if it has some behavior (accepts a word) that is not accepted by the specification. In other words a correct implementation has less behavior than the specification, or more formally $L(\mathcal{A}) \subseteq L(\mathcal{S})$.

Checking whether $L(\mathcal{A}) \subseteq L(\mathcal{S})$ holds is referred to as performing a language containment check.

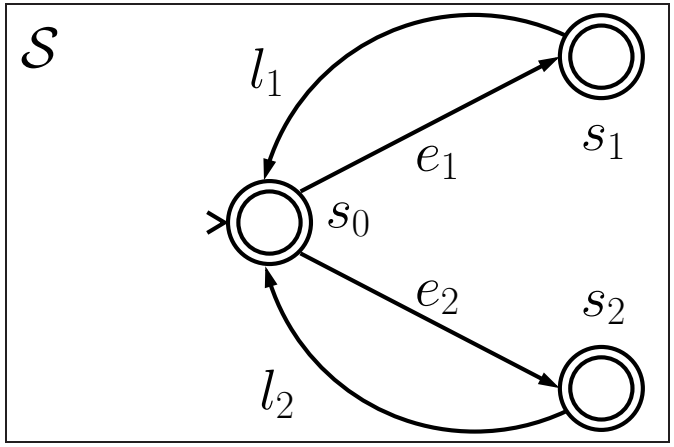
By using simple automata theoretic constructions given above, we can now check whether the system meets its specification. Namely, we can create a product automaton $\mathcal{P} = \mathcal{A} \cap \overline{\mathcal{S}}$ and then check whether $L(\mathcal{P}) = \emptyset$.

In case the safety property does **not** hold, the automaton $\mathcal{P}$ has a counterexample run r_p which accepts a word w, such that $w \in L(\mathcal{A})$ but $w \notin L(\mathcal{S})$.

By projecting r_p on the states of $\mathcal{A}$ one can obtain a run of r_a of the system (a sequence of states of the system) which violates the specification $\mathcal{S}$.

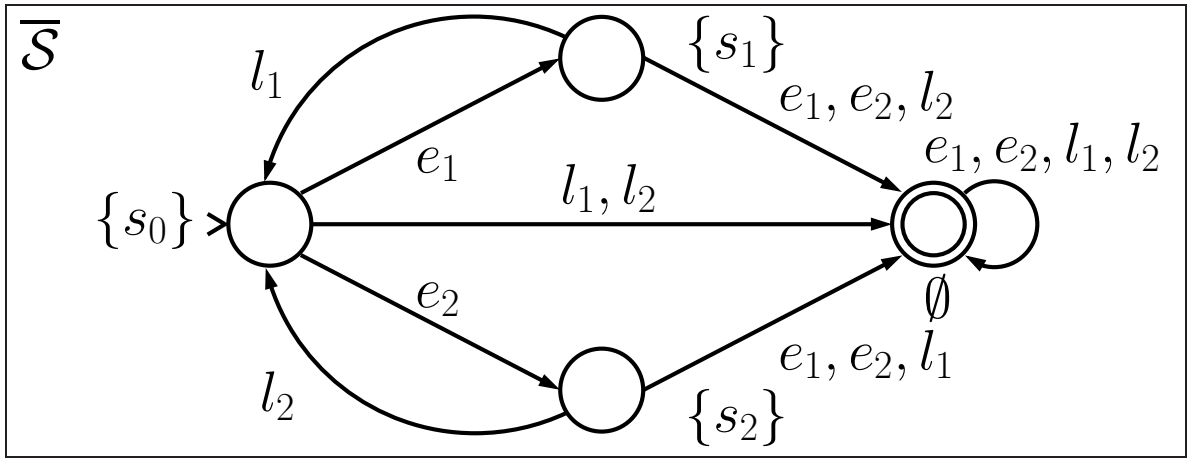
Example: Safety Property

Consider the problem of mutual exclusion. Assume that the alphabet is $\Sigma = \{e_1, e_2, l_1, l_2\}$, where e_1 means that process 1 enters the critical section and l_1 means that process 1 leaves the critical section. (For simplicity we assume these are the only actions possible in the system.) The automaton S specifying correct mutual exclusion property is the following.

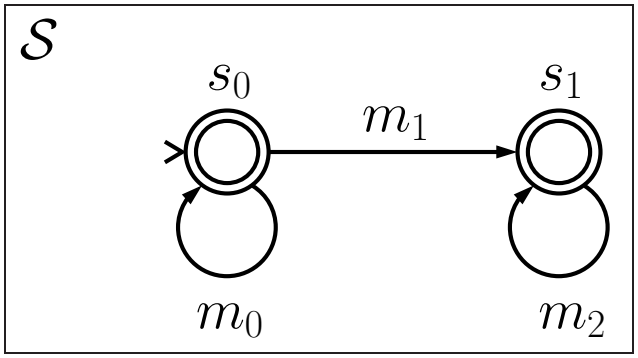


Example: Safety Property

If we want to check whether $L(\mathcal{A}) \subseteq L(\mathcal{S})$, we need to complement $\mathcal{S}$. We get the following:



If we now have an automaton $\mathcal{A}$ modeling the behavior of the mutex system, we can create the product automaton $\mathcal{P} = \mathcal{A} \cap \overline{det(\mathcal{S})}$. Now the mutex system is correct iff the automaton $\mathcal{P}$ does not accept any word.

Example 2: Safety Property

Assume we are testing a data-communications protocol for message duplication. We have already added as a data-source a sender, whose sends an arbitrarily long sequence of message m_0 messages, followed by a single m_1, followed by an arbitrary number of m_2 messages. If the data communication protocol does not look at its payload, we can see if the protocol duplicates messages by checking whether the data stream read by the receiver is within the following language over the alphabet $\Sigma = \{m_0, m_1, m_2\}$: