# Operations on automata

The purpose:

- We want to analyze automata
- We want to modify automata
- We want to combine automata

# Accessible part

States that never can be reached are clearly unnecessary.

As well as transitions associated with such states.

The operation for deleting these unnecessary states and transitions is denoted $Ac(A)$.

The $Ac(A)$ operation has no effect on $\mathcal{L}(A)$ or $\mathcal{L}_m(A)$

The term *reachable* is also used.

Relevant for cleaning up an automaton composed of several automata.

# Coaccessible part

A state $q$ of an automaton $A$ is said to be *coaccessible* if there is a string $s$ that takes us from $q$ to a marked state, that is $\delta_A(q, s) \in M_A$.

We denote the operation of deleting all the states of $A$ that are *not* coaccessible by $CoAc(A)$

The $CoAc$ operation may shrink $\mathcal{L}(A)$ but does not affect $\mathcal{L}_m(A)$

If $A = CoAc(A)$ then $A$ is said to be coaccessible.

If an automaton is nonblocking then it also have to be coaccessible. If there is no path from every state to a marked state then it can't be nonblocking.

# Trim operation

An automaton that is both accessible and coaccessible is said to be *trim*.

We define the trim operation as

$$Trim(A) := CoAc(Ac(A)) = Ac(CoAc(A))$$

It does not matter in which order $Ac$ and $CoAc$ is applied.

# Complement

Suppose we have a trim automaton $A = \langle Q_A, \Sigma_A, \delta_A, i_A, M_A \rangle$ that marks the language $L \subseteq \Sigma_A^*$

We can build another complement automaton that marks $\Sigma_A^* \setminus L$, which we denote $A^{comp}$.

1. Add an unmarked state $q_d$, called "dump" or "dead" state.

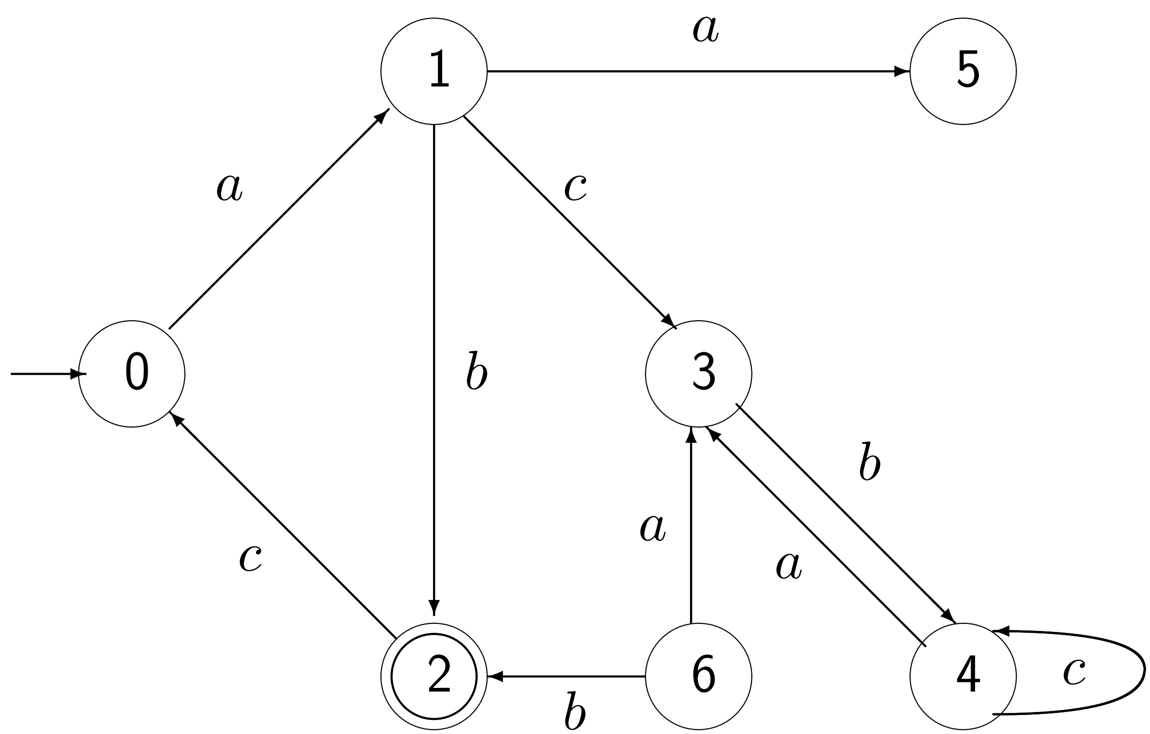
2. Complete the transition function $\delta_A$ of $A$ and make it a total function, $\delta_A^{tot}$, by assigning all undefined $\delta_A(q, e)$ in $A$ to $q_d$. Furthermore $\delta_A^{tot}(x_d, e) = x_d$ for all events $e \in \Sigma_A$.

3. Mark all unmarked states (including $q_d$), and unmark all marked states.

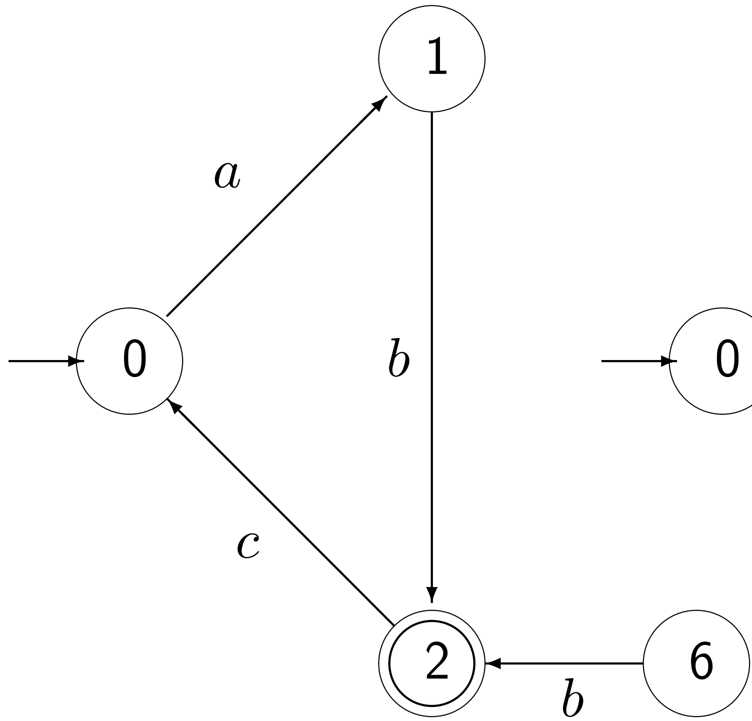
$$A^{comp} = \langle Q_A \cup \{x_d\}, \Sigma_A, \delta_A^{tot}, i_A, (Q_A \cup \{x_d\}) \setminus M_A \rangle$$

$\mathcal{L}(A^{comp}) = \Sigma_A^*$ and $\mathcal{L}_m(A^{comp}) = \Sigma_A^* \setminus \mathcal{L}_m(A)$ , as desired.

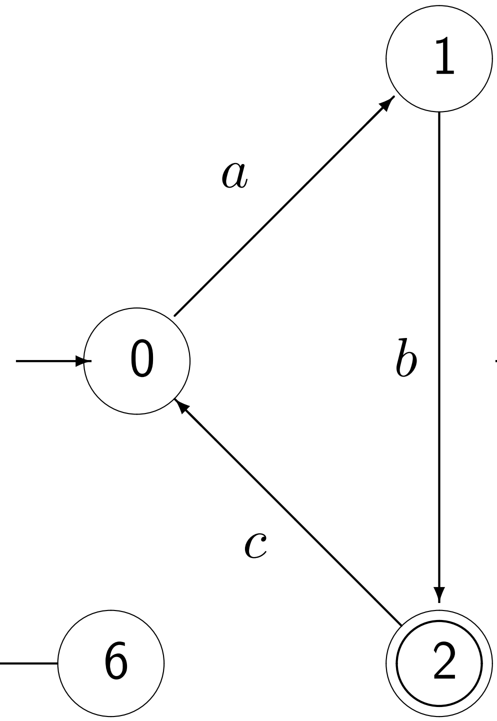
**Example 13.** Consider the automaton $A$ given below, previously used to illustrate deadlock and livelock.



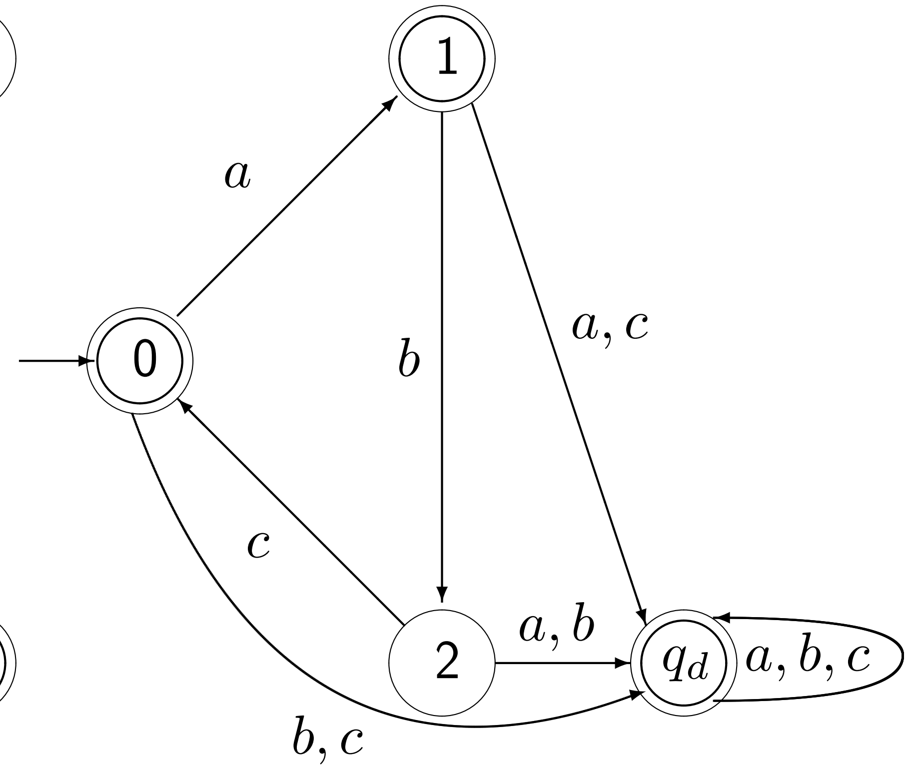

The new state 6 is clearly not accessible, $Ac(A)$ is obtained by removing it.




$CoAc(A)$




$Trim(A)$




Complement of $Trim(A)$

# Composition operations

We need operations for combining automata

For example a controller in feedback with a model

Two operations are considered

1. Parallel composition, denoted $||$. Sometimes called synchronous composition.

2. Product, denoted $\times$. Sometimes called completely synchronous composition.

We will use the automata $A = \langle Q_A, \Sigma_A, \delta_A, i_A, M_A \rangle$ and $B = \langle Q_B, \Sigma_B, \delta_B, i_B, M_B \rangle$ for illustration.

# Product

The product of $A$ and $B$ is the automaton

$$A \times B := Ac\langle Q_A \times Q_B, \Sigma_A \cap \Sigma_B, \delta, i_A.i_B, M_A \times M_B\rangle$$

where

$Q_A \times Q_B$ is the combination of all states. If $Q_A = \{a_1, a_2\}$ and $Q_B = \{b_1, b_2\}$ then $Q_A \times Q_B = \{a_1.b_1, a_1.b_2, a_2.b_1, a_2.b_2\}$

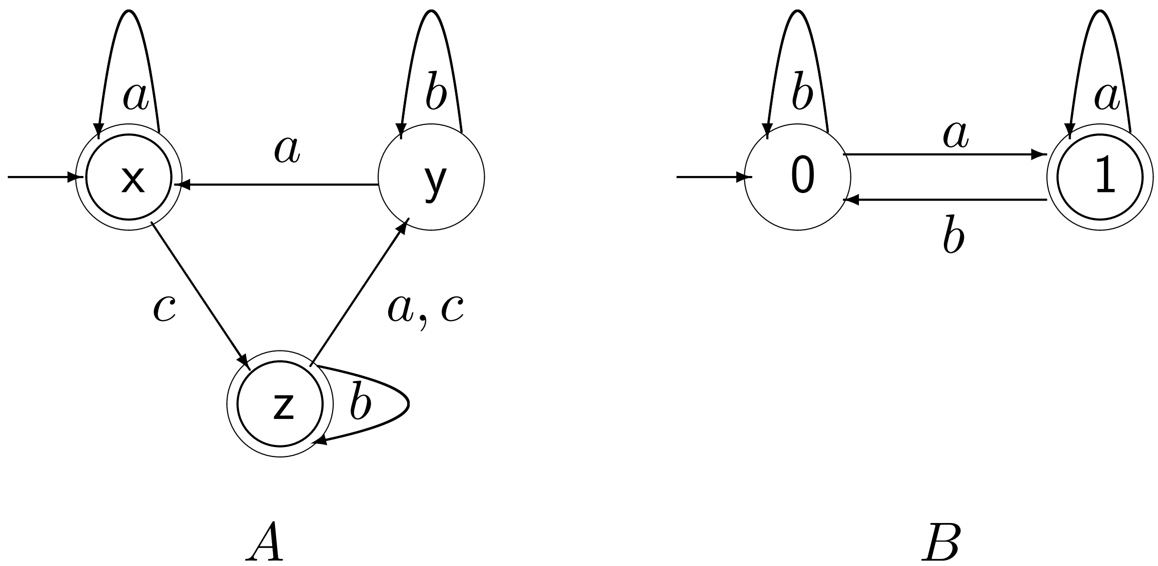
$$\delta(q_A.q_B, e) := \begin{cases} \delta_A(q_A, e).\delta_B(q_B, e) & \text{if } \delta_A(q_A, e) \text{ and } \delta_B(q_B, e) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$M_A \times M_B$ is combination of all marked states. Combination of a marked and an unmarked state is unmarked.

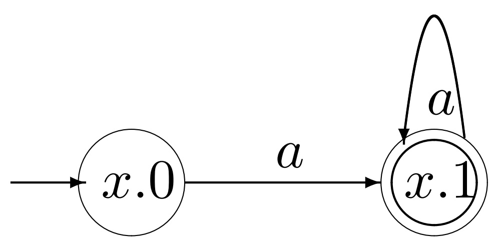
An event may occur if and only if it occurs in both automata, the events are completely synchronized.

$$\begin{aligned} \mathcal{L}(A \times B) &= \mathcal{L}(A) \cap \mathcal{L}(B) \\ \mathcal{L}_m(A \times B) &= \mathcal{L}_m(A) \cap \mathcal{L}_m(B) \end{aligned}$$

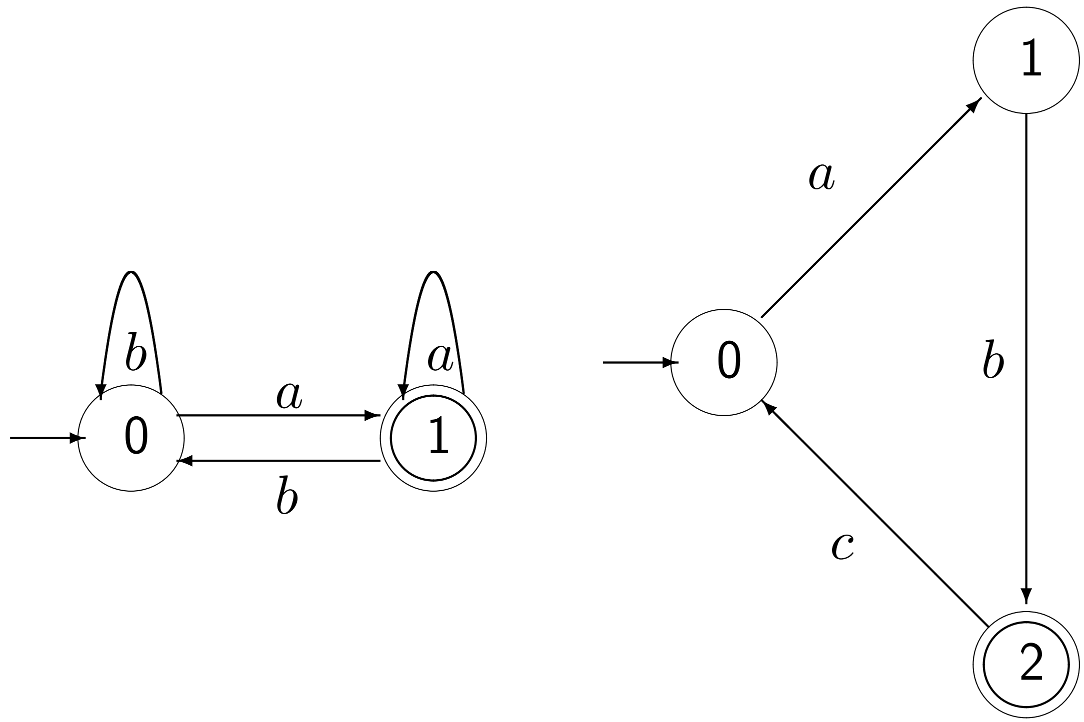
**Example 14.** Consider the following two automata




The product of $A$ and $B$ is the automaton

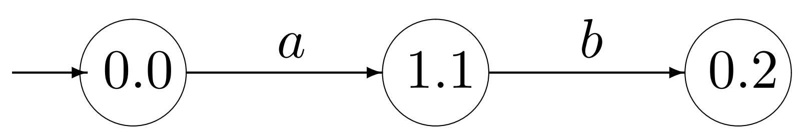


**Example 15.** Consider the following two automata




$B$ $\qquad\qquad$ $C$

The product of $B$ and $C$ is the automaton



# Parallel composition

The parallel composition of $A$ and $B$

$$A\|B := Ac\langle Q_A \times Q_B, \Sigma_A \cup \Sigma_B, \delta, i_A.i_B, M_A \times M_B\rangle$$

where

$$\delta(q_A.q_B, e) := \begin{cases} \delta_A(q_A, e).\delta_B(q_B, e) & \text{if } \delta_A(q_A, e) \text{ and } \delta_B(q_B, e) \text{ defined} \\ \delta_A(q_A, e).q_B & \text{if } \delta_A(q_A, e) \text{ defined and } e \notin \Sigma_B \\ q_A.\delta_B(q_B, e) & \text{if } e \notin \Sigma_A \text{ and } \delta_B(q_B, e) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Common events are synchronized.

Private events are not affected by the other automaton.

If $\Sigma_A = \Sigma_B$ the parallel composition reduces to a product.

If $\Sigma_A \cap \Sigma_B = \emptyset$ there are no synchronized transitions. This is called *concurrent* behavior or *shuffle* of $A$ and $B$

$A\|B = B\|A$ (state-names will be different) and $A\|(B\|C) = (A\|B)\|C$

# Projection

For the characterization of languages marked and generated by parallel compositions we need projection $P_i$

$$P_i : (\Sigma_A \cup \Sigma_B)^* \to \Sigma_i^* \text{ for } i = A, B$$

defined as follows

$$\begin{aligned} P_i(\varepsilon) &:= \varepsilon \\ P_i(e) &:= \begin{cases} e & \text{if } e \in \Sigma_i \\ \varepsilon & \text{if } e \notin \Sigma_i \end{cases} \\ P_i(se) &:= P_i(s)P_i(e) \text{ for } s \in (\Sigma_A \cup \Sigma_B)^*, e \in (\Sigma_A \cup \Sigma_B) \end{aligned}$$

$P_i$ removes events not in $\Sigma_i$. Compare to projections in $xy$-plane, when you remove either the $x$ or the $y$ coordinate.

# Inverse projection

$$P_i^{-1}(t) := \{s \in (\Sigma_A \cup \Sigma_B)^* : P_i(s) = t\}$$

Inverse projection of $t$ returns the set of strings that are projected on $t$.

Projections and their inverses are extended to languages by applying them to all the strings in the language.

Note that $P_i(P_i^{-1}(L)) = L$ but in general $L \subseteq P_i^{-1}(P_i(L))$

**Example 16.** Consider $\Sigma_A = \{a, b\}$ and $\Sigma_B = \{b, c\}$ and

$$L = \{c, ccb, abc, cacb, cabcbbca\}$$

Then

$$\begin{aligned}
P_A(L) &= \{\varepsilon, b, ab, abbba\} \\
P_B(L) &= \{c, ccb, bc, cbcbbc\} \\
P_A^{-1}(\varepsilon) &= \{c\}^* \\
P_A^{-1}(b) &= \{c\}^*\{b\}\{c\}^* \\
P_A^{-1}(ab) &= \{c\}^*\{a\}\{c\}^*\{b\}\{c\}^*
\end{aligned}$$

We can see that

$$P_A^{-1}(P_A(\{abc\})) = P_A^{-1}(\{ab\}) \supset \{abc\}$$

# Inverse projection using automata

If $S = \mathcal{L}_m(A) \subseteq \Sigma_A^* \subseteq \Sigma_B^*$ and $P_A$ is the projection from $\Sigma_B$ to $\Sigma_A$.

Then an automaton that marks $P_A^{-1}(S)$ is obtained by adding self-loops for all the events in $\Sigma_B \setminus \Sigma_A$ at all the states of $A$.

# Languages resulting from a parallel composition

1. $\mathcal{L}(A\|B) = P_A^{-1}(\mathcal{L}(A)) \cap P_B^{-1}(\mathcal{L}(B))$

2. $\mathcal{L}_m(A\|B) = P_A^{-1}(\mathcal{L}_m(A)) \cap P_B^{-1}(\mathcal{L}_m(B))$

You add self-loops for private events in one to the other.

And then take the product.

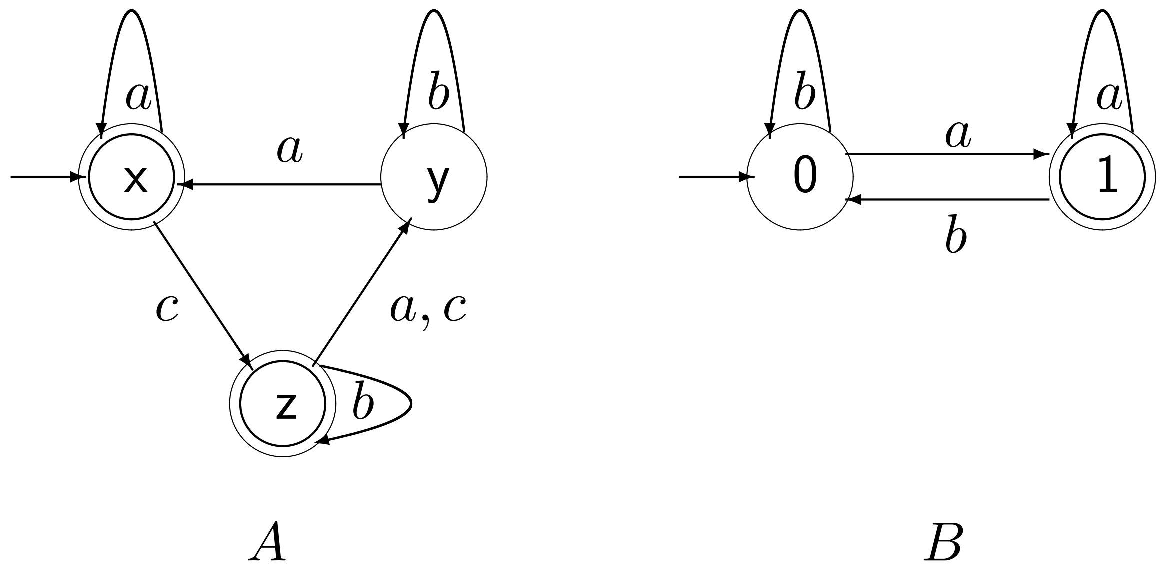
The self-loops will result in that the private events will not be affected by the other automaton.

The common events will be synchronized.

Parallel composition for languages is defined as:

$$L_1\|L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2)$$

**Example 17.** Consider the following two automata




Determine the parallel composition of $A$ and $B$

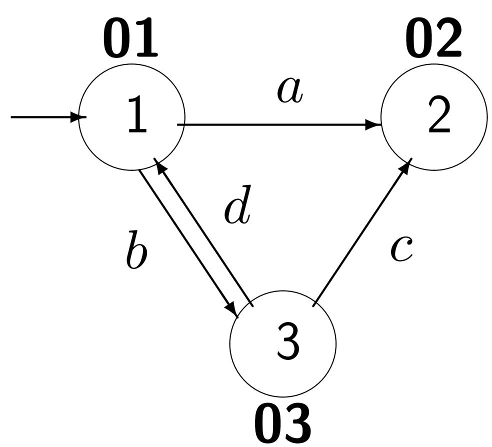
**Example 18.** Dining philosophers using Supremica. Tools ⇒ Test cases ⇒ Philos (2 is enough). In the new version of Supremica it is called Professors, pen and paper, found under Examples ⇒ Other Examples. Select all, and left-click ⇒ Synchronize will do a parallel composition. Select the new automaton and left-click ⇒ Synthesize to find the two deadlock states.

# Automata with Inputs and Outputs

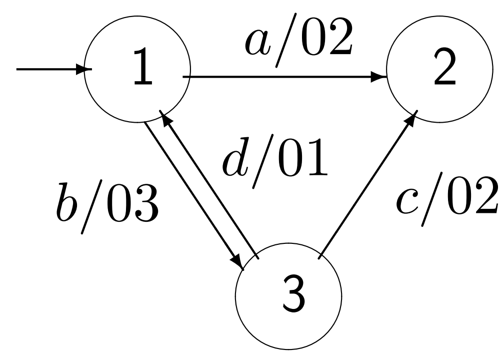
There are two variants to the definition of automaton given earlier, that explicitly takes into account inputs and/or outputs:

1. *Moore automata* with state outputs. Each state corresponds to a certain output, which is shown in bold above the state. Can be viewed as an extension of marking: Standard automata have two outputs, marked and unmarked.

2. *Mealy automata* are input/output automata. Transitions are labelled by events of the form *input event/output event*. Such events says which input can be handled at a certain state, and which output the automaton "emits" when it changes state.



Moore



Mealy

# Regular languages

**Definition** A language is said to be *regular* if it can be marked by a finite-state automaton. The class of regular languages is denoted $\mathcal{R}$

**Properties of $\mathcal{R}$:** Let $L_1$ and $L_2$ be in $\mathcal{R}$. Then the following are also in $\mathcal{R}$

1. $\overline{L_1}$, prefix-closure.
2. $L_1^*$, Kleene-closure.
3. $L_1^c := \Sigma^* \setminus L_1$, complement.
4. $L_1 \cup L_2$, union.
5. $L_1 L_2$, concatenation.
6. $L_1 \cap L_2$, intersection.

# Proof of properties of regular languages

The properties can be proven by constructing finite-state automata that marks the new languages.

It has been my intention to not introduce *non-deterministic* automata, for the proof we need a couple.

Allowing alternate transitions makes an automaton non-deterministic.

State changes by $\varepsilon$-transitions are transitions that take place without any event.

If there is one or several alternative transitions to a $\varepsilon$-transition from a state, the automaton becomes non-deterministic. $\varepsilon$ can take place before or after the alternative transitions, $e = \varepsilon e = e\varepsilon$

Let $A_1$ and $A_2$ be two automata that mark the languages $L_1$ and $L_2$ respectively.

1. $\overline{L_1}$. Take the trim on $A_1$ and mark all its states.

2. $L_1^*$. Mark the initial state. Then add $\varepsilon$-transitions from every marked state of $A_1$ to the initial state. The result is non-deterministic depending on if there are any other transitions going out from the marked states.

3. $L_1^c := \Sigma^* \setminus L_1$. This was proved when we considered the complement operation for automata. The automaton that marks $L_1^c$ has at most one more state than $A_1$.

4. $L_1 \cup L_2$. Create a new initial state and connect it, with two $\varepsilon$-transitions, to the initial states of $A_1$ and $A_2$. The result is a non-deterministic automaton that marks $L_1 \cup L_2$.

5. $L_1 L_2$. Connect the marked states of $A_1$ to the initial state state of $A_2$ by $\varepsilon$-transitions. Unmark all the states of $A_1$.

6. $L_1 \cap L_2$. We have earlier seen that $A_1 \times A_2$ marks $L_1 \cap L_2$

# Regular expressions

Regular expressions is a compact way of describing regular languages with possibly infinite number of strings.

- We have already defined concatenation, Kleene-closure, and union for languages.

- We adopt "$+$" instead of "$\cup$", logical OR

- We adopt $u^*$ instead of $\{u\}^*$, repetition

Recursive definition of regular expressions:

1. $\emptyset$ is a regular expression denoting the empty set, $\varepsilon$ is the regular expression denoting the set $\{\varepsilon\}$, $e$ is the regular expression denoting $\{e\}$, for all $e \in \Sigma$

2. If $r$ and $s$ are regular expressions, then $rs$, $(r+s)$ , $r^*$ and $s^*$ are regular expressions.

3. There are no regular expressions other than those constructed by applying the rules 1. and 2. above a finite number of times.

**Example 19.** Let $\Sigma = \{a, b, c\}$ be the set of events. The regular expression $(a + b)c^*$ denotes the language

$$L = \{a, b, ac, bc, acc, bcc, accc, bccc, \dots\}$$

The regular expression $(ab)^* + c$ denotes the language

$$L = \{\varepsilon, c, ab, abab, ababab, \dots\}$$

**Kleenes theorem:** Any language that can be denoted by a regular expression is a regular language, any regular language can be denoted by a regular expression