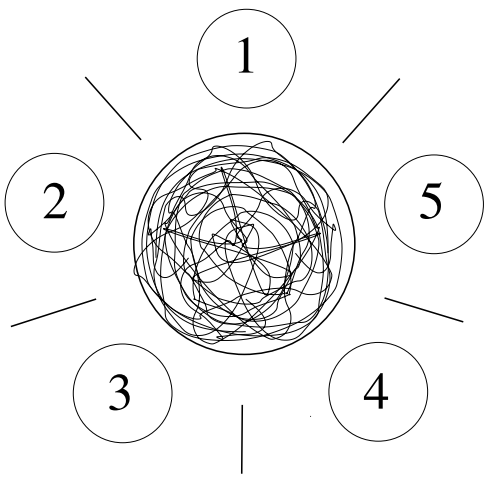
The dining philosophers Dijkstra'65 (en.wikipedia.org/wiki/Dining_philosophers_problem)

There are N philosophers sitting around a circular table either thinking or eating pasta. Each philosopher needs his left and right fork to eat, but there is only one fork between each 2 philosophers. Design an algorithm that the philosophers can follow.

Consider the following protocol (= sequence of interaction)




```
void philosopher()
    while(1) {
      think();
      get_left_fork();
      get_right_fork();
      eat();
      put_left_fork();
      put_right_fork();
    }
```

Problem?

Motivation (1)

- In the following we will develop a concise (mathematical) framework for formally describing systems of interest ($\longrightarrow$ formal model).
- This framework allows one to **formally, i. e., mathematically reason** about a model's and hence a system's correctness w. r. t. dedicated properties, e. g. deadlock-freeness etc.
- In principle we could start with any programming language. However, their interpretation is very complicated (address arithmetic, arbitrary data types, ...). Also only certain aspects of a system matter, where one may abstract away many details. Hence it appears useful to follow a more abstract view and speak here only about very simple "languages" for describing systems. Such methods are commonly denoted as **high-level model description methods**.

Motivation (2)

- Even though the high-level model description methods appear very simple, they possess a clearly defined (execution) semantics. This semantics allows us to map them to graphs.
- These graphs represent all possible behaviors of the specified high-level description.
- Hence the basic objects which represent the entities to be studied are graphs. Therefore we will briefly re-visit some basic definitions, which you probably have already seen before.
- Don't mind the formal notation, this will be made clear by examples and allows you to understand the resp. literature.

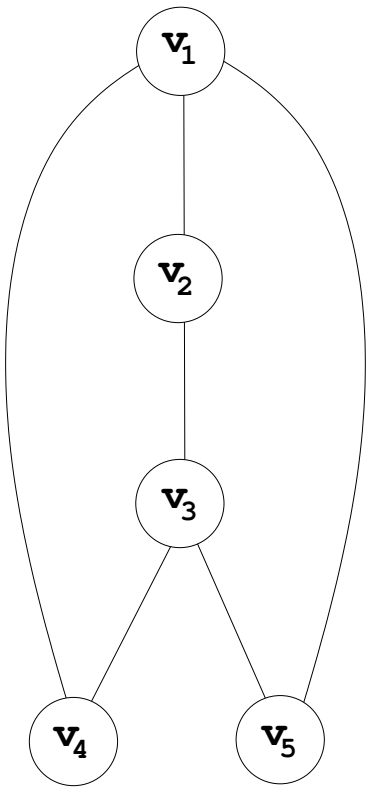
Agenda

1. Graph Theory: Some Definitions
2. Introduction to Petri Nets
3. Introduction to Computation Tree Logic and related model checking techniques
4. Introduction to Binary Decision Diagrams

Part I

Graph Theory: Some Definitions

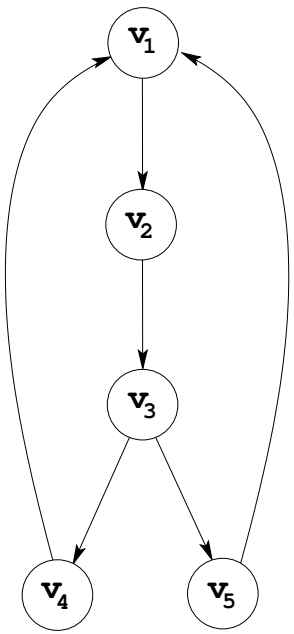
Preliminaries – Foundations of Graphs (at a glance)



Definition 1.1: Graph

A graph **G** is a pair $(\mathbb{V}, \mathbb{E})$ where

1. $\mathbb{V}$ is a discrete set of vertices (or nodes) $\{v_1, \ldots, v_m\}$
2. $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ is a discrete set of pairs $\{(v_i, v_j) \mid \text{for some } i, j \in \{1, \ldots, m\}\}$. One commonly denotes these pairs as edges or arcs.



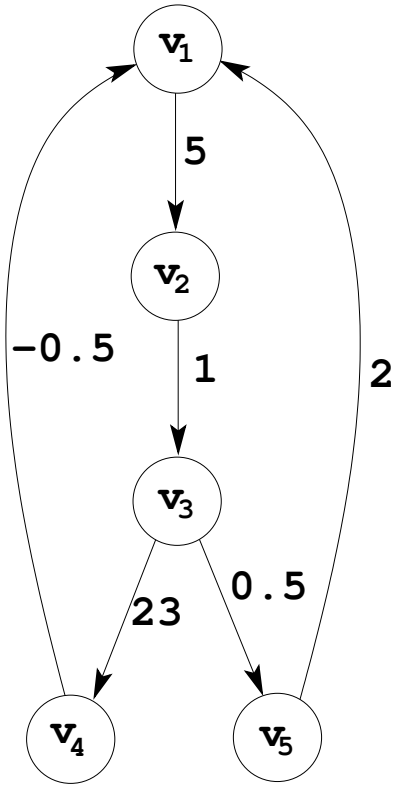
Definition 1.2: Directed Graph (Digraph)

- If the elements of $\mathbb{E}$ are ordered in such a way that $(v, w) \not\equiv (w, v)$ the elements of $\mathbb{E}$ are denoted as directed edges or directed arcs.
- A graph with such a property is denoted directed graph or *digraph* for short.

Preliminaries – Foundations of Graphs (at a glance)

1. In a digraph and for an ordered pair $(u, v) \in \mathbb{E}$ vertex u is denoted as **predecessor or parent** of vertex v and vertex v is denoted as **successor or child** of vertex u.
2. This can be extended to the level of sets of children and parent nodes as follows:
   - $\underline{\mathcal{P}ost(u) := \{v \in \mathbb{V} \,|\, (u, v) \in \mathbb{E}\}}$
     is the set of all **children or direct successors** of a vertex u.
   - $\underline{\mathcal{P}re(v) := \{u \in \mathbb{V} \,|\, (u, v) \in \mathbb{E}\}}$
     is the set of all **parents or direct predecessors** of a vertex v.
3. The above sets are very helpful, once we want to operate (traverse) on graphs.

Preliminaries – Foundations of Graphs (at a glance)



- $\mathcal{W}(v_3, v_5) =$
- $\mathcal{W}(v_3, v_1) =$

Definition 1.3: Weighted Graphs

- If the edges of a graph are labelled with elements from $\mathbb{R}$ one speaks of a *weighted graph*.
- In fact the set of edges of a weighted digraph is a ternary relation (set of triples): $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{R} \times \mathbb{V}$.
- Function $\mathcal{W} : \mathbb{V} \times \mathbb{V} \to \mathbb{R}$ gives one the weight associated with the respective edge (u, x, v): $(u, x, v) \in \mathbb{E} \Rightarrow \mathcal{W}(u, v) = x$ and $\mathcal{W}(u, v) := 0$ for all triples not contained in $\mathbb{E}$.

Preliminaries – Incidence/transition matrices

1. **Incidence matrix**: Each finite graph **G** consisting of n vertices can be mapped to a matrix $T \in \{0,1\}^{n \times n}$ as follows:

$$a_{ij} := \begin{cases} 1 & \Leftrightarrow (v_i, v_j) \in \mathbb{E} \\ 0 & \text{else} \end{cases}$$

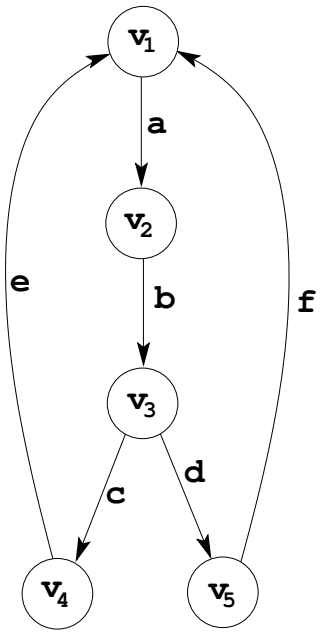
   Such a matrix is denoted as *incidence* matrix. Note, that this requires a consistent numbering of the vertices.

2. In case $\exists a_{ij} \notin \{0,1\}$ one speaks of a **weighted incidence** or **transition** matrix A of **G**.

Note: *Sometimes the incidence/transition matrix is defined as the transposed* A^T.

⟶ Example 1.1: A of the weighted graph shown before

Preliminaries – Foundations of Graphs (at a glance)



Definition 1.4: Labelled Graphs

- If the edges of a graph are labelled with elements from a finite set, e. g. $l \in \mathcal{A}ct$ one speaks commonly of a *labelled graph*.
- In fact the set of edges of a labelled digraph is once again a ternary relation: $\mathbb{E} \subseteq \mathbb{V} \times \mathcal{A}ct \times \mathbb{V}$.

Note:

A labelled graph is also often refered to as labelled transition system (LTS), where instead of vertices one speaks of states.

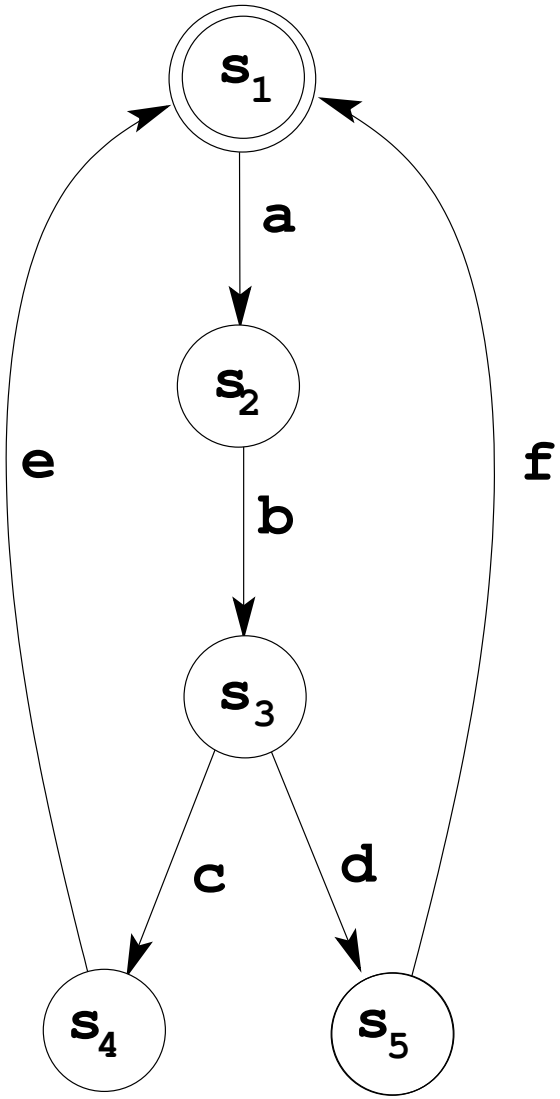
Preliminaries – Labelled Transition System

Definition 1.5: Labelled Transition System (LTS)

A LTS is a quadruple $\mathcal{T} := (\mathbb{S}, \mathbb{S}_0, \mathcal{A}ct, \mathbb{E})$, where

1. $\mathbb{S} := \{\vec{s}_1, \ldots, \vec{s}_n\}$ is an ordered (indexed) set of states with
2. $\mathbb{S}_0$ as the set of initial states.
3. $\mathcal{A}ct$ is the discrete set of transition labels,
4. $\mathbb{E} \subseteq \mathbb{S} \times \mathcal{A}ct \times \mathbb{S}$ is an ordered (indexed) set of labelled state-to-state transitions.

Preliminaries – Labelled Transition system



1. $\mathbb{S} := \{$
2. $\mathbb{S}_0 :=$
3. $\mathcal{A}ct := \{$
4. $\mathbb{E} := \{$

Preliminaries – Termination and Determinism of LTS

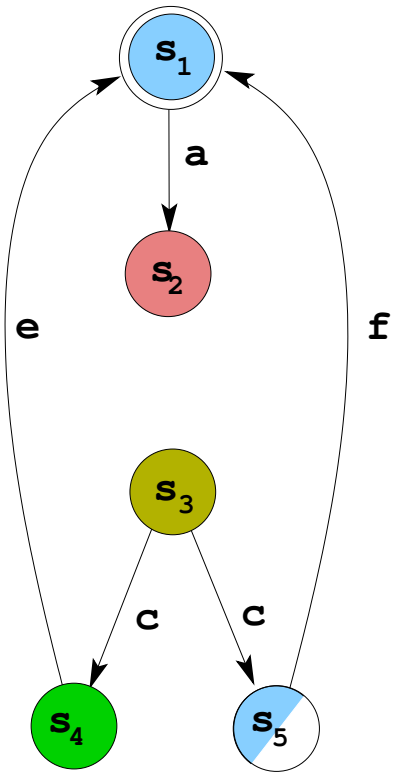
Note:

LTS are essentially the semantics of the here discussed high-level modelling techniques, where the techniques of model checking allow us to reason about their properties. In the following we briefly strive some important definitions.

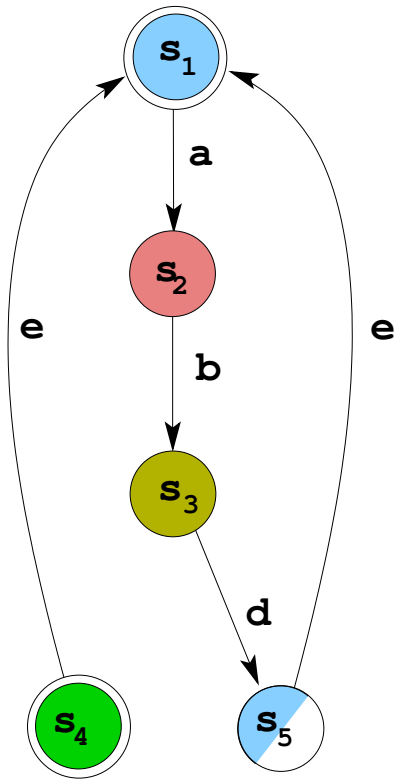
- A LTS $\mathcal{T}$ is defined *non-terminal* if each state has at least one out-going edge otherwise $\mathcal{T}$ is called *terminal*.
- A LTS $\mathcal{T}$ is defined deterministic if each state has at most one out-going edge with the same edge label otherwise $\mathcal{T}$ is denoted as *non-deterministic*.

Are non-deterministic finite LTS more expressive than deterministic ones?

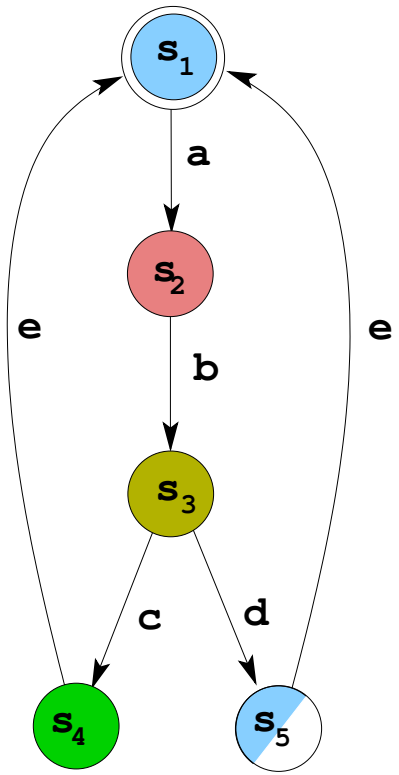
Preliminaries – Termination and Determinism (Examples)



non-terminal,deterministic?



non-terminal,deterministic?



non-terminal,deterministic?

Agenda

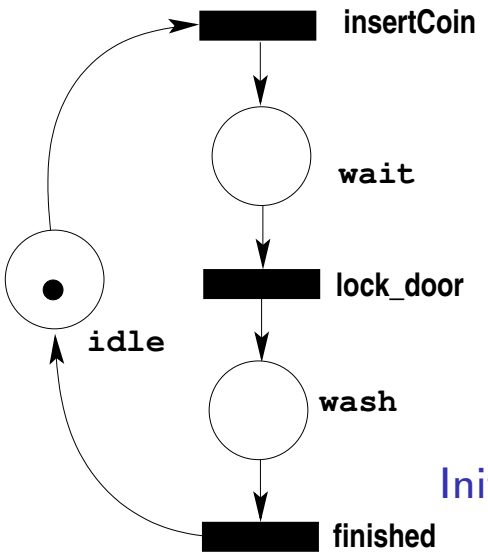
1. Graph Theory: Some Definitions
2. Introduction to Petri Nets ◁
3. Introduction to Computation Tree Logic and related model checking techniques
4. Introduction to Binary Decision Diagrams

Part II

Petri Nets

Petri Nets – Introduction

A Petri Net (PN) is a weighted(?), bipartite(?) digraph(?) invented by Carl Adam Petri in his PhD-thesis "Kommunikation mit Automaten" (1962). Many flavors of Petri nets are in use, where we start with the simplest kind (almost).
Example:



Circles $P := \{\text{wait}, \text{wash}, \ldots\}$ (set of places)

Boxes $T := \{\text{insertCoin}, \text{lock\_door}, \ldots\}$ (set of transitions)

Arcs $C := \{\ldots, (\text{lock\_door}, \text{wash}), \ldots\}$ (set of edges)

Weights $\mathcal{W} :=$ here constant 1 for each arc

Initial marking $\mathcal{M}_0 := \{\ m_0(\mathit{wait}) := 0,$
$m_0(\mathit{wash}) := 0, m_0(\mathit{idle}) := 1\}.$

Definition 2.1: (Weigthed) Petri Net

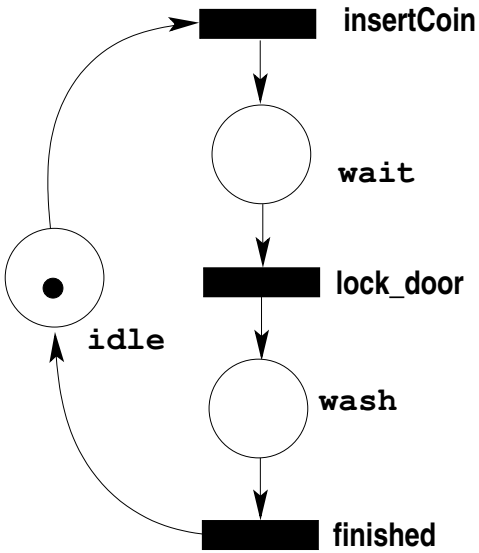
A Petri net (PN) **P** is a 5-tuple $(P, T, C, \mathcal{W}, \mathcal{M}_0)$, where

1. $P := \{p_1, \ldots, p_m\}$ is a finite, ordered (indexed) set of places and
2. $T := \{t_1, \ldots, t_n\}$ is a finite, ordered (indexed) set of transitions,
3. $C \subseteq (P \times T) \cup (T \times P)$, is a connection or flow relation,
4. $\mathcal{W} : C \mapsto \mathbb{N}_0$ assigns a weight to each element of C and
5. $m_0 : p \in P \mapsto \mathbb{N}_0$ gives the **initial** marking for place p, i. e., it assigns a number of token to place p. The set of all such initial markings is denoted $\mathcal{M}_0$ (= the initial marking of **P**).

---

1. An ordering defined on the places yields that the place markings $m_0(p_i)$ can be understood as the component of a vector $\vec{s}_0[i]$ s.t. we can map the initial marking $\mathcal{M}_0$ to the dedicated vector $\vec{s}_0$.
2. A vector of this kind is denoted in the following as state vector or simply as state.

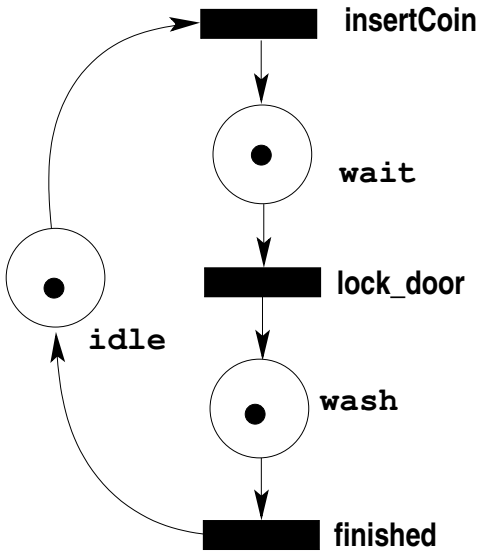
Petri Nets – Introduction



the initial state of the PN

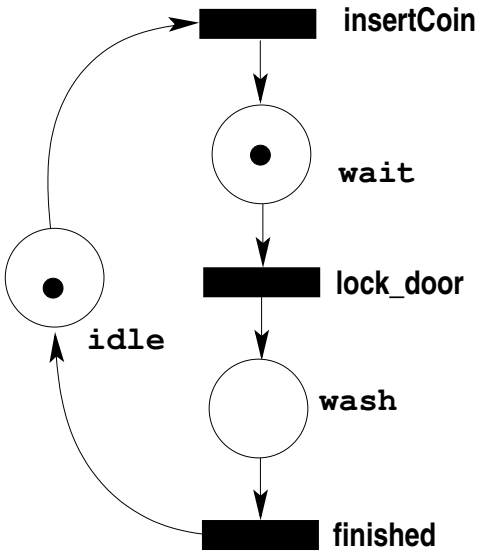
$\vec{s}_0 = (m_0(wait), m_0(wash), m_0(idle))$

$= (0, 0, 1)$



another state of the PN

$\vec{s} = (1, 1, 1)$



and another state

$\vec{s} = (1, 1, 0)$

Note:

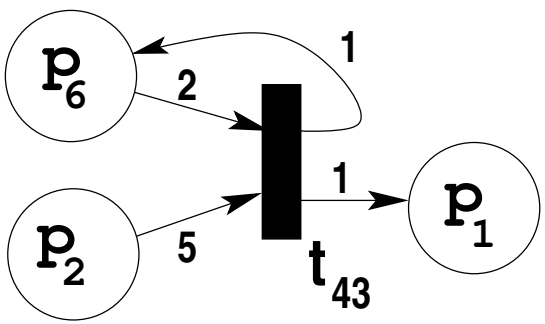
For differing among the different states of a PN we index them accordingly, i. e., we write $\vec{s}_k$ when referring to the k'th state.

Petri Nets – Operational Semantics (Pre - and Post sets)

Definition 2.2: Pre - and Post sets

1. Pre-set of a transition $t \in T$: $\bullet t := \{p | (\mathbf{p}, \mathbf{t}) \in C\}$
2. Post-set of a transition $t \in T$: $t\bullet = \{p | (\mathbf{t}, \mathbf{p}) \in C\}$

analogously we define pre ($\bullet p$) and post sets ($p\bullet$) for each place $p \in P$.



- $\bullet t_{43} := \{$
- $t_{43}\bullet := \{$

Petri Nets – Operational Semantics (Enabling)

The operational (or execution) semantics of PN stem from the movement of tokens in the net:

- Transition consume tokens from input places (pre set) and
- transition add tokens to their output places (post set),
- where we execute the transitions one by another (=interleaving semantics).

But when can we actually execute transitions ?

Petri Nets – Operational Semantics (Enabling)

- **Enabled transition:**

  a transition $t \in T$ is enabled in a state $\vec{s}$, denoted $\vec{s} \rhd t$, *iff* the places of its pre-set hold sufficiently many tokens

  $$\vec{s} \rhd t \Leftrightarrow (\forall p \in \bullet t : \vec{s}\,[p] \geq \mathcal{W}(p, t))$$

- **Enabling state:**

  a state $\vec{s}$ is denoted enabling for a transition t *iff* $\vec{s} \rhd t$ holds.

- Once a transition is enabled in a state $\vec{s}$ it can be executed (=fired):

> When we execute an enabled transition we destroy sufficiently many tokens on the input places and generate the required number of tokens on the output places of t.

Petri Nets – Operational Semantics (Firing)

Given a state $\vec{s}$ of a PN we want to compute its successor state $\vec{s}\,'$ w. r. t. an enabled transition. To do so we define a transfer or transition function of a transition t as follows:

$$\delta(\vec{s}, t) := \vec{s}\,' \text{ with } \vec{s}\,'\ [p] := \begin{cases} \vec{s}\,[p] & \Leftrightarrow p \notin t\bullet \cup \bullet t \\ \vec{s}\,[p] - \mathcal{W}(p,t) & \Leftrightarrow p \in \bullet t \cap \overline{t\bullet} \\ \vec{s}\,[p] + \mathcal{W}(t,p) & \Leftrightarrow p \in t\bullet \cap \overline{\bullet t} \\ \vec{s}\,[p] - \mathcal{W}(p,t) + \mathcal{W}(t,p) & \Leftrightarrow p \in t\bullet \cap \bullet t \end{cases}$$

- if $\vec{s} \rhd t$ then $\vec{s}\,' := \delta(\vec{s}, t)$,
- This gives a set of triples $(\vec{s}, t, \vec{s}\,')$ where we also use the notation $\vec{s} \xrightarrow{t} \vec{s}\,'$.

Petri Nets – Reachability set

- **Token game**: Moving the tokens around the net by executing enabled transitions.
- Sequential execution of transition functions of enabled transitions allows one to construct a set of reachable states, denoted as reachability set of a PN **P** w. r. t. its initial state $\vec{s}_0$.

Definition 2.3: Reachability set

---

1. $\mathbb{S}_0 := \{\vec{s}_0\}$
2. $\mathbb{S}_i := \mathbb{S}_{i-1} \cup \{\delta(\vec{s}, t) \mid \forall \vec{s} \in \mathbb{S}_{i-1}, \forall t \in T \text{ where } \vec{s} \rhd t\}$
3. we are intrested in the largest of such sets $\mathbb{S}_0 \subseteq \mathbb{S}_1 \subseteq \ldots \subseteq \mathbb{S}$ which we denote as set of reachable states $\mathbb{S}$ of a PN **P** and w. r. t. $\vec{s}_0$.

Note:

This allows one to construct a (not necessarily finite) LTS for each PN and an initial state $\vec{s}_0$. Such an LTS constitutes the semantic model of a PN.

Definition 2.4: Reachability graph

A reachability graph $RG(\mathbf{P}, \vec{s}_0)$ of a PN $\mathbf{P}$ and its initial state $\vec{s}_0$ is a LTS $\mathcal{L}(\mathbb{S}, \mathbb{S}_0, \mathcal{A}ct, \mathbb{E})$ with

- $\mathbb{S}$ which is the set of reachable states of the PN.
- $\mathbb{S}_0 := \{\vec{s}_0\}$ with $\vec{s}_0$ as the initial state of the PN
- $\mathcal{A}ct$ which is the set of the transition labels of the PN
- $\mathbb{E} \subseteq \mathbb{S} \times \mathcal{A}ct \times \mathbb{S}$ induced by the PN as follows:

$$(\vec{s} \in \mathbb{S} \wedge \vec{s} \rhd t) \Rightarrow \left(\vec{s} \xrightarrow{t} \delta(\vec{s}, t) \in \mathbb{E}\right)$$

Petri Nets – Token game (Reachability of states)

One the basis of the token game we can now pose interesting questions about the properties of a PN and ultimately about the modelled system itself.E.g.:

- Can we reach a state s.t. each place holds at least N but at most K tokens (under- or overflow of Buffers in a chip-design)?
- Can we reach a state where everything is blocked?

Such questions are denoted reachability problems, since they can be solved by checking if a respective state can be reached strating from the initial state $\vec{s}_0$.

⟶ Example 2.1: Washing maschine

Petri Nets – Reachability question

We formalize this as follows:

Definition 2.5: Reachability problem/question

Given a PN **P** and its initial state $\vec{s}_0$, is it possible to reach a dedicated state $\vec{s}_b$ by executing a transition sequence $\sigma := t_i, \ldots, t_j$ with $t_i, t_j \in T$ starting from the initial state $\vec{s}_0$. I.e. formally we are looking for a sequence $\sigma := t_i, t_j \ldots t_k$ s.t. $\vec{s}_0 \xrightarrow{t_i} \vec{s}_a \xrightarrow{t_j} \ldots \vec{s}_c \xrightarrow{t_k} \vec{s}_b$ where one also writes $\vec{s}_0 \xrightarrow{\sigma} \vec{s}_b$ for short.

How would you proceed?

Petri Nets – How-to solve the reachability problem

There are several ways to answer this question, we discuss two necessary (not sufficient!) methods for deciding the reachability of a state.

1. **Algebraic solution** based on state equations: Solution of a system of linear equation is a necessary condition for reachability for standard PN only.

   Absence of solution implies non-reachability of the resp. state.

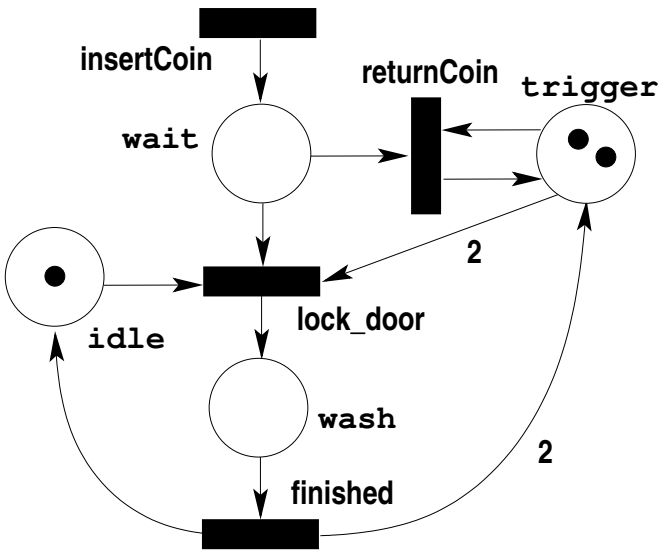
2. **Algorithmic solution** (brute-force method): Generation of the reachability graph and check the states on-the-fly.

   Termination implies reachability of the searched state.

Petri Nets – Algebraic approach (state equations)

The incidence matrix $A \in \mathbb{N}_0^{|P| \times |T|}$ describes the token-flow w. r. t. place i (row index) and transition j (column index).

$$a_{ij} = \mathcal{W}(t_j, p_i) - \mathcal{W}(p_i, t_j) \text{ (gain of place } i \text{ when transition } j \text{ fires)}$$



$$A := ?$$

$$\vec{s}_0 := ?$$

⟶ Example 2.2: Washing machine

Petri Nets – Algebraic approach (state equations)

- Let the firing vector $\vec{u}_i \in \mathbb{N}_0^{|T|}$ be the indicator for the firing of transition t_i, i. e., $\vec{u}_i[i] := 1$ and $\vec{u}_i[j] := 0$ for $i \neq j$.

  What is the PN semantics of $\vec{s}_c + A \cdot \vec{u}_i$?

  ⟶ Example 2.3: Washing machine

- For a transition sequence σ of length $|\sigma|$ one can construct the linear combination of the firing vectors:

$$\vec{f} := \sum_{k:=1}^{|\sigma|} \vec{u}_{\sigma[k]}$$

  where $\sigma[k]$ refers to the k'th transition symbol in the sequence.

  ⟶ Question 2.1: What is the PN semantics of $\vec{s}_c + A \cdot \vec{f}$?

Petri Nets – Algebraic approach (state equations)

This allows one to test for the reachability of state $\vec{s}$ as follows:

1. $\texttt{Solve}(A \cdot \vec{f} = \vec{s} - \vec{s}_0)$
2. if there is no such solution $\vec{f} \in \mathbb{N}_0^{|T|}$ than there is no sequence of transition firings leading from $\vec{s}_0$ to $\vec{s}$.
3. Formally:

$$(\nexists \vec{f} \in \mathbb{N}_0^{|T|} : A \cdot \vec{f} = \vec{s} - \vec{s}_0) \Rightarrow \nexists \sigma : \vec{s}_0 \xrightarrow{\sigma} \vec{s}$$

⟶ Example 2.4: Washing machine

⟶ Question 2.2: If there is a solution, what do we know, what can be the problem?

Petri Nets – Algorithmic approach (enumerative)

1. **Algebraic solution** based on state equations: Solution of a system of linear equation is a necessary condition for reachability for standard PN only.

   Absence of solution implies non-reachability of the resp. state.

2. **Algorithmic solution** (brute-force method): Generation of the reachability graph and check the states on-the-fly:

   Termination implies reachability of the searched state.

Petri Nets – Explicit Reachability Set Generation

Reachability algorithm

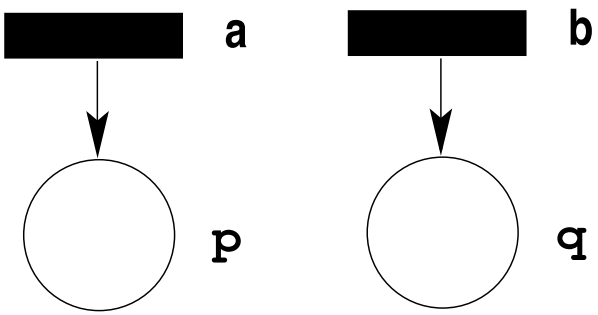
(1) $\mathbb{S}_{tmp} := \emptyset$, $\mathbb{S} := \emptyset$
(2) put $\vec{s}_0$ into $\mathbb{S}_{tmp}$
(3) put $\vec{s}_0$ into $\mathbb{S}$
(4) Call function DFS()

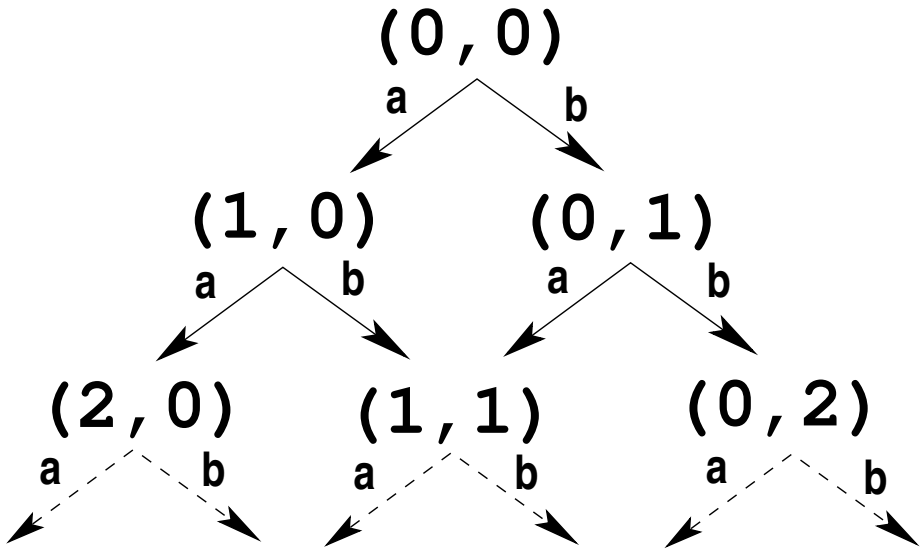
(5) Function DFS()
(6) While ($\mathbb{S}_{tmp} \neq \emptyset$)
(7) take $\vec{s}$ from $\mathbb{S}_{tmp}$
(8) forall $t \in T$ do
(9) $\vec{s}' := \delta(\vec{s}, t)$
(10) If ($\vec{s}' = \vec{s}_{test}$) terminate
(11) Else if ($\vec{s}' \notin \mathbb{S}$)
(12) put $\vec{s}'$ into $\mathbb{S}_{tmp}$
(13) put $\vec{s}'$ into $\mathbb{S}$
(14) Else do_nothing
(15) endif
(16) od
(17) endwhile

Once this algorithm terminates we know that $\vec{s}_{test}$ is reachable or not reachable.

⟶ Example 2.5: Washing machine: DFS/BFS

Petri Nets – Interleaving semantics





- The execution order of transitions is partly uncoordinated.
- This yields an exponential size of a PN's underlying LTS , i. e., $|\mathbb{S}|$ is exponential w. r. t. the number of concurrently enabled transitions

(= state space explosion problem)

⟶ Example 2.6: Washing machine with 2 tokens in place idle

Petri Nets – Properties to be verified

- After we have learned how to answer the reachability problem, (where the proposed techniques may fail), we will now deal with other properties to be verified.
- These properties can be posed as reachability problems. Hence they can be answered by either directly applying the discussed techniques or slightly adapted versions.

Excursion:

Properties that can be answered as reachability queries, are denoted safety properties. But remember, the here presented techniques may fail, but sometimes we can not do better, depending on the employed high-level formalism, e. g. PN with inhibitor arcs.

Deadlock-freeness

One of the most basic properties related to the reachability is the question about the reachability of deadlocks.

Definition 2.6: Deadlock

A state $\vec{s} \in \mathbb{S}$ of a PN **P** is denoted deadlock *iff*

$$\nexists t \in T : \vec{s} \rhd t$$

- A PN **P** where no deadlock exists is denoted deadlock-free.
- A PN **P** whose reachability graph is non-terminal is deadlock-free (or vice-versa)

⟶ Example 2.7: Dining philisophers

K-boundedness

Definition 2.7: K-Boundedness

1. A place p_j of a PN **P** with its initial state $\vec{s}_0$ is denoted K-bounded *iff* it never holds more than K_j tokens:

$$p_j \text{ is } K_j\text{-bounded} \Leftrightarrow \exists K_j \in \mathbb{N}_0 : \forall \vec{s} \in \mathbb{S} : \vec{s}_{p_j} \leq K_j$$

   otherwise p_j is denoted as *unbounded*.

2. A PN is denoted K-bounded if each of its places is K_j-bounded, i. e.,

$$\mathbf{P} \text{ is denoted } K\text{-bounded} \Leftrightarrow \forall p_j \in P : p_j \text{ is } K_j\text{-bounded}$$

- Example: Check a system design for buffer-overflows.
- In the literature 1-bounded PN are denoted as *safe*.

⟶ Question 2.3: Is the dining philisopher PN bounded?

Reversibility and home states

Besides reachability problems, where are properties which ask about the structure of the reachability graph.

Definition 2.8: Home state

---

1. A state $\vec{s}'$ is denoted as home state *iff* it is reachable from every other state, i. e.,

$$\vec{s}' \text{ is a home state } \Leftrightarrow \forall \vec{s} \in \mathbb{S} : \exists \sigma : \vec{s} \xrightarrow{\sigma} \vec{s}'.$$

2. If $\vec{s}_0$ is a home state than the PN **P** is denoted reversible.

In fact this is much more complex as a simple reachability query.

1. Algebraic approach: can we exploit this one for home state detection?
2. Algorithmic approach: is much more complicated (cycle detetction).

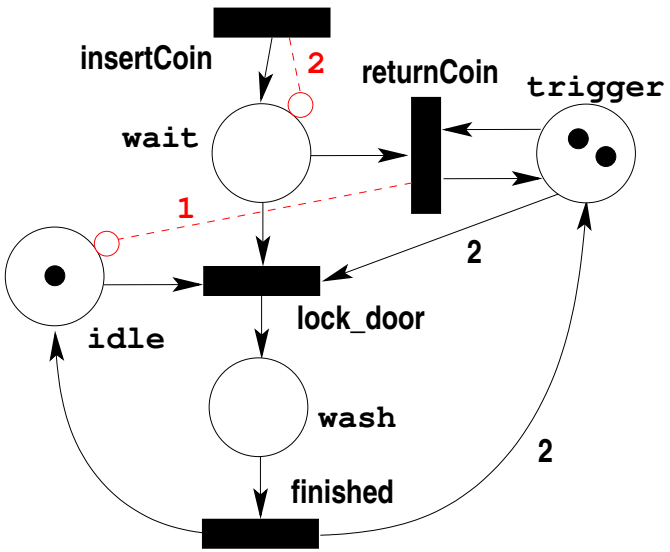
Petri Nets – Extensions

Many flavors of Petri nets are in use, e.g.

1. PN with inhibitor arcs
2. Colored PN
3. PN with timed behaviour

Petri Nets – Extensions

PN with inhibitor arcs



- A place p connected to a transition t via an inhibitor arc ($\Rightarrow p \in \triangle t$) suppresses the transition's execution, i. e., t can only fire *iff* $\vec{s}\,[p] < \mathcal{W}(p,t)$ holds.
- one solely needs to extend the rule for enabledness of t:

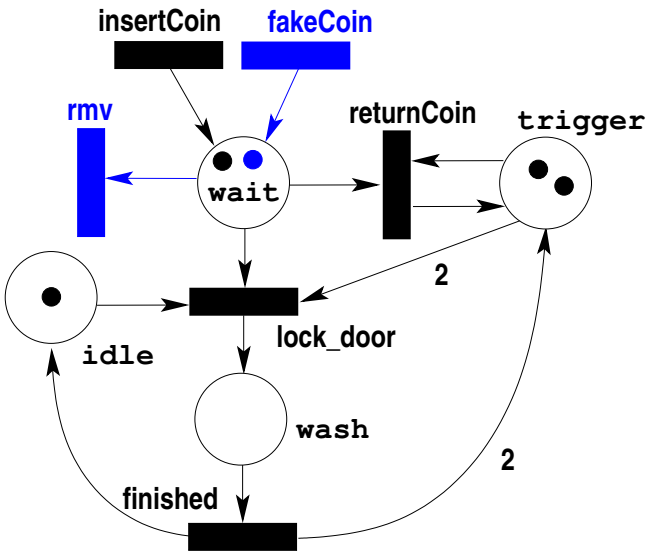
$$\vec{s} \rhd t \Leftrightarrow \{(\forall p \in \bullet t : \vec{s}\,[p] \geq \mathcal{W}(p,t)) \\ \wedge : (\forall p \in \triangle t : \vec{s}\,[p] < \mathcal{W}(p,t))\}$$

Note:

PN with more than one inhibitor arc posseses Turing-power, i. e.,

1. *they cannot be transformed into regular weighted PN*
2. *reachability of a state is not decidable; the methods we looked at are therefore the best one can do.*

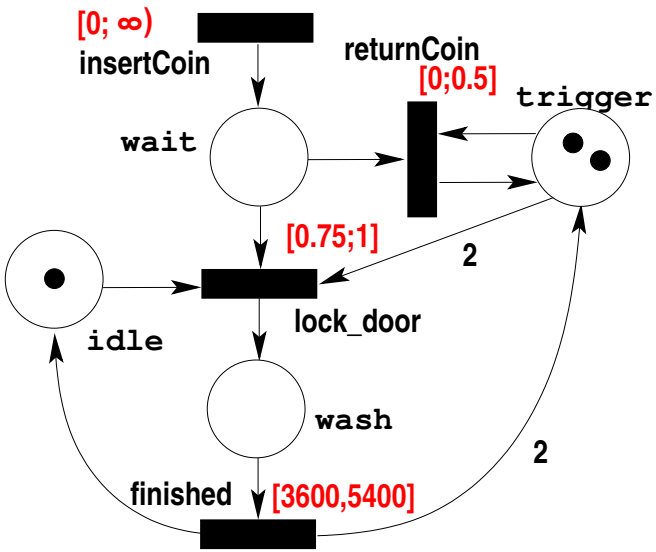
Colored PN



- Tokens carry colors
- Transition are colored, i. e., they only consume and generate tokens of a specific color

- Colored PN posses Turing-power:
  1. cannot be transformed into regular PN
  2. reachability is not decidable.

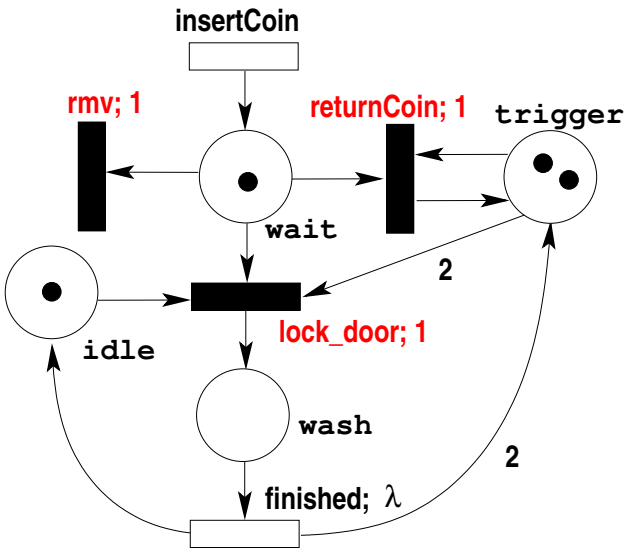
Timed PN (TPN)



Enabled transitions are executed within some time interval $[a, b]$

($\rightarrow$ Timed Automata)

Generalized Stochastic PN (GSPN)



In a GSPN we have 2 types of transitions

- Weigthed transition w is executed with some probability:

$$Prob_w(\vec{s}) := \frac{\mathcal{W}(w)}{\sum_{t_k \in \{t \mid \vec{s} \triangleright t\}} \mathcal{W}(t_k)}$$

- Markovian transition m is executed after an exponentially distributed delay time t

$$F_{delay\_of\_m}(t) := 1 - e^{-\lambda_m t}$$

($\rightarrow$ Continous-time Markov chains)