# Modeling and Verification

( 2018)

Didier Buchs

Petri net course based on Stefan Schwoon course, Stuttgart; other parts with contribution of Alexis Marechal, Steve Hostettler, SMV Lab

Semantics, Modeling and Verification Group

Department of Computer Science

University of Geneva

## Modeling and Verification

Lectures: Thursday Battelle 316 - 14:15-16.00,

Exercises: Thursday Battelle 316 - 16:15-18.00,

Credits: Lectures (4 credits): oral exam (2/3) + TPS (1/3)

(more than 3.0 to pass the exam and the second session)

Assistants: Stefan Klikovits, Damien Morard

## Teaching material

The outline of the Petri net course and temporal logic will follow a course by Stefan Schwoon and Keijo Heljanko, whose slides form the basis of this course.

The material is partly based on the course “Parallel and Distributed Digital Systems” at Helsinki University of Technology, by Marko Mäkelä, Teemu Tynjälä, Keijo Heljanko, and Johan Lilius.

ADT , APN and DD (model checking) with contribution from Alexis Marechal and Steve Hostettler from SMV group, Geneva.

## The objective of the course

The course aims to give the necessary skills for modelling , functional, parallel and distributed systems, specifying requirements for them, and for verifying these requirements hold on these systems.

- Modelling: Petri nets, Algebraic Specification, Algebraic Petri nets, transition systems and process algebra
- Specifying: formulae in temporal logic, equational logic.
- Verifying: reachability analysis, model checking with decision diagrams, rewrite systems.

Coding in Swift will be used to support concrete understanding of the theoretical concepts.

## Software Modelling

Software Modelling: What system to realize?

as opposed to

Software Design and Implementation: How are we producing the product?

Software Modelling needs expressive language to describe the expected functionalities of a system

## Software Verification

Software Validation: Are we producing the right product?

as opposed to

Software Verification: Are we producing the product right? - Boehm

Software verification deals with checking if a software system performs the specified functionalities correctly

## Analysis versus Modelling Systems

### Analysis

represent  problem domains from multiple perspectives

Discover characteristics of the system

### Modelling

Describe entirely and non-ambiguously the system

Need a very expressive language adapted to the problem domain

Focus on functional or non-functional aspects

## Parallel and distributed systems: what are they?

parallel: multiple processes act simultaneously (concurrently),

distributed: processes may co-operate to achieve a goal

> A distributed system is one on which I cannot get any work done, because a machine I have never heard of has crashed. *L. Lamport*

communication: the processes exchange messages (synchronously or asynchronously)

reactive: the system operates continuously, reacting on external events (rather than computing one result and then terminating)

nondeterminism: there are alternative execution orders (stemming from concurrency, or from incomplete system description)

## Parallel and distributed systems: problems

Systems with inherent parallelism often are so complex that they simply cannot be built by trial-and-error. Some problems caused by the specific nature of these systems are:

nondeterminism:

All possible execution sequences need to be considered to prove correctness of a system.

communication:

synchronous: parties may deadlock while waiting for another party.

asynchronous: message transmission may be unreliable, messages may arrive at any time (including at inconvenient times)

## Parallel and distributed systems: problems

reactivity:

need to consider potentially infinite executions.

distribution: operating in an unknown environment

number of processes (e.g. participants in a protocol) may be unknown

behaviour of other components may be unknown

## Parallel and distributed systems: what for?

They arise naturally, e.g. network file system, telecommunication systems, car electronics, . . .

Speeding up: it is cheaper to obtain many slow processors than a super-fast one, and the speed of the fastest available processor may be inadequate for the application

Redundancy: the system remains operational during maintenance breaks and partial hardware failures, or during large accidents and acts of war (geographical distributio, ArpaNet)

Modularity: it may be easier to manage several specialised processes than a single complex process that takes care of everything

## Specifications: properties of systems

Safety: “The system never reaches a bad state”; some property holds throughout the execution.

Examples: deadlock freedom, mutual exclusion, . . .

Liveness: “There is progress in the system”; some actions occur infinitely often.

Inevitability: “Eventually, something will happen.”

Response: “Whenever $X$ occurs, $Y$ will eventually occur.”

Examples: sent messages are eventually received, each request is served

Fairness assumptions:

“$X$ holds, assuming that no process is starved of CPU time.”

## Specifying systems

A system specification captures the assumptions and requirements of the operations.

Specifications should be unambiguous but not necessarily complete.

Specifications describe the allowed computations (or executions).

A specification is a “contract” between the customer and the software supplier.

In our setting, specifications are formal descriptions of systems.

# Advantages of formal description techniques

unambiguity: specifications with precise mathematical meaning instead of colloquial language

correctness: automated verification that the system fulfils its requirements

completeness: the specification forms a checking list

consistency: inconsistent or unreasonable requirements can be detected from specifications in an early phase

Many software description techniques, such as UML, often aren’t formal enough.

## Why formal methods?

Computers invade more and more aspects of our lives (home PCs, mobile phones, car electronics, . . . )

Software becomes increasingly more complex.

Therefore:

- Producing bug-free software becomes more difficult.
- Cost of bugs can be enormous (economic, reputation, even human lives).

The later an error is detected, the more expensive it is to correct.

## Some (in)famous bugs

Floating point error in Pentium processor (1994)

Toll collection on German motorways not working

Errors in space missions: Ariane 5, Mars polar lander

Bug in Needham-Schröder protocol (key exchange)

Other examples: `http://www5.in.tum.de/~huckle/bugs.html`

## Are formal methods a silver bullet?

No - we can’t hope to automatically analyse arbitrary properties of arbitrary programs, because

- they are too large;
- the problems may actually be undecidable in principle.

Thus, formal efforts are concentrated on:

- critical parts of systems;
- decidable subclasses of systems or properties.

## Applications for the methods

### Communication

verifying and testing communication protocols

evaluating the performance (queueing times, throughput, . . . )

### Safety-critical systems

railroad interlocking

aircraft and air traffic control systems

### Hardware design

processors, peripheral interfaces, memory caches and buses

(verification of Pentium 4 FPU logic)

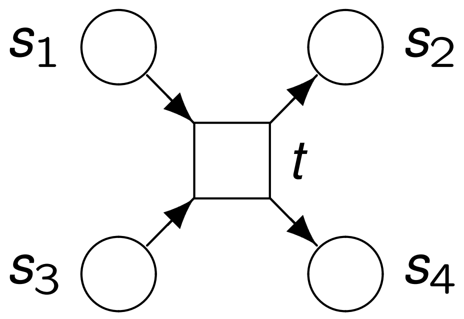
## Two examples

Dining philosophers, modeled using a Petri net.

Alternating-Bit protocol, modeled using different formalisms, e.g. pseudocode and communicating automata.

## Petri nets

Petri nets are a basic model of parallel and distributed systems, designed by Carl Adam Petri in 1962 in his PhD Thesis: “Kommunikation mit Automaten”. The basic idea is to describe state changes in a system with transitions.

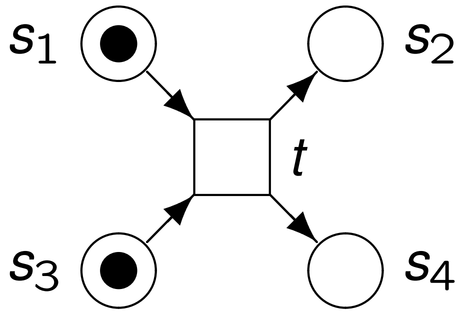


Petri nets contain places ○ (Stelle) and transitions □ (Transition) that may be connected by directed arcs.

Transitions symbolise actions; places symbolise states or conditions that need to be met before an action can be carried out.

## Behaviour of Petri nets

Places may contain tokens that may move to other places by executing ("firing") actions.

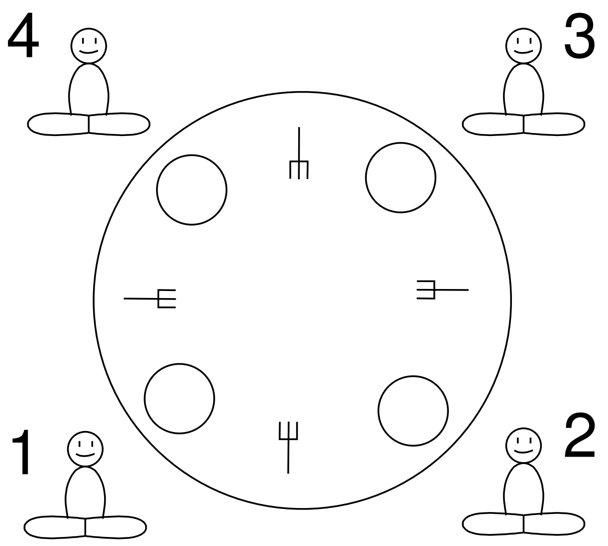


In the example, transition $t$ may "fire" if there are tokens on places $s_1$ and $s_3$. Firing $t$ will remove those tokens and place new tokens on $s_2$ and $s_4$.

## Example: Dining philosophers

There are philosophers sitting around a round table.

There are forks on the table, one between each pair of philosophers.



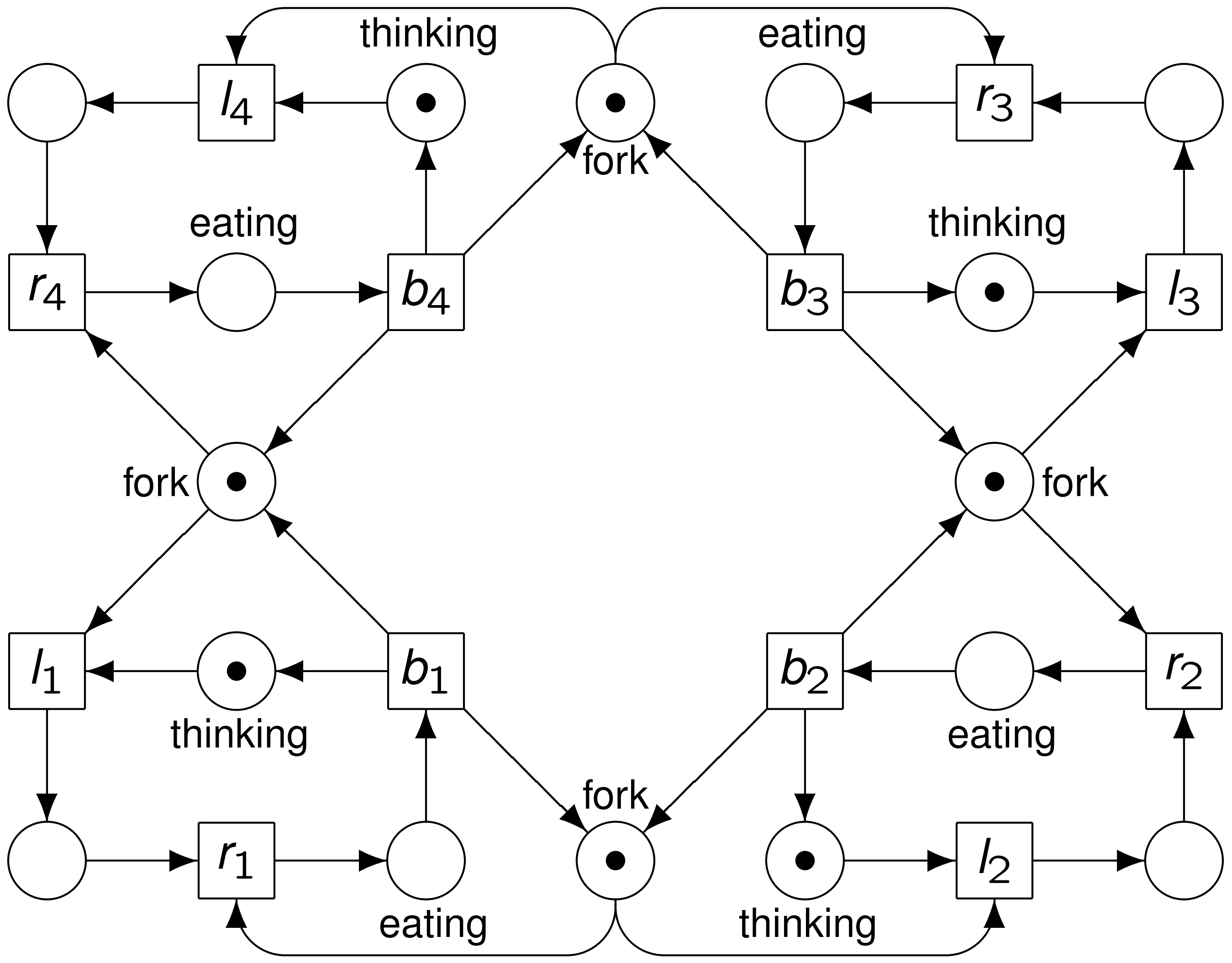
The philosophers want to eat spaghetti from a large bowl in the center of the table.

Unfortunately the spaghetti is of a particularly slippery type, and a philosopher needs both forks in order to eat it.

The philosophers have agreed on the following protocol to obtain the forks: Initially philosophers think about philosophy, when they get hungry they do the following: (1) take the left fork, (2) take the right fork and start eating, (3) return both forks simultaneously, and repeat from the beginning

How can we model the behaviour of the philosophers?

# Dining philosophers: Petri net



## Dining philosophers: Questions

Can two neighbouring philosophers eat at the same time?

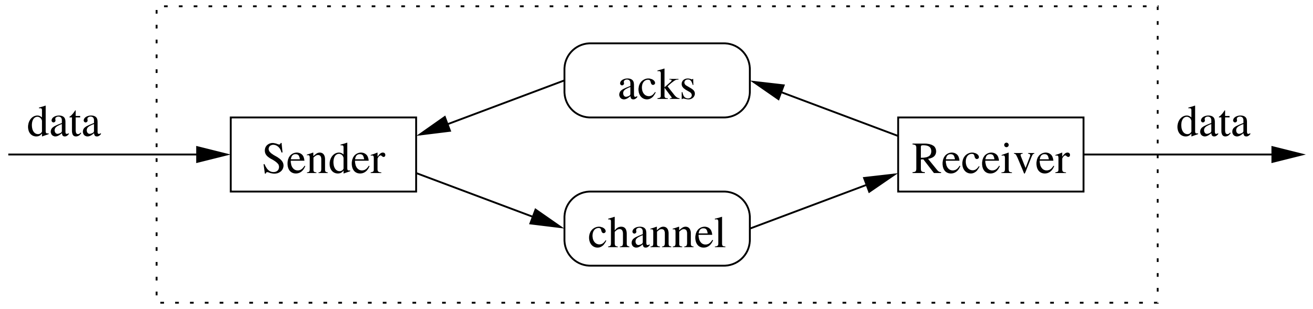
Can the philosophers starve to death?

Can an individual philosopher eventually eat, assuming he wants to?

## Example: Reliable connection on a lossy channel

Many data communication systems are based on unreliable connections that may distort, lose or duplicate messages. Distorted messages can be discarded by using checksums, and lost and duplicated messages can be detected by numbering the messages.

The basic solution assigns sequence numbers to the messages of transit in such a way that the recipient can detect a lost message and ask the sender to repeat previous messages. Whenever the recipient has obtained a contiguously numbered sequence of messages, it can relay them to the consumer.



## Alternating bit protocol

In the alternating bit protocol, there are two sequence numbers for messages: 0 and 1 (the alternating bit). Both the sender and the recipient have their own copy of the alternating bit. The sender holds it in the variable $s$, the recipient in $r$.

- Sender: send a message tagged with $s$.
  - if no acknowledgement tagged with $s$ arrives in due course, repeat the message;
  - if the recipient acknowledges with $s$, toggle $s$ and send next message.
- Recipient: receive a message tagged with $b$, i.e. the value of $s$ at sending time; acknowledge receipt of $b$.
  - if $b$ agrees with $r$, relay the message to consumer and toggle $r$;
  - otherwise discard the message.

## Alternating Bit protocol: Questions

Is the protocol correct?

Is every message eventually delivered, assuming that the channels are not permanently faulty?

Are messages received in the right order?

Can two messages with the same alternating bit value be confused?

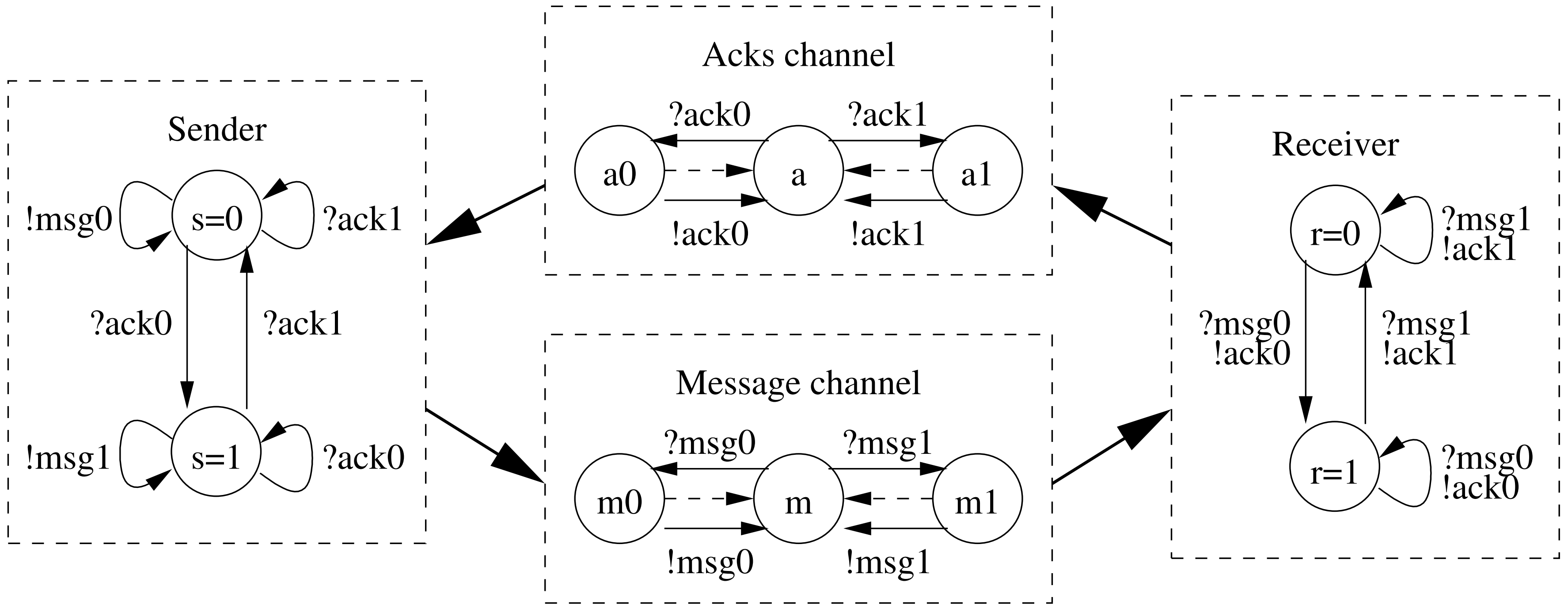
## Alternating bit protocol: Pseudocode

Assume that send and recv are primitives for sending and receiving a single piece of data over a channel.

```
void abp_send () {
  i = 0; s = 0;
  while (true) {
    i = i+1; s = !s;
    while (true) {
      send (data[i],s);
      wait (timeout);
      if (recv(ack) == s) break;
    }
  }
}
```

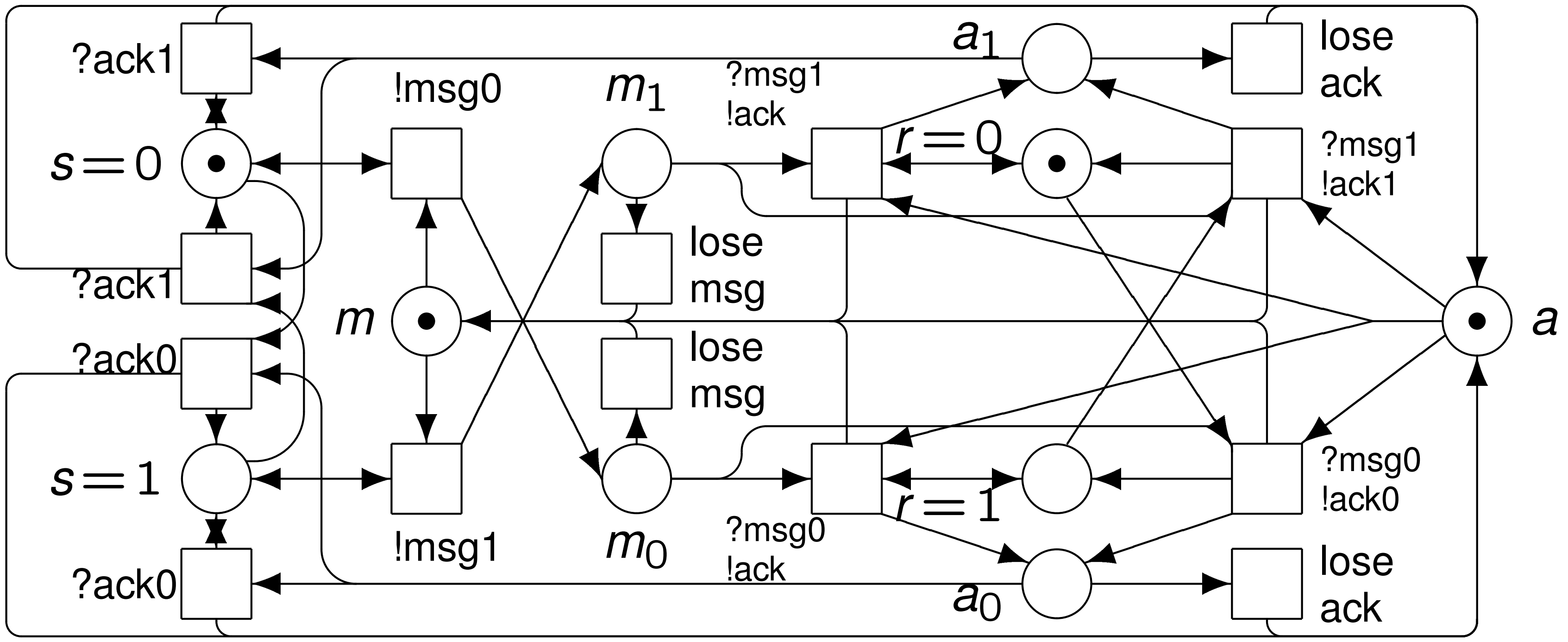
```
void abp_recv () {
  i = 0; r = 0;
  while (true) {
    i = i+1; r = !r;
    while (true) {
      recv (c,b);
      send (b);
      if (b == r) { data[i] = c; break; }
    }
  }
}
```

## Alternating bit protocol: Communicating FA



The figure depicts the operation of the sender, the recipient and the channels as communicating finite state automata. Transmissions and receptions of messages are marked with exclamation marks and question marks. A transmission and a reception with the same name in neighbouring components are carried out simultaneously. The channels have capacity for at most one message at a time.

## Alternating bit protocol: Petri net



## Alternating bit protocol: state space

Even though the data transmitted by the protocol may be the main concern of end users, the data payload of the messages is irrelevant for observing the operation of the protocol. The less memory a model contains, the easier it can be verified, because a system with $b$ bits of memory can assume at most $2^b$ states.

In the Petri net model, the data has been abstracted away, as well as the value of the timeout (the timeout is assumed to be able to expire at any moment). The timeout could be handled by modelling a clock, but it would make verification harder (and the precise value of the timeout is not important here).

The state of a distributed system consists of the states of its component systems, for instance $\{s=0, s=1\} \times \{r=0, r=1\} \times \{m, m0, m1\} \times \{a, a0, a1\}$. In the beginning, each component system is in its initial state, which corresponds to the initial state to the whole system, e.g. $\langle s=0, r=0, m, a\rangle$.

## Constructing the state space (idea)

The state of a Petri system is formed by the distribution of tokens in the places.

The state changes when enabled transitions are fired.

A transition is enabled if each of its pre-places contains a token.

When an enabled transition fires, a token is removed from each of its pre-places and a token is inserted to each of its post-places.

The state space can be represented by a graph.

Nodes of the graph correspond to distributions of tokens.

Edges of the graph correspond to firing of transitions.

By constructing the state space, it is “fairly” easy to ensure that the alternating bit protocol works. There are at most $2 \cdot 2 \cdot 3 \cdot 3 = 36$ states.

It turns out that from $\langle s=0, r=0, m, a\rangle$, there are 18 reachable nodes and 40 edges.

## Many views of one system

Parallel and distributed systems can be described in very many ways:

- with Petri nets (place/transition nets or high-level nets),
- in some programming language,
- with communicating automata,
- with process algebra, or
- in some semi-formal description languages (such as UML).

It makes sense to choose the presentation format according to the object being described and to the desired accuracy. Also formal descriptions can be presented in different equivalent notations: graphical, tabular, or plain text.

## Lecture 2: Place/Transition Nets

## Place/Transition Nets

Let us study Petri nets and their firing rule in more detail:

- A place may contain several tokens, which may be interpreted as resources.
- There may be several input and output arcs between a place and a transition. The number of these arcs is represented as the weight of a single arc.
- A transition is enabled if its each input place contains at least as many tokens as the corresponding input arc weight indicates.
- When an enabled transition is fired, its input arc weights are subtracted from the input place markings and its output arc weights are added to the output place markings.

## Place/Transition Net

A Place/Transition Net (P/T net) is a tuple $N = \langle P, T, F, W, M_0 \rangle$, where

- $P$ is a finite set of places,
- $T$ is a finite set of transitions,
- the places $P$ and transitions $T$ are disjoint ($P \cap T = \emptyset$),
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation,
- $W\colon F \to (\mathbb{N} \setminus \{0\})$ is the arc weight mapping, and
- $M_0\colon P \to \mathbb{N}$ is the initial marking representing the initial distribution of tokens.

## P/T nets: Remarks

If $\langle p, t \rangle \in F$ for a transition $t$ and a place $p$, then $p$ is an input place of $t$,

If $\langle t, p \rangle \in F$ for a transition $t$ and a place $p$, then $p$ is an output place of $t$,

Let $a \in P \cup T$. The set ${}^\bullet a = \{a' \mid \langle a', a \rangle \in F\}$ is called the pre-set of $a$, and the set $a^\bullet = \{a' \mid \langle a, a' \rangle \in F\}$ is its post-set.

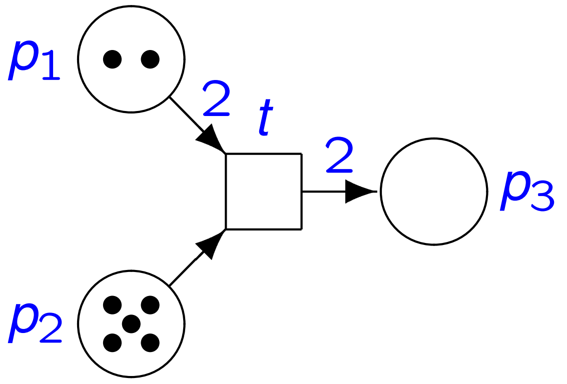
When drawing a Petri net, we usually omit arc weights of $1$. Also, we may either denote tokens on a place either by black circles, or by a number.

## Alternative definitions

Sometimes the notation $S$ (for Stellen) is used instead of $P$ (for places) in the definition of Place/Transition nets.

Some definitions also use the notion of a place capacity (the maximum number of tokens allowed in a place, possibly unbounded). Place capacities can be simulated by adding some additional places to the net (we will see how later), and thus for simplicity we will not define them in this course.

## Place/Transition Net: Example



The place/transition net $\langle P, T, F, W, M_0\rangle$ above is defined as follows:

- $P = \{p_1, p_2, p_3\}$,
- $T = \{t\}$,
- $F = \{\langle p_1, t\rangle, \langle p_2, t\rangle, \langle t, p_3\rangle\}$,
- $W = \{\langle p_1, t\rangle \mapsto 2, \langle p_2, t\rangle \mapsto 1, \langle t, p_3\rangle \mapsto 2\}$,
- $M_0 = \{p_1 \mapsto 2, p_2 \mapsto 5, p_3 \mapsto 0\}$.

## Notation for markings

Often we will fix an order on the places (e.g., matching the place numbering), and write, e.g., $M_0 = \langle 2, 5, 0 \rangle$ instead.

When no place contains more than one token, markings are in fact sets, in which case we often use set notation and write instead $M_0 = \{p_5, p_7, p_8\}$.

Alternatively, we could denote a marking as a multiset, e.g. $M_0 = \{p_1, p_1, p_2, p_2, p_2, p_2, p_2\}$.

The notation $M(p)$ denotes the number of tokens in place $p$ in marking $M$.

## The firing rule revisited

Let $\langle P, T, F, W, M_0 \rangle$ be a Place/Transition net and $M : P \rightarrow \mathbb{N}$ one of its markings.

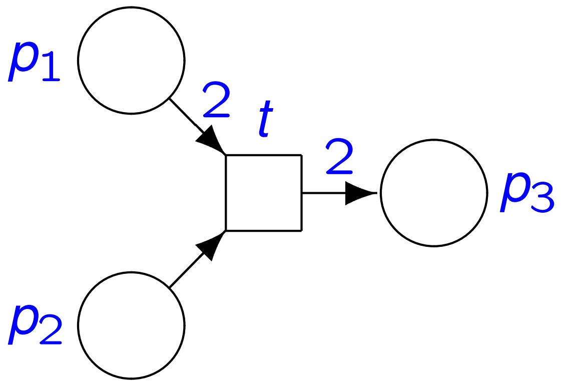
Firing condition:
Transition $t \in T$ is $M$-enabled, written $M \xrightarrow{t}$, iff $\forall p \in {}^{\bullet}t : M(p) \geq W(p, t)$.

Firing rule:
An $M$-enabled transition $t$ may fire, producing the successor marking $M'$, written $M \xrightarrow{t} M'$, where

$$\forall p \in P : M'(p) = M(p) - \bar{W}(p, t) + \bar{W}(t, p)$$

where $\bar{W}$ is defined as $\bar{W}(x, y) := W(x, y)$ for $\langle x, y \rangle \in F$ and $\bar{W}(x, y) := 0$ otherwise.

## The firing rule of Place/Transition Nets: Example



| Marking $M$ | $M \xrightarrow{t}$ | $M'$ |
|---|---|---|
| $\{p_1 \mapsto 2, p_2 \mapsto 5, p_3 \mapsto 0\}$ | enabled | $\{p_1 \mapsto 0, p_2 \mapsto 4, p_3 \mapsto 2\}$ |
| $\{p_1 \mapsto 0, p_2 \mapsto 4, p_3 \mapsto 2\}$ | disabled | |
| $\{p_1 \mapsto 1, p_2 \mapsto 5, p_3 \mapsto 0\}$ | disabled | |

Note: If $M \xrightarrow{t} M'$, then we call $M'$ the successor marking of $M$.

# Reachable markings

Let $M$ be a marking of a Place/Transition net $N = \langle P, T, F, W, M_0 \rangle$.

The set of markings reachable from $M$ (the reachability set of $M$, written $\mathit{reach}(M)$) is the smallest set of markings, such that:

1. $M \in \mathit{reach}(M)$, and
2. if $M' \xrightarrow{t} M''$ for some $t \in T$, $M' \in \mathit{reach}(M)$, then $M'' \in \mathit{reach}(M)$.

Let $\mathcal{M}$ be a set of markings. The previous notation is extended to sets of markings in the obvious way:

$$\mathit{reach}(\mathcal{M}) = \cup_{M \in \mathcal{M}}\, \mathit{reach}(M)$$

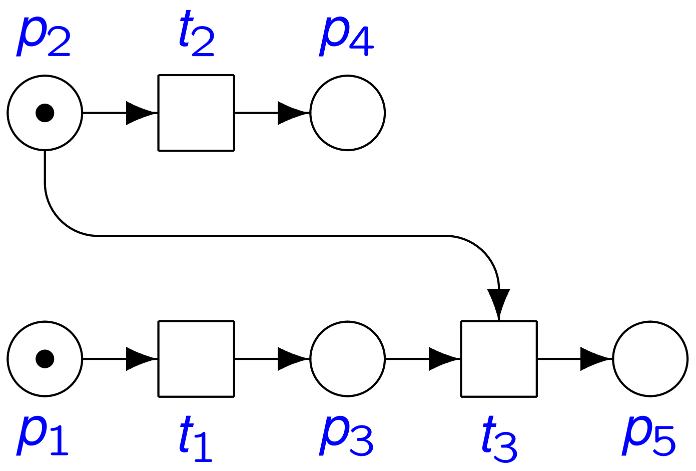
The set of reachable markings $\mathit{reach}(N)$ of a net $N = \langle P, T, F, W, M_0 \rangle$ is defined to be $\mathit{reach}(M_0)$.

## Reachability Graph

The reachability graph of a place/transition net $N = \langle P, T, F, W, M_0 \rangle$ is a rooted, directed graph $G = \langle V, E, v_0 \rangle$, where

- $V = \mathit{reach}(N)$ is the set of vertices, i.e. each reachable marking is a vertex;
- $v_0 = M_0$, i.e. the initial marking is the root node;
- $E = \left\{ \langle M, t, M' \rangle \;\middle|\; M \in V \text{ and } M \xrightarrow{t} M' \right\}$ is the set of edges, i.e. there is an edge from each marking (resp. vertex) $M$ to each of its successor markings, and the edge is labelled with the firing transition.

# Reachability Graph: Example



$$\begin{tikzcd}
& \langle 0, 0, 1, 1, 0\rangle & & \langle 0, 0, 0, 0, 1\rangle \\
\langle 1, 0, 0, 1, 0\rangle \arrow[ur, "t_1"] & & \langle 0, 1, 1, 0, 0\rangle \arrow[ul, "t_2"'] \arrow[ur, "t_3"'] & \\
& {>}\ \langle 1, 1, 0, 0, 0\rangle \arrow[ul, "t_2"] \arrow[ur, "t_1"'] & &
\end{tikzcd}$$

- The weight of each arc is $1$.
- The graph shows that $t_3$ cannot be fired if $t_2$ is fired before $t_1$. Thus, intuitively speaking, $t_1$ and $t_2$ are not independent, even though their presets and postsets are mutually disjunct.

## Computing the reachability graph

```
REACHABILITY-GRAPH($\langle P, T, F, W, M_0 \rangle$)
1   $\langle V, E, v_0 \rangle := \langle \{M_0\}, \emptyset, M_0 \rangle$;
2   $\mathit{Work}$ : set := $\{M_0\}$;
3   while $\mathit{Work} \neq \emptyset$
4   do select $M$ from $\mathit{Work}$;
5       $\mathit{Work} := \mathit{Work} \setminus \{M\}$;
6       for $t \in \mathrm{enabled}(M)$
7       do $M' := \mathrm{fire}(M, t)$;
8           if $M' \notin V$
9             then $V := V \cup \{M'\}$
10                 $\mathit{Work} := \mathit{Work} \cup \{M'\}$;
11          $E := E \cup \{\langle M, t, M' \rangle\}$;
12  return $\langle V, E, v_0 \rangle$;
```

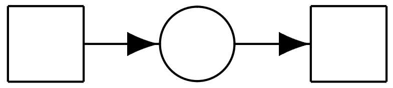
The algorithm makes use of two functions:

- $\mathrm{enabled}(M) := \{t \mid M \xrightarrow{t}\}$
- $\mathrm{fire}(M, t) := M'$
  if $M \xrightarrow{t} M'$

The set $\mathit{Work}$ may be implemented as a stack, in which case the graph will be constructed in a depth-first manner, or as a queue for breadth-first. Breadth first search will find the shortest transition path from the initial marking to a given (erroneous) marking. Some applications require depth first search.

## The size of the reachability graph

In general, the graph may be infinite, i.e. if there is no bound on the number tokens on some place. Example:



Definition: If each place of a place/transition net can contain at most $k$ tokens in each reachable marking, the net is said to be $k$-safe.

A $k$-safe net has at most $(k+1)^{|P|}$ markings; for $1$-safe nets, the limit is $2^{|P|}$.

## Use of reachability graphs

In practice, all analysis tools and methods for Petri nets compute the reachability graph in some way or other. The reachability graph can be effectively computed if the net is $k$-safe for some $k$.

If the net is not $k$-safe for any $k$, we may compute the coverability graph (see next lecture).

# Lecture 3:
# Properties of Place/Transition Nets

## Recap from last Session

We discussed the following topics:

Modeling: Petri nets (and others)

Specifying: reachability (and others, mostly informally)

Verifying: construction of reachability graph

## Programme for this Session

Example: modeling, specifying, and reachability analysis

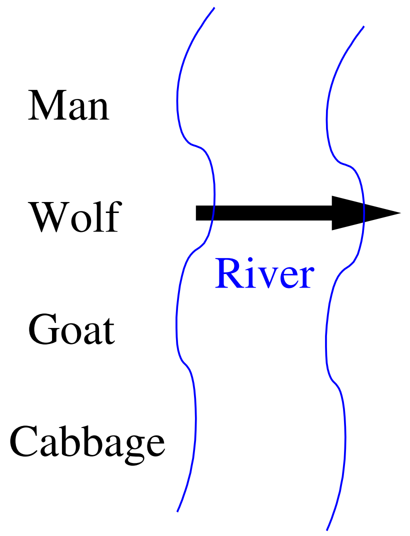
Some basic notions about Petri nets

Coverability graphs

## Example: A logical puzzle

A man is travelling with a wolf, a goat, and a cabbage. The four come to a river that they must cross. There is a boat available for crossing the river, but it can carry only the man and at most one other object. The wolf may eat the goat when the man is not around, and the goat may eat the cabbage when unattended.

Can the man bring everyone across the river without endangering the goat or the cabbage? And if so, how?



## Example: Modeling

We are going to model the situation with a Petri net.

The puzzle mentions the following objects:

> Man, wolf, goat, cabbage, boat. Both can be on either side of the river.

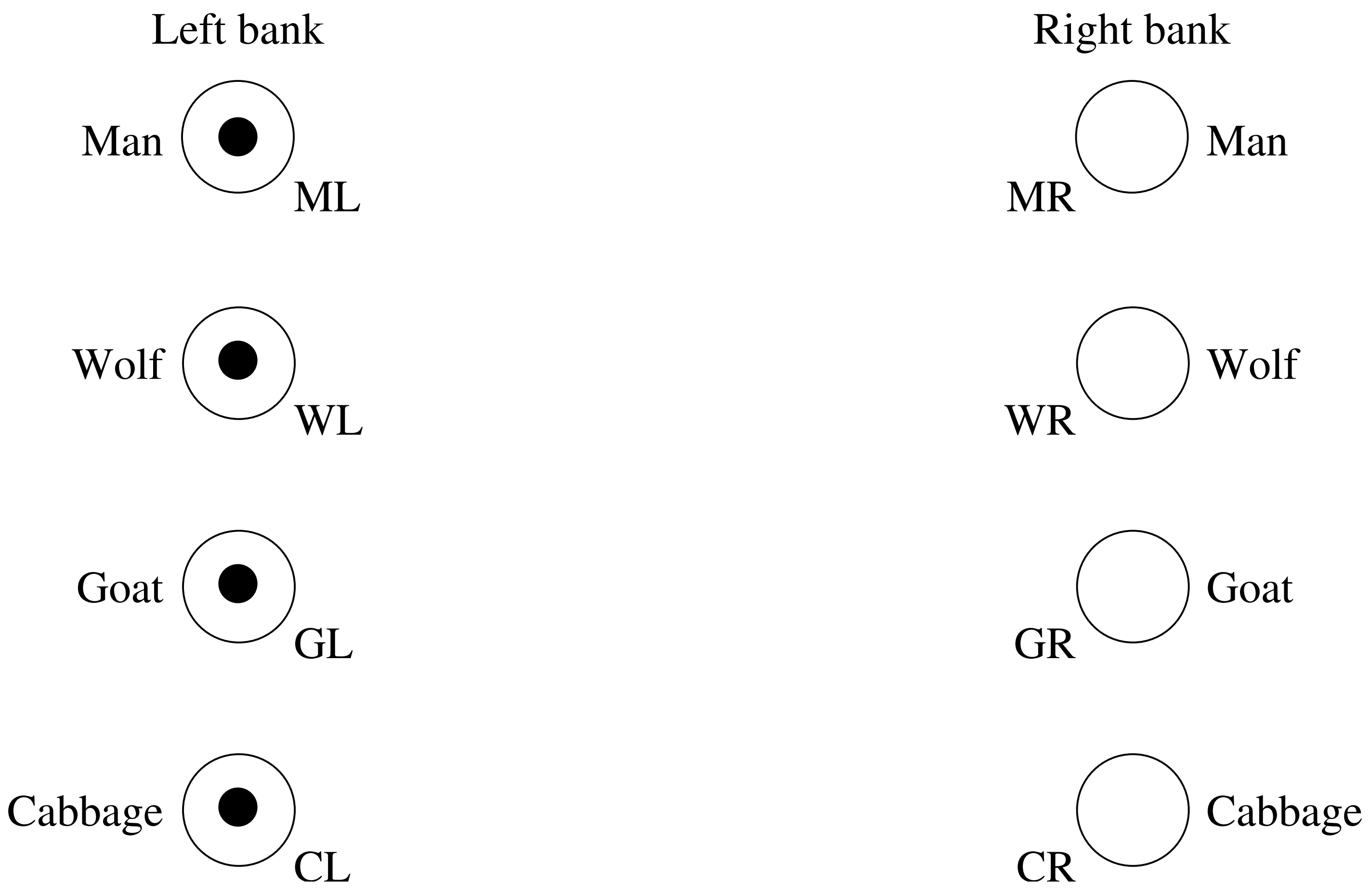
The puzzle mentions the following actions:

> Crossing the river, wolf eats goat, goat eats cabbage.

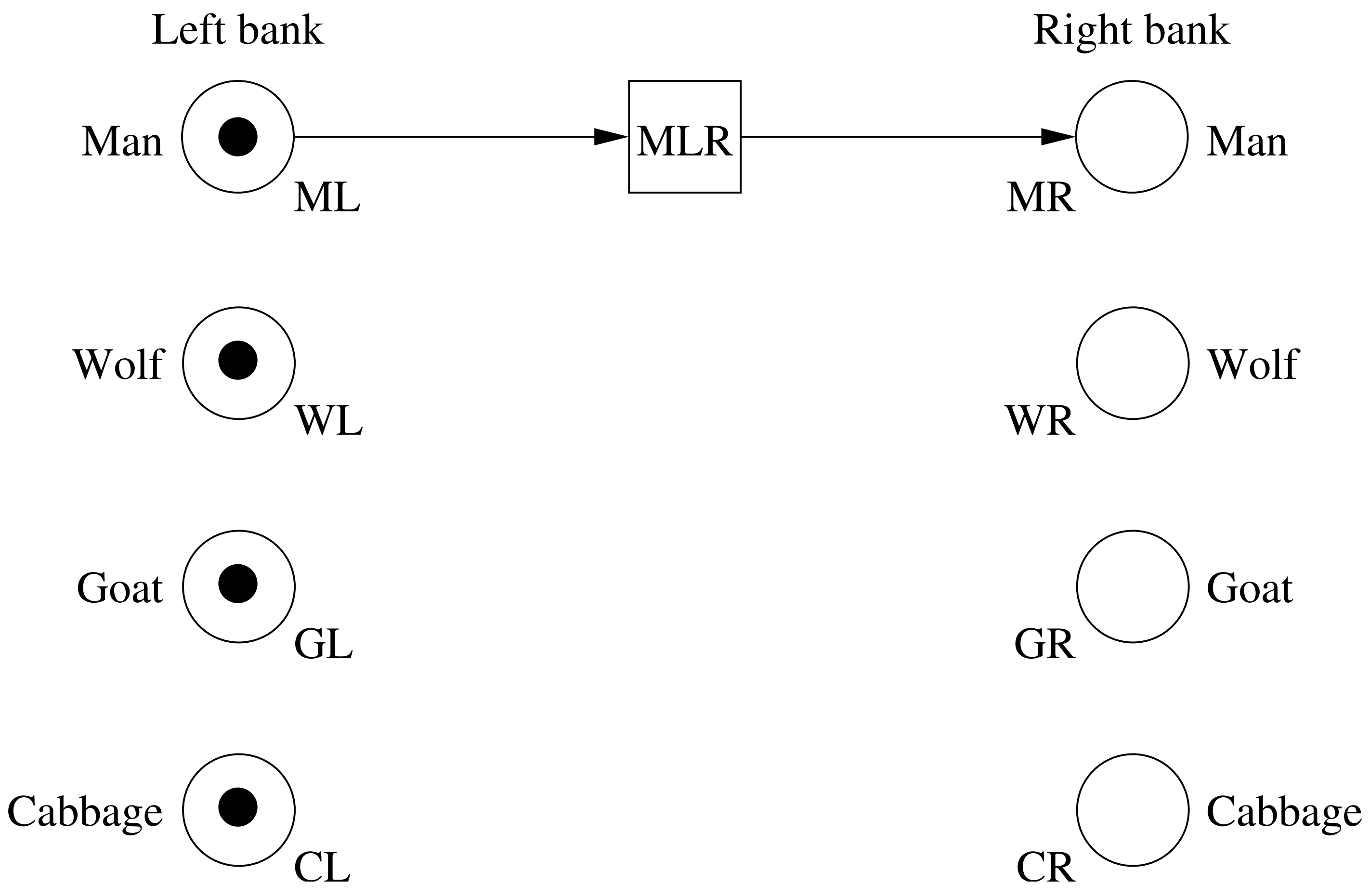
Objects and their states are modeled by places.
Actions are modeled by transitions.

Actually, we can omit the boat, because it is always going to be on the same side as the man.

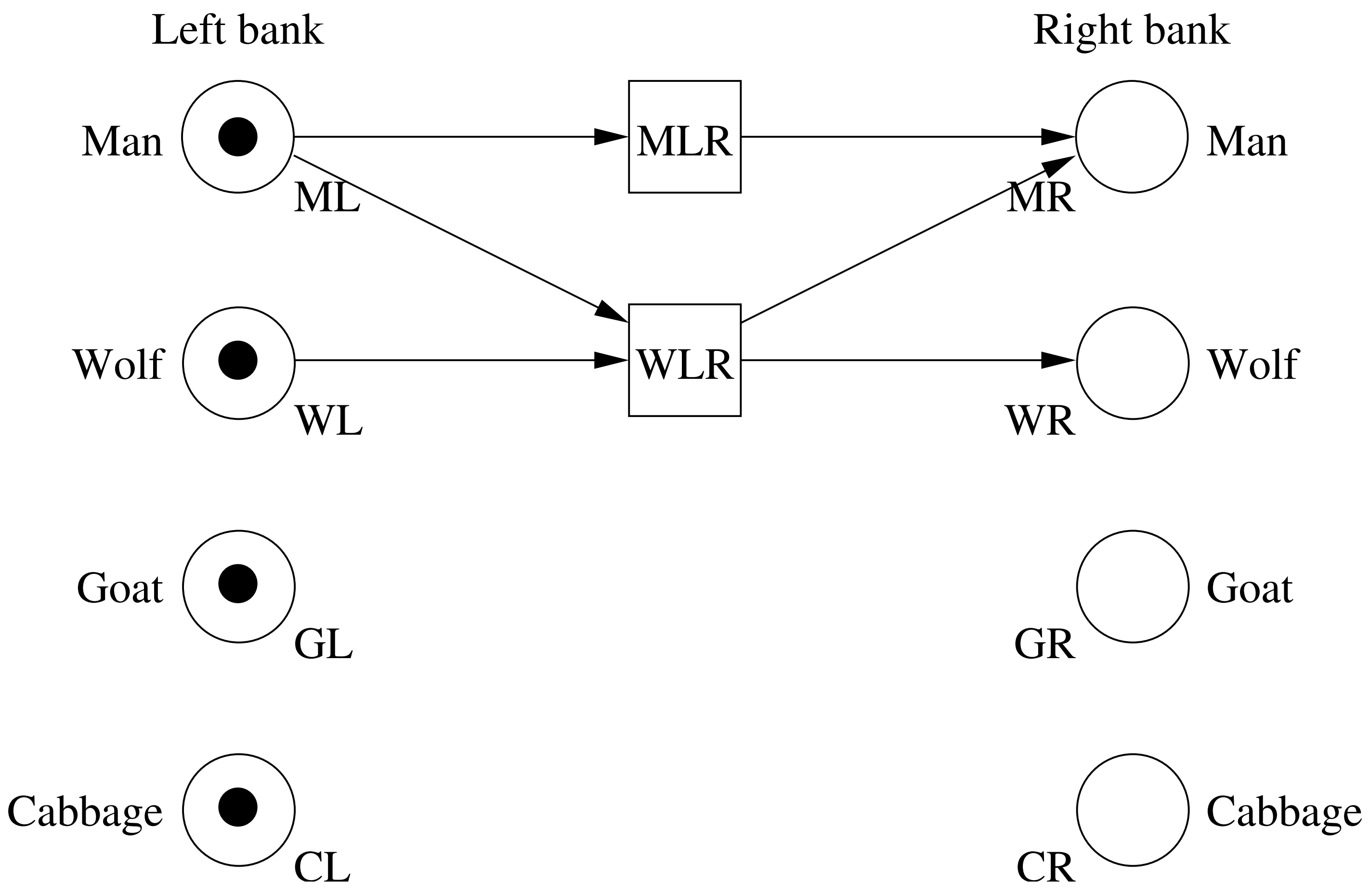
## Example: Places



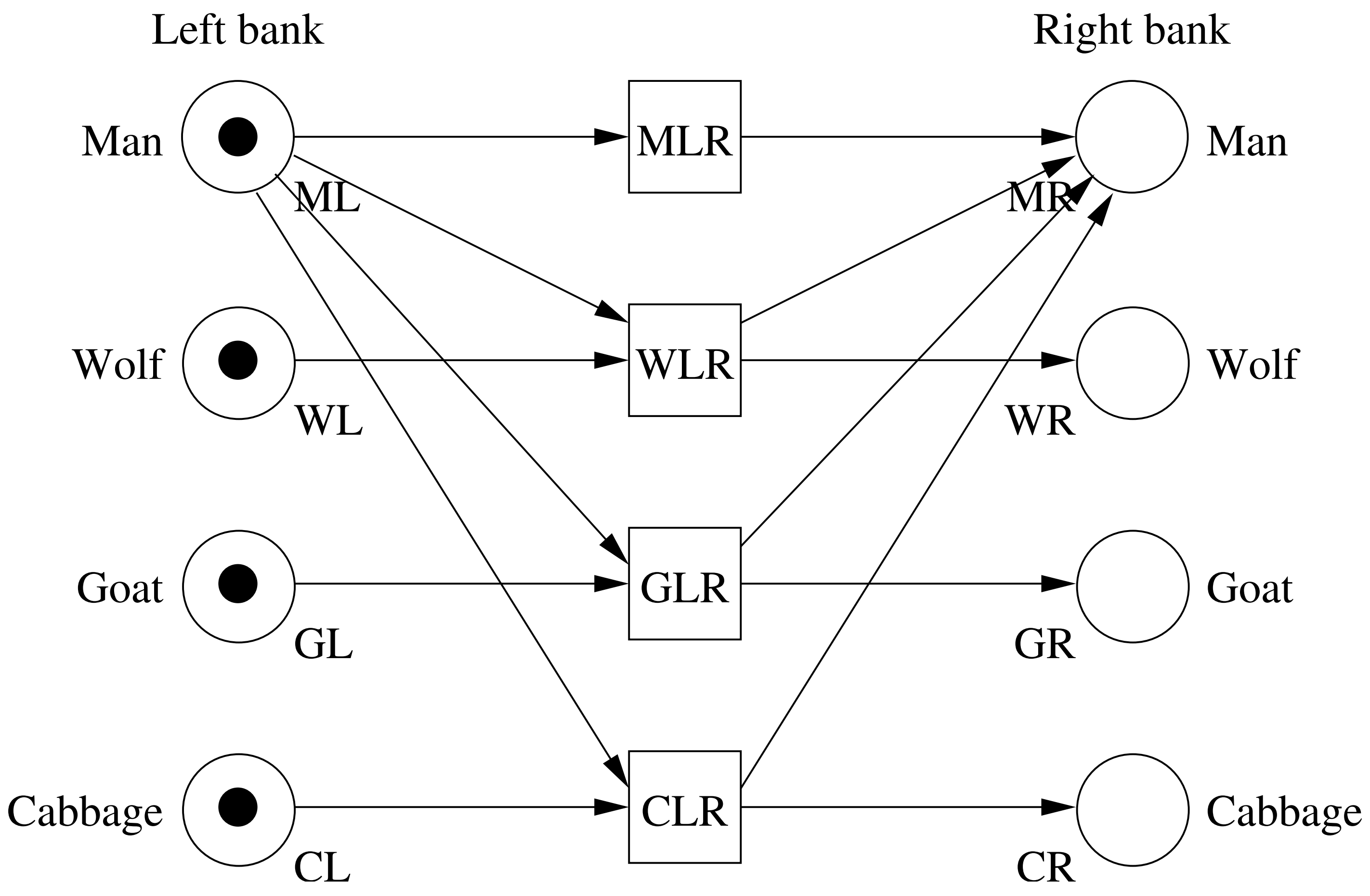
## Crossing the river (left to right)



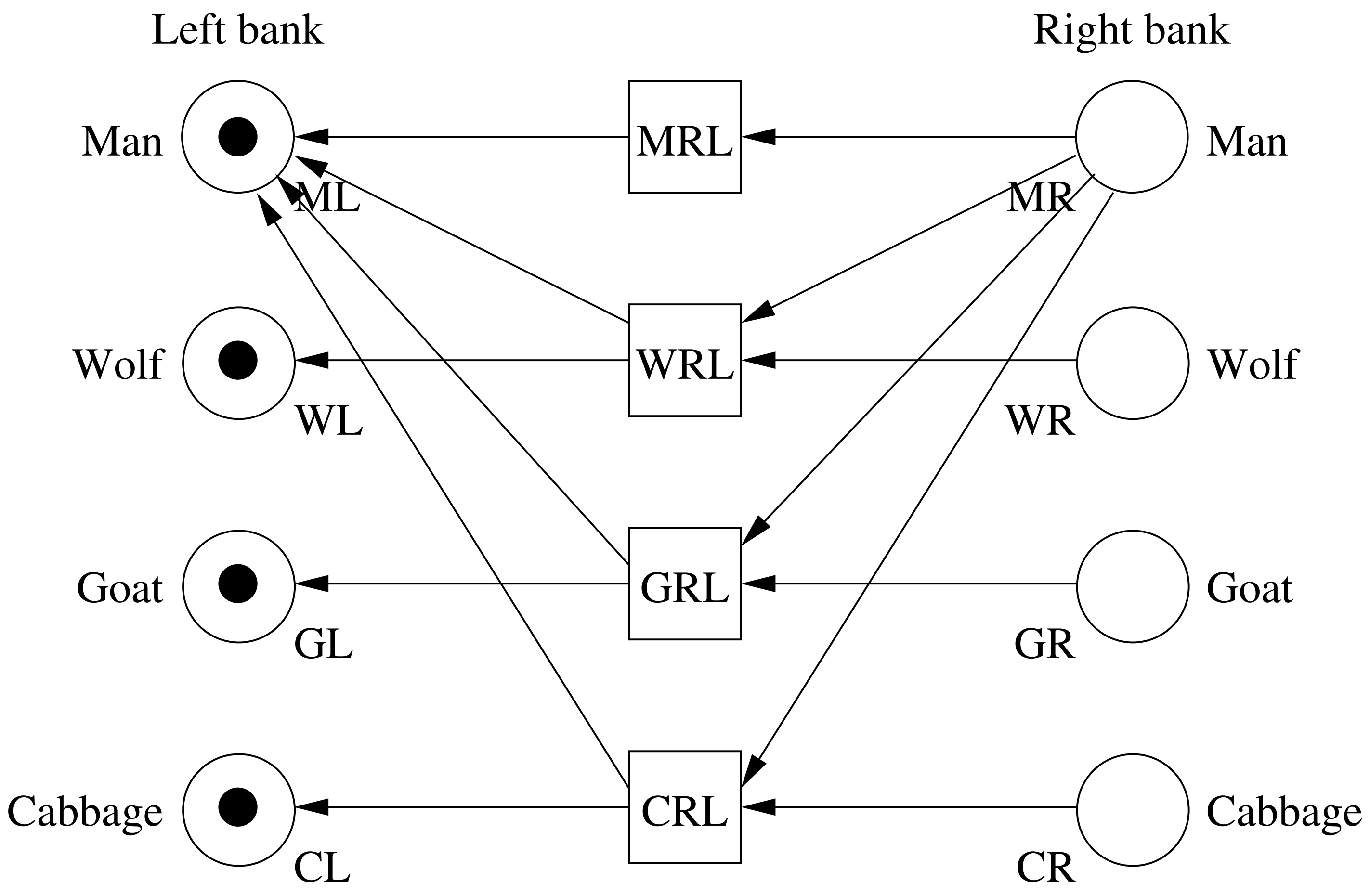
## Crossing the river (left to right)



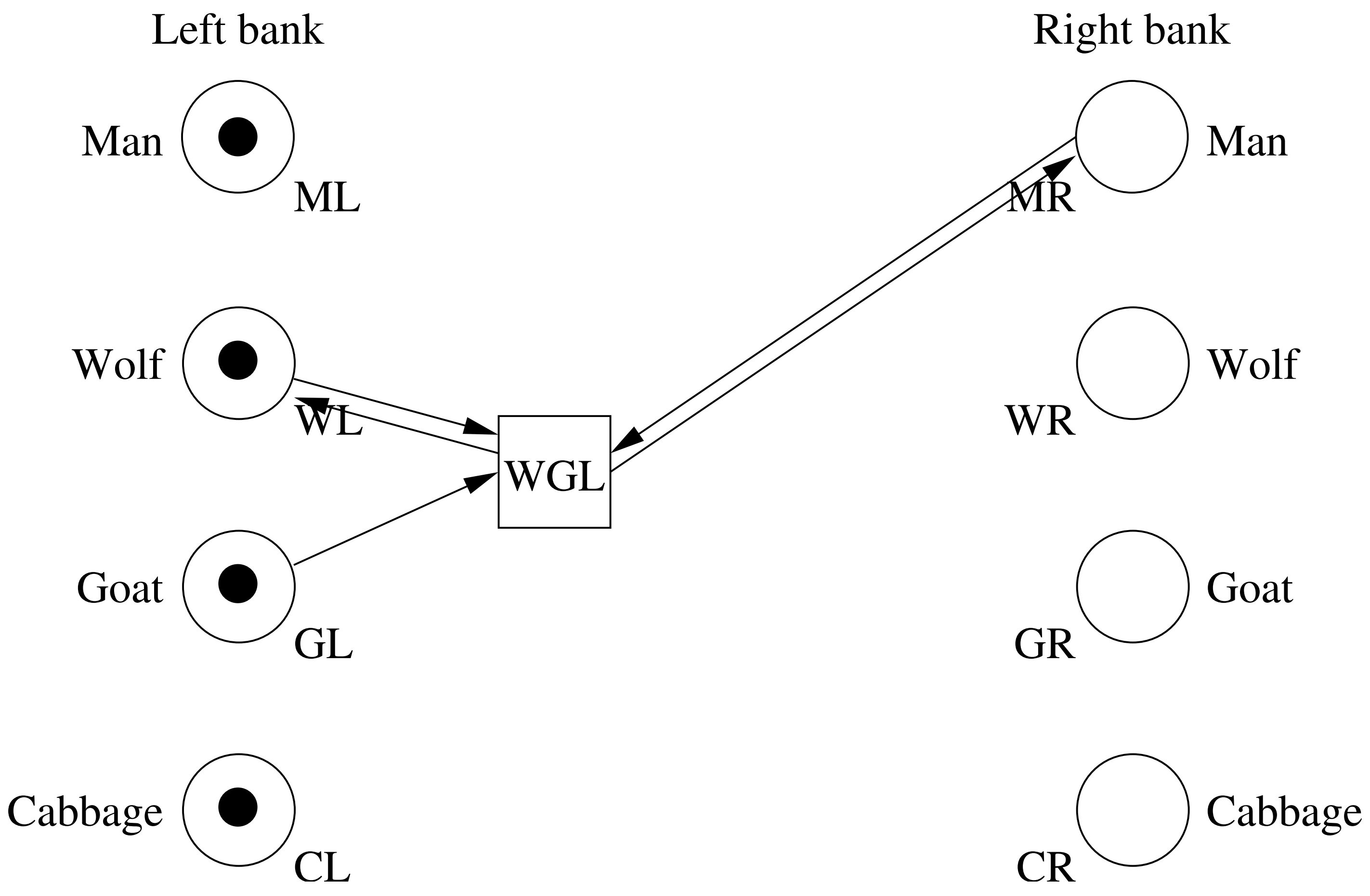
## Crossing the river (left to right)



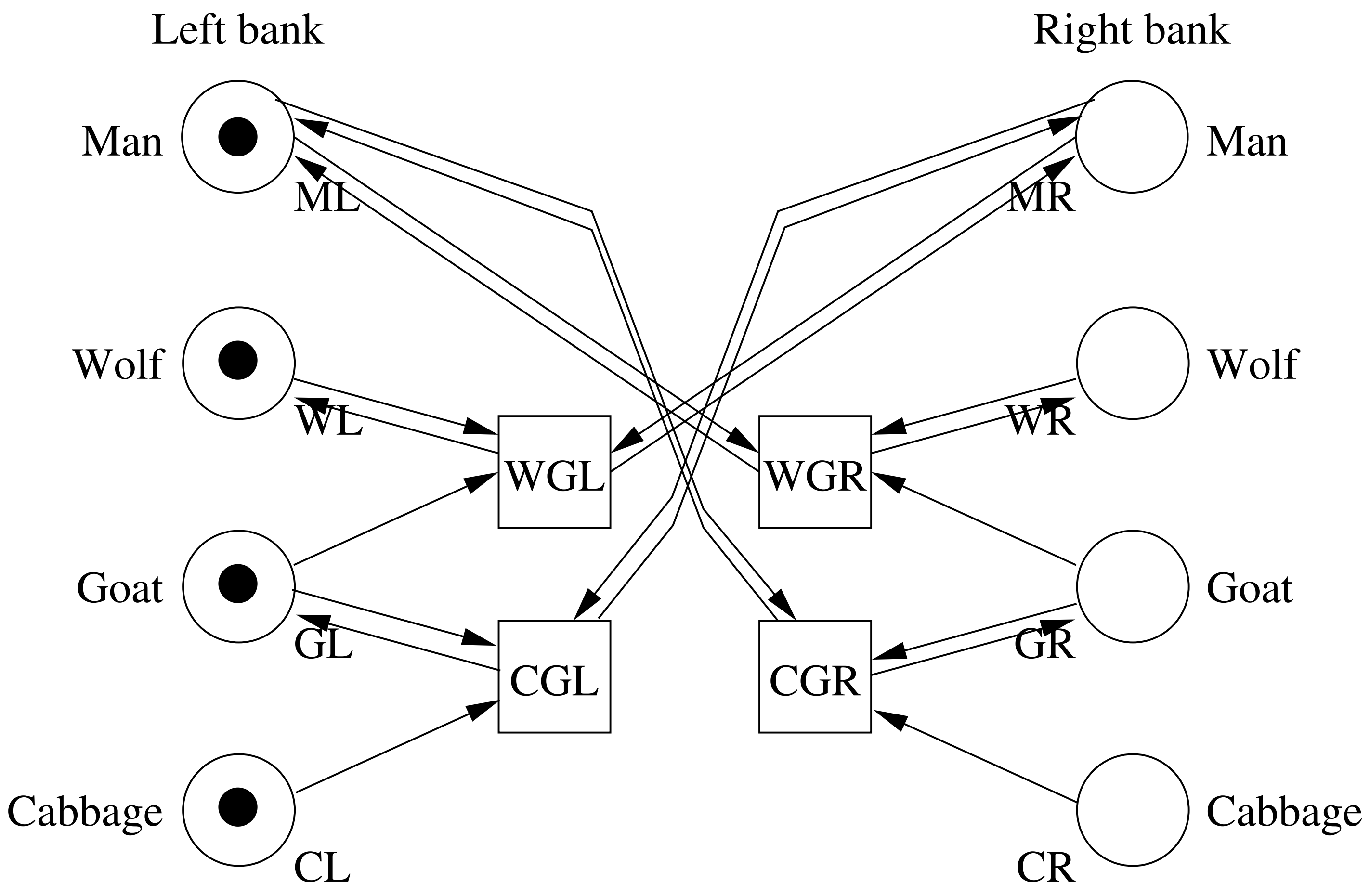
## Crossing the river (right to left)



## Wolf eats goat



# Wolf eats goat, goat eats cabbage



## Example: Specification

To solve the problem using the Petri net, we need to translate the questions *“Can the man bring everyone across the river without endangering the goat or the cabbage? And if so, how?”* into properties of the Petri net.

## Example: Specification

To solve the problem using the Petri net, we need to translate the questions *“Can the man bring everyone across the river without endangering the goat or the cabbage? And if so, how?”* into properties of the Petri net.

“Can the man bring everyone across the river?”

$\Rightarrow$ Is the marking $\{MR, WR, GR, CR\}$ reachable from $\{ML, WL, GL, CL\}$?

## Example: Specification

To solve the problem using the Petri net, we need to translate the questions *“Can the man bring everyone across the river without endangering the goat or the cabbage? And if so, how?”* into properties of the Petri net.

“Can the man bring everyone across the river?”

$\Rightarrow$ Is the marking $\{MR, WR, GR, CR\}$ reachable from $\{ML, WL, GL, CL\}$?

“$\ldots$without endangering the goat or the cabbage?”

$\Rightarrow$ We need to avoid states in which one of the eating transitions is enabled.

## Example: Specification

To solve the problem using the Petri net, we need to translate the questions *"Can the man bring everyone across the river without endangering the goat or the cabbage? And if so, how?"* into properties of the Petri net.

"Can the man bring everyone across the river?"

$\Rightarrow$ Is the marking $\{MR, WR, GR, CR\}$ reachable from $\{ML, WL, GL, CL\}$?

"...without endangering the goat or the cabbage?"

$\Rightarrow$ We need to avoid states in which one of the eating transitions is enabled.

"How?"

$\Rightarrow$ Give a path that leads from one marking to the other. (Optionally: Find a shortest path.)

## Result

Constructing the reachability graph yields a graph with (at most) 36 nodes.

The marking $\{MR, WR, GR, CR\}$ *is reachable* without enabling an “eating” transition!

The transitions fired along a shortest path (there are two) are:

| | |
|---|---|
| *GLR* | (man and goat cross the river), |
| *MRL* | (man goes back alone), |
| *WLR* | (man and wolf cross the river), |
| *GRL* | (man and goat go back), |
| *CLR* | (man and cabbage cross the river), |
| *MRL* | (man goes back alone), |
| *GLR* | (man and goat cross the river). |

## Properties of Place/Transition nets

There are several typical questions one can ask about a Place/Transition net and its behavior. Some of these will be introduced on the next couple of slides. They concern:

- deadlock freedom
- concurrency (conflict, independence)

## Deadlocks

We have already mentioned deadlocks informally in previous discussions. Let us assign a formal meaning to the term.

A marking $M$ of a Place/Transition net $N = \langle P, T, F, W, M_0 \rangle$ is called a deadlock if and only if no transition $t \in T$ is enabled in $M$ (i.e., $M \xrightarrow{t}$ does *not* hold for any $t \in T$).

A net $N$ is said to have a deadlock if one of its reachable markings (i.e. an element of $reach(N)$) is a deadlock, or deadlock-free otherwise.

The existence of deadlocks in a net is quite often a sign of a problem with the net model, e.g.

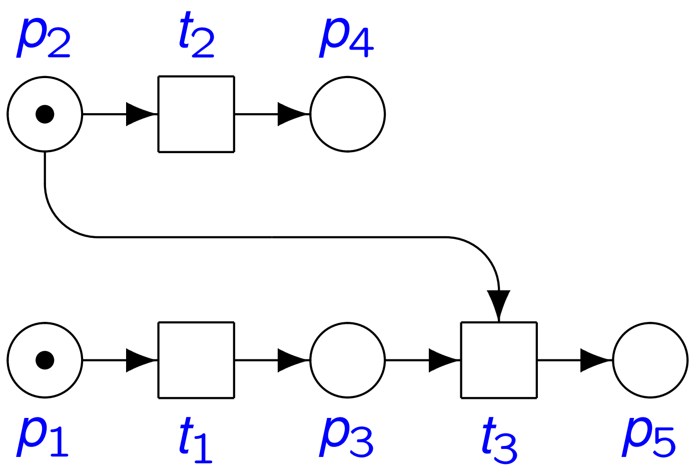
- processes waiting for resources that are never released, or
- processes wrongly waiting to receive messages from other processes.

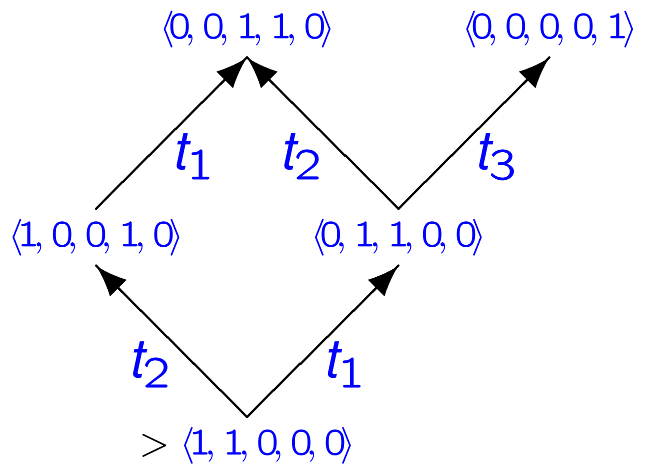
For instance, the deadlock in the Dining Philosophers example means that the philosophers may starve (presumably an undesired result).

However, if a Petri net is used to model a system that is designed to terminate, then a deadlock may also occur in a desired end state.

## Concurrency: Running Example

Petri nets are useful to argue about concurrent systems. We use the following example to illustrate some concepts related to this:




$\langle 0,0,1,1,0 \rangle$
$\langle 0,0,0,0,1 \rangle$
$t_1$
$t_2$
$t_3$
$\langle 1,0,0,1,0 \rangle$
$\langle 0,1,1,0,0 \rangle$
$t_2$
$t_1$
$> \langle 1,1,0,0,0 \rangle$

Note: From now on we assume that domain of the weight function $W(x, y)$ has been extended to $P \times T \cup T \times P$ (all pairs consisting of a place and a transition, instead of only those which appear as arcs in the flow relation) in the obvious way: if $\langle x, y \rangle \notin F$, then $W(x, y) := 0$.

## Definition of Concurrency

Let $N = \langle P, T, F, W, M_0 \rangle$ be a net. A set of transitions $\{t_1, t_2, \ldots, t_n\} \subseteq T$ is said to be concurrent (or: concurrently enabled) in a marking $M$ of $N$, iff

$$\forall p \in P\colon\ M(p) \geq W(p, t_1) + W(p, t_2) + \cdots + W(p, t_n).$$

Example: In the previous slide, $t_1$ and $t_2$ are concurrent in the initial marking:

- they are both enabled in the initial marking, and
- they do not disable each other, i.e, firing $t_1$ leaves $t_2$ enabled and vice versa.

## Definition of Conflict

A conflict occurs in a marking $M$ when two transitions $t$ and $t'$ are both enabled ($M \xrightarrow{t}$ and $M \xrightarrow{t'}$), but the set $\{t, t'\}$ is not concurrent.

In other words, the transitions $t$ and $t'$ are in conflict in marking $M$, if they are both enabled and

$$\exists p \in P\colon\ M(p) < W(p,t) + W(p,t').$$

Thus, firing one transition may (but need not!) disable the other.

Example: Consider the marking $M := \langle 0, 1, 1, 0, 0 \rangle$.
In $M$, the transitions $t_2$ and $t_3$ are in conflict.

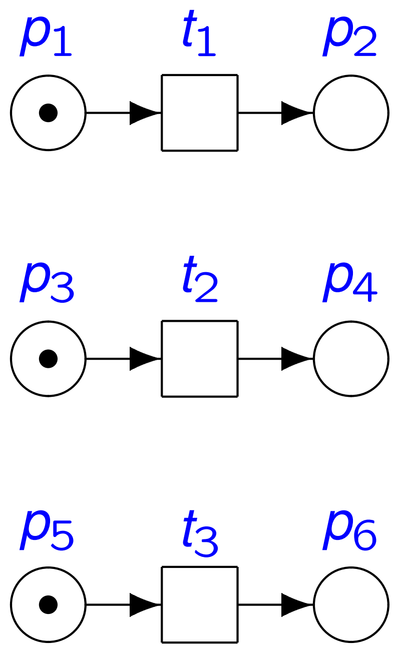
## Concurrency and the Reachability graph

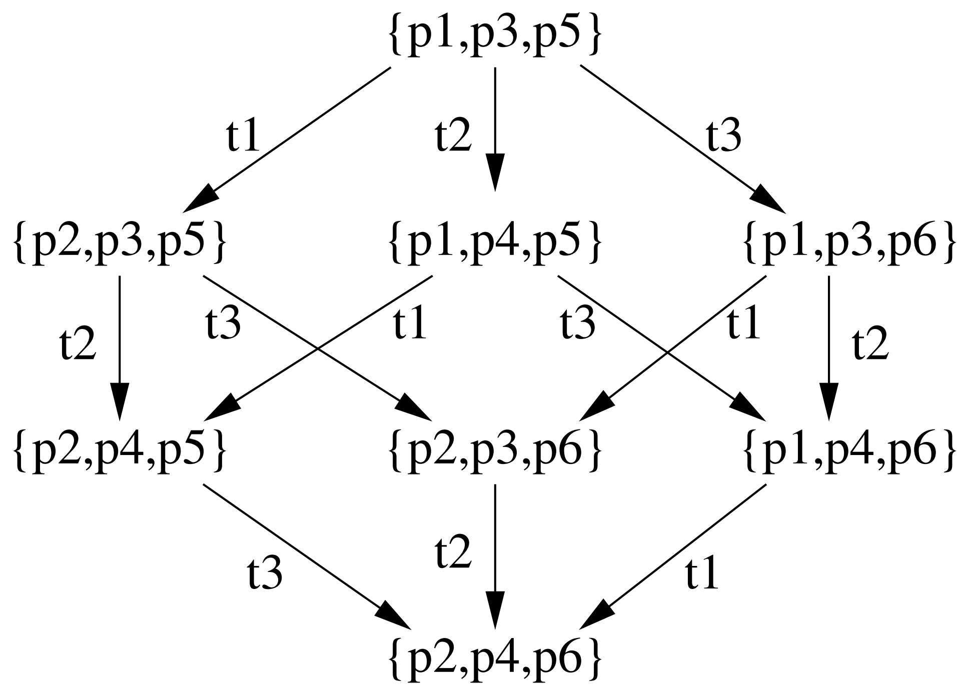
The concurrency of $t_1$ and $t_2$ manifests itself as a *diamond* structure in the reachability graph. This diamond consists of the part of the reachability graph reachable by firing the concurrent transitions (in this case $t_1$ and $t_2$) in any order.

If we have a marking $M$ with $n$ concurrently enabled transitions, then the set $reach(M)$ contains at least $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$ different paths (without repeated transitions), and up to $2^n$ reachable markings. (Concurrent transitions can be fired in any order.)

A diamond in the reachability graph does not, however, mean that concurrency is present (remember that conflicting transitions need not disable each other)!

# Example: Concurrent Transitions





The set $\{t_1, t_2, t_3\}$ is concurrent, there are $2^3 = 8$ reachable markings, and they can be visualized as a three-dimensional cube. A similar net with $n$ transitions will have $2^n$ reachable markings, and can be visualized as a $n$-dimensional cube.

## State Explosion Problem

State explosion problem is the problem of having very many possible reachable states for even small models of systems. It makes their analysis hard.

- We have now seen instances of nets with exponentially large state spaces in the net size even though each place only contains at most one token. (If we allow multiple tokens per place, the blowup is even worse.)
- Sources of large state spaces can be:
  - Concurrency (as demonstrated)
  - Variables with large value ranges (in a modeling language which allows variables)
  - Large number of variables (circuits with lots of state variables)

## Counteracting state space explosion

Techniques for counteracting state space explosion can be characterised as follows:

Abstraction: leave out unimportant information.
Example: boat omitted in man/wolf/goat/cabbage puzzle

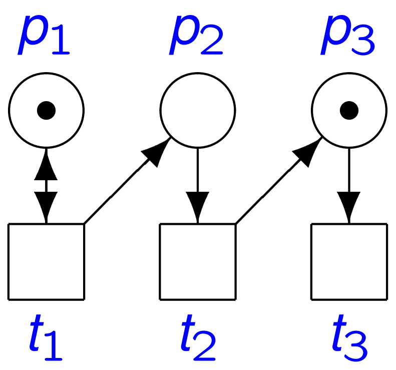
Compression: operate on *sets* of states (or markings), use efficient data structures to store them. Example: BDDs

Reduction: avoid exploring multiple ‘equivalent’ executions.
Example: exploit independence

Time permitting, we will treat some of these techniques in more detail during the course.

## Coverability Graph Method

As we have mentioned before, the reachability graph of P/T-net can be infinite (in which case the algorithm for computing the reachability graph will not terminate). For example, consider the following net.



We will show a method to find out whether the reachability graph of a P/T-net is infinite or not. This can be done by using the coverability graph method.

## $\omega$-Markings

First we introduce a new symbol $\omega$ to represent “arbitrarily many” tokens.

We extend the arithmetic on natural numbers with $\omega$ as follows. For all $n \in \mathbb{N}$:

$n + \omega = \omega + n = \omega$,

$\omega + \omega = \omega$,

$\omega - n = \omega$,

$0 \cdot \omega = 0$, $\omega \cdot \omega = \omega$,

$n \geq 1 \Rightarrow n \cdot \omega = \omega \cdot n = \omega$,

$n \leq \omega$, and $\omega \leq \omega$.

Note: $\omega - \omega$ remains undefined, but we will not need it.

We will extend the notion of markings to $\omega$-markings. In an $\omega$-marking, each place $p$ will either have $n \in \mathbb{N}$ tokens, or $\omega$ tokens (infinitely many).

## Firing Rule and $\omega$-markings

The firing condition and firing rule (reproduced below) neatly extend to $\omega$-markings with the extended arithmetic rules:

> Firing condition:
> Transition $t \in T$ is $M$-enabled, written $M \xrightarrow{t}$, iff $\forall p \in {}^{\bullet}t : M(p) \geq W(p,t)$.
>
> Firing rule:
> An $M$-enabled transition $t$ may fire, producing the successor marking $M'$, where
> $$\forall p \in P : M'(p) = M(p) - \bar{W}(p,t) + \bar{W}(t,p).$$

Basically, if a transition has a place with $\omega$ tokens in its preset, that place is considered to have sufficiently many tokens for the transition to fire, regardless of the arc weight.

If a place contains an $\omega$-marking, then firing any transition connected with an arc to that place will not change its marking.

## Definition of Covering

An $\omega$-marking $M'$ covers an $\omega$-marking $M$, denoted $M \leq M'$, iff

$$\forall p \in P\colon\ M(p) \leq M'(p).$$

An $\omega$-marking $M'$ strictly covers an $\omega$-marking $M$, denoted $M < M'$, iff

$$M \leq M' \quad \text{and} \quad M' \neq M.$$

# Coverability and Transition Sequences (1/2)

Observation: Let $M$ and $M'$ be two markings such that $M \leq M'$.
Then for all transitions $t$, the following holds (monotony):

$$\text{If } M \xrightarrow{t} \text{ then } M' \xrightarrow{t}.$$

In other words, if $M'$ has at least as many tokens as $M$ has (on each place), then $M'$ enables at least the same transitions as $M$ does.

This observation can be extended to *sequences* of transitions:
Define $M \xrightarrow{t_1 t_2 \dots t_n} M'$ to denote:

$$\exists M_1, M_2, \dots, M_n : M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \cdots \xrightarrow{t_n} M_n = M'.$$

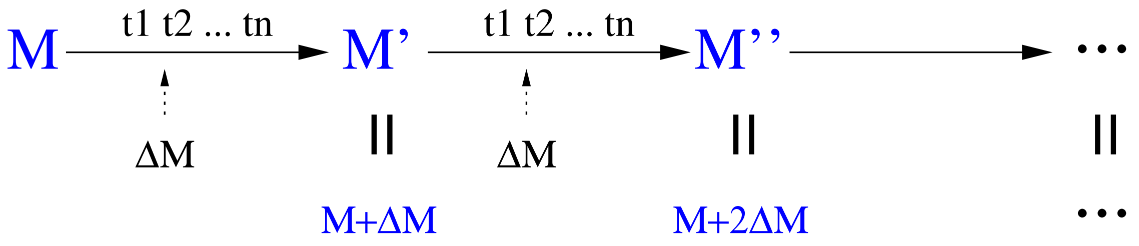
Now, if $M \xrightarrow{t_1 t_2 \dots t_n}$ and $M \leq M'$, then $M' \xrightarrow{t_1 t_2 \dots t_n}$.

## Coverability and Transition Sequences (2/2)

Assume that $M' \in \mathit{reach}(M)$ (with $M < M'$). Then clearly there is some sequence of transitions $t_1 t_2 \ldots t_n$ such that $M \xrightarrow{t_1 t_2 \ldots t_n} M'$. Thus, there is a marking $M''$ with $M' \xrightarrow{t_1 t_2 \ldots t_n} M''$.

Let $\Delta M := M' - M$ (place-wise difference). Because $M < M'$, the values of $\Delta M$ are non-negative and at least one value is non-zero.

Clearly, $M'' = M' + \Delta M = M + 2\Delta M$.



By firing the transition sequence $t_1 t_2 \ldots t_n$ repeatedly we can “pump” an arbitrary number of tokens to all the places having a non-zero marking in $\triangle M$.

The basic idea for constructing the coverability graph is now to replace the marking $M'$ with a marking where all the places with non-zero tokens in $\triangle M$ are replaced by $\omega$.

## Coverability Graph Algorithm (1/2)

```
COVERABILITY-GRAPH($\langle P, T, F, W, M_0 \rangle$)
 1  $\langle V, E, v_0 \rangle := \langle \{M_0\}, \emptyset, M_0 \rangle$;
 2  $\mathit{Work}$ : set $:= \{M_0\}$;
 3  while $\mathit{Work} \neq \emptyset$
 4  do select $M$ from $\mathit{Work}$;
 5      $\mathit{Work} := \mathit{Work} \setminus \{M\}$;
 6      for $t \in$ enabled$(M)$
 7      do $M' :=$ fire$(M, t)$;
 8          $M' :=$ AddOmegas$(M, t, M', V, E)$;
 9          if $M' \notin V$
10            then $V := V \cup \{M'\}$
11                 $\mathit{Work} := \mathit{Work} \cup \{M'\}$;
12          $E := E \cup \{\langle M, t, M' \rangle\}$;
13  return $\langle V, E, v_0 \rangle$;
```

The coverability graph algorithm is almost exactly the same as the reachability graph algorithm, with the addition of the call to subroutine AddOmegas$(M, t, M', V, E)$, where all the details w.r.t. coverability graphs are contained. As for the implementation of $\mathit{Work}$, the same comments as for the reachability graph apply.

## Coverability Graph Algorithm (2/2)

The following notations are used in the AddOmegas subroutine:

- $M'' \rightarrow_E M$ iff $\langle M'', t, M\rangle \in E$ for some $t \in T$.
- $M'' \rightarrow_{E^*} M$ iff
  $\exists n \geq 0\colon \exists M_0, M_1, \ldots, M_n\colon\ M'' = M_0 \rightarrow_E M_1 \rightarrow_E M_2 \rightarrow_E \cdots \rightarrow_E M_n = M$.

```
AddOmegas($M, t, M', V, E$)
1  for $M'' \in V$
2  do if $M'' < M'$ and $M'' \rightarrow_{E^*} M$
3        then $M' := M' + ((M' - M'') \cdot \omega)$;
4  return $M'$;
```

Line 3 causes all places whose marking in $M'$ is strictly larger than in the “parent” $M''$ to contain $\omega$, while markings of other places remain unchanged.

## Remarks on the Coverability Graph Algorithm

If the reachability graph is finite, the algorithm $\mathsf{AddOmegas}(M, t, M', V, E)$ will always return $M'$ as its output (i.e., the third parameter).

In this case the coverability graph algorithm will return the reachability graph (but it will run more slowly).

Implementations of the algorithm are bound to be slow because of the **for** loop in $\mathsf{AddOmegas}$, which has to traverse the potentially large size of the graph.

The result of the algorithm is not unique, e.g. it depends on the implementation of $Work$ and on the exact order of fired transitions on line 5 of the main routine.

# Lecture 4:
# Properties of P/T Nets, Part II

## Programme for this Lecture

Coverability graphs (continued)

Comparison between reachability and coverability analysis

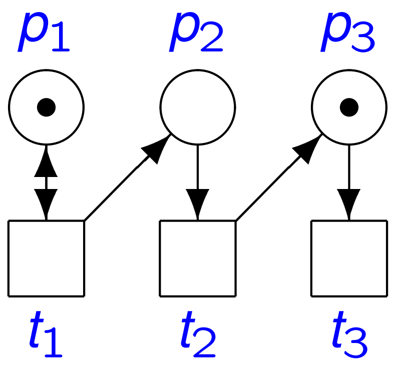
- Which properties can we analyse so far?

Variants of the P/T model

- Nets with capacities

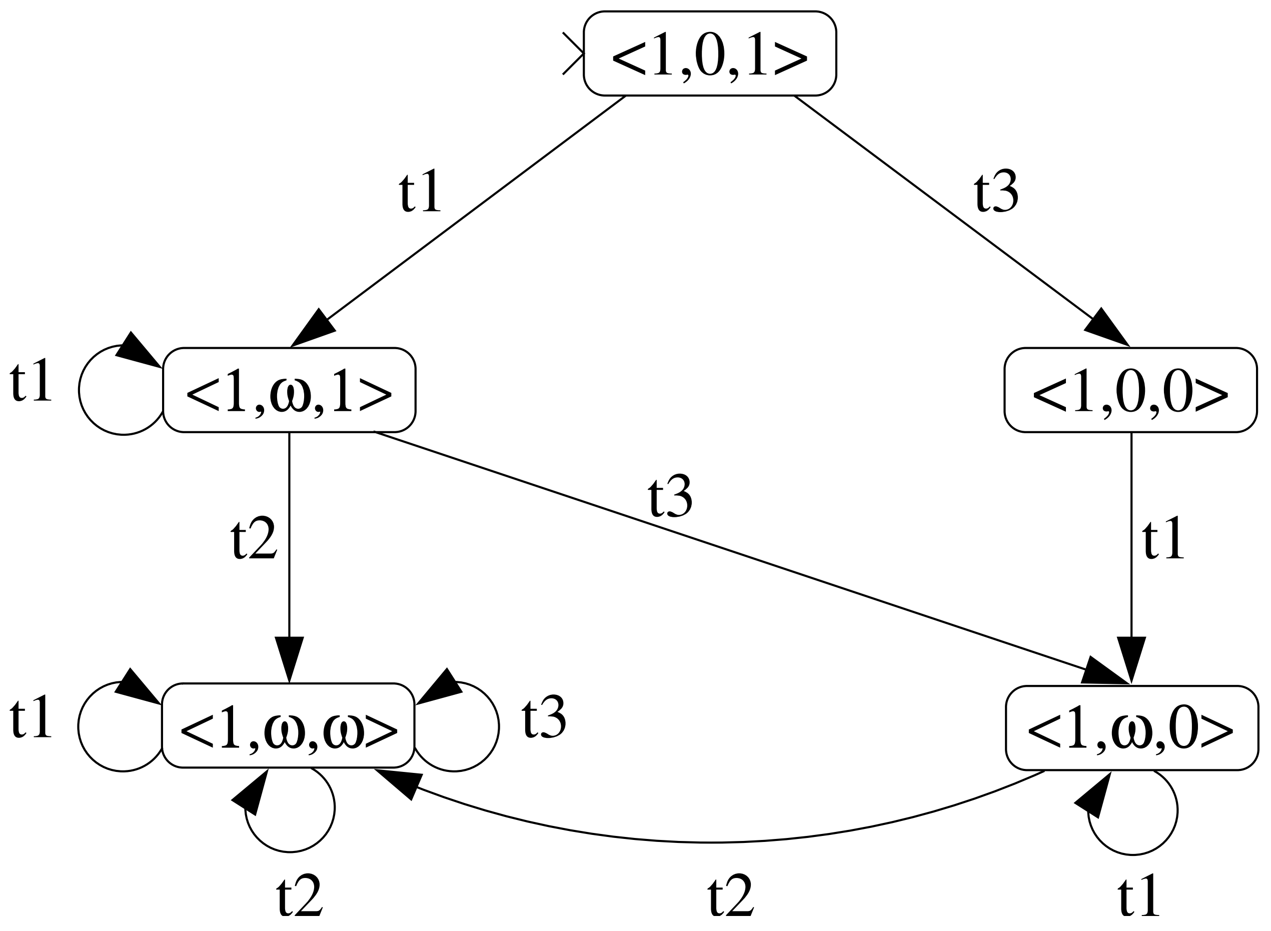
## Example 1: Coverability Graph

Recall the P/T-net example given in the previous lecture:



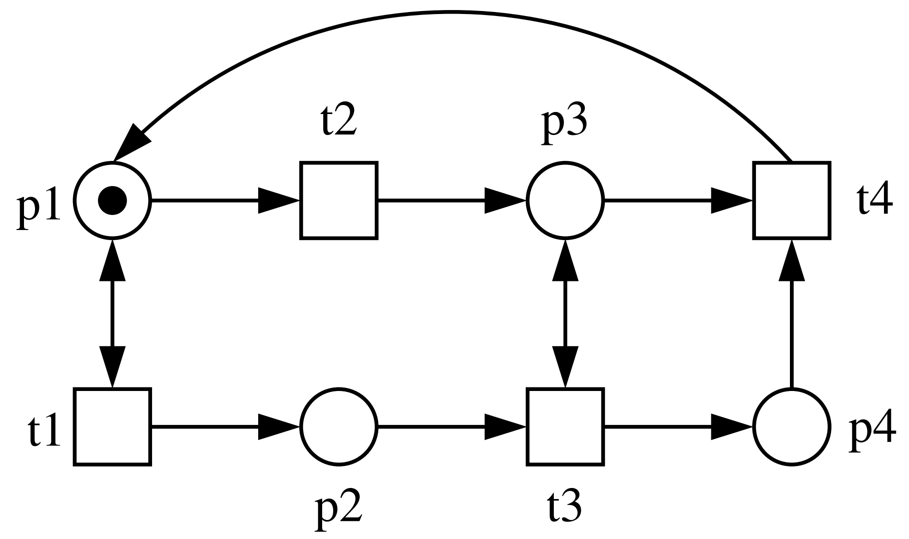
We will now compute the coverability graph for it using the algorithm presented in the previous lecture.

# Example 1: Coverability Graph

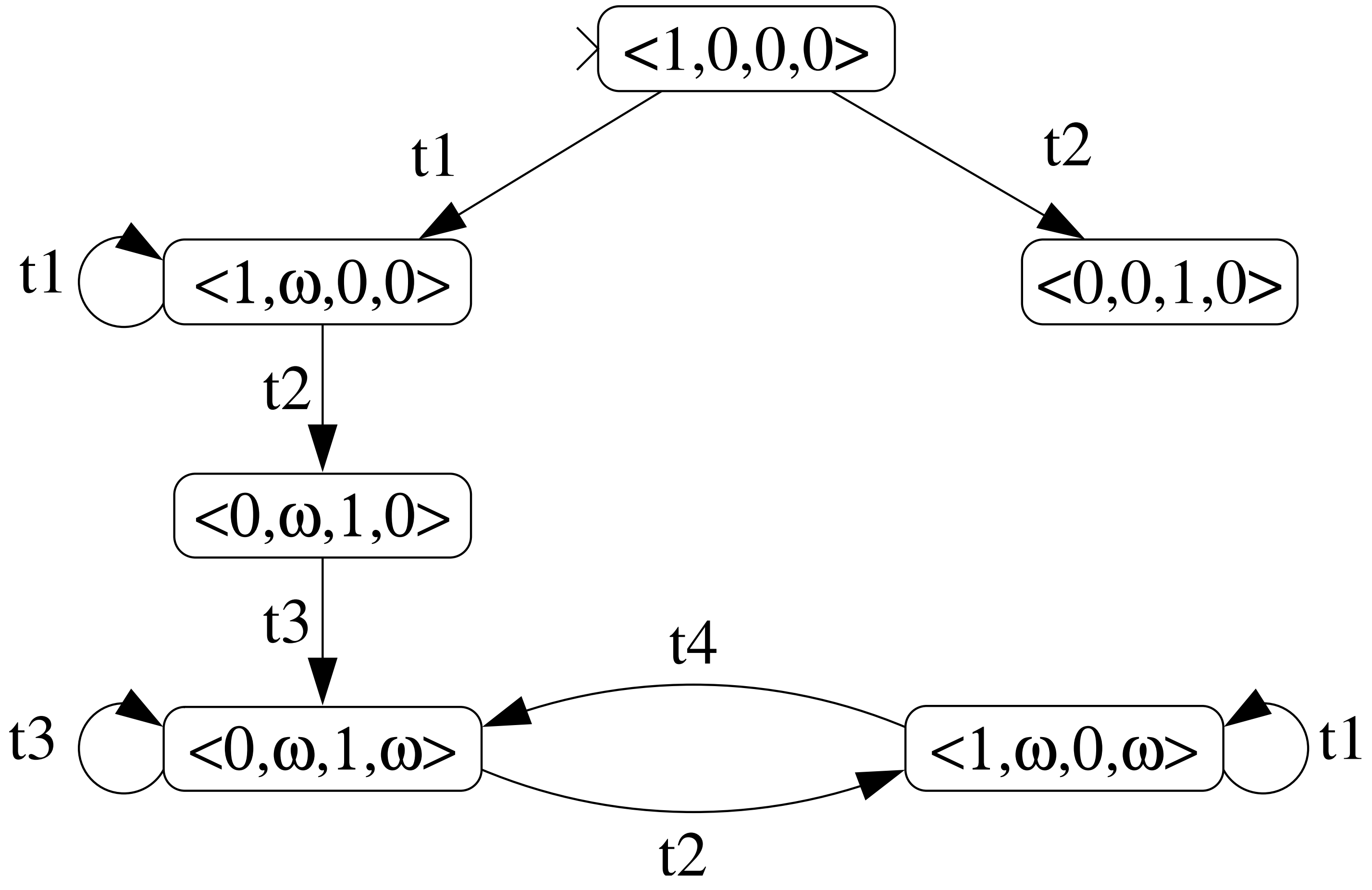


## Example 2

Consider the following P/T-net. We will now compute a coverability graph for it.



## Example 2: Coverability graph



# Reachability and coverability graphs: Comparison (1)

Let $N = \langle P, T, F, W, M_0 \rangle$ be a net.

The reachability graph has the following fundamental property:

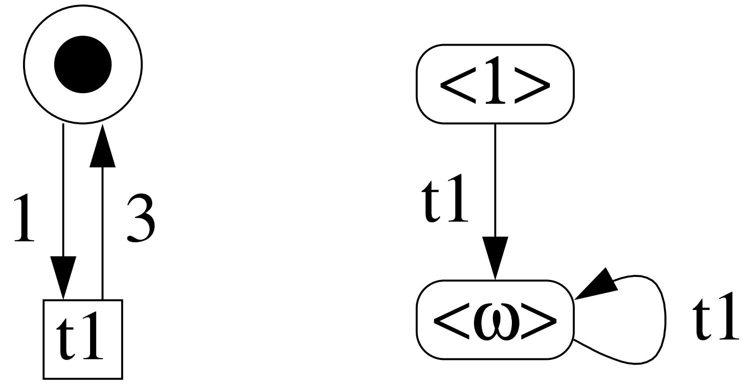
> A marking $M$ of $N$ is reachable *if and only if* $M$ is a vertex of the reachability graph of $N$.

The coverability graph has the following fundamental property:

> If a marking $M$ of $N$ is reachable, then $M$ is covered by some vertex of the coverability graph of $N$.

Notice that the first property is an equivalence, the second one an implication!

More specifically, the reverse implication *does not* hold: A marking that is covered by some vertex of the coverability graph is not necessarily reachable, as shown by the following example:



In the net, only markings with an odd number of tokens are reachable, but markings with an even number of tokens are also covered.

## Reachability and coverability graphs: Comparison (2)

The reachability graph captures exact information about the reachable markings (but its computation may not terminate).

The coverability graph computes an overapproximation
(but remains exact as long as the number of markings is finite).

## Summary: Which properties can we check so far?

Reachability: Given some marking $M$ and a net $N$, is $M$ reachable in $N$?
More generally: Given a set of markings $\mathcal{M}$, is some marking of $\mathcal{M}$ reachable?

Application: This is often used to check whether some ‘bad’ state can occur (classical example: violation of mutual exclusion property) if $\mathcal{M}$ is taken to be the set of ‘error’ states. Sometimes (as in the man/wolf/etc example), this analysis can check for the existence of a solution to some problem.

Using the reachability graph: Exact answer is obtained.

Using the coverability graph: Approximate answer. When looking for ‘bad’ states, this analysis is safe in the sense that bad states will not be missed, but the graph may indicate ‘spurious’ errors.

## Summary (cont'd)

Finding paths: Given a reachable marking $M$, find a firing sequence that leads from $M_0$ to $M$.

Application: Used to supplement reachability queries. If $M$ represents an error state, the firing sequence can be useful for debugging. When solving puzzles, the path represents actions leading to the solution.

Using the reachability graph: Find a path from $M_0$ to $M$ in the graph, obtain sequence from edge labels.

Using the coverability graph: Not so suitable – edges may represent 'shortcuts' (unspecified repetitions of some loop).

## Summary (cont'd)

Enabledness: Given some transition $t$, is there a reachable marking in which $t$ is enabled?
(Sometimes, $t$ is called dead if the answer is no. Actually, this is a special case of reachability.)

Application: Check whether some 'bad' action is possible. Also, is some desirable action is never enabled, a 'no' answer is an indication of some problem with the model.
In some Petri-net tools, checking for enabledness is easier to specify than checking for reachability. In that case, reachability queries can be framed as enabledness queries by adding 'artificial' transitions that can fire iff a given marking is reachable.

Using the reachability graph: Check whether there is an edge labeled with $t$.

Using the coverability graph: ?

## Summary (cont'd)

Conflicts, concurrency, independence: Given two transitions $t_1$ and $t_2$, are they independent? Are they in conflict in some marking?

Application: Analyse the dependency between two actions? Can an implementation of the model distribute the actions onto two different machines?

Using the reachability graph: Check whether the markings fulfil the respective definitions.

Using the coverability graph: Unsuitable, as the graph conflates different markings.

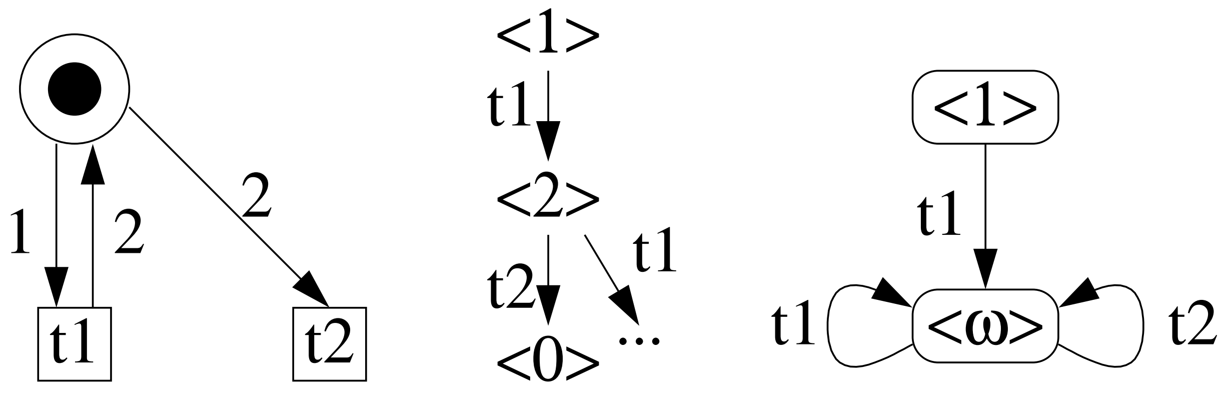
# Summary (cont'd)

Deadlocks: Given a net *N*, is *N* deadlock-free?

Application: Deadlocks tend to indicate errors, see discussion in Lection 3 (classical example: philosophers may starve).

Using the reachability graph: Check whether there is a vertex without an outgoing edge.

Using the coverability graph: Unsuitable – the graph may miss deadlocks!



## Summary (cont'd)

Boundedness: Given a net $N$, is there a constant $k$ such that $N$ is $k$-safe? Otherwise, which places can assume an unbounded number of tokens?

Application: If tokens represent available resources, unbounded numbers of tokens may indicate some problem (e.g. a resource leak). Also, this property should be checked *before* computing the reachability graph!

Using the reachability graph: Unsuitable, computation may not terminate.

Using the coverability graph:

> A place $p$ can assume an unbounded number of tokens iff the coverability graph contains a vertex $M$ where $M(p) = \omega$.
> Iff no vertex with an $\omega$ exists, then the net is $k$-safe, where $k$ is the largest natural number in a marking of the graph.

## What is missing? (Outlook)

Sometimes, properties mentioned in the summary can be checked even *without constructing the reachability graph* (which can be pretty large, after all).

Methods for doing this are collectively called structural analyses. These will be covered in the next lecture.

So far, we have not learnt how to express (and check) properties like these:

> Marking $M$ can be reached infinitely often.
>
> Whenever transition $t$ occurs, transition $t'$ occurs later.
>
> No marking with some property $x$ occurs before some marking with property $y$ has occurred.

Properties like these can be expressed using temporal logic. They will be covered in lectures 6 and following.

## Variations of P/T nets

In the rest of the lecture, we shall have a look at two extensions of the P/T net model:

- Nets with capacities
- High-level nets (next lecture)

These extensions are equally expressive as P/T nets (in the sense that these extended nets can be replaced by ‘equivalent’ P/T nets).

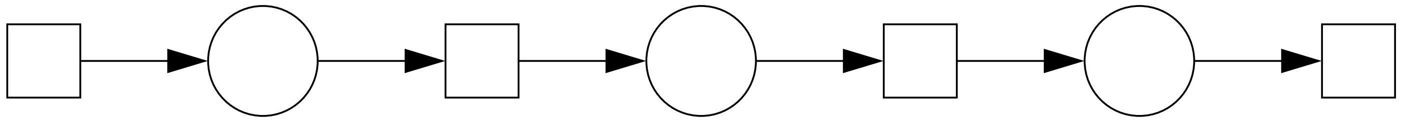
However, they allow easier modeling of some problems.

On the other hand, their formalisms are more complicated.

## Nets with capacities: Motivation

Quite often, we may want to restrict the number of allowable tokens on some places.

For instance, consider the following net, which may be seen as a railway track (tokens represent trains moving along the track):



No section of the track should contain more than one train at a time.

However, the P/T model allows unbounded numbers of tokens. We could in fact fix this by adding extra places and transitions, but that would make the model harder to understand. It is much easier to work with a formalism that makes restrictions on the numbers of tokens explicit.

## Nets with capacities: Definition

For this, we extend the P/T net model with a function that assigns a capacity to each place.

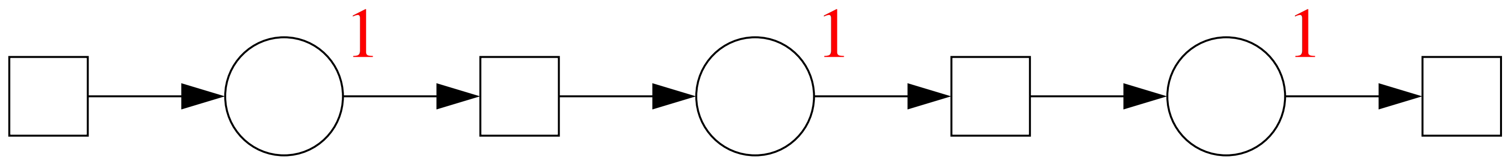
Formally, a tuple $N = \langle P, T, F, W, K, M_0 \rangle$ is called a Place/Transition Net with capacities where

- $P$, $T$, $F$, $W$, $M_0$ are defined as in normal P/T nets;
- $K \colon P \to (\mathbb{N} \cup \{\infty\})$ is a capacity function.

The basic idea is the following: If firing a transition in marking $M$ would lead to a marking that exceeds the capacity of some place, that transition is defined not to be enabled in $M$.

## Nets with capacities: Notation

In our figures, we shall denote capacities by (red) numbers next to the place.
Example:



(We will usually omit capacities with the value $\infty$.)

## Nets with capacities: Firing Rule

The new formalism requires a firing rule that captures the extra restriction:

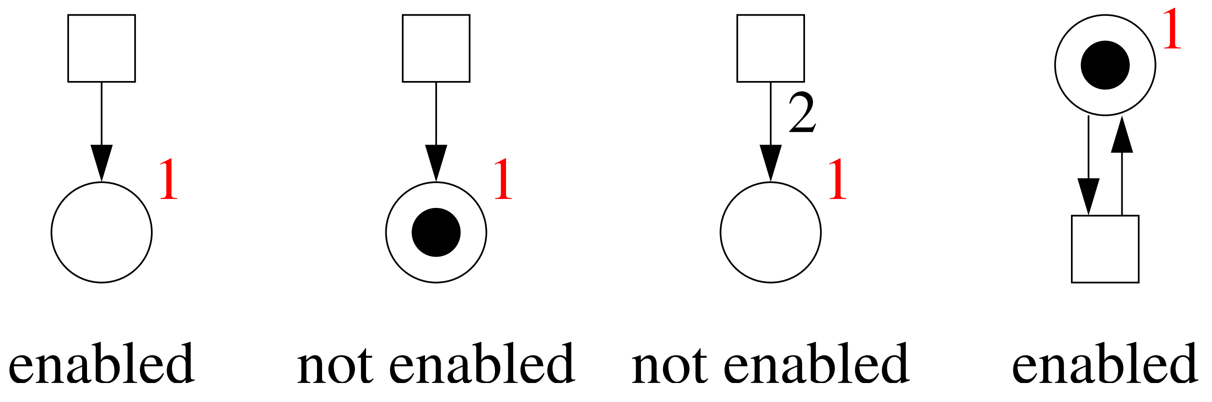
Let $\langle P, T, F, W, K, M_0 \rangle$ be a net and $M$ be a marking.

> Firing condition: (changed)
> Transition $t \in T$ is $M$-enabled, written $M \xrightarrow{t}$, iff $\forall p \in {}^\bullet t : M(p) \geq W(p, t)$ and $\forall p \in t^\bullet : M(p) - W(p, t) + W(t, p) \leq K(p)$.
>
> Firing rule: (unchanged)
> An $M$-enabled transition $t$ may fire, producing the successor marking $M'$, where $\forall p \in P : M'(p) = M(p) - W(p, t) + W(t, p)$.

Clearly, a net where all places have capacity $\infty$ has exactly the same behaviour as a net in the definition without capacities.

## Nets with capacities: Some examples

Here are some examples of transitions that are enabled or disabled according to the new firing rule:



Note: Nets with capacities allow easier modeling in some cases. The following slides show how capacities can be *automatically* eliminated (using an algorithm), so that we obtain a net without capacity restrictions (to which we can then apply all our analysis methods).

## Complement Places: Definition

Let $\langle P, T, F, W, M_0 \rangle$ be a net (without capacities).

Two places $p$ and $p'$ are called a pair of complement places iff there is a constant $k$ such that for all markings in $M \in \mathit{reach}(M_0)$ it holds that $M(p) + M(p') = k$.

Example: In the man/wolf/goat/cabbage example, $ML$ and $MR$ are a pair of complement places (and so are $WL$ and $WR$ etc).

Intuitively, in each marking $p$ contains those of the $k$ tokens that are not contained in $p'$.

# Complement Construction

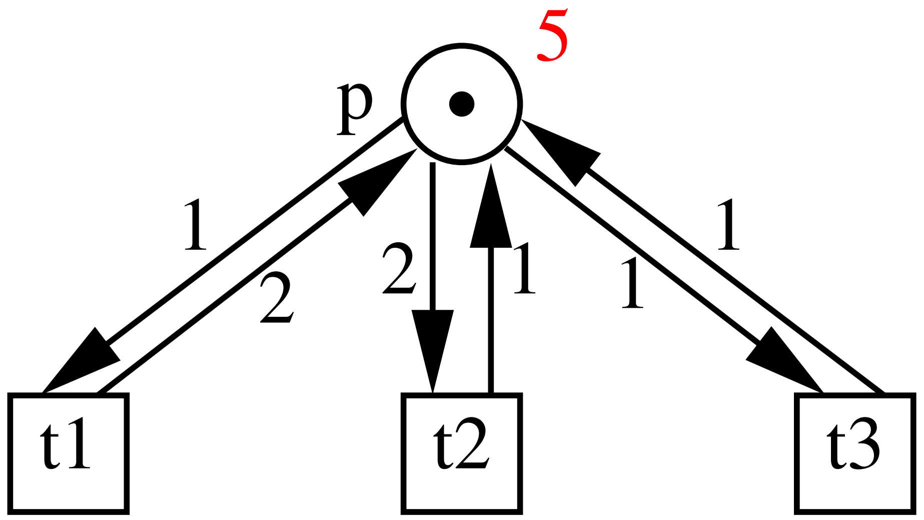
Let $N = \langle P, T, F, W, K, M_0 \rangle$ be a net (*with* capacities).

We define the net $N' = \langle P', T', F', W', M_0' \rangle$ (*without* capacities) as follows:

- $P' = P \cup \{ p' \mid \exists p \in P \colon K(p) \neq \infty \}$
  (i.e. we add an extra place for each place with restricted capacity);
- $T' = T$ (no change);
- $F'$ and $W'$ extend $F$ and $W$ as follows: For all $t \in T$ and $p \in P$ with $K(p) \neq \infty$, let $\Delta_{p,t} := W(t,p) - W(p,t)$. Then:
  - $(p', t) \in F'$ and $W'(p', t) = \Delta_{p,t}$ iff $\Delta_{p,t} > 0$;
  - $(t, p') \in F'$ and $W'(t, p') = -\Delta_{p,t}$ iff $\Delta_{p,t} < 0$.
- $M_0'(p) = M_0(p)$ for all $p \in P$;
  $M_0'(p') = K(p) - M_0(p)$ for all $p \in P$ with $K(p) \neq \infty$.

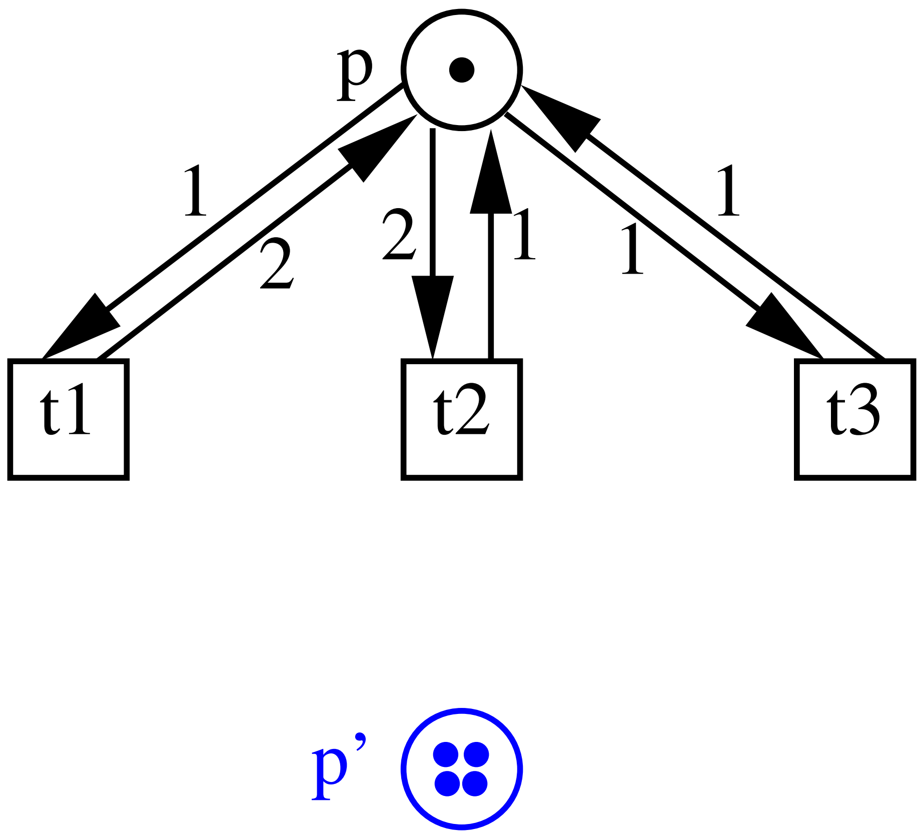
## Complement Construction: Example

Consider the following net with capacities:



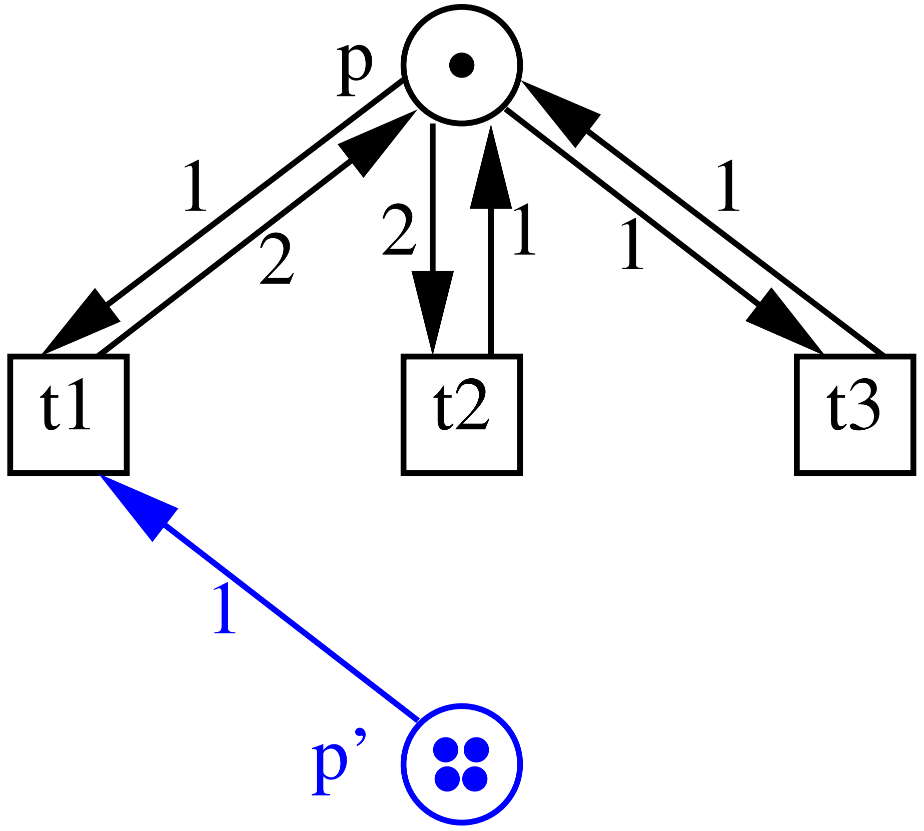
## Complement Construction: Example

We remove capacities; $p'$ gets $5 - M_0(p) = 4$ initial tokens.



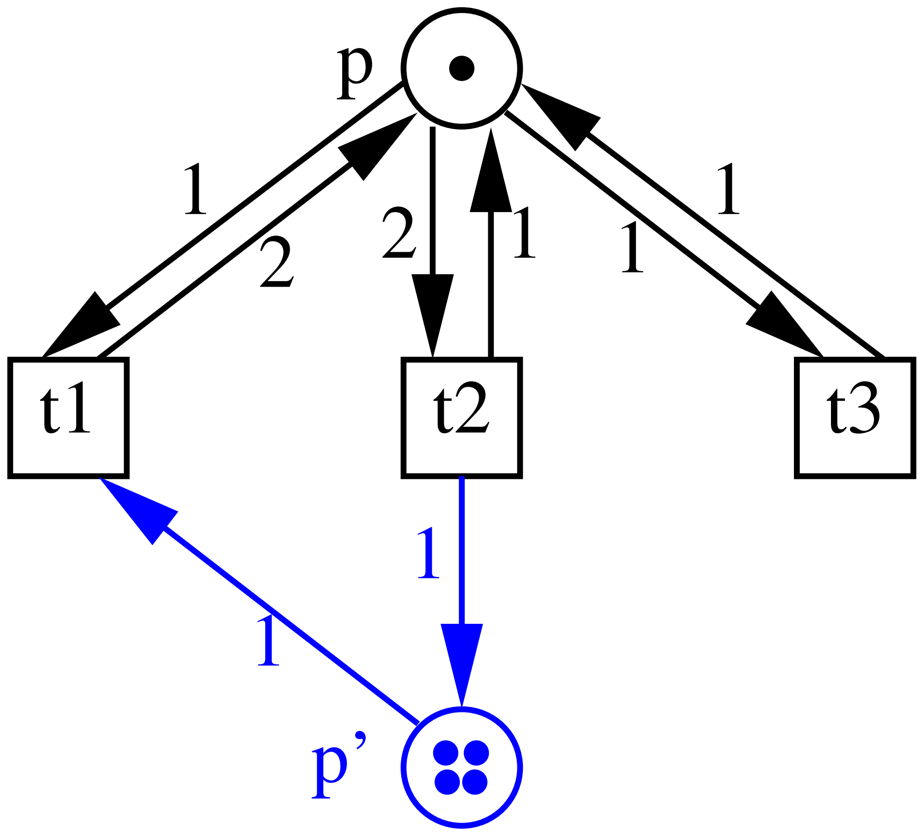
## Complement Construction: Example

$t_1$ adds a token to $p$, so one token is taken away from $p'$.



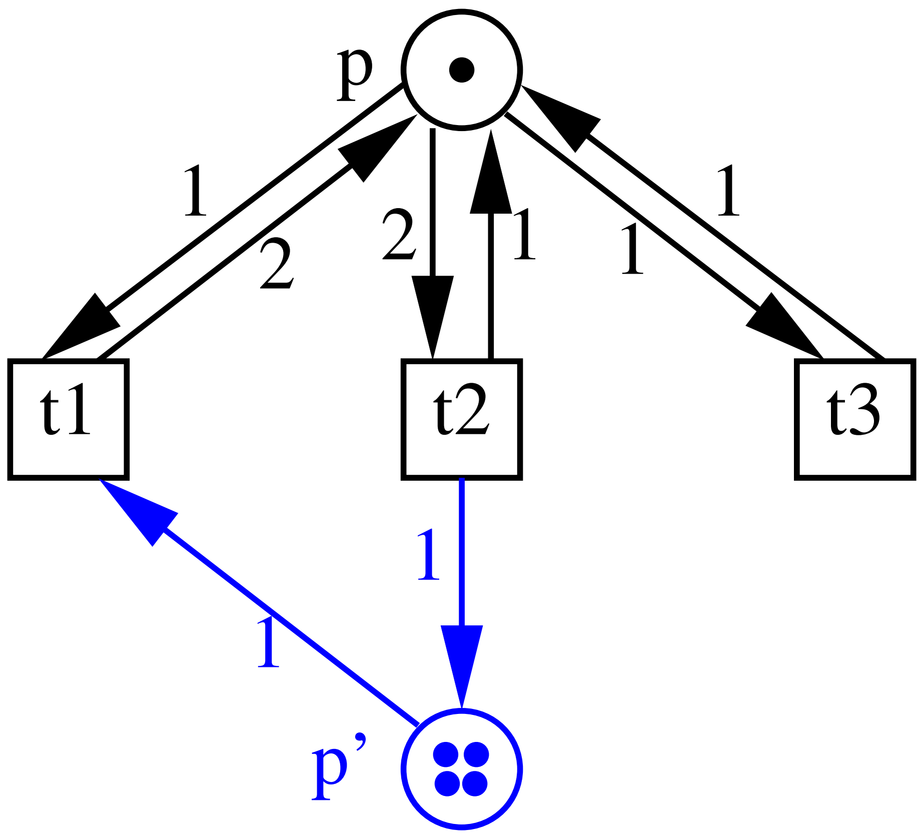
# Complement Construction: Example

$t_2$ removes a token from $p$, so one token added to $p'$.



## Complement Construction: Example

$t_3$ does not change the number of $p$; no change required.



## Complement Construction (cont'd)

Definition: If $M'$ is a marking of $N'$, denote by $M'|_P$ its restriction to places in $P$. $M'|_P$ is in fact a marking of $N$.

$N'$ has the following properties:

- The pairs $p$ and $p'$ (where $p'$ is one of the added places) are *complement places*.
- A marking $M'$ is reachable in $N'$ iff $M'|_P$ is reachable in $N$.

Thus, we can study the behaviour of $N$ (with capacities) by analysing the net $N'$ (without capacities).

## Note on complement construction

The firing rule of Petri nets does not allow to express the condition that some place $p$ *does not* contain a token.

However, this restriction can be circumvented with the complement construction:

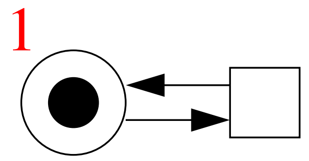
> Add a new place $p'$.
>
> Do the complement construction with assumed capacity $K(p) = 1$.
>
> For a transition to check whether $p$ does not contain a token, it can now check for the existence of a token on $p'$.

Example: Recall the man/wolf/etc puzzle; to check whether $ML$ did not contain a token (i.e. that the man was not on the left bank), we queried the complement place $MR$.

## Remark: Alternative definition of firing rule

Our firing condition for nets with capacities corresponds to the intuition *“Remove tokens first, add new tokens, check if capacity is still within limits”*.

Sometimes, the literature uses a different intuition: *“Produce new tokens first, check that capacity is still within limits, then remove tokens”*.



E.g., in the net shown above, the transition would be enabled under the first intuition, but not under the second one.

The second intuition is more restrictive and is designed to be more ‘secure’. However, unless stated otherwise, we use the first intuition (like on the previous slides).

## Differences with the alternative definition

With the alternative definition, the firing condition changes to:

> Transition $t \in T$ is $M$-enabled, written $M \xrightarrow{t}$, iff $\forall p \in {}^\bullet t : M(p) \geq W(p, t)$ and $\forall p \in t^\bullet : M(p) + W(t, p) \leq K(p)$.

In the complement construction, $F'$ and $W'$ behave as follows:

- $(p', t) \in F'$ and $W'(p', t) = W(t, p)$ iff $(t, p) \in F$;
- $(t, p') \in F'$ and $W'(t, p') = W(p, t)$ iff $(p, t) \in F$;

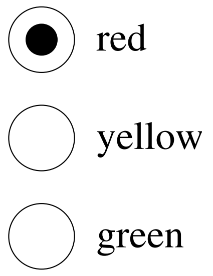
## Lecture 5:

## High-level nets
## Structural analysis of P/T nets

## High-level nets

Up to now, we allowed places to be occupied by black tokens.

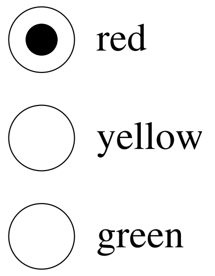
Traffic light example: Three lights, one colour per light, one place for every light.



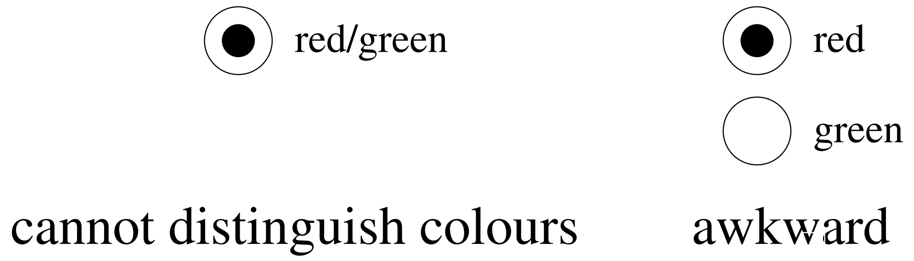
## High-level nets: Motivation

Up to now, we allowed places to be occupied by black tokens.

Traffic light example: Three lights, one colour per light, one place for every light.



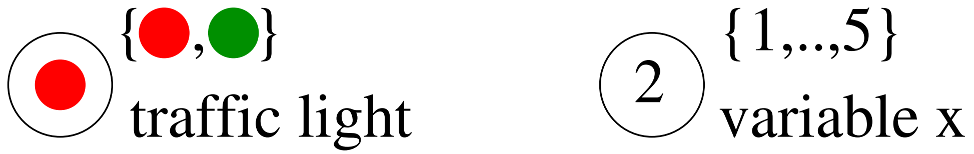
Suppose we have one light that can be either red or green.
Two (flawed) attempts at modelling this situation:



If we had not just black tokens, but coloured ones (e.g. red, green), we could construct a more natural model:

More generally, we could allow arbitrary values as tokens, e.g. to model numeric variables:

A general solution is to assign a type to every place, i.e. a set of token values that are permitted on the place:

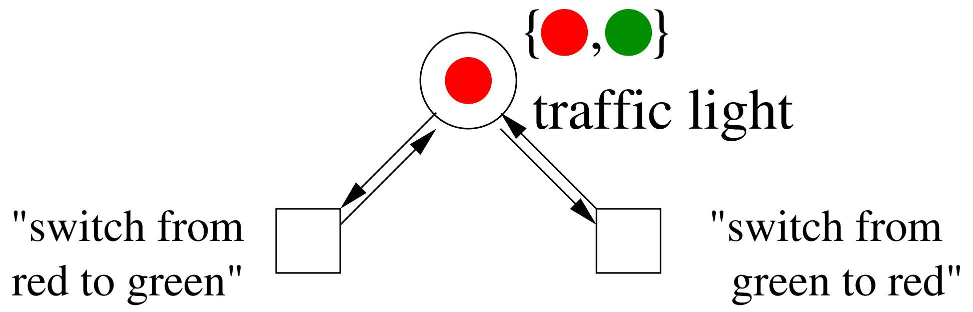


In general, a place may contain a multiset of its type.

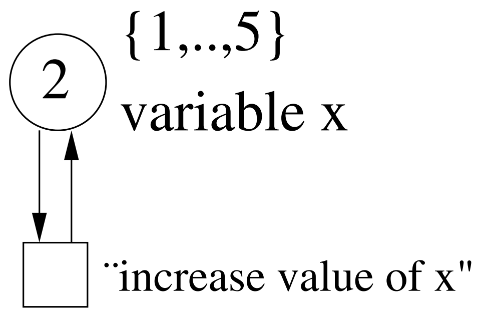
## High-level nets: Transitions

In meaningful models, we need transitions to reason about the values of tokens:

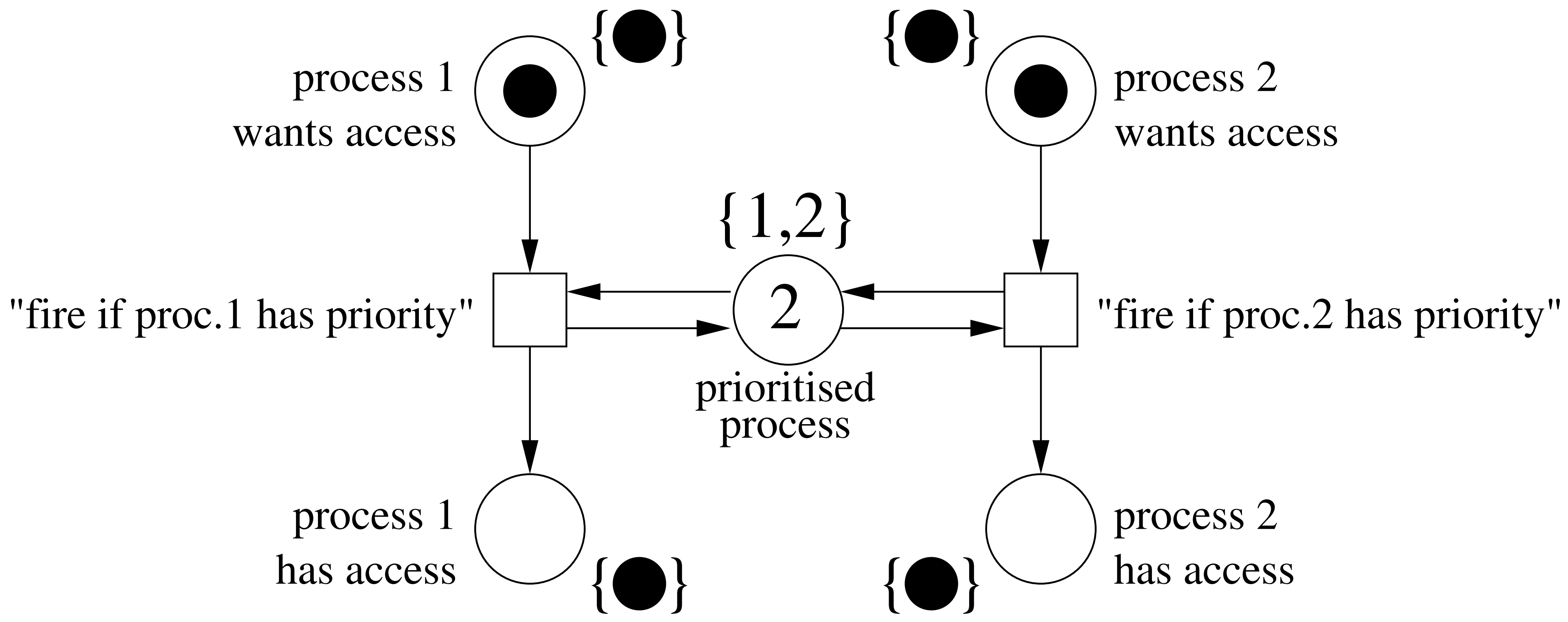
E.g., switching the traffic light:



Increasing a variable:



Suppose we have two processes competing for a common resource. If both try to access the resource simultaneously, there is a ‘referee’ who decides which process should have priority over the other:



## High-level nets: Definition

(This looks more complicated than it actually is. . . )

A High-Level Net (HL-net) is a tuple $N = \langle P, T, F, W, V, S, C, M_0 \rangle$, where

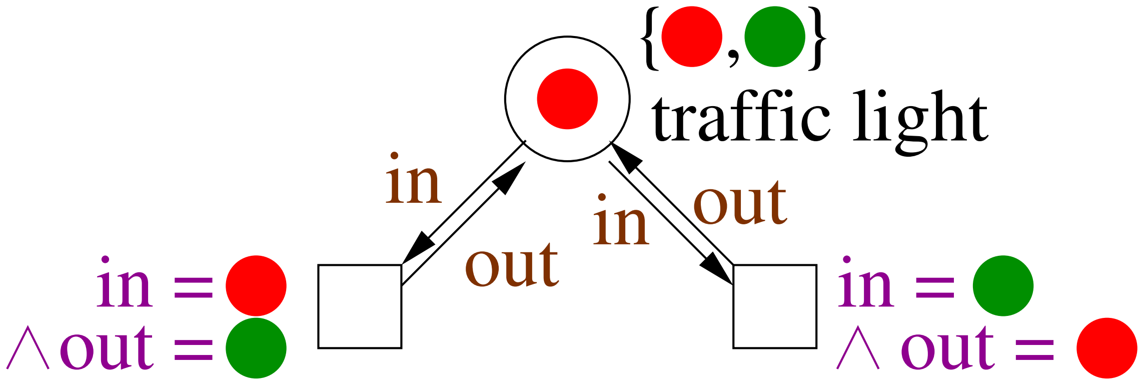
- $P$, $T$, $F$, $W$ are as usual;
- $V$ is a set of token values;
- $S\colon P \to 2^V$ is a type assignment for places;
- $(C_t)_{t \in T}$ is a collection of firing conditions (see next slide).
- $M_0\colon P \times V \to \mathbb{N}$ is the initial marking. Note: $M_0(p, v) = 0$ if $v \notin S(p)$.

## High-level transitions

A firing condition decides which tokens may flow out of the pre-places and into the post-places of a transition.

Formally, if we let $\Sigma_t$ be the sum of the arc weights leading into and out of $t$, then the signature of $C_t$ is $C_t\colon V^{\Sigma_t} \to \{\mathsf{false}, \mathsf{true}\}$.

In figures, we place variable names onto the arcs and equip transitions with boolean expressions over these variables, like this:



## Notes on high-level transitions

If a particular assignment of token values to variables evaluates to true, then the transition may fire *under that assignment*.

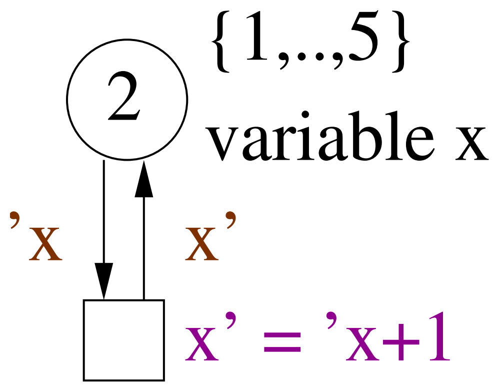
Firing under some assignment is possible if for every pre-place $p$, $p$ contains the token values assigned to the variables that are on the arc from $p$ to $t$.

Firing removes those tokens and puts corresponding tokens on the post-places.

The assignment must respect the types; e.g. if $v$ is the variable on the arc from place $p$ to transition $t$, then $v$ must be assigned to some value from $S(p)$.

## High-level transitions: Examples

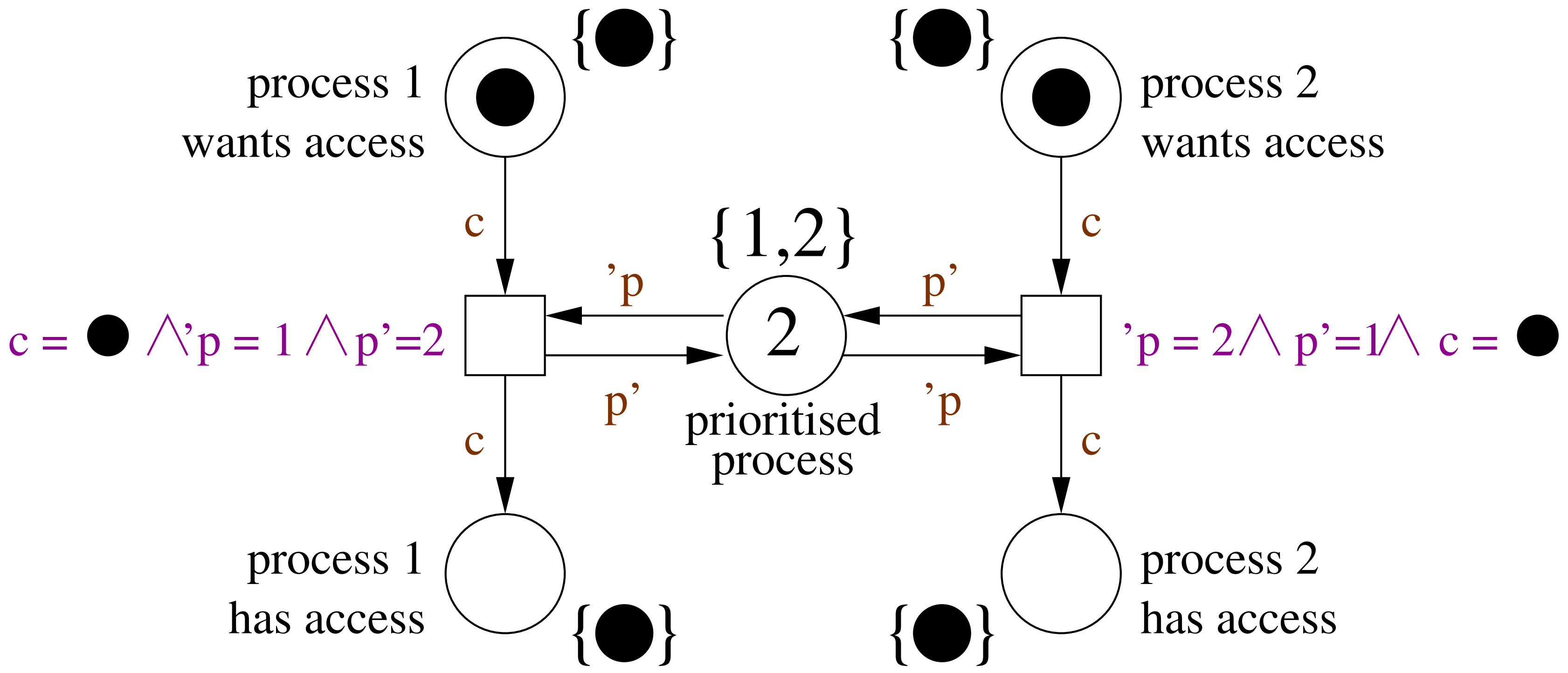
For instance, in the following example, we could fire under the assignments
$'x = 1, x' = 2,\quad 'x = 2, x' = 3,\quad 'x = 3, x' = 4,\quad 'x = 4, x' = 5$.



In the given marking, we can remove the 2 token and replace it by a 3 token.

## High-level transitions: Examples

In the common-resource example, suppose the prioritised process is changed after every access:



Note that the condition on *c* is actually unnecessary.

## Equivalence of P/T nets and High-Level nets

High-level nets allow easier modelling, but they are equally expressive, provided that the set of token values is finite.

We shall show that P/T nets can be translated into high-level nets and vice versa.

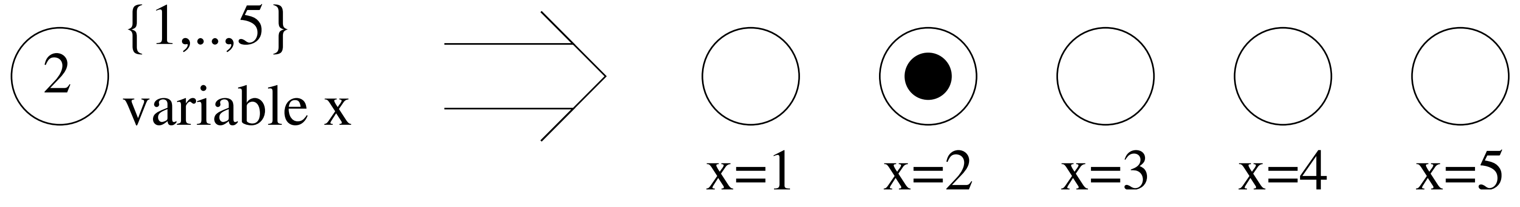
From P/T nets to high-level nets: trivial

Assign the type $\{\circ\}$ to each place and the condition 'true' to each transition.

## From high-level nets to P/T nets: Places

For each high-level place $p$, create a P/T place $p_v$ for each $v \in S(p)$.

If $M_0(p, v) = k$, then put $k$ initial tokens on the P/T place $p_v$.

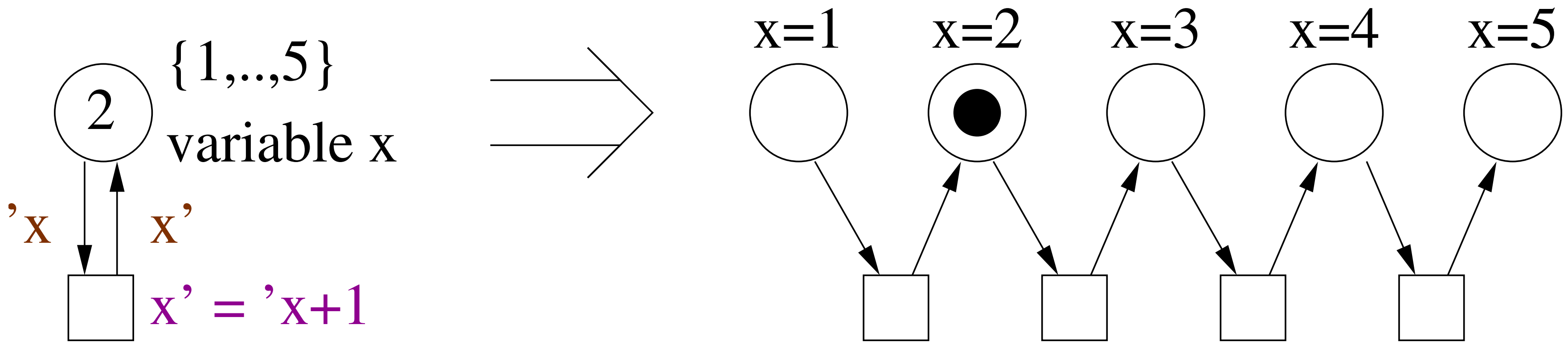


## From high-level nets to P/T nets: Transitions

For each high-level transition $t$, create a P/T transition $t_a$ for each assignment under which $t$ may fire.

If $(p, t)$ is a high-level arc with variable $x$, connect $p_v$ to $t_a$ in the P/T net, where $v$ is the value of $x$ in $a$.

Arcs from transitions to places are treated analogously.



## Structural analysis of P/T nets

## Structural Analysis: Motivation

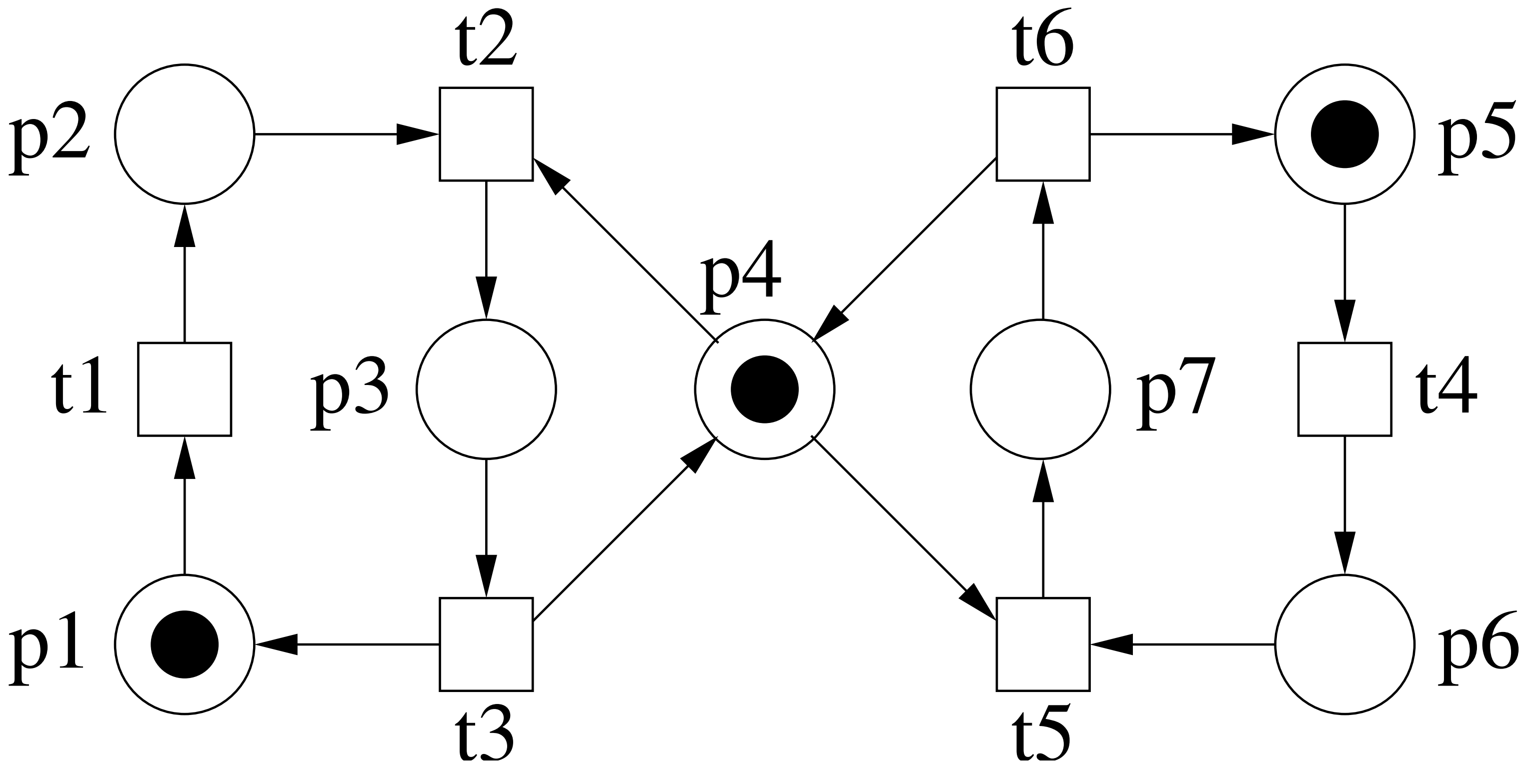
We have seen how properties of Petri nets can be proved by constructing the reachability graph and analysing it.

However, the reachability graph may become huge: exponential in the number of places (if it is finite at all).

Structural analysis makes it possible to prove some properties *without* constructing the reachability graph. The main techniques are:

- Place invariants
- Traps

## Example 1



## Incidence Matrix: Definition

Let $N = \langle P, T, F, W, M_0 \rangle$ be a P/T net. The corresponding incidence matrix $C_N \colon P \times T \to \mathbb{Z}$ is the matrix whose rows correspond to places and whose columns correspond to transitions. Column $t \in T$ denotes how the firing of $t$ affects the marking of the net: $C(t, p) = W(t, p) - W(p, t)$.

The incidence matrix of the example from the previous slide:

$$\begin{array}{c} \begin{pmatrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \end{array} \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{array}$$

## Markings as vectors

Let us now write marking as column vectors. E.g., the initial marking is $M_0 = (1\ 0\ 0\ 1\ 1\ 0\ 0)^T$.

Likewise, we can write firing counts as column vectors with one entry for each transition. E.g., if $t_1$, $t_2$, and $t_4$ are to fire once each, we can express this with $u = (1\ 1\ 0\ 1\ 0\ 0)^T$.

Then, the result of firing these transitions can be computed as $M_0 + C \cdot u$.

$$\begin{pmatrix}1\\0\\0\\1\\1\\0\\0\end{pmatrix} + \begin{pmatrix}-1 & 0 & 1 & 0 & 0 & 0\\ 1 & -1 & 0 & 0 & 0 & 0\\ 0 & 1 & -1 & 0 & 0 & 0\\ 0 & -1 & 1 & 0 & -1 & 1\\ 0 & 0 & 0 & -1 & 0 & 1\\ 0 & 0 & 0 & 1 & -1 & 0\\ 0 & 0 & 0 & 0 & 1 & -1\end{pmatrix} \cdot \begin{pmatrix}1\\1\\0\\1\\0\\0\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\\0\\1\\0\end{pmatrix}$$

## Caveat

Notice: Bi-directional arcs (an arc from a place to a transition and back) cancel each other out in the matrix!

Thus, when a marking arises as the result of a matrix equation (like on the previous slide), this does not guarantee that the marking is reachable!

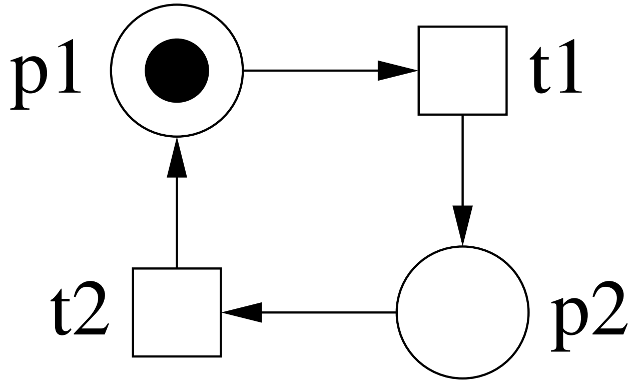
I.e., the markings obtained by the incidence markings are an over-approximation of the actual reachable markings (compare coverability graphs. . . ).

However, we *can* sometimes use the matrix equations to show that a marking $M$ is unreachable, i.e. if $M_0 + Cu = M$ has no natural solution for $u$.

Note: When we are talking about natural (integral) solutions of equations, we mean those whose components are natural (integral) numbers.

## Example 2

Consider the following net and the marking $M = (1\ 1)^T$.



$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

has no solution, and therefore $M$ is not reachable.

## Invariants

The natural solutions of the equation $Cu = \mathbf{0}$ are called transition invariants (or: T-invariants). They indicate (possible) loops.

For instance, in Example 2, $u = (1\ 1)^T$ is a T-invariant.

The integral solutions of the equation $C^T x = \mathbf{0}$ are called place invariants (or: P-invariants). A proper P-invariant is a natural solution of $C^T x = \mathbf{0}$ if $x \neq \mathbf{0}$.

For instance, in Example 1, $x_1 = (1\ 1\ 1\ 0\ 0\ 0\ 0)^T$, $x_2 = (0\ 0\ 1\ 1\ 0\ 0\ 1)^T$, and $x_3 = (0\ 0\ 0\ 0\ 1\ 1\ 1)^T$ are all (proper) P-invariants.

A P-invariant indicates that the number of tokens in all reachable markings satisfies some linear invariant (see next slide).

## Properties of P-invariants

Let $M$ be marking reachable with a transition sequence whose firing count is expressed by $u$, i.e. $M = M_0 + Cu$. Let $x$ be a P-invariant. Then, the following holds:

$$M^T x = (M_0 + Cu)^T x = M_0^T x + (Cu)^T x = M_0^T x + u^T C^T x = M_0^T x$$

For instance, invariant $x_2$ means that all reachable markings $M$ satisfy (reverting back to the function notation for markings):

$$M(p_3) + M(p_4) + M(p_7) = M_0(p_3) + M_0(p_4) + M_0(p_7) = 1 \qquad (1)$$

As a consequence, a P-invariant in which all entries are either 0 or 1 indicates a set of places in which the number of tokens remains unchanged in all reachable markings.

Note that multiplying an invariant by an integer constant or component-wise addition of two invariants will again yield a P-invariant.

We can use P-invariants to prove mutual exclusion properties:

> According to equation 1, in every reachable marking of Example 1 exactly one of the places $p_3$, $p_4$, and $p_7$ is marked. In particular, $p_3$ and $p_7$ cannot be marked concurrently!

Another example: Mutual exclusion with token passing (demo)

## More remarks on P-invariants

P-invariants can also be useful as a *pre-processing step* for reachability analysis.

Suppose that when computing the reachability graph, the marking of a place is normally represented with $n$ bits of storage. E.g. the places $p_3$, $p_4$, and $p_7$ together would require $3n$ bits.

However, as we have discovered invariant $x_2$, we know that exactly one of the three places is marked in each reachable marking.

Thus, we just need to store in each marking *which* of the three is marked, which required just 2 bits.

## Algorithms for P-invariants

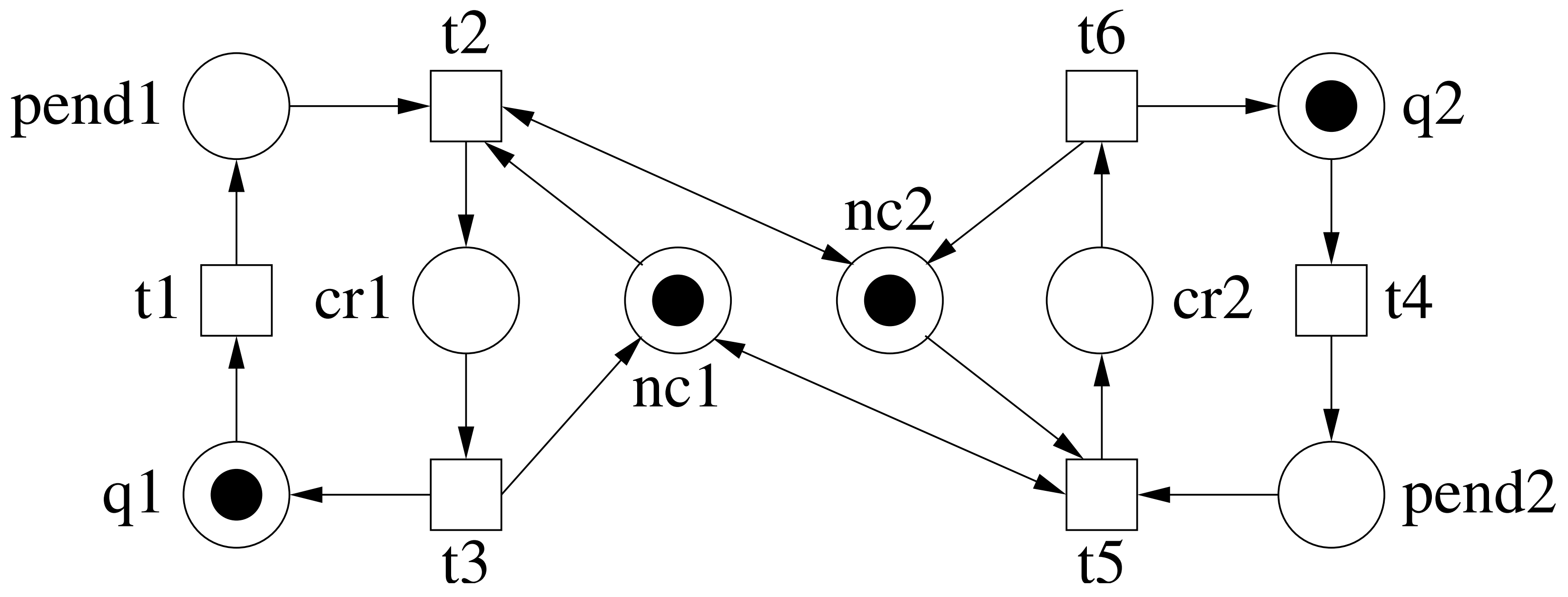
There is an algorithm called "Farkas Algorithm" (by *J. Farkas*, 1902) to compute a set of so called minimal P-invariants. These are proper place invariants from which any other invariant can be computed by a linear combination.

Unfortunately there are P/T-nets with an exponential number of minimal P-invariants (in the number of places of the net). Thus the Farkas algorithm needs (at least) exponential time in the worst case.

The INA tool of the group of *Peter Starke* (Humboldt University of Berlin) contains a large number of algorithms for structural analysis of P/T-nets, including invariant generation.

## Example 3

Consider the following attempt at a mutual exlusion algorithm for $cr_1$ and $cr_2$:



The idea is to achieve mutual exclusion by entering the critical section only if the other process is not already there.

Thus, we want to prove that in all reachable markings $M$:

$$M(cr_1) + M(cr_2) \leq 1$$

The P-invariants we can derive in the net yield:

$$\begin{aligned} M(q_1) + M(pend_1) + M(cr_1) &= 1 && (2) \\ M(q_2) + M(pend_2) + M(cr_2) &= 1 && (3) \\ M(cr_1) + M(nc_1) &= 1 && (4) \\ M(cr_2) + M(nc_2) &= 1 && (5) \end{aligned}$$

But try as we might, we cannot show the desired property just with these four equations!, needs more notions (traps).