Petri Nets
Lecture Notes

Prof. Javier Esparza

April 23, 2019

Contents

Sources

The main sources are:

> J. Desel. **Struktur und Analyse von Free-Choice-Petrinetzen**. Deutscher Universitäts Verlag, 1992.
>
> J. Desel und J. Esparza. **Free-choice Petri nets**. Cambridge Tracts in Theoretical Computer Science 40, Cambridge University Press, 1995.

The Petri net model of Peterson's algorithm is taken from

> E. Best. **Semantics of Sequential and Parallel Programs**. Prentice-Hall, 1996.

The action-reaction protocol is taken from

> R. Walter. **Petrinetzmodelle verteilter Algorithmen – Intuition und Beweistechnik**. Dieter Bertz Verlag, 1996.

The train examples of Chapter 2 belong to the Petri net folklore. They were first introduced by H. Genrich.

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, 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 $\sigma\colon \{1, \ldots, n\} \to A$, denoted by the string $a_1 a_2 \ldots a_n$, 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 $\sigma = a_1 a_2 a_3 \ldots$ with $a_i = \sigma(i)$.

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* $\sigma\sigma\sigma \ldots$.

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.

Given vectors $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$, 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 $x_1 \geq y_1 \wedge \ldots \wedge x_n \geq y_n$,a nd $X > Y$ 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

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

and 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

We recall some basic notions of complexity theory. Formal definitions can be found in standard textbooks.

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 $f\colon \mathbb{N} \to \mathbb{N}$ if for every input of length n (measured in bits) every computation takes at most $f(n)$ time. Given a set $\mathcal{C}$ 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 $f(n)$.

Similarly, a program needs $f(n)$-memory or $f(n)$-space for a function $f\colon \mathbb{N} \to \mathbb{N}$ 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.

Informally, 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 (P, U) if for every input $u \in U$:

- 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.

- **P** is the class of problems that can be solved by polynomial-time dterministic programs.
- **NP** is the class of problems that can be solved by polynomial-time nondterministic programs.

- **PSPACE** is the class of problems that can be solved by polynomial-space determinstic programs.
- **NPSPACE** is the class of problems that can be solved by polynomial-space nondterministic 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 fact) **P** $\subset$ **EXPTIME**. We also know:

Theorem 1.1.1 *[Savitch's theorem]*
NPSPACE $=$ **PSPACE**.

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. It is *complete* for the 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) $F \subseteq (S \times T) \cup (T \times S)$.

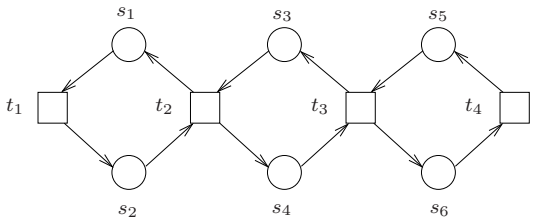


Figure 1.1: Graphical representation of the net N

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

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

Figure 1.1 shows the graphical representation of N. For example we have ${}^\bullet t_2 = \{s_2, s_3\}$ and ${}^\bullet S = S^\bullet = T$.

Remark: Nets with empty S, T or F are allowed!

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 $F' \subseteq F \cap ((S' \times T') \cup (T' \times S'))$!).

Figure 1.2 shows some subnets and non-subnets of the net of Figure 1.1.

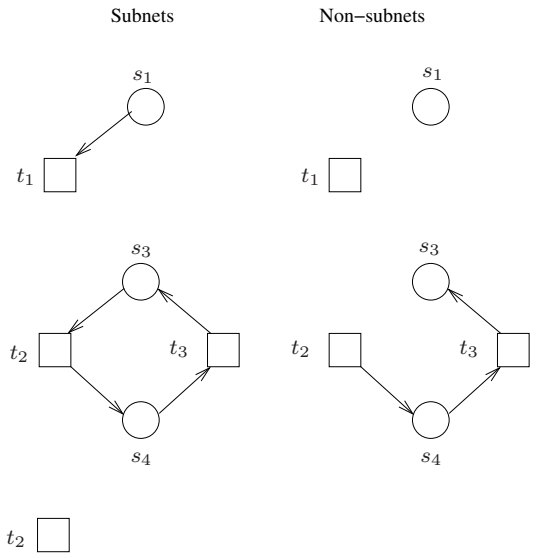


Figure 1.2: Subnets and non-subnets of the net of Figure 1.1

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

Remarks:

- Every net with 0 or 1 node is strongly connected!
- If N is strongly connected then it is also connected.

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.

Proof. Exercise. □

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.

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 *Let M be the marking of the net N in Figure 1.1 given by $M(s_1) = M(s_4) = M(s_5) = 1$ and $M(s_2) = M(s_3) = M(s_6) = 0$. We denote this marking by the vector $(1, 0, 0, 1, 1, 0)$.*

The marking enables transitions 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

$$\begin{aligned}(1,0,0,1,1,0) &\xrightarrow{t_1} (0,1,0,1,1,0)\\ (1,0,0,1,1,0) &\xrightarrow{t_3} (1,0,1,0,0,1)\end{aligned}$$

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*. $[M\rangle$ denotes the set of markings that are 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 *Let N be the net of Figure 1.1 and let $M = (1,0,0,1,1,0)$ be a marking of N. We have*

$$\begin{array}{l}(1,0,0,1,1,0) \xrightarrow{t_1} (0,1,0,1,1,0) \xrightarrow{t_3} (0,1,1,0,0,1)\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \downarrow t_2\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (1,0,0,1,0,1) \xrightarrow{t_4} (1,0,0,1,1,0)\end{array}$$

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

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

Proof. Easy exercise. □

The following simple lemma plays a fundamental role in many results about Petri nets.

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 1.3.6. By Proposition 1.3.6, 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. □

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'.

```
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)
```

Figure 1.3: Algorithm for computing the reachability graph

The algorithm of Figure 1.3 computes the reachability graph. It 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. Breadth first search will find the shortest transition path from the initial marking to a given (erroneous) marking. Some applications require depth first search.

Chapter 2

Modelling with Petri nets

2.1 A buffer of capacity n

We model a buffer with capacity for n items. Figure 2.1 shows the Petri net for $n = 3$. The model consists of n

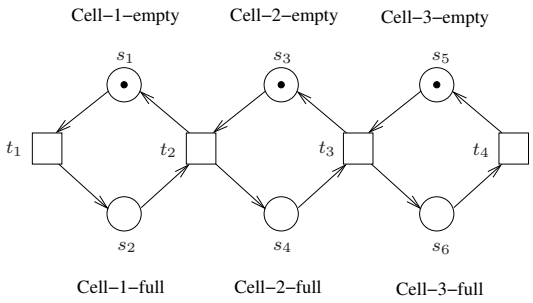


Figure 2.1: A 3-buffer

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. Observe that the buffer is concurrent: 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.

Figure 2.2 shows the reachability graph of the buffer with capacity 3. By inspection of the reachability graph

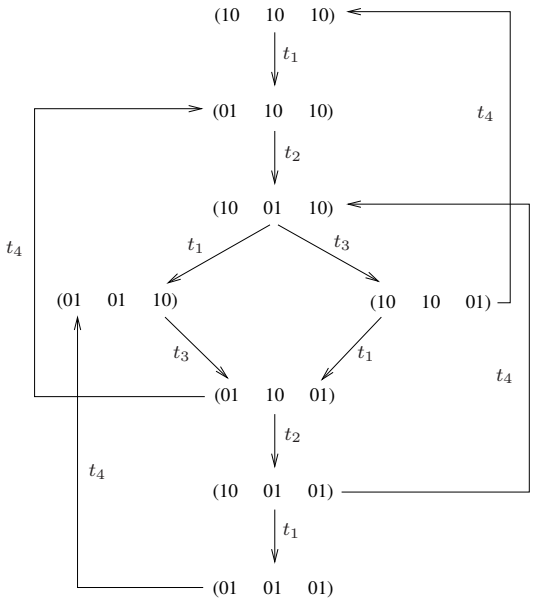


Figure 2.2: Reachability graph of the 3-buffer

we can see that the following properties hold:

- Consistency: no cell is simultaneously empty and full (that is, no marking puts tokens on s_i and s_{i+1} for $i = 1, 2, 3$).
- 1-boundedness: every reachable marking puts at most one token in a given place.
- 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).
- 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.
- The initial marking is reachable from any reachable marking (that is, it is always possible to empty the buffer).
- 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. Our task

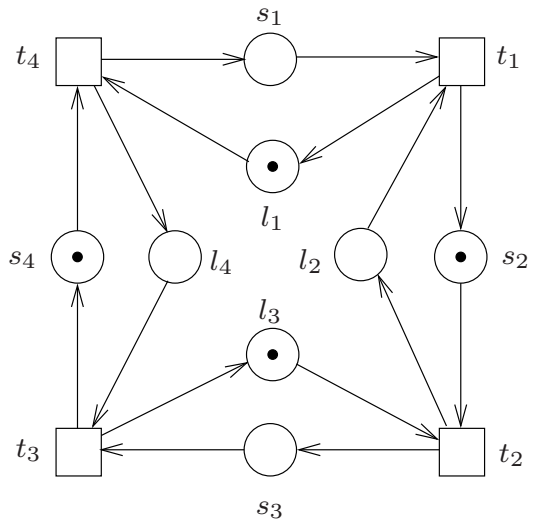


Figure 2.3: Train tracks (first version)

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

Figure 2.3 shows a solution of the problem modeled as a Petri net. the four tracks are modeled by places $s_1, \ldots, s_4$. A token on s_i means that there is train in the i-th track.

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 shown in Figure 2.4. Since every reachable marking puts at most one token on a place, we denote a marking by the set of places marked by it. For instance, we denote by $\{l_1, s_2, l_3, s_4\}$ the marking that puts a token on l_1, s_2, l_3 and s_4.

Consider now a slightly different system. 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.

The Petri net of Figure 2.5 is a solution of the problem: The reader can construct the reachability graph and show that the desired property holds. However, the graph is pretty large!

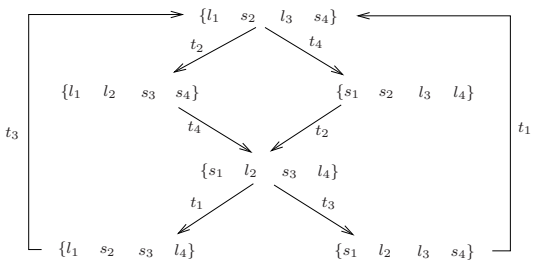


Figure 2.4: Reachability graph of the Petri net of Figure 2.3

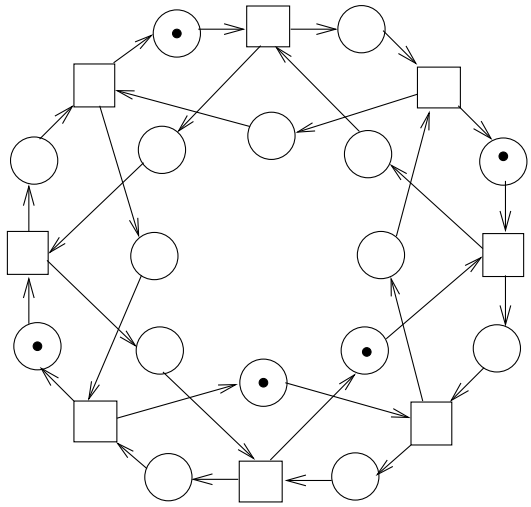


Figure 2.5: Train tracks (second version)

2.3 Dining philosophers

Four philosophers sit around a round table. There are forks on the table, one between each pair of philosophers. The philosophers want to eat spaghetti from a large bowl in the center of the table (see the top of Figure 2.6). Unfortunately the spaghetti is of a particularly slippery type, and a philosopher needs both forks in order to eat it. The philosophers have agreed on the following protocol to obtain the forks: Initially philosophers think about philosophy, when they get hungry they do the following: (1) take the left fork, (2) take the right fork and start eating, (3) return both forks simultaneously, and repeat from the beginning. Figure 2.6 shows a Petri net model of the system.

Two interesting questions about this systems are:

- Can the philosophers starve to death (because the system reaches a deadlock)?
- Will an individual philosopher eventually eat, assuming she wants to?

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 (see Figure 2.7)

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 the following *objects*: Man, wolf, goat, cabbage, boat. Both can be on either side of the river. It also mentions the following *actions*: Crossing the river, wolf eats goat, goat eats cabbage.

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 are modeled by transitions. Figure 2.7 shows the transitions for the three actions.

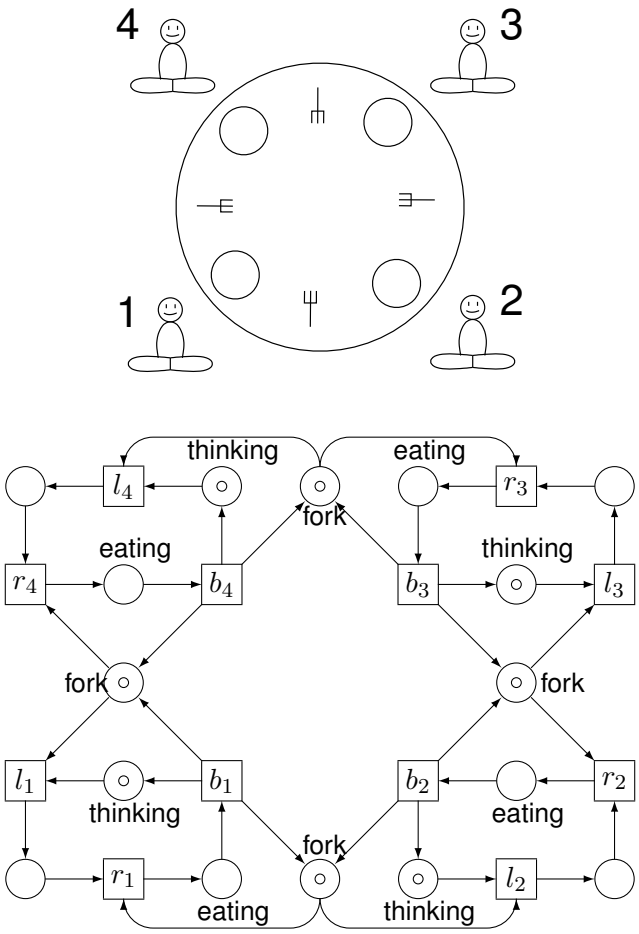


Figure 2.6: Petri net model of the dining philosophers

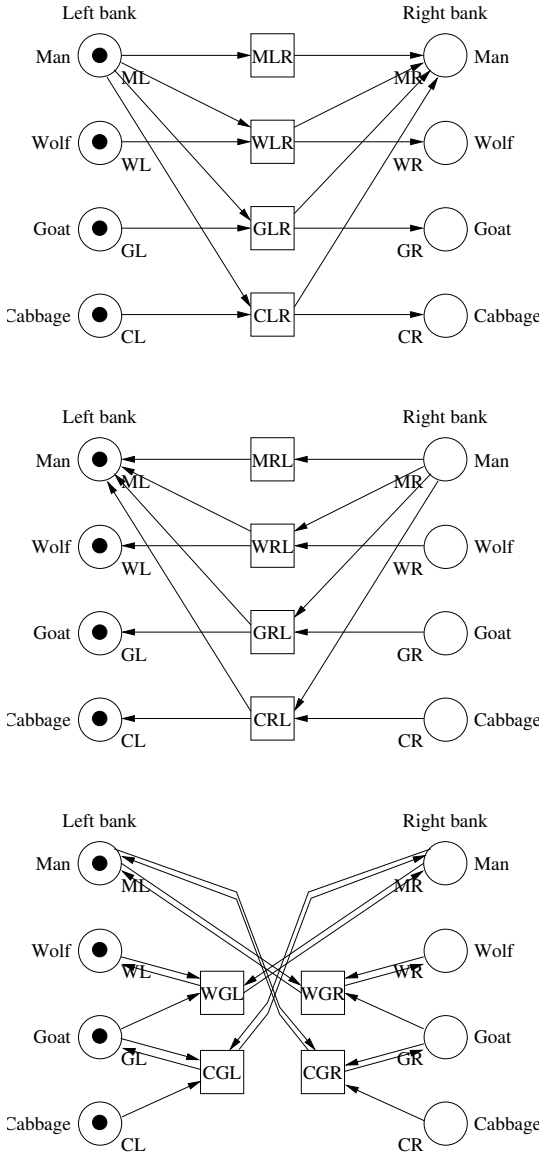


Figure 2.7: Transitions modelling the actions of the puzzle

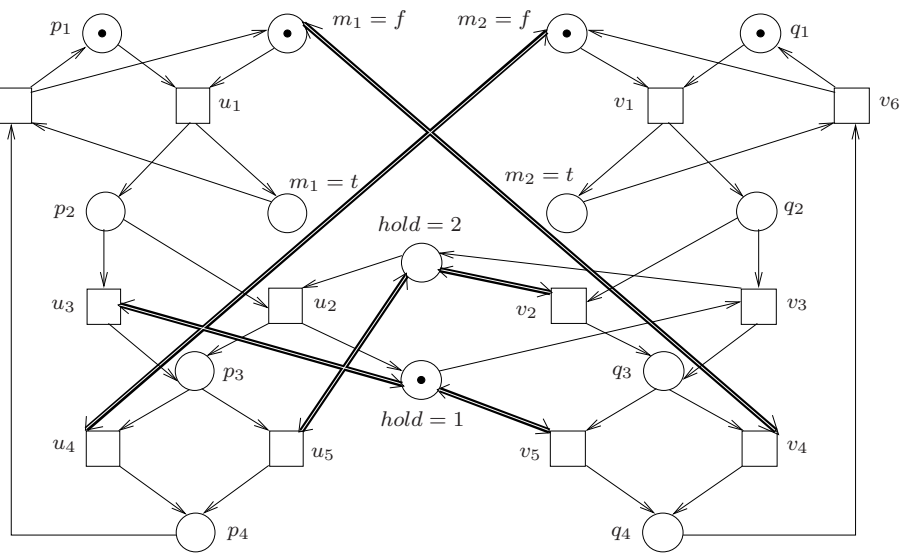


Figure 2.8: Petri net model of Peterson's algorithm

2.5 Peterson's algorithm

Peterson's algorithm is a well-known solution to the mutual exclusion problem for two processes.

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

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

The Petri net of Figure 2.8 models this algorithm. The variable m_i is modeled by the places $m_i = true$ and $m_i = false$. A token on $m_i = true$ means that at the current state of the program (marking) the variable m_i has the value $true$ (so the Petri net must satisfy the property that no reachable marking puts tokens on both $m_i = true$

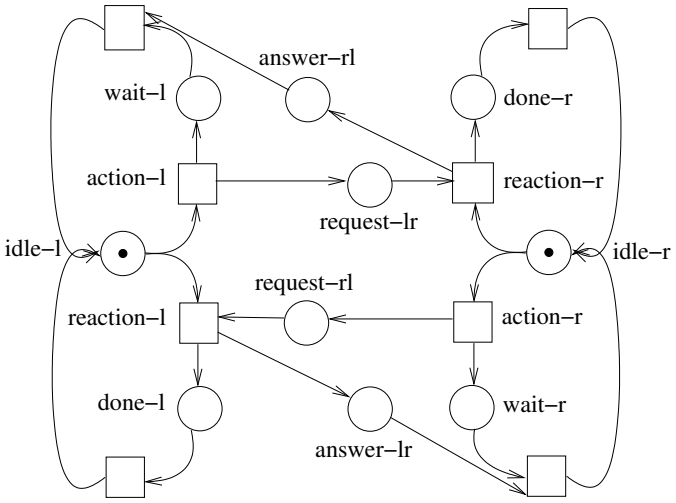


Figure 2.9: First attempt

and $m_i = \mathit{false}$ at the same time). Variable $hold$ is modeled analogously.

A token on p_4 (q_4) indicates that the left (right) process is in its critical section. Mutual exclusion holds if no reachable marking puts a token on p_4 and q_4. The Petri net has 20 reachable markings.

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. The task is to design a protocol for the exchanges. In particular, the protocol must guarantee that it is not possible to reach a situation in which both processes are waiting from an answer from the other one.

A first attempt at a solution is shown in Figure 2.9. Requests are modeled by the Action transitions, and replies by the $\mathit{Reaction}$ transitions. However, 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*. Figure 2.10 shows a second attempt. Now processes can detect that a crosstalk

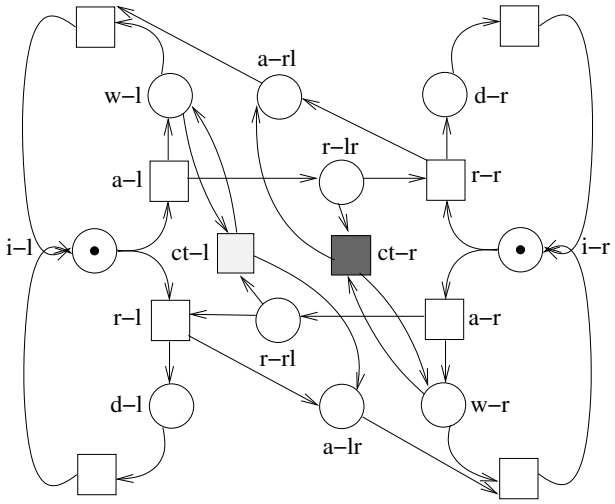


Figure 2.10: Second attempt

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 it exhibits the following problem: 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*. The third attempt (Figure 2.11) is fair. If a process detects a crosstalk, then it answers the request of its partner, as before, but then it moves to a state in which it is only willing to receive an answer to its own question. Unfortunately, the system has again a deadlock (can you find it?).

The final attempt (Figure 2.12) 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. Such a round continues as follows: both processes detect the crosstalk, send each other an "end-of-round" signal, wait for the same signal from their partner, and then 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.

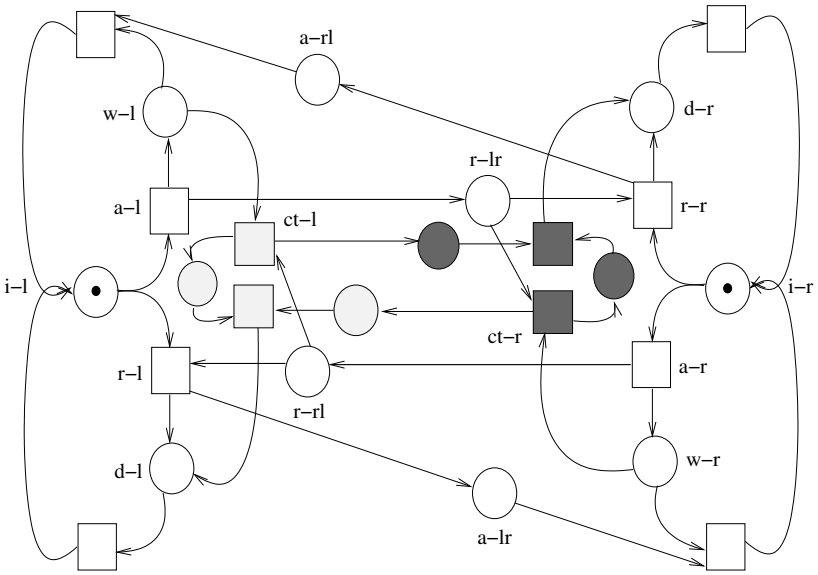


Figure 2.11: Third attempt

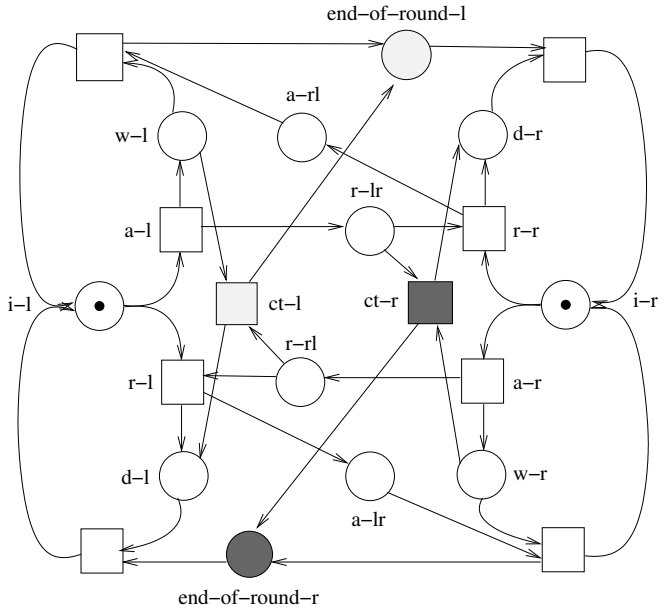


Figure 2.12: Last attempt

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 of places and transitions and a *weight function* $W : (S \times T) \cup (T \times S) \to \mathbb{N}$. A transition t is *enabled* at a marking M of N if $M(s) \geq W(s, t)$ for every $s \in S$. If t is enabled then it 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, for example, $W(s, t) = 3$, then the preset of t contains 3 copies of the place s. Other notions are defined as in the standard model.

Definition 2.7.3 (Nets with inhibitor arcs)
A *net with inhibitor arcs* $N = (S, T, F, I)$ consists of two disjoint sets of places and transitions, a set $F \subseteq (S \times T) \cup (T \times S)$ of arcs, and a set $I \subseteq S \times T$, disjoint with F, of *inhibitor arcs*. A transition t is *enabled* at a marking M of N 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 t is enabled then it can *occur* leading to the marking M', defined as for standard Petri nets.

Definition 2.7.4 (Nets with reset arcs)
A *net with reset arcs* $N = (S, T, F, R)$ consists of two disjoint sets of places and transitions, a set $F \subseteq (S \times T) \cup (T \times S)$ of arcs, and a set $R \subseteq S \times T$, disjoint with F, of

reset arcs. A transition t is *enabled* at a marking M of N if $M(s) > 0$ for every place s such that $(s,t) \in F \cup R$. If t is enabled then it can *occur* leading to the marking obtained after the following operations:

- 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 Some sytems modeled by Petri nets with weight arcs

2.8.1 Readers and writers

The Petri net with weighted arcs of Figure 2.13 models a solution to the "readers and writers" problem. 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.

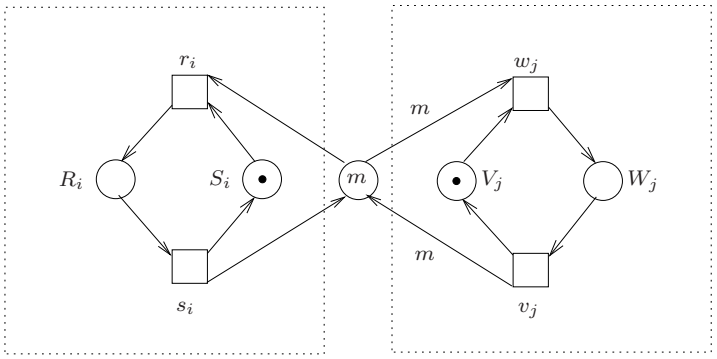
Exercise: Modify the Petri net so that reading processes can not indefinitely prevent another process from writing.

2.8.2 Population protocols

Population protocols are a model of distributed computation by anonymous, identical finite-state agents. While they were initially introduced to model networks of passively mobile sensors, they capture the essence of distributed computation in diverse areas such as trust propagation and chemical reactions.

We introduce them by means of several examples. Then we present the formal definitions, and show how they can be translated into Petri nets.

The Black Ninjas. The *Black Ninjas* are an ancient secret society of warriors. It is so secret that its members do not even know each other and how many they are. When



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

Figure 2.13: Readers and writers

there is a matter to discuss, Sensei, the founder of the society, asks the ninjas to meet at night, preferably during a storm as it minimizes the chance of being surprised by the enemy.

As it happens, all ninjas have just received a note asking them to meet in a certain Zen garden at midnight, wearing their black uniform, in order to decide whether they should attack a nearby castle at dawn. The decision is taken by majority, and in the case of a tie the ninjas will not attack. All ninjas must decide their vote in advance, the only purpose of the meeting is to compute the final outcome.

When the ninjas reach the garden in the gloomy night, dark clouds cover the sky as rain pours vociferously. The weather is so dreadful that it is impossible to see or hear anything at all. For this reason, voting procedures based on visual or oral communication are hopeless. Is there a way for the ninjas to conduct their vote in spite of these adverse conditions?

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

Table 2.1: A first protocol for computing majority.

A first protocol. Sensei has foreseen this situation and made preparations. The note sent to the ninjas contains detailed instructions on how to proceed. Each ninja must wander *randomly* around the garden. Two ninjas that happen to bump into each other exchange information using touch according to the following protocol. Each ninja maintains two bits of information:

- the first bit indicates whether the ninja is currently *active* (A) or *passive* (P); and
- the second bit indicates the current *expectation* of each ninja on the final outcome of the vote: *yes, we will attack* (Y) or *no, we will not attack* (N).

This gives four possible states for each ninja: $\mathrm{A_Y}$, $\mathrm{A_N}$, $\mathrm{P_Y}$, $\mathrm{P_N}$. Initially the ninjas set their first bit to A, *i.e.*, they are all active, and their second bit to their vote. State changes obey *interaction rules* or *transitions* of the form $p, q \mapsto p', q'$, meaning that if the interacting ninjas are in states p and q, respectively, they move to states p' and q'. Sensei specifies two rules, shown on Table 2.1, with the implicit assumption that for any combination of states not covered by the rules, the ninjas must simply keep their current states.

A second protocol. The protocol works fine for a time, but then disaster strikes. At one gathering there is an equal number of Y-ninjas and N-ninjas. In this case — and only in this case — the protocol is incorrect. There is an execution in which the ninjas do not reach consensus, and after which the states of the ninjas cannot change anymore. At dawn only some ninjas attack, they are decimated, and Sensei commits harakiri.

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

Table 2.2: A second protocol for computing majority.

The newly elected Sensei II analyzes the problem and quickly comes up with a repair for the protocol. It is shown in Table 2.2: a new rule $\mathrm{P_Y}, \mathrm{P_N} \mapsto \mathrm{P_N}, \mathrm{P_N}$ is added. If all ninjas become passive, which can only happen in the case of a tie, then the new rule guarantees that an N-consensus is eventually reached.

A third protocol. Again, the new protocol works fine ... until it doesn't. The story repeats itself: At dawn no consensus has been reached, only some ninjas attack, they are decimated. The successor, Sensei III, considers the general scenario in which Y has a majority of only one ninja, and finds the following explanation: In this situation, the protocol reaches with high probability a configuration with one single ninja in state $\mathrm{A_Y}$ and many ninjas in states $\mathrm{P_N}$. There is now a struggle between the single $\mathrm{A_Y}$ ninja, who turns $\mathrm{P_N}$-ninjas to $\mathrm{P_Y}$ using the second rule, against the many $\mathrm{P_N}$-ninjas, who turn $\mathrm{P_Y}$-ninjas back to $\mathrm{P_N}$ using the new rule. The $\mathrm{A_Y}$-ninja eventually "wins", and consensus Y is reached, but only after she turns *all* $\mathrm{P_N}$-ninjas to $\mathrm{P_Y}$ *before* any of the $\mathrm{P_N}$-ninjas converts any of them back to $\mathrm{P_Y}$.

Sensei III wants a new protocol with a clean design. Since ties are the source of all problems, she decides that the protocol should explicitly deal with them. So, apart from being active or passive, ninjas can now have a more refined expectation of the outcome: Y, N, and T (for "tie"). The protocol is shown on Table 2.3. When two active ninjas meet, only one of them becomes passive, and both change their expectation in the natural way. For example, if the expectations are Y and T, then the ninja with expectation T changes it to Y. This explains rules 1 to 4. Rule 5

Rule Nr.	Rule
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	$\mathrm{A}_\alpha, \mathrm{P}_\beta \mapsto \mathrm{A}_\alpha, \mathrm{P}_\alpha \quad \alpha, \beta \in \{\mathrm{Y}, \mathrm{T}, \mathrm{N}\}$

Table 2.3: A third protocol for computing majority.

is the usual one: passive ninjas adopt the expectation of active ninjas.

Formal definition of population protocols. As mentioned in the introduction, a population protocol consists of a set of states Q and a set of transitions $T \subseteq Q^2 \times Q^2$. 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 connection to Petri nets is immediate: The Petri net modeling a protocol has one place for each state, and one transition for every transition of the protocol. If transition t of the Petri net models $(q_1, q_2) \mapsto (q_3, q_4)$, then ${}^\bullet = \wr q_1, q_2 \wr$, and $t^\bullet = \wr q_3, q_4 \wr$, where $\wr q_1, q_2 \wr$. 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 putting $C(q)$ tokens in place q for every $q \in Q$.

Figure 2.14 shows the Petri net for the first two majority protocols. Transitions t such that ${}^\bullet t = t^\bullet$ (whose firing does not change the current marking) have been omitted. Population protocols are designed to compute predicates $\varphi \colon \mathbb{N}^k \to \{0, 1\}$. We first give an informal explanation of how a protocol computes a predicate, and then a formal definition using Petri net terminology. A protocol for φ has a distinguished set of input states $\{q_1, q_2, \ldots, q_k\} \subseteq Q$. Further, 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

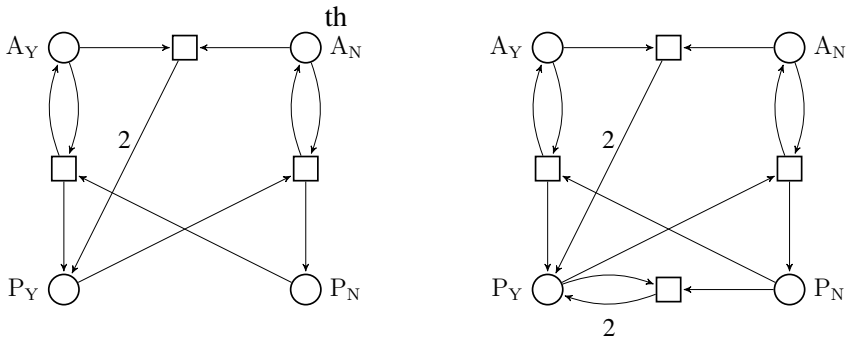


Figure 2.14: Weighted Petri nets for the first and second protocol. Transitions t such that ${}^\bullet t = t^\bullet$ are not shown.

$i = 1, 2$, 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 that in every *fair* run starting at the initial configuration (fair runs are defined formally below), eventually all agents reach states labeled with 1, and stay in such states forever, or they reach states of labeled with 0, and stay in such states forever. So, intuitively, in all fair runs all agents eventually "agree" on a boolean value. By definition, this value is the result of the computation, i.e, the value of $\varphi(n_1, n_2)$.

Formally, and in Petri net terms, fix a Petri net $N = (S, T, W)$ with $|{}^\bullet t| = 2 = |t^\bullet|$ for every transition t. Further, 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 ends at a deadlock marking, or if it is infinite and the following condition holds for all markings M, M' and $t \in T$: if $M \xrightarrow{t} M'$ and $M = M_i$ for infinitely many $i \geq 0$, then $M_j \xrightarrow{t_{j+1}} M_{j+1} = M \xrightarrow{t} M'$ for infinitely many $j \geq 0$. In other words, if a fair sequence reaches a marking infinitely often, then all the transitions enabled at that marking will be fired infinitely often from that marking. A fair firing sequence *converges to* b if there is $i \geq 0$ such that M_j is a b-consensus for every marking $j \geq i$ of the sequence. For every $\boldsymbol{v} \in \mathbb{N}^k$ with $|\boldsymbol{v}| \geq 2$ let $M_{\boldsymbol{v}}$ be

the marking given by $M_{\boldsymbol{v}}(p_i) = \boldsymbol{v}_i$ for every $p_i \in I$, and $M_{\boldsymbol{v}}(p) = 0$ for every $p \in P \setminus I$. We call $M_{\boldsymbol{v}}$ the *initial marking for input* $\boldsymbol{v}$. The net N *computes the predicate* $\varphi\colon \mathbb{N}^k \to \{0,1\}$ if for every $\boldsymbol{v} \in \mathbb{N}^k$, every fair firing sequence starting at $M_{\boldsymbol{v}}$ converges to b.

2.9 Some Petri net models taken from the literature

Petri nets have multiple applications. To complete our collection of models, we present three examples taken from scientific papers in three different areas: biology, manufacturing, and business administration.

A model of a biological system. Petri nets are often used to model biological systems. In these applications, tokens represent molecules or cells, and transitions correspond to chemical reactions or biological processes. Figure 2.15, taken from the paper "Executable cell biology", by J. Fisher and T.A. Henzinger (Nature biotechnology, 2007), shows in part (a) a simple, 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. All nodes are binary (that is, can take the value 0 or 1), except Trp, which is represented by a ternary variable (taking the values 0, 1, 2). Arrows represent activation and bars denote inhibition (inhibitor arcs). Part (c) shows Petri net of the Trp regulatory network. Each of the four components of part (b) is represented by two *complementary places* and all the different situations that lead to a change of the state of the system are modeled by one of the nine transitions $(t_1, \ldots, t_9)$.

A model of a flexible manufacturing system. Figure 2.16, taken from the paper "Optimal Petri-Net-Based Polynomial-Complexity Deadlock-Avoidance Policies for Automated

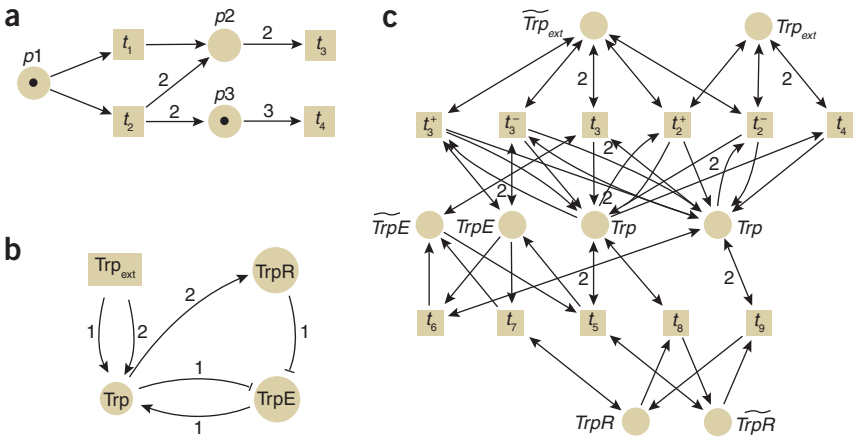


Figure 2.15: Petri net of the Trp regulatory network

Manufacturing Systems" by Xing *et al.* (IEEE Trans. on Systems, Man, and Cybernetics, 2009) shows a flexible manufactuting cell with has 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 model of a business process. Figure 2.16, taken from the paper "Business process management as the Killer App for Petri nets" by van der Aalst (Software and Systems Modeling, 2014), 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 a number of properties which capture, in an abstract way, the types of properties we are interested about when analyzing one of the models of the chapter.

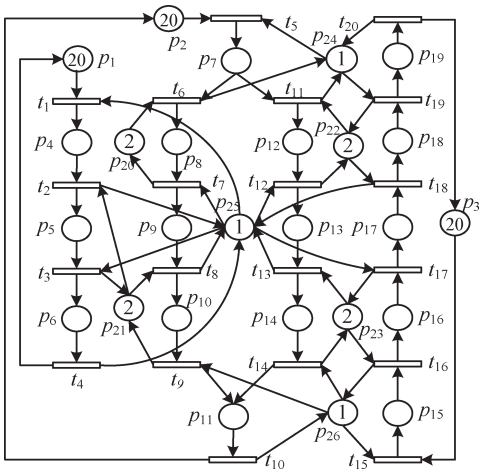


Figure 2.16: Petri net model of a flexible manufacturing system

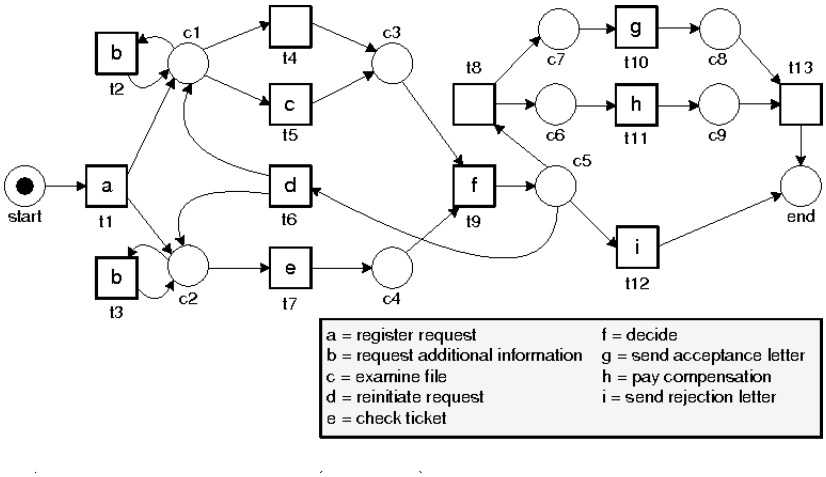


Figure 2.17: Petri net model of a business process

We assume that nets have at least one place and one transition.

Definition 2.10.1 (System properties)
Let (N, M_0) be a Petri net.

(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 b.

In these notes we study the following problems:

- **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 $M' \geq M$?

There are some simple connections between these problems:

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 shows (sometimes without proofs) that **Deadlock-freedom**, **Liveness**, **Boundedness**, b**-Boundedness**, **Coverability**, and **Reachability** are all decidable. The decision procedures for these problems have high complexity, but, at the same time, results of complexity theory show that no efficient algorithms exist for them.

Since better runtimes are often required in many practical applications, we often use algorithms that can be applied to arbitrary Petri nets, but sometimes answer '' don't know", or do not terminate. We call them semi-decision procedures. We also use faster decision procedures for special Petri net classes.

Chapter 4 is devoted to semi-decision procedures. Chapter 5 presents efficient decision algorithms for three classes: S-nets, T-nets, and Free-Choice nets

Chapter 3

Decision procedures

3.1 Decision procedures for 1-bounded Petri nets

In many practical cases Petri nets are bounded by construction. 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**, **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**:

Let $G = (V, E)$ be the reachability graph of a Petri net(N, M_0). We define the relation $\overset{*}{\longleftrightarrow} \subseteq V \times V$ as follows: $M \overset{*}{\longleftrightarrow} M'$ gdw. $M \overset{*}{\longrightarrow} M'$ und $M' \overset{*}{\longrightarrow} M$.

The relation $\overset{*}{\longleftrightarrow}$ is clearly an equivalence relation on V. Each equivalence class $V' \subseteq V$ of $\overset{*}{\longleftrightarrow}$ yields together with $E' = E \cap (V' \times V)$ a strongly connected component (SCC) (V', E') of G.

Strongly connected components are partially ordered by the relation $<$ defined as follows: $(V', E') < (V'', E'')$ if $V' \neq V''$ and $\forall M' \in V',\ M'' \in V'' : M'' \in [M'\rangle$. The *bottom* SCCs of the reachability graph are the maximal SCCs 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 ot the SCC enables t, we are done. □

The condition of Proposition 3.1.1 can be checked in linear time using Tarjan's algorithm, whch 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

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 for 1-bounded Petri nets. Using Savitch's theorem it is also easy to show that they are in **PSPACE**. For example, the following polynomial-memory nondeterministic program solves *Reachability*. Let M_g be the goal marking, that is, the marking whose reachability should be checked. The program stores a marking M; initially $M = M_0$. (Since the net is 1-bounded, if the net has n places then a marking can be stored using n bits, and so the program only needs linear space in the size of the net.) 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 we want the program to always terminate, then we can add to it a n-bit counter that counts the num-

ber 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.

We now show that the problems are **PSPACE**-complete. We do so by means of a universal lower bound for the complexity of deciding whether a 1-bounded Petri net satisfies an interesting behavioural property:

Rule of thumb 1: All interesting questions about the behaviour of 1-bounded Petri nets are **PSPACE**-hard.

Notice that a rule of thumb is not a theorem. There are behavioural properties of 1-bounded Petri nets that can be solved in polynomial time. For instance, the question "Is the initial marking a deadlock?" can be answered very efficiently; however, it is so trivial that hardly anybody would consider it really interesting. So a more careful formulation of the rule of thumb would be that all questions described in the literature as interesting are at least **PSPACE**-hard. Here are 14 examples:

- Is the Petri net live?
- Is some reachable marking a deadlock?
- Is a given marking reachable from the initial marking?
- Is there a reachable marking that puts a token in a given place?
- Is there a reachable marking that does not put a token in a given place?
- Is there a reachable marking that enables a given transition?
- Is there a reachable marking that enables more than one transition?

- Is the initial marking reachable from every reachable marking?
- Is there an infinite run?
- Is there exactly one run?
- Is there a run containing a given transition?
- Is there a run that does not contain a given transition?
- Is there a run containing a given transition infinitely often?
- Is there a run which enables a transition infinitely often but contains it only finitely often?

We need some preliminaries.

Turing machines. We use single tape Turing machines with one-way infinite tapes, i.e., the tape has a first but not a last cell. For our purposes it suffices to consider Turing machines starting on empty tape, i.e., initially the tape containing only blank symbols. So we define a *Turing machine* as a tuple $M = (Q, \Gamma, \delta, q_0, F)$, where Q is the set of states, Γ the set of tape symbols (containing a special *blank* symbol), $\delta\colon (Q \times \Gamma) \to Q \times \Gamma \times \{R, L\})$ 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.

Linearly and exponentially bounded automata. We work with Turing machines that can only use a finite tape fragment, or equivalently, with Turing machines whose tape has both a first and a last cell. We call them *bounded automata*. We assume that the 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.

A function $f\colon \mathbb{N} \to \mathbb{N}$ induces the class of *$f(n)$-bounded automata*, which contains for all $k \geq 0$ the bounded automata of size k that can use $f(k)$ tape cells (including the

first and last cells). [1] When $f(n) = n$ we get the class of *linearly bounded automata*.

The **PSPACE**-hardness of all these problems is a consequence of one single fundamental fact.

> 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.

The notion of simulation used here is very strong: a 1-bounded Petri net simulates a Turing machine if there is bijection f between the configurations of the machine and the markings of the net such that the machine can move from a configuration c_1 to a configuration c_2 in one step if and only if the Petri net can move from the marking $f(c_1)$ to the marking $f(c_2)$ through the firing of exactly one transition.

Let $A = (Q, \Gamma, \Sigma, \delta, q_0, F)$ be a linearly bounded automaton of size n. The computations of M visit at most the cells $c_1, \ldots, c_n$. Let C be this set of cells. The simulating Petri net $N(A)$ contains a place $s(q)$ for each state $q \in Q$, a place $s(c)$ for each cell $c \in C$, and 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 the machine is in state q. A token on $s(c)$ signals that the machine 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 $|Q| + n \cdot (1 + |\Sigma|)$.

The transitions of $N(A)$ are determined by the state transition relation of A. If $(q', a', R) \in \delta(q, a)$, then we have for each cell c a transition $t(q, a, c)$ whose input places are $s(q)$, $s(c)$, and $s(a, c)$ and whose output places are $s(q')$, $s(a', c)$ and $s(c')$, where c' is the cell to the right of c. If $(q', a', L) \in \delta(q, a)$ then we add a similar set of transitions. The total number of transitions is at most

[1] Notice that we deviate from the standard definition, which says that an automaton is $f(n)$-bounded if it can use at most $f(k)$ tape cells for an *input word* of length k. Since we only consider bounded automata working on empty tape, the standard definition is not appropriate for us.

$2 \cdot |Q|^2 \cdot |\Gamma|^2 \cdot n$, and so $O(n^2)$, because the size of A is $O(|Q|^2 \cdot |\Gamma|^2)$.

The initial marking of $N(A)$ puts one token on $s(q_0)$, on $s(c_1)$, and on the place $s(B, c_i)$ for $1 \leq i \leq n$, where B denotes the blank symbol. The total size of the Petri net is $O(n^2)$.

It follows immediately from this definition that each move of A corresponds to the firing of one transition. The configurations reached by A along a computation correspond to the markings 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 $\{s(a, c) \mid a \in \Sigma\}$ 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 any of the computations of A terminate?" corresponds to "has the Petri net $N(A)$ a deadlock?"

It turns out that most questions about the computations of linearly bounded automata are **PSPACE**-hard. To begin with, the *(empty tape) acceptance problem* is **PSPACE**-complete:

> Given: a linearly bounded automaton A.
> To decide: if A accepts the empty input.

Moreover, the **PSPACE**-hardness of this problem is very robust: it remains **PSPACE**-complete if we restrict it to

- bounded automata having one single accepting state,
- bounded automata having one single accepting configuration.

Many other problems can be easily reduced to the acceptance problem in polynomial time, and so 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?

We obtain in this way a large variety of **PSPACE**-hard problems. Since $N(A)$ is only polynomially larger than A, all the corresponding Petri net problems are **PSPACE**-hard as well. For instance, a reduction from the problem "does A ever visit a given configuration?" proves **PSPACE**-hardness of the reachability problem for 1-bounded Petri nets. Furthermore, once we have some **PSPACE**-hard problems for 1-bounded Petri nets we can use them to obtain new ones by reduction. For instance, the following problems can be easily reduced to the problem of deciding if there is a reachable marking that puts a token on a given place:

- 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?

13 out of the 14 problems at the beginning of the section (and many others) can be easily proved **PSPACE**-hard using these techniques. Only the liveness problem, the first in our list, is a bit more complicated.

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-**Boundednesss** 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 **Boundedeness**. We need two lemmas. The first one is a simple adaptation of König's Lemma; the second is known as 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 there is a simple path π_M from M_0 to M. Since M_0 has finitely many immediate successors (at most one for each transition of N), and each simple path π_M visits one of them, at least one immediate successor M_1 of M_0 has infinitely many successors in $(V \setminus \{M_0\}, E)$, that is, $[M_1\rangle \setminus \{M_0\}$ is infinite. Iterating this argument we construct an infinite simple path $M_0M_1M_2\cdots$. □

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 indices 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 $A'_1 A'_2 A'_3 \cdots$ 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. □

Remark: Lemma 3.2.2 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 $a_1 a_2 a_3 \cdots \in A^\omega$ contains an infinite chain $a_{i_1} \preceq a_{i_2} \preceq \cdots$. In the next section we examine well-quasi-orders in more detail.

We use König's Lemma and Dickson's lemma to provide the following characterization of unboundedness.

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 thatthe markings $M_0, M_1, M_2, \ldots$ are pairwise distinct. By Dickson's Lemma there are indices $i < j$ 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 $L \neq 0$. □

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".

If (N, M_0) is bounded, then by Theorem 3.2.3 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 the algorithm cannot terminate because it runs out of reachable markings. On the other hand, by Theorem 3.2.3 the algorithm eventually finds markings M' and M as above, and so it correctly answers "unbounded". □

3.2.2 Decision procedures for Coverability

The reachability graph of a Petri net can be infinite, in which case the algorithm for computing the reachability graph will not terminate. Therefore, the algorithm cannot decide that a given marking is not coverable. In this section we introduce several decision procedures that overcome this problem.

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

We introduce a new symbol ω. Intuitively, it stands for an arbitrarily large number. We extend the arithmetic on natural numbers with ω as follows. For all $n \in \mathbb{N}$:

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

Observe that $\omega - \omega$ remains undefined, but we will not need it.

We extend the notion of markings to *ω-markings*. 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 s has either a certain number of tokens or "arbitrarily many" tokens.

The enabledness condition and the firing rule neatly extend to ω-markings with the extended arithmetic rules: recall that a transition t is enabled at a marking M if $M(s) > 0$ for every $s \in {}^\bullet t$. Now $M(s) > 0$ may hold because $M(s) = \omega$. Further, recall that if t is enabled, then it can fire, leading from M to the marking M' given by:

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

If $s \in {}^\bullet t \cup t^\bullet$ and $M(s) = \omega$, then we have $M'(s) = \omega$. That is, if a place contains ω tokens, then firing a transition will not change its number of tokens, even if the transition is connected with an arc to the place.

Assume $M' \in [M\rangle$ and $M \leq M'$. Then there is some sequence of transitions $t_1 t_2 \ldots t_n$ such that $M \xrightarrow{t_1 t_2 \ldots t_n} M'$. By the Monotonicity Lemma, there is a marking M'' with $M' \xrightarrow{t_1 t_2 \ldots t_n} M''$. Further, if we denote $\Delta M := M' - M$, then $M'' = M' + \Delta M = M + 2\Delta M$ (see Figure 3.1). By firing the transition sequence $t_1 t_2 \ldots t_n$ repeatedly we can "pump" an arbitrary number of tokens to all the places s for which $\Delta M(s) > 0$.

The main idea for the construction of the coverability graph is to replace the marking M' by the ω-marking

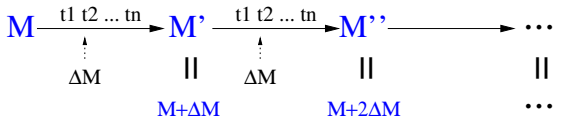


Figure 3.1: Pumping tokens.

$M' + \omega \cdot \Delta M$. The algorithm is shown in Figure 3.2. The following notations are used in the AddOmegas subroutine:

- $M'' \rightarrow_E M$ iff $(M'', t, M) \in E$ for some $t \in T$.
- $M'' \xrightarrow{*}_E M$ iff $\exists n \geq 0$: $\exists M_0, M_1, \ldots, M_n$: $M'' = M_0 \rightarrow_E M_1 \rightarrow_E M_2 \rightarrow_E \cdots \rightarrow_E M_n = M$.

Observe that COVERABILITY-GRAPH is very similar to REACHABILITY-GRAPH, it just adds a call to subroutine AddOmegas(M, t, M', V, E). Line 3 causes all places whose marking in M' is strictly larger than in the "parent" M'' to contain ω, while markings of other places remain unchanged.

We show that COVERABILITY-GRAPH terminates, and that a marking M of (N, M_0) is coverable iff some node M' of the coverabilitygraph of (N, M_0) covers M, i.e., satisfies $M' \geq M$

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

```
COVERABILITY-GRAPH((P, T, F, M_0))
 1  (V, E, v_0) := ({M_0}, ∅, M_0);
 2  Work : set := {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          M' := AddOmegas(M, t, M', V, E);
 9          if M' ∉ V
10            then V := V ∪ {M'}
11                 Work := Work ∪ {M'};
12          E := E ∪ {(M, t, M')};
13  return (V, E, v_0);
```

```
ADDOMEGAS(M, t, M', V, E)
1  for M'' ∈ V
2  do if M'' < M' and M'' →*_E M
3        then M' := M' + ((M' − M'') · ω);
4  return M';
```

Figure 3.2: Algorithm for the construction of the coverability graph

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 $\mathsf{AddOmegas}(M_{l-1}, t, M_l, V, E)$ the algorithm adds at least one ω to M_{l-1}. □

For the rest of the proof we start with a lemma. It states that if COVERABILITY-GRAPH adds an ω-marking M', say $M' = (\omega, 2, 0, \omega, 3, \omega)$, then for every k, say 15, there is a reachable marking where the ω-components have at least the value 15; for example, for 15 the marking could be $(17, 2, 0, 234, 3, 15)$. In other words, if the algorithm adds an ω-marking $(\omega, 2, 0, \omega, 3, \omega)$, it is possible to reach arbitrarily large values for all ω-components simultaneously.

Definition 3.2.6 *G*iven an ω-marking M, we say that s is an *ω-place of 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 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 is currently exploring marking M and transition t, and let $\overline{M}$ be the ω-marking such that $M \xrightarrow{t} \overline{M}$. 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.

Assume AddOmegas does not add any new ω to $\overline{M}$, and $\overline{M} \notin V$. Then COVERABILITY-GRAPH adds $\overline{M}$ to V, and we can take $\overline{M}_k$ as the marking given by $M_{k+1} \xrightarrow{t} \overline{M}_k$: Indeed, since $M_{k+1}(s) > k + 1$ for every ω-place of

M, and t removes at most one token from s, we have $\overline{M}_k(s) > k$.

Assume now that AddOmegas finds a unique ω-marking M'' satsifying the condition $M'' \xrightarrow{*}_E M \xrightarrow{t} \overline{M}$ and $M'' \leq \overline{M}$. Assume further that $M''(s_0) < \overline{M}(s_0)$ holds for a unique place s_0 (the case where there is more than one such M'' and more than one such s_0 is similar). Then COVERABILITY-GRAPH adds to V the marking $M' = \overline{M} + (\overline{M} - M'') \cdot \omega$, which has exacly one more ω-place than M'', namely the place s_0. We prove that for every $k > 0$ it is possible to reach a 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'.

Fix an arbitrary k. Our goal is to construct the marking M'_k.

Let σ be a firing sequence such that $M'' \xrightarrow{\sigma} M \xrightarrow{t} \overline{M}$. 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''.

Observe that the sequence σt may not be firable from M''_ℓ: Indeed, consider an ω-place s of M''. The sequence σt may remove tokens from s, but if the sequence is fired from M'' then we do not "notice" because $M''(s) = \omega$. If ℓ is not large enough, then $M''_\ell(s)$ may not have enough tokens to fire σt. However, M''_ℓ enables σt for every sufficiently large value of ℓ. [2] Even more: since $M''(s) \leq \overline{M}(s)$ for every normal place s of M'', by choosing a sufficiently large ℓ we can execute σt from M''_ℓ not only once, but as many times as we want. Moreover, since $M''(s_0) < \overline{M}(s_0)$, by executing σt a sufficiently large number of times we can put arbitrarily many tokens in the place s_0.

We can now finally define the marking M'_k. Choose a number ℓ large enough to guarantee that

(1) M''_ℓ enables $(\sigma t)^{k+1}$, and

[2] For instance, take $\ell = |\sigma t|$, since $\sigma\, t$ can remove at most $|\sigma t|$ tokens from a place.

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

Since the execution of σt adds at least one token to s_0, after the execution of $(\sigma t)^{k+1}$ the place s_0 has at least $k+1$ tokens, and so $\overline{M}_\ell(s) > k$. Since $\overline{M}_\ell$ also satisfies (2), we can take $M'_k := \overline{M}_\ell$. □

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 3.2.7, 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. The question is whether we can give a bound on the size of the *fragment* of the coverability graph we need to construct to find an ω-marking covering M.

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 3.2.9 (Integer nets) Let $N = (S, T, F)$ be a net. A *generalized marking* of N (*g-marking* for short) is a mapping $G\colon S \to \mathbb{Z}$. An *integer 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' given by

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

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 net is a sequence $G_0 \overset{t_1}{\hookrightarrow} G_1 \overset{t_2}{\hookrightarrow} \cdots \overset{t_n}{\hookrightarrow} G_m$.

Every marking is also a g-marking, ever Petri net is also an integer net, and every firing sequence is also an integer firing sequence, but the converse does not hold.

In the rest of the section we fix a net N with places $\{s_1, \ldots, s_k\}$, and identify g-markings with vectors of $\mathbb{Z}^k$.

Definition 3.2.10 *L*et $G \in \mathbb{Z}^k$ 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 $1 \leq j \leq i$.[3]

Let $\sigma = G_0 \overset{t_1}{\hookrightarrow} \cdots \overset{t_m}{\hookrightarrow} G_m$ be an integer firing sequence, and let $G \in \mathbb{Z}^k$.

- σ is *i-natural* if all of $G_0, G_1, \ldots, G_m$ are i-natural g-markings.
- σ is *(i, r)-natural* if all of $G_0, G_1, \ldots, G_m$ are (i, r)-natural g-markings.
- σ is *(i, G)-covering* if $G_m(j) \geq G(j)$ for every $1 \leq j \leq i$. [4]

[3] Observe that we require $G(j) < r$, not $G(j) \leq r$.

[4] But possibly $G_m(j) < G(j)$ for components $j > i$.

Intuitively, G is i-natural if its restriction to the first i places is a "normal" marking, and σ is i-natural if its restriction to the first i places is a "normal" firing sequence. In particular, since N has k places, the k-natural sequences ane the "normal" firing sequences coincide. So deciding if M is coverable in (N, M_0) is equivalent to deciding if (N, M_0) has a (k, M)-covering sequence.

We prove the following result:

Theorem 3.2.11 *[Rackoff 1978] Let $M \in \mathbb{N}^k$ be a marking of N, and let $n = 1 + \sum_{i=1}^{k} M(i)$. For every marking $M_0 \in \mathbb{N}^k$ of N, if (N, M_0) has a (k, M)-covering sequence, then it has one of length at most $(2n)^{(k+1)!} \in n^{2^{O(k \log k)}}$.*

Before proving the theorem, we make two observations:

- The bound $(2n)^{(k+1)!}$ depends on the number of tokens of M and on k, the number of places of N. However, it does *not* depend on the number of tokens of M_0.

- The bound is polynomial on n (for Petri nets with a fixed number k of places, the bound is of the form $O(n^c)$ for some constant c), but double exponential on k (for markings with a fixed number n of tokens, the bound is of the form $2^{2^{ck \log k}}$ for some constant c).

The proof of the theorem is based on the following 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, G)-covering sequence, 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* $i \geq 1$.

Proof The proof is by induction on i.
Base: $i = 0$. Follows vacuously from the fact that G_0 is a $(0, G)$-covering sequence (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) an (i, G)-covering sequence. We consider two cases:

Case 1: (N, G_0) has an (i, G)-covering, $(i, nf(i-1))$-natural sequence.
Assume the sequence is

$$\sigma = G_0 \stackrel{t_1}{\hookrightarrow} \cdots \stackrel{t_m}{\hookrightarrow} G_m$$

and assume further that it has minimal length.

We claim that $G_0, G_1, \ldots, G_m$ are pairwise different. Assume the contrary: there exist $\alpha < \beta$ such that $G_\alpha(j) = G_\beta(j)$ for every $1 \leq j \leq i$. Then the sequence

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

is also (i, G)-covering and $(i, nf(i-1))$-natural, contradicting the minimality of σ. This proves the claim.

Since σ is $(i, nf(i-1))$-natural, for every g-marking G' appearing in σ and for for every $1 \leq j \leq i$ we have $0 \leq G'(j) < nf(i-1)$. That is, there are $nf(i-1)$ possible values for $G'(j)$. It follows that there are at most $(nf(i-1))^i$ different possible values for the segment $(G'(1), G'(2), \ldots, G'(i))$ of G'. By the claim above, the length of σ is at most $(nf(i-1))^i \leq f(i)$.

Case 2: (N, G_0) has no (i, G)-covering, $(i, nf(i-1))$-natural sequence.
Then there is an (i, G)-covering sequence that is *not* $(i, nf(i-1))$-natural. Let this sequence be

$$\sigma = G_0 \stackrel{t_1}{\hookrightarrow} G_1 \stackrel{t_2}{\hookrightarrow} \cdots G_{m-1} \stackrel{t_m}{\hookrightarrow} G_m$$

Let $G_{\alpha+1}$ be the first vector of σ that is not $(i, nf(i-1))$-natural. Without loss of generality, we can assume $G_{\alpha+1}(i) \geq nf(i-1)$ (if this would not be the case, we can just reorder the places of the net). Then the prefix

$$G_0 \stackrel{t_1}{\hookrightarrow} \cdots \stackrel{t_\alpha}{\hookrightarrow} G_\alpha$$

is (i, G_α)-covering and $(i, nf(i-1))$-natural. As in the previous case, we can assume $\alpha \leq (nf(i-1))^i$.

Since

$$G_{\alpha+1} \overset{t_{\alpha+1}}{\hookrightarrow} G_{\alpha+2} \overset{t_{\alpha+2}}{\hookrightarrow} \cdots \overset{t_m}{\hookrightarrow} G_m$$

is an $(i-1, G)$-covering and $(i-1)$-natural sequence of $(N, G_{\alpha+1})$, by induction hypothesis there exists another $(i-1, G)$-covering and $(i-1)$-natural sequence

$$G_{\alpha+1} \overset{u_1}{\hookrightarrow} H_1 \overset{u_2}{\hookrightarrow} \cdots \overset{u_\ell}{\hookrightarrow} H_\ell$$

of $(N, G_{\alpha+1})$ of length at most $f(i-1)$, that is, $\ell \leq f(i-1)$. Since $G_{\alpha+1}(i) \geq nf(i-1)$, and a sequence of length $f(i-1)$ can remove at most $f(i-1)$ tokens from the place s_i, after the execution of the new sequence the number of tokens in s_i is at least $(n-1)f(n-1) \geq n-1$. By the definition of n, we have $n-1 \geq G(i)$. So the sequence

$$\sigma' = G_0 \overset{t_1}{\hookrightarrow} \cdots \overset{t_\alpha}{\hookrightarrow} G_\alpha \overset{t_{\alpha+1}}{\hookrightarrow} \overset{u_1}{\hookrightarrow} H_1 \overset{u_2}{\hookrightarrow} \cdots \overset{u_\ell}{\hookrightarrow} H_\ell$$

is an (i, G)-covering and i-natural sequence of (N, G_0) of length at most $(nf(i-1))^i + f(i-1) = f(i)$. □

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 $f(i+1) \in O(n^{(1+j)(1+i)})$. We now proceed to prove Theorem 3.2.11:

Proof of Theorem 3.2.11. Assume that (N, M_0) has a (k, M)-covering sequence. By Lemma 3.2.12, 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 immediately 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, let us prove $(2n)^{(k+1)!} \in n^{2^{O(k \log k)}}$. We first show $(k+1)! \in 2^{O(k \log k)}$.

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

Now, 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 Theorem 3.2.11, in order to decide coverability of M we can just construct the reachability graph using breadth-first search up to depth $(2n)^{(k+1)!}$, where n is the number of tokens of M plus 1, and k is the number of places in the net. Clearly, the same holds for the coverability graph, because, loosely speaking, it just "improves" our chances of covering M.

The backwards-reachability algorithm

The backwards reachability 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 $M' \neq M$.

Observe that an upward closed set is completely determined by its minimal elements: two upwards closed sets are equal iff their sets of minimal elements are equal.

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

Proof. Assume $\mathcal{M}$ is upward closed and has infinitely many minimal markings $M_1, M_2, \ldots$. By Dickson's Lemma there are $i \neq j$ such that $M_i \leq M_j$. But then M_j is not minimal. Contradiction. □

An important consequence of the lemma is that every upwards closed set can be *finitely represented* by its set of minimal elements.

We define the set $pre(\mathcal{M})$ of predecessors of a set of markings $\mathcal{M}$, and the set $pre^*(\mathcal{M})$ of markings from which one ca reach some marking of $\mathcal{M}$.

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

$$\begin{aligned} pre(\mathcal{M}, t) &= \{M' \mid M' \xrightarrow{t} M \text{ for some } M \in \mathcal{M}\} \\ pre(\mathcal{M}) &= \bigcup_{t \in T}^{\infty} pre(\mathcal{M}, t) \end{aligned}$$

and further

$$\begin{aligned} pre^0(\mathcal{M}) &= \mathcal{M} \\ pre^{i+1}(\mathcal{M}) &= pre\big(pre^i(\mathcal{M})\big) \text{ for every } i \geq 0 \\ pre^*(\mathcal{M}) &= \bigcup_{i=0}^{\infty} pre^i(\mathcal{M}) \end{aligned}$$

We have the following fundamental lemma:

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

Proof. We first show that $pre(\mathcal{M})$ is upward closed. Let $M' \in pre(\mathcal{M})$. We have to prove that $M' + M'' \in pre(\mathcal{M})$ holds for every marking M''.

Since $M' \in pre(\mathcal{M})$ there is $M \in \mathcal{M}$ and a transition t such that $M' \xrightarrow{t} M$. 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 $M' + M'' \in pre(\mathcal{M})$.

Now we prove that $pre^*(\mathcal{M})$ is upward closed. By repeated application of teh first part of this lemma we obtain that $pre^j(X)$ is upward closed for every $j \geq 0$. So $pre^*(\mathcal{M})$ is a union of upward-closed sets. But it follows immediately from the definition of an upward-closed set that the union of upward-closed sets if also upward closed. □

Combining Lemma 3.2.16 and Lemma 3.2.14 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 3.2.16, $pre^*(\mathcal{M})$ is upward closed. By Lemma 3.2.14, the set m^* of minimal markings of $pre^*(\mathcal{M})$ is finite. therefore, there exists an index i such that $m^* \subseteq \bigcup_{j=0}^{i} pre^j(\mathcal{M})$. Since this union is upward

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

```
BACK1((P, T, F, 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 covered end
8     if M = Old_M
9       then return not covered end
```

```
BACK2((P, T, F, M))
1  m := {M};
2  old_m := ∅;
3  while true
4  do old_m := m;
5     m := min(m ∪ ⋃_{t∈T} pre(R[t] ∧ m));
6     if ∃M' ∈ m : M_0 ≥ M'
7       then return covered end
8     if m = old_m
9       then return not covered end
```

Figure 3.3: Backwards reachability algorithm.

This theorem leads to the algorithm on the left of of Figure 3.3. Observe that the termination of the algorithm follows from Dickson's Lemma, and does not require knowledge of Petri nets. In particular, the termination argument does not provide information on the number of iterations of the while loop. However, using Rackoff's theorem we can obtain an upper bound.

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 $\mathcal{M}$, let $n_i = 1 + \sum_{s \in S} M_i(s)$ for every $1 \leq i \leq m$, 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 a marking $M' \in \mathcal{M}$ such that $M \xrightarrow{*} M'$, and so $M' \geq M_i$ for some minimal marking M_i. By Theorem 3.2.11 and the definition of n, there exists a firing sequence $M \xrightarrow{\sigma} M'' \geq M_i$ such that $|\sigma| \leq (2n)^{(k+1)!}$. Since $\mathcal{M}$ is upward closed, we have $M'' \in \mathcal{M}$, and so $M \in pre^j(\mathcal{M})$ for $j = |\sigma| \leq (2n)^{(k+1)!}$. □

The algorithm on the left of of Figure 3.3 is not yet directly implementable, because it manipulates infinite sets. For each operation (union and pre) and for each test (the tests $\mathcal{M} \stackrel{?}{=} \mathit{Old\_M}$ and $M_0 \stackrel{?}{\in} \mathcal{M}$ of the while-loop), we have to supply an implementation that uses only the *finite representation* of the set, that is, its set of minimal elements. For the tests this is easy. Given a set $\mathcal{M}$, let $\min(\mathcal{M})$ denote the set of minimal elements of $\mathcal{M}$. We have:

(1) $M_0 \in \mathcal{M}$ iff there exists $M' \in \min(\mathcal{M})$ such that $M_0 \geq M'$.

(2) $\mathcal{M} = \mathit{Old\_M}$ iff $\min(\mathcal{M}) = \min(\mathit{Old\_M})$.

Let us now derive implementations for the operations $\cup$ and pre. Given a transition t, let $R[t]$ be the marking that puts one token in each output place of t, and no tokens elsewhere. (This is the minimal marking that *reverse-enables* t, i.e., the minimal marking that allos us to fire t "backwards"). Further, given a set $\mathcal{M}$ of markings let $\mathcal{M}[t] = \{M \in \mathcal{M} \mid M \geq R[t]\}$. That is, $\mathcal{M}[t]$ is the set of markings of $\mathcal{M}$ that enable t "backwards" Observe that if $\mathcal{M}$ is upward closed, then so is $\mathcal{M}[t]$. We have (exercise):

$$\begin{aligned} (3)\quad \min(\mathcal{M} \cup pre(\mathcal{M})) &= \min(\min(\mathcal{M}) \cup \min(pre(\mathcal{M})) \\ &= \min(\min(\mathcal{M}) \cup \bigcup_{t \in T} \min(pre(\mathcal{M}, t)) \end{aligned}$$

(4) $\min(pre(\mathcal{M}, t)) = pre(\min(\mathcal{M}[t]))$.

These equations reduce the problem of computing $\min(\mathcal{M} \cup pre(\mathcal{M}))$ to computing $\min(\mathcal{M}[t])$. For this we introduce a definition. Given two markings M, M', let $M \wedge M'$ be the marking defined by $(M \wedge M')(s) = \max\{M(s), M'(s)\}$ for every place s. Further, given a marking M and a set of markings $\mathcal{M}$, define $M \wedge \mathcal{M} = \{M \wedge M' \mid M' \in \mathcal{M}\}$.

Then we have (exercise)

(5) If $\mathcal{M}$ is upward closed, then $\min(\mathcal{M}[t]) = R[t] \wedge \min(\mathcal{M})$. [5]

That is, the minimal markings of $\mathcal{M}$ that reverse-enable t are obtained by taking the minimal markings of $\mathcal{M}$, and computing their join with the marking $R[t]$. Putting together (3)-(5) we obtain

(6) If $\mathcal{M}$ is upward closed, then

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

Using these observations, we obtain the implementable version of the backwards-reachability algorithm shown on the right of Figure 3.3.

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. This is an important advantage of the backwards reachability algorithm over the coverability graph technique.

Definition 3.2.19 *G*iven a set A, and a partial order $\preceq \subseteq A \times A$, 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 $a_{i_1} \preceq a_{i_2} \preceq \cdots$ (where $i_1 < i_2 < i_3 \ldots$).

Here are some examples of well-quasi-orders:

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

- The pointwise order $\leq$ on $\mathbb{N}^k$.
- The subword order on Σ^* for any finite alphabet Σ. We say $w_1 \preceq w_2$ if w_1 is a scattered subword of w_2, that is, 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 $\preceq\ A \times A$ be a wqo. 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

$$pre(X) = \{y \in A \mid y \rightarrow x \text{ and } x \in X\}$$

Further we define:

$$\begin{aligned} pre^0(X) &= X \\ pre^{i+1}(X) &= pre\big(pre^i(X)\big) \text{ for every } i \geq 0 \\ pre^*(X) &= \bigcup_{i=0}^{\infty} pre^i(X) \end{aligned}$$

We can now prove the following theorem:

Theorem 3.2.21 *Let A be a set and let $\preceq\ A \times A$ be a wqo. Let $X_0 \subseteq A$ 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, or *lossy channel systems*, whose transition relation is monotonic. 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.*

3.2.3 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 simplied 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, very simple 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 $r \in \mathbb{N}^k$ (the root) and a finite set $P \subseteq \mathbb{N}^k$ (the periods) such that

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

A *semilinear set* is a finite union of linear sets.

Observe that 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 (N, M_0) be a Petri net and let M be a marking of M. If M 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 *of* N*, then* $M' \in \mathcal{M}$*, and*

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

Given the root r and periods $p_1, \ldots, p_n$ of a semilinear set $\mathcal{M}$, we can check whether $\mathcal{M}$ satisfies (a)-(c). Indeed, checking (a) amounts to solving the linear system of diophantine equations

$$M_0 = r + \sum_{i=1}^{n} \lambda_i p_i$$

with unknowns $\lambda_1, \ldots, \lambda_n$. Similarly, checking (c) amounts to showing that

$$M = r + \sum_{i=1}^{n} \lambda_i p_i$$

has no solution. Finally, checking (b) is more complicated, but reduces to checking validity of a formula of a theory called Presburger arithmetic for which decision procedures exist.

Now, Theorem 3.2.24 can be used to give an algorithm for **Reachability** consisting of two semi-decision procedures, one that explores the reachability graph breadth-first and stops if it finds the goal marking M, and another one that enumerates all semilinear sets, and stops if one of them satisfies (a)-(c). The two procedures run in parallel, and, since one of the two is bound to terminate, yield together a decision procedure for **Reachability**.

Deadlock-freedom

We reduce **Deadlock-freedom** to **Reachability**. We proceed in two stapes. First, we reduce **Deadlock-freedom** to an auxiliary problem **P**, 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

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

that is, an element of $\mathcal{S}$ contains for every transition t at least on ef the input places of 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 N 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 $s \in R$. It follos from (2) that (N, M_0) is deadlock-free if the answer is negative in all cases. Since, by (1), we only have to solve a finite number of instances of **P**, **Deadlock-freedom** is decidable. □

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 (see Figure 3.4):

- Add two new places s_0 and r_0. Put one token on s_0.
- Add a transition t_0 and arcs (s_0, t_0) and (t_0, r_0).
- For every transition $t \in T$, add two arcs (s_0, t) und (t, s_0).

While s_0 remains marked, all transitions of T can fire. However, transition t_0 can occur at any time, and when this happens all transitions of T become "dead". Intuitively, the firing of t_0 "freezes" (N, M_0).

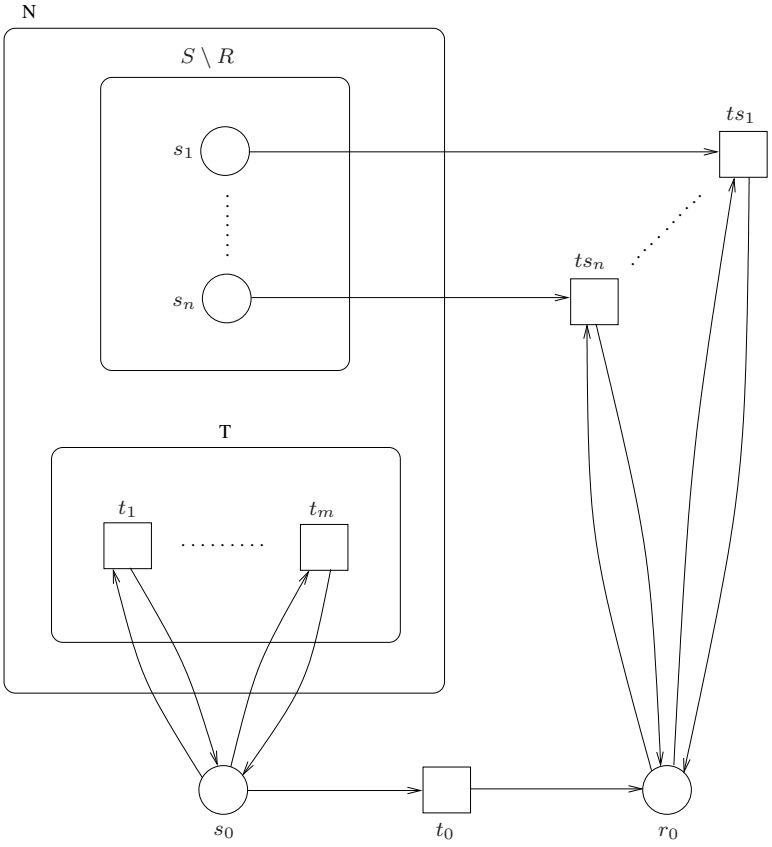


Figure 3.4: Construction of Theorem 3.2.26

- For every place $s \in S \setminus R$ add a new transition t_s and arcs $(s, t_s), (r_0, t_s), (t_s, r_0)$.

If r_0 is marked, then the t_s transitions can occur. These transitions "empty" the places of $S \setminus R$.

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 transiitons of T to reach M, then we fire t_0, and finally we fire t_s transitions until S is 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**, but the proof is more complex. We sketch the reduction for the problem whether a given transition t of a Petri net (N, M_0) is live.

Let E_t be the set of markings of N that enable t. Clearly, E_t is upward closed. By Lemma 3.2.16, the set $pre^*(E_t)$ is also upward closed. Now, $pre^*(E_t)$ 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 $[M_0\rangle \cap D_t = \emptyset$.

If D_t is a finite set of markings, and we are able to compute it, then we are done: we have reduced the liveness problem to a finite number of instances of **Reachability**. However, the set D_t may be infinite, and we do not yet know how to compute it. We show how to deal with these problems.

Every upward-closed set of markings is semilinear (exercise). Using the backwards reachability algorithm, we can compute the finite set $\min(pre^*(E_t))$, and from it we can compute a representation of $pre^*(E_t)$ as a semilinear set. Now we use a powerful result: 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 $\mathbb{N}^k \setminus X$. So we are left with the problem: given a Petri net (N, M_0) and a semilinear set X, decide if some marking of X is reachable from M_0.

This problem can be reduced to **Reachability** as follows (brief sketch). We construct a Petri net that first simulates (N, M_0), and then transfers control to another Petri net which 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 reeachble, then the first net can produce a marking of X, the second net can produce the same marking, and the thrid 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.

3.2.4 Complexity

Unfortunately, all the problems we have seen so far have very high complexity. We prove that all of them are **EXPSPACE**-hard. That is, the memory needed by any algorithm solving one of these problems grows at least exponentially in the size of the input Petri net. Rackoff's algorithm shows that **Coverability** is **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: these 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 *elemen-*

tary problems is defined as

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

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.

The rest of the section is the counterpart of Section 3.1.1 for arbitrary Petri nets. The rule of thumb is now:

All interesting questions about the behaviour of Petri nets are **EXPSPACE**-hard. More precisely, they require at least $2^{O(\sqrt{n})}$-space.

In particular, all the questions we asked about 1-safe Petri nets can be reformulated for Petri nets, and turn out to have at least this space complexity. As in the case of 1-safe Petri nets, this is a consequence of one single fundamental fact:

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.

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 *labelled 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. In this paper we assume that the initial values are always 0. The semantics of programs is that 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. Observe in particular that counter programs are deterministic.

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 make use of 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 simulating the computation of A on empty tape; 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.

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 can and will look at the programming notation as a compact description language for Petri nets.

A *net program* is rather 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**	

Syntactical correctness is defined as for counter programs. We also assume that programs are well-structured. Loosely speaking, a program is *well-structured* if it can be decomposed into a main program that only calls first-level subroutines, which in turn only call second-level subroutines, etc., and the jump commands in a subroutine can only have commands of the same subroutine as destinations.[6] We do not formally define well-structured programs, it suffices to know that all the programs of this section are well-structured.

We sketch a (Place/Transition) Petri net semantics of well-structured net programs. The Petri net corresponding to a program has a place for each label, a place for each variable, a distinguished *halt* place, and some additional places used to store the calling address of a subroutine call. There is a transition for each assignment and for each unconditional jump, and two transitions for each nondeterministic jump, as shown in Figure 3.5. We illustrate the semantics of the subroutine command by means of the program

[6] Here we consider the main program as a zero-level subroutine, i.e., jump commands in the main program can only have commands of the main program as destinations.

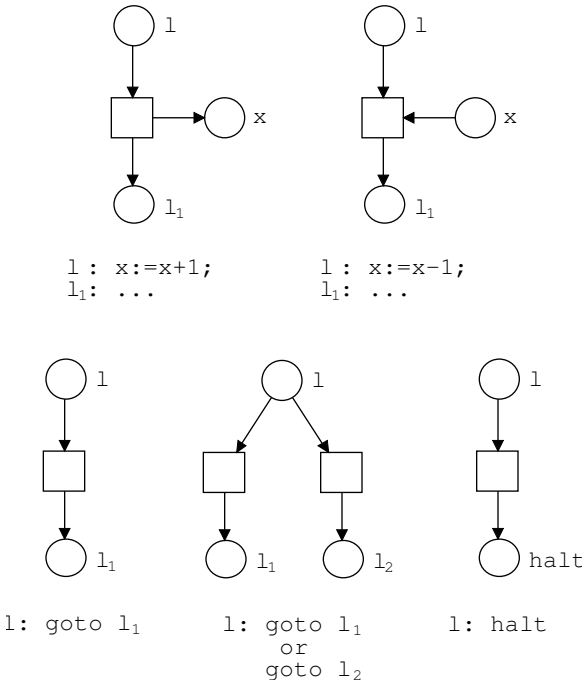


Figure 3.5: Net semantics of assignments and jumps

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

The corresponding Petri net is shown in Figure 3.6. Observe that the places *1_calls_4* and *2_calls_4* are used to remember the address from which the subroutine was called.

Clearly, 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 $O(n^2)$. Unfortunately, the construction of $N(C)$ requires quite a bit of low-level programming. But the reward is worth the hacking effort.

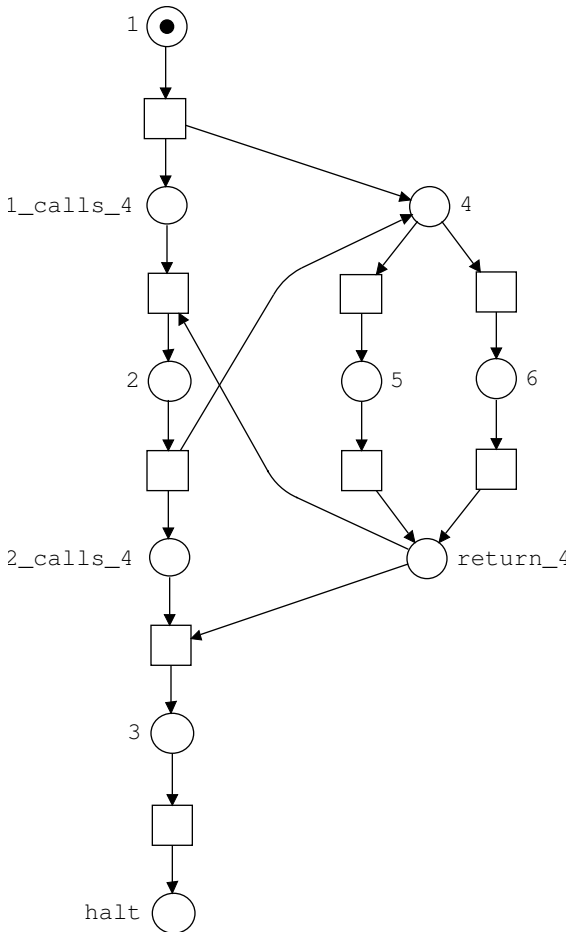


Figure 3.6: Net semantics of subroutines

The notion of simulation is not as strong as in the case of 1-safe Petri nets. In particular, net programs are nondeterministic, while counter programs are deterministic. A net program N simulates a counter program C if the following property holds: C halts (executes the command **halt**) if and only if *some* computation of N halts (other computations may fail).

Each variable x of N (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 $N_{init}(C)$.[7] The rest of $N(C)$, called $N_{sim}(C)$, simulates C and takes care of keeping the invariant $\overline{x} = 2^{2^n} - x$.

We design $N_{sim}(C)$ first. This program 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

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

to replace a conditional jump of the form

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

The specification of 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 $\text{Test}'_n(x, \texttt{ZERO}, \texttt{NONZERO})$ with the same specification but a *side-effect*: after an execution leading to ZERO, the values of x and $\overline{x}$ are swapped.[8] Once Test'_n has been designed, we can take:

[7] Recall that by definition all variables of N have initial value 0. Therefore, if we need $\overline{x} = 2^{2^n}$ initially, then we have to design preprocessing code for it.

[8] Executions leading to NONZERO must still be free of side-effects.

Program Test$_n$(x, ZERO, NONZERO):

Test$'_n$(x, continue, NONZERO);
continue: Test$'_n$($\overline{x}$, ZERO, NONZERO)

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

The key to the design of 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. Decreasing $\overline{x}$ by 2^{2^n} is the difficult part. We leave it for a routine Dec$_n$ to be designed, which must satisfy the following specification:

> If the initial value of s is smaller than 2^{2^n}, then every execution of Dec$_n$ 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 $s := s - 2^{2^n}; \overline{s} := \overline{s} + 2^{2^n}$; in particular, there are no side-effects. All other executions fail.

Test$'_n$ proceeds by transferring the value of x to a special variable s_n, and then calling the routine Dec$_n$, which decreases s_n by 2^{2^n}. In this way we need one single routine Dec$_n$, instead of one for each different variable to be decreased, which leads to a smaller net program.

Program Test$'_n$(x, ZERO, 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** $\mathtt{dec_n}$; **goto** ZERO
** the routine called at $\mathtt{dec_n}$ is Dec$_n(s_n)$ **

It is easy to see that Test$'_n$ meets its specification: if $x > 0$, then we may choose the nonzero branch and reach NONZERO. 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 the subroutine Dec$_n$, and then the program moves to ZERO. Moreover, if $x = 0$ then no execution reaches the NONZERO branch, because the program fails at $x := x-1$. If $x > 0$, then no execution reaches the ZERO branch, because s_n cannot reach the value 2^{2^n}, and so Dec$_n$ fails.

The next step is to design Dec$_n$. We proceed by induction on n, starting with Dec$_0$. This is easy, because it suffices to decrease s by $2^{2^0} = 2$. So we can take

Subroutine 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. The definition of Dec$_{i+1}$ contains two copies of a program Test$'_i$, called with different parameters. We define this program by substituting i for n everywhere in Test$'_n$. Test$'_i$ calls the routine Dec$_i$ at the address dec_i. Notice that this is correct, because we are assuming that the routine Dec$_i$ has already been defined.

The key to the design of Dec$_{i+1}$ is that 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, such that the

body of the inner loop decreases s by 1. The loop variables have initial values 2^{2^i}, and termination of the loops is detected by testing the loop variables for 0. This is done by the Test$'_i$ programs.

Subroutine 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`: $z_i := z_i - 1$; $\overline{z}_i := \overline{z}_i + 1$;
$s := s - 1$; $\overline{s} := \overline{s} + 1$;
Test$'_i$(z_i, `inner_exit` , `inner_loop`);
`inner_exit`: Test$'_i$(y_i, `outer_exit`, `outer_loop`);
`outer_exit`: **return**

Observe also that both instances of Test$'_i$ call the same routine at the same label.

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. But 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 $\overline{y}_i$.

This concludes the description of the program Test$_n$, and so the description of the program $N_{sim}(C)$. It remains to design $N_{init}(C)$. Let us first make a list of the initialisations 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.

Now, the specification of $N_{init}(C)$ is simple

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

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

These programs are defined by induction on i, and are very similar to the family of Dec_i programs. We start with Inc_0:

Program $\text{Inc}_0(v_1, \ldots, v_m)$:

$v_1 := v_1 + 1; v_1 := v_1 + 1;$
$\ldots$
$v_m := v_m + 1; v_m := v_m + 1$

and now give the inductive definition of Inc_{i+1}:

Program $\text{Inc}_{i+1}(v_1, \ldots, v_m)$:

** 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`: $z_i := z_i - 1; \overline{z}_i := \overline{z}_i + 1;$
$v_1 := v_1 + 1;$
$\ldots$
$v_m := v_m + 1;$
$\text{Test}'_i(z_i,$ `inner_exit` , `inner_loop`$)$;
`inner_exit`: $\text{Test}'_i(y_i,$ **`outer_exit`**, `outer_loop`$)$;
`outer_exit`: ...

It is easy to see that these programs satisfy their specifications. Now, let us consider $\text{N}_{init}(C)$. Apparently, we face a problem: in order to initialise the variables $v_1, \ldots, v_m$ to $2^{2^{i+1}}$ the variables y_i and z_i must have already been initialised to 2^{2^i}! Fortunately, we find a solution by just carrying out the initialisations in the right order:

Program $\mathrm{N}_{init}(C)$:

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

This concludes the description of $N(C)$, and it is now time to analyse its size. Consider $N_{sim}(C)$ first. It contains two assignments for each assignment of C, an unconditional jump for each unconditional jump in C, and a different instance of Test_k for each conditional jump. Moreover, it contains (one single instance of) the routines Dec_n, Dec_{n-1}, ..., Dec_0 (notice that Test_n calls Dec_n, which calls Dec_{n-1}, etc.). Both 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 Inc_i $1 \leq i \leq n$. The programs $\mathrm{Inc}_1, \ldots, \mathrm{Inc}_{n-1}$ have constant size, since they initialise a constant number of variables. The number of commands of Inc_n is $O(n)$, since it initialises $O(n)$ variables.

So we have proved that $N(C)$ contains $O(n)$ commands. It follows that its corresponding Petri net has size $O(n^2)$, which concludes our presentation of Lipton's result.

Chapter 4

Semi-decision procedures

4.1 Linear systems of equations and linear programming

In the next two sections we will construct linear systems of equations with integer or rational coefficients that provide partial information about our analysis problems. 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 $A = (a_{ij})_{i=1,\ldots n, j=1,\ldots,m}$ and $b = (b_j)_{j=1,\ldots,m}$ 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 (though not by means of Gauss elimination!).

- 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. [1]
- an integer solution is NP-complete.
- a nonnegative integer solution is NP-complete.

Given a linear objective function $f(X) = c_1 x_1 + \ldots c_m$ 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

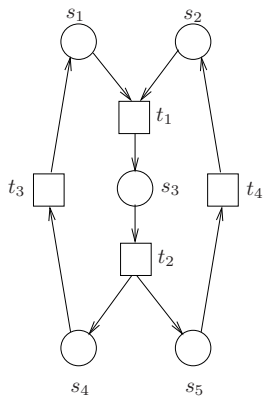
Definition 4.2.1 (Incidence matrix)
Let $N = (S, T, F)$ be a net. The *incidence matrix* $\mathbf{N} : (S \times T) \to \{-1, 0, 1\}$ is given by

$$\mathbf{N}(s,t) = \begin{cases} 0 & \text{if } (s,t) \notin F \text{ and } (t,s) \notin F \text{ or} \\ & \quad (s,t) \in F \text{ and } (t,s) \in F \\ -1 & \text{if } (s,t) \in F \text{ and } (t,s) \notin F \\ 1 & \text{if } (s,t) \notin F \text{ and } (t,s) \in F \end{cases}$$

The column $\mathbf{N}(-,t)$ is denoted by $\mathbf{t}$, and the row $\mathbf{N}(s,-)$ by $\mathbf{s}$.

Example 4.2.2

[1] In practice we often use the Simplex algorithm, which has exponential worst-case complexity, but is very efficient for most instances.



	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* $\boldsymbol{\sigma} : T \to \mathbb{N}$ *von* σ is defined by

$$\boldsymbol{\sigma}(t) = \text{ number of occurrences of } t \text{ in } \sigma$$

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

$$\begin{array}{lll} M' & = L + \mathbf{t} & \text{(Definition of } \mathbf{t}) \\ & = L + \mathbf{N} \cdot \boldsymbol{t} & \text{(Definition of } \boldsymbol{t}) \\ & = M + \mathbf{N} \cdot \boldsymbol{\tau} + \mathbf{N} \cdot \boldsymbol{t} & \text{(Induction hyp.)} \\ & = M + \mathbf{N} \cdot (\boldsymbol{\tau} + \boldsymbol{t}) & \\ & = M + \mathbf{N} \cdot \boldsymbol{\tau t} & \text{(Definition of Parikh-vector)} \\ & = M + \mathbf{N} \cdot \boldsymbol{\sigma} & (\sigma = \tau t) \end{array}$$

□

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 (The 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 4.2.4 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 4.2.4. □

Proposition 4.2.9 (A sufficient condition for deadlock-freedom)

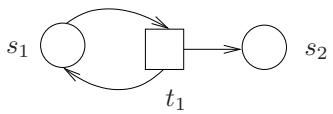
Let (N, M_0) be a 1-bounded Petri net where $N = (S, T, F)$. If the following system of inequations has no solution then (N, M_0) is deadlock-free.

$$M = M_0 + \mathbf{N} \cdot X$$
$$\sum_{s \in \cdot t} M(s) < |^\bullet t| \textit{ for every transition } t.$$

Proof. We show: if there is a reachable dead marking M, then M is a solution of the system. By Lemma 4.2.4 and the reachability of M there is a vector X satisfying $M = M_0 + \mathbf{N} \cdot X$. Since (N, M_0) is 1-bounded, we have $M(s) \leq 1$ for every place s. Let t be an arbitrary transiton. Since M does not enable t, we have $M(s) = 0$ for at least one place $s \in {}^\bullet t$. Since M does not enable any transition, we get $\sum_{s \in \cdot t} M(s) < |^\bullet t|$. □

Remark 4.2.10 *The converses of these propositions do not hold (that is why they are semi-algorithms!). Counterexamples are:*

- *To Proposition 4.2.7:*



	t_1
s_1	0
s_2	1

(N, M_0) *ist bounded but*

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

for every n (that is, the Marking Equation has a solution for every marking of the form $(0, n)$).

- *To Proposition 4.2.8:*

  Peterson's algorithm: the marking $(p_4, q_4, m_1 = true, m_2 = true, hold = 1)$ ist not reachable, but the Marking Equation has a solution (Exercise: find a smaller example).

- *To Proposition 4.2.9:*

  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 Proposition 4.2.9 (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 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

$$\begin{aligned} I \cdot M &= I \cdot M_0 + I \cdot \mathbf{N} \cdot \boldsymbol{\sigma} && \text{(Marking Equation)} \\ &= I \cdot M_0 && (I \cdot \mathbf{N} = 0) \end{aligned}$$

□

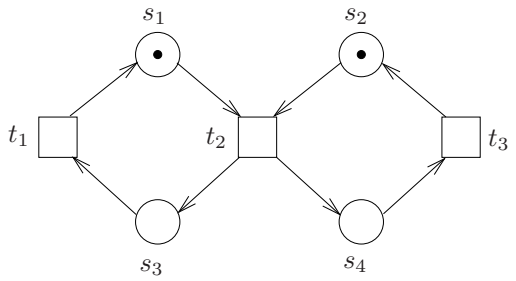


Figure 4.1

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 net of Figure 4.1*

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 an immediate consequence of the definition of S-invariants:

Proposition 4.3.4 *The S-invariants of a net form a vector space over the real numbers.*

This definition of S-invariant is very suitable for machines, but not for humans, who can only solve very small

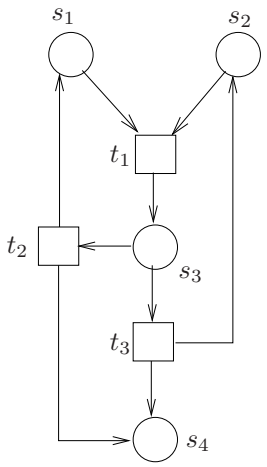


Figure 4.2

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. $\forall t \in T : \sum_{s \in {}^\bullet t} I(s) = \sum_{s \in t^\bullet} I(s)$.*

Proof. $I \cdot \mathbf{N} = 0$ is equivalent to $I \cdot \mathbf{t} = 0$ for every transition t. So for every transition t 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 net of Figure 4.2. The condition of Proposition 4.3.5 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 give sufficient conditions for boundedness and necessary conditions for liveness and 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 $I \neq 0$, and *positive* if $I > 0$ (that is, if $I(s) > 0$ for every $s \in S$). The *support* of an S-invariant is the set $\langle I \rangle = \{s \in S \mid I(s) > 0\}$.

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 $M(s) \leq \frac{I \cdot M_0}{I(s)}$. □

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. □

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. ($\Rightarrow$): Since $M \sim L$, we have $I \cdot (L - M) = 0$ for every S-invariant I.

We now recall a well-known theorem of linear algebra. Given a $n \times m$ matrix A, let $U = \{u \in \mathbb{N}^n \mid u \cdot A = 0\}$, and let $V = \{v \in \mathbb{N}^m \mid u \cdot v = 0 \text{ for every } u \in U\}$. Then both U and V 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, and 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 $(L - M)$ is one of these vectors, $(L - M)$ is a linear combination over $\mathbb{Q}$ of the columns of $\mathbf{N}$, and so the equation $\mathbf{N} \cdot X = (L - M)$ has a rational solution.

($\Leftarrow$) : Let I be an S-invariant of N. Since $I \cdot \mathbf{N} = 0$ we have $I \cdot L = I \cdot M + I \cdot \mathbf{N} \cdot X = I \cdot M$. □

We also have the following consequences:

$$
\begin{array}{c}
M \text{ is reachable from } L \\
\not\Uparrow \qquad \Downarrow \\
M = L + \mathbf{N} \cdot X \text{ has a solution } X \in \mathbb{N}^{|T|} \\
\not\Uparrow \qquad \Downarrow \\
M = L + \mathbf{N} \cdot X \text{ has a solution } X \in \mathbb{Q}^{|T|} \\
\Updownarrow \\
M \sim L
\end{array}
$$

4.3.2 T-invariants

Definition 4.3.13 (T-invariants)
Let $N = (S, T, F)$ be a net. A vector $J : T \to \mathbb{Q}$ is a T-invariant of N if $\mathbf{N} \cdot J = 0$.

Proposition 4.3.14 *J is a T-invariant of $N = (S, T, F)$ iff $\forall s \in S : \sum_{t \in {}^\bullet s} J(t) = \sum_{t \in s^\bullet} J(t)$.*

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 $M = M + \mathbf{N} \cdot \boldsymbol{\sigma}$ and so $\mathbf{N} \cdot \boldsymbol{\sigma} = 0$. □

Example 4.3.16 *We compute the T-invariants of the net of Figure 4.1 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-invarints is (x, x, x), where $x \in \mathbb{Q}$.

Using T-invariants we obtain a necessary condition for well-formedness of a net:

Theorem 4.3.17 *[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

$$M_i \xrightarrow{\sigma_{i+1}\dots\sigma_j} M_i$$

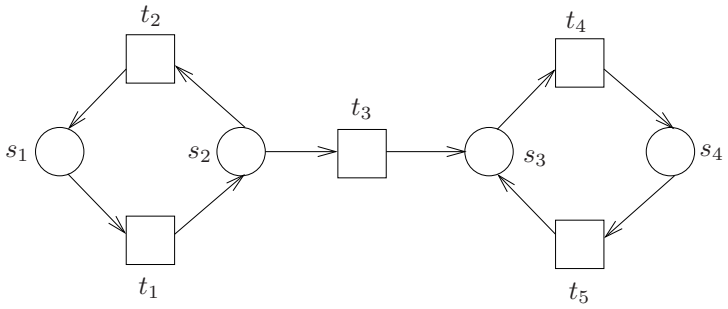


Figure 4.3

and so $J = \boldsymbol{\sigma}_{i+1} + \ldots + \boldsymbol{\sigma}_j$ is a T-invariant of N. Further, J is positive because every transition occurs at least once in $\sigma_{i+1} \ldots \sigma_j$. □

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 $R \neq \emptyset$.

$\{s_1, s_2\}$ is a siphon of the net of Figure 4.3 because

$$ {}^\bullet\{s_1, s_2\} = {}^\bullet s_1 \cup {}^\bullet s_2 = \{t_2\} \cup \{t_1\} = \{t_1, t_2\} $$

und

$$ \{s_1, s_2\}^\bullet = s_1^\bullet \cup s_2^\bullet = \{t_1\} \cup \{t_2, t_3\} = \{t_1, t_2, t_3\} $$

Proposition 4.4.2 *[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 4.4.3 *[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. For this we first observe that, if R_1 and R_2 are siphons of N, then so is $R_1 \cup R_2$ (exercise). 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 holds if and only if $M(Q_0) = 0$.

- If the condition holds, then, since $M_0(Q_0) = 0$ by definition, we get $M(Q_0) = 0$.
- If 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 following algorithm, which computes the largest siphon Q contained in a given set R of places—it suffices then to choose R as the set of places unmarked at M_0.

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 that after termination Q is the largest siphon contained in R.

Proposition 4.4.4 *[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 4.4.3 the initial marking M_0 also marks R. $\square$
Again, the condition can be checked with the help of the algorithm above: the condition holds if and only if $Q_0 = \emptyset$. We now look at deadlock-freedom. We can obtain a sufficient condition for it, but not one that is easy to check.

Proposition 4.4.5 *If M is a dead marking of N, then the set of places unmarked at M is a siphon of N.*

Proof. Let $R = \{s \mid M(s) = 0\}$. For every transition t there is a place $s \in {}^\bullet t$ such that $M(s) = 0$ (otherwise t would be enabled). So $R^\bullet$ contains all transitions of N, and therefore ${}^\bullet R \subseteq R^\bullet$. $\square$

Corollary 4.4.6 *[A sufficient condition for deadlock-freedom] Let (N, M_0) be a Petri net. If every reachable marking marks all siphons of N, then (N, M_0) is deadlock-free.*

4.4.2 Traps

Definition 4.4.7 (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 $R \neq \emptyset$.

$\{s_3, s_4\}$ is a trap of the net of Figure 4.3.

Proposition 4.4.8 *[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. $\square$

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.

Corollary 4.4.9 *[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 $R_1 \cup R_2$ (exercise). So there exists a unique largest trap Q_0 marked at M_0 (more precisely, $R \subseteq Q_0$ for every trap R such that $M(R) > 0$). It is easy to see that the condition holds if and only if $M_0(Q_0) > 0$ (exercise).

To compute the largest trap unmarked at M, we can transform the algorithm that computes the largest siphon contained in a given set of places into an algorithm for computing the largest trap (exercise).

Recall that checking 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 4.4.10 *[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 4.4.6 and Proposition 4.4.8. □

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).

We finally show how to combine S-invariants and traps to prove that Peterson's algorithm satisfies the mutual exclusion property. For the Petri net model of Figure 2.8 mutual exclusion means that no reachable marking M satisfies $M(p_4) \geq 1 \wedge M(q_4) \geq 1$. We first compute three S-invariants:

(1) $M(hold = 1) + M(hold = 2) = 1$

(2) $M(p_2) + M(p_3) + M(p_4) + M(m_1 = f) = 1$

(3) $M(q_2) + M(q_3) + M(q_4) + M(m_1 = f) = 1$

and two constraints derived from traps:

(4) $M(m_1 = f)+M(p_2)+M(hold = 1)+M(q_3) > 0$

(5) $M(m_2 = f)+M(q_2)+M(hold = 2)+M(p_3) > 0$

Assume now $M(p_4) \geq 1 \wedge M(q_4) \geq 1$ holds. We have:

$$M(p_4) \geq 1 \wedge M(q_4) \geq 1$$

$$\Rightarrow \{(2), (3)\}$$

$$M(p_2) + M(p_3) + M(m_1 = f) = 0 \quad \wedge \quad M(q_2) + M(q_3) + M(m_2 = f) = 0$$

$$\Rightarrow \{(1)\}$$

$$\begin{array}{lcl} M(m_1 = f) + M(p_2)+ & & M(m_2 = f) + M(q_2)+ \\ M(hold = 1) + M(q_3) = 0 & \vee & M(hold = 2) + M(p_3) = 0 \end{array}$$

Contradicts (4) Contradicts (5)

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. The sections have a similar structure. After the definition of the class, we introduce three theorems: the Liveness, Boundedness, and Reachability Theorem. 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 proof 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.

At the end of the section we present a final theorem, the Shortest Path Theorem, which gives an upper bound for the length of the shortest firing sequence leading to a given reachable marking.

The reader may ask why boundedness only for live Petri nets, and why reachability only for live and bounded Petri nets. A first reason is that, in many application areas, a Petri net model of a correct system must typically

be live *and* bounded, and so, when one of these properties fails, it does not make much sense to further analyze the model. The second reason is that, interestingly, the general characterization of the bounded systems or the reachable markings is more complicated and less elegant than the corresponding characterization for live or live and bounded 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 some detail for T-systems, and omit them 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 if N is a S-net.

Proposition 5.1.2 (Fundamental property of S-systems)

Let (N, M_0) be a S-system with $N = (S, T, F)$. Then $M_0(S) = M(S)$ for every reachable marking M.

Proof. Every transition consumes one token and produces one token. □

Theorem 5.1.3 *[Liveness Theorem] A S-system (N, M_0) where $N = (S, T, F)$ is live iff N is strongly connected and $M_0(S) > 0$.*

Proof. (Sketch.)
($\Rightarrow$): If N is not strongly connected, then there is an arc (s, t) such that N has no path from t to s. For every marked place s' such that there is a path from s' to s, we fire the transitions of the path to bring the tokens in s' to s, and then fire t to empty s. We have then reached a marking from which no tokens can "travel" back to s, and so a marking from which t cannot occur again. So (N, M_0) 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. □

Theorem 5.1.4 *[Boundedness Theorem] A live S-system (N, M_0) where $N = (S, T, F)$ 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] Let (N, M_0) be a live S-system 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 5.1.3. So we are free to distribute the tokens of M_0 in an arbitrary way, and reach any marking M, as long as $M(S) = M_0(S)$. □

Proposition 5.1.6 *[S-invariants of S-nets] Let $N = (S, T, F)$ be a connected 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

I is a S-invariant
$\Leftrightarrow$ {Proposition 4.3.5 (alternative definition of S-invariant)}
$\forall t \in T : I(s_t) = I(s'_t)$
$\Leftrightarrow$ {N is connected}
$\forall s_1, s_2 \in S : I(s_1) = I(s_2)$
$\Leftrightarrow$ { }
$\exists x \in \mathbb{Q} \forall s \in S : I(s) = x.$

□

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: Let γ be a circuit of a net N and let M be a marking of N. We denote by $M(\gamma)$ the number if tokens of γ under M, that is, $M(\gamma) = \sum_{s \in \gamma} M(s)$.

Proposition 5.2.2 (Fundamental property of T-systems) *Let γ be a circuit of a T-systems (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. The token count does not change. □

5.2.1 Liveness

Theorem 5.2.3 *[Liveness Theorem] 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 5.2.2 we have $M(\gamma) = 0$ for every reachable marking M. So no transition of γ can ever occur.

($\Leftarrow$) 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 satisfying the following property: there is a path from s to t that contains no place marked at M. We proceed by induction on $|S_M|$. **Basis:** $|S_M| = 0$. Then $M(s) > 0$ for every place $s \in {}^\bullet t$, and 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 path Π such that:

(1) Π leads to t;

(2) M marks no place of Π;

(3) Π has maximal length (that is, no path longer than Π satisfies (1) and (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. Moreover, we have $u \neq t$ because M does not enable t. Let $M \xrightarrow{u} M'$. We show that $S_{M'} \subset S_M$, and so that $|S_{M'}| < |S_M|$.

1. $S_{M'} \subseteq S_M$
   Let $s \in S_{M'}$. We show $s \in S_M$. There is a path $\Pi' = s \ldots t$ containing no place marked at M'. Assume Π' contains a place r marked at M. Since $M'(r) = 0$ and $M \xrightarrow{u} M'$ we have $u \in r^\bullet$ and so $\{u\} = r^\bullet$. So u is the successor of r in Π'. Since $u \neq t$, M' marks the successor of u in Π', contradicting the definition of Π'.

2. $S_{M'} \neq S_M$. Let s be the successor of u in Π. Then $s \in S_M$ but $s \notin S_{M'}$, because $M'(s) > 0$.

By induction hypothesis there is a firing sequence $M' \xrightarrow{\sigma} M''$ such that M'' enables t. It follows $M \xrightarrow{u} M' \xrightarrow{\sigma} M''$, and so M'' is a marking reachable from M that enables t. □

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. ($\Leftarrow$) Follows from the fundamental property of T-systems (Proposition 5.2.2).

($\Rightarrow$) 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. Otherwise there would be a firing sequence $L \xrightarrow{\sigma} L'$ such that $L'(s) > 0$, and by the Monotonicity Lemma we would have $M \xrightarrow{\sigma} M'$ for some marking M' satisfying $M'(s) = L'(s) + M(s) > M(s)$, contradicting the maximality of $M(s)$. By the Liveness Theorem some circuit γ is unmarked at L but marked at M. Since L and M only differ in the place s, the circuit γ contains s. Further, 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

We need to have a closer look at the T-invariants of T-systems.

Proposition 5.2.6 *[T-invariants of T-nets] Let $N = (S, T, F)$ be a connected 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 5.1.6. □

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. ($\Rightarrow$) Proposition 4.3.11
($\Leftarrow$) By Theorem 4.3.12 there is a rational vector X such that

$$M = M_0 + \mathbf{N}.X \tag{5.1}$$

The vector $J = (1, 1, \ldots, 1)$ is a T-invariant of N (Proposition 5.2.6). So 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 show:

(1) There is a vector $Y : T \to \mathbb{N}$ such that $M = M_0 + \mathbf{N} \cdot Y$. Let Y be the vector with $Y(t) = \lceil X(t) \rceil$ for every transition t ($\lceil x \rceil$ denotes the smallest integer larger than or equal to x). By (5.1) we have

$$M(s) = M_0(s) + X(t_1) - X(t_2)$$

for every place s, where $\{t_1\} = {}^\bullet s$ and $\{t_2\} = s^\bullet$. Both $M(s)$ and $M_0(s)$ are integers. By the definition of Y we get

$$X(t_1) - X(t_2) = Y(t_1) - Y(t_2)$$

So $M(s) = M_0 + Y(t_1) - Y(t_2)$, which implies $M = M_0 + \mathbf{N} \cdot Y$.

(2) $M_0 \xrightarrow{*} M$
By induction over $|Y| = \sum_{t \in T} Y(t)$.
Basis: $|Y| = 0$. Then $Y = 0$ and $M = M_0$.
Step: $|Y| > 0$.
We show that M_0 enables some transition of $\langle Y \rangle$. Let

$$Sy = \{s \in {}^\bullet\langle Y \rangle \mid M_0(s) = 0\}$$

Let $s \in Sy$. By $M_0(s) = 0$ 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 *Sy* and transitions of $\langle Y \rangle$ (such a path exists, because otherwise N would contain a circuit unmarked at M_0). By (*), the first node of Π is a transition $t \in \langle Y \rangle$, and no place of ${}^\bullet t$ belongs to *Sy*. So M_0 marks every place of ${}^\bullet t$, that is, M_0 enables t.

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 have $M_1 \xrightarrow{*} M$. Since $M_0 \xrightarrow{t} M_1 \xrightarrow{*} M$, we get $M_0 \xrightarrow{*} M$.

□

5.2.4 Other properties

The theorems we have introduced have many interesting consequences. Her are two of them.

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) ⇒ (2) ⇒ (3) follow immediately from the definitions. We show (3) ⇒ (1).

Let $M_0 \xrightarrow{\sigma}$ be an infinite firing sequence. We claim that every transition of N occurs in σ. Since N is strongly

connected, (N, M_0) is bounded (Theorem 5.2.4). 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 indices i and j with $i < j$ such that $M_i = M_j$. Let σ_{ij} be the subsequence of σ containing the transitions between M_i and M_j. By the fundamental property of T-invariants (Proposition 4.3.15) $\boldsymbol{\sigma_{ij}}$ is a T-Invariant . By Proposition 5.2.6 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.

Since every transition of N occurs in σ, for every place and every circuit of N some marking reached during the execution of N marks the place or the circuit. By the fundamental property of T-systems, all circuits of N are marked at M_0. By the Liveness Theorem (Theorem 5.2.3), (N, M_0) is live. □

Theorem 5.2.9 *[Genrich's Theorem] Let N be a strongly connected T-net with at least one place and one transition. 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 (Corollary 5.2.5).

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 5.2.4, 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.

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). □

Finally we prove a result stating 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 $n(n-1)/2$ steps where n is the number of transitions of the T-system.

The result is proved with the help of two lemmas.

Definition 5.2.10 *Given a sequence σ of transitions, we denote by $\mathcal{A}(\sigma)$ the set of transitions occurring at least once in σ.*

Lemma 5.2.11 *Let (N, M_0) be a T-system and let $M_0 \xrightarrow{\sigma_1\,\sigma_2\,t}$ for some sequences $\sigma_1\sigma_2 \in T^*$, some $t \in T$ such that*

- *$t \notin \mathcal{A}(\sigma_1)$, and*
- *$\mathcal{A}(\sigma_2) \subseteq \mathcal{A}(\sigma_1)$.*

Then $M_0 \xrightarrow{\sigma_1\,t\,\sigma_2}$.

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 $M_0 \xrightarrow{\sigma_1\,\sigma_2'\,t\,u}$, and then the result follows by applying the induction hypothesis to $\sigma_1\,\sigma_2'\,t$.

Let $M_0 \xrightarrow{\sigma_1\,\sigma_2'} M_1 \xrightarrow{u} M_2 \xrightarrow{t}$. Consider two cases:

- $u^\bullet \cap {}^\bullet t = \emptyset$. Then t is already enabled at M_1, and we are done.
- $u^\bullet \cap {}^\bullet t \neq \emptyset$. 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 $\sigma_1\,\sigma_2$, while t occurs zero times. It follows $M_2(s) \geq 2$, and therefore $M_1(s) \geq 1$. Further, for every $s \in {}^\bullet t \setminus u^\bullet$ we have $M_1(s) = M_2(s) \geq 1$. So 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 $M_0 \xrightarrow{\sigma} M$. Then there exist sequences $\sigma_1\sigma_2$ such that*

(1) $M_0 \xrightarrow{\sigma_1\,\sigma_2} M$.

(2) no transition occurs more than once in σ_1,

(3) $\mathcal{A}(\sigma_2) \subseteq \mathcal{A}(\sigma_1)$, and

(4) if σ is nonempty then $\mathcal{A}(\sigma_2) \subset \mathcal{A}(\sigma_1)$.

Proof. We first prove that (1)-(3) 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 5.2.11 we have $M_0 \xrightarrow{\tau_1\, t\, \tau_2} M$, and we take $\sigma_1 = \tau_1\, t$ and $\sigma_2 = \tau_2$.

To prove (4), assume we have σ_1, σ_2 satisfying (1)-(3). We consider several cases.

- $\mathcal{A}(\sigma_1) = \emptyset$. Then, by (3), we have $\sigma_1 = \sigma_2 = \epsilon$, and (4) holds vacuously.
- $\mathcal{A}(\sigma_1) = T$. If $\mathcal{A}(\sigma_2) \subset \mathcal{A}(\sigma_1)$ then we are done. If $\mathcal{A}(\sigma_2) = \mathcal{A}(\sigma_1)$, then $\mathcal{A}(\sigma_2) = T$, and by (2) both σ_1 and σ_2 contains every transition exactly once. Since N is a T-system, we then have $M_0 \xrightarrow{\sigma_1} M_0 \xrightarrow{\sigma_2} M_0$. But then we can replace σ_2 by ϵ, and now the pair ϵ, σ_1 satisfies (1)-(4).
- $\emptyset \neq \mathcal{A}(\sigma_1) \neq T$. Since N is 1-bounded, by the Boundedness Theorem it is strongly connected. So there is a place s with input and output transitions t and u, respectively, such that $t \in \mathcal{A}(\sigma_1)$ and $u \notin \mathcal{A}(\sigma_1)$. By (3) we have $u \notin \mathcal{A}(\sigma_2)$. If $t \in \mathcal{A}(\sigma_2)$ then $M(s) \geq 2$, contradicting 1-boundedness. So $t \notin \mathcal{A}(\sigma_2)$, and so $\mathcal{A}(\sigma_2) \subset \mathcal{A}(\sigma_1)$.

□

Theorem 5.2.13 *[Shortest Sequence Theorem] Let (N, M_0) be a b-bounded T-system and let M be a reachable marking. Then there is an occurrence sequence $M_0 \xrightarrow{\sigma} M$ such that $|sigma| \leq b \cdot n(n-1)/2$, where n is the number of transitions of N.*

Proof. We only prove the case $b = 1$. The general case requires a slight generalization of Lemma 5.2.11 and 5.2.12.

By repeated application of Lemma 5.2.12 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
- $\mathcal{A}(\sigma_1) \subset \mathcal{A}(\sigma_2) \subset \cdots \subset \mathcal{A}(\sigma_n)$.

Then we have $|\sigma_i| \leq n - i + 1$ for every $1 \leq i \leq n$, and so $|\sigma| \leq \sum_{i=1}^{n} i = \frac{n(n-1)}{2}$. □

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 $s^\bullet \times {}^\bullet t \subseteq F$ for every $s \in S$ and $t \in T$ such that $(s, t) \in F$. A Petri net (N, M_0) is *free-choice* if N is a free-choice net..

This definition is very concise and moreover symmetric with respect to places and transitions. If the reader finds it cryptic, the following equivalent definitions may help.

Proposition 5.3.2 *[Alternative definitions of free-choice nets]*

(1) A net is free-choice if for every two transitions t_1, t_2:

$$(t_1 \neq t_2 \wedge {}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset) \Rightarrow {}^\bullet t_1 = {}^\bullet t_2$$

(2) A net is free-choice if for every two places s_1, s_2:

$$(s_1 \neq s_2 \wedge s_1^\bullet \cap s_2^\bullet \neq \emptyset) \Rightarrow s_1^\bullet = s_2^\bullet$$

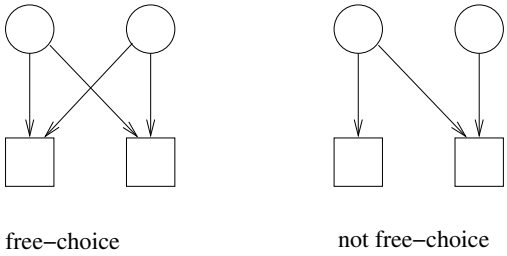


Figure 5.1

Proof. Exercise. □

Figure 5.1 illustrates these definitions.

Clearly, S- and T-systems are special cases of free-choice systems (see Figure 5.2).

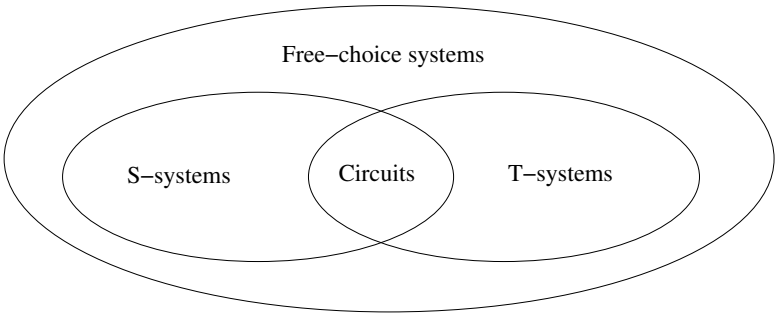


Figure 5.2: Net classes

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 Theorem, a much stronger result characterizing liveness.

Theorem 5.3.3 *[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 need the following definitions. Let M be a marking of N. A transition t is *dead at* M if it is not enabled at any marking of $[M\rangle$. 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 $M' \in [M\rangle$. Let L_M be the set of transitions live at M. Notice that a transition may be neither live nor dead at a marking. We have

- If $t \in L_M$ and $M' \in [M\rangle$, then $t \in L_{M'}$, that is, live transitions stay live.
- If $t \in D_M$ and $M' \in [M\rangle$, then $t \in L_{M'}$, that is, dead transitions stay dead.
- If $t \notin L_M \cup D_M$ then there is a marking M' reachable from M such that $t \in D_{M'}$. That is, transitions that are neither live nor dead may die.

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 the definitions above, 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 $D_M \neq \emptyset$.

We 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. Therefore, if all transitions of ${}^\bullet S_t$ are live at M then we can reach a marking that marks all of them. But such a marking enables t, contradicting that t is dead at M. So at least one place $s_t \in {}^\bullet t$ is dead 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

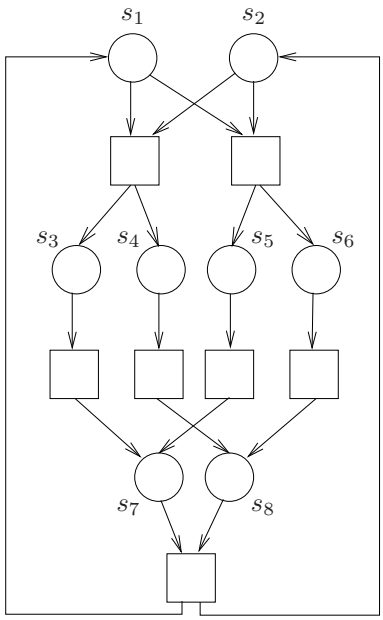


Figure 5.3: A free-choice system

contain any trap marked at M_0. □

A siphon is *minimal* if it does not properly contain any proper siphon. Clearly, the Liveness Theorem still holds if we replace "siphon" by "minimal siphon". The net of Figure 5.3 has four minimal siphons: $R_1 = \{s_1, s_3, s_5, s_7\}$, $R_2 = \{s_2, s_4, s_6, s_8\}$, $R_3 = \{s_2, s_3, s_5, s_7\}$ and $R_4 = \{s_1, s_4, s_6, s_8\}$. R_1, R_2, R_3 and R_4 are also traps, and so, in particular, they contain traps. By the Liveness Theorem, every marking that marks R_1, R_2, R_3 and R_4 is live.

We now proceed to prove the second part of the theorem. We have to 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 $D := R \setminus Q$. Loosely speaking, 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.

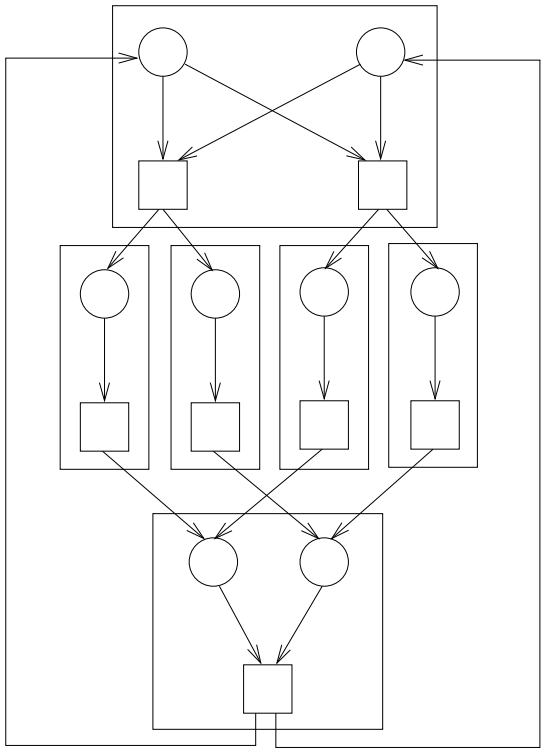


Figure 5.4: Clusters of the net of Figure 5.3

We need the notion of a cluster.

Definition 5.3.4 (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 denote $[x]$ the cluster of the node $x \in S \cup T$.

It follows from the definition that every node of a net belongs to exactly one cluster, that is, the set of clusters is a partition of $S \cup T$.

Figure 5.4 shows the clusters of the net of Figure 5.3.

The following proposition is easy to prove:

Proposition 5.3.5 *Let (N, M_0) be a free-choice system with $N = (S, T, F)$, and let c be a cluster of c.*

(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 define allocations.

Definition 5.3.6 (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 allocated transition 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. Notice that this part does not require the free-choice property.

Lemma 5.3.7 *Let N be a net, let R be a set of places of N, and let Q be the maximal trap included in Q, and let $D = R \setminus Q$. 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 $|R|$. If $|R| = 0$ then $C = \emptyset$ and we take the empty allocation. If $|R| > 0$ and 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 have to show that α is circuit-free and $\alpha(C) \cap {}^\bullet Q' = \emptyset$. We first prove the following facts, which we leave as an exercise:

(i) Q is the maximal trap included in R'.

(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.)

To show that 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-fee. So γ contains transition 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.

To prove $\alpha(C) \cap {}^\bullet Q' = \emptyset$ we first observe that $\alpha(C) \cap {}^\bullet Q' \subseteq (\alpha(C') \cup \{t\}) \cap {}^\bullet Q'$, which is equal to $\{t\} \cap {}^\bullet Q'$ by induction hypothesis, and equal to $\emptyset$ because $t \notin {}^\bullet R$ and ${}^\bullet Q \subseteq {}^\bullet R$. □

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 5.3.8 *[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 σ as their concatenation.

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 5.3.9 *[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 F be a proper siphon of N, and let Q be the maximal trap included in Q. We prove $M_0(Q) > 0$.

Since (N, M_0) is live, we have $M_0(R) > 0$ by Proposition 4.4.4. Let $D = R \setminus Q$. If $D^\bullet = \emptyset$ then D is a trap and so $D \subseteq Q$, but then $D = \emptyset$ and we are done.

If $D^\bullet \neq \emptyset$ then let $C = \{[t] \mid t \in D^\bullet\}$. By Lemma 5.3.7 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 5.3.8. 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.

□

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 5.3.10 *[Complexity]*
The problem

Given: A free-choice system (N, M_0)
Decide: Is (N, M_0) *not live?*

is NP-complete.

Proof. Membership in NP follows from Commoner's theorem: guess a siphon of N, compute in polynomial time the maximal trap contained in R, and check that it is unmarked at M_0.

The proof of NP-hardness is by reduction from SAT, the satisfiability problem for boolean formulas. The reduction is illustrated in Figure 5.5, which shows the free-choice system for the formula

$$\Phi = (x_1 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3})$$

□

5.3.2 Boundedness

Definition 5.3.11 (S-component) Let $N = (S, T, F)$ be a net. A subnet $N' = (S', T', F')$ of N is an *S-component* of N if

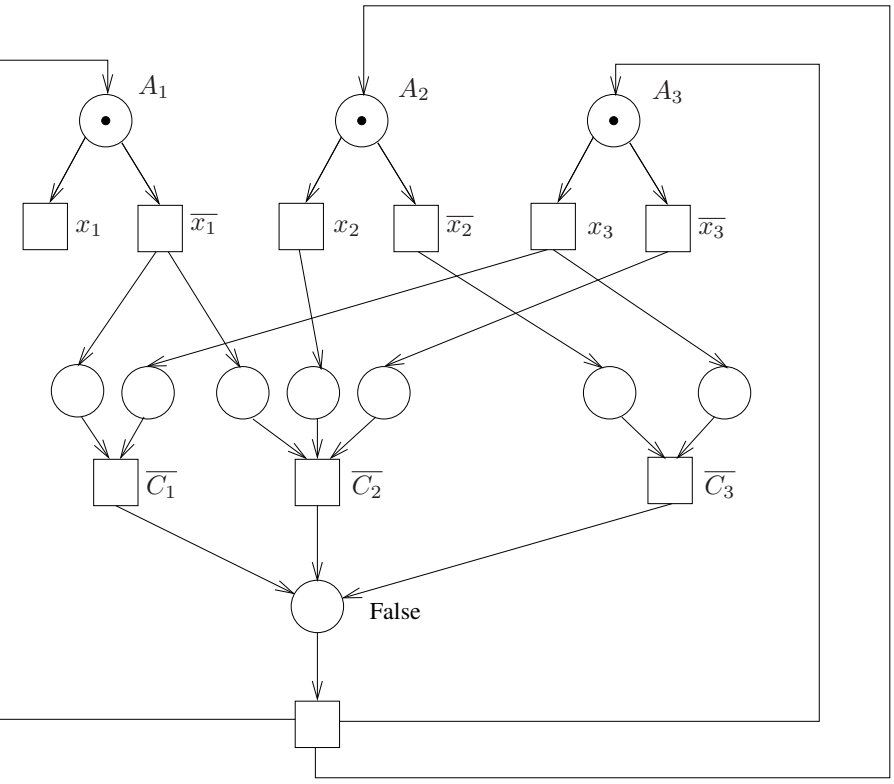


Figure 5.5: Free-choice system for the formula Φ

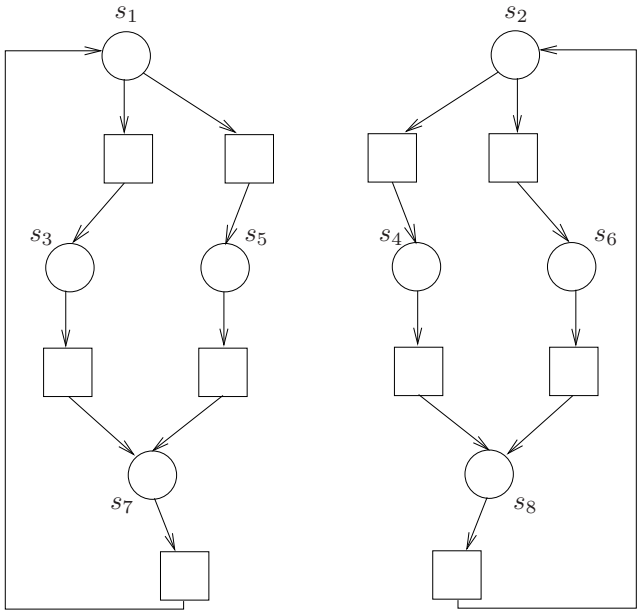


Figure 5.6: S-components of the net of Figure 5.3

1. N' is a strongly connected S-net, and
2. $T' = {}^\bullet S' \cup S'^\bullet$ (where $s^\bullet = \{t \in T \mid (t,s) \in F\}$, and analogously for ${}^\bullet s$).

Figure 5.6 shows two S-components of the net of Figure 5.3.

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 5.3.12 *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 5.3.13 *[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 5.3.14 *[Place bounds]*
Let (N, M_0) be a live and bounded free-choice system and let s be a place of N. We have

$$max\{M(s) \mid M_0 \xrightarrow{*} M\} =$$
$$min\{M_0(S') \mid S' \text{ is the set of places of a S-component of } N\}$$

Proof. Analogous to the Boundedness Theorem for T-systems. □

Theorem 5.3.10 shows that there is no polynomial algorithm for **Liveness** (unless $P = NP$). Now we ask ourselves what is the complexity of deciding if a free-choice system is simultaneously live and bounded. We can of course first use the decision procedure for liveness, and then, if the net is live, check the condition of the Boundedness Theorem. But there are more efficient algorithms.[1]. 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 immediately from the next theorem:

Theorem 5.3.15 *[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 siphon of N is marked under M_0.*

[1] 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.

Proof. Omitted. □

Conditions (1) and (2) can be checked using linear programming, condition (3) using well-known algorithms of linear algebra, and condition (4) with the algorithm of Section 4.4.1.

5.3.3 Reachability

The reachability problem is NP-hard for live and bounded free-choice nets.

Theorem 5.3.16 Reachability *is NP-hard for live and bounded free-choice nets.*

Proof. We reduce SAT to the following problem:

> Given: A live and bounded free-choice system (N, M_0) where $N = (S, T, F)$, two disjoint sets $T_{=1}, T_{\geq 1} \subseteq T$, 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?

Figure 5.7 shows the net N, the markings M_0 and M, and the sets $T_{=1}, T_{\geq 1}$ for the formula $x_1 \wedge (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2)$. the formula has three clauses C_1, C_2, C_3. The black tokens correspond to M_0, and the 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) and each transition of $T_{\geq 1}$ at least one (so that at least one of the literals of each clause is true under the assignment).

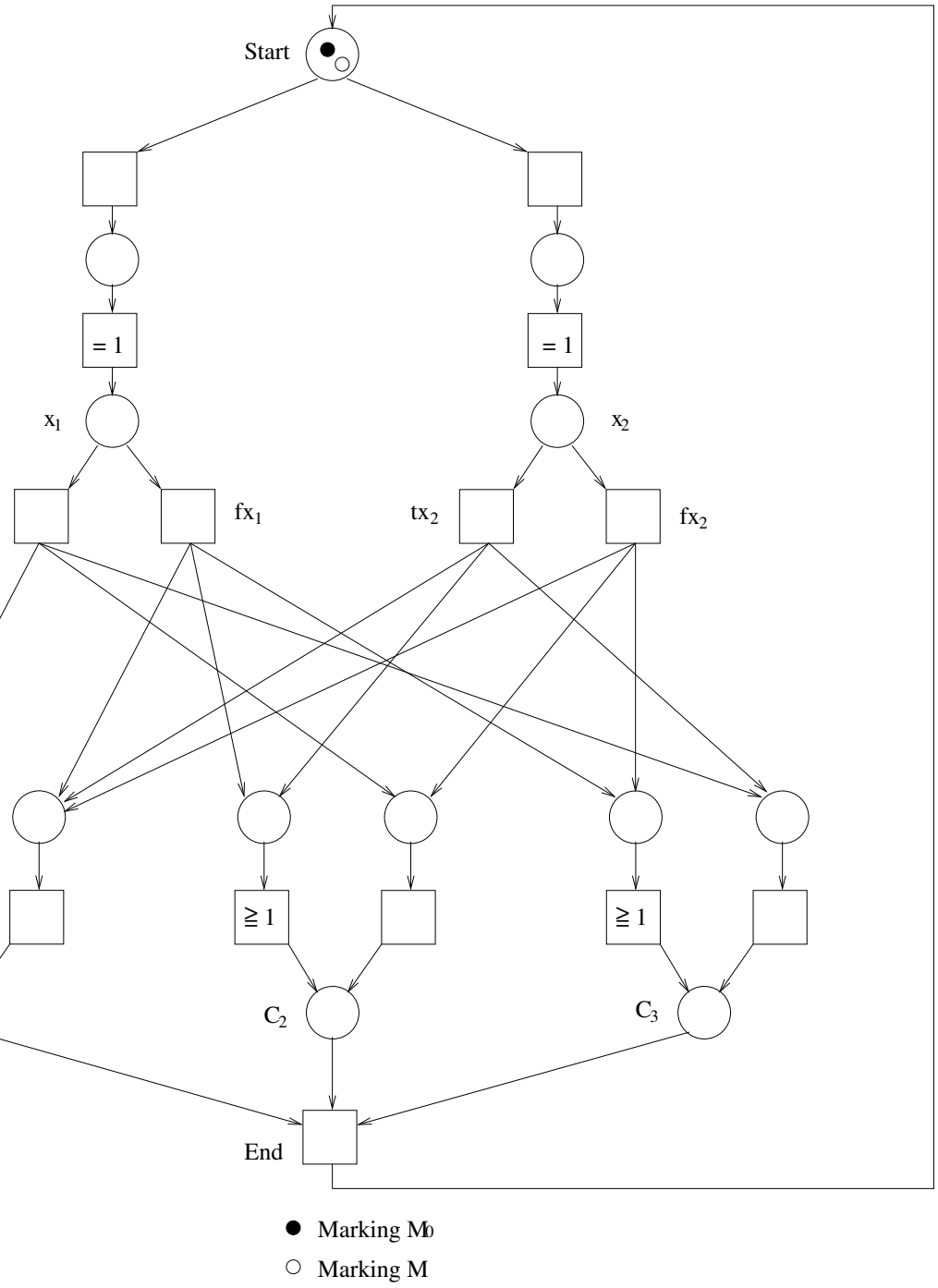


Figure 5.7: Result of the reduction for the formula $x_1 \wedge (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2)$

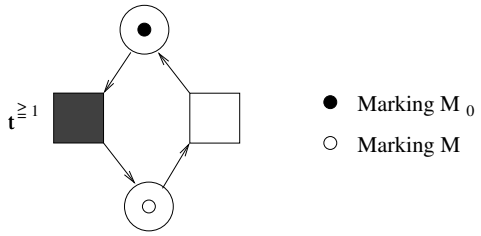


Figure 5.8: The first module

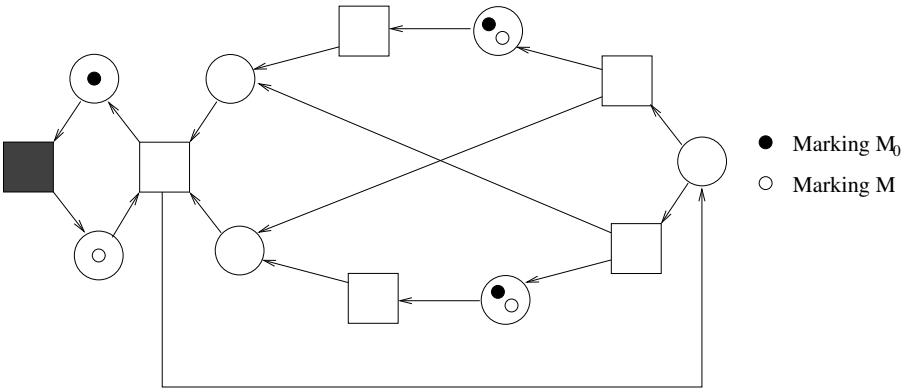


Figure 5.9: The second module.

Now we reduce the problem above to the reachability problem for live and bounded free-choice nets. Given a 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 the "module" shown in Figure 5.8. Similarly, we merge each transition of $T_{=1}$ with the transition $t^{=1}$ of a separate copy of the "module" shown in Figure 5.9.

The first module ensures that in order to reach the marking M the transition $t^{\geq 1}$ has to be fired at least once. The second module ensures that the transition $t^{=1}$ has to be fired exactly once. □

As for Commoner's Theorem, membership in NP is harder to prove. It follows from this theorem, due to Yamasaki *et al.*

Definition 5.3.17 *Let* $N = (S, T, F)$ be a net, and let $U \subseteq T$. The subnet $N_U = (S', T', F')$ generated by U is given

by:

- $T' = U$,
- $S' = {}^\bullet U \cup U^\bullet$, and
- $F' = F \cap ((S' \times T') \cup (T' \times S'))$.

Theorem 5.3.18 *[Reachability Theorem]*
Let (N, M_0) be a live and bounded free-choice system. M is reachable from M_0 iff there $X \in \mathbb{N}^{|T|}$ such that

- $M = M_0 + \mathbf{N} \cdot X$, *and*
- (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. Omitted. □

Membership in NP can then be proved as follows: Guess a set $U \subseteq T$, construct N_U, compute in polynomial time the maximal trap of N_U unmarked at M, 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 it is a solution of $M = M_0 + \mathbf{N} \cdot X$. Proving that the vector can be guessed in polynomial time follows from the fact that Integer Linear Programming is also in NP. A more direct proof of membership in NP follows from the Shortest Sequence Theorem for free-choice systems (see Theorem 5.3.21 below).

For systems satisfying an additional condition there is a polynomial algorithm. A Petri net (N, M_0) is *cyclic* if, loosely speaking, it is always possible to return to the initial marking. Formally: $\forall M \in [M_0\rangle : M_0 \in [M\rangle$. We have:

Theorem 5.3.19 *[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 5.3.20 *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.

This result is only useful if we are able to check efficiently if a live and bounded free-choice system is cyclic. The following theorem shows that this is the case:

Theorem 5.3.21 *A live and bounded free-choice system* (N, M_0) *is cyclic iff* M_0 *marks every proper trap of* N.

Proof. Omitted. □

5.3.4 Other properties

There is also a Shortest Sequence Theorem for live and bounded free-choice nets.

Theorem 5.3.22 *[Shortest Sequence Theorem]*
Let (N, M_0) *be a* 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 prove 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.