Chapter 3
SPECIFICATION MODELS

Distributed Computing Group

Discrete Event Systems
Winter 2005 / 2006

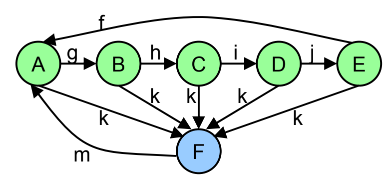
Overview

- **StateCharts**
  - Hierarchy
  - Concurrency
  - Events and Actions
  - Simulation Semantics
  - Non-Determinism and Conflicts
- **Petri Nets**
  - Notation
  - Concurrency
  - Petri Net Languages
  - Behavioral Properties
  - Analysis

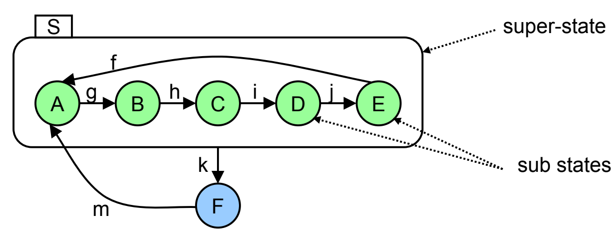
StateCharts

- **Deficits of finite automata for modeling:**
  - Only one sequential process, **no** concurrency
  - No hierarchical structuring capabilities
- **Extension StateCharts:**
  - Model of David Harel [1987]
  - StateCharts introduces hierarchy, concurrency and computation
  - Model is used in many tools for the specification, analysis and simulation of discrete event systems, e.g. Matlab-Stateflow, UML, Rhapsody, Magnum
  - Complicated semantics: We will only cover some basic mechanisms.

Introducing Hierarchy

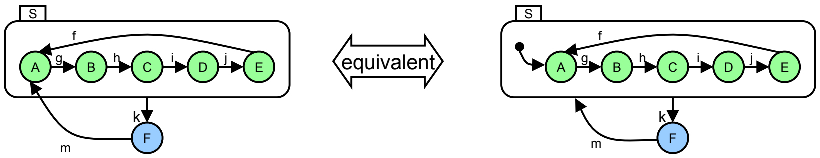


FSM is in *exactly one* of the sub states of S if S is active
(either in A xor B xor …)



Definitions

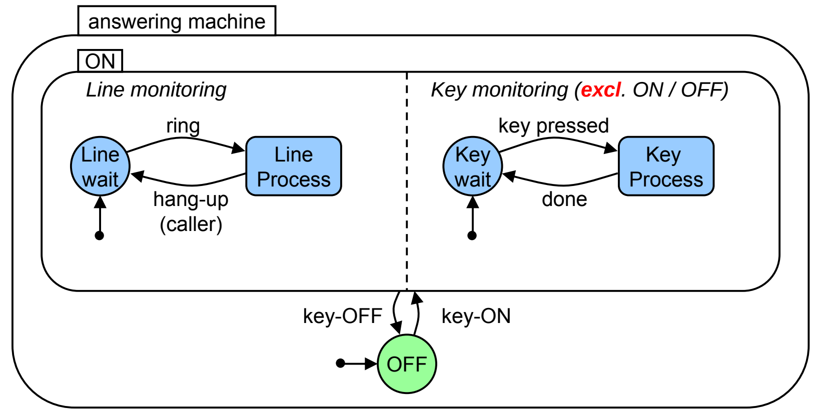
A super-state *S* is called OR-super-state, if *exactly one* of its sub states is active when *S* is active.



- Current states of FSMs are called active states
- States which are not composed of other states are called basic
- States containing other states are called super states
- For each basic state *s*, the super-states containing *s* are called ancestor states

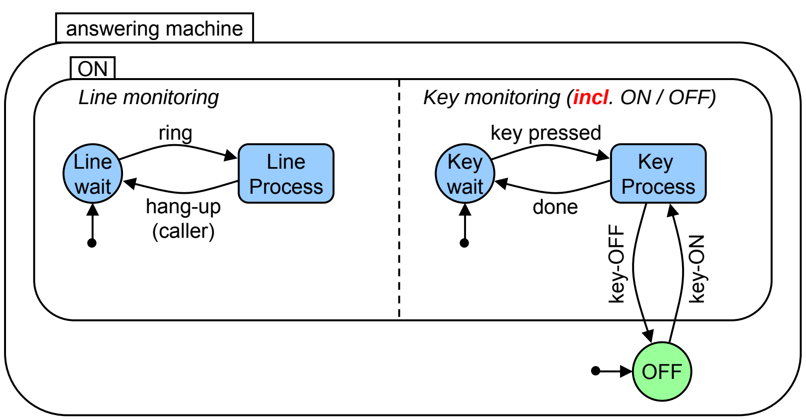
Introducing Concurrency

A super-state *S* is called AND-super-state, if *all* (immediate) sub-states are active when *S* is active.



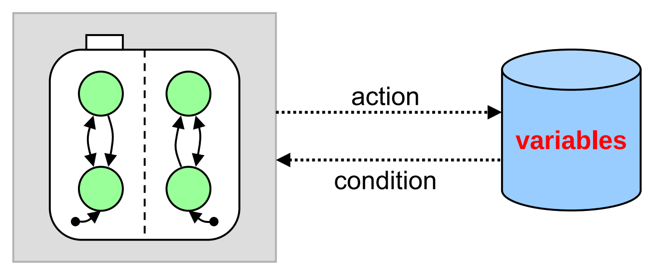
Entering and leaving AND-Super-States

New: **on** / **off** events handled by key process.

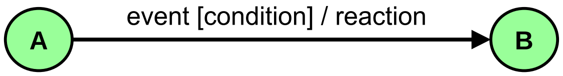


Representation of Computations

- Besides states, arbitrary many other variables can be defined. This way, not all states of the system are modeled explicitly.
- The variables can be changed as a result of a state transaction ("action"). State transitions can be dependent on these variables ("conditions").



General form of edge labels



Event

Can be either internally or externally generated.

Condition

Refer to values of variables that keep their value until they are reassigned.

State transition

Transition is enabled if event exists *and* condition holds

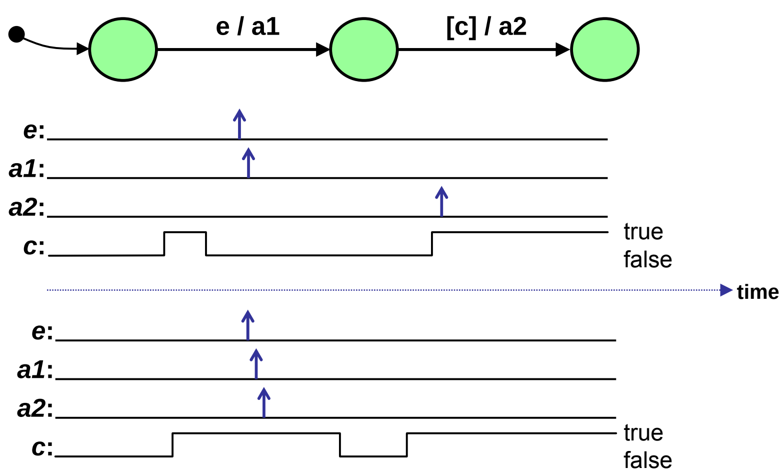
Reaction / action

Can be assignment to variables and/or creation of events

Events and Actions

- An event can be composed of several events:
  - **(*e1* and *e2*)** event that corresponds to the simultaneous occurrence of e1 and e2.
  - **(*e1* or *e2*)** event that corresponds to the occurrence of either e1 or e2 or both.
  - **(not *e*)** event that corresponds to the absence of event e.
- Similarly for conditions
- A reaction can also be composed:
  - **(*a1*; *a2*)** actions a1 und a2 are executed sequentially.
- All events, states and actions are globally visible.

Example

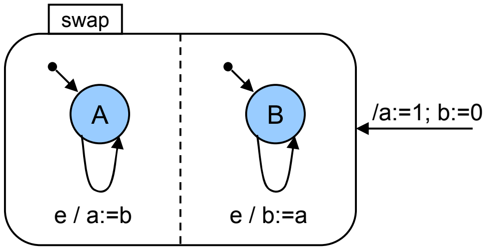


The StateCharts Simulation Phases

- The transitions are evaluated in simulation steps.
- Each step is divided in three phases:

1. Effect of changes on events and conditions is evaluated
2. The set of transitions to be made in the current step and right hand sides of assignments are computed
3. Transitions become effective, variables obtain new values.

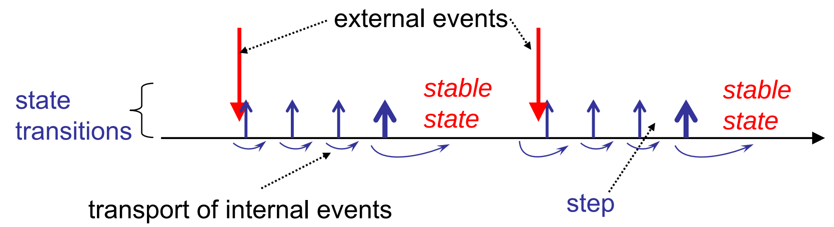
Example – Swap



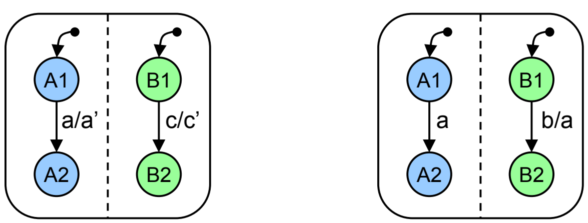
- In phase 2, variables *a* and *b* are assigned to temporary variables
- In phase 3, these are assigned to *b* and *a*, respectively
- As a result, variables *a* and *b* are swapped

More on semantics of StateCharts

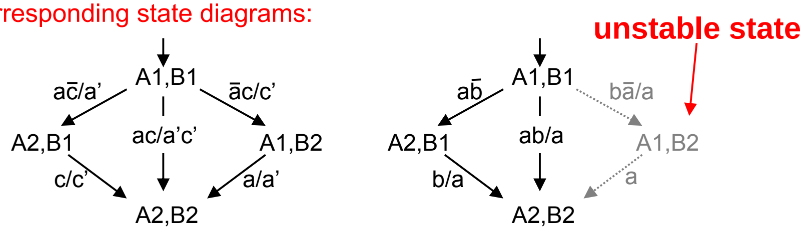
- Unfortunately, there are several time-semantics of StateCharts in use. This is one possibility:
  - A step is executed in arbitrarily small time.
  - Internal (generated) events exist only within the next step.
  - External events can only be detected after a stable state has been reached.



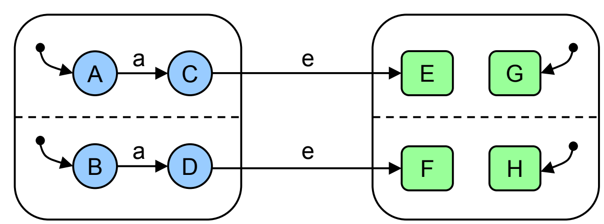
Example, State Diagram



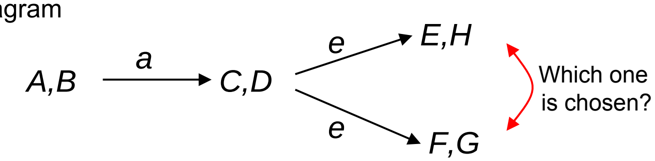
Corresponding state diagrams:



Example – Non-Determinism

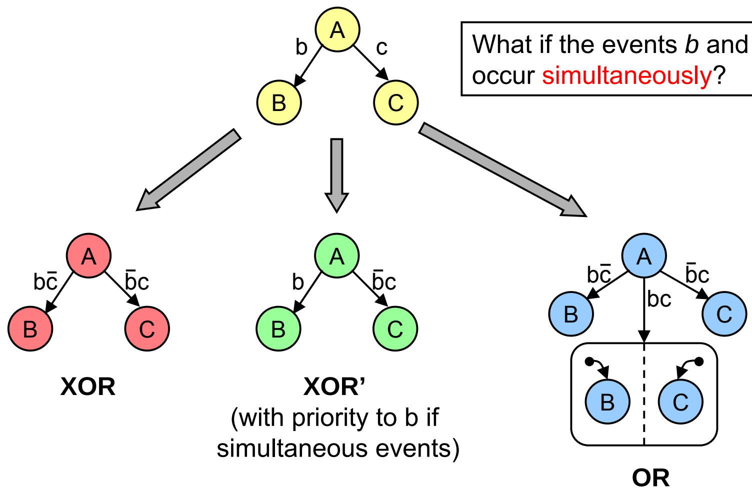


State Diagram



Conflicts – OR *or* XOR?

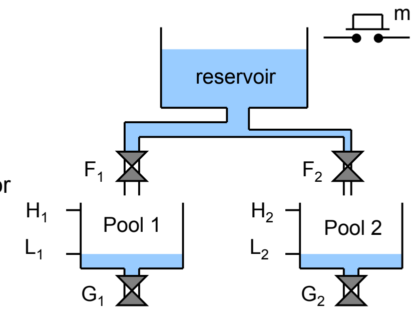
What if the events *b* and *c* occur simultaneously?



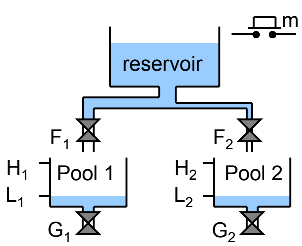
Real Time Exercise – Reservoir

- Initial Condition
  - Empty pools, faucets closed
- Sensors & regulators
  - F_i, G_i = 1 if closed
  - H_i, L_i = 1 if water is above sensor
- Operation

  After pressing m, the pools are filled up to level H_i. When pool i has reached H_i, close F_i and open G_i until the water level reaches L_i. Restarting is only possible after both pools have been emptied.
- Q: Draw a StateChart that models this system.

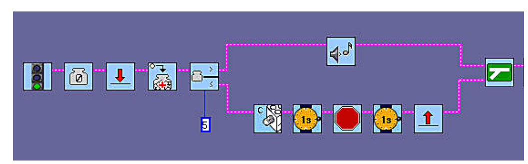


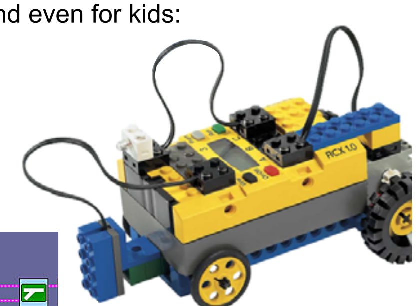
Real Time Solution – Reservoir



Usability

- Intuitive language to describe event driven automata
- New: Concurrency incl. synchronization
- Used in different flavors in industry and even for kids:





Summary

- ***Advantages*** of hierarchical state machines:
  - Simple transformation into efficient hardware and software implementations.
  - Efficient simulation.
  - Basis for formal verification (usually via symbolic model checking), if in reactions only events are generated.
- ***Disadvantages***:
  - Intricate for large systems, limited re-usability of models.
  - No formal representation of operations on data.
  - Large part of the system state is hidden in variables. This limits possibilities for efficient implementation and formal verification.

Where are we?

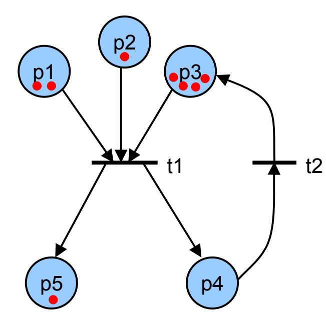
- **StateCharts**
  - Hierarchy
  - Concurrency
  - Events and Actions
  - Simulation Semantics
  - Non-Determinism and Conflicts
- **Petri Nets**
  - Notation
  - Concurrency
  - Petri Net Languages
  - Behavioral Properties
  - Analysis

Petri nets – Motivation

- In contrast to hierarchical state machines, state transitions in Petri nets are asynchronous. The ordering of transitions is partly uncoordinated; it is specified by a partial order.
- Therefore, Petri nets can be used to model concurrent distributed systems.
- Many flavors of Petri nets are in use, e.g.
  - Activity charts (UML)
  - Data flow graphs and marked graphs
- Invented by Carl Adam Petri in 1962 in his thesis "*Kommunikation mit Automaten*"

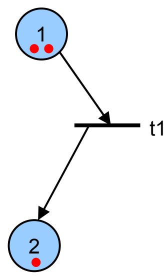
Petri net – Definition

- A Petri net is a bipartite, directed graph defined by a tuple (S, T, F, M_0), where
  - S is a set of places $\mathbf{p_i}$
  - T is a set of transitions $\mathbf{t_i}$
  - F is a set of edges (flow relations) $\mathbf{f_i}$
  - $M_0 : S \to \mathbb{N}$; the initial marking



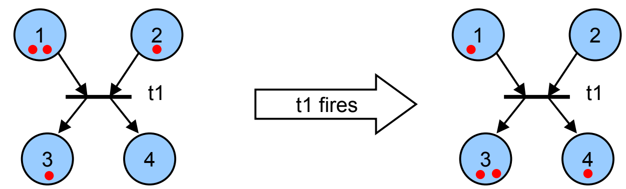
Token marking

- Each place p_i is marked with a certain number of tokens
- The initial distribution of the tokens is given by M_0
- M(*s*) denotes the marking of a place *s*
- The distribution of tokens on places defines the state of a Petri net
- The dynamics of a Petri net is defined by a token game



Token game of Petri nets

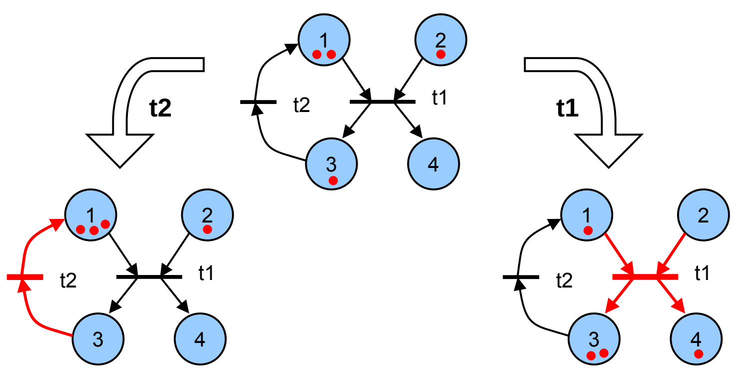
- A marking M activates a transition **t** ∈ T if each place $\mathbf{p_i}$ connected through an edge $\mathbf{f_i}$ to **t** contains at least one token.
- If a transition t is activated by M, a state transition to M' fires (happens) eventually.
- Only one transition is fired at any time.
- When a transition fires, it
  - Consumes a token from each of its input places
  - Adds a token to each of its output places



Non-Deterministic Evolution

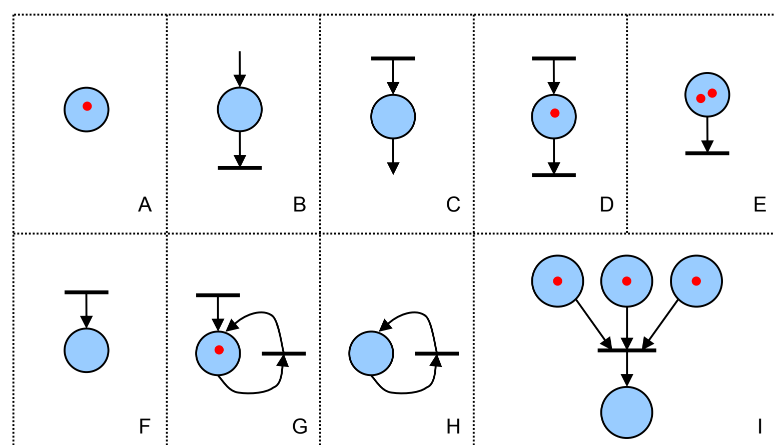
The evolution of Petri nets is not deterministic.

- Any of the activated transactions might fire



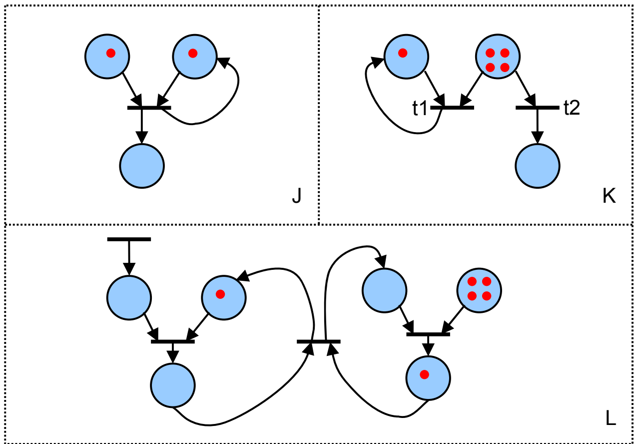
Syntax Exercise

Q: Is it a valid Petri Net? Which transitions are activated? Marking after firing?



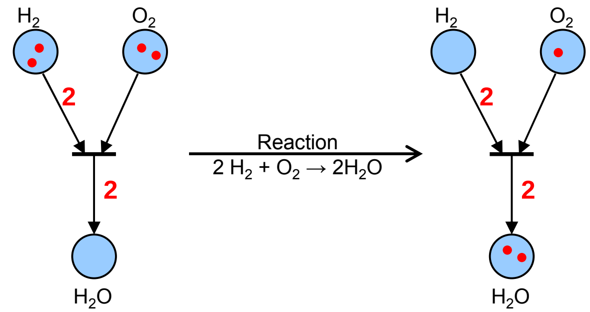
Syntax Exercise (2)

Q: Is it a valid Petri Net? Which transitions are activated? Marking after firing?



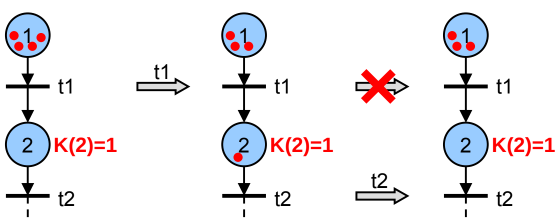
Weighted Edges

- Associating weights to edges:
  - Each edge f_i has an associated weight $W(f_i)$ (defaults to 1)
  - A transition **t** is active if each place p_i connected through an edge f_i to **t** contains at least W(f) tokens.



Finite Capacity Petri Net

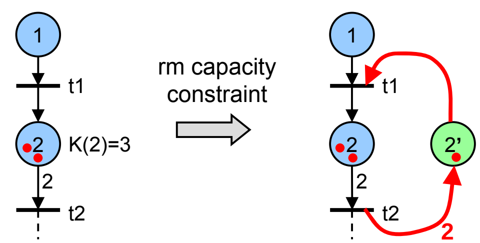
- Each place p_i can hold maximally $K(p_i)$ tokens
- A transition **t** is only active if all output places p_i of **t** cannot exceed $K(p_i)$ after firing **t**.



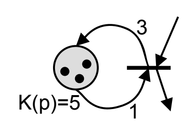
- *Pure* finite capacity Petri Nets can be transformed into equivalent infinite capacity Petri Nets (without capacity restrictions).
- Equivalence: Both nets have the same set of all possible firing sequences

Removing Capacity Constraints

- For each place p with K(p) > 1, add a complementary place **p'** with initial marking M_0(**p'**) = K(**p**) – M_0(**p**).
- For each outgoing edge **e** = (**p**, **t**), add an edge **e'** from **t** to **p'** with weight W(**e**).
- For each incoming edge **e** = (**t**, **p**), add an edge **e'** from **p'** to **t** with weight W(**e**).

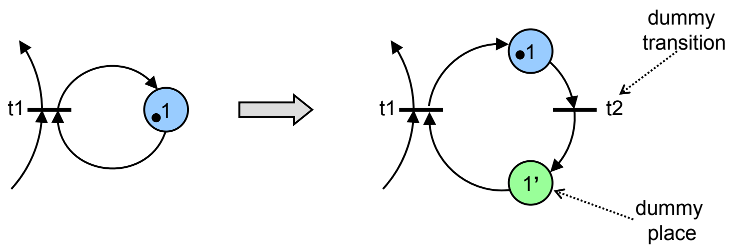


Note: Only works for pure Petri nets, i.e. without self loops.



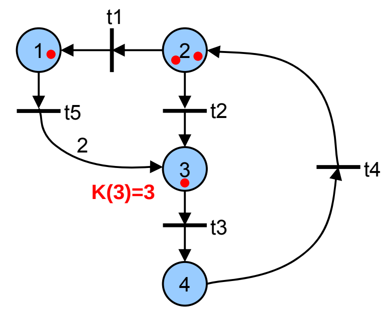
Resolving Self-Loops

- The algorithm to remove capacity constraints works if the Petri net has no self loops (is pure).
- No Problem! Rewrite the Petri net without self loops:



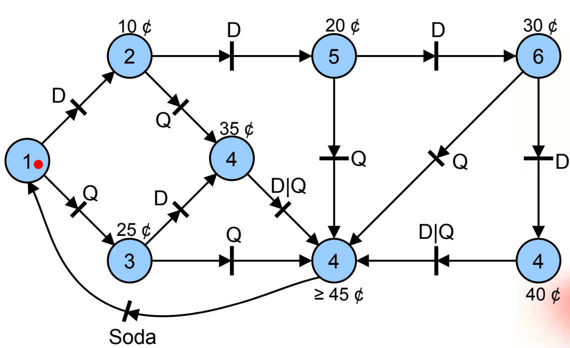
Your turn!

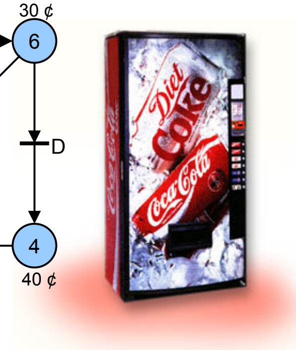
- Remove the capacity constraint from place 3



Modeling FSM

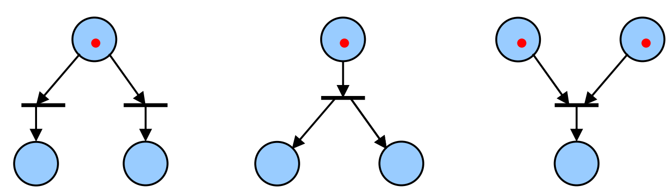
- FSM can be represented by a subclass of Petri nets, where each transition has *exactly* one incoming edge and one outgoing edge.
- Such Petri nets are called state machines
- Coke vending machine revisited





Concurrent Activities

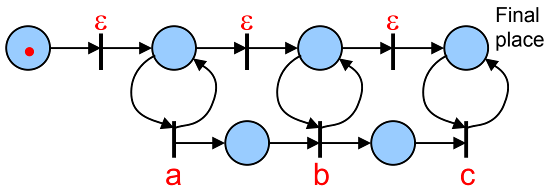
- State machines allow representation of decision, but no synchronization.
- General Petri nets support concurrency with intuit notation:



decision / conflict fork join / synchronization

Petri Net Languages

- Transitions labeled with (not necessarily distinct) symbols
- Sequence of firing the transitions generates string of symbols



$L(M_0)$ = ???

- Every finite-state machine can be modeled by a Petri net

Every regular language is a Petri net language

Behavioral Properties

Reachability

A marking M_n is *reachable* iff there exists a sequence of firings $\{t_1, t_2, \ldots t_n\}$ s.t. $\mathbf{M_n = M_0 \cdot t_1 \cdot t_2 \cdot \ldots \cdot t_n}$

Reachability is decidable, but takes exponential space (and time) for the general case

K-Boundedness

A Petri net (N, M_0) is *K-bounded if* the number of tokens in every place never exceeds *K*.

Safety

1-Boundedness: Every node holds at most 1 token at any time

Behavioral Properties (2)

Liveness

Having reached M_n from M_0, can we eventually fire any transition?

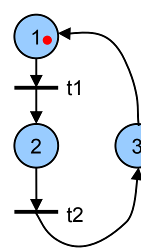
Closely related to the *complete absence of dead locks*

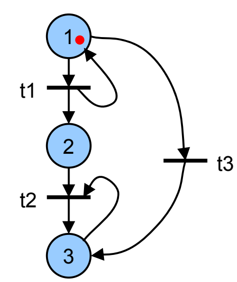
A transition t in a Petri net (N, M_0) is

dead	if t cannot be fired in any firing sequence of $L(M_0)$
L1-live	if t can be fired at least once in some sequence of $L(M_0)$
L2-live	if, $\forall k \in \mathbb{N}^+$, t can be fired at least *k* times in some sequence of $L(M_0)$
L3-live	if t appears infinitely often in some sequence of $L(M_0)$
L4-live	(live) if t is L1-live for every marking reachable from M_0

Note: L4-liveness $\Rightarrow$ L3-liveness $\Rightarrow$ L2-liveness $\Rightarrow$ L1-liveness

Liveness Example





Analysis Methods

Coverability tree

Enumeration of all reachable markings, limited to small nets

Incidence Matrix

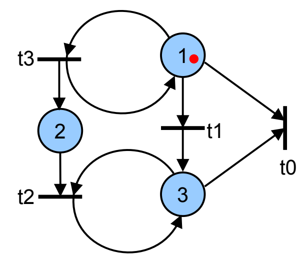
A necessary condition for reachability

Reduction Rules

Simplification rules to rewrite a Petri net, conserving liveness, safeness and boundedness properties.

Coverability Tree

- Question: What token distributions are reachable?
- Problem: There might be infinitely many $\Rightarrow$ must avoid infinite tree
- Solution: Detect & handle infinite cycles
  - Special symbol ω to denote an arbitrary number of tokens



Coverability Tree – the Algorithm

Special symbol ω, similar to ∞: $\forall n \in \mathbb{N}$: $\omega > n$; $\omega = \omega + n$; $\omega \geq \omega$

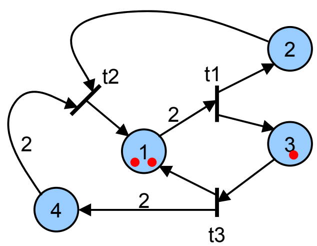
- Label initial marking M_0 as root and tag it as *new*
- **while** *new* markings exist, pick one, say M
  - If M is identical to a marking on the way from the root to M, mark it as *old*; **break**;
  - If no transitions are enabled at M, tag it as *deadend*;
  - For each enabled transition t at M do
    - Obtain marking M' = M · t
    - If there exists a marking M'' on the way from the root to M s.t. $M'(p) \geq M''(p)$ for each place p and $M' \neq M''$, replace $M'(p)$ with ω for *p* where $M'(p) > M''(p)$.
    - Introduce M' as a node, draw an arc with label t from M to M' and tag M' *new*.

Results from the Coverability Tree T

- The net is **bounded** iff ω does not appear in any node label of T
- The net is **safe** iff only '0' and '1' appear in the node labels of T
- A transition t is **dead** iff it does not appear as an arc in T
- If M is **reachable** from M_0, then there exists a node M' s.t. $M \leq M'$.
  (This is a necessary, but not sufficient condition for reachability.)
- For *bounded* Petri nets, this tree is also called reachability tree, as all reachable markings are contained in it.

Incidence Matrix

- Goal: Describe a Petri net through equations
- The incidence matrix **A** describes the token-flow according for the different transitions
- **A_{ij}** = gain of tokens at node **i** when transition **j** fires
- A marking M is written as a $m \times 1$ column vectors



$$A = \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix}$$

$$M_0 = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

State Equation

- The firing vector u_i describes the firing of transition *i*. It consists of all '0', except for the *i*-th position, where it has a '1'.

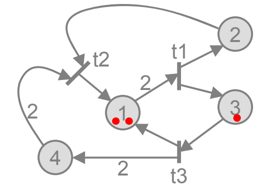
  E.g. $t1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ $t2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ $t3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

- A transition *t* from M_k to M_{k+1} is written as

$$M_{k+1} = M_k + A \cdot u_i$$

M_1 is obtained from M_0 by firing t3

$$\begin{bmatrix} 3 \\ 0 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$



$$A = \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix}$$

$$M_0 = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

State Equation: Reachability

- A marking M_k is reachable from M_0 if there is a sequence of transitions {t1, t2, …, tk} such that $M_k = M_0 \cdot t1 \cdot t2 \cdot \ldots \cdot tk$.
- Expressed with the incidence matrix:

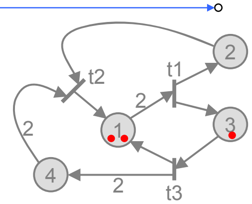
$$M_k = M_0 + A \cdot \sum_{i=1}^{k} u_i \qquad (1)$$

which can be rewritten as

$$M_k - M_0 = \Delta M = A \cdot \vec{x} \qquad (2)$$

If M_k is reachable from M_0, equation (2) must have a solution where all components of $\vec{x}$ are positive integers.

(This is a necessary, but not sufficient condition for reachability.)



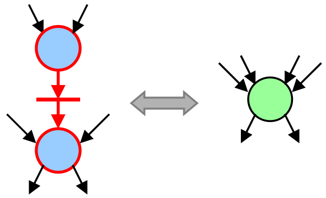
$$A = \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix}$$

$$M_0 = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

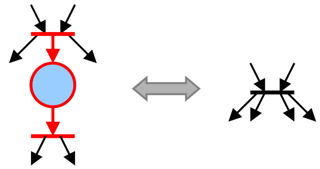
Reduction Rules

- Analysis of Petri nets tedious, especially for large, complex nets
- Often, the complexity for analysis increases exponentially with the size of the Petri net
- Solution: Simplify the net while retaining the properties to analyze.
- In our case, the properties in question are
  - Liveness
  - Safeness
  - Boundedness
- 6 of the simplest reduction rules are shown in the sequel

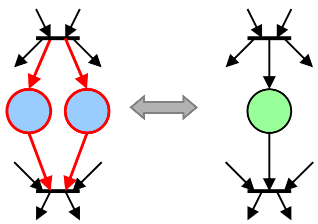
Reduction Rules (2)



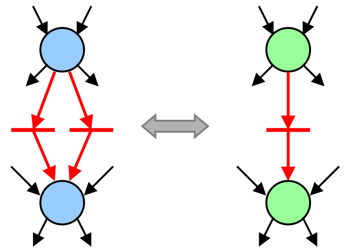
Fusion of Series Places (FSP)



Fusion of Series Transformations (FST)

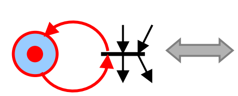


Fusion of Parallel Places (FPP)

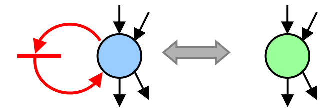


Fusion of Parallel Transformations (FPT)

Reduction Rules (3)

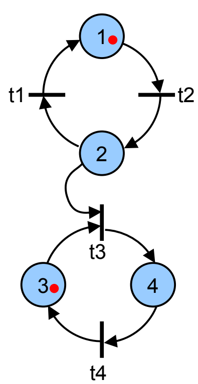


Elimination of Self Loop Places (ESP)



Elimination of Self Loop Transitions (EST)

Reduction Example



Common Extensions

- Colored Petri nets: Tokens carry values (colors)
  Any Petri net with finite number of colors can be transformed into a regular Petri net.
- Continuous Petri nets: The number of tokens can be real.
  Cannot be transformed to a regular Petri net
- Inhibitor Arcs: Enable a transition if a place contains **no** tokens
  Cannot be transformed to a regular Petri net