Petri Nets
Lecture Notes

Prof. Javier Esparza

April 25, 2022

Contents

Part I

Petri Nets: Syntax, Semantics, Models

Chapter 1

Basic definitions

1.1 Preliminaries

Numbers

$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ denote the natural, integer, rational, and real numbers.

Relations

Let X be a set and $R \subseteq X \times X$ a relation. R^* denotes the transitive and reflexive closure of R.

R^{-1} is the inverse of R, that is, the relation defined by

$$(x, y) \in R^{-1} \Leftrightarrow (y, x) \in R\,.$$

Sequences

A finite sequence over a set A is a mapping $\sigma\colon \{1,\dots,n\} \to A$, denoted by the string $a_1a_2\dots a_n$, where $a_i = \sigma(i)$ for every $1 \leq i \leq n$, or the mapping $\epsilon\colon \emptyset \to A$, the empty sequence.

The length of σ is n and the length of ϵ is 0.

An infinite sequence is a mapping $\sigma : \mathbb{N} \to A$. We write $\sigma = a_1a_2a_3\dots$.

The concatenation of two finite sequences or of a finite and an infinite sequence is defined as usual.

Given a finite sequence σ, we denote by σ^ω the infinite concatenation $\sigma\sigma\sigma\dots$.

σ is a prefix of τ if $\sigma = \tau$ or $\sigma\sigma' = \tau$ for some sequence σ'.

The alphabet of a sequence σ is the set of elements of A occurring in σ.

Given a sequence σ over A and $B \subseteq A$, the projection or restriction $\sigma|_B$ is the result of removing all occurrences of elements $a \in A \setminus B$ in σ.

Vectors and matrices

Let $A = \{a_1, \ldots, a_n\}$ be a finite set and let K be one of $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$.

We represent a mapping $X\colon A \to K$ by the vector $(X(a_1), \ldots, X(a_n))$. We identify the mapping X and its vector representation.

Let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ be vectors.

The (scalar) product $X \cdot Y$ is the number $x_1y_1 + \ldots + x_ny_n$ (we do not distinguish between row and column vectors!).

We write $X \geq Y$ to denote $x_1 \geq y_1 \wedge \ldots \wedge x_n \geq y_n$, and $X > Y$ to denote $x_1 > y_1 \wedge \ldots \wedge x_n > y_n$.

Let $B = \{b_1, \ldots, b_m\}$ be a finite set. A mapping $C\colon A \times B \to K$ is represented by the $n \times m$ matrix

$$\begin{pmatrix} C(a_1,b_1) & C(a_1,b_2) & \cdots & C(a_1,b_m) \\ C(a_2,b_1) & C(a_2,b_2) & \cdots & C(a_2,b_m) \\ \cdots & \cdots & \cdots & \cdots \\ C(a_n,b_1) & C(a_n,b_2) & \cdots & C(a_n,b_m) \end{pmatrix}$$

We also write $C = (c_{ij})_{i=1,\ldots,n,j=1,\ldots,m}$, where $c_{ij} = C(a_i, b_j)$.

Let $X = (x_1, \ldots, x_m)$ be a vector and let C be a $n \times m$ matrix.

The product $C \cdot X$ is the vector $Y = (y_1, \ldots, y_n)$ given by

$$y(i) = c_{i1}x_1 + \ldots + c_{im}x_m$$

For $X = (x_1, \ldots, x_n)$ the product $X \cdot C$ is the vector $Y = (y_1, \ldots, y_m)$ given by

$$y(i) = c_{1i}x_1 + \ldots + c_{ni}x_n$$

Complexity Classes

Time and space

A program is deterministic if it only has one possible computation for each input.

A program is nondeterministic if it may execute different computations for the same input.

A program (deterministic or not) runs in $f(n)$-time for a function $f : \mathbb{N} \to \mathbb{N}$ if for every input of length n (measured in bits) every computation takes at most $f(n)$ time.

Let $\mathcal{C}$ be a set of functions $\mathbb{N} \to \mathbb{N}$ (for example, $\mathcal{C}$ can be the set of all polynomial functions).

A program runs in $\mathcal{C}$-time if it runs in $f(n)$ time for some function $f(n)$ of $\mathcal{C}$. Often we speak of a "polynomial-time program" or "exponential-time" program, meaning a program that runs in time $f(n)$ for some polynomial resp. exponential function $f(n)$.

A program needs $f(n)$-memory or $f(n)$-space for a function $f : \mathbb{N} \to \mathbb{N}$ if it uses at most $f(n)$ bits of memory for every input of length n. The $f(n)$ bits do not include the memory needed to store the input. We speak of "polynomial-space" or "exponential-space" programs.

Problems

A problem consists of a universe U of possible inputs, and a predicate P on U assigning to each $u \in U$ a value $P(u) \in \{0, 1\}$. For example, U can be the set of all finite graphs, and $P(u)$ the predicate with $P(u) = 1$ iff u has a cycle.

A deterministic program solves a problem (U, P) if it terminates for every input $u \in U$ and returns $P(u)$.

A nondeterministic program solves a problem (U, P) if for every $u \in U$:

- if $P(u) = 1$ then at least one computation of the program returns 1; and
- if $P(u) = 0$ then every computation of the program returns 0.

Observe: if the program returns 1 then we know $P(u) = 1$, otherwise we do not know anything.

Time and space classes

- **P** is the class of problems that can be solved by polynomial-time deterministic programs.
- **NP** is the class of problems that can be solved by polynomial-time nondeterministic programs.
- **PSPACE** is the class of problems that can be solved by polynomial-space deterministic programs.
- **NPSPACE** is the class of problems that can be solved by polynomial-space nondeterministic programs.
- **EXPTIME** is the class of problems that can be solved by exponential-time deterministic programs.
- **EXPSPACE** is the class of problems that can be solved by exponential-space deterministic programs.

We have : **P** $\subseteq$ **NP** $\subseteq$ **PSPACE** $\subseteq$ **EXPTIME** $\subseteq$ **EXPSPACE**.

It is widely believed that all these inclusions are strict. However, all we know for sure is the (rather trivial facts) **P** $\subset$ **EXPTIME** and **PSPACE** $\subset$ **EXPSPACE**. We also know

Theorem 1.1.1 *[Savitch's theorem]*

$$\textbf{NPSPACE} = \textbf{PSPACE}$$

Reductions, hardness and completeness

A problem $\Pi_1 = (U_1, P_1)$ can be polynomially reduced to $\Pi_2 = (U_2, P_2)$ if there is a function $f : U_1 \to U_2$ satisfying the following two properties:

- for every $u_1 \in U_1$: $P_1(u_1) = 1$ iff $P_2(f(u_1))$, and
- there is a polynomial-time deterministic program that computes f.

For all the complexity classes above, if Π_1 can be reduced to Π_2 and Π_2 belongs to the class, then so does Π_1.

A problem is hard for a complexity class if all problems in the class can be reduced to it.

A problem is complete for a class if it is hard for the class, and belongs to the class.

1.2 Syntax

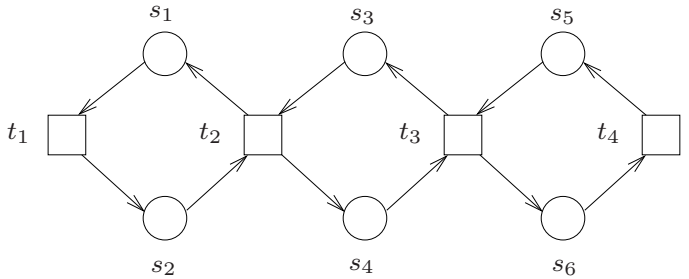
Definition 1.2.1 (Net, preset, postset)
A net $N = (S, T, F)$ consists of a finite set S of places (represented by circles), a finite set T of transitions disjoint from S (squares), and a flow relation (arrows) $F \subseteq (S \times T) \cup (T \times S)$.

The places and transitions of N are called elements or nodes. The elements of F are called arcs.

Given $x \in S \cup T$, the set ${}^\bullet x = \{y \mid (y, x) \in F\}$ is the preset of x and $x^\bullet = \{y \mid (x, y) \in F\}$ is the postset of x. For $X \subseteq S \cup T$ we denote ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$.

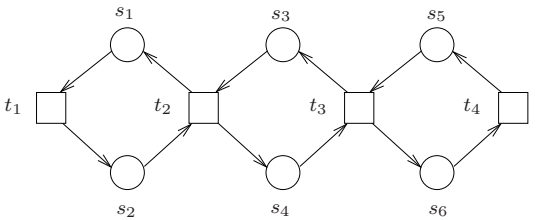
Example. Let $N = (S, T, F)$ be the net with $S = \{s_1, \ldots, s_6\}$, $T = \{t_1, \ldots, t_4\}$, and

$$\begin{array}{rcl} S & = & \{s_1, \ldots, s_6\} \\ T & = & \{t_1, \ldots, t_4\} \\ F & = & \{(s_1, t_1), (t_1, s_2), (s_2, t_2), (t_2, s_1), \\ & & (s_3, t_2), (t_2, s_4), (s_4, t_3), (t_3, s_3), \\ & & (s_5, t_3), (t_3, s_6), (s_6, t_4), (t_4, s_5)\} \end{array}$$

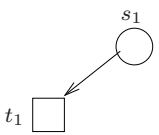


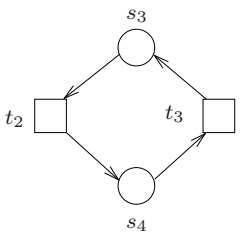
Definition 1.2.2 (Subnet)
$N' = (S', T', F')$ is a subnet of $N = (S, T, F)$ if

- $S' \subseteq S$,
- $T' \subseteq T$, and
- $F' = F \cap ((S' \times T') \cup (T' \times S'))$ (not $F' \subseteq F \cap ((S' \times T') \cup (T' \times S'))$).

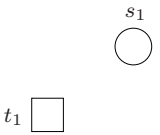


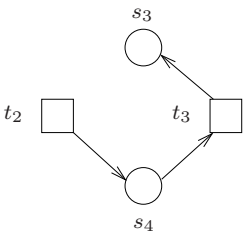
Subnets





Non−subnets





Definition 1.2.3 (Path, circuit)
A path of a net $N = (S, T, F)$ is a finite, nonempty sequence $x_1 \dots x_n$ of nodes of N such that $(x_1, x_2), \dots, (x_{n-1}, x_n) \in F$.

We say that a path $x_1 \dots x_n$ leads from x_1 to x_n.

A path is a circuit if $(x_n, x_1) \in F$ and $(x_i = x_j) \Rightarrow i = j$ for every $1 \leq i, j \leq n$.

N is connected if $(x, y) \in (F \cup F^{-1})^*$ for every $x, y \in S \cup T$.

N is strongly connected if $(x, y) \in F^*$ for every $x, y \in S \cup T$.

Proposition 1.2.4 *Let $N = (S, T, F)$ be a net.*

(1) N is connected iff there are no two subnets (S_1, T_1, F_1) and (S_2, T_2, F_2) of N such that

- $S_1 \cup T_1 \neq \emptyset$, $S_2 \cup T_2 \neq \emptyset$;
- $S_1 \cup S_2 = S$, $T_1 \cup T_2 = T$, $F_1 \cup F_2 = F$;
- $S_1 \cap S_2 = \emptyset$, $T_1 \cap T_2 = \emptyset$.

(2) A connected net is strongly connected iff for every $(x, y) \in F$ there is a path leading from y to x.

1.3 Semantics

Definition 1.3.1 (Markings)
Let $N = (S, T, F)$ be a net. A marking of N is a mapping $M : S \to \mathbb{N}$.

Given $R \subseteq S$ we write $M(R) = \sum_{s \in R} M(s)$.

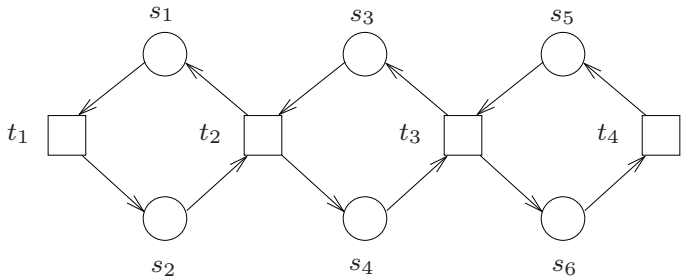
A place s is marked at M if $M(s) > 0$.

A set of places R is marked at M if $M(R) > 0$, that is, if at least one place of R is marked at M.

Instead of mappings $S \to \mathbb{N}$ sometimes we use vectors. For this we fix a total order on the places of N. With this convention we can represent a marking $M : S \to \mathbb{N}$ as a vector of dimension $|S|$.

Markings are graphically represented by drawing black dots ("tokens") on the places.

Consider the net



Let M be the marking given by

$$M(s_1) = M(s_4) = M(s_5) = 1 \qquad M(s_2) = M(s_3) = M(s_6) = 0$$

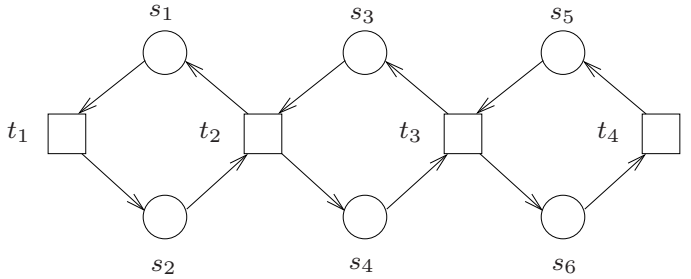
We denote this marking by the vector $(1, 0, 0, 1, 1, 0)$.

Definition 1.3.2 (Firing rule, dead markings)
A transition is enabled at a marking M if $M(s) \geq 1$ for every place $s \in {}^\bullet t$.
If t is enabled, then it can occur or fire, leading from M to the marking M' (denoted $M \xrightarrow{t} M'$) given by:

$$M'(s) = \begin{cases} M(s) - 1 & \text{if } s \in {}^\bullet t \setminus t^\bullet \\ M(s) + 1 & \text{if } s \in t^\bullet \setminus {}^\bullet t \\ M(s) & \text{otherwise} \end{cases}$$

A marking is dead if it does not enable any transition.

Example 1.3.3 *C*onsider the net



Let M be the marking $(1, 0, 0, 1, 1, 0)$.

The marking enables t_1 and t_3, because ${}^\bullet t_1 = \{s_1\}$ and ${}^\bullet t_3 = \{s_4, s_5\}$.

Transition t_2 is not enabled, because $M(s_2) = 0$.

Transition t_4 is not enabled, because $M(s_6) = 0$.

We have

$$(1, 0, 0, 1, 1, 0) \xrightarrow{t_1} (0, 1, 0, 1, 1, 0)$$
$$(1, 0, 0, 1, 1, 0) \xrightarrow{t_3} (1, 0, 1, 0, 0, 1)$$

Definition 1.3.4 (Firing sequence, reachable marking)
Let $N = (S, T, F)$ be a net and let M be a marking of N.

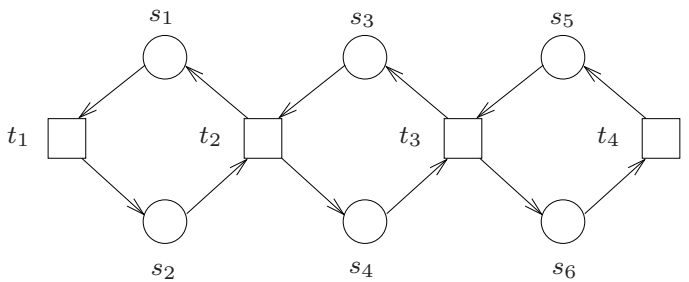
A finite sequence $\sigma = t_1 \dots t_n$ is enabled at a marking M if there are markings $M_1, M_2, \dots, M_n$ such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} \dots \xrightarrow{t_n} M_n$. We write $M \xrightarrow{\sigma} M_n$.

The empty sequence ϵ is enabled at any marking and we have $M \xrightarrow{\epsilon} M$.

If $M \xrightarrow{\sigma} M'$ for some markings M, M' and some sequence σ, then we write $M \xrightarrow{*} M'$ and say that M' is reachable from M.

An infinite sequence $\sigma = t_1 t_2 \dots$ is enabled at a marking if there are markings $M_1, M_2, \dots$ such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \longrightarrow \dots$

Example 1.3.5 *C*onsider the net



We have

$$(1,0,0,1,1,0) \xrightarrow{t_1} (0,1,0,1,1,0) \downarrow t_3 (0,1,1,0,0,1) \xrightarrow{t_2} (1,0,0,1,0,1) \xrightarrow{t_4} (1,0,0,1,1,0)$$

So M enables $t_1\ t_3\ t_2\ t_4$ and the infinite sequence $(t_1\ t_3\ t_2\ t_4)^\omega$.

Exercises in AutomataTutor

This script contains training excercises in AutomataTutor. They can be found in boxes like the one below this paragraph. To access them click on the excercise name (in blue). The symbols before the excercise tell you what how difficult an exercise is and what its main type is.

Difficulty	Symbol
Standard	☆
Harder	⯪
Challenging	★

Type	Symbol
Construction	⧈
Algorithm execution	⚙
Proof	■
Find a ...	🔍

Training Exercises in Automata Tutor:

- ☆ 🔍 All ones: Find a firing sequence that puts one token in every place.
- ☆ 🔍 Shortest Firing Sequence: Try to find a firing sequence with as little steps as possible that puts a token in a certain place.
- ⯪ 🔍 Longest Firing Sequence: Try to find a firing sequence with as many steps as possible.

The Monotonicity Lemma

Proposition 1.3.6 *A (finite or infinite) sequence σ is enabled at M iff every finite prefix of σ is enabled at M.*

Lemma 1.3.7 *[Monotonicity lemma]*
Let M and L be two markings of a net.

(1) If $M \xrightarrow{\sigma} M'$ for a finite sequence σ, then $(M + L) \xrightarrow{\sigma} (M' + L)$ for every marking L.

(2) If $M \xrightarrow{\sigma}$ for an infinite sequence σ, then $(M + L) \xrightarrow{\sigma}$ for every marking L.

Proof. (1): by induction on the length of σ.
Basis: $\sigma = \epsilon$. ϵ is enabled at any marking.
Step: Let $\sigma = \tau t$ (t transition) such that $M \xrightarrow{\tau} M'' \xrightarrow{t} M'$.

By induction hypothesis $(M + L) \xrightarrow{\tau} (M'' + L)$.

From the firing rule and $M'' \xrightarrow{t} M'$ we get $(M'' + L) \xrightarrow{t} (M' + L)$.

So $(M + L) \xrightarrow{\tau t} (M' + L)$.

(2): We show that every finite prefix of σ is enabled at $M + L$. The result then follows from Proposition **??**.

By Proposition **??**, every finite prefix of σ is enabled at M.

That is, for every finite prefix τ of σ there is a marking M' such that $M \xrightarrow{\tau} M'$.

By (1) we get $(M + L) \xrightarrow{\tau} (M' + L)$, and we are done. □

Petri nets and reachability graph

Definition 1.3.8 (Petri nets)
A Petri net, net system, or just a system is a pair (N, M_0) where N is a connected net $N = (S, T, F)$ with nonempty sets of places and transitions, and an initial marking $M_0 \colon S \to \mathbb{N}$.

A marking M is reachable in (N, M_0) or a reachable marking of (N, M_0) if $M_0 \xrightarrow{*} M$.

Definition 1.3.9 (Reachability graph)
The reachability graph G of a Petri net (N,M_0) where $N = (S, T, F)$ is the directed, labeled graph satisfying:

- The nodes of G are the reachable markings of (N, M_0).
- The edges of G are labeled with transitions from T.
- There is an edge from M to M' labeled by t iff $M \xrightarrow{t} M$, that is, iff M enables t and the firing of t leads from M to M'.

Algorithm for computing the reachability graph

```
REACHABILITY-GRAPH((S, T, F, M_0))
 1  (V, E, v_0) := ({M_0}, ∅, M_0);
 2  Work := {M_0};
 3  while Work ≠ ∅
 4  do select M from Work;
 5     Work := Work \ {M};
 6     for t ∈ enabled(M)
 7     do M' := fire(M, t);
 8        if M' ∉ V
 9          then V := V ∪ {M'}
10               Work := Work ∪ {M'};
11        E := E ∪ {(M, t, M')};
12  return (V, E, v_0)
```

The algorithm uses two functions:

- $\mathsf{enabled}(M)$: returns the set of transitions enabled at M.
- $\mathsf{fire}(M, t)$: returns the marking M' such that $M \xrightarrow{t} M'$.

The set $Work$ 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.

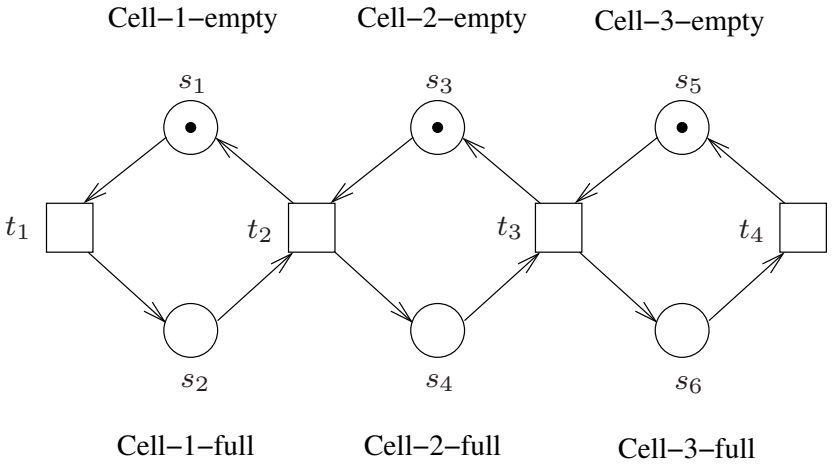
Training Exercise in AutomataTutor:
☆ ✿ Construct Reachability Graph: Construct the complete reachability graph of the given Petri net step-by-step.

Chapter 2

Modelling with Petri nets

2.1 A buffer of capacity n

We model a buffer with capacity for n items. For $n = 3$:



The model consists of n cells, each of them with capacity for one item.

The addition of a new item is modeled by the firing of t_1.

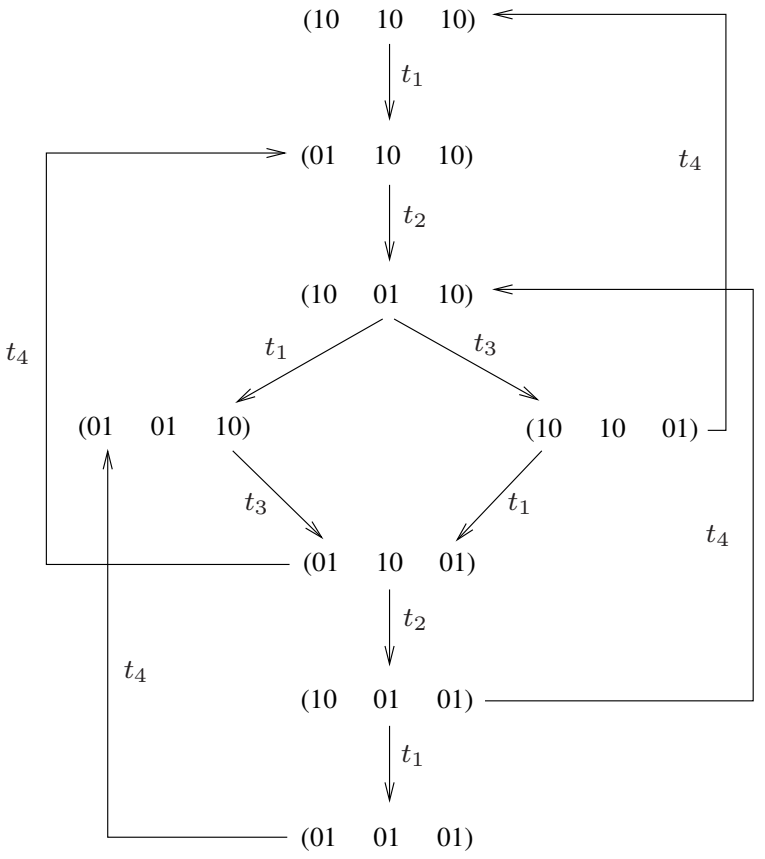
The firing of transition t_i models moving the item in cell $i-1$ to cell i.

Firing t_{n+1} models removing one item.

There are reachable markings at which transitions t_1 and t_{n+1} can occur independently of each other, that is, an item can be added while another one is being removed.

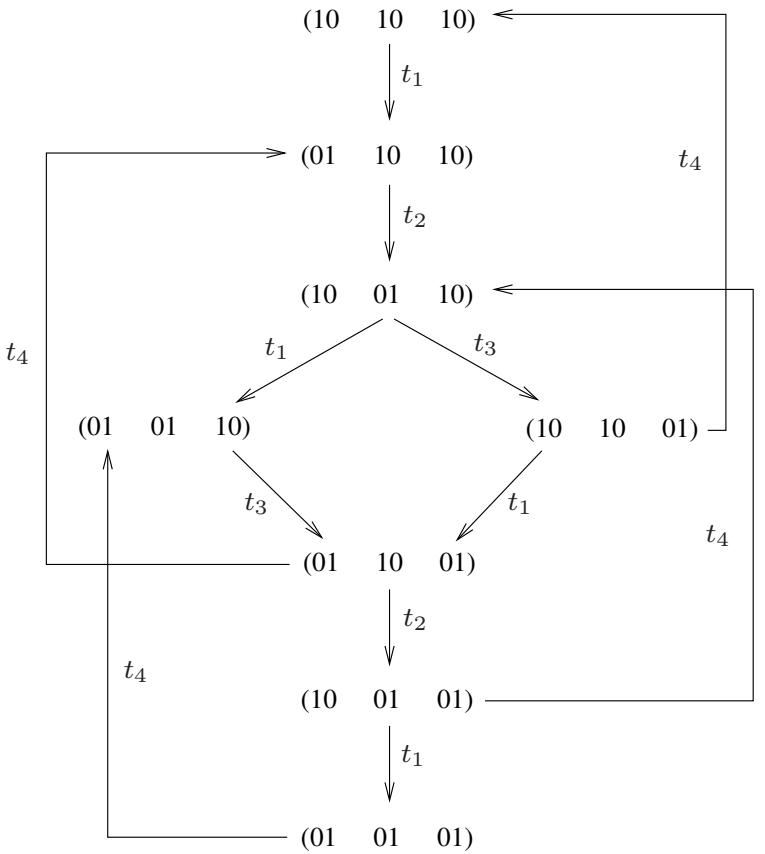
Training Exercise in AutomataTutor:
☆ ⚙ Construct Reachability Graph: Create a reachability graph for a buffer of capacity 2.

Reachability graph of the buffer with capacity 3



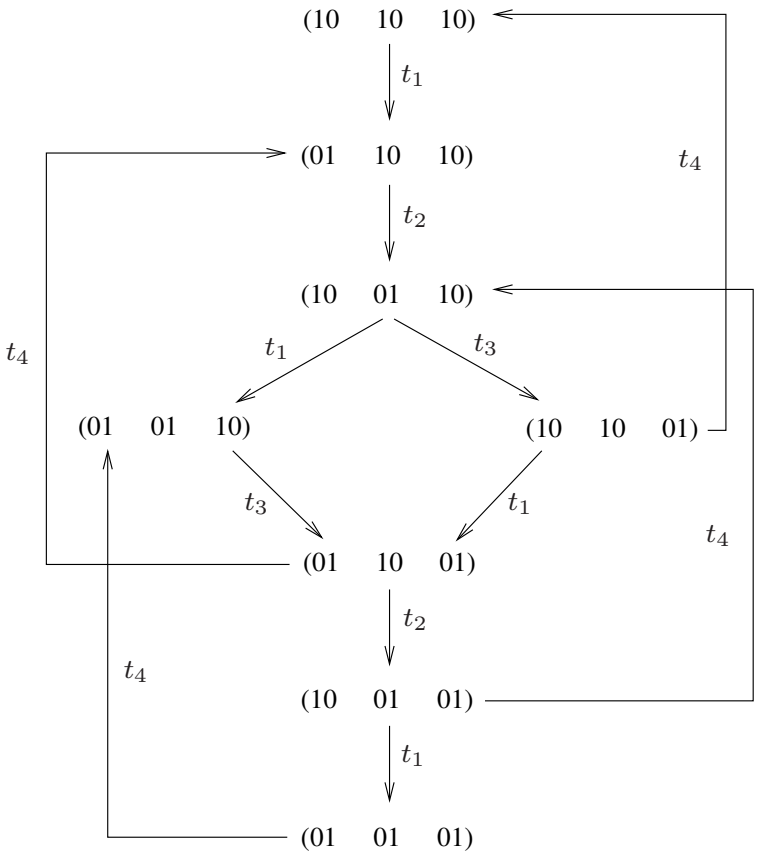
By inspection of the reachability graph we can see that a number of properties hold.

Reachability graph of the buffer with capacity 3



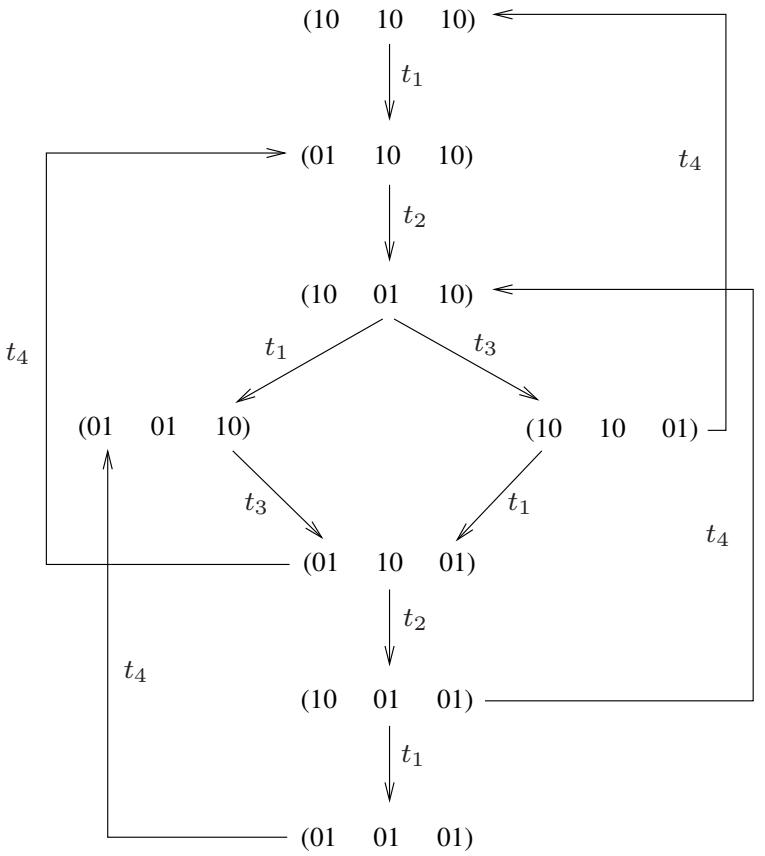
Consistency: no cell is simultaneously empty and full (that is, no marking puts tokens on s_i and s_{i+1} for $i = 1, 3, 5$).

Reachability graph of the buffer with capacity 3



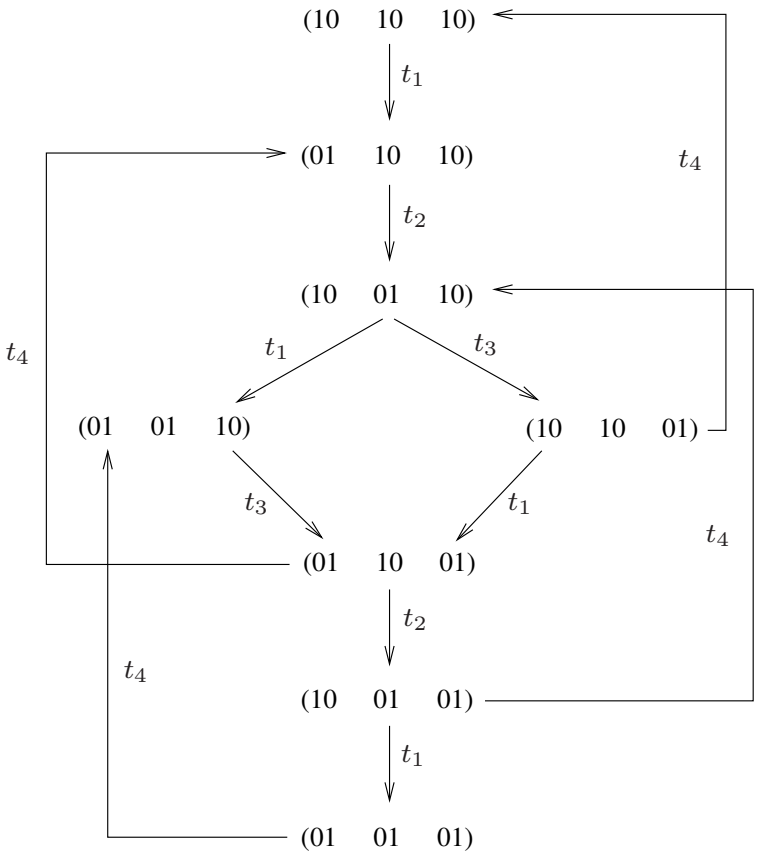
1-boundedness: every reachable marking puts at most one token in a given place.

Reachability graph of the buffer with capacity 3



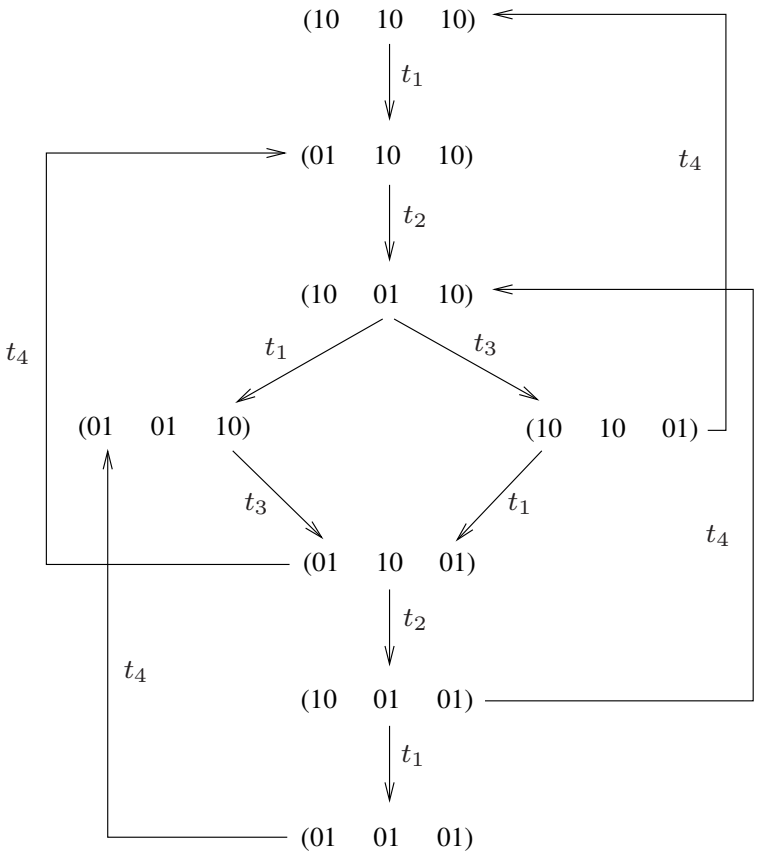
Deadlock freedom: every reachable marking has at least one successor marking.
Even more: every cell can always be filled and emptied again (every transition can occur again).

Reachability graph of the buffer with capacity 3



Capacity 3: the buffer has indeed capacity 3, that is, there is a reachable marking that puts one token in s_2, s_4, s_6.

Reachability graph of the buffer with capacity 3



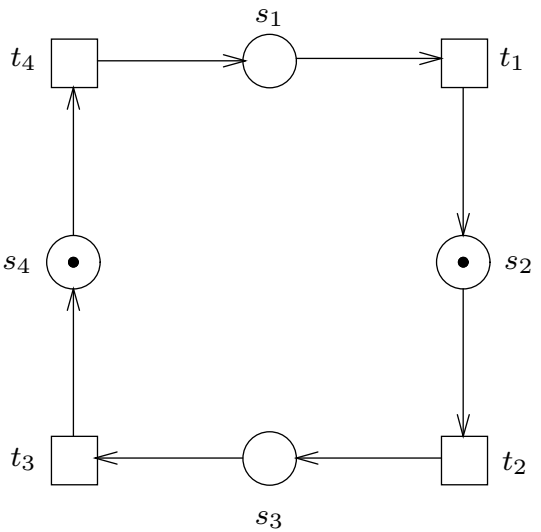
Between any two reachable markings there is a path of length at most 6.

2.2 Train tracks

Four cities are connected by unidirectional train tracks building a circle. Two trains circulate on the tracks.

We model the four tracks by places $s_1, \ldots, s_4$.

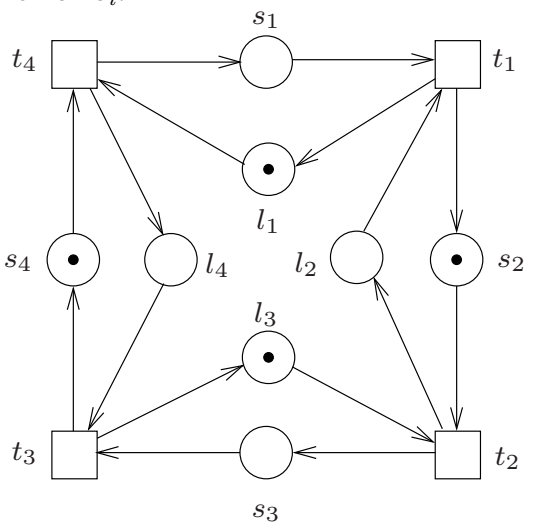
A token on s_i means that there is train in the i-th track.



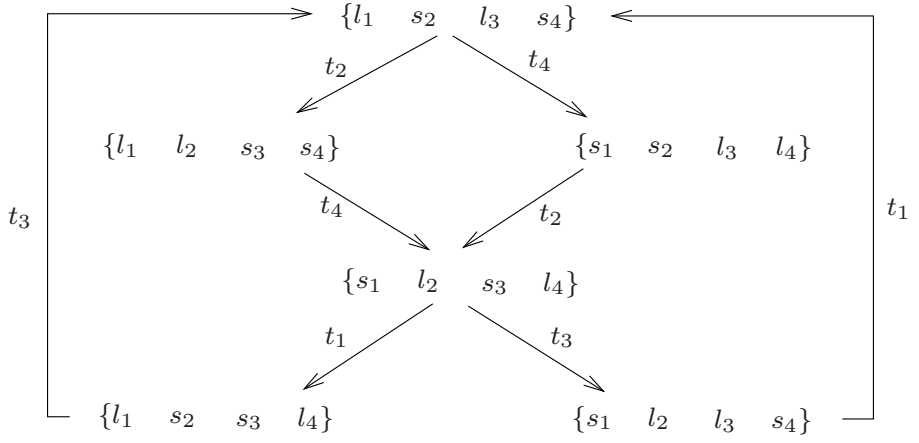
Our task is to ensure that it will never be the case that two trains occupy the same track.

Training Exercise in AutomataTutor:
Construct Reachability Graph: Create a reachability graph for the train tracks.

The four control places $l_1, \ldots, l_4$ guarantee that no reachable marking puts more than one token on s_i.

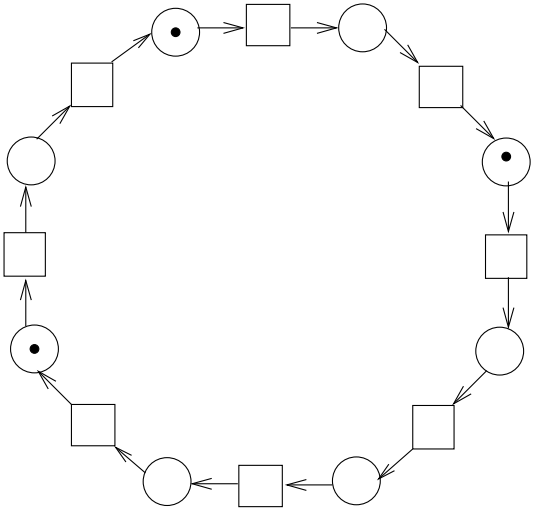


This property can be proven by means of the reachability graph. Since every reachable marking puts at most one token on a place, we denote a marking by the set of places marked by it.



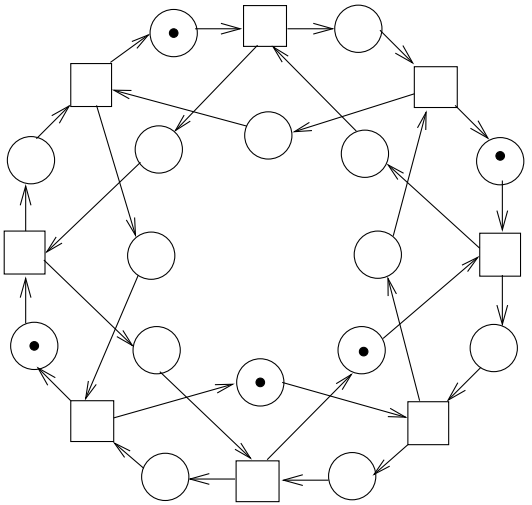
Another train-tracks problem

Now we have 8 cities connected in a circuit, and three trains use the tracks.



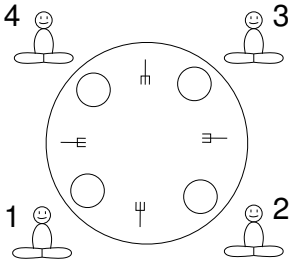
To increase safety, we have to guarantee that there always is at least one empty track between any two trains.

Here is a solution.

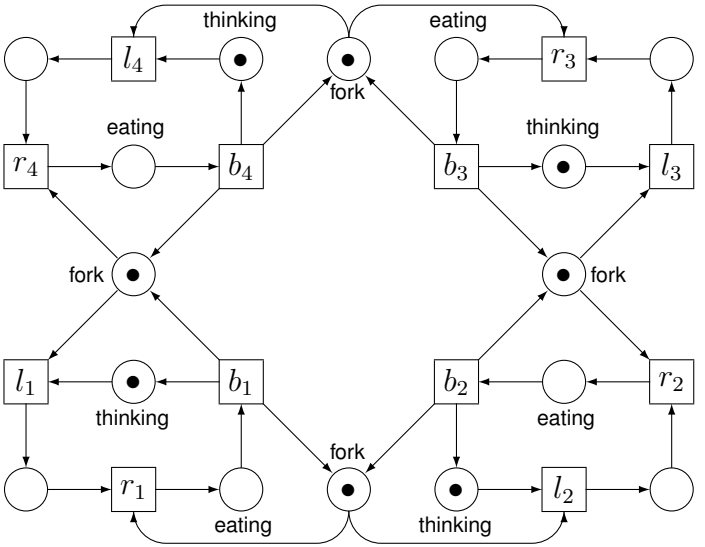


2.3 Dining philosophers

Four philosophers sit around a round table. There are forks on the table, one between each pair of philosophers. A philosopher needs both forks to eat.



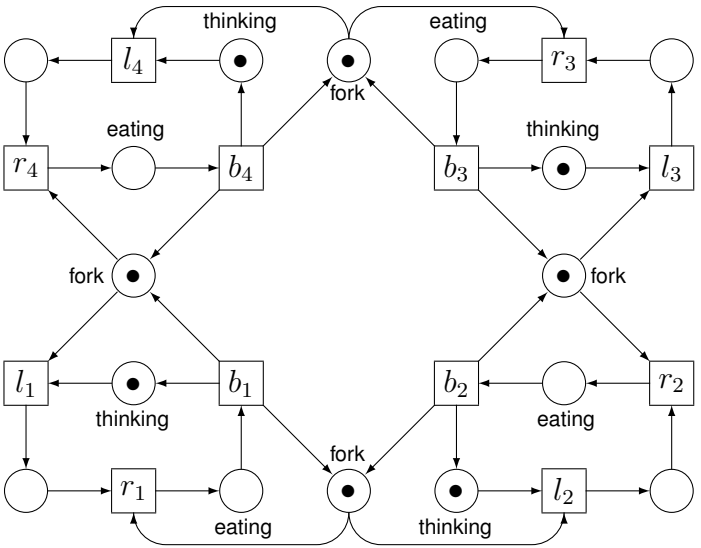
Philosophers agree to this protocol: Initially they think; when they get hungry, they first take the left fork, then take the right fork, and start eating; when they ae full, they return both forks simultaneously, and go back to thinking. Here is a Petri net model.



Two interesting questions about this system:

- Can the system reach a deadlock?
- After a philosopher picks the first fork, will he or she eventually eat?

Do they hold?



Training Exercise in AutomataTutor:
☆ 🔍 Construct Reachability Graph: Answer the first question by looking for a firing sequence where no philosopher can eat.

2.4 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?

We model the system with a Petri net.

The puzzle mentions:

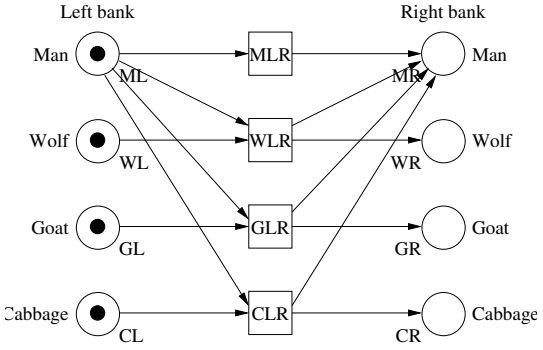
- *objects*: Man, wolf, goat, cabbage, boat. Both can be on either side of the river.

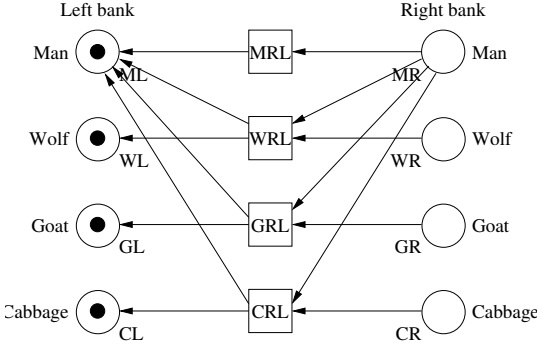
  Objects and their states are modeled by places.

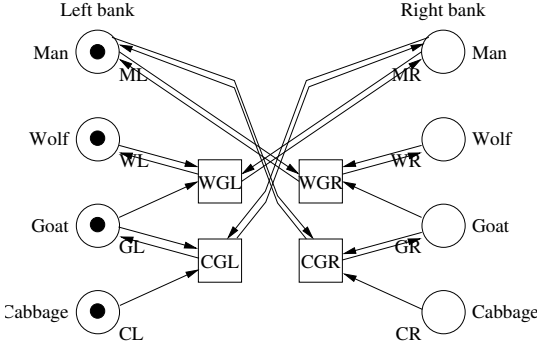
  (We can omit the boat, because it is always going to be on the same side as the man.)

- *actions*: Crossing the river, wolf eats goat, goat eats cabbage.

  Actions are modeled by transitions.


Left bank
Right bank
Man
Wolf
Goat
Cabbage
ML
WL
GL
CL
MLR
WLR
GLR
CLR
MR
WR
GR
CR
Man
Wolf
Goat
Cabbage


Left bank
Right bank
Man
Wolf
Goat
Cabbage
ML
WL
GL
CL
MRL
WRL
GRL
CRL
MR
WR
GR
CR
Man
Wolf
Goat
Cabbage


Left bank
Right bank
Man
Wolf
Goat
Cabbage
ML
WL
GL
CL
WGL
CGL
WGR
CGR
MR
WR
GR
CR
Man
Wolf
Goat
Cabbage

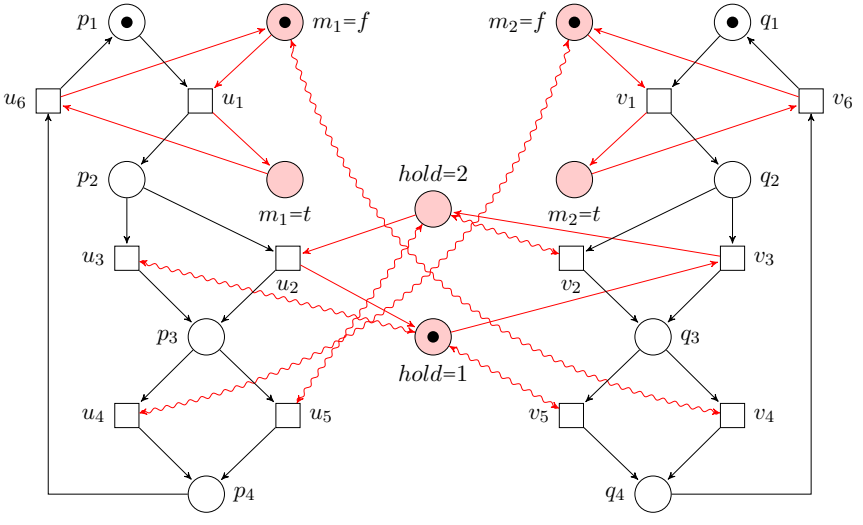
2.5 Peterson's algorithm

A solution to the mutual exclusion problem for two processes.

var $m_1, m_2 : \{false, true\}$, **init** $false$
$hold : \{1, 2\}$, **init** 1

while $true$ **do**	**while** $true$ **do**
p_1: $m_1 := true$	q_1: $m_2 := true$
p_2: $hold := 1$	q_2: $hold := 2$
p_3: **await**$(\neg m_2 \vee hold = 2)$	q_3: **await**$(\neg m_1 \vee hold = 1)$
(critical section)	(critical section)
p_4: $m_1 := false$	q_4: $m_2 := false$
od	**od**

Petri net model:



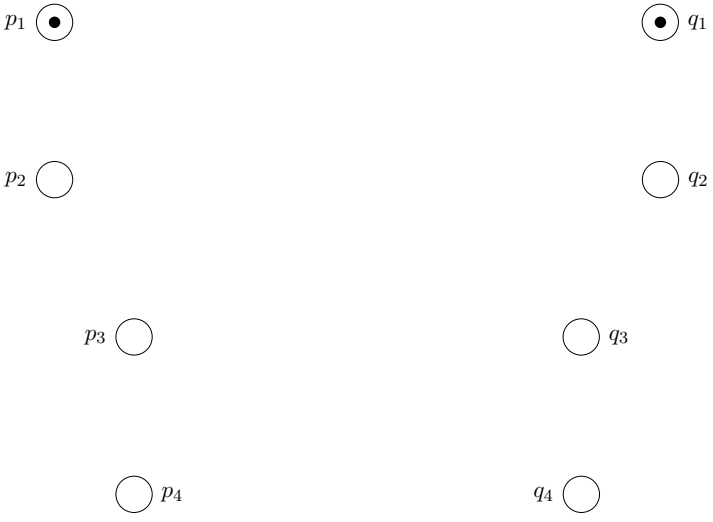
We show how to generate it.

Control places

var $m_1, m_2 : \{false, true\}$, **init** *false*
$hold : \{1, 2\}$, **init** 1

while *true* **do**
p_1: $m_1 := true$
p_2: $hold := 1$
p_3: **await**$(\neg m_2 \vee hold = 2)$
(critical section)
p_4: $m_1 := false$
od

while *true* **do**
q_1: $m_2 := true$
q_2: $hold := 2$
q_3: **await**$(\neg m_1 \vee hold = 1)$
(critical section)
q_4: $m_2 := false$
od

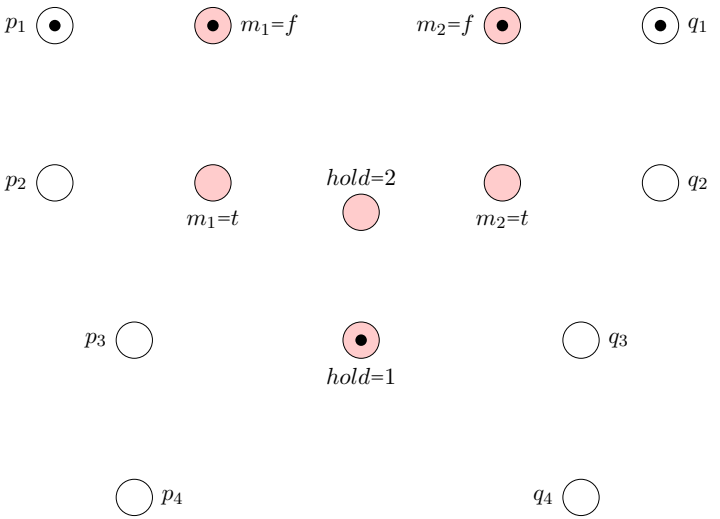


Variable places

var $m_1, m_2 : \{false, true\}$, **init** *false*
$hold : \{1, 2\}$, **init** 1

while *true* **do**
p_1: $m_1 := true$
p_2: $hold := 1$
p_3: **await**$(\neg m_2 \vee hold = 2)$
(critical section)
p_4: $m_1 := false$
od

while *true* **do**
q_1: $m_2 := true$
q_2: $hold := 2$
q_3: **await**$(\neg m_1 \vee hold = 1)$
(critical section)
q_4: $m_2 := false$
od



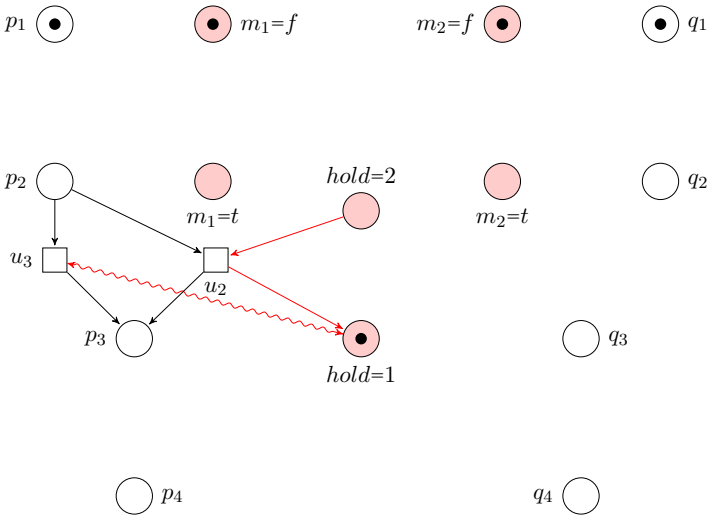
Modelling $hold := 1$

var $m_1, m_2 : \{false, true\}$, **init** *false*
$hold : \{1, 2\}$, **init** 1

while *true* **do**
p_1: $m_1 := true$
p_2: $hold := 1$
p_3: **await**$(\neg m_2 \vee hold = 2)$
(critical section)
p_4: $m_1 := false$
od

while *true* **do**
q_1: $m_2 := true$
q_2: $hold := 2$
q_3: **await**$(\neg m_1 \vee hold = 1)$
(critical section)
q_4: $m_2 := false$
od

The execution of $hold := 1$ is modeled by two transitions, depending on the previous value of $hold$.



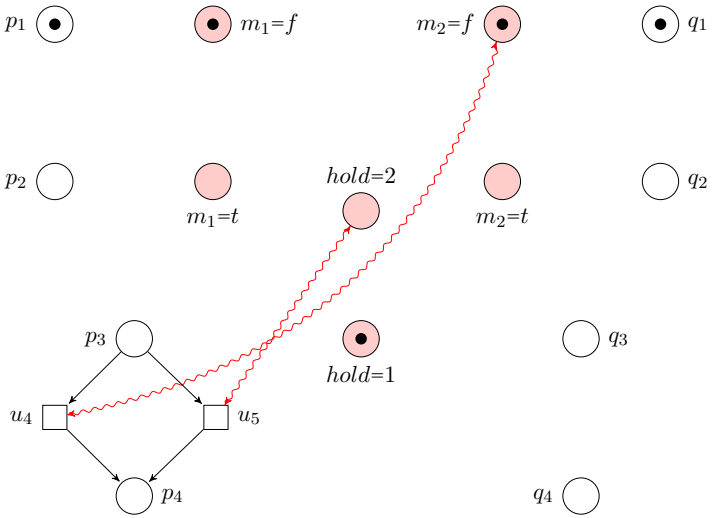
Modelling await$(\neg m_2 \vee hold = 2)$

var $m_1, m_2 : \{false, true\}$, **init** *false*
$hold : \{1, 2\}$, **init** 1

while *true* **do**
p_1: $m_1 := true$
p_2: $hold := 1$
p_3: **await**$(\neg m_2 \vee hold = 2)$
(critical section)
p_4: $m_1 := false$
od

while *true* **do**
q_1: $m_2 := true$
q_2: $hold := 2$
q_3: **await**$(\neg m_1 \vee hold = 1)$
(critical section)
q_4: $m_2 := false$
od

Again two transitions, due to the disjunction in the guard.



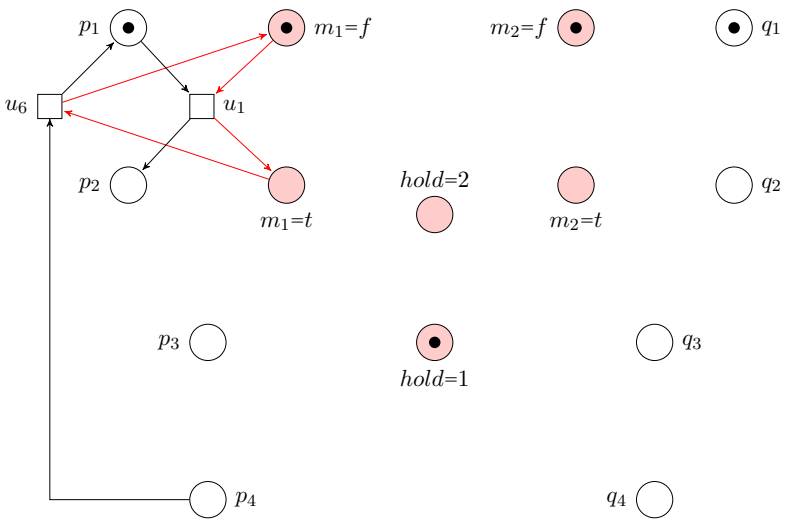
Modelling $m_1 := true$ and $m_1 := false$

var $m_1, m_2 : \{false, true\}$, **init** *false*
$hold : \{1, 2\}$, **init** 1

while *true* **do**
p_1: $m_1 := true$
p_2: $hold := 1$
p_3: **await**$(\neg m_2 \vee hold = 2)$
(critical section)
p_4: $m_1 := false$
od

while *true* **do**
q_1: $m_2 := true$
q_2: $hold := 2$
q_3: **await**$(\neg m_1 \vee hold = 1)$
(critical section)
q_4: $m_2 := false$
od

In principle we would need two transitions for each of $m_1 := true$, and $m_1 := false$, four in total, but we observe that two of them can never occur, and remove them to simplify the picture.

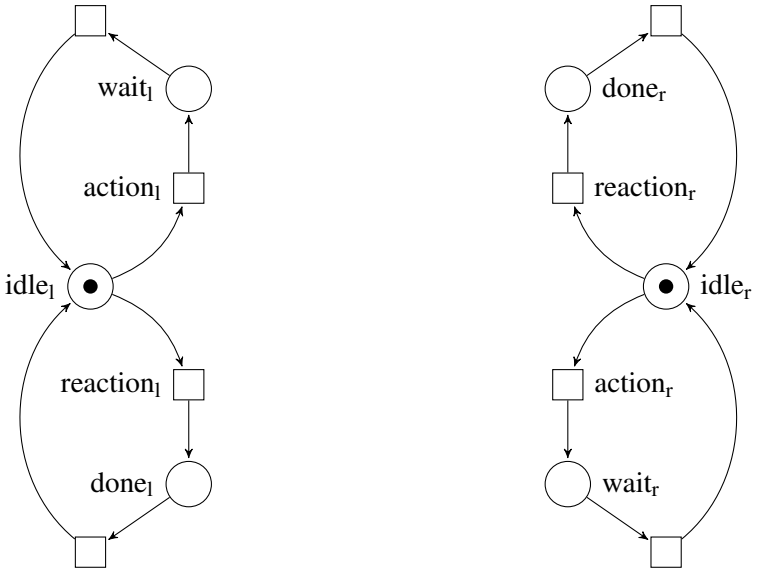


2.6 The action/reaction protocol

Two agents must repeatedly exchange informations.

When an agent requests an information from the other one, it must wait for an answer before proceeding.

Basic model:

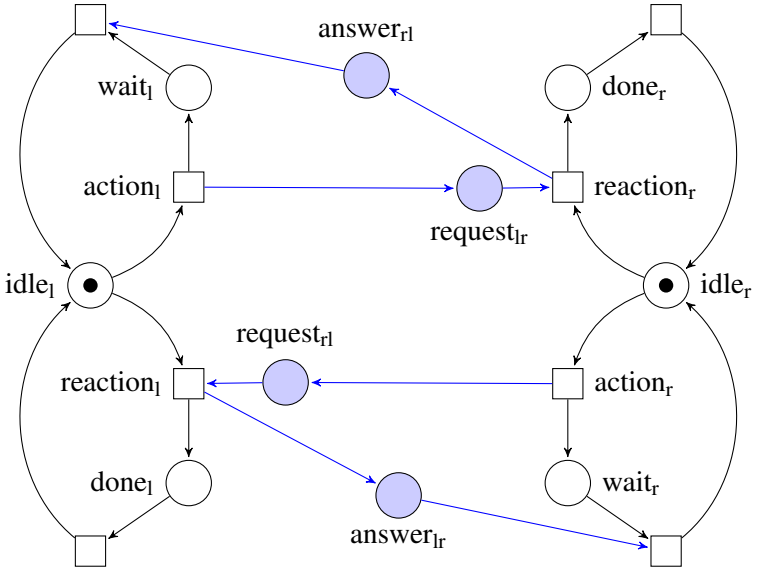


The task is to design a protocol for the exchanges.

In particular, the protocol must guarantee that it is never the case that both processes are waiting from an answer from the other one.

Training Exercise in AutomataTutor:
☆ 🔍 Action/reaction protocol 1: Find a firing sequence where the first attempt at the action/reaction protocol reaches a deadlock.

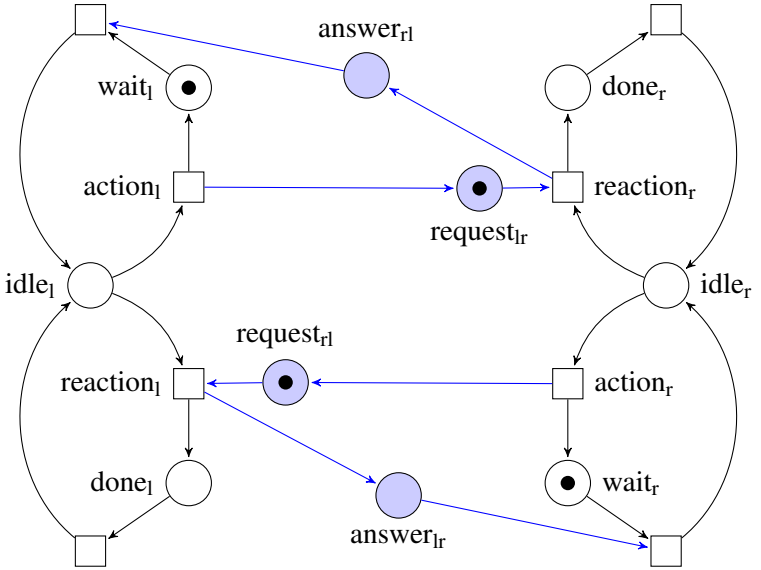
First attempt



This solution can reach a deadlock: both processes can issue a request simultaneously, after which they wait forever for an answer.

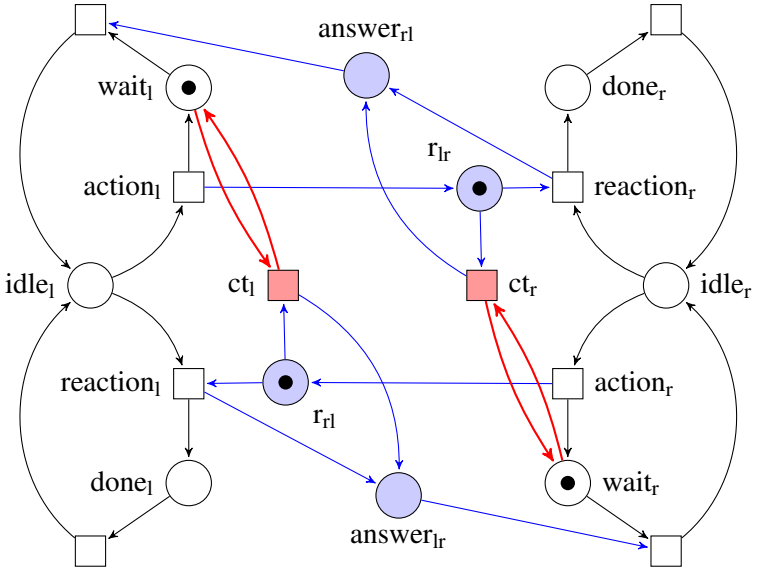
We call such a situation a crosstalk.

First attempt



Deadlocked crosstalk marking

Second attempt



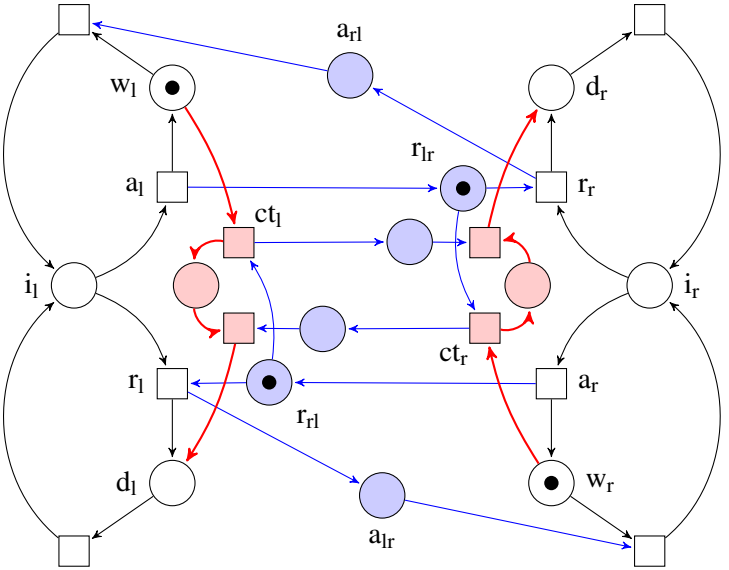
Processes can detect that a crosstalk has taken place.

If a process detects a crosstalk, it answers the request of its partner, and then continues to wait for an answer to its own request.

This solution has no deadlocks (prove it!), but a non-cooperative process can always get answers to its requests, without ever answering any request from its partner.

The solution is deadlock-free, but unfair.

Third attempt



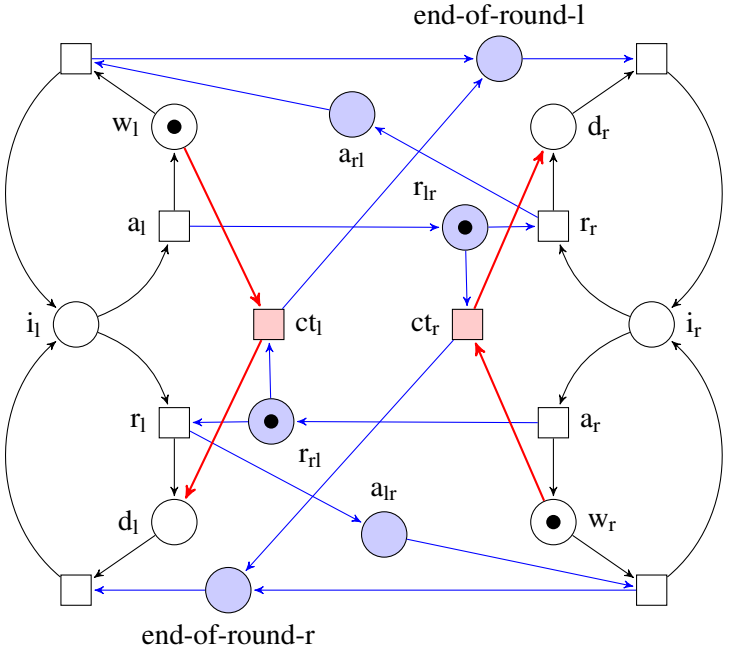
This attempt is fair.

If a process detects a crosstalk, it answers the request of its partner; however, after that it is only willing to receive an answer to its own question.

Unfortunately, the system has again a deadlock (can you find it?).

Training Exercise in AutomataTutor:
★ 🔍 Action/reaction protocol 2: Find a firing sequence where the third attempt at the action/reaction protocol reaches a deadlock.

Fourth attempt



This final attempt is both deadlock-free and fair.

The protocol works in rounds.

A “good” round consists of a request and an answer.

In a “bad” round both processes issue a request and they reach a crosstalk situation. Then both processes (i) detect the crosstalk, (ii) send each other an “end-of-round” signal, (iii) wait for the same signal from their partner, and (iv) move to their initial states.

The solution is not perfect. In the worst case there are only bad rounds, and no requests are answered at all.

2.7 Some variants of the main model

Definition 2.7.1 (Nets with place capacities)
A net with capacities $N = (S, T, F, K)$ consists of a net (S, T, F) and a mapping $K : S \to \mathbb{N}$.

A transition t is enabled at a marking M of N if

- $M(s) \geq 1$ for every place $s \in {}^\bullet t$ and
- $M(s) < K(s)$ for every place $s \in t^\bullet \setminus {}^\bullet t$

The notions of firing, Petri net with capacities, etc. are defined as in the capacity-free case.

Definition 2.7.2 (Nets with weighted arcs)
A net with weighted arcs $N = (S, T, W)$ consists of

- two disjoint sets S and T of places and transitions; and
- a weight function $W\colon (S \times T) \cup (T \times S) \to \mathbb{N}$.

A marking M enables a transition t if $M(s) \geq W(s,t)$ for every $s \in S$.

If M enables t then t can occur, leading to the marking M' defined by

$$M'(s) = M(s) + W(t,s) - W(s,t)$$

for every place s.

In Petri nets with weighted arcs, the preset and postset of a transition is not a set, but a multiset, i.e., a set that can contain multiple copies of an object.

If, say, $W(s,t) = 3$, then the preset of t contains 3 copies of the place s.

Training Exercises in Automata Tutor:

- ☆ Weighted arcs 1: Make a Petri net deadlock-free by changing the weights of the transitions.
- ★ Weighted arcs 2: Find a initial marking such that the Petri Net is deadlock free.

Definition 2.7.3 (Nets with inhibitor arcs)
A net with inhibitor arcs $N = (S, T, F, I)$ consists of a net (S, T, F) and an additional set $I \subseteq S \times T$ of inhibitor arcs.

A marking M enables a transition t if

- $M(s) > 0$ for every place s such that $(s, t) \in F$, and
- $M(s) = 0$ for every place s such that $(s, t) \in I$.

If M enables t then t can occur, leading to the marking M', defined as for standard Petri nets (M' depends only on F, not on I).

Definition 2.7.4 (Nets with reset arcs)
A net with reset arcs $N = (S, T, F, R)$ consists of a net (S, T, F) and an additional set $R \subseteq S \times T$ of reset arcs.

A marking M enables a transition t if $M(s) > 0$ for every place s such that $(s, t) \in F$.

If M enables t then t can occur, leading to the marking obtained after doing the folllowing:

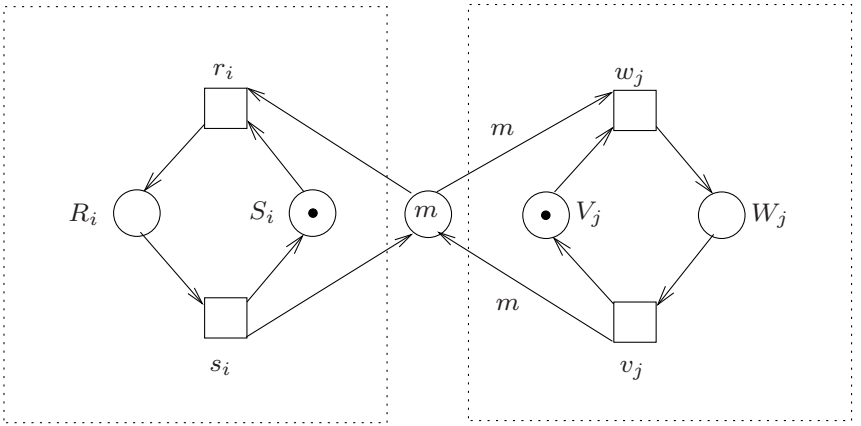
- Remove one token from every place s such that $(s, t) \in F$.
- Remove all tokens from every place s such that $(s, t) \in R$.
- Add one token to every place s such that $(t, s) \in F$.

2.8 Petri nets with weighted arcs: Some models

Readers and writers

A set of processes has access to a database.

Processes can read concurrently, but a process can only write if no other processes reads nor writes.



R_i:	Process i reads	W_i:	Process j writes
S_i:	Process i idle	V_j:	Process j idle
r_i:	Process i starts reading	w_j:	Process j starts writing
s_i:	Process i stops reading	v_j:	Process j stops writing

m readers n writers

Exercise: Modify the Petri net so that reading processes can not indefinitely prevent another process from writing.

Population protocols

Population protocols are a model of distributed computation by anonymous, identical, finite-state agents.

A population protocol consists of

- a set Q of states, and
- a set $T \subseteq Q^2 \times Q^2$ of transitions.

A transition $\big((q_1, q_2), (q_3, q_4)\big) \in T$ is denoted $q_1, q_2 \mapsto q_3, q_4$.

A configuration is a multiset of states.

A configuration, say C, such that $C(q_1) = 2$ and $C(q_2) = 1$, indicates that currently there are two agents in state q_1 and one agent in state q_2.

The Petri net modeling a protocol has one place for each state, and one transition for every transition of the protocol.

If a transition t of the Petri net models a transition $q_1, q_2 \mapsto q_3, q_4$ of the protocol, then

$$^\bullet t = \wr q_1, q_2 \wr \quad \text{and} \quad t^\bullet = \wr q_3, q_4 \wr$$

(Here $\wr q, q' \wr$ denotes the multiset containing one copy of q and one of q'. If $q = q'$, then $\wr q, q' \wr$ contains *two* copies of q. In particular, $\wr q, q \wr \neq \wr q \wr$.)

An agent in state q is modeled by a token in place q.

A configuration C with $C(q)$ agents in state q is modeled by the marking that puts $C(q)$ tokens in place q for every $q \in Q$.

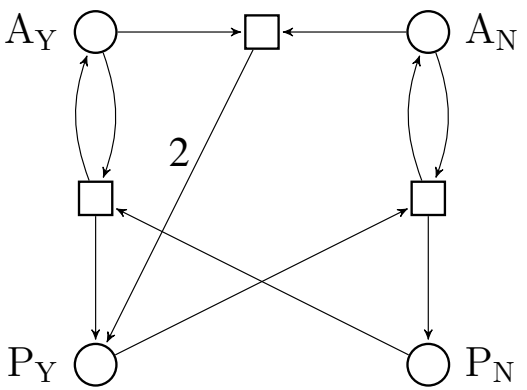
An example

States: $\mathrm{A_Y}, \mathrm{A_N}, \mathrm{P_Y}, \mathrm{P_N}$

Transitions:

Nr.	Transition
1	$\mathrm{A_Y}, \mathrm{A_N} \mapsto \mathrm{P_N}, \mathrm{P_N}$
2	$\mathrm{A}_\alpha, \mathrm{P}_\beta \mapsto \mathrm{A}_\alpha, \mathrm{P}_\alpha \quad \alpha, \beta \in \{\mathbf{Y}, \mathbf{N}\}$

Associated Petri net:



Informal definition of the predicate computed by a protocol

Population protocols are designed to compute predicates $\varphi\colon \mathbb{N}^k \to \{0,1\}$.

A protocol for φ has a distinguished set of input states $\{q_1, q_2, \ldots, q_k\} \subseteq Q$.

Each state of Q, initial or not, is labeled with an output, either 0 or 1.

Assume for example $k = 2$. In order to compute $\varphi(n_1, n_2)$, we first place n_i agents in q_i for $i = 1, 2$, and 0 agents in all other states.

This is the initial configuration of the protocol for the input (n_1, n_2). Then we let the protocol run.

The protocol satisfies the following property for every input (n_1, n_2):

> there exists $b \in \{0,1\}$ such that in every fair run starting at the initial configuration for (n_1, n_2) (fair runs are defined in the next slide), eventually all agents reach states labeled with b, and stay in such states forever.

So, intuitively, in all fair runs from (n_1, n_2) all agents eventually "agree" on the boolean value b.

By definition, b is the result of the computation, i.e, the $\varphi(n_1, n_2) = b$.

Formal definition of the predicate computed by a protocol

Fix a Petri net $N = (S, T, W)$ with $|{}^\bullet t| = 2 = |t^\bullet|$ for every transition t. (The net of a procotol.)

Fix a set $I = \{p_1, \ldots, p_k\}$ of input places, and a function $O \colon P \to \{0, 1\}$.

A marking M of N is a b-consensus if $M(p) > 0$ implies $O(p) = b$.

A b-consensus M is stable if every marking reachable from M is also a b-consensus.

A firing sequence $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \cdots$ of N is fair if

- it is finite and leads to a deadlock marking; or
- it is infinite and the following condition holds for all markings M, M' and $t \in T$:

$$\text{if } M \xrightarrow{t} M' \text{ and } M = M_i \text{ for infinitely many } i \geq 0$$

$$\text{then}$$

$$M_j \xrightarrow{t_{j+1}} M_{j+1} \;=\; M \xrightarrow{t} M' \text{ for infinitely many } j \geq 0.$$

N computes the value b for the input $\mathbf{v} = (n_1, \ldots, n_k)$ if the initial marking $M_\mathbf{v}$ given by $M_\mathbf{v}(p_i) = n_i$ for every $1 \leq i \leq k$ and $M_\mathbf{v}(p) = 0$ for every $P \setminus I$ satisfies:

every fair firing sequence starting at $M_\mathbf{v}$ reaches a b-consensus.

An incorrect protocol for the majority predicate $Y > N$

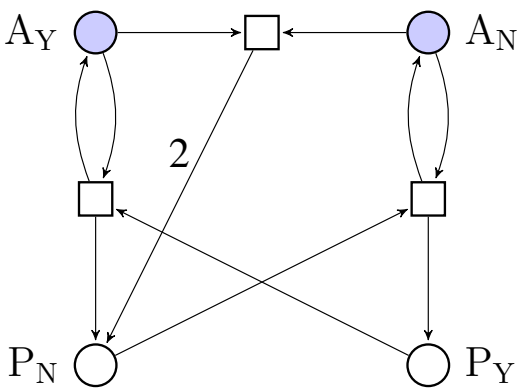
States: $\mathrm{A_Y}, \mathrm{A_N}, \mathrm{P_Y}, \mathrm{P_N}$

Transitions:

Nr.	Transition
1	$\mathrm{A_Y}, \mathrm{A_N} \mapsto \mathrm{P_N}, \mathrm{P_N}$
2	$\mathrm{A}_\alpha, \mathrm{P}_\beta \mapsto \mathrm{A}_\alpha, \mathrm{P}_\alpha \quad \alpha, \beta \in \{\mathbf{Y}, \mathbf{N}\}$

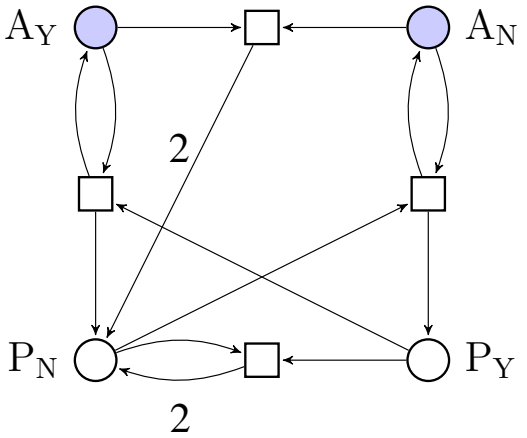
Initial states: $\mathrm{A_Y}, \mathrm{A_N}$

Associated Petri net:



A correct but slow protocol for majority.

Nr.	Transition
1	$\mathrm{A_Y}, \mathrm{A_N} \mapsto \mathrm{P_N}, \mathrm{P_N}$
2	$\mathrm{A}_\alpha, \mathrm{P}_\beta \mapsto \mathrm{A}_\alpha, \mathrm{P}_\alpha \quad \alpha, \beta \in \{\mathbf{Y}, \mathbf{N}\}$
3	$\mathrm{P_N}, \mathrm{P_Y} \mapsto \mathrm{P_N}, \mathrm{P_N}$



A correct and efficient protocol for majority.

Nr.	Transition
1	$\mathrm{A_Y}, \mathrm{A_T} \mapsto \mathrm{A_Y}, \mathrm{P_Y}$
2	$\mathrm{A_Y}, \mathrm{A_N} \mapsto \mathrm{A_T}, \mathrm{P_T}$
3	$\mathrm{A_T}, \mathrm{A_N} \mapsto \mathrm{A_N}, \mathrm{P_N}$
4	$\mathrm{A_T}, \mathrm{A_T} \mapsto \mathrm{A_T}, \mathrm{P_T}$
5	$\mathrm{A}_\alpha, \mathrm{P}_\beta \mapsto \mathrm{A}_\alpha, \mathrm{P}_\alpha \quad \alpha, \beta \in \{\mathrm{Y}, \mathrm{T}, \mathrm{N}\}$

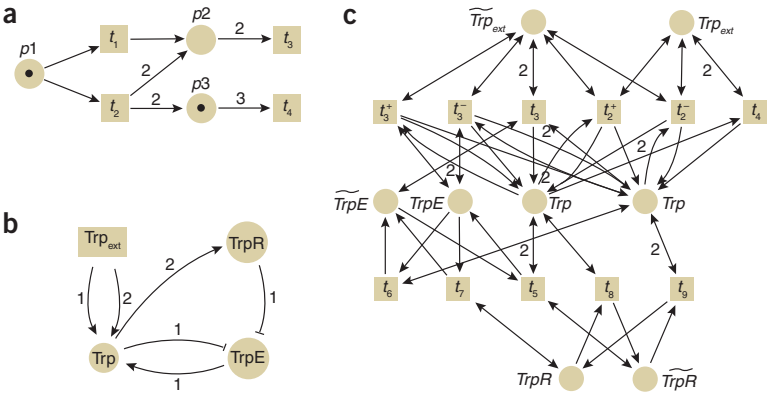
2.9 Three Petri net models from the literature

A biological system.

In biology models, tokens represent molecules or cells, and transitions correspond to chemical reactions or biological processes.

This Petri net is taken from

"Executable cell biology", by J. Fisher and T.A. Henzinger (Nature biotechnology, 2007).



Part (a) is a standard weighted Petri net.

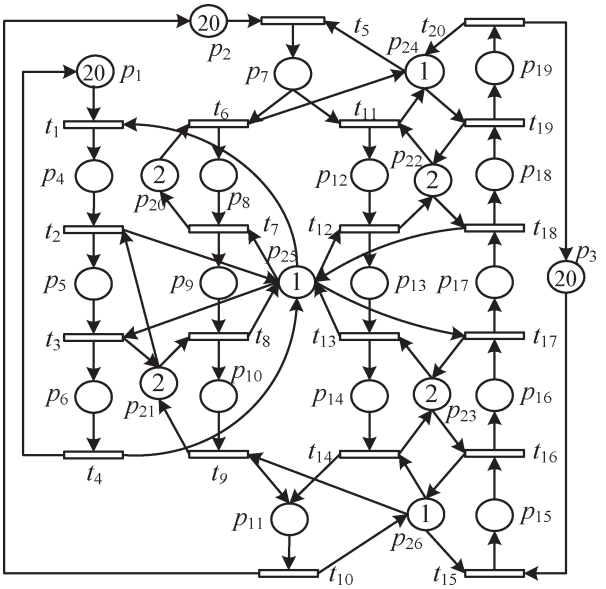
Part (b) shows a simplified logical regulatory graph for the biosynthesis of tryptophan in *E. coli*. Each node of the regulatory graph represents an active component: tryptophan (Trp), the active enzyme (TrpE) and the active repressor (TrpR). The node marked by a rectangle accounts for the import of Trp from external medium. Arrows represent activation and bars denote inhibition (inhibitor arcs).

Part (c) shows the Petri net of the Trp regulatory network.

A flexible manufacturing system.

This Petri net is taken from

"Optimal Petri-Net-Based Polynomial-Complexity Deadlock-Avoidance Policies for Automated Manufacturing Systems" by Xing *et al.* (IEEE Trans. on Systems, Man, and Cybernetics, 2009).



It shows a flexible manufacturing cell with four machines, modeled by places p_{20} to p_{23}, and three robots, modeled by places p_{24} to p_{26}.

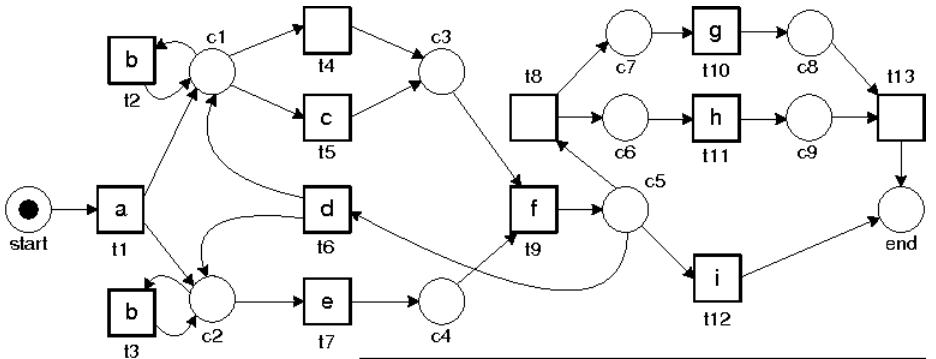
Tokens model parts, and so, for example, a token at p_{20} means that the part represented by the token is currently being processed at the first machine.

Each machine can hold two parts at the same time, and each robot can hold one part.

A business process.

This Petri net is taken from

“Business process management as the “Killer App” for Petri nets” by van der Aalst (Software and Systems Modeling, 2014).



It shows a Petri net model of the life-cycle of a request for compensation.

A transition may carry a label referring to some activity.

Transitions without a label are “silent”.

2.10 Analysis problems

We introduce formal properties capturing questions of interest about the models of the chapter.

Definition 2.10.1 (System properties)
Let (N, M_0) be a Petri net with at least one place and one transition.

(N, M_0) is deadlock free if every reachable marking enables at least one transition (that is, no reachable marking is dead).

(N, M_0) is live if for every reachable marking M and every transition t there is a marking $M' \in [M\rangle$ that enables t. (Intuitively: every transition can always fire again).

(N, M_0) is bounded, if for every place s there is a number $b \geq 0$ such that $M(s) \leq b$ for every reachable marking M.

M_0 is a bounded marking of N if (N, M_0) is bounded.

The bound of a place s of a bounded Petri net (N, M_0) is the number

$$max\{M(s) \mid M \in [M_0\rangle\}$$

(N, M_0) is b-bounded if every place has bound at most b.

Problems studied throughout the course

- **Deadlock freedom**: is a given Petri net (N, M_0) deadlock-free?
- **Liveness**: is a given Petri net (N, M_0) live?
- **Boundedness**: is a given Petri net (N, M_0) bounded?
- **b-boundedness**: given $b \in \mathbb{N}$ and a Petri net (N, M_0), is (N, M_0) b-bounded?
- **Reachability**: given a Petri net (N, M_0) and a marking M of N, is M reachable?
- **Coverability**: given a Petri net (N, M_0) and a marking M of N, is there a reachable marking $M' \geq M$?

Simple connections:

Proposition 2.10.2

(1) Liveness implies deadlock freedom.

(2) If (N, M_0) is bounded then there is a number b such that (N, M_0) is b-bounded.

(3) If (N, M_0) is bounded, then it has finitely many reachable markings.

Proof. (1) follows immediately from the definitions. (2) and (3) follow from the definitions and from the fact that a Petri net has finitely many places. □

Sometimes we also use the following notion

Definition 2.10.3 (Well-formed nets) A net N is well formed if there is a marking M_0 such that the Petri net $(N, M_0)$is live and bounded.

and consider the following problem:

- **Well-formedness**: is a given net well formed?

Part II

Analysis Techniques for Petri Nets

Chapter 3

Decision procedures

3.1 Decision procedures for bounded Petri nets

A bounded Petri net has finitely many reachable markings, and so the reachability graph can be computed and stored, at least in principle.

If the reachability graph is available, then it is easy to give algorithms for b-Boundedness, Coverability, Reachability, and Deadlock-freedom running in linear time in the size of the reachability graph.

We show now that this is also the case for Liveness.

Deciding liveness of a bounded Petri net

Let $G = (V, E)$ be the reachability graph of a Petri net (N, M_0).

Define the equivalence relation $\overset{*}{\longleftrightarrow} \subseteq V \times V$ as follows:

$$M \overset{*}{\longleftrightarrow} M' \text{ gdw. } M \overset{*}{\longrightarrow} M' \text{ and } M' \overset{*}{\longrightarrow} M \ .$$

A strongly connected component (SCC) of G is a graph (V', E') where $V' \subseteq V$ is an equivalence class of of $\overset{*}{\longleftrightarrow}$ and $E' = E \cap (V' \times V')$.

Strongly connected components are partially ordered by the relation $<$ defined as follows:

$(V', E') < (V'', E'')$ iff $V' \neq V''$ and $\forall M' \in V', M'' \in V'' : M'' \in [M'\rangle$.

The bottom SCCs of G are the maximal SCCs of G with respect to $<$.

Proposition 3.1.1 *Let (N, M_0) be a bounded Petri net.*

(N, M_0) is live iff for every bottom SCC of the reachability graph of (N, M_0) and for every transition t, some marking of the SCC enables t.

Proof. ($\Rightarrow$) Assume (N, M_0) is live.

Let M be a marking of a bottom SCC of the reachability graph of (N, M_0), and let t be a transition of N.

By the definition of liveness, some marking reachable from M enables t.

By the definition of bottom SCC, this marking belongs to the same bottom SCC as M.

($\Leftarrow$) Assume that for every bottom SCC of the reachability graph of (N, M_0) and for every transition t, some marking of the SCC enables t.

We show that (N, M_0) is live.

Let M be an arbitrary marking reachable from M_0, and let t be a transition.

By the definition of a bottom SCC, there is a bottom SCC such that every marking of it is reachable from M.

Since some marking of the SCC enables t, we are done. □

The condition of Proposition **??** can be checked in linear time using Tarjan's algorithm, which computes all the SCCs of a directed graph in linear time. The algorithm can be easily adapted to compute the bottom SCCs.

3.1.1 Complexity for 1-bounded Petri nets

Membership in PSPACE

A 1-bounded Petri net with n places may have up to 2^n reachable markings.

Therefore, all algorithms based on the construction of the reachability graph have exponential worst-case runtime. They only show that the problems are in **EXPTIME**.

Using Savitch's theorem we can show that they are in **PSPACE**. For example, the following polynomial-memory nondeterministic program solves *Reachability*:

Input: a 1-bounded Petri net (N, M_0), a goal marking M_g (a marking whose reachability should be checked).

The program has a variable M; initially $M = M_0$.

A 1-bounded marking can be stored using 1 bits per place. So M uses linear space in the input.

While $M \neq M_g$, the program nondeterministically chooses a transition t enabled at M, computes the marking M' such that $M \xrightarrow{t} M'$, and sets $M := M'$.

If the marking is not reachable, this program does not terminate. If you want the program to always terminate, add a n-bit counter that counts the number of steps.

Since the Petri net has at most 2^n reachable markings, if M is reachable then it is reachable in at most $2^n - 1$ steps.

If the counter reaches the value $2^n - 1$ without reaching the marking M, the program stops.

PSPACE-hardness

We now show that Coverability, Reachability, and Deadlock-freedom, and Liveness are PSPACE-hard, and so PSPACE-complete.

Turing machines and linearly bounded automata. A (deterministic) Turing machine is a tuple $(Q, \Gamma, \delta, q_0, F)$, where

- Q is the set of states
- Γ is the set of tape symbols (containing a special blank symbol B),
- $\delta\colon (Q \times \Gamma) \to Q \times \Gamma \times \{R, L\}$ is the transition function,
- q_0 the initial state, and
- F the set of final states.

The size of a Turing machine is the number of bits needed to encode its transition relation.

A linearly bounded automaton is a Turing machine that can only work on a tape with n cells, where n is the size of the machine.

We assume that two special first and last cells are marked with two special symbols, say \$ and $\#$, and that the transition function guarantees that the head never moves to the left of the first cell or to the right of the last cell.

Most questions about the computations of linearly bounded automata are PSPACE-hard.

The acceptance problem is PSPACE-complete:

> Given: a linearly bounded automaton A of size n
>
> Decide: Does A accept, i.e., does it accept from the configuration $\$\, q_0\, B^n\, \#$?

The problem remains PSPACE-complete even for automata with one single accepting state or one single accepting configuration.

The acceptance problem can be easily reduced to many other problems in polynomial time, so they are PSPACE-hard too. Examples are:

- does A halt?,
- does A visit a given state?,
- does A visit a given configuration?
- does A visit a given configuration infinitely often?

Simulating linearly bounded automata with 1-bounded Petri nets.

> A linearly bounded automaton of size n can be simulated by a 1-bounded Petri net of size $O(n^2)$. Moreover, there is a polynomial time procedure which constructs this Petri net.

Let $A = (Q, \Gamma, \delta, q_0, F)$ be a linearly bounded automaton of size n.

Let $C = \{c_1, \ldots, c_n\}$ be the set set of cells of A.

We define the simulating Petri net $N(A)$.

Places. $N(A)$ has:

- a place $s(q)$ for each state $q \in Q$,
- a place $s(c)$ for each cell $c \in C$,
- a place $s(a, c)$ for each symbol $a \in \Gamma$ and for each cell $c \in C$.

A token on $s(q)$ signals that A is currently is in state q.

A token on $s(c)$ signals that A currently reads the cell c.

A token on $s(a, c)$ signals that the cell c contains the symbol a.

The total number of places is $|Q| + n \cdot (1 + |\Gamma|)$.

Transitions. The transitions of $N(A)$ are determined by the transition relation δ.

For example, if $\delta(q, a) = (q', a', R)$, then, for each cell c_i but the last, $N(A)$ has a transition $t(q, a, c_i)$ with

- input places $s(q)$, $s(c_i)$, and $s(a, c_i)$, and
- output places $s(q')$, $s(a', c_i)$, and $s(c_{i+1})$.

The total number of transitions is bounded by $|Q| \cdot |\Gamma| \cdot n$.

Since the size of A is $O(|Q| \cdot |\Gamma|)$, the number of transitions is $O(n^2)$.

Initial marking. The initial marking of $N(A)$ puts

- one token on $s(q_0)$,
- one token on $s(c_1)$, and
- one token on the place $s(B, c_i)$ for $1 \leq i \leq n$ (recall: B denotes the blank symbol).

The total size of the Petri net is $O(n^2)$.

Each move of A corresponds to the firing of one transition.

The configurations reached by A along a computation correspond to the markings of $N(A)$ reached along its corresponding run.

These markings put one token in exactly one of the places $\{s(q) \mid q \in Q\}$, in exactly one of the places $\{s(c) \mid c \in C\}$, and in exactly one of the places $\{s(a, c) \mid a \in \Gamma\}$ for each cell $c \in C$.

So $N(A)$ is 1-bounded.

In order to answer a question about a linearly bounded automaton A we can construct the net $N(A)$, which is only polynomially larger than A, and solve the corresponding question about the runs of A.

For instance, the question "does the computation of A terminate?" corresponds to "has $N(A)$ a deadlock?"

Since $N(A)$ is only polynomially larger than A: every PSPACE-hard problem for linearly bounded automata translates into a PSPACE-hard problem for 1-bounded Petri nets.

For instance:

- A reduction from "does A ever visit a given control state?" proves PSPACE-hardness of Coverability for 1-bounded Petri nets.
- A reduction from "does A ever visit a given configuration?" proves PSPACE-hardness of Reachability for 1-bounded Petri nets.

Further PSPACE-hard problems can be obtained by reduction from Coverability or Reachability. For instance, Coverability can be reduced (exercise!) to the following problems :

- is there a reachable marking that concurrently enables two given transitions t_1 and t_2?
- can a given transition t ever occur?
- is there a run containing a given transition t infinitely often?

More difficult exercise: prove that Liveness of 1-bounded nets is also PSPACE-hard.

3.2 Decision procedures for general Petri nets

We study the decidability and complexity of Boundedness, Coverability, Reachability, Deadlock-freedom and Liveness for general Petri nets, not necessarily bounded.

The algorithms of the bounded case no longer work, because the construction of the reachability graph may not terminate.

3.2.1 A decision procedure for Boundedness

The b-Boundedness problem is clearly decidable: if the input Petri net (N, M_0) has n places, then the number of b-bounded markings of N is n^{b+1}.

So we can decide b-Boundedness by constructing the reachability graph of (N, M_0) until either the construction terminates, or we find a reachable marking that is not b-bounded.

The same idea gives a semi-decision procedure for Boundedness: again, we construct the reachability graph. If the input (N, M_0) is bounded, then there are finitely many reachable markings, the construction terminates, and we can return "bounded".

However, if the net is unbounded then this procedure does not terminate.

We now give a decision procedure for Boundedness.

We need two lemmas: (a variant of) König's lemma and Dickson's lemma.

Lemma 3.2.1 (Königs lemma) *Let $G = (V, E)$ be the reachability graph of a Petri net (N, M_0).*

If V is infinite, then G contains an infinite (simple) path.

Proof. Assume $V = [M_0\rangle$ is infinite.

For every reachable marking M, the graph G contains isa simple path π_M leading from M_0 to M.

We have:

- M_0 has finitely many immediate successors.
  (At most one for each transition of N).
- Each simple path π_M visits (at least) one immediate successor of M_0.

It follows: at least one immediate successor M_1 is visited by infinitely many paths π_M.

So M_1 has infinitely many successors in the graph $(V \setminus \{M_0\}, E)$. In other words: $[M_1\rangle \setminus \{M_0\}$ is infinite.

Applying the same argument to M_1 we construct an immediate successor M_2 of $(V \setminus \{M_0, M_1)\}, E)$.

Iterating we construct an infinite simple path $M_0M_1M_2 \ldots$ □

Lemma 3.2.2 (Dickson's lemma) *For every infinite sequence $A_1A_2A_3\ldots$ of vectors of $\mathbb{N}^k$ there is an infinite sequence $i_1 < i_2 < i_3 \ldots$ of indexes such that $A_{i_1} \leq A_{i_2} \leq A_{i_3} \ldots$.*

Proof. By induction on k.

Basis: $k = 1$.

Then the elements of $\mathcal{A}$ are just numbers.

The set $\{A_1, A_2, \cdots\}$ has a minimum, say c_1. Choose i_1 as some index (say, the smallest), such that $A_{i_1} = c_1$.

Consider now the set $\{A_{i_1+1}, A_{i_1+2}, \cdots\}$. The set has a minimum c_2, which by definition satisfies $c_1 \leq c_2$.

Choose i_2 as the the smallest index $i_2 > i_1$ such that $A_{i_2} = c_2$, etc.

Step: $k > 1$.

Given a vector A_i, let A'_i be the vector of dimension $k-1$ consisting of the first $k-1$ components of A_i, and let a_i be the last component of A_i.

We write $A_i = (A'_i \mid a_i)$.

Since the vectors of $A'_1A'_2A'_3\cdots$ have dimension $k-1$, by induction hypothesis there is an infinite subsequence $A'_{i_1} \leq A'_{i_2} \leq A'_{i_3} \cdots$.

Consider now the sequence $a_{i_1}a_{i_2}a_{i_3}\cdots$. By induction hypothesis there is a subsequence $a_{j_1} \leq a_{j_2} \leq a_{j_3} \cdots$.

But then we have $A_{j_1} \leq A_{j_2} \leq A_{j_3} \cdots$, and we are done. □

Dickson's lemma shows that the partial order $\leq \subseteq \mathbb{N}^k \times \mathbb{N}^k$ is a well-quasi-order:

Given a set A, and a partial order $\preceq \subseteq A \times A$, we say that $\preceq$ is a well-quasi-order if every infinite sequence $a_1a_2a_3\cdots \in A^\omega$ contains an infinite chain $a_{i_1} \preceq a_{i_2} \preceq \cdots$.

Theorem 3.2.3 *(N, M_0) is unbounded iff there are markings M and L such that $L \neq 0$ and $M_0 \xrightarrow{*} M \xrightarrow{*} (M + L)$*

Proof. $(\Leftarrow)$: Assume there are such markings M, L.

By the Monotonicity Lemma we have

$$M_1 \xrightarrow{*} (M_1 + L) \xrightarrow{*} (M_1 + 2 \cdot L) \xrightarrow{*} \ldots$$

Since $L \neq 0$, the set $[M_0\rangle$ of reachable markings is infinite and (N, M_0) is unbounded.

$(\Rightarrow)$ Assume (N, M_0) is unbounded.

Then the set $[M_0\rangle$ of reachable markings is infinite.

By Königs lemma there is an infinite firing sequence $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_1} M_2 \ldots$ such that the markings $M_0, M_1, M_2, \ldots$ are pairwise distinct.

By Dickson's Lemma there are indexes $i < j$ such that $M_0 \xrightarrow{*} M_i \xrightarrow{*} M_j$ and $M_i \leq M_j$.

Choose $M := M_i$ and $L := M_j - M_i$.

Since M_i and M_j are distinct, we have $L \neq 0$. $\square$

Theorem 3.2.4 *Boundedness is decidable.*

Proof. We give an algorithm that always terminates and always returns the correct answer: " bounded" or "unbounded".

The algorithm explores the reachability graph of the input Petri net (N, M_0) using breadth-first search.

After adding a new marking M', the algorithm checks if the part of the graph already constructed contains a sequence $M_0 \xrightarrow{*} M \xrightarrow{*} M'$ such that $M \leq M'$ (and $M \neq M'$, because M' is new).

The algorithm terminates if it finds such a sequence, in which case it returns "unbounded", or if it cannot add any new marking, in which case it returns "bounded".

We show that the algorithm is correct.

- If (N, M_0) is bounded, then by Theorem **??** the algorithm never finds a new marking M' satisfying the condition above.

  So, since the Petri net has only finitely many reachable markings, the algorithm terminates because it cannot find any new marking, and correctly returns "bounded".

- If (N, M_0) is unbounded, then there are infinitely many reachable markings, and so the algorithm never runs out of reachable markings.

  By Theorem **??** the algorithm eventually finds distinct markings $M' \leq M\ M_0 \xrightarrow{*} M \xrightarrow{*} M'$, and so it correctly answers "unbounded".

□

3.2.2 Decision procedures for Coverability

We present three different algorithms to decide, given (N, M_0) and M, whether M can be covered from M_0:

- The coverability graph algorithm.

  The algorithm constructs the so-called coverability graph of a Petri net (N, M_0).

  For every marking M, one can decide if M can be covered from (N, M_0) by inspecting the graph.

- Rackoff's algorithm.

  The algorithm constructs the reachability graph, starting from M_0, but only until a certain depth that depends on M.

- The backwards reachability algorithm.

  The algorithm constructs (a finite representation of) the set of all markings from which one can cover M, and checks if M_0 belongs to it.

Why three algorithms?

- Coverability graph.

  Advantage: Once the graph is constructed, all coverability questions can be checked efficiently in the size of the graph.

  Disadvantage: In the worst case the graph can be HUGE.

- Rackoff's algorithm.

  Advantage: Worst-case complexity matches the theoretical complexity of the problem.

  Disadvantage: Very inefficient in practice.

- The backwards reachability algorithm.

  Advantage: Worst-case complexity matches the theoretical complexity of the problem. Can be extended to more general models.

  Disadvantage: Often less efficient in practice than the construction of the coverability graph.

Coverability graphs

We show how to construct a coverability graph of a Petri net (N, M_0).

The coverability graph is always finite, and satisfies the following property:

> a marking M of N is coverable from M_0 iff some node M' of the coverability graph of (N, M_0) covers M, i.e., satisfies $M' \geq M$.

ω-markings

We introduce a new symbol ω, standing for an arbitrarily large number.

We extend arithmetic on natural numbers with ω. For all $n \in \mathbb{N}$:

$$\begin{array}{ll} n + \omega = \omega + n = \omega & 0 \cdot \omega = 0 \\ \omega + \omega = \omega & n \geq 1 \Rightarrow n \cdot \omega = \omega \cdot n = \omega \\ \omega - n = \omega & n \leq \omega \text{ and } \omega \leq \omega \end{array}$$

Observe that $\omega - \omega$ remains undefined (we won't need it).

An ω-marking of a net $N = (S, T, F)$ is a mapping $M \colon S \to \mathbb{N} \cup \{\omega\}$.

Intuitively, in an ω-marking, each place has either a certain number of tokens or "arbitrarily many" tokens.

The enabling condition and the firing rule for ω-markings are as for normal markings.

In particular, if a place contains ω tokens, then after firing a transition it still has ω tokens, even if the transition is connected with an arc to the place.

Intuition behind the coverability graph

Assume $M' \in [M\rangle$ and $M \leq M'$.

Then $M \xrightarrow{t_1 t_2 \ldots t_n} M'$ for some sequence $t_1 t_2 \ldots t_n$ of transitions.

By the Monotonicity Lemma, there is a marking M'' with $M' \xrightarrow{t_1 t_2 \ldots t_n} M''$.

Further, letting $\Delta := M' - M$ we have $M'' = M' + \Delta = M + 2\Delta$

$$\begin{array}{ccccccc} M & \xrightarrow{t_1 t_2 \cdots t_n} & M' & \xrightarrow{t_1 t_2 \cdots t_n} & M'' & \longrightarrow & \cdots \\ & & \| & & \| & & \| \\ & & M + \Delta & & M + 2\Delta & & \cdots \end{array}$$

Firing $t_1 t_2 \ldots t_n$ repeatedly we can "pump" an arbitrary number of tokens to all places s such that $\Delta(s) > 0$.

Main idea for the construction:

replace $M \xrightarrow{t_1 t_2 \cdots t_n} M'$ by $M \xrightarrow{t_1 t_2 \cdots t_n} M' + \omega \cdot \Delta$

Training Exercise in AutomataTutor:
Construct Coverability Graph: Construct the complete coverability graph of the given Petri net step-by-step with the algorithm on the next page.

Algorithm for the construction of the coverability graph

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

ADDOMEGAS(M, M', V, E)
1 **for** $M'' \in V$
2 **do if** $M'' < M'$ and $M'' \xrightarrow{*}_E M$
3 **then** $M' := M' + (M' - M'') \cdot \omega$;
4 **return** M';

Notations used in ADDOMEGAS:

- $M'' \rightarrow_E M$ iff $(M'', t, M) \in E$ for some $t \in T$.
- $M'' \xrightarrow{*}_E M$ iff

$$M'' = M' \text{ or } M'' \rightarrow_E M_1 \rightarrow_E \cdots \rightarrow_E M_n = M$$

for some $n \geq 1$ and some $M_1, \ldots, M_n$.

We show that

- COVERABILITY-GRAPH terminates.
- A marking M of (N, M_0) is coverable iff some node M' of the coverability graph of (N, M_0) covers M.

Theorem 3.2.5 COVERABILITY-GRAPH *terminates.*

Proof. Assume that COVERABILITY-GRAPH does not terminate. We derive a contradiction.

If COVERABILITY-GRAPH does not terminate, then it constructs an infinite graph.

Since every node of the graph has at most $|T|$ successors, by König's lemma the graph contains an infinite path $\Pi = M_1 M_2 \ldots$.

If an ω-marking M_i of Π satisfies $M_i(p) = \omega$ for some place p, then $M_{i+1}(p) = M_{i+2}(p) = \ldots = \omega$.

So Π contains a marking M_j such that all markings $M_{j+1}, M_{j+2}, \ldots$ have ω's at exactly the same places as M_j.

Let Π' be the suffix of Π starting at M_j.

Consider the projection $\Pi'' = m_j m_{j+1} \ldots$ of Π' onto the non-ω places.

Let n be the number of non-ω places. Π'' is an infinite sequence of distinct n-tuples of natural numbers.

By Dickson's lemma, this sequence contains markings M_k, M_l such that $k < l$ and $M_k \leq M_l$.

This is a contradiction, because, since $M_k \neq M_l$, when executing $\mathsf{AddOmegas}(M_{l-1}, M_l, V, E)$ the algorithm adds at least one ω to M_{l-1}. $\square$

Coverability property

The lemma below states that for every ω-marking M' added by COVERABILITY-GRAPH one can reach markings with arbitrarily large values for all ω-components of M' simultaneously.

Definition 3.2.6 *A* place s is an ω-place of an ω-marking M if $M(s) = \omega$, and a normal place of M if $M(s) \in \mathbb{N}$.

Lemma 3.2.7 *For every ω-marking M' added by* COVERABILITY-GRAPH *to V and for every $k > 0$, there is a reachable marking M'_k satisfying*

- $M'_k(s) = M'(s)$ *for every normal place s of M', and*
- $M'_k(s) > k$ *for every ω-place of M'.*

Proof. Consider an arbitrary point during execution of COVERABILITY-GRAPH. We prove that if all ω-markings added to V until this point satisfy the lemma, then the next ω-marking that is added to V also does.

Assume the algorithm has just executed line 7 with $M \xrightarrow{t} M'$.

By induction hypothesis, for every $k > 0$ there is a reachable marking M_k satisfying

- $M_k(s) = M(s)$ for every place normal place s of M, and
- $M_k(s) > k$ for every ω-place s of M.

Case 1. Assume AddOmegas does not add any new ω to M' at line 8 and $M' \notin V$ holds at line 9.

Then COVERABILITY-GRAPH adds M' to V, and we can take M'_k as the marking given by $M_{k+1} \xrightarrow{t} M'_k$:

Indeed, we have $M'_k(s) > k$ because $M_{k+1}(s) > k+1$ for every ω-place s of M, and t removes at most one token from s.

Case 2. Assume that the **for**-loop in line 1 of AddOmegas(M, M', V, E)

- finds a unique ω-marking M'' satisfying $M'' \xrightarrow{*}_E M \xrightarrow{t} M'$ and $M'' \leq M'$, and
- adds a unique ω to M' for place s_0, i.e., $M''(s_0) < \overline{M}(s_0) = \omega$.

(General case is similar.)

Then COVERABILITY-GRAPH adds in line 11 the marking $\overline{M} = M' + (M' - M'') \cdot \omega$, which has exacly one more ω-place than M'', namely the place s_0.

We prove: for every $k > 0$ there is a reachable marking $\overline{M}_k$ satisfying

- $\overline{M}_k(s) = \overline{M}(s)$ for every place normal place s of $\overline{M}$, and
- $\overline{M}_k(s) > k$ for every ω-place s of $\overline{M}$.

Fix an arbitrary k. Our goal is to construct the marking $\overline{M}_k$.

Let σ be a firing sequence such that $M'' \xrightarrow{\sigma} M \xrightarrow{t} M'$. By induction hypothesis, for every $\ell > 0$ there is a reachable marking M_ℓ satisfying

- $M_\ell(s) = M(s)$ for every normal place s of M, and
- $M_\ell(s) > \ell$ for every ω-place s of M.

Fix $k > 0$. Choose ℓ large enough to guarantee that

(1) M_ℓ enables $t(\sigma t)^{k+1}$, and

(2) the marking M''' given by $M_\ell \xrightarrow{t(\sigma t)^{k+1}} M'''$ satisfies $M'''(s) > k$ for every ω-place s of M and M'.

We show that we can take $\overline{M}_k := M'''$.

First, by (2), $M'''_\ell(s) > k$ for every ω-place s of M'.

Since the execution of σt adds at least one token to s_0, after the execution of $t(\sigma t)^{k+1}$ the place s_0 has at least $k+1$ tokens, and so $M'''(s) > k$. $\square$

Theorem 3.2.8 *Let (N, M_0) be a Petri net and let M be a marking of N. There is a reachable marking $M' \geq M$ iff the coverability graph of (N, M_0) contains an ω-marking $M'' \geq M$.*

Proof. ($\Rightarrow$): Assume there is a reachable marking $M' \geq M$.

Then some firing sequence

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \cdots M_{n-1} \xrightarrow{t_n} M'$$

of (N, M_0) leads from M_0 to M'.

By the definition of the algorithm, the coverability graph contains a path

$$M_0 \xrightarrow{t_1} M_1' \xrightarrow{t_2} M_2' \cdots M_{n-1} \xrightarrow{t_n} M_n'$$

such that $M_i' \geq M_i$ for every $1 \leq i \leq n$.

Take $M'' = M_n'$.

($\Leftarrow$): Assume the coverability graph of (N, M_0) contains an ω-marking $M'' \geq M$.

By Lemma **??**, there is a reachable marking M_k'' satisfying $M_k''(s) = M''(s)$ for every normal place s of M'', and $M_k''(s) > k$ for every ω-place s of M''.

Take k larger than any of the components of M, and set $M' := M_k''$.

We have $M' \geq M$. □

Rackoff's algorithm

The coverability graph allows us to answer coverability of any marking.

However, Coverability asks whether a particular marking M can be covered.

We give a bound on how much of the reachability graph we need to explore to find a marking covering M from another marking M_0.

The bound depends on M, but not on M_0.

Theorem 3.2.9 *[Rackoff 1978]*

Let N be a net and let M be a marking of N.

Let k be the number of places of N, and let $n = 1 + \sum_{i=1}^{k} M(i)$.

For every marking M_0 of N:

> *if $M_0 \xrightarrow{*} M' \geq M$ for some M'*
> *then $M_0 \xrightarrow{\sigma} M'' \geq M$ for some M'' and some σ*
> *of length at most $(2n)^{(k+1)!} \in n^{2^{O(k \log k)}}$.*

Integer Petri nets

We consider Petri nets in which places may have a negative number of tokens. Transitions can occur independently of the number of tokens in their input places.

Definition 3.2.10 (Integer Petri nets) Let $N = (S, T, F)$ be a net. A generalized marking of N (g-marking) is a mapping $G\colon S \to \mathbb{Z}$.

An integer Petri net is a pair (N, G_0) where N is a net and G_0 is a g-marking.

The firing rule of integer nets is defined as follows:

> A g-marking G enables all transitions, and the occurrence of t at G leads to the marking G' defined as for standard Petri nets.

We denote by $G \overset{t}{\hookrightarrow} G'$ that firing t at G leads to the g-marking G'.

An integer firing sequence of an integer Petri net is a sequence $G_0 \overset{t_1}{\hookrightarrow} G_1 \overset{t_2}{\hookrightarrow} \cdots \overset{t_n}{\hookrightarrow} G_m$.

Fix a net N with places $\{s_1, \ldots, s_k\}$.

We identify g-markings with vectors of $\mathbb{Z}^k$.

Definition 3.2.11 *Let $G \in \mathbb{Z}^k$ be a g-marking of N and let $0 \leq i \leq k$.*

- G is i-natural if its first i-components are natural numbers, i.e., if $0 \leq G(j)$ for every $1 \leq j \leq i$.
- G is i-r-natural if $0 \leq G(j) < r$ for every $1 \leq j \leq i$.

Let $G \in \mathbb{Z}^k$. An integer firing sequence $G_0 \overset{t_1}{\hookrightarrow} \cdots \overset{t_m}{\hookrightarrow} G_m$

- is i-natural if all of $G_0, G_1, \ldots, G_m$ are i-natural g-markings.
- is i-r-natural if all of $G_0, G_1, \ldots, G_m$ are i-r-natural g-markings.
- i-covers G if $G_m(j) \geq G(j)$ for every $1 \leq j \leq i$.
  (But possibly $G_m(j) < G(j)$ for components $j > i$.)

Loosely speaking, we have:

- G is i-natural if its restriction to $\{s_1, \ldots, s_i\}$ is a "normal" marking,
- $G_0 \overset{t_1}{\hookrightarrow} \cdots \overset{t_m}{\hookrightarrow} G_m$ is i-natural if its restriction to $\{s_1, \ldots, s_i\}$ is a "normal" firing sequence.
- The k-natural sequences are the "normal" firing sequences (recall k is the number of places).
- A firing sequence covers M iff it k-covers M.
- M is coverable from M_0 iff some k-natural firing sequence of (N, M_0) k-covers M.

The key lemma

Lemma 3.2.12 *Let $G \in \mathbb{Z}^k$ be a g-marking of N, and let $n = 1+\sum_{i=1}^{k} |G(i)|$. For every $G_0 \in \mathbb{Z}^k$ and for every $0 \leq i \leq k$:*

if (N, G_0) has an i-natural sequence that i-covers G
then it has one of length at most $f(i)$,
where f is inductively defined as follows:

- $f(0) = 1$, *and*
- $f(i) = (nf(i-1))^i + f(i-1)$ *for every* $i \geq 1$. □

The function f grows rapidly in k:

$$\begin{array}{lcll} f(0) & = & 1 & \\ f(1) & = & (nf(0))^1 + f(0) = n+1 & \in O(n) \\ f(2) & = & (nf(1))^2 + f(1) = (n(n+1))^2 + n + 1 & \in O(n^4) \\ f(3) & = & (nf(2))^3 + f(2) & \in O(n^{15}) \\ f(4) & = & (nf(3))^4 + f(3) & \in O(n^{64}) \end{array}$$

In general, if $f(i) \in O(n^j)$, then $f(i+1) \in O(n^{(1+j)(1+i)})$.

Proof. By induction on i.

Base: $i = 0$. Follows vacuously from the fact that G_0 is a 0-natural sequence that 0-covers G.

(Every sequence is 0-natural, and the condition $G_0(j) \geq G(j)$ for every $1 \leq j \leq i$ holds vacuously when $i = 0$.)

Step: $i > 0$. Assume (N, G_0) has an i-natural sequence that i-covers G.

In the next two slides we consider two cases.

Case 1: Some sequence that i-covers G is i-$nf(i-1)$-natural.

Assume the sequence is

$$G_0 \overset{t_1}{\hookrightarrow} G_1 \cdots G_{m-1} \overset{t_m}{\hookrightarrow} G_m$$

and assume further that its length m is minimal.

We show $m \leq (nf(i-1))^i \leq f(i)$.

Claim: The projections of $G_0, G_1, \ldots, G_m$ onto their first i components are pairwise different.

Indeed: if there are $\alpha < \beta$ such that

$$(G_\alpha(1), \ldots, G_\alpha(i)) = (G_\beta(1), \ldots, G_\beta(i))$$

then the sequence

$$G_0 \overset{t_1}{\hookrightarrow} \cdots \xrightarrow{t_\alpha} G_\alpha \overset{t_{\beta+1}}{\hookrightarrow} G'_{\beta+1} \overset{t_{\beta+2}}{\hookrightarrow} \cdots \overset{t_m}{\hookrightarrow} G'_m$$

is also i-$nf(i-1)$-natural, i-covers G, and has shorter length.

Since $G_0 \overset{t_1}{\hookrightarrow} G_1 \cdots G_{m-1} \overset{t_m}{\hookrightarrow} G_m$ is i-$nf(i-1)$-natural, we have

$$(0, \ldots, 0) \leq (G_\ell(1), \ldots, G_\ell(i)) \leq (nf(i-1)-1, \ldots, nf(i-1)-1)$$

for every $1 \leq \ell \leq m$.

So there are $(nf(i-1))^i$ different possible value tuples for $(G_\ell(1), \ldots, G_\ell(i))$.

If $m > (nf(i-1))^i$, then there are $0 \leq \alpha < \beta \leq m$ such that

$$(G_\alpha(1), \ldots, G_\alpha(i)) = (G_\beta(1), \ldots, G_\beta(i))$$

contradicting the **Claim**.

So $m \leq (nf(i-1))^i \leq f(i)$.

Case 2: No sequence that i-covers G is i-$nf(i-1)$-natural.

Then there is a sequence $G_0 \overset{t_1}{\hookrightarrow} G_1 \overset{t_2}{\hookrightarrow} \cdots \overset{t_{m-1}}{\hookrightarrow} G_{m-1} \overset{t_m}{\hookrightarrow} G_m$

that i-covers G but is **not** i-$nf(i-1)$-natural.

Let G_α be the first not i-$nf(i-1)$-natural g-marking of the sequence.

We can assume $G_\alpha(i) \geq nf(i-1)$.

(If this would not be the case, reorder the places of the net.)

Then the prefix $G_0 \overset{t_1}{\hookrightarrow} \cdots \overset{t_{\alpha-1}}{\hookrightarrow} G_{\alpha-1}$ i-covers $G_{\alpha-1}$ and is i-$nf(i-1)$-natural.

As in Case 1, we can assume that the g-markings of the prefix are pairwise different, and so $\alpha - 1 \leq (nf(i-1))^i$.

The suffix $G_\alpha \overset{t_\alpha}{\hookrightarrow} G_{\alpha+1} \overset{t_{\alpha+1}}{\hookrightarrow} \cdots \overset{t_m}{\hookrightarrow} G_m$ is $(i-1)$-natural and $(i-1)$-covers G.

By IH there is a $(i-1)$-natural sequence $G_\alpha \overset{u_1}{\hookrightarrow} H_1 \overset{u_2}{\hookrightarrow} \cdots \overset{u_\ell}{\hookrightarrow} H_\ell$ of length at most $f(i-1)$ that $(i-1)$-covers G. So we have $\ell \leq f(i-1)$.

We have $G_\alpha(i) \geq nf(i-1)$, and a sequence of length $f(i-1)$ can remove at most $f(i-1)$ tokens from a place.

So the final g-marking H_ℓ satisfies

$$H_\ell(i) \geq G_\alpha(i) - f(n-1) \geq nf(n-1) - f(n-1) \geq n-1 \geq G(i)$$

So the sequence $G_0 \overset{t_1}{\hookrightarrow} \cdots \overset{t_{\alpha-1}}{\hookrightarrow} G_{\alpha-1} \overset{t_\alpha}{\hookrightarrow} G_\alpha \overset{u_1}{\hookrightarrow} H_1 \overset{u_2}{\hookrightarrow} \cdots \overset{u_\ell}{\hookrightarrow} H_\ell$

- is i-natural,
- has length at most $(nf(i-1))^i + f(i-1) = f(i)$, and
- i-covers G. □

Proof of Theorem ??

Theorem. [Rackoff 1978]

Let N be a net and let M be a marking of N.

Let k be the number of places of N, and let $n = 1 + \sum_{i=1}^{k} M(i)$.

For every marking M_0 of N:

> if $M_0 \xrightarrow{*} M' \geq M$ for some M'
> then $M_0 \xrightarrow{\sigma} M'' \geq M$ for some M'' and some σ
> of length at most $(2n)^{(k+1)!} \in n^{2^{O(k \log k)}}$.

Proof. Assume that (N, M_0) has a k-natural sequence that k-covers M.

By Lemma **??**, it has one of length at most $f(k)$.

So it suffices to prove $f(k) \leq (2n)^{(k+1)!}$.

We prove by induction on i that $f(i) \leq (2n)^{(i+1)!}$ for every $i \geq 0$.

Recall that $n \geq 1$ holds by the definition of n.

Further, it follows from the definition of f that $f(i) \geq 1$ for every $i \geq 0$.

Base: $i = 0$. Since $n \geq 1$, we have $f(0) = 1 \leq 2n = (2n)^{1!}$.

Step: $i > 0$. Assume $f(i-1) \leq (2n)^{i!}$. We prove $f(i) \leq (2n)^{(i+1)!}$. We have:

$$\begin{array}{rcll} f(i) & = & (nf(i-1))^i + f(i-1) & \text{(definition of } f\text{)} \\ & \leq & (n(2n)^{i!})^i + (2n)^{i!} & \text{(ind. hyp.)} \\ & \leq & (n(2n)^{i!})^i + (n(2n)^{i!})^i & (n \geq 1, i \geq 1) \\ & = & 2(n(2n)^{i!})^i & \\ & = & 2^{ii!+1} n^{ii!+i} & \\ & \leq & 2^{(i+1)!} n^{(i+1)!} & (n \geq 1, i \geq 1) \\ & = & (2n)^{(i+1)!} & \end{array}$$

Finally, we prove $(2n)^{(k+1)!} \in n^{2^{O(k \log k)}}$.

We first show $(k+1)! \in 2^{O(k \log k)}$.

$$\begin{array}{rcll} (k+1)! & \leq & (k+1)^{k+1} & \text{(definition of factorial)} \\ & \leq & (2k)^{k+1} & (k \geq 1) \\ & = & \left(2^{(\log k + 1)}\right)^{k+1} = 2^{(\log k+1)(k+1)} & \\ & \in & 2^{O(k \log k)} & \end{array}$$

Since $2n \leq n^2$ for every $n \geq 1$, we get

$$\begin{array}{rcl} (2n)^{(k+1)!} & \leq & n^{2(k+1)!} \\ & \in & n^{2 \cdot 2^{O(k \log k)}} = n^{2^{1+O(k \log k)}} \\ & = & n^{2^{O(k \log k)}} \end{array}$$ □

By Rackoff's theorem, in order to decide coverability of M we just construct the reachability graph up to depth $(2n)^{(k+1)!}$ using e.g. breadth-first search.

The backwards-reachability algorithm

The algorithm decides if a marking M is coverable in (N, M_0) by

- considering the set $\mathcal{M}$ of all markings that cover M,
- computing the set of predecessors of $\mathcal{M}$, i.e., the set of all markings M' (reachable or not) such that $M' \xrightarrow{*} M''$ for some $M'' \in \mathcal{M}$, and
- checking if M_0 belongs to this set.

Since even the set $\mathcal{M}$ is infinite, this computation requires to use a finite representation of infinite set of markings.

Definition 3.2.13 (Upward-closed sets of markings)
A set $\mathcal{M}$ of markings of a net N is upward closed if $M \in \mathcal{M}$ and $M' \geq M$ imply $M' \in \mathcal{M}$.

A marking M of an upward-closed set $\mathcal{M}$ is minimal if there is no $M' \in \mathcal{M}$ such that $M' \leq M$ and $M' \neq M$.

Upward-closed sets are equal iff their sets of minimal elements are equal.

Lemma 3.2.14 *Every upward-closed set has finitely many minimal elements.*

Proof. Assume some upward-closed set has infinitely many minimal markings $M_1, M_2, M_3, \ldots$.

By Dickson's Lemma there are $i \neq j$ such that $M_i \leq M_j$.

But then M_j is not minimal. Contradiction. □

Consequence: every upwards closed set can be finitely represented by its set of minimal elements.

Predecessors

Definition 3.2.15 *Let* $\mathcal{M}$ be a set of markings of a net $N = (S, T, F)$, and let $t \in T$ be a transition. We define

$$\begin{aligned} pre(\mathcal{M}, t) &= \{M' \mid M' \xrightarrow{t} M \text{ for some } M \in \mathcal{M}\} \\ pre(\mathcal{M}) &= \bigcup_{t \in T} pre(\mathcal{M}, t) \\ pre^0(\mathcal{M}) &= \mathcal{M} \\ pre^{i+1}(\mathcal{M}) &= pre\big(pre^i(\mathcal{M})\big) \text{ for every } i \geq 0 \\ pre^*(\mathcal{M}) &= \bigcup_{i=0}^{\infty} pre^i(\mathcal{M}) \end{aligned}$$

Lemma 3.2.16 *If* $\mathcal{M}$ *is upward closed, then* $pre(\mathcal{M})$ *and* $pre^*(\mathcal{M})$ *are also upward closed.*

Proof. (i) $pre(\mathcal{M})$ is upward closed.

Let $M' \in pre(\mathcal{M})$. We have to prove $M' + M'' \in pre(\mathcal{M})$ for every M''.

Since $M' \in pre(\mathcal{M})$ then $M' \xrightarrow{t} M$ for some is $M \in \mathcal{M}$ and $t \in T$.

By the firing rule we have $M' + M'' \xrightarrow{t} M + M''$ for every marking M''.

Since $\mathcal{M}$ is upward closed, we have $M + M'' \in \mathcal{M}$.

Since $M' + M'' \xrightarrow{t} M + M''$, we get $M' + M'' \in pre(\mathcal{M})$.

(ii) $pre^*(\mathcal{M})$ is upward closed.

By repeated application of (i): $pre^j(\mathcal{M})$ is upward closed for every $j \geq 0$.

So $pre^*(\mathcal{M})$ is a union of upward-closed sets.

By the definition of an upward-closed set: The union of upward-closed sets if also upward closed. □

Combining Lemma **??** and Lemma **??** we obtain:

Theorem 3.2.17 *Let $\mathcal{M}$ be an upward-closed set of markings of a net. Then there is $i \geq 0$ such that*

$$pre^*(\mathcal{M}) = \bigcup_{j=0}^{i} pre^j(\mathcal{M})$$

Proof. By Lemma **??**, $pre^*(\mathcal{M})$ is upward closed.

By Lemma **??**, the set m^* of minimal markings of $pre^*(\mathcal{M})$ is finite.

Since m^* is finite, there exists an index i such that $m^* \subseteq \bigcup_{j=0}^{i} pre^j(\mathcal{M})$.

Since $\bigcup_{j=0}^{i} pre^j(\mathcal{M})$ is upward closed, we have $pre^*(\mathcal{M}) \subseteq \bigcup_{j=0}^{i} pre^j(\mathcal{M})$.

By the definition of $pre^*(\mathcal{M})$, we get $pre^*(\mathcal{M}) = \bigcup_{j=0}^{i} pre^j(\mathcal{M})$. □

This Theorem leads to the backwards-reachability algorithm:

```
BACK(S, T, F, M_0, M)
 1  M := {M' | M' ≥ M};
 2  Old_M := ∅;
 3  while true
 4  do Old_M := M;
 5     M := M ∪ pre(M);
 6     if M_0 ∈ M
 7        then return  M can be covered
 8     if M = Old_M
 9        then return  M cannot be covered
```

Using Rackoff's theorem we can obtain an upper bound on the number of iterations of the while loop of BACK.

Theorem 3.2.18 *Let $\mathcal{M}$ be an upward-closed set of markings of a net with a set of places S.*

Let $\{M_1, \ldots, M_m\}$ be the set of minimal markings of $\mathcal{M}$.

For every $1 \leq i \leq m$, let $n_i = 1 + \sum_{s \in S} M_i(s)$, and let $n = \max\{n_1, \ldots, n_m\}$.

Then

$$pre^*(\mathcal{M}) = \bigcup_{j=0}^{(2n)^{(k+1)!}} pre^j(\mathcal{M})$$

Proof. Let $M \in pre^*(\mathcal{M})$.

By the definition of $pre^*(\mathcal{M})$, there is $M' \in \mathcal{M}$ such that $M \xrightarrow{*} M'$.

So $M' \geq M_i$ for some minimal marking M_i of $\mathcal{M}$.

By Theorem **??** and the definition of n, there exists a firing sequence

$$M \xrightarrow{\sigma} M'' \geq M_i$$

such that $|\sigma| \leq (2n_i)^{(k+1)!} \leq (2n)^{(k+1)!}$.

Since $M'' \geq M_i$ and $\mathcal{M}$ is upward closed, we have $M'' \in \mathcal{M}$.

So $M \in pre^j(\mathcal{M})$ for $j = |\sigma| \leq (2n)^{(k+1)!}$. □

Algorithm BACK is not directly implementable, because it manipulates infinite sets.

To solve this problem we give implementations of

- the tests $\mathcal{M} \stackrel{?}{=} \mathit{Old\_M}$ (line 8) and $M_0 \stackrel{?}{\in} \mathcal{M}$ (line 6), and
- the operation $\mathcal{M} \mapsto \mathcal{M} \cup \mathit{pre}(\mathcal{M})$ (line 5)

that use the finite representations of $\mathcal{M}$ and $\mathit{Old\_M}$, that is, their finite sets of minimal elements.

Implementing the tests

Given a set $\mathcal{M}$, let $\min(\mathcal{M})$ denote the set of minimal elements of $\mathcal{M}$. We have:

- $M_0 \in \mathcal{M}$ iff there exists $M' \in \min(\mathcal{M})$ such that $M_0 \geq M'$.
- $\mathcal{M} = \mathit{Old\_M}$ iff $\min(\mathcal{M}) = \min(\mathit{Old\_M})$.

Implementing $\mathcal{M} := \mathcal{M} \cup pre(\mathcal{M})$

We need several definitions.

Given a transition t, a marking M, and a set of markings $\mathcal{M}$:

- Let $R[t]$ denote the marking that puts one token in each output place of t, and no tokens elsewhere.
  ($R[t]$ is the minimal marking allowing to fire t "backwards").
- Let $\mathcal{M}[t]$ denote the set $\{M \in \mathcal{M} \mid M \geq R[t]\}$.
  ($\mathcal{M}[t]$ is the set of markings of $\mathcal{M}$ that enable t "backwards".)
- Let $M \wedge \mathcal{M}$ denote the set $\{M \wedge M' \mid M' \in \mathcal{M}\}$,
  where $(M \wedge M')(s) := \max\{M(s), M'(s)\}$ for every place s.

We make two observations:

- $$\begin{aligned} \min(\mathcal{M} \cup pre(\mathcal{M})) &= \min\big(\ \min(\mathcal{M}) \cup \min(pre(\mathcal{M})\ \big) \\ &= \min\Big(\min(\mathcal{M}) \cup \bigcup_{t \in T} \min\big(pre(\mathcal{M}, t)\big)\ \Big) \\ &= \min\Big(\min(\mathcal{M}) \cup \bigcup_{t \in T} pre\big(\min(\mathcal{M}[t]), t\big)\Big) \end{aligned}$$

  This reduces computing $\min(\mathcal{M} \cup pre(\mathcal{M}))$ to computing $\min(\mathcal{M}[t])$.

- If $\mathcal{M}$ is upward closed, then $\min(\mathcal{M}[t]) = R[t] \wedge \min(\mathcal{M})$.
  (Since $M \wedge R[t] \geq M$, if $\mathcal{M}$ is upward closed we have $M \wedge R[t] \in \mathcal{M}$ for every $M \in \min(\mathcal{M})$.)

  In words, the minimal markings of $\mathcal{M}$ that enable t "backwards" are obtained by taking the minimal markings of $\mathcal{M}$, and computing their join with $R[t]$.

Putting these two observations together, we get:

- If $\mathcal{M}$ is upward closed, then

$$\min(\mathcal{M} \cup pre(\mathcal{M})) = \min\Big(\ \min(\mathcal{M}) \cup \bigcup_{t \in T} pre\big(R[t] \wedge \min(\mathcal{M}), t\big)\ \Big)$$

which computes $\min(\mathcal{M} \cup pre(\mathcal{M}))$ as a function of $\min(\mathcal{M})$.

This leads to the algorithm BACK2:

BACK(S, T, F, M_0, M)
1 $\mathcal{M} := \{M' \mid M' \geq M\}$;
2 $Old\_\mathcal{M} := \emptyset$;
3 **while true**
4 **do** $Old\_\mathcal{M} := \mathcal{M}$;
5 $\mathcal{M} := \mathcal{M} \cup pre(\mathcal{M})$;
6 **if** $M_0 \in \mathcal{M}$
7 **then return** coverable
8 **if** $\mathcal{M} = Old\_\mathcal{M}$
9 **then return** not coverable

BACK2$((P, T, F, M_0, M))$
1 $m := \{M\}$;
2 $old\_m := \emptyset$;
3 **while true**
4 **do** $old\_m := m$;
5 $m := \min(m \cup \bigcup_{t \in T} pre(R[t] \wedge m, t))$;
6 **if** $\exists M' \in m : M_0 \geq M'$
7 **then return** coverable
8 **if** $m = old\_m$
9 **then return** not coverable

The type of the variables m and old_m is now "finite set of markings", and so their values can be stored in a computer.

Training Exercises in Automata Tutor:

- ☆ ⚙ Backwards Reachability 1: Apply the backwards reachability algorithm step-by-step.
- ⯪ ⚙ Backwards Reachability 2: Apply the backwards reachability algorithm step-by-step.
- ★ ⚙ Backwards Reachability 3: Apply the backwards reachability algorithm step-by-step.

3.2.3 The abstract backwards-reachability algorithm

The backwards reachability algorithm can be reformulated in more general terms, which allows to apply it to other models of concurrency more general than Petri nets.

Definition 3.2.19 *Let* A be a set and let $\preceq$ be a partial order on A. (I.e., a reflexive, antisymmetric, and transitive subset of $A \times A$.)

We say that $\preceq$ is a well-quasi-order (wqo) if every infinite sequence $a_1 a_2 a_3 \cdots \in A^\omega$ contains an infinite chain $a_{i_1} \preceq a_{i_2} \preceq a_{i_3} \cdots$ with $i_1 < i_2 < i_3 \ldots$.

Examples of well-quasi-orders:

- The pointwise order $\leq$ on $\mathbb{N}^k$.

  By Dickson's lemma.

- The subword order on Σ^* for any finite alphabet Σ.

  We say $w_1 \preceq w_2$ if w_1 can be obtained from w_2 by deleting letters.

  Higman's lemma states that every infinite sequence of words contains an infinite chain with respect to the subword order.

- The subtree order on the set of finite trees over a finite alphabet Σ.

  We say that $t_1 \preceq t_2$ if there is an injective mapping from the nodes of tree t_1 into the nodes of t_2 that preserves reachability: n' is reachable from n in t_1 iff the image of n' is reachable from the image of n in t_2.

  Kruskal's lemma states that every infinite sequence of trees contains an infinite chain with respect to the subtree order.

Definition 3.2.20 *Let A be a set and let $\preceq$ be a wqo on A.*

A set $X \subseteq A$ is upward closed if $x \in X$ and $x \preceq y$ implies $y \in X$ for every $x, y \in A$.

In particular, given $x \in A$, the set $\{y \in A \mid y \succeq x\}$ is upward-closed.

A relation $\rightarrow \subseteq A \times A$ is monotonic if for every $x \rightarrow y$ and every $x' \succeq x$ there is $y' \succeq y$ such that $x' \rightarrow y'$.

Given $X \subseteq A$, we define

$$\begin{aligned} pre(X) &= \{y \in A \mid y \rightarrow x \text{ and } x \in X\} \\ pre^0(X) &= X \\ pre^{i+1}(X) &= pre\big(pre^i(X)\big) \text{ for every } i \geq 0 \\ pre^*(X) &= \bigcup_{i=0}^{\infty} pre^i(X) \end{aligned}$$

Following the same steps as in the proof of Theorem **??** we can show:

Theorem 3.2.21 *Let A be a set and let $\preceq$ be a wqo on A.*

Let $X \subseteq A$ be an upward-closed set and let $\rightarrow \subseteq A \times A$ be monotonic.

Then there is $j \in \mathbb{N}$ such that

$$pre^*(X) = \bigcup_{i=0}^{j} pre^i(X)$$

This theorem can be used to obtain a backwards-reachability algorithm for generalizations of Petri nets, like

- reset Petri nets,
- lossy channel systems, or
- broadcast protocols,

whose transition relation is monotonic w.r.t. a suitable wpo.

Other net models, like Petri nets with inhibitor arcs, do not have a monotonic transition relations (adding tokens may disable a transition), and the theorem cannot be applied to them.

In fact we have:

Theorem 3.2.22 *Deadlock freedom, Liveness, Boundedness, b-boundedness, Reachability, and Coverability are all undecidable for Petri nets with inhibitor arcs.*

3.2.4 Decision procedures for other problems

Reachability

The decidability of Reachability was open for about 10 years until it was proved by Mayr in 1980.

Kosaraju and Lambert simplified the proof in 1982 and 1992, respectively.

All these algorithms and their proofs exceed the framework of this course.

In 2012 Leroux provided a new algorithm.

Its proof is as complicated as the proofs of the previous ones, but the algorithm is very simple.

Definition 3.2.23 (Semilinear set) A set $X \subseteq \mathbb{N}^k$ is linear if there is a root $r \in \mathbb{N}^k$ (the) and a finite set $P \subseteq \mathbb{N}^k$ of periods) such that

$$X = \{r + \sum_{p \in P} \lambda_p \cdot p \mid \forall p \in P \colon \lambda_p \in \mathbb{N}\}$$

A semilinear set is a finite union of linear sets.

A semilinear set can be finitely represented as a set of pairs

$$\{(r_1, P_1), \ldots, (r_n, P_n)\}$$

giving the roots and periods of its linear sets.

Theorem 3.2.24 *[Leroux 2012] Let M_0 and M_1 be markings of a net N. If M_1 is not reachable from M_0, then there exists a semilinear set $\mathcal{M}$ of markings of N such that*

(a) $M_0 \in \mathcal{M}$;

(b) if $M \in \mathcal{M}$ *and* $M \xrightarrow{t} M'$ *for some transition t, then* $M' \in \mathcal{M}$; *and*

(c) $M_1 \notin \mathcal{M}$.

Given a semilinear set $\mathcal{M}$ represented by $\{(r_1, P_1), \ldots, (r_n, P_n)\}$, we can check whether $\mathcal{M}$ satisfies (a)-(c).

Indeed, checking (a) and (c) amounts to determining if the systems of linear equations

$$M_0 = r_i + \sum_{p \in P_i} \lambda_p \cdot p \quad \text{and } M_1 = r_i + \sum_{p \in P_i} \lambda_p \cdot p$$

with unknowns $\lambda_1, \ldots, \lambda_n$ (two systems for each $1 \leq i \leq n$) have an integer nonnegative solution.

Checking (b) reduces to checking validity of a formula of a theory called Presburger arithmetic, for which decision procedures exist.

Theorem **??** leads to an algorithm for Reachability of a marking M, consisting of two semi-decision procedures:

- The first procedure explores the reachability graph breadth-first, and stops if it reaches M.
- The second procedure enumerates all semilinear sets, and stops if one of them satisfies (a)-(c).

The procedures run in parallel. Since for every M one or the other terminates, they yield together a decision procedure for Reachability.

Deadlock-freedom

We reduce Deadlock-freedom to Reachability.

We proceed in two steps. First, we reduce Deadlock-freedom to an auxiliary problem P, defined next, and then we reduce P to Reachability.

> P: Given a Petri net (N, M_0) and a subset R of places of N, is there a reachable marking M such that $M(s) = 0$ for every $s \in R$?

Theorem 3.2.25 *Deadlock-freedom can be reduced to P.*

Proof. Let (N, M_0) be a Petri net such that $N = (S, T, F)$. Define

$$\mathcal{S} = \{R \subseteq S \mid \forall t \in T : {}^\bullet t \cap R \neq \emptyset\}$$

(An element of $\mathcal{S}$ contains at least one of each transition t.) We have

(1) $\mathcal{S}$ is a finite set.

(2) A marking M of N is dead iff the set of places unmarked at M is an element of $\mathcal{S}$.

Suppose now that there is an algorithm that decides P.

We can then decide Deadlock-freedom as follows.

For every $R \in \mathcal{S}$ we use the algorithm for P to decide if some reachable marking M satisfies $M(s) = 0$ for every $s \in R$.

By (2), (N, M_0) is deadlock-free if the answer is negative in all cases.

By (1), we only have to solve a finite number of instances of P. □

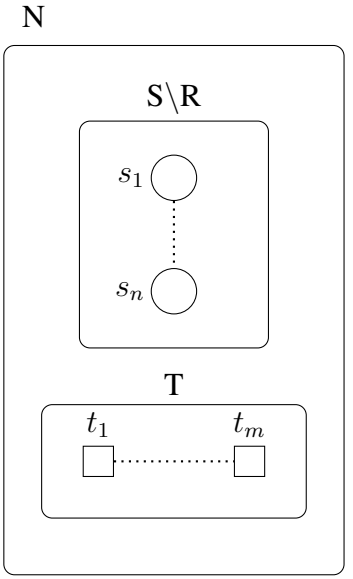
Theorem 3.2.26 *P can be reduced to Reachability.*

Proof. Let (N, M_0) be a Petri net where $N = (S, T, F)$, and let R be a set of places of N.

We construct a new Petri net (N', M_0') by adding new places, transitions, and arcs to (N, M_0).

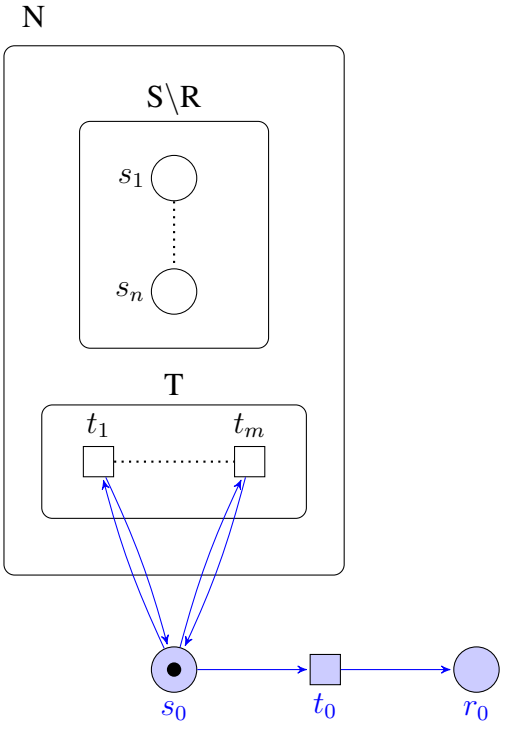
We proceed in two steps.

We start with (N, M_0) (drawn with the transitions at the bottom and the places of $S \setminus R$ highlighted).



In a first step, we add

- two new places s_0 and r_0, putting one token on s_0.
- a transition t_0 and arcs (s_0, t_0) and (t_0, r_0).
- two arcs (s_0, t) and (t, s_0) for every transition $t \in T$.

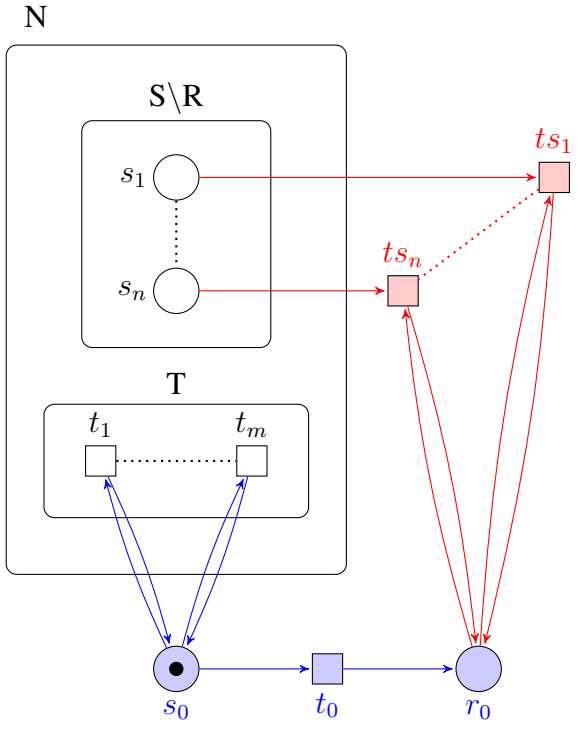


Observe: transition t_0 can occur at any time, and when this happens all transitions of T become "dead".

Intuitively, firing t_0 "freezes" the current marking of N.

In a second step we add:

- a transition t_s and arcs $(s, t_s), (r_0, t_s), (t_s, r_0)$ for every $s \in S \setminus R$.



Intuitively, these transitions are "garbage collectors". The "garbage" are the tokens in the places of $S \setminus R$.

If r_0 becomes marked, then the garbage collectors can "empty" $S \setminus R$.

This concludes the definition of (N', M'_0).

Let M_{r_0} be the marking of N' that puts one token on r_0 and no tokens elsewhere. We have

(1) If some reachable marking M of (N, M_0) puts no tokens in R, then M_{r_0} is a reachable marking of (N', M_0').

To reach M_{r_0}, we first fire a sequence of transitions of T leading from M_0 to M, then we fire t_0, and finally we fire the transitions $\{t_s \mid s \in S\}$ until all places of S are empty.

(2) If M_{r_0} is a reachable marking of (N', M_0'), then some marking M reachable from (N, M_0) puts no tokens in R.

M_{r_0} can only be reached by firing t_0 at a marking that puts no tokens in R (because after firing t_0 the places of R cannot be emptied anymore).

So we can choose M as the marking reached immediately before firing t_0.

By (1) and (2), we can decide if some reachable marking M of (N, M_0) puts no tokens in R as follows: construct (N', M_0') and decide if M_{r_0} is reachable.

Therefore, if there is an algorithm for Reachability, then there is also one for P. □

Liveness

Liveness can also be reduced to Reachability. The proof is more complex.

We sketch how to reduce the problem

> Given: A Petri net (N, M_0), a transition t of N.
> Decide: Is t live in (N, M_0)?

to Reachability.

Let E_t be the set of markings of N that enable t.

E_t is upward closed, and therefore so is $pre^*(E_t)$ (Lemma **??**).

We have: $pre^*(E_t)$ is the set of markings of N that enable some firing sequence ending with t.

Let D_t be the complement of $pre^*(E_t)$, that is, the set of markings from which t cannot be enabled anymore.

We have: (N, M_0) is live iff no marking of D_t is reachable from M_0.

So liveness of t reduces to the reachability of D_t.

However, D_t may be infinite, and so this is not a reduction to Reachability (yet).

We solve this problem.

Claim 1: D_t is a semilinear set and we can compute a semilinear representation of it.

We can compute the finite set $\min(pre^*(E_t))$ using the backwards-rechabiilty algorithm.

From $\min(pre^*(E_t))$ we can compute a semilinear representation of $pre^*(E_t)$ (exercise).

Now we use a powerful result from the literature (without proof):

> The complement of a semilinear set is also semilinear.
>
> Moreover, there is an algorithm that, given a representation of a semilinear set $X \subseteq \mathbb{N}^k$, computes a representation of the complement $\mathbb{N}^k \setminus X$.

This proves the claim.

Claim 2: The problem:

> Given: a Petri net (N, M_0) and a (representation of) a semilinear set X.
>
> Decide: Is some marking of X reachable from M_0 ?

can be reduced to Reachability.

The reduction is as follows.

We construct a Petri net that first simulates (N, M_0), and then transfers control to another Petri net that nondeterministically generates a marking of X on "copies" of the places of N.

This second net then transfers control to a third, whose transitions remove one token from a place of N and a token from its "copy".

If X is reachable, then

- The first net can produce a marking of X.
- The second net can produce the same marking.
- The third net can then remove all tokens from the first and second nets, reaching the empty marking.

Conversely, if the net consisting of the three nets together can reach the empty marking, then (N, M_0) can reach some marking of X.

3.2.5 Complexity

Unfortunately, all the problems we have seen so far have very high complexity.

We prove that all of them are EXPSPACE-hard.

Rackoff's algorithm shows that Coverability is in EXPSPACE-complete, that is, that exponentially growing memory suffices. The same can be proved for Boundedness.

For a long time it was conjecture that Deadlock-freedom, Liveness, and Reachability were EXPSPACE-complete as well. However, the conjecture was disproved in 2019 by Czerwiński *et al.*. All three problems have non-elementary complexity.

To explain what this means, define inductively the functions $exp_k(x)$ as follows:

- $exp_0(x) = x$;
- $exp_{k+1}(x) = 2^{exp_k(x)}$.

The complexity class k-EXPSPACE contains the problems that can be solved by a Turing machine using at most $exp_k(n)$ space for inputs of length n.

The class of *elementary problems* is defined as

$$\bigcup_{k=0}^{\infty} k\text{-EXPSPACE}$$

In other words, a problem is elementary if there is a number k and a Turing machine that solves every instance of the problem of size n using at most $exp_k(n)$ space.

Some problems related to logical theories have non-elementary complexity. Logical theories are sets of formulas, typically defined by closing a set of *atomic formulas* under boolean operations and quantification. Without getting into details, in some logical theories the complexity of deciding if a formula is true is given by a tower of exponentials whose height is equal to the number of quantifiers in the formula times some constant. Since formulas can have an arbitrary number of quantifiers, these problems are non-elementary.

Consider the function that assigns to each n the number $exp_n(n)$. This function grows faster than any tower-of-exponentials function of fixed height. However, the function belongs to the class of *primitive recursive* functions. Loosely speaking, these are the functions computable by programs using only **for**-loops. In particular, such programs are guaranteed to terminate, because no **for**-loop can run forever. There are functions that grow even faster than every primitive-recursive function. The best known example is the *Ackermann function*, inductively defined by

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1,A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

All known algorithms for Deadlock-freedom, Liveness, and Reachability have non-primitive recursive runtime, that is, their runtime grows faster than any elementary function.

Simulating exponentially bounded automata with 1-bounded Petri nets
An exponentially bounded automaton is a Turing machine that can only work on a tape with 2^n cells, where n is the size of the machine.

> A deterministic, exponentially bounded automaton of size n can be simulated by a Petri net of size $O(n^2)$.
> Moreover, there is a polynomial time procedure which constructs this net.

In order to answer a question about the computation of an exponentially space bounded automaton A, we can construct the net that simulates A, which has size $O(n^2)$, and solve the corresponding question. If the original question requires 2^n space, as is the case for many properties, then the corresponding question about nets requires at least $2^{O(\sqrt{n})}$-space.

Counter programs

Bounded automata and general Place/Transition Petri nets do not "fit" well.

It is not appropriate to model a cell of a bounded automaton as a place, as we did in the 1-safe case, because the cell contains one out of a finite number of possible symbols, while the place can contain infinitely many tokens, and so the same information as a nonnegative integer variable.

So we use an intermediate model, namely counter programs.

It is well-known that so-called bounded counter programs can simulate bounded automata (see below), and we show that Petri nets can simulate bounded counter programs.

A counter program is a sequence of labeled commands separated by semicolons.

Basic commands have the following form, where l, l_1, l_2 are labels or addresses taken from some arbitrary set, for instance the natural numbers, and x is a variable over the natural numbers, also called a counter:

l: $x := x + 1$	
l: $x := x - 1$	
l: **goto** l_1	unconditional jump
l: **if** $x = 0$ **then goto** l_1 **else goto** l_2	conditional jump
l: **halt**	

A program is syntactically correct if the labels of commands are pairwise different, and if the destinations of jumps correspond to existing labels.

For convenience we can also require the last command to be a halt command.

A program can only be executed once its variables have received initial values. We assume that the initial values are always 0.

The semantics of programs is the one suggested by the syntax. The only point to be remarked is that the command $l : x := x - 1$ fails if $x = 0$, and causes abortion of the program.

Abortion must be distinguished from proper termination, which corresponds to the execution of a halt command.

A counter program C is k-bounded if after any step in its unique execution the contents of all counters are smaller than or equal to k.

We use a well-known construction of computability theory:

> There is a polynomial time procedure which accepts a deterministic bounded automaton A of size n and returns a counter program C with $O(n)$ commands that simulates the computation of A on empty tape. I
> In particular, A halts if and only if C halts. Moreover, if A is exponentially bounded, then C is 2^{2^n}-bounded.

Now, it suffices to show that a 2^{2^n}-bounded counter program of size $O(n)$ can be simulated by a Petri net of size $O(n^2)$. This is the goal of the rest of this section.

Simulating bounded counter programs by Petri nets

We describe the Petri net that simulates a 2^{2^n}-bounded counter program.

Since a direct description of the sets of places and transitions of the simulating net would be very confusing, we introduce a net programming notation with a very simple net semantics. It is very easy to obtain the net corresponding to a program, and execution of a command corresponds exactly to the firing of a transition. So we look at the programming notation as a compact description language for Petri nets.

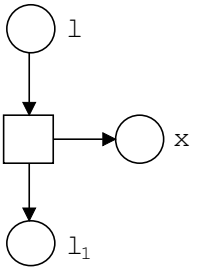
A net program is similar to a counter program, but does not have the possibility to branch on zero; it can only branch nondeterministically.

However, it has the possibility of transferring control to a subroutine.

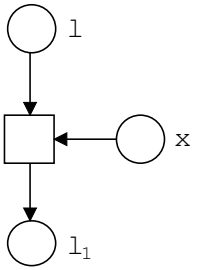
The basic commands are as follows:

l: $x := x + 1$	
l: $x := x - 1$	
l: **goto** l_1	unconditional jump
l: **goto** l_1 **or goto** l_2	nondeterministic jump
l: **gosub** l_1	subroutine call
l: **return**	end of subroutine
l: **halt**	

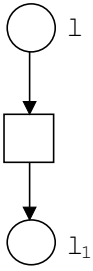
The Petri net semantics of net programs is explained in this figure:



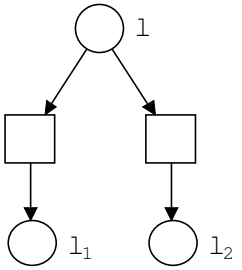

```
l : x:=x+1;
l1: ...
```



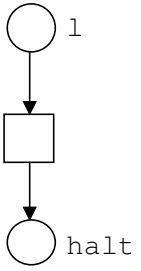

```
l : x:=x-1;
l1: ...
```




```
l: goto l1
```



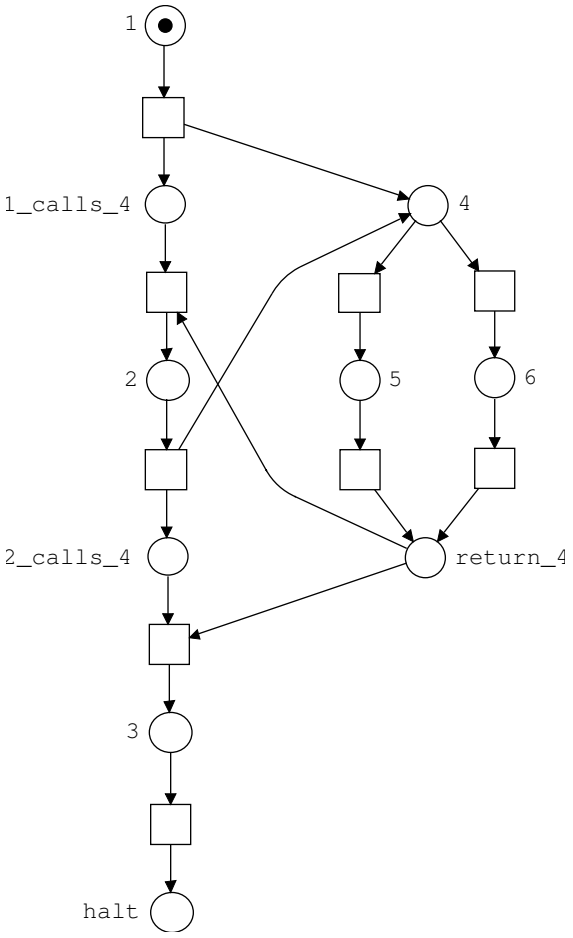

```
l: goto l1
    or
   goto l2
```




```
l: halt
```

Example:

1: **gosub** 4;
2: **gosub** 4;
3: **halt**;
4: **goto** 5 **or goto** 6;
5: **return**;
6: **return**



The places `1_calls_4` and `2_calls_4` are used to remember the address from which the subroutine was called.

The Petri net corresponding to a net program with k commands has $O(k)$ places and $O(k)$ transitions, and its initial marking has size $O(k)$. So it is of size $O(k^2)$.

Let C be a 2^{2^n}-bounded counter program with $O(n)$ commands.

We show that C can be simulated by a net program $N(C)$ with $O(n)$ commands, which corresponds to a Petri net of size $O(n^2)$.

$N(C)$ will be a non-deterministic net program, even though C is deterministic. So what does "simulate" mean?

$N(C)$ simulates C in the following way:

> C halts (executes the command **halt**) if and only if some computation of N halts (other computations may fail).

Each variable x of $N(C)$ (be it a variable from C or an auxiliary variable) has an auxiliary complement variable $\overline{x}$.

N takes care of setting $\overline{x} = 2^{2^n}$ at the beginning of the program. We call the code that takes care of this $N_{init}(C)$.

The rest of $N(C)$, called $N_{sim}(C)$, simulates C and takes care of keeping the invariant $\overline{x} = 2^{2^n} - x$.

The net program $N_{sim}(C)$

$N_{sim}(C)$ is obtained through replacement of each command of C by an adequate net program.

Commands of the form $x := x + 1$ ($x := x - 1$) are replaced by the net program $x := x + 1; \overline{x} := \overline{x} - 1$ ($x := x - 1; \overline{x} := \overline{x} + 1$).

Unconditional jumps are replaced by themselves.

Let us now design a program

$$\textsc{Test}_n(x,\texttt{ZERO},\texttt{NONZERO})$$

to replace a conditional jump of the form

l: **if** $x = 0$ **then goto** ZERO
else goto NONZERO

The specification of $\textsc{Test}_n$ is as follows:

> If $x = 0$ ($1 \leq x \leq 2^{2^n}$), then some execution of the program leads to ZERO (NONZERO), and no computation leads to NONZERO (ZERO).
>
> Moreover, the program has no side-effects: after any execution leading to ZERO or NONZERO no variable has changed its value.

Actually, it is easier to design a program $\textsc{Test}'_n(x, \texttt{ZERO}, \texttt{NONZERO})$ with the same specification but a side-effect: after an execution leading to ZERO, the values of x and $\overline{x}$ are swapped.

Once $\textsc{Test}'_n$ has been designed, we can take:

$\textsc{Test}_n(x, \texttt{ZERO}, \texttt{NONZERO})$:

$$\begin{aligned} & \textsc{Test}'_n(x, \texttt{continue}, \texttt{NONZERO}); \\ \texttt{continue}:\ & \textsc{Test}'_n(\overline{x}, \texttt{ZERO}, \texttt{NONZERO}) \end{aligned}$$

because the values of x and $\overline{x}$ are swapped 0 times if $x > 0$ or twice if $x = 0$, and so $\textsc{Test}_n$ has no side effects.

The net program $\textsc{Test}'_n$

The key to the design of $\textsc{Test}'_n$ lies in the following observation: Since x never exceeds 2^{2^n}, testing $x = 0$ can be replaced by nondeterministically choosing

- to decrease x by 1, and if we succeed then we know that $x > 0$, or
- to decrease $\overline{x}$ by 2^{2^n}, and if we succeed then we know that $\overline{x} = 2^{2^n}$, and so $x = 0$.

If we choose wrongly, that is, if for instance $x = 0$ holds and we try to decrease x by 1, then the program fails; this is not a problem, because we only have to guarantee that the program may (not must!) terminate, and that if it terminates then it provides the right answer.

Decreasing x by 1 is easy (exercise).

Decreasing $\overline{x}$ by 2^{2^n} is the difficult part. We leave it for a routine $\textsc{Dec}_n(s_n)$ (designed later), which must satisfy the following specification:

> If the initial value of s is smaller than 2^{2^n}, then every execution of $\textsc{Dec}_n$ fails.
>
> If the value of s_n is greater than or equal to 2^{2^n}, then
>
> - all executions terminating with a **return** command have the same effect as $s_n := s_n - 2^{2^n}; \overline{s}_n := \overline{s}_n + 2^{2^n}$; in particular, there are no side-effects;
> - all other executions fail.

We give the code of $\textsc{Test}'_n$.

$\textsc{Test}'_n(x, \texttt{ZERO}, \texttt{NONZERO})$

// initially $s_n = 0$ and $\overline{s}_n = 2^{2^n}$ //

goto `nonzero` **or goto** `loop`;

`nonzero`: $x := x - 1; x := x + 1;$

goto `NONZERO`;

`loop`: $\overline{x} := \overline{x} - 1; x := x + 1;$

$s_n := s_n + 1; \overline{s}_n := \overline{s}_n - 1;$

goto `exit` **or goto** `loop`

`exit`: **gosub** $\texttt{dec}_n$; // the routine called at $\texttt{dec}_n$ is $\textsc{Dec}_n(s_n)$ //

goto `ZERO`

Recall the specification of $\textsc{Test}'_n$:

> If $x = 0$ ($1 \leq x \leq 2^{2^n}$), then some execution of the program leads to `ZERO` (`NONZERO`), and no computation leads to `NONZERO` (`ZERO`).
>
> After any execution leading to `ZERO` the values of x and $\overline{x}$ are swapped, and no other variable changes its value; and after any execution leading to `NONZERO` no variable changes its value.

If $x > 0$, then we can choose the `nonzero` branch and reach `NONZERO`. No execution ever reaches `ZERO`, because s_n cannot reach the value 2^{2^n}, and so $\textsc{Dec}_n$ fails.

If $x = 0$, then $\overline{x} = 2^{2^n}$. After looping 2^{2^n} times on `loop` the values of x, $\overline{x}$ and s_n, $\overline{s}_n$ have been swapped. The values of s_n and $\overline{s}_n$ are swapped again by $\textsc{Dec}_n$, and then the program moves to `ZERO`. No execution reaches the `NONZERO` branch, because the program fails at $x := x - 1$.

The next step is to design $\textsc{Dec}_n$.

The net program DEC_n

We proceed by induction on n. DEC_0 has to decrease s by $2^{2^0} = 2$. We take

$\text{DEC}_0(s)$

$s := s - 1; \overline{s} := \overline{s} + 1;$
$s := s - 1; \overline{s} := \overline{s} + 1;$
return

Now we design DEC_{i+1} under the assumption that DEC_i is already known.

Key idea: decreasing by $2^{2^{i+1}}$ amounts to decreasing 2^{2^i} times by 2^{2^i}, because

$$2^{2^{i+1}} = (2^{2^i})^2 = 2^{2^i} \cdot 2^{2^i}$$

So decreasing by $2^{2^{i+1}}$ can be implemented by two nested loops, each of which is executed 2^{2^i} times. The body of the inner loop decreases s by 1.

The loop counters have initial values 2^{2^i}. The loops terminate when the counters reach 0, which is tested with TEST'_i.

$\text{DEC}_{i+1}(s)$

// initially $y_i = 2^{2^i} = z_i, \overline{y}_i = 0 = \overline{z}_i$ //
// the initialisation is carried out by N_{init} //
`outer_loop`: $y_i := y_i - 1; \overline{y}_i := \overline{y}_i + 1;$
`inner_loop`: $z_i := z_i - 1; \overline{z}_i := \overline{z}_i + 1;$
$s := s - 1; \overline{s} := \overline{s} + 1;$
$\text{TEST}'_i(z_i,$ `inner_exit` , `inner_loop`$);$
`inner_exit`: $\text{TEST}'_i(y_i,$ `outer_exit`, `outer_loop`$);$
`outer_exit`: **return**

Recall the specification of DEC$_i$:

> If the initial value of s is smaller than 2^{2^i}, then every execution of DEC$_i$ fails.
>
> If the value of s is greater than or equal to 2^{2^n}, then
>
> - all executions terminating with a **return** command have the same effect as $s := s - 2^{2^i}; \overline{s} := \overline{s} + 2^{2^i}$; in particular, there are no side-effects;
> - all other executions fail.

It could seem that DEC$_{i+1}$ swaps the values of y_i, $\overline{y}_i$ and z_i, $\overline{z}_i$, which would be a side-effect contrary to the specification.

This is not the case. These swaps are compensated by the side-effects of the `ZERO` branches of the TEST$'_i$ programs! Notice that these branches are now the `inner_exit` and `outer_exit` branches. When the program leaves the inner loop, TEST$'_i$ swaps the values of z_i and $\overline{z}_i$. When the program leaves the outer loop, TEST$'_i$ swaps the values of y_i and $\overline{y}_i$.

The net program $N_{init}(C)$

Let us first make a list of the initializations that have to be carried out.
$N_{sim}(C)$ contains

- the variables $x_1, \ldots, x_l$ of C with initial value 0; their complementary variables $\overline{x}_1, \ldots, \overline{x}_l$ with initial value 2^{2^n};
- a variable s with initial value 0; its complementary variable $\overline{s}$ with initial value 2^{2^n};
- two variables y_i, z_i for each i, $0 \leq i \leq n-1$, with initial value 2^{2^i}; their complementary variables $\overline{y}_i, \overline{z}_i$ for each i, $0 \leq i \leq n-1$, with initial value 0.

The specification of $N_{init}(C)$ is:

> $N_{init}(C)$ uses only the variables in the list above; every successful execution leads to a state in which the variables have the correct initial values.

$N_{init}(C)$ calls programs $\textsc{Inc}_i(v_1, \ldots, v_m)$ with the following specification:

> All successful executions have the same effect as
>
> $$\begin{aligned} v_1 &:= v_1 + 2^{2^i}; \\ &\ldots; \\ v_m &:= v_m + 2^{2^i} \end{aligned}$$
>
> In particular, there are no side-effects.

The net program $\textsc{Inc}_i$

$\textsc{Inc}_i$ is defined by induction on i, and is very similar to the family of $\textsc{Dec}_i$ programs.

We start with $\textsc{Inc}_0$:

$\textsc{Inc}_0(v_1, \ldots, v_m)$:
$v_1 := v_1 + 1; v_1 := v_1 + 1;$
$\ldots$
$v_m := v_m + 1; v_m := v_m + 1$

and now give the inductive definition of $\textsc{Inc}_{i+1}$:

$\textsc{Inc}_{i+1}(v_1, \ldots, v_m)$
// initially $y_i = 2^{2^i} = z_i$, $\overline{y}_i = 0 = \overline{z}_i$ //
`outer_loop`: $y_i := y_i - 1; \overline{y}_i := \overline{y}_i + 1;$
`inner_loop`: $z_i := z_i - 1; \overline{z}_i := \overline{z}_i + 1;$
$v_1 := v_1 + 1;$
$\ldots$
$v_m := v_m + 1;$
$\textsc{Test}'_i(z_i,$ `inner_exit` , `inner_loop`$)$;
`inner_exit`: $\textsc{Test}'_i(y_i,$ `outer_exit`, `outer_loop`$)$;
`outer_exit`: ...

It is easy to see that these programs satisfy their specifications.

We now give the code for $N_{init}(C)$.

Apparently, we face a problem: in order to initialize the variables $v_1, \ldots, v_m$ to $2^{2^{i+1}}$ the variables y_i and z_i must have already been initialized to 2^{2^i}.

But it suffices to carry out the initializations in the right order:

$\underline{N_{init}(C)}$

$\textsc{Inc}_0(y_0, z_0)$; $\textsc{Inc}_1(y_1, z_1)$; $\ldots$; $\textsc{Inc}_{n-1}(y_{n-1}, z_{n-1})$; $\textsc{Inc}_n(\overline{s}, \overline{x}_1, \ldots, \overline{x}_l)$

Size of $N(C)$

We show that $N(C)$ has size $O(n^2)$, where n is the size of C.

$N_{sim}(C)$ It contains two assignments for each assignment of C, an unconditional jump for each unconditional jump in C, and a different instance of $\textsc{Test}_n$ for each conditional jump.

Moreover, it contains (one single instance of) the routines $\textsc{Dec}_n$, $\textsc{Dec}_{n-1}$, $\ldots$, $\textsc{Dec}_0$ (notice that $\textsc{Test}_n$ calls $\textsc{Dec}_n$, which calls $\textsc{Dec}_{n-1}$, etc.).

Both $\textsc{Test}_n$ and the routines have constant length.

So the number of commands of $N_{sim}(C)$ is $O(n)$.

$N_{init}(C)$ contains (one single instance of) the programs $\textsc{Inc}_i$ for $1 \leq i \leq n$.

The programs $\textsc{Inc}_1$, $\ldots$, $\textsc{Inc}_{n-1}$ have constant size, since they initialize a constant number of variables.

The number of commands of $\textsc{Inc}_n$ is $O(n)$, since it initializes $O(n)$ variables.

So $N(C)$ contains $O(n)$ commands. It follows that its corresponding Petri net has size $O(n^2)$.

Chapter 4

Semi-decision procedures

4.1 Linear systems of equations and linear programming

In the next two sections we construct linear systems of equations with integer or rational coefficients that provide partial information about our analysis problems. (More precisely, they will be systems of equations and inequations, but, abusing language, we let "equations" stand for both.)

We will prove propositions like "if the system of equations $A \cdot X \leq b$ (we will see how this system looks like) has a rational positive solution, then the Petri net (N, M_0) is bounded" (sufficient condition), or "if M is reachable in (N, M_0), then the system of equations $B \cdot X = b$ has a solution over the natural numbers" (necessary condition).

Such propositions lead to semi-decision procedures to prove or disprove a property. The complexity of these procedures depends on the complexity of solving the different systems of equations.

We define the size of a linear system of equations $A \cdot X = b$ or $A \cdot X \leq b$ where $A = (a_{ij})_{i=1,...n,j=1,...,m}$ and $b = (b_j)_{j=1,...,m}$ as

$$\sum\{log_2|a_{ij}| \mid 1 \leq i \leq n, 1 \leq j \leq m\} + \sum\{log_2|b_j| \mid 1 \leq j \leq m\}$$

The problem of deciding whether $A \cdot X = b$ has

- a rational solution can be solved in polynomial time.
  (Naive Gauss elimination is not polynomial, because coefficients can become very large. However, there exist improved versions with polynomial complexity.).
- an integer solution can be solved in polynomial time.
- a nonnegative integer solution is NP-complete.

The problem of deciding whether $A \cdot X \leq b$ has

- a rational solution can be solved in polynomial time.
  (In practice we use the simplex algorithm, which has exponential worst-case complexity, but is very efficient for most instances.)
- an integer solution is NP-complete.
- a nonnegative integer solution is NP-complete.

Given a linear objective function $f(X) = c_1x_1 + \ldots c_m$ we can decide with the same runtime whether there is a solution X_{op} that maximizes $f(X)$ and, if so, the value $f(X_{op})$.

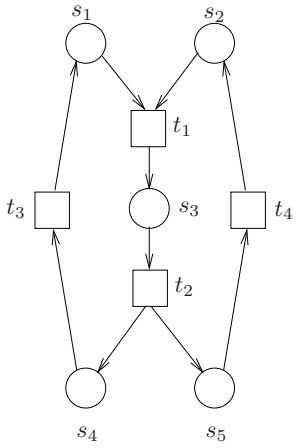
4.2 The Marking Equation

Definition 4.2.1 (Incidence matrix)
Let $N = (S, T, F)$ be a net. The incidence matrix $\mathbf{N} : (S \times T) \to \{-1, 0, 1\}$ is given by

$$\mathbf{N}(s,t) = \begin{cases} 0 & \text{if } (s,t) \notin F \text{ and } (t,s) \notin F \text{ or} \\ & \quad (s,t) \in F \text{ and } (t,s) \in F \\ -1 & \text{if } (s,t) \in F \text{ and } (t,s) \notin F \\ 1 & \text{if } (s,t) \notin F \text{ and } (t,s) \in F \end{cases}$$

The column $\mathbf{N}(-,t)$ is denoted by $\mathbf{t}$, and the row $\mathbf{N}(s,-)$ by $\mathbf{s}$.

Example 4.2.2



	t_1	t_2	t_3	t_4
s_1	-1	0	1	0
s_2	-1	0	0	1
s_3	1	-1	0	0
s_4	0	1	-1	0
s_5	0	1	0	-1

Definition 4.2.3 (Parikh-vector of a sequence of transitions)
Let $N = (S, T, F)$ be a net and let σ be a finite sequence of transitions. The Parikh vector $\boldsymbol{\sigma} : T \to \mathbb{N}$ of σ is defined by

$$\boldsymbol{\sigma}(t) = \textit{ number of occurrences of } t \textit{ in } \sigma$$

Lemma 4.2.4 (Marking Equation Lemma)
Let N be a net and let $M \xrightarrow{\sigma} M'$ be a firing sequence of N. Then:

$$M' = M + \mathbf{N} \cdot \boldsymbol{\sigma}$$

Proof. By induction on the length of σ.
Basis: $\sigma = \epsilon$. Then $M = M'$ and $\boldsymbol{\sigma} = 0$
Step: $\sigma = \tau t$ for some sequence τ and transition t. Let $M \xrightarrow{\tau} L \xrightarrow{t} M'$.
We have

$$\begin{array}{lll} M' & = L + \mathbf{t} & \text{(Definition of } \mathbf{t}\text{)} \\ & = L + \mathbf{N} \cdot \boldsymbol{t} & \text{(Definition of } \boldsymbol{t}\text{)} \\ & = M + \mathbf{N} \cdot \boldsymbol{\tau} + \mathbf{N} \cdot \boldsymbol{t} & \text{(Induction hyp.)} \\ & = M + \mathbf{N} \cdot (\boldsymbol{\tau} + \boldsymbol{t}) & \\ & = M + \mathbf{N} \cdot \boldsymbol{\tau t} & \text{(Definition of Parikh-vector)} \\ & = M + \mathbf{N} \cdot \boldsymbol{\sigma} & (\sigma = \tau t) \end{array}$$

□

Example 4.2.5 In the previous net we have $(11000) \xrightarrow{t_1 t_2 t_3} (10001)$, and

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

The marking reached by firing a sequence σ from a marking M depends only on the Parikh-vector $\boldsymbol{\sigma}$.

In other words, if M enables two sequences σ and τ with $\boldsymbol{\sigma} = \boldsymbol{\tau}$, then both σ and τ lead to the same marking.

Definition 4.2.6 (Marking Equation)
The Marking Equation of a Petri net (N, M_0) is $M = M_0 + \mathbf{N} \cdot X$ with variables M and X.

The Marking equation leads to the following semi-algorithms for Boundedness, b-Boundedness, (Non)-Reachability, and Deadlock-freedom:

Proposition 4.2.7 (A sufficient condition for boundedness)
Let (N, M_0) be a Petri net. If the optimization problem

$$\begin{array}{ll} \textit{maximize} & \sum_{s \in S} M(s) \\ \textit{subject to} & M = M_0 + \mathbf{N} \cdot X \end{array}$$

has an optimal solution, then (N, M_0) is bounded.

Proof. Let n be the optimal solution of the problem.

Then $n \geq \sum_{s \in S} M(s)$ holds for every marking M for which there exists a vector X such that $M = M_0 + \mathbf{N} \cdot X$.

By Lemma **??** we have $n \geq \sum_{s \in S} M(s)$ for every *reachable* marking M, and so $n \geq M(s)$ for every reachable marking M and every place s. □

Exercise: Change the algorithm so that it checks whether a given place is bounded.

Proposition 4.2.8 (A sufficient condition for non-reachability)
Let (N, M_0) be a Petri net and let L be a marking of N.
If the equation

$$L = M_0 + \mathbf{N} \cdot X \qquad (\textit{with only } X \textit{ as variable})$$

has no solution, then L is not reachable from M_0.

Proof. Immediate consequence of Lemma **??**. □

Proposition 4.2.9 (A sufficient condition for deadlock-freedom)
Let (N, M_0) be a 1-bounded Petri net where $N = (S, T, F)$. If the following system of equations has no solution then (N, M_0) is deadlock-free.

$$M = M_0 + \mathbf{N} \cdot X$$
$$\sum_{s \in {}^\bullet t} M(s) < |{}^\bullet t| \ \textit{for every transition } t.$$

Proof. We show: if there is a reachable dead marking M, then M is a solution of the system.

Let M be a reachable dead marking. By Lemma **??** there is a vector X satisfying $M = M_0 + \mathbf{N} \cdot X$.

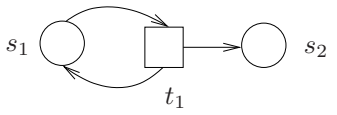
Let t be an arbitrary transition. We prove $\sum_{s \in {}^\bullet t} M(s) < |{}^\bullet t|$.

Since (N, M_0) is 1-bounded, we have $M(s) \leq 1$ for every place s. In particular, $\sum_{s \in {}^\bullet t} M(s) \leq |{}^\bullet t|$.

Since M does not enable t, we have $M(s) = 0$ for at least one place $s \in {}^\bullet t$, and so $\sum_{s \in {}^\bullet t} M(s) < |{}^\bullet t|$. □

The converses of these propositions do not hold (that is why they are semi-algorithms!). Counterexamples are:

- To Proposition **??**:



	t_1
s_1	0
s_2	1

  (N, M_0) ist bounded but

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

  holds for every n (that is, the Marking Equation has a solution for every marking of the form $(0, n)$).

- To Proposition **??**:

  Peterson's algorithm: the marking

$$(p_4, q_4, m_1{=}true, m_2{=}true, \textit{hold}{=}1)$$

  ist not reachable, but the Marking Equation has a solution.

  (Exercise: find a smaller example).

- To Proposition **??**:

  Peterson's algorithm with an additional transition t satisfying ${}^\bullet t = \{p_4, q_4\}$ and $t^\bullet = \emptyset$. The Petri net is deadlock free, but the Marking Equation has a solution for $(m_1{=}true, m_2{=}true, hold{=}1)$ that satisfies the conditions of the proposition.

  (Exercise: find a smaller example).

4.3 S- and T-invariants

4.3.1 S-invariants

Definition 4.3.1 (S-invariants)
Let $N = (S, T, F)$ be a net. An S-invariant of N (aka P-invariant, P-flow, or place invariant) is a vector $I : S \to \mathbb{Q}$ such that $I \cdot \mathbf{N} = 0$.

Proposition 4.3.2 (Fundamental property of S-invariants)
Let (N, M_0) be a Petri net and let I be a S-invariant of N. If $M_0 \xrightarrow{*} M$, then $I \cdot M = I \cdot M_0$.

Proof. We have $M_0 \xrightarrow{\sigma} M$ for some firing sequence σ. By the Marking Equation Lemma we get

$$M = M_0 + \mathbf{N} \cdot \boldsymbol{\sigma}$$

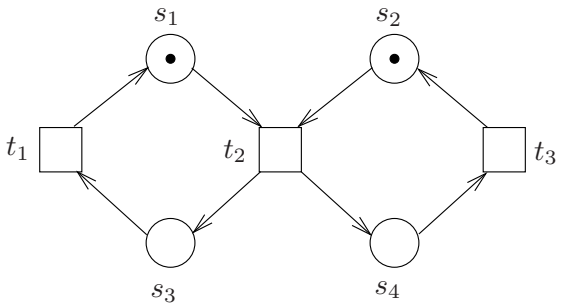
and so

$$\begin{array}{rcll} I \cdot M & = & I \cdot M_0 + I \cdot \mathbf{N} \cdot \boldsymbol{\sigma} & \text{(Marking Equation)} \\ & = & I \cdot M_0 & (I \cdot \mathbf{N} = 0) \end{array}$$

□

The value of the expression $I \cdot M$ is therefore the same for every reachable marking M, and so it constitutes an invariant of (N, M_0).

Example 4.3.3 *We* compute the S-invariants of



The incidence matrix is:

	t_1	t_2	t_3
s_1	1	-1	0
s_2	0	-1	1
s_3	-1	1	0
s_4	0	1	-1

We compute the solutions of the system of equations

$$(i_1, i_2, i_3, i_4) \cdot \begin{pmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \\ -1 & 1 & 0 \\ 0 & 1 & -1 \end{pmatrix} = 0$$

The general form of the S-invariants is therefore (x, y, x, y) with $x, y \in \mathbb{Q}$.

The following propositions are immediate consequences of the definition of S-invariants:

Proposition 4.3.4 *The S-invariants of a net form a vector space over the rational numbers.*

This definition of S-invariant is very suitable for machines, but not for humans, who can only solve very small systems of equations by hand.

There is an equivalent definition which allows people to decide, even for nets with several dozens of places, if a given vector is an S-invariant.

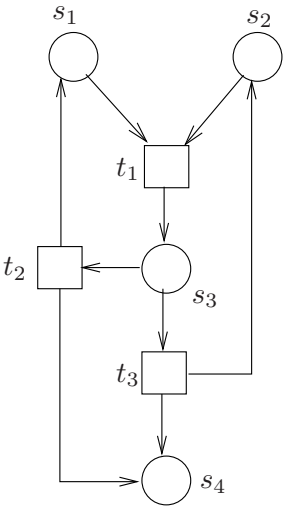
Proposition 4.3.5 *I is an S-invariant of $N = (S, T, F)$ iff*

$$\sum_{s \in {}^\bullet t} I(s) = \sum_{s \in t^\bullet} I(s)$$

holds for every transition $t \in T$.

Proof. $I \cdot \mathbf{N} = 0$ is equivalent to $I \cdot \mathbf{t} = 0$ for every transition t, and we have $I \cdot \mathbf{t} = \sum_{s \in t^\bullet} I(s) - \sum_{s \in {}^\bullet t} I(s)$. □

Example 4.3.6 *We* show that $I = (1, 1, 2, 1)$ is an S-invariant of



The condition of Proposition **??** must hold for transitions t_1, t_2 und t_3.

- Transition t_1: $I(s_1) + I(s_2) = I(s_3) = 2$.
- Transition t_2: $I(s_3) = I(s_1) + I(s_4) = 2$.
- Transition t_3: $I(s_3) = I(s_4) + I(s_2) = 2$.

With the help of S-invariants we can obtain

- a sufficient condition for boundedness,
- a necessary conditions for liveness, and
- a necessary condition for the reachability of a marking.

Definition 4.3.7 (Semi-positive and positive S-invariants)
Let I be an S-invariant of $N = (S, T, F)$.

I is semi-positive if $I \geq 0$ and $I \neq 0$.

I is positive if $I > 0$ (that is, if $I(s) > 0$ for every $s \in S$).

The support of an S-invariant is the set $\langle I \rangle = \{s \in S \mid I(s) > 0\}$.

Training Exercises on AutomataTutor:

- ☆ 🔍 Semipositive Place Invariant 1: Find a semipositive place invariant.
- ☆ 🔍 Semipositive Place Invariant 2: Find three different minimal semipositive place invariants.
- ★ 🔍 Positive Place invariant: Find a positive place invariant.

Proposition 4.3.8 (A sufficient condition for boundedness)
Let (N, M_0) be a Petri net. If N has a positive S-invariant I, then (N, M_0) is bounded. More precisely: (N, M_0) is n-bounded for

$$n = max\left\{\frac{I \cdot M_0}{I(s)} \mid s \text{ is a place of } N\right\}$$

Proof. Let M be any reachable marking.

By the fundamental property of S-invariants we have $I \cdot M = I \cdot M_0$.

Let s be an arbitrary place of N.

Since $I > 0$ we have $I(s) \cdot M(s) \leq I \cdot M = I \cdot M_0$

and so $M(s) \leq \frac{I \cdot M_0}{I(s)}$. □

Proposition 4.3.9 (A necessary condition for liveness)
If (N, M_0) is live, then $I \cdot M_0 > 0$ for every semi-positive S-invariant of N.

Proof. Let I be a semi-positive S-invariant and let s be a place of $\langle I \rangle$.

Since (N, M_0) is live, some reachable marking M satisfies $M(s) > 0$.

Since I is semi-positive, we have $I \cdot M \geq I(s) \cdot M(s) > 0$.

Since I is a S-invariant, we get $I \cdot M_0 = I \cdot M > 0$ □

These two propositions lead immediately to semi-algorithms for Boundedness and Liveness.

Definition 4.3.10 (The $\sim$ relation)
Let M and L be markings and let I be a S-invariant of a net N.

M und L agree on I if $I \cdot M = I \cdot L$.

We write $M \sim L$ if M and L agree on all invariants of N.

Proposition 4.3.11 (A necessary condition for reachability)
Let (N, M_0) *be a Petri net.* $M \sim M_0$ *holds for every* $M \in [M_0\rangle$.

Proof. Follows from the fundamental property of S-invariants. □

Training Exercise in AutomataTutor:
☆ ■ Decide Reachability with place invariants: Decide whether the given marking is reachable or unreachable. Proof it by giving either a firing sequence that reaches the marking or a place invariant that shows unreachability.

The following theorem allows one to decide if $M \sim L$ holds for two given markings M and L.

Theorem 4.3.12 *Let N be a net and let M, L be two markings of N. $M \sim L$ iff the equation $M = L + \mathbf{N} \cdot X$ has a rational solution.*

Proof. ($\Rightarrow$): $M \sim L$ implies $I \cdot (M - L) = 0$ for every S-invariant I.

We now recall a well-known theorem of linear algebra. Given a $n \times m$ matrix A, the sets

$$\begin{aligned} U =& \{u \in \mathbb{Q}^n \mid u \cdot A = 0\} \\ V =& \{v \in \mathbb{Q}^n \mid u \cdot v = 0 \text{ for every } u \in U\} \end{aligned}$$

are vector spaces, and the columns of A contain a basis of V.

If we take $A := \mathbf{N}$, then U is the set of S-invariants of N.

So, by the theorem, the columns of $\mathbf{N}$ contain a basis of the vector space of vectors v satisfying $I \cdot v = 0$ for every S-invariant I.

In particular, since $(M - L)$ is one of these vectors, $(M - L)$ is a linear combination over $\mathbb{Q}$ of the columns of $\mathbf{N}$.

So the equation $\mathbf{N} \cdot X = (M - L)$ has a rational solution.

($\Leftarrow$) : Assume $M = L + \mathbf{N} \cdot X$ has a rational solution X_0.

Let I be an S-invariant of N.

Since $I \cdot \mathbf{N} = 0$ we have

$$I \cdot M = I \cdot (L + \mathbf{N} \cdot X_0) = I \cdot L + \mathbf{0} = I \cdot L \, .$$

□

More generally, we have:

$$
\begin{array}{c}
M \text{ is reachable from } L \\
\not\Uparrow \quad \Downarrow \\
M = L + \mathbf{N} \cdot X \text{ has a solution } X \in \mathbb{N}^{|T|} \\
\not\Uparrow \quad \Downarrow \\
M = L + \mathbf{N} \cdot X \text{ has a solution } X \in \mathbb{Q}^{|T|} \\
\Updownarrow \\
M \sim L
\end{array}
$$

4.3.2 T-invariants

Definition 4.3.13 (T-invariants)
Let $N = (S, T, F)$ be a net. A T-invariant of N (aka T-flow) is a vector $J : T \to \mathbb{Q}$ such that $\mathbf{N} \cdot J = 0$.

Proposition 4.3.14 *J is a T-invariant of $N = (S, T, F)$ iff*

$$\sum_{t \in {}^\bullet s} J(t) = \sum_{t \in s^\bullet} J(t)$$

for every place $s \in S$.

Proposition 4.3.15 (Fundamental property of T-invariants)
Let N be a net, let M be a marking of N, and let σ be a sequence of transitions of N enabled at M.
The vector $\boldsymbol{\sigma}$ is a T-invariant of N iff $M \xrightarrow{\sigma} M$.

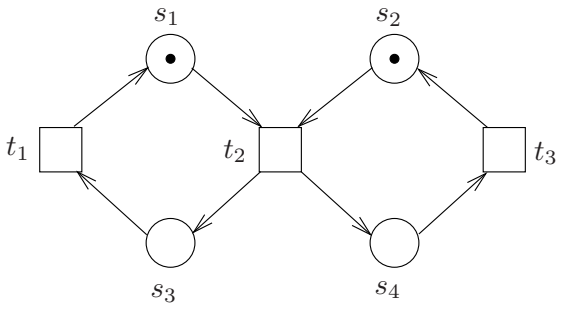
Proof. $(\Rightarrow)$: Let M' be the marking satisfying $M \xrightarrow{\sigma} M'$.

By the Marking Equation we have $M' = M + \mathbf{N} \cdot \boldsymbol{\sigma}$. Since $\mathbf{N} \cdot \boldsymbol{\sigma} = 0$ we get $M' = M$

$(\Leftarrow)$: By the Marking Equation we have $M = M + \mathbf{N} \cdot \boldsymbol{\sigma}$.

So $\mathbf{N} \cdot \boldsymbol{\sigma} = 0$. □

We compute the T-invariants of



as the solutions of the system of equations

$$\begin{pmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \\ -1 & 1 & 0 \\ 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} j_1 \\ j_2 \\ j_3 \end{pmatrix} = 0$$

The general form of the T-invariants is (x, x, x), where $x \in \mathbb{Q}$.

Training Exercises on AutomataTutor:

- ☆ 🔍 Positive Transition Invariant 1: Find a positive transition invariant.
- ☆ 🔍 Semipositive Transition Invariant: Find a semipositive transition invariant.
- ★ 🔍 Positive Transition invariant 2: Find a positive transition invariant.

Using T-invariants we obtain a necessary condition for well-formedness:

Theorem 4.3.16 (Necessary condition for well-formedness)
Every well-formed net has a positive T-invariant.

Proof. Let N be a well-formed net and let M_0 be a live and bounded marking of N.

By liveness there is an infinite firing sequence $\sigma_1\sigma_2\sigma_3\cdots$ such that every σ_i is a finite firing sequence containing all transitions of N.

We have

$$M_0 \xrightarrow{\sigma_1} M_1 \xrightarrow{\sigma_2} M_2 \xrightarrow{\sigma_3} \dots$$

By boundedness there are indices $i < j$ such that $M_i = M_j$. So the sequence $\sigma_{i+1} \dots \sigma_j$ satisfies

$$M_i \xrightarrow{\sigma_{i+1}\dots\sigma_j} M_i$$

and so $J = \boldsymbol{\sigma}_{i+1} + \dots + \boldsymbol{\sigma}_j$ is a T-invariant of N.

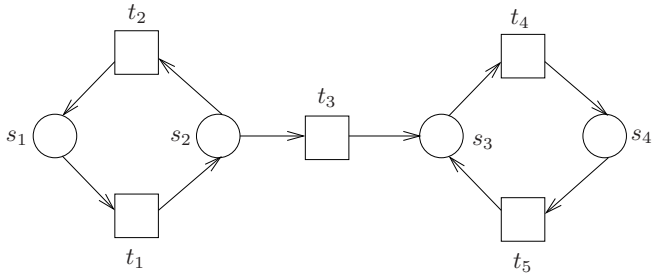
J is positive because every transition occurs at least once in $\sigma_{i+1} \dots \sigma_j$. $\square$

4.4 Siphons and Traps

4.4.1 Siphons

Definition 4.4.1 (Siphon)
Let $N = (S, T, F)$ be a net. A set $R \subseteq S$ of places is a siphon of N if ${}^\bullet R \subseteq R^\bullet$. A siphon R is proper if $R \neq \emptyset$.

Example 4.4.2 *T*he set $\{s_1, s_2\}$ is a siphon of



$$
\begin{array}{lllll}
{}^\bullet\{s_1, s_2\} & = & {}^\bullet s_1 \cup {}^\bullet s_2 & = & \{t_2\} \cup \{t_1\} & = & \{t_1, t_2\} \\
\{s_1, s_2\}^\bullet & = & s_1^\bullet \cup s_2^\bullet & = & \{t_1\} \cup \{t_2, t_3\} & = & \{t_1, t_2, t_3\}
\end{array}
$$

Training Exercises in Automata Tutor:

- ☆ 🔍 Minimal siphon: Find a proper minimal siphon of the Petri net.
- ☆ 🔍 Siphon containing a certain place: Find a proper siphon containing at least one specified place.
- ☆ 🔍 Largest siphon: Find the largest siphon of the Petri net.

Proposition 4.4.3 (Fundamental property of siphons)
Let R be a siphon of a net N, and let $M \xrightarrow{\sigma} M'$ be a firing sequence of N. If $M(R) = 0$, then $M'(R) = 0$.

Proof. Since ${}^\bullet R \subseteq R^\bullet$, the transitions that can mark R can only occur at markings that already mark R. □

Loosely speaking, a siphon that becomes unmarked (or "empty"), remains unmarked forever.

Corollary 4.4.4 (A necessary condition for reachability)
If M is reachable in (N, M_0), then for every siphon R, if $M_0(R) = 0$ then $M(R) = 0$.

We can easily check in polynomial time if this condition holds.

We first observe that, if R_1 and R_2 are siphons of N, then so is $R_1 \cup R_2$.

It follows that there exists a unique largest siphon Q_0 unmarked at M_0 (more precisely, $R \subseteq Q_0$ for every siphon R such that $M_0(R) = 0$).

We claim that the condition of Corollary **??** holds for every siphon R if and only if $M(Q_0) = 0$.

- Assume the condition holds for every siphon R.
  Since $M_0(Q_0) = 0$ by definition, we get $M(Q_0) = 0$.

- Assume the condition does not hold.
  Then there is a siphon R such that $M_0(R) = 0$ and $M(R) > 0$.
  Since $R \subseteq Q_0$, we also have $M(Q_0) > 0$.

The siphon Q_0 can be determined with the help of the algorithm in the next slide, which computes the largest siphon contained in a given set R of places—it suffices then to choose R as the set of places unmarked at M_0.

Training Exercises in Automata Tutor:

- ☆ ■ Decide reachability with siphons: Decide whether the given marking is reachable or unreachable. Proof it by giving either a firing sequence that reaches the marking or a siphon that shows unreachability.
- ★ ■ Decide Reachability with siphons and place invariants: Decide whether the given marking is reachable or unreachable. Proof it by giving either a firing sequence that reaches the marking or a place invariant or siphon that shows unreachability.

Input: A net $N = (S, T, F)$ and $R \subseteq S$.
Output: The largest siphon $Q \subseteq R$.
Initialization: $Q := R$.

begin
  while there are $s \in Q$ and $t \in {}^\bullet s$ such that $t \notin Q^\bullet$ **do**
    $Q := Q \setminus \{s\}$
  endwhile
end

Exercise: Show that the algorithm is correct. That is, prove that

- the algorithm terminates, and
- after termination Q is the largest siphon contained in R.

Siphons lead to a necessary condition for liveness, and a sufficient condition for deadlock-freedom.

Proposition 4.4.5 (A necessary condition for liveness)
If (N, M_0) is live, then M_0 marks every proper siphon of N.

Proof. Let R be a proper siphon of N and let $s \in R$.

Since we assume that N is connected, ${}^\bullet s \cup s^\bullet \neq \emptyset$, and, since R is a siphon, $s^\bullet \neq \emptyset$.

Let $t \in s^\bullet \neq \emptyset$. By liveness some reachable marking enables t, and so some reachable marking marks s, and therefore also the siphon R.

By Proposition **??** the initial marking M_0 also marks R. □
Again, the condition can be checked with the help of the algorithm above: the condition holds if and only if $Q_0 = \emptyset$.

We also obtain a sufficient condition for deadlock-freedom, but not one that is easy to check.

Proposition 4.4.6 *If M is a dead marking of N, then the set of places unmarked at M is a proper siphon of N.*

Proof. Let $R = \{s \mid M(s) = 0\}$. Since M is dead, $R \neq \emptyset$.

We show ${}^\bullet R \subseteq R^\bullet$.

For every transition t there is a place $s \in {}^\bullet t$ such that $M(s) = 0$ (otherwise t would be enabled). We have $s \in R$.

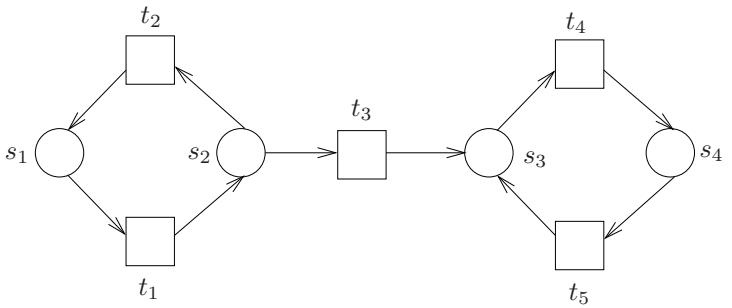
So $R^\bullet$ contains all transitions of N, and therefore ${}^\bullet R \subseteq R^\bullet$. □

Corollary 4.4.7 (A sufficient condition for deadlock-freedom) *Let (N, M_0) be a Petri net. If every reachable marking marks all proper siphons of N, then (N, M_0) is deadlock-free.*

4.4.2 Traps

Definition 4.4.8 (Trap)
Let $N = (S, T, F)$ be a trap. A set $R \subseteq S$ of places is a *trap* if $R^\bullet \subseteq {}^\bullet R$. A trap R is *proper* if $R \neq \emptyset$.

Example 4.4.9 The set $\{s_3, s_4\}$ is a trap of



Training Exercises in Automata Tutor:

- ☆ 🔍 Minimal trap: Find a proper minimal trap of the Petri net.
- ★ 🔍 Trap containing a certain place: Find a proper trap containing at least one specified place.
- ★ 🔍 Largest trap: Find the largest trap of the Petri net.

Proposition 4.4.10 *[Fundamental property of traps]*
Let R be a trap of a net N and let $M \xrightarrow{\sigma} M'$ be a firing sequence of N. If $M(R) > 0$, then $M'(R) > 0$.

Proof. Since ${}^\bullet R \subseteq {}^\bullet R$, transitions that take tokens from R put tokens in R. □

So, loosely speaking, marked traps stay marked.

Notice, however, that this does not mean that the number of tokens of a trap cannot decrease. The number can go up or down, just not become 0.

Traps lead to a necessary condition for reachability:

Corollary 4.4.11 (A necessary condition for reachability)
If M is reachable in (N, M_0), then for every trap R, if $M_0(R) > 0$ then $M(R) > 0$.

As in the case of siphons, we can check in polynomial time if this condition holds.

If R_1 and R_2 are traps of N, then so is $R_1 \cup R_2$.

So there exists a unique largest trap Q_0 unmarked at M (more precisely, $R \subseteq Q_0$ for every trap R such that $M(R) = 0$). This condition holds if and only if $M_0(Q_0) = 0$.

To compute the largest trap unmarked at M, we transform the algorithm that computes the largest siphon contained in a given set of places into an algorithm for computing the largest trap (exercise).

Training Exercises in Automata Tutor:

- ☆ ■ Decide reachability with traps: Decide whether the given marking is reachable or unreachable. Proof it by giving either a firing sequence that reaches the marking or a trap that shows unreachability.
- ★ ■ Decide Reachability in any way: Decide whether the given marking is reachable or unreachable. Proof it by giving either a firing sequence that reaches the marking or a place invariant, trap or siphon that shows unreachability.

The sufficient condition for deadlock-freedom was computationally expensive, because of the form "for every reachable marking ...". Combining siphons and traps we obtain an easier-to-check condition.

Proposition 4.4.12 (A sufficient condition for deadlock-freedom)
Let (N, M_0) be a Petri net. If every proper siphon of N contains a trap marked at M_0, then (N, M_0) is deadlock-free.

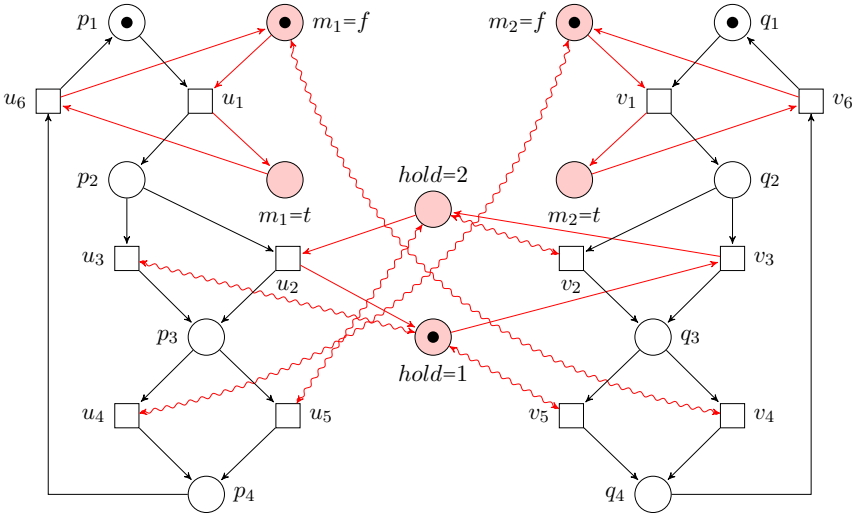
Proof. Easy consequence of Corollary **??** and Proposition **??**. □

The siphon-trap condition cannot be checked in polynomial time unless P=NP (whether every proper siphon contains a marked trap is an NP-complete problem), but can be checked with the help of a SAT-solver (see "New algorithms for deciding the siphon-trap property" by O. Oanea, H. Wimmel, and K. Wolf).

Proving mutual exclusion of Peterson's algorithm

We show how to combine S-invariants and traps to prove that Peterson's algorithm satisfies the mutual exclusion property.

The Petri net model is (see Chapter 2):



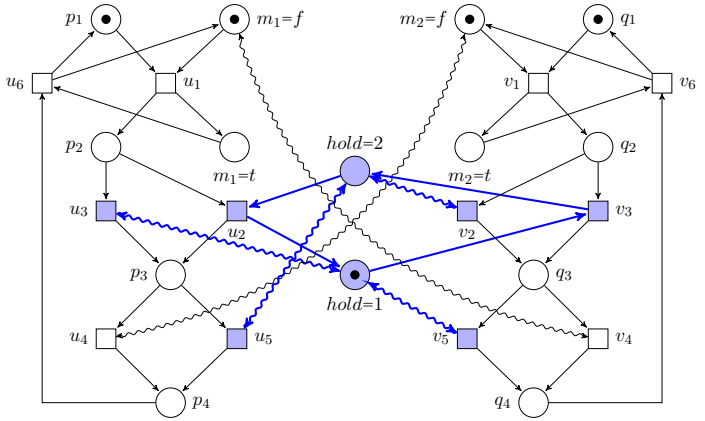
Recall that the the algorithm satisfies the mutual exclusion property iff no reachable marking M of the Petri net model satisfies

$$M(p_4) \geq 1 \wedge M(q_4) \geq 1 \, .$$

We first compute some constraints derived from S-invariants and traps.

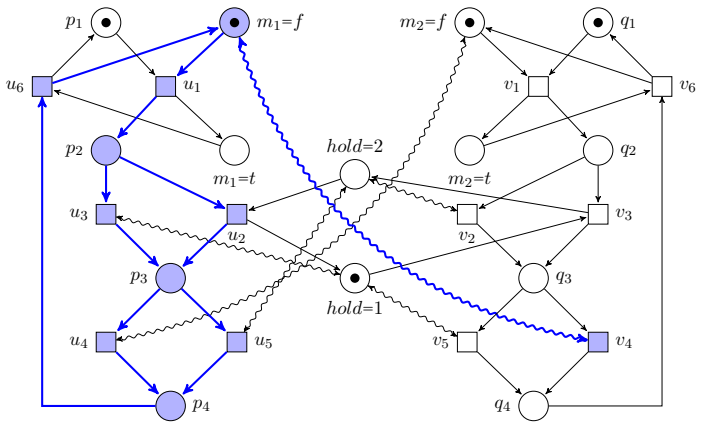
Constraints from S-invariants

$$(1) \quad M(hold{=}1) + M(hold{=}2) = 1$$



$$(2) \quad M(p_2) + M(p_3) + M(p_4) + M(m_1{=}f) = 1$$

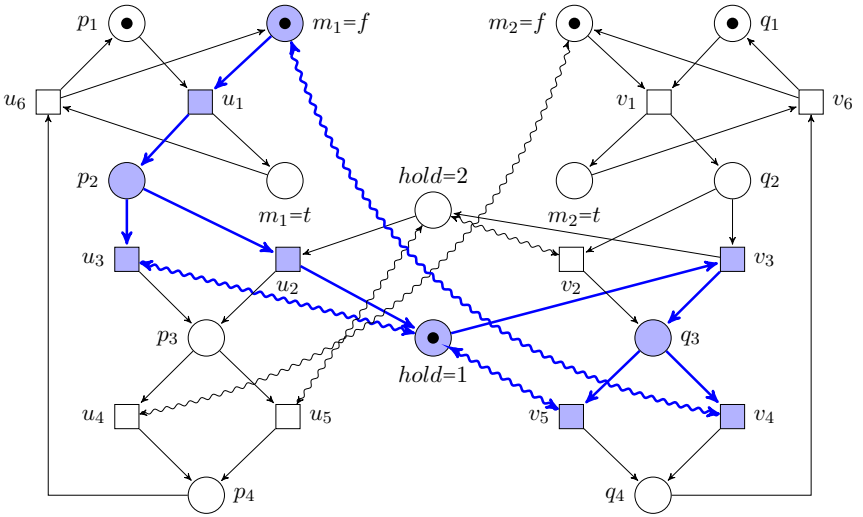
$$(3) \quad M(q_2) + M(q_3) + M(q_4) + M(m_2{=}f) = 1$$



Constraints from traps

$$(4) \quad M(m_1{=}f) + M(p_2) + M(hold{=}1) + M(q_3) > 0$$

$$(5) \quad M(m_2{=}f) + M(q_2) + M(hold{=}2) + M(p_3) > 0$$



The set of places

$$\{m_1{=}f,\ p_2, hold{=}1,\ q_3\}$$

(shown in blue in the picture) is a trap.

Every transition that removes some token from the set (shown in blue again) also add some token to the set.

Proving mutual inclusion

Assume that some reachable marking M violates the mutual exclusion property. Then M satisfies all of

$$
\begin{array}{ll}
(0) & M(p_4) \geq 1 \wedge M(q_4) \geq 1 \\
(1) & M(hold{=}1) + M(hold{=}2) = 1 \\
(2) & M(p_2) + M(p_3) + M(p_4) + M(m_1{=}f) = 1 \\
(3) & M(q_2) + M(q_3) + M(q_4) + M(m_2{=}f) = 1 \\
(4) & M(m_1{=}f) + M(p_2) + M(hold{=}1) + M(q_3) > 0 \\
(5) & M(m_2{=}f) + M(q_2) + M(hold{=}2) + M(p_3) > 0
\end{array}
$$

We show this is a contradiction:

$$
\begin{array}{c}
M(p_4) \geq 1 \wedge M(q_4) \geq 1 \\
\Longrightarrow \text{ ((2) and (3))} \\
M(p_2) + M(p_3) + M(m_1{=}f) = 0 \\
\wedge \\
M(q_2) + M(q_3) + M(m_2{=}f) = 0 \\
\Longrightarrow \{ \text{ ((1)) } \} \\
M(p_2) + M(p_3) + M(m_1{=}f) = 0 \\
\wedge \\
M(q_2) + M(q_3) + M(m_2{=}f) = 0 \\
\wedge \\
(\ M(hold{=}1) = 0 \ \vee \ M(hold{=}2) = 0\ ) \\
\Longrightarrow \{ \text{ (logic and arithmetic) } \}
\end{array}
$$

$$
\begin{array}{ccc}
M(m_1 = f) + M(p_2) + & & M(m_2 = f) + M(q_2) + \\
M(hold = 1) + M(q_3) = 0 & \vee & M(hold = 2) + M(p_3) = 0 \\
\text{contradicts (4)} & & \text{contradicts (5)}
\end{array}
$$

Automatizing the proof process

We describe a procedure that automatically computes the proof of mutual exclusion.

We assume that the set of markings that violate the property can be expressed as the markings satisfying a system V of linear constraints (linear equations or inequations). In our case $V = \{M(p_4) \geq 1, M(q_4) \geq 1\}$.

We maintain a second system of linear constraints $\mathcal{L}$ that is satisfied by all reachable markings. Initially $\mathcal{L} = \emptyset$.

In the j-th iteration of the procedure:

- We compute a marking M_j (not necessarily reachable!) satisfying all constraints of $\mathcal{L}$ and V.

  If no such marking exists, then every reachable marking satisfies the property.

- Otherwise, we compute an S-invariant I_j or a trap R_j such that M_j does not satisfy the linear constraint L_j for I_j or R_j.

  If no such S-invariant or trap exist, we stop without a conclusion.

- Otherwise, we set $\mathcal{L} := \mathcal{L} \cup \{L_j\}$.

The marking M_j is computed by solving the set of linear equations corresponding to V and the current constraints of $\mathcal{L}$.

The existence of an S-invariant is checked by solving the systems

$$\begin{array}{rcl} I_j \cdot \mathbf{N} & = & \mathbf{0} \\ I_j \cdot M_j & < & I_j \cdot M_0 \end{array} \qquad \begin{array}{rcl} I_j \cdot \mathbf{N} & = & \mathbf{0} \\ I_j \cdot M_j & > & I_j \cdot M_0 \end{array}$$

The existence of a trap is checked by computing the largest trap Q_j contained in the set $\{s \mid M_j(s) = 0\}$, and checking if $M_0(Q_j) > 0$.

In our example, the procedure can produce the following results:

- $V = \{M(p_4) \geq 1, M(q_4) \geq 1\}$.
- $M_1 = \{p_4, q_4\}$ satisfies V.
- We compute (1): $M(hold{=}1) + M(hold{=}2) = 1$,
  which is not satisfied by M_1, from an S-invariant.
- $M_2 = \{p_4, q_4, hold{=}1\}$ satisfies (1) and V.
- We compute (5): $M(m_2{=}f) + M(q_2) + M(hold{=}2) + M(p_3) > 0$,
  which is not satisfied by M_2, from a trap.
- $M_3 = \{p_4, q_4, hold{=}1, m_2{=}f\}$ satisfies (1), (5), and V.
- We compute (3): $M(q_2) + M(q_3) + M(q_4) + M(m_2{=}f) = 1$,
  which is not satisfied by M_3, from an S-invariant.
- $M_4 = \{p_4, q_4, hold{=}2, m_1{=}f\}$ satisfies (1), (5), (3), and V.
- We compute (2): $M(p_2) + M(p_3) + M(p_4) + M(m_1{=}f) = 1$,
  which is not satisfied by M_4, from an S-invariant.
- $M_5 = \{p_4, q_4, hold{=}2\}$ satisfies (1), (5), (3), (2), and V.
- We compute (4): $M(m_1{=}f) + M(p_2) + M(hold{=}1) + M(q_3) > 0$,
  which is not satisfied by M_5, from a trap.
- There is no reachable marking satisfying (1)-(5) and V.

Chapter 5

Petri net classes with efficient decision procedures

In the three sections of this chapter we study three classes of Petri nets: S-systems, T-systems, and free-choice systems.

All sections have a similar structure. After the definition of the class, we introduce three theorems: the Liveness, Boundedness, and Reachability Theorems.

The Liveness Theorem characterizes the live Petri nets in the class.

The Boundedness Theorem characterizes the live and bounded systems.

The Reachability Theorem characterizes the reachable markings of the live and bounded systems.

The proofs of the theorems requires some results about the structure of S- and T-invariants of the class, which we also present.

The theorems immediately yield decision procedures for Liveness, Boundedness and Reachability whose complexity is much lower than those for general Petri nets.

Why boundedness only for live Petri nets, and reachability only for live and bounded Petri nets?

First, in many application areas, a Petri net model of a correct system must typically be live *and* bounded, and so, if one of these properties fails, it does not make sense to check the other.

Second, the general characterization of the bounded systems or the reachable markings is less elegant than the corresponding characterization for live or live and bounded Petri nets.

Third, the computational complexity also improves in some cases. For example, Reachability for arbitrary free-choice systems is non-elementary, but it is NP-complete for live and bounded free-choice Petri nets.

The proofs of the theorems are very easy for S-systems, a bit more involved for T-systems, and relatively complex for free-choice systems. For this reason we just sketch the proofs for S-systems, explain the proofs in detail for T-systems, and omit some of the proofs for free-choice systems.

5.1 S-Systems

Definition 5.1.1 (S-nets, S-systems)
A net $N = (S, T, F)$ is a S-net if $|{}^\bullet t| = 1 = |t^\bullet|$ for every transition $t \in T$.
A Petri net (N, M_0) is a S-system if N is a S-net.

Training Exercise in AutomataTutor:
☆ Not deadlock-free S-system: Construct a S-System that is not deadlock-free.

Proposition 5.1.2 (Fundamental property of S-systems)
Let (N, M_0) be a S-system where $N = (S, T, F)$.
$M_0(S) = M(S)$ holds for every reachable marking M.

Proof. Every transition consumes one token and produces one token. □

Theorem 5.1.3 (Liveness Theorem for S-systems)
Let (N, M_0) be a S-system where $N = (S, T, F)$.
(N, M_0) is live iff N is strongly connected and $M_0(S) > 0$.

Proof. (Sketch.)
($\Rightarrow$): Assume N is not strongly connected. Then there is an arc (s, t) such that N has no path from t to s. We show that t is not live.

For every marked place s' such that there is a path from s' to s, fire the transitions of the path to bring the tokens in s' to s, and then fire transition t until s contains no tokens.

We have then reached a marking from which no tokens can "travel" back to s, and so t cannot occur again. So t is not live.

If M_0 marks no places, then no transition can occur, and (N, M_0) is not live.

($\Leftarrow$): If N is strongly connected and M_0 puts at least one token somewhere, then the token can freely move, reach any other place, and so enable any transition again. $\square$

Theorem 5.1.4 (Boundedness Theorem for S-systems)
Let (N, M_0) be a live S-system where $N = (S, T, F)$.
(N, M_0) is b-bounded iff $M_0(S) \leq b$.

Proof. Trivial. □

Exercise: give a counterexample for non-live S-systems.

Theorem 5.1.5 (Reachability Theorem for S-systems)
Let (N, M_0) be a live S-system where $N = (S, T, F)$, and let M be a marking of N. M is reachable from M_0 iff $M_0(S) = M(S)$.

Proof. N is strongly connected by Proposition **??**. So tokens can move to any place, and reach any marking M such that $M(S) = M_0(S)$. □

We characterize the S-invariants of S-nets. Recall that by assumption nets are connected.

Proposition 5.1.6 (S-invariants of S-nets)
Let $N = (S, T, F)$ be a S-net. A vector $I : S \to \mathbb{Q}$ is a S-invariant of N iff $I = (x, \ldots, x)$ for some $x \in \mathbb{Q}$.

Proof. Each transition $t \in T$ has exactly one input place s_t and an output place s'_t. So we have

$$\sum_{s \in {}^\bullet t} I(s) = I(s_t) \quad \text{and} \quad \sum_{s \in t^\bullet} I(s) = I(s'_t)$$

and therefore

$$\begin{aligned}
& I \text{ is a S-invariant} \\
\Leftrightarrow\ & \forall t \in T : I(s_t) = I(s'_t) && \text{(Def. of S-invariant and property above)} \\
\Leftrightarrow\ & \forall s_1, s_2 \in S : I(s_1) = I(s_2) && (N \text{ is connected}) \\
\Leftrightarrow\ & \exists x \in \mathbb{Q}\ \forall s \in S : I(s) = x
\end{aligned}$$

□

5.2 T-systems

Definition 5.2.1 (T-nets, T-systems)
A net $N = (S, T, F)$ is a T-net if $|{}^\bullet s| = 1 = |s^\bullet|$ for every place $s \in S$.
A system (N, M_0) is a T-system if N is a T-net.

Notation: We let γ denote a circuit of a net N. Given a marking M, we let $M(\gamma)$ denote the number of tokens of γ under M, that is,

$$M(\gamma) = \sum_{s \in \gamma} M(s) \ .$$

Training Exercises in Automata Tutor:

- ☆ Unbounded T-system: Create an unbounded T-system.
- ☆ Not deadlock-free T-system: Create a T-system that is not deadlock-free.

Proposition 5.2.2 (Fundamental property of T-systems)
Let γ be a circuit of a T-system (N, M_0) and let M be a reachable marking. Then $M(\gamma) = M_0(\gamma)$.

Proof. Firing a transition does not change the number of tokens of γ.

If the transition does not belong to the circuit, then the distribution of tokens in the circuit does not change.

If the transition belongs to the circuit, then it removes one token from a place of the circuit, and adds a token to another place. So the token count does not change. □

5.2.1 Liveness

Theorem 5.2.3 (Liveness Theorem for T-systems)
A T-system (N, M_0) is live iff $M_0(\gamma) > 0$ for every circuit γ of N.

Proof. ($\Rightarrow$): Let γ be a circuit with $M_0(\gamma) = 0$.

By Proposition **??** we have $M(\gamma) = 0$ for every reachable marking M.

So no transition of γ can ever occur, and (N, M_0) is not live.

($\Leftarrow$): Let t be an arbitrary transition and let M be a reachable marking. We show that some marking reachable from M enables t.

Let S_M be the set of places s of N such that some path leading from s to t contains no place marked at M.

We proceed by induction on $|S_M|$.

Basis: $|S_M| = 0$. Then $M(s) > 0$ for every place $s \in {}^\bullet t$. So M enables t.

Step: $|S_M| > 0$. By the fundamental property of T-systems, every circuit of N is marked at M. So there is a simple path Π such that:

(1) Π leads from some node (possibly t) to t.

(2) M marks no place of Π.

(3) Π has maximal length (no path longer than Π satisfies (1)-(2)).

Let u be the first element of Π. By (3), u is a transition, and M marks all places of ${}^\bullet u$. So M enables u.

Let $M \xrightarrow{u} L$. We show that $S_L \subset S_M$, and so that $|S_L| < |S_M|$.

We prove $S_L \subseteq S_M$ and $S_L \neq S_M$.

1. $S_L \subseteq S_M$

   Let $s \in S_L$. We show $s \in S_M$.

   By assumption there is a simple path $\Pi' = s \ldots t$ containing no place marked at L.

   Assume Π' contains a place r marked at M.

   Since $L(r) = 0$ and $M \xrightarrow{u} L$ we have $u \in r^\bullet$, and so $\{u\} = r^\bullet$. So u is the (unique) successor of r in Π'.

   Since $|S_M| > 0$, the marking M does not enable t. Since M enables u, we have $u \neq t$. So L marks the successor of u in Π', contradicting the definition of Π'.

2. $S_L \neq S_M$. Let s be the successor of u in Π. Then $s \in S_M$ but $s \notin S_L$, because $L(s) > 0$.

By induction hypothesis there is a firing sequence $L \xrightarrow{\sigma} L'$ such that L' enables t. It follows $M \xrightarrow{u} L \xrightarrow{\sigma} L'$, and so L' is a marking reachable from M that enables t. $\square$

5.2.2 Boundedness

Theorem 5.2.4 (Boundedness Theorem)
A place s of a live T-system (N, M_0) is b-bounded iff it belongs to some circuit γ such that $M_0(\gamma) \leq b$.

Proof. ($\Leftarrow$): Assume s belongs to some circuit γ such that $M_0(\gamma) \leq b$. By the fundamental property of T-systems the same holds for every reachable marking.

($\Rightarrow$): Assume s is b-bounded. Let M be a reachable marking such that $M(s)$ is maximal. We have $M(s) \leq b$.

Define the marking L as follows:

$$L(r) = \begin{cases} M(r) & \text{if } r \neq s \\ 0 & \text{if } r = s \end{cases}$$

We claim that (N, L) is not live.

Assume (N, L) is live. Then there is a firing sequence $L \xrightarrow{\sigma} L'$ such that $L'(s) > 0$.

By the Monotonicity Lemma, $M \xrightarrow{\sigma} M'$ for some marking M' satisfying

$$M'(s) = L'(s) + M(s) > M(s) \; .$$

This contradicts the maximality of $M(s)$, and proves the claim.

Since (N, L) is not live, by the Liveness Theorem some circuit γ is unmarked at L.

Since M is reachable from M_0, the system (N, M) is live, and so γ is marked at M.

Since L and M only differ in the place s, the circuit γ contains s, and s is the only place of γ marked at M.

So $M(\gamma) = M(s)$, and since $M(s) \leq b$ we get $M(\gamma) \leq b$. □

Corollary 5.2.5 *Let (N, M_0) be a live T-system*

1. *A place of N is bounded iff it belongs to some circuit.*
2. *Let s be a bounded place. Then*

$$max\{M(s) \mid M_0 \xrightarrow{*} M\} = min\{M_0(\gamma) \mid \gamma \text{ contains } s\}$$

3. *(N, M_0) is bounded iff N is strongly connected.*

Proof. Exercise □

5.2.3 Reachability

Proposition 5.2.6 (T-invariants of T-nets)
Let $N = (S, T, F)$ be a T-net. A vector $J : T \to \mathbb{Q}$ is a T-invariant iff $J = (x \dots x)$ for some $x \in \mathbb{Q}$.

Proof. Dual of the proof of Proposition **??**. □

Theorem 5.2.7 (Reachability Theorem)
Let (N, M_0) be a live T-system. A marking M is reachable from M_0 iff $M_0 \sim M$.

Proof. ($\Rightarrow$): Immediate consequence of the fundamental propety of S-invariants.

($\Leftarrow$) By Theorem **??** there is a rational vector X such that

$$M = M_0 + \mathbf{N}.X \tag{5.1}$$

Since $J = (1, 1, \dots, 1)$ is a T-invariant of N (Proposition **??**), we have

$$\mathbf{N} \cdot (X + \lambda J) = \mathbf{N} \cdot X$$

for every $\lambda \in \mathbb{Q}$.

So without loss of generality we can assume $X \geq 0$.

Let T be the set of transitions of N. We proceed in two steps:

(1) There is a vector $Y : T \to \mathbb{N}$ such that $M = M_0 + \mathbf{N} \cdot Y$.

Let Y be the vector $Y(t) := \lceil X(t) \rceil$ for every transition t.
(Where $\lceil x \rceil$ denotes the smallest integer larger than or equal to x).

By (**??**) , for every place s with $\{t_1\} = {}^\bullet s$ and $\{t_2\} = s^\bullet$ we have

$$M(s) = M_0(s) + X(t_1) - X(t_2) \ .$$

Since both $M(s)$ and $M_0(s)$ are integers we get

$$X(t_1) - X(t_2) = Y(t_1) - Y(t_2) \ .$$

So $M(s) = M_0(s) + Y(t_1) - Y(t_2)$, which implies $M = M_0 + \mathbf{N} \cdot Y$.

(2) $M_0 \xrightarrow{*} M$.

By induction on $|Y| = \sum_{t \in T} Y(t)$.

Basis: $|Y| = 0$. Then $Y = 0$ and $M = M_0$.

Step: $|Y| > 0$.

Claim: M_0 enables some transition of $\langle Y \rangle := \{t \in T \mid Y(t) > 0\}$.

Let

$$R = \{s \in {}^\bullet\langle Y \rangle \mid M_0(s) = 0\}$$

Let $s \in R$. Since $M_0(s) = 0$ and $M_0 + \mathbf{N} \cdot Y = M \geq 0$ we have:

> If some transition of $s^\bullet$ belongs to $\langle Y \rangle$, then some transition of ${}^\bullet s$ belongs to $\langle Y \rangle$.

Let Π be a path of maximal length containing places of R and transitions of $\langle Y \rangle$ (such a path exists, because otherwise N would contain a circuit unmarked at M_0).

By the property above, the first node of Π is a transition $t \in \langle Y \rangle$, and no place of ${}^\bullet t$ belongs to R.

So M_0 marks every place of ${}^\bullet t$, that is, M_0 enables t, and the claim is proved.

Let $M_0 \xrightarrow{t} M_1$. We have

$$M_1 + \mathbf{N}(Y - \boldsymbol{t}) = M$$

where

$$|Y - \boldsymbol{t}| = |Y| - 1 < |Y| \,.$$

By induction hypothesis we get $M_1 \xrightarrow{*} M$.

Since $M_0 \xrightarrow{t} M_1 \xrightarrow{*} M$, we get $M_0 \xrightarrow{*} M$.

□

5.2.4 Other useful results

Theorem 5.2.8 *Let N be a strongly connected T-net. For every marking M_0 the following statements are equivalent:*

(1) (N, M_0) *is live.*

(2) (N, M_0) *is deadlock-free.*

(3) (N, M_0) *has an infinite firing sequence.*

Proof. (1) $\Rightarrow$ (2) $\Rightarrow$ (3) follow from the definitions. We show (3) $\Rightarrow$ (1).

Let $M_0 \xrightarrow{\sigma}$ be an infinite firing sequence.

Claim: Every transition of N occurs in σ.

Since N is strongly connected, (N, M_0) is bounded (Theorem **??**).

Let $\sigma = t_1\, t_2\, t_3 \ldots$, and $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} \ldots$.

Since (N, M_0) is bounded, there are $i < j$ such that $M_i = M_j$.

Let σ_{ij} be the infix of σ containing the transitions between M_i and M_j.

By the fundamental property of T-invariants (Proposition **??**) $\boldsymbol{\sigma_{ij}}$ is a T-Invariant .

By Proposition **??** there is $n \in \mathbb{N}$ such that $\boldsymbol{\sigma_{ij}} = (n \ldots n)$.

So every transition of N occurs in σ_{ij}, and so the same holds for σ, proving the claim.

By the claim, every place and every circuit of N becomes marked during the execution of σ.

By the fundamental property of T-systems, all circuits are marked at M_0.

By the Liveness Theorem (Theorem **??**), (N, M_0) is live. $\square$

Theorem 5.2.9 (Genrich's Theorem)
Let N be a strongly connected T-net. There is a marking M_0 such that (N, M_0) is live and 1-bounded.

Proof. Since N is strongly connected, any marking that puts tokens on all places of N is live, because it marks all circuits (Liveness Theorem), and bounded, because all markings of N are bounded (Corollary **??**).

Let (N, M) be live and bounded, but not 1-bounded. We construct another live marking L of N satisfying the following two conditions:

(1) $L(\gamma) \leq M(\gamma)$ for every circuit γ of N, and

(2) $L(\gamma) < M(\gamma)$ for at least one circuit γ.

By Theorem **??**, at least one place of N has a smaller bound under L as under M. Iterating this construction we obtain a 1-bounded marking of N.

Let s be a non-1-bounded place of (N, M). Some reachable marking M' satisfies $M'(s) \geq 2$. Let L be the marking that puts exactly one token in s, and as many tokens as M elsewhere.

Since M is live, it marks all circuits of N. By construction L also marks all circuits, and so L is also live.

We show that L satisfies (1) and (2).

Condition (1) is a consequence of the definition of L.

Condition (2) holds for all circuits containing s (and there is at least one, because N is strongly connected). □

The Shortest Sequence Theorem for 1-bounded T-systems

We prove that for any two markings M_1, M_2 of a 1-bounded T-system (live or not), if M_2 is reachable from M_1, then it can be reached from M_1 in at most $n \cdot (n-1)/2$ steps, where n is the number of transitions of the T-system.

Definition 5.2.10 *Given a sequence σ of transitions, we let $\mathcal{A}(\sigma)$ denote the set of transitions occurring at least once in σ.*

Lemma 5.2.11 *Let (N, M_0) be a T-system. If M_0 enables $\sigma_1\,\sigma_2\,t$ for some $\sigma_1, \sigma_2 \in T^*$, and some $t \in T$ such that $t \notin \mathcal{A}(\sigma_1)$ and $\mathcal{A}(\sigma_2) \subseteq \mathcal{A}(\sigma_1)$, then M_0 also enables $\sigma_1\,t\,\sigma_2$.*

Proof. By induction on the length of σ_2.

If $|\sigma_2| = 0$ there is nothing to prove.

Assume $\sigma_2 = \sigma_2' u$ for some $u \in T$. We prove that M_0 enables $\sigma_1\,\sigma_2'\,t\,u$. The result follows by applying the induction hypothesis to $\sigma_1\,\sigma_2'\,t$.

Let $M_0 \xrightarrow{\sigma_1\,\sigma_2'} M_1 \xrightarrow{u} M_2 \xrightarrow{t} M_3$. Consider two cases:

Case 1: $u^\bullet \cap {}^\bullet t = \emptyset$. Then t is already enabled at M_1, and we are done.

Case 2: $u^\bullet \cap {}^\bullet t \neq \emptyset$. Let $s \in u^\bullet \cap {}^\bullet t$.

Since $u \in \mathcal{A}(\sigma_2)$ and $\mathcal{A}(\sigma_2) \subseteq \mathcal{A}(\sigma_1)$, we have $u \in \mathcal{A}(\sigma_1)$.

Since $t \notin \mathcal{A}(\sigma_1)$, we have $t \notin \mathcal{A}(\sigma_2)$.

So u occurs at least twice in $\sigma_1\,\sigma_2$, and t occurs zero times.

It follows $M_2(s) \geq 2$, and so $M_1(s) \geq 1$.

Since $M_1(s) = M_2(s) \geq 1$ for every $s \in {}^\bullet t \setminus u^\bullet$, transition t is already enabled at M_1, and we are done. □

Lemma 5.2.12 *Let (N, M_0) be a 1-bounded T-system with $N = (S, T, F)$, and let $M_0 \xrightarrow{\sigma} M$. There exist sequences σ_1, σ_2 such that*

(a) $M_0 \xrightarrow{\sigma_1 \sigma_2} M$*;*

(b) no transition occurs more than once in σ_1;

(c) $\mathcal{A}(\sigma_2) \subseteq \mathcal{A}(\sigma_1)$*; and*

(d) if σ is nonempty, then $\mathcal{A}(\sigma_2) \subset \mathcal{A}(\sigma_1)$.

Proof. We first prove that (a)-(c) hold.

By induction on $|\sigma|$.

If $|\sigma| = 0$, then take $\sigma_1, \sigma_2 = \epsilon$.

Assume $\sigma = \tau\, t$ for some $t \in T$ and $M_0 \xrightarrow{\tau} M' \xrightarrow{t} M$.

By induction hypothesis there are τ_1, τ_2 such that

- $M_0 \xrightarrow{\tau_1 \tau_2} M'$;
- no transition occurs more than once in τ_1; and
- $\mathcal{A}(\tau_2) \subseteq \mathcal{A}(\tau_1)$.

It $t \in \mathcal{A}(\tau_1)$, then take $\sigma_1 := \tau_1$ and $\sigma_2 := \tau_2\, t$.

If $t \notin \mathcal{A}(\tau_1)$, then by Lemma **??** we have $M_0 \xrightarrow{\tau_1\, t\, \tau_2} M$.

Take $\sigma_1 := \tau_1\, t$ and $\sigma_2 := \tau_2$.

We now prove (d).

Assume $M_0 \xrightarrow{\sigma} M$ such that σ is nonempty and of minimal length.

Let σ_1, σ_2 be sequences satisfying (a)-(c).

In particular we have $\mathcal{A}(\sigma_2) \subseteq \mathcal{A}(\sigma_1)$.

We show $\mathcal{A}(\sigma_2) \subset \mathcal{A}(\sigma_1)$.

Claim 1: $\mathcal{A}(\sigma_1) \neq \emptyset$.

Follows from $\sigma \neq \epsilon$ and $\mathcal{A}(\sigma_2) \subseteq \mathcal{A}(\sigma_1)$.

Claim 2: $\mathcal{A}(\sigma_1) \neq T$.

If $\mathcal{A}(\sigma_1) = T$ then σ_1 contains every transition exactly once, and so $M_0 \xrightarrow{\tau_1} M_0$.

It follows $M_0 \xrightarrow{\tau_2} M$, contradicting that σ has minimal length.

We prove $\mathcal{A}(\sigma_2) \subset \mathcal{A}(\sigma_1)$ with the help of the claims.

Since N is 1-bounded, it is strongly connected (Boundedness Theorem).

By Claim 1 and Claim 2 there is a place s with input and output transitions t and u, respectively, such that $t \in \mathcal{A}(\sigma_1)$ and $u \notin \mathcal{A}(\sigma_1)$.

We show that $t \notin \mathcal{A}(\sigma_2)$, which proves $\mathcal{A}(\sigma_2) \subset \mathcal{A}(\sigma_1)$.

Assume $t \in \mathcal{A}(\sigma_2)$. Then t occurs at least twice in $\sigma_1\sigma_2$.

We have $u \notin \mathcal{A}(\sigma_1)$ and $\mathcal{A}(\sigma_2) \subseteq \mathcal{A}(\sigma_1)$, and so u occurs zero times in $\sigma_1\sigma_2$.

This contradicts 1-boundedness.

□

Theorem 5.2.13 (Shortest Sequence Theorem)
Let (N, M_0) be a 1-bounded T-system and let M be a reachable marking. Then there is an occurrence sequence $M_0 \xrightarrow{\sigma} M$ such that

$$|\sigma| \leq n(n-1)/2$$

where n is the number of transitions of N.

Proof. By repeated application of Lemma **??**, there exists an occurrence sequence

$$M_0 \xrightarrow{\sigma_1\, \sigma_2 \cdots \sigma_n} M$$

such that

- $\sigma_i \neq \epsilon$ for every $1 \leq i \leq n$;
- no transition occurs more than once in any of $\sigma_1, \ldots, \sigma_n$; and
- $\mathcal{A}(\sigma_1) \supset \mathcal{A}(\sigma_2) \supset \cdots \supset \mathcal{A}(\sigma_n)$.

Further, we can assume $T \supset \mathcal{A}(\sigma_1)$. Indeed, if $T = \mathcal{A}(\sigma_1)$ then $M_0 \xrightarrow{\sigma_1} M_0$, and we can choose $\sigma_2 \cdots \sigma_n$ instead of $\sigma_1 \cdots \sigma_n$.

It follows that $|\sigma_i| \leq n - i$ for every $1 \leq i \leq n$. We get

$$|\sigma| \leq \sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}\ .$$

□

Exercise: Show that the bound of the theorem is tight.

Exercise: Show that for b-bounded T-systems the corresponding bound is $b \cdot n \cdot (n-1)/2$.

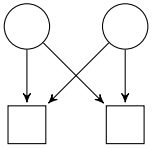
5.3 Free-Choice Systems

Definition 5.3.1 (Free-Choice nets, Free-Choice systems)
A net $N = (S, T, F)$ is free-choice if for every $s \in S$ and $t \in T$:

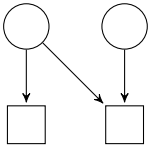
$$\text{if } (s,t) \in F \text{ then } {}^\bullet t \times s^\bullet \subseteq F.$$

A Petri net (N, M_0) is free-choice if N is a free-choice net..

For example:



free-choice net



non-free-choice net

The following less concise but equivalent definitions may be easier to understand.

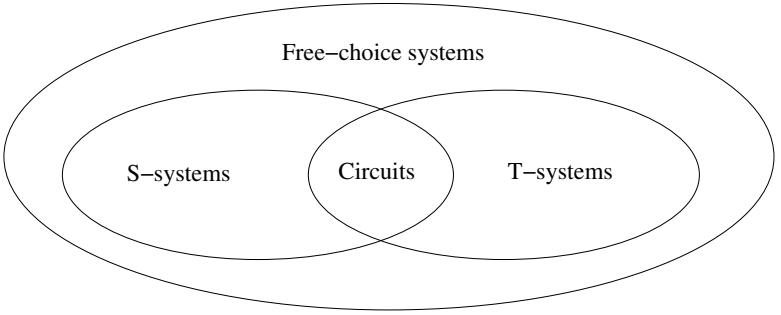
Proposition 5.3.2 (Alternative definitions of free-choice nets)
A net is free-choice if the presets of any two transitions are disjoint or identical: For every two transitions t_1, t_2: ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$ or ${}^\bullet t_1 = {}^\bullet t_2$.

A net is free-choice if the postsets of any two places are disjoint or identical: For every two places s_1, s_2: $s_1^\bullet \cap s_2^\bullet = \emptyset$ or $s_1^\bullet = s_2^\bullet$.

Proof. Exercise. □

Training Exercise in AutomataTutor:
☆ Free-choice system: Make a given Petri net free-choice by adding arcs.

Observe that S- and T-systems are special cases of free-choice systems:



5.3.1 Liveness

We showed in the last chapter that a Petri net in which every siphon contains an initially marked trap is deadlock-free, but the converse does not hold.

For free-choice systems we obtain Commoner's Liveness Theorem: A free-choice Petri net is live iff every proper siphon contains an initially marked trap.

We prove both directions separately.

If-direction of Commoner's Liveness Theorem

Definition 5.3.3 (Live and dead transitions) *Let M be a marking of N.*

A transition t is dead at M if it is not enabled at any marking of $[M\rangle$. We let D_M denote the set of transitions dead at M.

A transition t is live at M if $t \notin D_{M'}$ for every marking $M' \in [M\rangle$. We let L_M denote the set of transitions live at M.

Notice that a transition may be neither live nor dead at a marking. (In this proof, a better name for "live" would be "inmortal"). We have:

- Live transitions stay live: If $t \in L_M$ and $M' \in [M\rangle$, then $t \in L_{M'}$.
- Dead transitions stay dead: f $t \in D_M$ and $M' \in [M\rangle$, then $t \in D_{M'}$.
- Transitions that are neither live nor dead can die:
  If $t \notin L_M \cup D_M$ then $t \in D_{M'}$ for some M' reachable from M.

Theorem 5.3.4 (First part of Commoner's Liveness Theorem)
Let (N, M_0) be a free-choice system. If every proper siphon of N contains a trap marked at M_0, then (N, M_0) is live.

Proof. We prove that if (N, M_0) is not live, then some proper siphon of N does not contain any trap marked at M_0.

Let T be the set of transitions of N.

Since (N, M_0) is not live, then, by Definition **??**, there is a marking M reachable from M_0 such that $T = D_M \cup L_M$, that is, every transition is either live or dead at M, and $D_M \neq \emptyset$.

Claim: For every transition $t \in D_M$ there exists $s_t \in {}^\bullet t$ such that $M(s_t) = 0$ and every $t' \in {}^\bullet s_t$ is dead at M.

Let S_t be the set of input places of t not marked at M.

Since $t \in D_M$, the set S_t is nonempty.

Since N is free-choice, for every $s \in S_t$ every transition of $s_t^\bullet$ is dead at M (otherwise we could fire t).

So along any occurrence sequence starting at M the number of tokens in each place of S_t does not decrease.

It follows: if every place $s_t \in {}^\bullet t$ such that $M(s_t) = 0$ has a live input transition at M, then we can reach a marking that marks all places of S_t.

But such a marking enables t, contradicting that t is dead at M.

So at least one place $s_t \in {}^\bullet t$ has only dead input transitions at M, which proves the claim.

Let now $R = \{s_t \mid t \in D_M\}$. By the claim, and since $D_M \neq \emptyset$, the set R is a siphon unmarked at M. If R would contain a trap marked at M_0 then, since marked traps remain marked, R would be marked at M. So R does not contain any trap marked at M_0. □

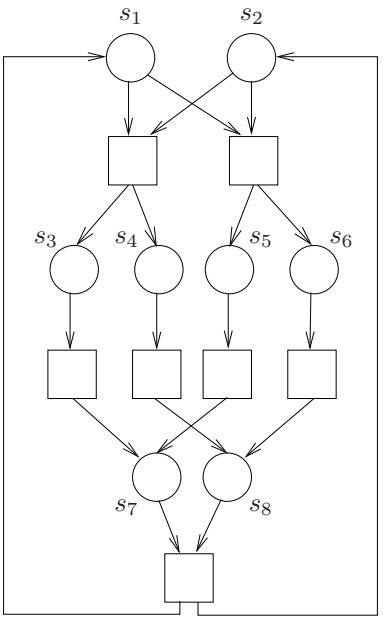


Figure 5.1: A free-choice net

A siphon is minimal if it does not properly contain any proper siphon.

Theorem **??** still holds if we replace "siphon" by "minimal siphon".

The net of Figure **??** has four minimal siphons:

$$R_1 = \{s_1, s_3, s_5, s_7\} \qquad R_2 = \{s_2, s_4, s_6, s_8\}$$
$$R_3 = \{s_2, s_3, s_5, s_7\} \qquad R_4 = \{s_1, s_4, s_6, s_8\}$$

R_1, R_2, R_3 and R_4 are also traps, and so, in particular, they contain traps.

By Theorem **??**, every marking that marks R_1, R_2, R_3 and R_4 is live.

Only-if-direction of Commoner's Theorem

We show that if some proper siphon R of a free-choice system (N, M_0) does not contain an initially marked trap, then (N, M_0) is not live.

If such a siphon exists, then the maximal trap $Q \subseteq R$ is unmarked at M_0, and so M_0 only can mark places of $D := R \setminus Q$.

Intuitively, we construct a firing sequence that "empties" the places of D without marking the places of Q. In this way we reach a marking at which the siphon R is empty, which proves that (N, M_0) is not live.

We need the notion of a cluster.

Definition 5.3.5 (Cluster) Let $N = (S, T, F)$ be a net. A cluster is an equivalence class of the equivalence relation

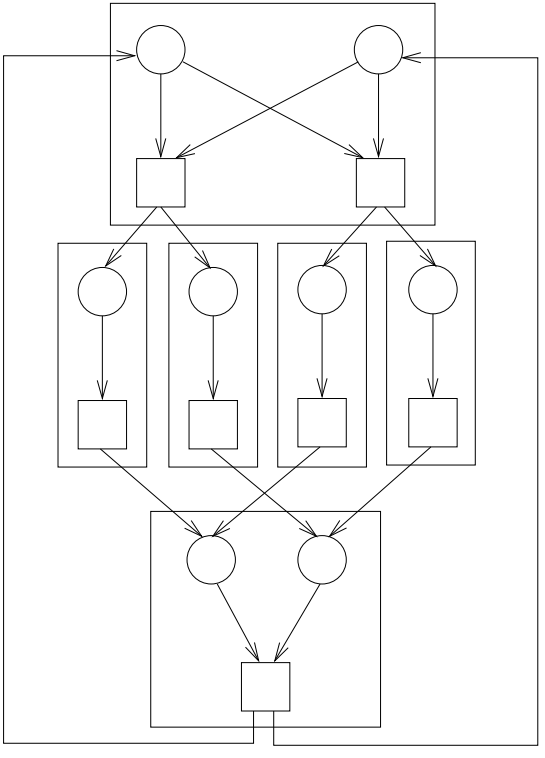
$$((F \cap (S \times T)) \cup (F \cap (S \times T))^{-1})^* \ .$$

We let $[x]$ denote the cluster of the node $x \in S \cup T$.

Informally: two nodes x, y are equivalent if one can travel from x to y by moving from a place to one of its output transitions, and from a transition to one of its input places.

Observe: every node of a net belongs to exactly one cluster, that is, the set of clusters is a partition of $S \cup T$.

The figure in the next page shows the clusters of the net of Figure **??**.



The following proposition is easy to prove:

Proposition 5.3.6 *Let (N, M_0) be a free-choice system with $N = (S, T, F)$, and let c be a cluster of N.*

(1) $(s, t) \in F$ for every $s \in c \cap S$ and $t \in c \cap T$.

(2) A marking enables some transition of c iff it enables every transition of c.

By (2) we can say that M enables a cluster.

The firing sequence σ that empties the siphon R is constructed as follows.

- We define an allocation that assigns to each cluster c of N containing places of D a transition of $c \cap T$.

  Intuitively, the allocation is a recipe indicating which transition to fire: Whenever the transitions of the cluster are enabled, we fire the allocated transition, and never any of the others.

- The sequence σ is constructed by repeatedly enabling the clusters of D, which is possible by liveness, and then firing the allocated transition.

We formally define allocations.

Definition 5.3.7 (Allocation)
Let $N = (S, T, F)$ be a net and let C be a set of clusters of N. An allocation of C is a mapping $\alpha\colon C \to T$ such that $\alpha(c) \in c$ for every $c \in C$.

Let $C = \{[t] \mid t \in D^\bullet\}$. We construct an allocation $\alpha\colon C \to T$ satisfying the following properties.

(a) α is circuit-free, that is, there is no cycle containing only places of D and allocated transitions.

If there were such a cycle, then by firing only allocated transitions we might never be able to empty D, because tokens in the cycle would never "leave" it.

(b) α does not allocate any transition of ${}^\bullet Q$.

Otherwise firing this transition would mark the trap Q, which would make it impossible to empty the siphon.

(c) While there are tokens in D it is always possible to fire any of the allocated transitions again, without firing any of the non-allocated transitions.

The recipe to construct an allocation satisfying (a) and (b) is given in the proof of the following lemma. This part does not require the free-choice property.

Lemma 5.3.8 (Circuit-free Allocation Lemma)
Let N be a net, let R be a set of places of N, let Q be the maximal trap included in R, let $D = R \setminus Q$, and let $C = \{[t] \mid t \in D^\bullet\}$.

There exists a circuit-free allocation $\alpha\colon C \to T$ such that $\alpha(C) \cap {}^\bullet Q = \emptyset$.

Proof. By induction on $|R|$.

Basis: $|R| = 0$. Then $C = \emptyset$ and we take the empty allocation.

Step: $|R| > 0$.

If R is a trap then $D = \emptyset$, and again $C = \emptyset$.

If R is not a trap then there exists $t \in R^\bullet \setminus {}^\bullet R$ (intuitively, t is a way-out through which tokens can leave R).

Let $R' = R \setminus {}^\bullet t$, let Q' be the maximal trap of R', let $D' = R' \setminus Q'$, and let $C' = \{[t] \mid t \in (D')^\bullet\}$.

By induction hypothesis there exists an allocation $\alpha'\colon C' \to T$, circuit-free for D', such that $\alpha'(C') \cap {}^\bullet Q' = \emptyset$.

Define $\alpha\colon C \to T$ as follows:

$$\alpha(c) = \begin{cases} t & \text{if } t \in c \\ \alpha'(c) & \text{otherwise} \end{cases}$$

We show that (a) α is circuit-free and (b) $\alpha(C) \cap {}^\bullet Q = \emptyset$.

We first prove the following facts, which we leave as an exercise:

(i) Q is the maximal trap included in R', i.e., $Q' = Q$.

(ii) $D \subseteq D' \cup {}^\bullet t$. (Use (i).)

(iii) $C \subseteq C' \cup \{[t]\}$. (Use (ii) and the definition of C.)

(iv) $\alpha(C) \subseteq \alpha(C') \cup \{t\}$. (Use (iii) and the definition of α.)

(a) α is circuit-free.

Assume $D \cup \alpha(C)$ contains a circuit γ.

By (ii) and (iv) we have

$$D \cup \alpha(C) \subseteq D' \cup \alpha'(C') \cup \{t\} \cup {}^\bullet t \; .$$

By induction hypothesis $D \cup \alpha'(C')$ is circuit-free. So γ contains t.

Since all places of γ belong to R and $t \notin {}^\bullet R$, we have that γ contains no place of $t^\bullet$, contradicting that γ is a circuit.

(b) $\alpha(C) \cap {}^\bullet Q = \emptyset$.

We have

$$\begin{array}{lll} & \alpha(C) \cap {}^\bullet Q' & \\ \subseteq & (\alpha(C') \cup \{t\}) \cap {}^\bullet Q' & \text{(def. of } \alpha \text{ and } Q = Q') \\ = & \{t\} \cap {}^\bullet Q' & \text{(induction hypothesis)} \\ = & \emptyset & (t \notin {}^\bullet R \text{ and } {}^\bullet Q \subseteq {}^\bullet R) \end{array}$$

□

We now prove that we can find an infinite occurrence sequence that "respects a given allocation". This part crucially requires the free-choice property.

Lemma 5.3.9 (Allocation Lemma)
Let (N, M_0) be a live free-choice system, let C be a set of clusters of N, and let $\alpha\colon C \to T$ be an allocation of C.

There is an infinite occurrence sequence $M_0 \xrightarrow{\sigma}$ such that σ contains

- *infinitely many occurrences of allocated transitions, and*
- *no occurrences of non-allocated transitions of C, i.e., of transitions of $\bigcup_{c\in C} c \setminus \{\alpha(c)\}$.*

Proof. We iteratively define occurrence sequences $\sigma_0, \sigma_1, \sigma_2, \ldots$, and define $\sigma := \sigma_0\, \sigma_1\, \sigma_2 \ldots$.

Given a marking M_i, let τ_i be a minimal occurrence sequence that enables some cluster $c \in C$. The sequence exists by liveness.

By the free-choice property, the sequence $\sigma_i = \tau_i \alpha(c)$ is also a firing sequence.

Let M_{i+1} be the marking given by $M_i \xrightarrow{\sigma_i} M_{i+1}$. □

Theorem 5.3.10 (Second half of Commoner's Liveness Theorem)
Let (N, M_0) be a free-choice system. If (N, M_0) is live, then every proper siphon of N contains a trap marked at M_0.

Proof. Let R be a proper siphon of N, and let Q be the maximal trap included in R. We prove $M_0(Q) > 0$.

Since (N, M_0) is live, we have $M_0(R) > 0$ by Proposition **??**.

Let $D = R \setminus Q$.

If $D^\bullet = \emptyset$ then D is a trap. So $D \subseteq Q$, which implies $D = \emptyset$, and we are done.

If $D^\bullet \neq \emptyset$ then let $C = \{[t] \mid t \in D^\bullet\}$.

By Lemma **??** there is an allocation with domain C and circuit-free for D satisfying $\alpha(C) \cap {}^\bullet Q = \emptyset$.

Let $M_0 \xrightarrow{\sigma}$ be the occurrence sequence of Lemma **??**. It is easy to see that

- Q cannot become marked during the occurrence of σ.

  Because transitions of ${}^\bullet Q$ are not allocated, and so do not occur in σ.

- Q is marked at some point during the occurrence of σ.

  Since α is circuit-free, there is an allocated transition t that occurs infinitely often in σ, and whose input places are not output places of any allocated transition.

  So the input places of t must get tokens from transitions that do not belong to the clusters of C.

  But these transitions are necessarily output transitions of Q.

□

Non-liveness of free-choice systems is NP-complete

The non-liveness problem for free-choice systems is NP-complete, and so we cannot expect to find a polynomial algorithm to check the condition of Commoner's Theorem:

Theorem 5.3.11 (Non-liveness is NP-complete)
The following problem is NP-complete:

Given: *A free-choice system* (N, M_0)
Decide: *Is* (N, M_0) *not live?*

Proof. Membership in NP follows from Commoner's theorem.

The following nondeterministic algorithm decides non-liveness:

- Guess a siphon R of N.
- Compute (in polynomial time) the maximal trap contained in R.
- Check that the trap is unmarked at M_0.

The proof of NP-hardness is by reduction from SAT, the satisfiability problem for boolean formulas.

Given a formula φ, we construct in polynomial time a free-choice Petri net $(N_\varphi, M_{0\varphi})$ such that φ is satisfiable iff $(N_\varphi, M_{0\varphi})$ is not live.

The construction is described by example in the next page.

□

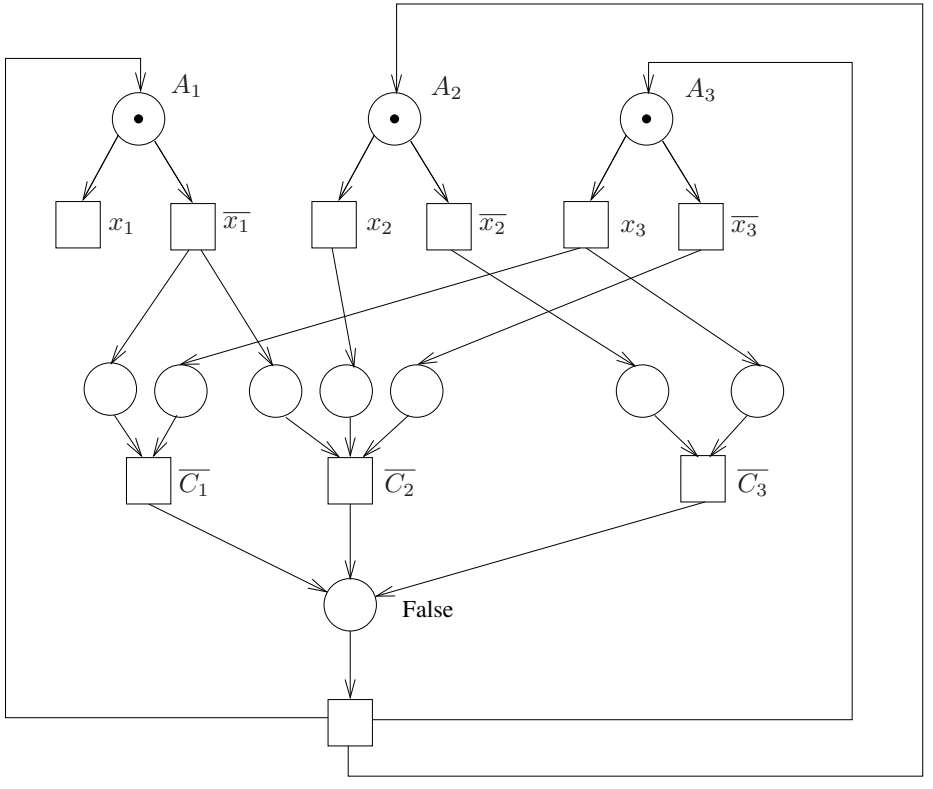


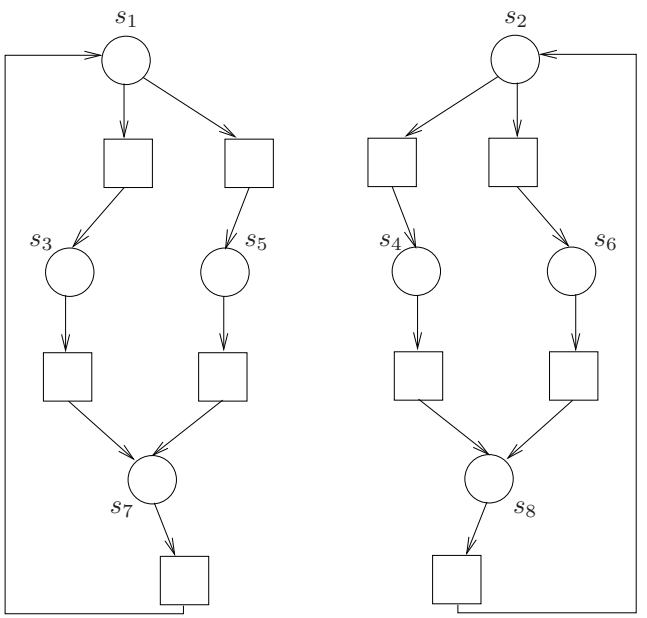
Figure 5.2: Free-choice system for $(x_1 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3})$

5.3.2 Boundedness

Definition 5.3.12 (S-component)
Let $N = (S, T, F)$ be a net. A subnet $N' = (S', T', F')$ of N is a S-component of N if

1. *N' is a strongly connected S-net, and*
2. *$T' = {}^\bullet S' \cup S'^\bullet$.*

Two S-components of the net of Figure **??**:



S-components are for free-choice systems what circuits are for T-systems: firing a transition does not change the number of tokens of an S-component.

Proposition 5.3.13
Let (N, M_0) be a Petri net and let $N' = (S', T', F')$ be an S-component of N. Then $M_0(S') = M(S')$ for every marking M reachable from M_0.

Proof. Firing a transition either takes no tokens from a place of the component and adds none, or it takes exactly one token and adds exactly one token. □

Theorem 5.3.14 (Hack's Boundedness Theorem)
Let (N, M_0) be a live free-choice system. (N, M_0) is bounded iff every place of N belongs to a S-component.

Proof. ($\Leftarrow$) Exercise

($\Rightarrow$) (Sketch). We first show that every minimal siphon N is the set of places of a S-component. Then we show that every place is contained in some minimal siphon. □

Proposition 5.3.15 (Place bounds)
Let (N, M_0) be a live and bounded free-choice system and let s be a place of N. We have

$$max\{M(s) \mid M_0 \xrightarrow{*} M\} =$$
$$min\{M_0(S') \mid S' \textit{ is the set of places of a S-component of } Nandcontainss\}$$

Proof. Analogous to the Boundedness Theorem for T-systems. □

The Rank Theorem

Theorem **??** shows that there is no polynomial algorithm for Liveness (unless $P = NP$). But what is the complexity of deciding if a free-choice system is simultaneously live and bounded.

We can first use the decision procedure for liveness, and then check the condition of the Boundedness Theorem. But there are more efficient algorithms.

(Compare with this: in order to decide if a number is divisible by 100.000, we can first check if it is divisible by 3125, and, if so, if it is divisible by 32. However, there is a faster procedure: check if the last five digits are zeros.)

The fastest known algorithm runs in $O(n \cdot m)$ time for a net with n places and m transitions. A not so efficient but simpler algorithm follows from the Rank Theorem:

Theorem 5.3.16 (Rank Theorem)
A free-choice system (N, M_0) is live and bounded iff

1. *N has a positive S-invariant.*
2. *N has a positive T-invariant.*
3. *The rank of the incidence matrix $\mathbf{N}$ is equal to $c - 1$, where c is the number of clusters of N.*
4. *Every proper siphon of N is marked under M_0.*

Proof. Omitted. □

(1) and (2) can be checked using linear programming, (3) using well-known algorithms of linear algebra, (4) by computing the largest siphon unmarked at M_0.

5.3.3 Reachability

Reachability is NP-complete for live and bounded free-choice Petri nets.

We prove separately that it is NP-hard and in NP.

Theorem 5.3.17 *Reachability is NP-hard for live and bounded free-choice systems.*

Proof. We proceed in two steps

(1) We reduce SAT to the following problem:

Given: A live and bounded free-choice system (N, M_0), two disjoint sets $T_{=1}, T_{\geq 1}$ of transitions of N, and a marking M.

Decide: Is M reachable from M_0 by means of a firing sequence that fires each transition of $T_{=1}$ exactly once, and each transition of $T_{\geq 1}$ at least once?

(2) We reduce this problem to the reachability problem for live and bounded free-choice systems.

(1) Figure **??** shows the net N, the markings M_0 and M, and the sets $T_{=1}, T_{\geq 1}$ for the formula

$$x_1 \ \wedge \ (\overline{x}_1 \vee x_2) \ \wedge (\overline{x}_1 \vee \overline{x}_2) \ .$$

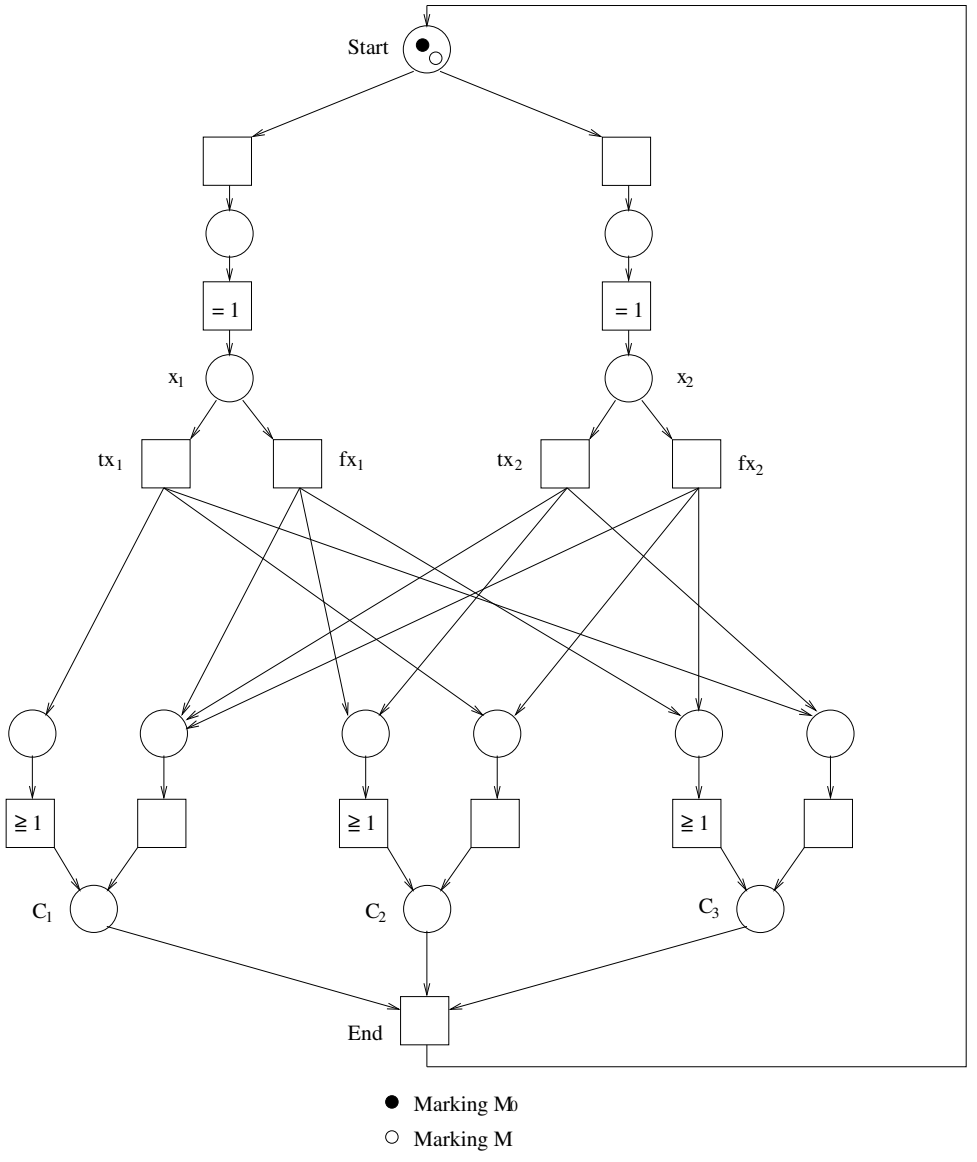


Figure 5.3: Result of the reduction for the formula $x_1 \wedge (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2)$

The formula has three clauses C_1, C_2, C_3.

Black tokens correspond to M_0 and white tokens to M.

Intuitively, the net chooses a variable x_i, and assigns it a value by firing tx_i or fx_i. This sends tokens to the three modules at the bottom of the figure, one for each clause.

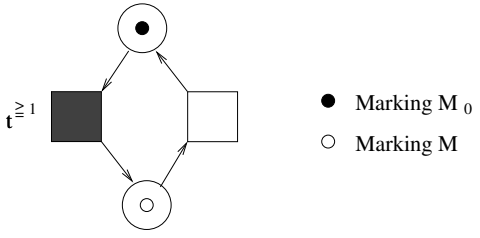
More precisely, for each clause the transition sends exactly one token to one of the two transitions of the module: if the value makes the clause true, then the token goes to the input place of the transition that belongs to $T_{\geq 1}$; otherwise the token goes to the input place of the other transition.

The formula is satisfiable iff the Petri net has a firing sequence that fires

- each transition of $T_{=1}$ exactly once,
  (this corresponds to choosing a truth assignment)
- each transition of $T_{\geq 1}$ at least once
  (so that at least one of the literals of each clause is true under the assignment).

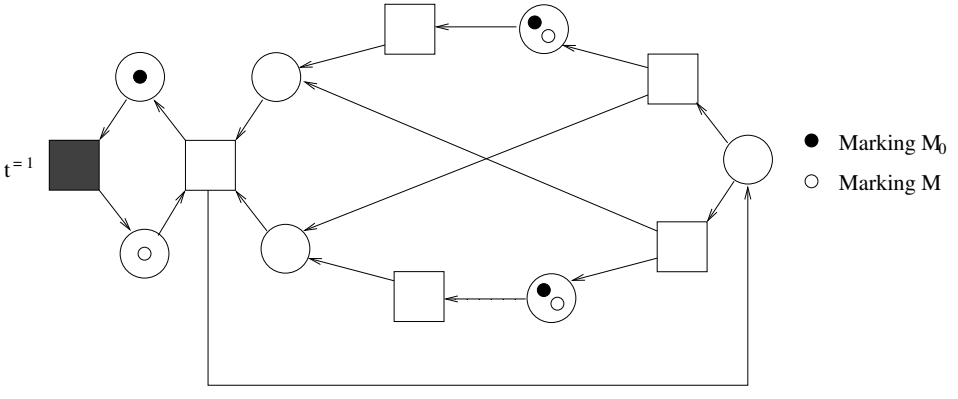
(2) Now we reduce the problem above to the reachability problem for live and bounded free-choice systems.

Given a free-choice net with sets $T_{=1}, T_{\geq 1} \subseteq T$, we "merge" each transition of $T_{\geq 1}$ with the transition $t^{\geq 1}$ of a separate copy of this module



The module ensures that in order to reach the marking M the transition $t^{\geq 1}$ has to be fired at least once.

Similarly, we merge each transition of $T_{=1}$ with the transition $t^{=1}$ of a separate copy of this module



The second module ensures that the transition $t^{=1}$ has to be fired exactly once. □

Like for Commoner's Theorem, membership in NP is harder to prove. It follows from this theorem, due to Yamasaki *et al.*

Definition 5.3.18 (Subnet generated by a set of transitions)
Let $N = (S, T, F)$ be a net, and let $U \subseteq T$. The subnet of N generated by U, denoted N_U, is the unique subnet of N having U as set of transitions and ${}^\bullet U \cup U^\bullet$ as set of places.

Theorem 5.3.19 (Reachability Theorem)
Let (N, M_0) be a live and bounded free-choice system. M is reachable from M_0 iff there exists a vector $X \in \mathbb{N}^{|T|}$ such that

- $M = M_0 + \mathbf{N} \cdot X$, *and*
- *The system (N_U, M_U) has no unmarked traps, where $U = \{t \in T \mid X(t) > 0\}$ and M_U is the projection of M onto the places of N_U.*

Proof. ($\Leftarrow$): Exercise.

($\Rightarrow$): Omitted. □

Membership in NP can then be proved as follows. To check that M is reachable in (N, M_0), where $N = (S, T, F)$:

- Guess a set $U \subseteq T$.
- Construct N_U, compute in polynomial time the maximal trap of N_U unmarked at M, and check that it is the empty trap.
- Guess in polynomial time a vector $X \in \mathbb{N}^{|T|}$ such that $X(t) \geq 1$ for every $t \in U$, and check that $M = M_0 + \mathbf{N} \cdot X$ holds.

Proving that the vector can be guessed in polynomial time follows from the fact that Integer Linear Programming is also in NP.

5.3.4 Other useful results

A Petri net (N, M_0) is cyclic if, loosely speaking, it is always possible to return to the initial marking. In other words, for every marking M reachable from M_0, the marking M_0 is reachable from M.

Theorem 5.3.20 (Cyclicity Theorem)
A live and bounded free-choice system (N, M_0) is cyclic iff M_0 marks every proper trap of N.

Proof. Omitted. □

Reachabiilty is polynomial for live, bounded, and cyclic free-choice systems.

Theorem 5.3.21 (Reachability Theorem for Cyclic Free-Choice Nets)
Let (N, M_0) be a live, bounded, and cyclic free-choice system. A marking M of N is reachable from M_0 iff $M_0 \sim M$.

Proof. Omitted. □

Corollary 5.3.22 *The problem*

> *Given: a live, bounded, and cyclic free-choice system (N, M_0)*
> *and a marking M*
> *Decide: Is M reachable?*

can be solved in polynomial time.

There is also a Shortest Sequence Theorem for live and bounded free-choice nets.

Theorem 5.3.23 (Shortest Sequence Theorem)
Let (N, M_0) be a live and b-bounded free-choice system and let M be a reachable marking. Then there is an occurrence sequence $M_0 \xrightarrow{\sigma} M$ such that

$$|\sigma| \leq b\, n(n+1)(n+2)/6$$

where n is the number of transitions of N.

This gives a simpler proof that the reachability problem for live and bounded free-choice nets is in NP: just guess in polynomial time an occurrence sequence leading to M.