

Petri Nets

David Delfieu

October 7, 2018

Plan du cours

1 Petri Nets

- ▶ Basic Definitions
- ▶ Characteristics
- ▶ Properties which are depending on initial Marking
- ▶ Decidability of properties
- ▶ Properties which are not depending on initial Marking
- ▶ Computing of repetitive and conservative component
- ▶ Reduction techniques

2 Coding

- ▶ Introduction to functional language
- ▶ Coding a RdP in LISP/Racket

History

- 1960-1962 : Carl Adam Petri : *Communication with automaton*
- 1975 : First symposium at the MIT
- 1978 :
 - ▶ AFCET normalize the description of automaton
 - ▶ Definition of GRAFCET
- 1980 : Springer Verlag - Lecture notes in Computer Science : *Advance in Petri Nets* .

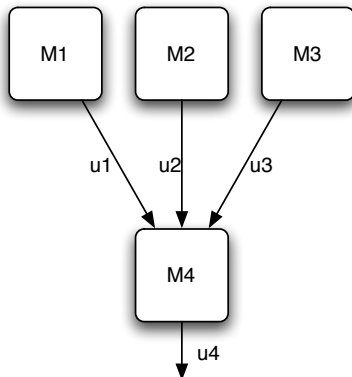
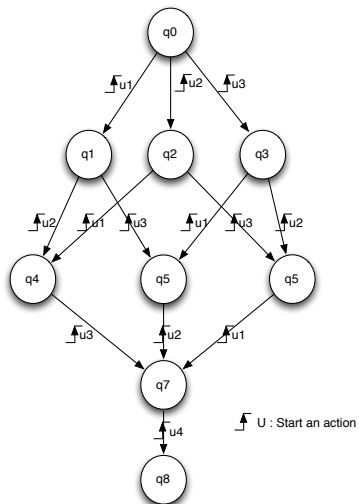
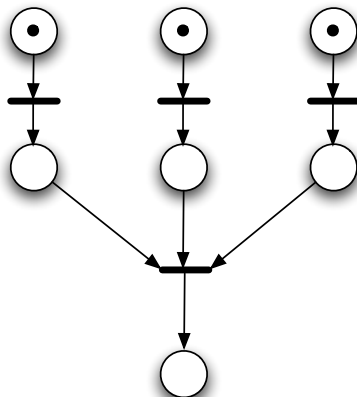


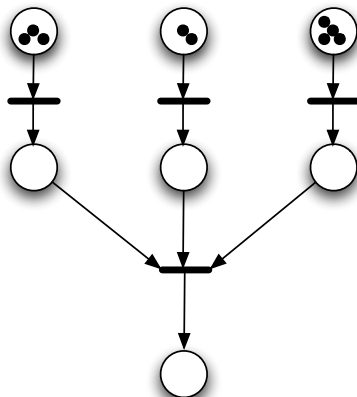
Figure: Manufacturing system



Motivation

- Moore Machine
- Combinatoty Explosion of the number of states
 - ▶ Modelling of parallelism by interleaving
 - ▶ Quantitative representation of ressources
- Need to modelize abstract concepts





Petri Nets

Definition

A Petri Nets is a quadruplet $R = \langle P, T, Pre, Post \rangle$ where :

- P : finite set of places,
- T : finite set of transitions,
- $Pre : P * T \longrightarrow N$ is an application defining previous places
- $Post : P * T \longrightarrow N$ is an application defining following places

Alternative notation:

- $C = Post - Pre$: *inCidence matrix*
- $\forall p \in P, Pre(p, t) \Leftrightarrow Pre(., t) \Leftrightarrow |{}^\bullet t|$
- $\forall p \in P, Post(p, t) \Leftrightarrow Post(., t) \Leftrightarrow |t^\bullet|$

Petri Nets

Definition

A Petri Nets is a quadruplet $R = \langle P, T, Pre, Post \rangle$ where :

- P : finite set of places,
- T : finite set of transitions,
- $Pre : P * T \longrightarrow N$ is an application defining previous places
- $Post : P * T \longrightarrow N$ is an application defining following places

Alternative notation:

- $C = Post - Pre$: *inCidence matrix*
- $\forall p \in P, Pre(p, t) \Leftrightarrow Pre(., t) \Leftrightarrow |\bullet t|$
- $\forall p \in P, Post(p, t) \Leftrightarrow Post(., t) \Leftrightarrow |t\bullet|$

Marked Petri Nets

Definition

A Marked Petri Net is a quadruplet

$$R = \langle R, M_0 \rangle$$

where :

- R : Petri Nets
- M_0 : is the Initial Marking :

$$M_0 : P \longrightarrow N$$

$M_0(p)$ is the number of tokens in the place p .

Graph and matrix

Arc

- An arc connects a place p to a transition t iff $Pre(p, t) > 0$
- An arc connects a transition t to a place p iff $Post(p, t) > 0$

Pre , $Post$ and C

Let \mathcal{P} and \mathcal{T} the dimensions of P and T .

- M is a vector of dimension \mathcal{P}
- Pre , $Post$ and C are matrix $(\mathcal{P}, \mathcal{T})$
- Let's note $Pre(., t)$, $Post(., t)$ and $C(., t)$, the associated vectors to t of dimension \mathcal{P}

Example

$$P = \{P_1, P_2, P_3\} \text{ and } T = \{T_1, T_2, T_3, T_4\}$$

- Compute Pre, Post and C

- $$Pre = \begin{array}{c|cccc} & T_1 & T_2 & T_3 & T_4 \\ \hline P_1 & 0 & 1 & 0 & 0 \\ P_2 & 1 & 0 & 3 & 0 \\ P_3 & 0 & 0 & 0 & 1 \end{array}$$

- $$Post = \begin{array}{c|cccc} & T_1 & T_2 & T_3 & T_4 \\ \hline P_1 & 1 & 0 & 0 & 0 \\ P_2 & 0 & 1 & 0 & 3 \\ P_3 & 0 & 0 & 1 & 0 \end{array}$$

- $$C = \begin{array}{c|cccc} & T_1 & T_2 & T_3 & T_4 \\ \hline P_1 & 1 & -1 & 0 & 0 \\ P_2 & -1 & 1 & -3 & 3 \\ P_3 & 0 & 0 & 1 & -1 \end{array}$$

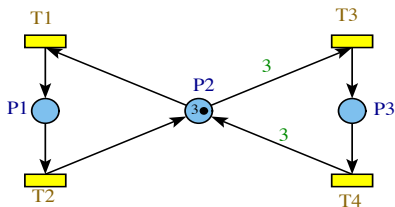


Figure: Example of PN

Example

$$P = \{P_1, P_2, P_3\} \text{ and } T = \{T_1, T_2, T_3, T_4\}$$

- Compute Pre, Post and C

- $$Pre = \begin{matrix} & T_1 & T_2 & T_3 & T_4 \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & P_1 \\ & P_2 \\ & P_3 \end{matrix}$$

- $$Post = \begin{matrix} & T_1 & T_2 & T_3 & T_4 \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{bmatrix} & P_1 \\ & P_2 \\ & P_3 \end{matrix}$$

- $$C = \begin{matrix} & T_1 & T_2 & T_3 & T_4 \\ \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \end{bmatrix} & P_1 \\ & P_2 \\ & P_3 \end{matrix}$$

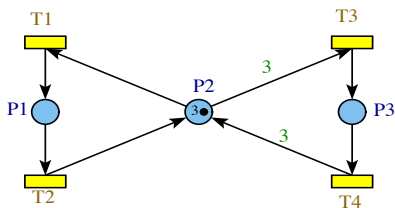


Figure: Example of PN

Example

$$P = \{P_1, P_2, P_3\} \text{ and } T = \{T_1, T_2, T_3, T_4\}$$

- Compute Pre, Post and C

- $$Pre = \begin{matrix} & T_1 & T_2 & T_3 & T_4 \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & P_1 \\ & P_2 \\ & P_3 \end{matrix}$$

- $$Post = \begin{matrix} & T_1 & T_2 & T_3 & T_4 \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{bmatrix} & P_1 \\ & P_2 \\ & P_3 \end{matrix}$$

- $$C = \begin{matrix} & T_1 & T_2 & T_3 & T_4 \\ \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \end{bmatrix} & P_1 \\ & P_2 \\ & P_3 \end{matrix}$$

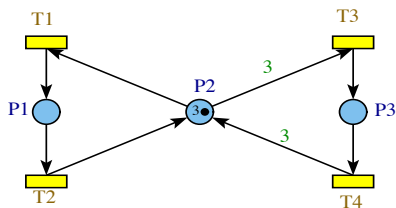


Figure: Example of PN

Example

$$P = \{P_1, P_2, P_3\} \text{ and } T = \{T_1, T_2, T_3, T_4\}$$

- Compute Pre, Post and C

- $$Pre = \begin{array}{c} T_1 \quad T_2 \quad T_3 \quad T_4 \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{array}{l} P_1 \\ P_2 \\ P_3 \end{array} \end{array}$$

- $$Post = \begin{array}{c} T_1 \quad T_2 \quad T_3 \quad T_4 \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{array}{l} P_1 \\ P_2 \\ P_3 \end{array} \end{array}$$

- $$C = \begin{array}{c} T_1 \quad T_2 \quad T_3 \quad T_4 \\ \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{array}{l} P_1 \\ P_2 \\ P_3 \end{array} \end{array}$$

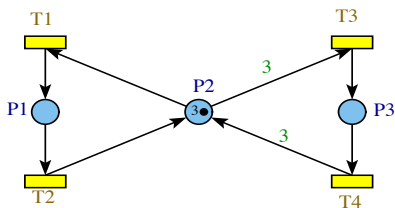


Figure: Example of PN

Example (suite)

The Initial Marking is :

$$M = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} \quad \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix}$$

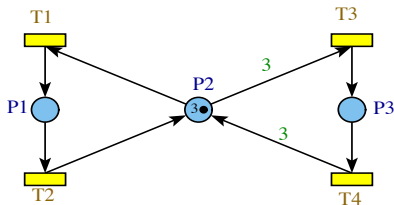


Figure: Example of PN

Enabled Transition

Definition

A transition t is enabled *iff*

$$\forall p \in P : M(p, t) \geq Pre(p, t)$$

Alternative notation :

$$M \geq Pre(., t)$$

$$M \xrightarrow{t}$$

Example

In the PN of the following figure the Initial Marking is

$$M = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}$$

The transitions t_1 and t_3 are enabled because

$$Pre(., t_1) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{et} \quad Pre(., t_3) = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}$$

and then $M \geq Pre(., t_1)$ et $M \geq Pre(., t_3)$

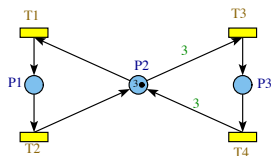


Figure: Example of PN NON pur

Firing a transition

Definition

If $M \geq Pre(., t)$ then the firing of t is expressed by $M \xrightarrow{t} M'$ where M' can be obtained by

$$\forall p \in P : M'(p) = M(p) - Pre(p, t) + Post(p, t)$$

Alternative notations :

$$M' = M - Pre(., t) + Post(., t)$$

$$M \xrightarrow{t} M'$$

In the previous example after the firing of t_1 :

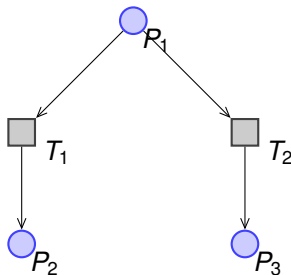
$$\begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

State Graph

Textual Definition:

Definition

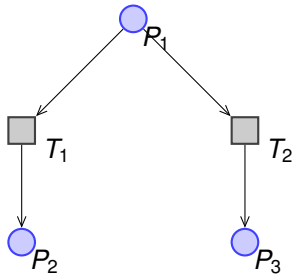
A PN is a state graph *iff* every transition t has at most, one input place place and one output place.



Formal Definition:

Definition

A PN is a state graph $\iff \forall t \in T, |\bullet t| \leq 1 \wedge |t\bullet| \leq 1$

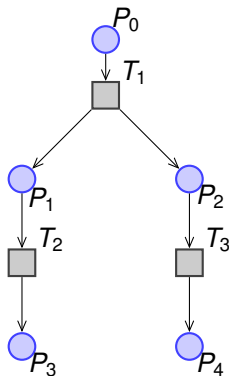


Event Graph

Textual Definition:

Definition

A PN is an event graph *iff* every place p has at most, one incoming arc and one outgoing arc.

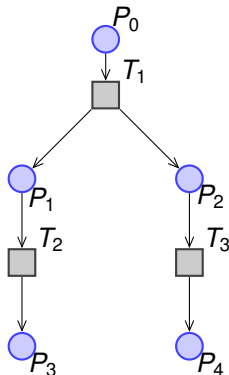


Event Graph

Formal Definition:

Definition

A PN is a state graph $\iff \forall p \in P, |\bullet p| \leq 1 \wedge |p\bullet| \leq 1$



PN without conflicts

Definition

A PN is without conflict *iff* every place p has at most one outgoing arc.

The set of event graph is included in the set of *PN* without conflicts.

PN without conflicts

Definition

A PN is without conflicts $\iff \forall p \in P, |p^\bullet| < 2$

Conflicts

Definition

A conflict can be defined by a place p and a set $\{t_1, t_2, \dots, t_n\}$ such as $p^\bullet = \{t_1, t_2, \dots, t_n\}$

Free choice PN

Definition

A PN is “free choice” is a PN for which every conflict $c = \langle p_c, \{t_1, t_2, \dots, t_n\}_c \rangle$ of the PN is such as

$$\neg(\exists p' \in P / p' \neq p_c \wedge p'^{\bullet} \cap \{t_1, t_2, \dots, t_n\}_c \neq \emptyset)$$

- The set of PN without conflict is included in the set of free choice PN .
- The set state graph is included in the set of free choice PN .

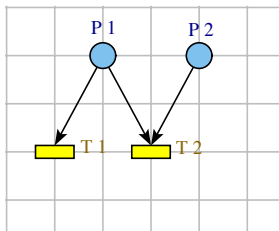
Simple choice PN

Definition

A simple choice PN is a PN for which every transition is involved in at most only one conflict. Let C the set of conflicts:

$$\forall c_1, c_2 \in C \wedge c_1 \neq c_2, \forall t_i \in c_1, t_i \notin c_2$$

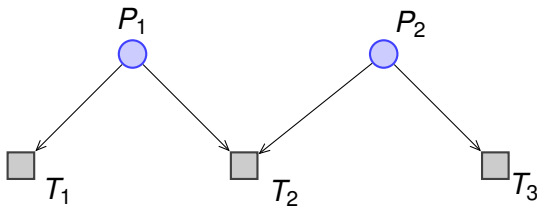
- The set of free choice PN is included in the set of simple choice PN.



Generalized PN

No constraints on conflicts

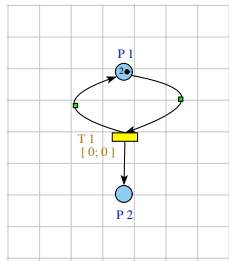
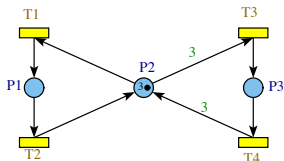
- The set of simple choice PN is included in the set of in generalized PN .



Pure PN

Definition

A PN is pure *iff* $\forall p \in P$ and $\forall t \in T : Pre(., t).Post(., t) = 0$



PN without loop

Definition

A PN is “without loop” is such as if it exists a transition t_j and a place p_i which is in the same time input place and output place of t_j then t_j has at least one other input place.

$$\forall t \in T, \text{ IF } (\exists p \in P / p \in \bullet t \wedge p \in t^\bullet) \text{ THEN } \exists p' \in \bullet t \wedge p' \notin t^\bullet$$

Effective conflict

Definition

structural conflict : The transitions t_1 and t_2 are in a *Structural conflict* iff it exists p such as $Pre(p, t_1) > 0$ and $Pre(p, t_2) > 0$

Definition

Effective conflict : The Transitions $t_1 \dots t_n$ are in an *Effective conflict* iff they are in structural conflict and it $\exists M$ reachable from M_0 such as :

$$M \geq Pre(., t_1)$$

...

$$M \geq Pre(., t_n)$$

Parallelism

Definition

structural parallelism : Two transitions t_1 and t_2 are in *structural parallelism* iff :

$$Pre(., t_1) \times Pre(., t_2) = 0 \quad (\text{scalar product})$$

Definition

effective parallelism : Two transitions t_1 and t_2 are in *effective parallelism* iff they are in structural parallelism and that it $\exists M$ reachable from M_0 such as :

$$M \geq Pre(., t_1)$$

$$M \geq Pre(., t_2)$$

Example of conflict and of parallelism

In the figure 2 :

- The transitions t_1 and t_3 are in withstructural conflict because :

$$\text{For } p_2 \text{ we have : } Pre(p_2, t_1) \cdot Pre(p_2, t_3) \neq 0$$

- with the Initial Marking M , t_1 and t_3 are in effective conflict
- t_2 and t_4 are in structural parallelism because :

$$Pre(., t_2) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \times Pre(., t_4) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0$$

- for the Initial Marking $\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$ the transitions t_2 and t_4 are in effective parallelism.

Example of conflict and of parallelism

In the figure 2 :

- The transitions t_1 and t_3 are in withstructural conflict because :

$$\text{For } p_2 \text{ we have : } Pre(p_2, t_1) \cdot Pre(p_2, t_3) \neq 0$$

- with the Initial Marking M , t_1 and t_3 are in effective conflict
- t_2 and t_4 are in structural parallelism because :

$$Pre(., t_2) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \times Pre(., t_4) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0$$

- for the Initial Marking $\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$ the transitions t_2 and t_4 are in effective parallelism.

Example of conflict and of parallelism

In the figure 2 :

- The transitions t_1 and t_3 are in withstructural conflict because :

$$\text{For } p_2 \text{ we have : } Pre(p_2, t_1) \cdot Pre(p_2, t_3) \neq 0$$

- with the Initial Marking M , t_1 and t_3 are in effective conflict
- t_2 and t_4 are in structural parallelism because :

$$Pre(., t_2) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \times Pre(., t_4) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0$$

- for the Initial Marking $\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$ the transitions t_2 and t_4 are in effective parallelism.

Example of conflict and of parallelism

In the figure 2 :

- The transitions t_1 and t_3 are in withstructural conflict because :

$$\text{For } p_2 \text{ we have : } Pre(p_2, t_1) \cdot Pre(p_2, t_3) \neq 0$$

- with the Initial Marking M , t_1 and t_3 are in effective conflict
- t_2 and t_4 are in structural parallelism because :

$$Pre(., t_2) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \times Pre(., t_4) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0$$

- for the Initial Marking $\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$ the transitions t_2 and t_4 are in effective parallelism.

Firing sequence

the If $M_0 \xrightarrow{t_1} M_1$ and $M_1 \xrightarrow{t_2} M_2$ it can be declared that :
 $ta; tb$ can be fired from M_0 which is noted

$$M_0 \xrightarrow{t_1; t_2} M_1$$

Definition

characteristic vector : Let \bar{s} the vector which components $\bar{s}(t)$ are the occurrence numbers of the transitions t in a firing sequence s . This vector is called the characteristic vector of s . Its dimension, is equal to the number of transitions in the PN .

Example of firing sequence

In the figure 2, if we consider the sequence $s = t_1; t_1; t_2$:

$$\begin{pmatrix} 0 \\ 3 \\ 0 \end{pmatrix} \xrightarrow{t_1} \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} \xrightarrow{t_1} \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} \xrightarrow{t_2} \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}$$

$$\overline{t_1; t_1; t_2} = \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{matrix}$$

The evolutions of the marking are given by the equation:

$$M' = M - Pre.\bar{s} + Post.\bar{s}$$

that can be noted by $M' = M + C\bar{s}$ with $M \geq 0, \bar{s} \geq 0$

Fondamental equation of a PN

The evolutions of a marking M are given by the equation:

$$M' = M - Pre.\bar{s} + Post.\bar{s}$$

that can be noted by: $M' = M + C\bar{s}$ with $M \geq 0, \bar{s} \geq 0$ It is the fondamental equation of a PN .

Remark

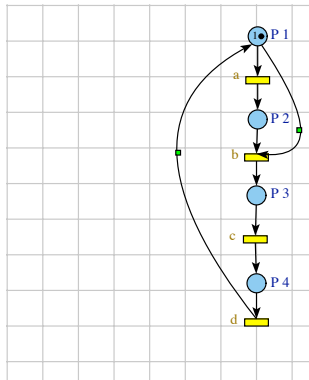
If a vector s verifies the fondamental equation it does not imply that s is a sequence that can be effectively fired.

Example : Fondamental equation and firing sequence

Let's consider the sequence $s_{abcd} = a; b; c; d$ and the PN

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} -1 & -1 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \quad \overline{s_{abcd}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Compute M'



Let's fire the
sequence $a; b; c; d$

Example : reachable markings

Set of reachable markings

The set of reachable markings $A(R; M_0)$ of a Marked Petri Net is the set of reachable markings from Initial Marking M_0 by a firing sequence :

$$A(R; M_0) = \{M_i, \exists s \quad M_0 \xrightarrow{s} M_i\}$$

Marking graph

If the set of reachable markings is finite, one can represent it as a graph from which the nodes belong to $A(R; M_0)$ and the arcs t verify $M_i \xrightarrow{t} M_j$.

Exercise : Computing the marking graph

- Can we schedule the events b and d in the marking graph?
- Can we schedule the events b and d in the PN ?

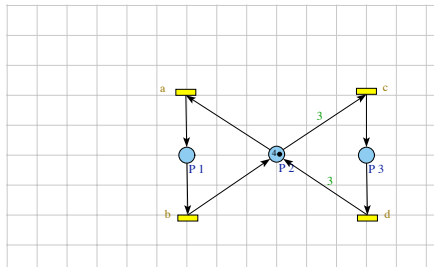


Figure: Marking graph

Example : marking graph

Attention

In this graph, the information that b and d can be executed concurrently is “lost”

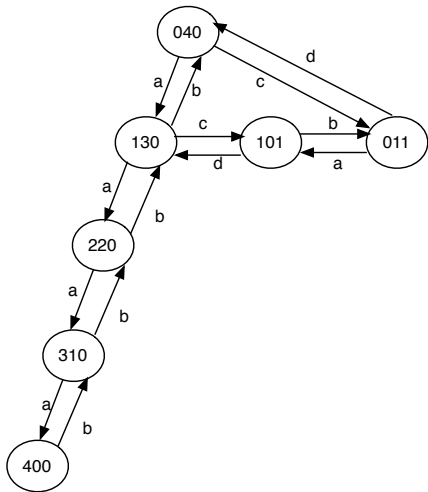


Figure: Marking graph

Grammar

Grammar

A grammar is defined by : $\langle NT, A, Q, \Sigma \rangle$, where :

- NT : set of non terminal symbol
- A : is the axiome, is a non terminal symbol,
- Q : is a set of production rule,
- Σ : is the alphabet, set of terminal symbol constituting words

Example of Grammar

Example : $G = \langle NT, S, Q, \Sigma, \rangle$

- $NT = \{S, T, U\}$
- $A = S$
- $Q :$

$$S \rightarrow aT$$

$$T \rightarrow bU$$

$$U \rightarrow cS + d$$

- $\Sigma = a, b, c, d$

This grammar allows to build words : $abd, abcabd, (abc)^n abd, \dots$

PN and Grammar

Grammatical Expression of a PN

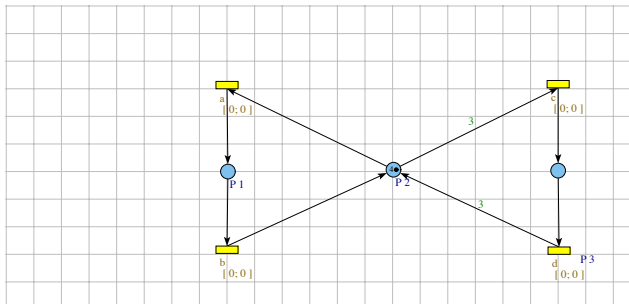
A PN is a degenerated grammar defined by : $\langle \Sigma, M_0, Q \rangle$ where :

- Σ : the alphabet is constituted by the set of places,
- $\mu(M_0)$: is the associated word to the Initial Marking ,
- Q : A set of rules with the following form:

$$\mu(Pre(., t_i)) \longrightarrow \mu(Post(., t_i))$$

where $\mu : M \longrightarrow P^*$: μ is an application associating à a marking to a word containing the labels of places.

Example



Grammatical expression of a PN

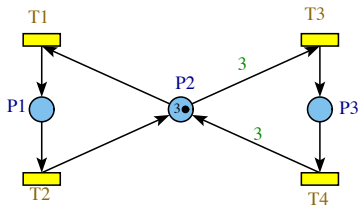
In the example of the previous figure:

- $P = \{p_1, p_2, p_3\}$
- $\mu(M_0) = p_2^3$: is the associated word to the Initial Marking ,
- $Q = \left\{ \begin{array}{lcl} a : & p_2 & \rightarrow p_1 \\ b : & p_1 & \rightarrow p_2 \\ c : & p_2^3 & \rightarrow p_3 \\ d : & p_3 & \rightarrow p_3^3 \end{array} \right.$

$$A(R; M_0) = \{p_2^3, p_1 p_2^2, p_1^2 p_2, p_1^3, p_3\}$$

Grammar : Bridge toward the systems of re-writting and Artificial Intelligence.

example



Place K-bounded

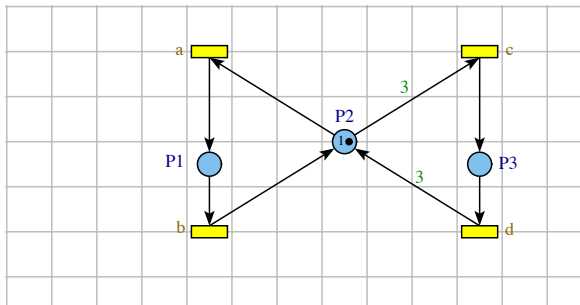
A place p of a Marked Petri Net is k -bounded *iff*

$$\forall M' \in A(R; M_0) \quad M'(p) \leq k$$

If $k = 1$ the place is called “binary”. In the PN of the following figure $M_0(p_2) = 3$, p_3 is binary while p_1 and p_2 are 3-bounded.

example

Let's modify the example :



With that Initial Marking , the PN is 1-bounded.

K-bounded and binary PN

PN K-bounded

A Marked Petri Net is K-bounded *iff* every places are K-bounded. A Marked Petri Net is is binary *iff* every places are binary. The PN is called Safe.

In the previous figure, with an Initial Marking of $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ the PN is safe, c and d are be never fired anymore.

Examples

Each time the sequence $s = a; b$ is fired, a token is added in p_3 . This place is unbounded thus the PN is unbounded as well.

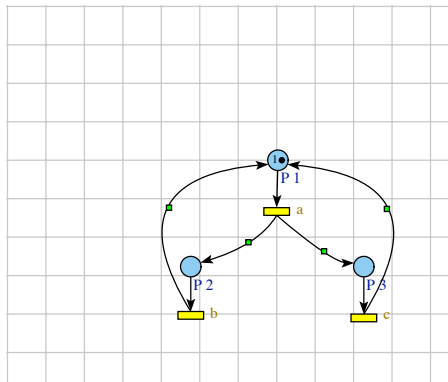


Figure: Unbounded PN

Quasi-Liveness

Quasi-alive Transition

A transition t of a Marked Petri Net is quasi-alive *iff* it exists a firing sequence s such as:

$$M_0 \xrightarrow{s} M' \text{ et } M' \xrightarrow{t}$$

Notation: $M_0 \xrightarrow{s;t}$

Alive transition

A transition t of a Marked Petri Net N is alive *iff*

$$\forall M' \in A(R; M_0) \exists s M' \xrightarrow{s;t}$$

Examples of Liveness

The transition d of the following figure is quasi-alive but not alive : d is reachable from the Initial Marking . But once it has been fired, it cannot be fired anymore.

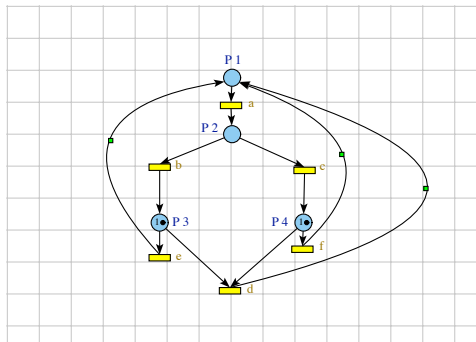
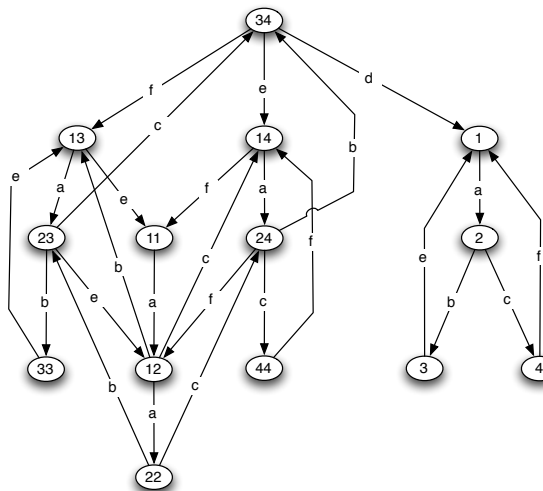


Figure: Quasi-alive transition and unlive transition

Examples of Liveness

- a, b, c, e, f alive
- d quasi-alive



Strong connexity

Strong connex component

G_1 is a *CFC* of the G iff $\forall (s_0, s_1) \in G_1, \exists$ a path from s_0 to s_1

- Previous example : The entire figure is not strongly connected. After the firing of d , the control cannot go back in the left part.
- Suppress arc d : Two subfigures strongly connected.

Alive Marked Petri Net

Alive Marked Petri Net

A Marked Petri Net $N = \langle R, M_0 \rangle$ is alive *iff* every transitions are alive.

Remark

- An Alive PN guaranties that no blocking can be imputed to the Marked Petri Net
- An alive PN guaranties that none parties are dead.

Example of alive Marked Petri Net

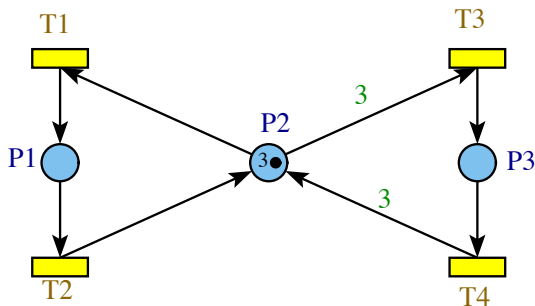


Figure: Alive Marked Petri Net

Example of unlive PN

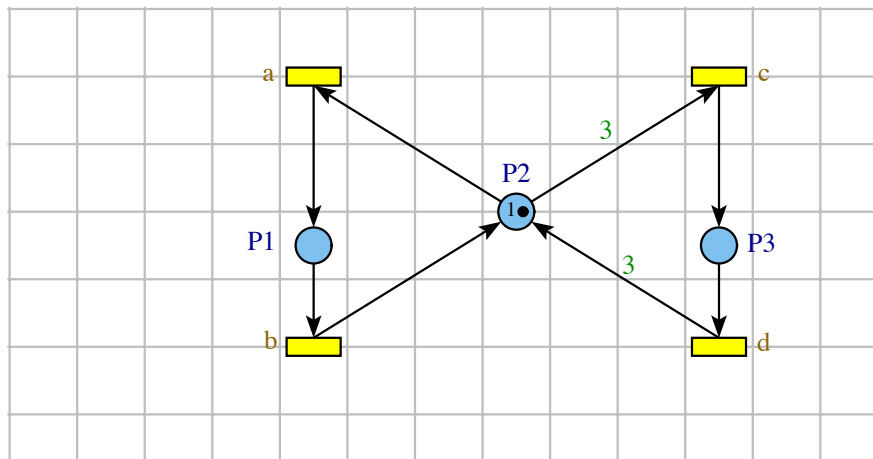
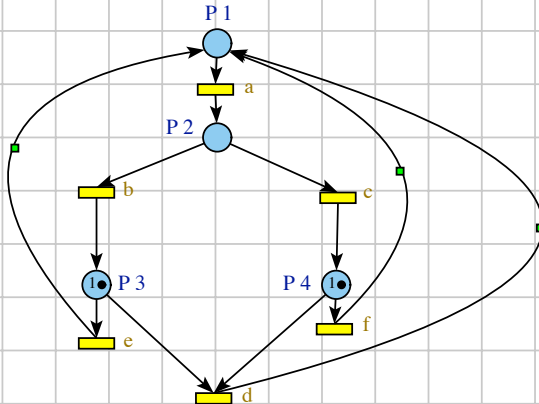


Figure: unlive PN

Liveness and boundedness

- A PN can be unbounded and Alive.
- A alive PN gauranties that nones parts are dead.

Example of unbounded alive PN



Reinitialisability PN

Reinitialisability PN

A marked PN $N = \langle R, M_0 \rangle$ is Reinitialisability *iff* its marking graph is strongly connex:

$$\forall M' \in A(R; M_0) \exists s M' \xrightarrow{s} M_0$$

Example of PN Reinitialisability

- Strong connexity \Rightarrow
every transition is firable
 \Rightarrow PN is then Alive.

- if $M_0 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$ then PN

is Alive AND
Reinitialisability.

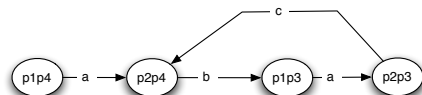


Figure: non Reinitialisability PN

Exercices

Let's find the good properties in the following exercises:

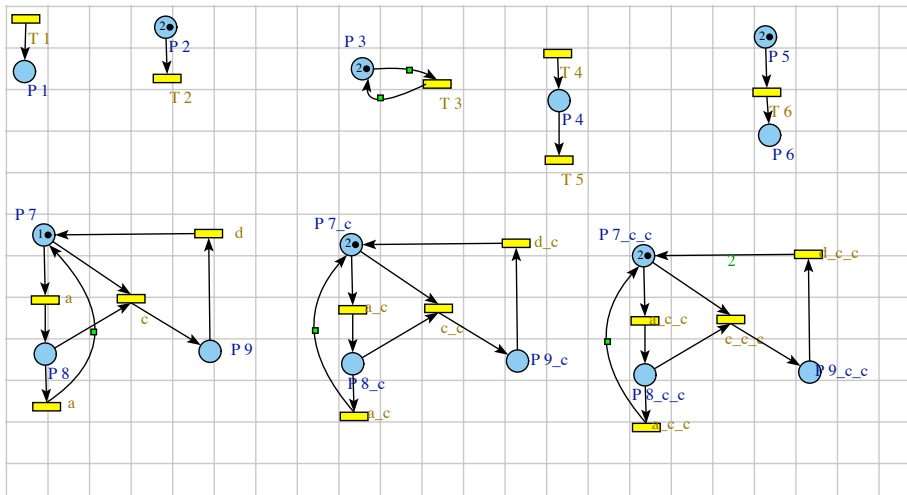


Figure: Non Reinitialisability PN

Conclusion

The presented properties depend of the Initial Marking . In the next section we focuss on the properties that are not dependant of the initial marking.

Conservative components and place invariants

$M(p_1) + M(p_2) = 1$.
 a and b let this sum
 unchanged.

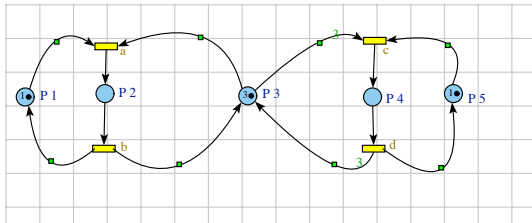


Figure: Conservative component

- $\forall M \in A(R, M_0) \quad M(p_1) + M(p_2) = 1$
- $\forall M \in A(R, M_0) \quad M(p_1) + M(p_2) = M_0(p_1) + M_0(p_2)$

Conservative components and place invariants

- The linear form $M(p_1) + M(p_2)$: *linear invariant of places*.
- Region $(p_1, p_2) (a, b)$: Conservative component.

linear invariant of place

A linear invariant of place is a linear function of the marking places and for which “only the value” depends of Initial Marking .

Conservative Component

An equality binding two linear relations of the marking places defines a region called a conservative component. It does not depend of Initial Marking .

Exercise

Exercise : Find intuitively the place invariants and the conservative components of

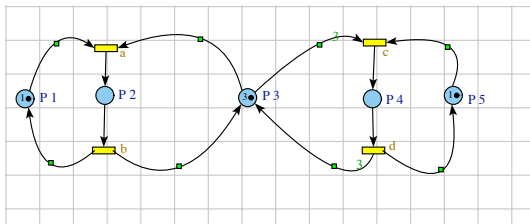


Figure: Place Invariant

Solution

- $M(p_2) + M(p_3) + 3M(p_4) = 3$ Linear Invariant
- $M(p_2) + M(p_3) + 3 * M(p_4) = M_0(p_2) + M_0(p_3) + 3 * M_0(p_4)$
 - ▶ Defines a conservative component
 - ▶ The region that contains p_2, p_3, p_4, c, a

Computing the linear invariants of places

- Let's Consider the fundamental equation of a PN .
- Let consider a column vector f^T of dimension \mathcal{P}
- Let's Multiply the fundamental equation by the vector f^T :

$$f^T.M' = f^T.M + f^T.C.\bar{s}$$

Computing the linear invariants of places

Conservative component

A Conservative Component (*CC*) of a PN is a solution of the equation $f^T.C = 0$

- A *CC* defines a sub Petri net from which the places corresponds to the non null components of f with the transitions of the initial PN.
- If $f^T.C = 0$ we have $f^T.M' = f^T.M_0 \forall M \in A(R, M_0)$. The solutions of the equation are the linear invariant of places.

Invariant of place

If $f^T.C = 0$, consequently, we have :

$$f^T.M' = f^T.M_0 \quad \forall M \in A(R, M_0)$$

This equation is the corresponding place invariant.

Remarks :

- The expression of a place invariant depends of the Initial Marking .
- A Conservative Component (CC) of a *PN* defines a conservative subnet whatever the Initial Marking .

Positive CC

Only positives solutions are of interest :

$$f^T . C = 0$$

with $f > 0$.

Moreover, as C is a integer matrix the solutions of the equations will be rational and we will be able to bring back those rationals to integer numbers.

Linear invariant of transitions and repetitives stationnary component

- * $c; d$: unchanged
- * $(c, d) + (P_3, P_4, P_5)$
- * $(cd)^n$: unchanged

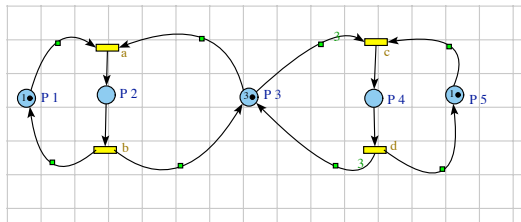


Figure: Linear invariant of transitions

Invariant of transitions

Invariant of transitions

An invariant of transitions is a firing sequence of transitions effectively firable from Initial Marking but that does not modify the marking of the net.

Remarks :

- This sequence must be effectively reachable from the Initial Marking .
- This sequence can be fired an arbitrary number of time.

Stationary repetitive component

Stationary repetitive component

A Stationary repetitive component is a solution \bar{s} of the equation $C.\bar{s} = 0$.

If $C.\bar{s} = 0$ that's mean that the firing of s is neutral for the net.

Remarks: Stationary repetitive component

Remarks:

- We will restrain to the positive solutions of this equation : $C \cdot \bar{s} = 0$ with $\bar{s} > 0$.
- A negative value would correspond to a backward firing of a transition.
- The subnet that is defined from the transitions belonging to a sequence which is an invariant of transition.
- This subnet (region) contains the transitions of the sequence, with every places connected to the transitions.

Summary

Conservative component (CC)

A CC is a solution of $f^T.C = 0$ that defines a region independantly of the Initial Marking .

Invariant of places

- Let's f a vector, then $f^T.M = f^T.M_0$ is an invariant of places.
- The value $f^T.M_0$ depends obviously of the Initial Marking .

Summary (suite)

Stationary repetitive component

A Stationary repetitive component is a solution \bar{s} of the equation $C \cdot \bar{s} = 0$. This is a subnet of the PN initial. it does not depend of the Initial Marking .

Invariant of transitions

This is a sequence of transition s such as $C \cdot \bar{s} = 0$ with $\bar{s} > 0$. it depends of the Initial Marking because s must be effectively firable from Initial Marking .

K-bounded Petri Net

Remarks :

- A PN is k -bounded *iff* if it possesses a finite marking graph.
- If the marking graph is not finite, it exists at least a sequence which is not stationary repetitive.

Properties allowing to establish the decidability of the Boundedness

The decidability of the Boundedness results of the following properties:

Monotony

$\exists M, M', s$ such as $M \xrightarrow{s}$ and $(M' > M) \implies (M' \xrightarrow{s})$

Non K-bounded Petri Net

If $\exists M, M' \in A(R; M)$ such as $M \xrightarrow{s} M'$ and $M' > M$ then the PN is not K-bounded.

Properties allowing to establish the decidability of the Boundedness

Lemma of Karp and Miller

Every infinite series of vectors constituted of positive or null integers :

$$v_1, v_2, \dots, v_k, \dots$$

is such as it exists at least 2 where :

$$v_i, v_j \text{ with } i < j \text{ such as } v_i < v_j \text{ where } v_i = v_j$$

Properties allowing to establish the decidability of the k-boundedness

An increasing monotone infinite series of markings

- ① Let s an infinite sequence of firings of transitions. The series of markings is a series of vectors of positive or null integers.
- ② The previous lemma implies then the existence of a sub-sequence s such as : $M \xrightarrow{s} M'$ and $M \leq M'$

The sequences defined at point 2 are characteristics of non stationary sequences of infinite length. It follows that the net is not bounded.

Algorithm

The Algorithm is based on the computing of the marking graph.

- 1 The starting point is the Initial Marking :
 - ▶ For every firable transition the new marking is computed.
 - ▶ Then we iterate on the set of new markings.
 - 2 Stop condition:
 - a We found an already found marking, the current branch in way of exploration is stopped.
 - b We found a marking strictly greater than a marking already found.
⇒ The algorithm is totally stopped.
-
- a If the Algorithm is ended by the stop condition *a*, the net is *k*-bounded. So, it hasn't got any infinite branch.
 - b If the Algorithm is ended by the stop condition *b* then the net is unbounded.

Termination of the algorithm

The algorithm ends in any case because:

- We cannot get branch of infinite length without encounter a marking that is strictly greater to an already found marking: Karp and Miller lemma
- The graph is finite since :
 - ▶ the number of arcs is finite (T is of finite dimension)
 - ▶ the number of markings is finite (stop of the algo).

Decidability of Properties

- The Boundedness is decidable for a Marked Petri Net
- Re-initialisable, Alive and quasi-alive are decidable

The establishing of properties

Prove order of properties :

- 1 Computation of marking graph \Rightarrow k-bornitude,
- 2 Reinitialisability,
- 3 quasi-alive,
- 4 Alive.

Relations between the Properties

Réinitialisable and connexity

The PN R is Reinitialisability for the Initial Marking $M_0 \iff GA(R, M_0)$ strongly connex.

Sketch of Prove

- Reinitialisability: $\forall M_i, M_j \in A(R, M), \exists$ then a path from M_i to M_0 .
- In the same manner, M_j is reachable from M_0 by the definition of the establishing of the marking graph. Then it exists a path of M_0 a M_j .
- By transitivity it exists then a path from M_i a M_j . This for every pair of existing marking.
- The graph is strongly connex.

Relations between properties

Réinitialisable and Liveness

R Reinitialisable for $M \implies (R \text{ quasi-alive} \iff R \text{ Alive})$

Sketch of prove

- 1 R alive \implies quasi-alive.
- 2 Reciprocal:
 - If R is Reinitialisable: From every reachable marking M_i , we can go back to M_0
 - As it is quasi-alive, considering any transition t , it exists at least one path from M_0 leading to M_j where t is firable
 - By transitivity for any marking M_i , for any transition t , we can exhibit a path leading to a marking M_j where the transition t is firable.
 - R is alive

Liveness

- ① If the net is k -bounded, the graph marking can be computed
- ② If the graph is strongly connex, it is then Reinitialisable,
- ③ If for every transition t , it can be checked that every transition t is quasi-alive (t appears at least one time in the marking graph)
- ④ The net is Alive.

Reduction

Equivalence relative to Good properties

Two nets are equivalent relatively to the good properties (Alive, Reinitialisability, quasi-alive, K-bounded) *iff* the nets verify the same subset of Properties.

Goal : Find the reduction rules which preserve these properties.

Simple cases of substitution of places

Let's t_e and t_s the incoming and outtransitions d'betweenne and of sortie.

If this place verifie the conditions suivantes :

- If $Post(p, t_e) = Pre(p, t_s)$: egalite the weights entrant and sortant
- $\forall p' \in P$ If $p' \neq p$ then $Pre(p', t_s) = 0$: pas d'another place that p en betweenne of t_s

Alors this place is substituable.

Substitution

p can be suppressed by substituting t_e and t_s by the transition t_{es}

$$\forall p' \in P \text{ Pre}(p', t_{es}) = \text{Pre}(p', t_e)$$

$$\forall p' \in P \text{ Post}(p', t_{es}) = \text{Post}(p', t_e) + \text{Post}(p', t_s)$$

Examples : Simple substitutions of places

The transitions a and b are substituted by the transition ab

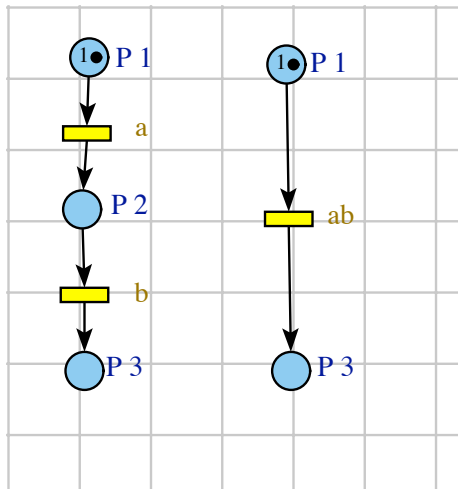


Figure: Simple substitution

Examples : Simple substitutions of places

This example shows that there is no restriction on the transition a . We will note that the place p_3 is marked. As this token must necessarily come in P_5 , in the reduced net, we put it in this place.

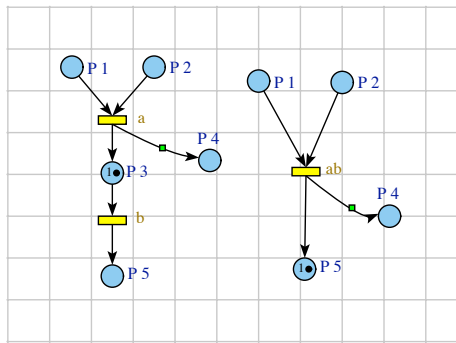


Figure: Simple case of substitution

Places with several transitions in input and in output

Condition : every weight must be equal.

Moreover, if this place verifies the following conditions:

- The outgoing transitions of P must not have any other input places except P
- \nexists an outgoing transition t_j which is in the same time an input transition and an output transition (loop).
- \exists at least an output transition which is not a well transition.
- A well transition does not have any outcomming places.

Then this place is substituable

Remark : There is no condition on the input transition.

Examples : Restrictions

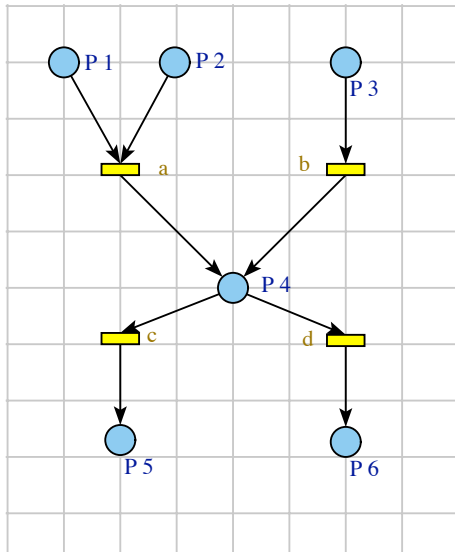
this may not be the case

- If the reduction carries on a marked place, the reduced net may not be reinitialisable anymore.
- If in the initial net, an output transition has no output place, there is no more equivalence in the point of view of the of k -boundedness.
- If in the initial net is K -bounded the value k is lost.

Examples : complex substitutions of places

Considering every pair formed with an input transition and an output transition, we have the equality of the weights :

$$\begin{aligned} Post(P_4, a) &= Post(P_4, b) = \\ &= Pre(P_4, c) = Pre(P_4, d) \end{aligned}$$



Examples : Complexes of substitution of places

net réduit

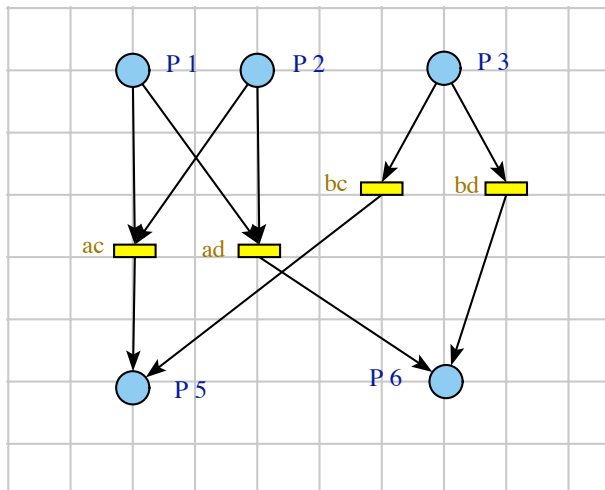


Figure: Complex substitutions

Implicit places

Implicit place

- An implicit place is a place from which the marking is a linear combination of places.
- Moreover, these output transitions does not introduce any extra firing condition.

Degenerated implicit places

If a place is connected only by elementary loops, its marking is constant. And this is an *implicite degenerated* place.

Implicit places

Remarks :

- An implicit place is quite useless because it does not influence on these output transitions.
- An implicit place belongs to a conservative component.

Exercise

- Let's show that p_1 is implicit, by the computing of the C.C. relative to the places p_2 and p_3 .
- Compute the linear invariants of places for $M_0 = (0, 0, 0, 1, 0)$ and $M'_0 = (0, 1, 0, 0, 0)$

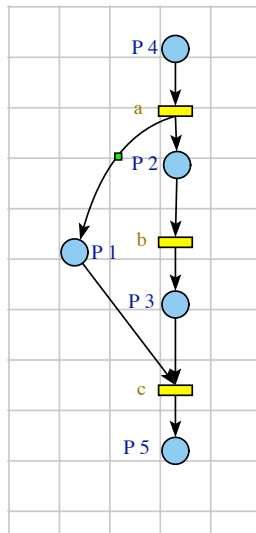


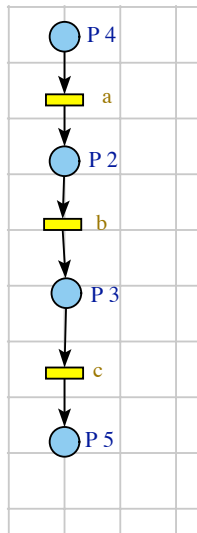
Figure: Implicit places

Solution

- For M_0 we have

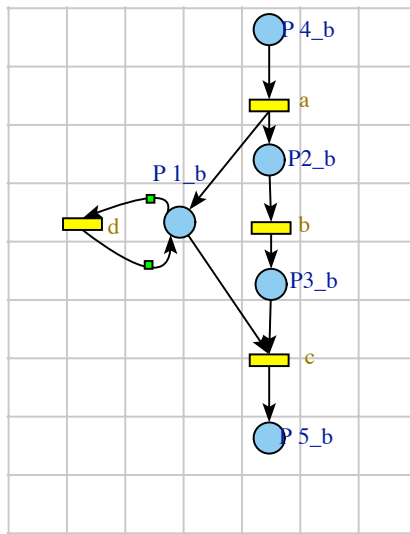
$$M(p_1) = M(p_2) + M(p_3)$$
- For M'_0 we have

$$M(p_1) = M(p_2) + M(p_3) - 1$$



Counterexample

In this example, d can occur only between the firing of a and c . The suppression of p_{1b} would cancel this property. Firing d implies a condition on the firing of c , then the place p_{1b} cannot be implicate.



Degenerated implicit place

If a place is only connected by elementary loops, this transition is implicit relatively to the empty set, Its marking is constant. Its a degenerated implicit place.

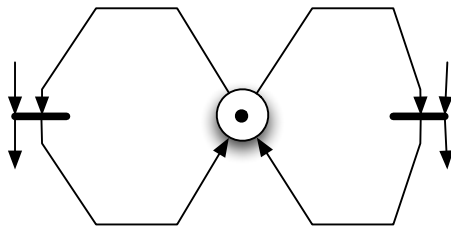


Figure: Degenerated implicit place

Identical places

Identical places

Two places p_1 and p_2 are identical if they have the same input and output transitions.

Remark : p_1 is then implicit relative to p_2 .

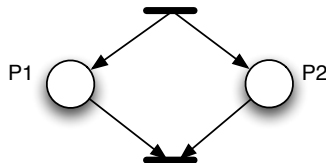


Figure: Identical places

Neutral transitions

neutral transition

A transition t is neutral (also called an identity transition), if it is only connected to the net by elementary loops, such as :

$$Pre(., t) = Post(., t)$$

Remark :

Its firing does not modify the net:

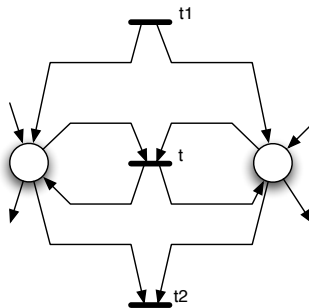


Figure: t is neutral

reduction of a neutral transition

remarks

To suppress a neutral transition t while respecting the liveness property, one must make sure that t is alive or else it could suppress a dead part of the net:

One could then transform a non alive net in an alive net !

counterexample of the reduction of a neutral transition

In this figure, d is not alive.
After the firing of a , d is no
more firable. If one suppress
the neutral transition d the
reduced RdP will be alive.

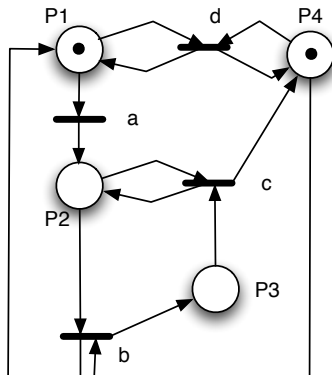


Figure: d is neutral goal cannot be suppressed

reduction of a neutral transition

Conditions of suppression of a neutral transition t

To suppress a neutral transition t :

- Either it exists a transition t_1 (present in the reduced net) which enable t in the initial net:

$$Post(., t_1) = Pre(., t)$$

- Either it exists t_2 which carry, in the initial net, the same firing conditions:

$$Pre(., t_2) = Pre(., t)$$

Identical transitions

Identical transitions

Transitions t_1 and t_2 are identical iff:

$$Pre(., t_1) = Pre(., t_2)$$

$$Post(., t_1) = Post(., t_2)$$

One can be suppresses preserving the properties of the RdP.

Example: Simplification of Identical transitions

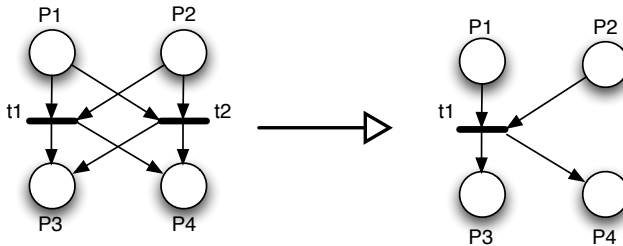


Figure: Suppression of $t2$

Computing principle

Computing principle

The principles of the computing are the same for the types of Components. One looks to resolve : $f^T.C = 0$:

- For the conservative components: One only has to solve this equation.
- For the repetitive stationary sequence, one has to apply the method we the transpose of C .

Principe

This equation can be expressed by:

$$C^T.f = 0$$

Considering a net constituted of n places and m transitions then we get a system of m linear equations with a null solution vector:

$$\left. \begin{array}{ccccccc} c_{11}.f_1 & + & \dots & + & c_{n1}.f_n & = & 0 \\ \dots & + & \dots & + & \dots & = & 0 \\ c_{1m}.f_1 & + & \dots & + & c_{nm}.f_n & = & 0 \end{array} \right\}$$

Reasoning

$$C^T.f = 0$$

- The set of the solutions form a vectorial space.
- The degenerated solution $f^T = 0$ has no interest.
- The base of the solution vectorial space gives the set of conservative components of the PN.

Dimension of the vectorial spaces

Rank of matrix

The rank of a matrix is the dimension the greater square su-matrix where the determinant is non zero.

Let's r the rank of C . The dimension of the vectorial space will be

$$\dim_p = n - r$$

The dimension of the vectorial space of the components will be:

$$\dim_t = m - r$$

Gauss method

Gauss method

It's a systematic method of resolution for the linear equations. It is based on the fact that the following transformations do not change the solution:

- Exchange of 2 lines.
- Multiplication of a line by a non null scalar.
- Addition of a line to another.

Variable changing

La Gauss method triangularizes C . For that one has to make linear combinations of lines of C . These operations will be memorised by changing the variables of the vector f :

- Multiplication: f_i by $a \cdot f_i$ with $a \neq 0$,
- Addition: f_i par $f'_i + f'_j$
- Exchange of lines: f_i by f'_j and f_j by f'_i

Obtention of the components

- C is not necessarily square. If r is the rank, it exists S of dimension r such as:
 $\det(S) \neq 0$ and then the sub-system is formed of independant linear equations with one solution.
- Let's consider C' the result of the resolution.

$$C' = \left[\begin{array}{c|c} S & S' \\ \hline (0) & (0) \end{array} \right]$$

- the columns that does not belong to S : Sub-matrix S' will be linear combinaisons of the columns of S .

Obtention of the components

$$C' = \left[\begin{array}{c|c} S & S' \\ \hline (0) & (0) \end{array} \right]$$

- The $n - r$ lines that does not belong to S neither to S' will only contains null values.
- Indeed, they corresponds to variables that does not appear in no equation then their values are free.
- Then we have the following system:

$$\begin{array}{lcl} f'^T.C' = 0 & \text{with} & f = F.f' \\ & \text{et} & C' = F^T.C \end{array}$$

Obtention of the components

$$f'^T.C' = 0 \quad \text{with} \quad f = F.f'$$

- F is a regular matrix (non null determinant) which depicts all the changes in variables which have been realized for the resolution.
- We obtain S we the form:

$$\begin{bmatrix} s_{11} & s_{12} & s_{13} & \dots & s_{1r} \\ 0 & s_{22} = 1 & s_{23} & \dots & s_{2r} \\ 0 & 0 & s_{33} = 1 & \dots & s_{3r} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & s_{rr} = 1 \end{bmatrix}$$

Solution of S

- The square sub-system S :

$$\begin{bmatrix} s_{11} & s_{12} & s_{13} & \dots & s_{1r} \\ 0 & s_{22} = 1 & s_{23} & \dots & s_{2r} \\ 0 & 0 & s_{33} = 1 & \dots & s_{3r} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & s_{rr} = 1 \end{bmatrix} \begin{bmatrix} f'_1 \\ f'_2 \\ f'_3 \\ \dots \\ f'_r \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- Gives:

$$\begin{aligned} f'_1 &= 0 \\ s_{12}f'_1 + f'_2 &= 0 \\ &\dots \\ s_{1r}f'_1 + s_{2r}f'_2 + \dots + f'_r &= 0 \end{aligned}$$

Solution of S

- The system has an unique solution relativeley to r variable f'_1, f'_2, \dots, f'_r which is the degenerated solution .
- Then the solutions, upon the form f' are such as the first r components of the vector are ever equal to zero.
- In contrast, the $n - r$ other components can be freely chosen.

Vectorial space

- The obtention of the vectorial space is given with the choice of the solutions for which one and only one in the components of f' is not null.
- goal it is the computing of F which gives the base of the conservative components.

Triangularisation

- Let's suppose we the way of the triangularisation of the system and that the $i - 1$ lines and the $i - 1$ columns have been triangularised-:

$$\begin{bmatrix} 1 & c_{21} & \dots & c_{i1} & \dots & c_{m1} & 0 \\ 0 & 1 & \dots & \dots & \dots & c_{m2} & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & c_{ii} & \dots & c_{mi} & 0 \\ 0 & 0 & 0 & c_{i(i+1)} & \dots & c_{m(i+1)} & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & c_{in} & \dots & c_{mn} & 0 \end{bmatrix}$$

- Considering the column i : all the $c_{i(i+1)} \dots c_{in}$ must be cancelled.

Triangularisation

- To cancel these coefficients, we compute:

$$\forall k \in [i + 1, n] \quad \text{Ligne}_k \leftarrow \text{Ligne}_k * c_{ij} - \text{Ligne}_i * c_{ki}$$

- if $c_{ij} = 0$ Then
We realize the permutations into the $m - i$ columns of C to obtain a non null c_{ij} .
- if all the elements are null, the line i is nulle. Then f_i has been eliminated of the system, and will produce an element of the vectorial space of the solutions.
- In this case we permute the $n - i$ lines of C to make appear a non null line.
- if we find one: we permute the columns to have a non null c_{ij} .
- or else : the algo has ended: C is transformed in C' , and F give us the base of the vectorial space of solutions.

Change of variables

$$\forall k \in [i + 1, n] \quad \text{Ligne}_k \leftarrow \text{Ligne}_k * c_{ii} - \text{Ligne}_i * c_{ki}$$

- This operation we C is memorized we the vector column by the following change:

$$\begin{aligned} f_k &= c_{ii} \cdot f'_k \\ f_j &= f'_j \quad \forall j \neq i, k \end{aligned}$$

Change of variables

- for a column j of C :

$$c_{1j}.f_1 + \dots + c_{ij}.f_i + \dots + c_{kj}.f_k + \dots + c_{nj}.f_n = 0$$

- with the change of variable :

$$c_{1j}.f'_1 + \dots + c_{ij}.(f'_i - c_{ki}f'_k) + \dots + c_{kj}.c_{ii}f'_k + \dots + c_{nj}.f'_n = 0$$

- we develop c_{ij} and we factorise f'_k :

$$c_{1j}.f'_1 + \dots + c_{ij}.f'_i + \dots + (c_{kj}.c_{ii} - c_{ij}.c_{ki})f'_k + \dots + c_{nj}.f'_n = 0$$

- for the column $i \rightarrow j = i$: we verify that f'_k is eliminated of the equation.

Successives transformations of f

- In the beginning f is an identity matrix.
- Then f subit the transformations effectués we C .

$$\left\{ \begin{array}{l} f_k = c_{ii} \cdot f'_k \\ f_i = f'_i - c_{ki} \cdot f'_k \\ f_j = f'_j \end{array} \right. \quad \forall j \neq i, k \quad \begin{bmatrix} f_1 \\ f_i \\ f_k \\ f_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -c_{ki} & 0 \\ 0 & 0 & c_{ii} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f'_1 \\ f'_i \\ f'_k \\ f'_n \end{bmatrix}$$

- the matrix F is subject to several change of variables.

Exercise

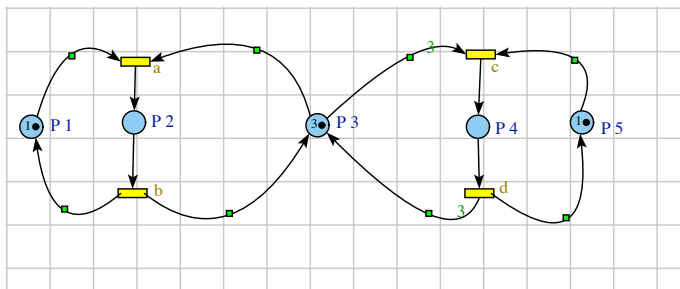


Figure: t is neutral

Let's find the conservative components and the invariants of places for the initial marking.

Simplified algorithm for positive solutions

- Only the computing of the columns of the matrix F corresponding to the $n - r$ components of the base the solutions of f is usefull. The other columns can be erased.
- At each step of the elimination of a variable:
 - ▶ the column of C which has allowed to eliminate a variable is erased
 - ▶ and the corresponding line is erased.
 - ▶ moreover the line in F is also erased.
- f memorizes the computed operations.

Simplified algorithm for positive solutions

- The goal is to realize an algorithm which favour in a first step, the positive linear combinations in the erasing of variable :
 $f_i + n.f_j$ with $n > 0$.
- we examine the columns of C and the signs the termes of theses columns.
- This test allows to distinguish four cases that we examine par increasing order.
- According to the case, we realize a specific action and we iter the examination of C .

Simplified algorithm for positive solutions

Step 1

We search a column with **only one** non null term:

- 1 we erase of C the line that is associated at the non null variable.
- 2 we erase of C the column that is associated at the non null variable.
- 3 we erase of F the column that is associated at this variable.

We iter this step then we go to the step 2.

Simplified algorithm for positive solutions

Step 2

- We search a column with **only on term** non nul with a given sign.
- If this column does not existe let's go in step 3.
- All the other terms are null or opposite signs.
- We erase then of C the line that is associated at this variable by positive combinaisons of lines of type : $Ligne_k * c_{ij} + Ligne_i * c_{ki}$

Back to step 1.

Simplified algorithm for positive solutions

Step 2 : Erase operations

- 1 We erase of C the line that is associated to this variable.
- 2 We erase of C the column that is associated to this variable.
- 3 We memorize in the vector f the linear combinaison that has been used.

Simplified algorithm for positive solutions

Step 3

- We search a column with $i \geq 2$ strictly positive components and $j \geq 2$ strictly negative components.
- if such a column does not exist go to step 4.
- With the help of a positive component, let erase the $j - 1$ negative components (step 2)
- Then it remains **only on** negative component and a set of components positives that we are going to erase with the step 2.

Remark :

We have chosen a positive component among i and a component among j : There may be $i * j$ different solutions.

Simplified algorithm for positive solutions

Step 4

- We now search a column with all the components that non null and of the same sign.
- We carry on the triangularisation with the combinaisons that are not positives.
- Now : either we can go to step 1: it exists a column with a unique non null component,
- Either the algorithm stop and then:
 - ▶ Either all the columns are null
 - ▶ Either there is no column anymore in the matrix
- the vector F gives the base of the solutions.

Exercise 1

Find the conservative components and the invariant of places with the Simplified algorithm for positive solutions for the following example:

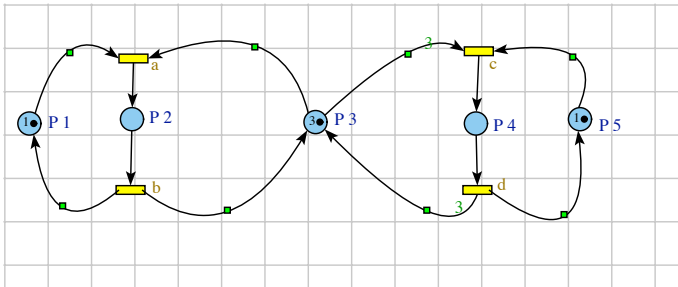


Figure: t is neutral

Exercise 2

Find the conservative components and the invariant of places with the Simplified algorithm for positive solutions for the following example:

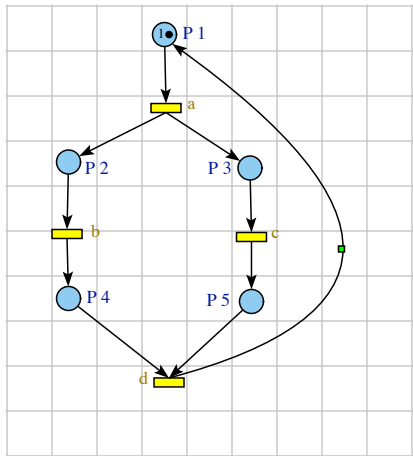


Figure:

Exercise 3

Find the conservative components and the invariant of places with the Simplified algorithm for positive solutions for the following example:

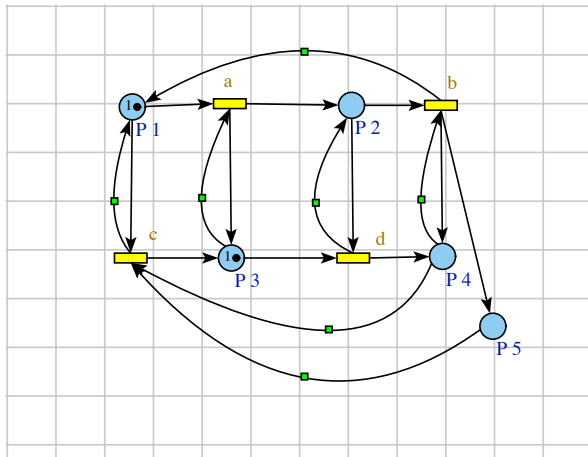


Figure:

Relations between the good properties

UNBOUNDED Marked PN $\not\Rightarrow^{MG}$ liveness or reinitialisability

- Place invariants allow to show that certain places are bounded.
- Transition Invariants are not sufficient conditions of bounded region : the transitions belonging to the invariant may not be reachable.

Cover of conservative componants

Cover of conservative componants (*CCC*)

It exists a *Cover* of conservative componants (resp. repetitive stationnary) if can find a *positive* component or a set of elementary *positives* components which cover all the places (resp. transitions).

Cover of places

A RdP for which it exists a cover of conservative componants $f^T > 0$ is k-bounded **whatever its initial marking**

Cover of conservative components

Calcul the k-bornes

The linear forme $f^T.M = f^T.M_0$ allows to compute a bound for all the places belonging to a cover of conservative components.

- For the places p of a *CCC* we have the property :

$$\forall p \in P, f(p).M(p) \leq f^T.M_0$$

- Thus because p belongs to the *CCC* :

$$f(p).M(p) \leq f'^T.C' = f^T.M_0$$

- and then :

$$M(p) \leq \frac{f^T.M_0}{f(p)}$$

Cover of conservative componants

Remark : A RdP can be bounded for a marking without CCC like in this example:

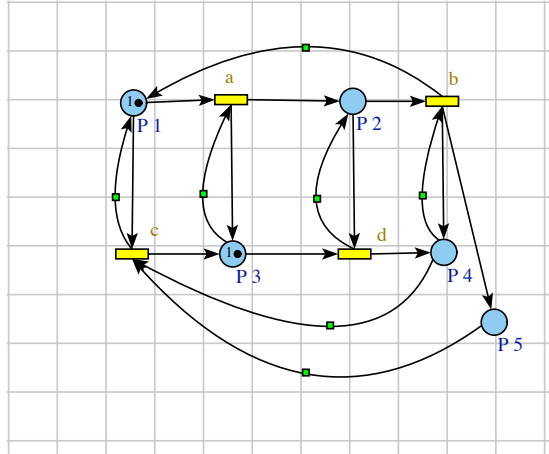


Figure: CCC

boundedness, liveness and Cover of transition

Cover of transition and the good properties

Every PN which is in the same time alive, bounded for an initial marking is such that a cover of stationnary repetitive transition \bar{s} exists.

Indeed:

- Bounded \Rightarrow Finite Number fini of marking that enable a given transition t .
- Alive \Rightarrow sequences of finite lenght
- Alive + bounded \Rightarrow it exists a loop passing through $t \Rightarrow$ it exists a stationnary repetitive transition.

La reciprocal is false :

One only has to take a null initial marking and the PN is no more alive as no transitions is firable.

Complementarity between the analyze by reduction and the calcul of the components

- Let's consider a PN which does not have the good properties.
- The method by reduction is not able to bring a diagnostic to the anomalies : non boundedness or deadlock
- The computing of the CCC or stationnary repetitive transition allow to determine the places which are bounded. The non bounded places does not belong to theses components.
- For the non alive transitions : no conservative component nor repetitive stationnary positives pass through par them.

Characterization of the marking

The different ways to characterize the markings :

- Enumeration of reachable the reachable markings.
- Fundamental equation : we obtain a set which includes the set of the reachable markings : we can find a sequence such as $\bar{s} > 0$ but s is not necessarily firable.
- the set the *CCC* which lui aussi inclut the deux ensembThe précédents comme go the illustrer the example suivant.

Exercice

- Find $A(R, M_0)$
- Find the set the vectors \bar{s} which give a marking $M' > 0$ at partir of marking initial.
- Find the set the reachable marking by the fundamental equation
- Find the conservative componants

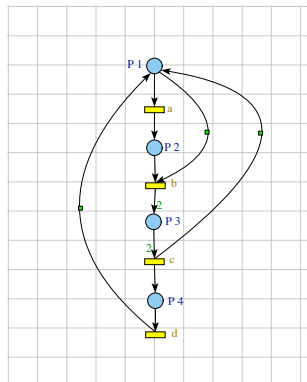


Figure: Characterization of the markings

Characterization of the markings

- The reachable markings constitutes the smallest set.
- The fundamental equation does not give any information about the reachability of markings. This equation furnish an overset of reachable markings.
- The *CCC* are an overset of the fundamental equation.

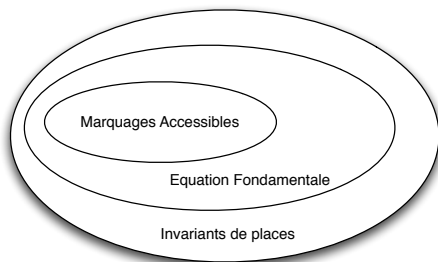


Figure: Characterization of the markings

Racket

Language :

- Write script shell ;
- Prototype a complex animation, GUI including regular expression, threads.
- Use of classes, modules, components.
- Use of files, flux, ..
- Compilation, cross-compilation.

Racket et LISP

- *LISP* : Artificial Intelligence ¹
- LISP functional language
- Racket recent evolution of LISP
- MacOS, LINUX and Windows
- Prefixed Language with parenthesis.
 - ▶ $f(a, b, c, d)$ s'crit $(f\ a\ b\ c\ d)$
 - ▶ $1 + 2 + 3 + 4$ s'crit $(+ 1\ 2\ 3\ 4)$

¹McCarthy, 1958, "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I ", CACM

Language Syntax : EBNF

<i>expression</i>	→	<i>atom</i> <i>list</i>
<i>atom</i>	→	<i>number</i> <i>identifier</i> <i>string</i> <i>character</i> <i>operator</i>
<i>list</i>	→	(<i>expression</i> *)

Numbers

- Exact number:
 - ▶ 17 ou 5,999999999999999999
 - ▶ rational number : $1/2$, $-3/4$
- non exact number:
 - ▶ Real: 2.0, 3.14e+87
 - ▶ +inf.0, -inf.0,
 - ▶ +nan.0, -nan.0
 - ▶ Complex : 2.0+3.0i

Boolean

#t et #f

Examples:

```
> (= 2 (+ 1 1))
```

```
#t
```

```
> (boolean? #t)
```

```
#t
```

```
> (boolean? #f)
```

```
#t
```

```
> (boolean? "no")
```

```
#f
```

```
> (if "no" 1 0)
```

```
1
```

The characters

A character is prefixed by #

Non printable charcaters are prefixed by #\ u

Examples:

```
> #\A
```

```
#\A
```

```
> #\u03BB
```

```
#\λ
```

```
> (display #\A)
```

```
A
```


String

A string is defined by "

Examples:

```
> "Apple"  
"Apple"  
> (display "Apple")  
Apple
```

Symbol

Symbol

A Symbol is an object which can represent:

- a constant,
- a variable,
- a function.

The evaluation of a symbol gives its value. The quote prevent the evaluation of the symbol.

Symbol Racket

- A Symbol corresponds to an atomic value.
- The external representation of a symbol is string character chain of characters : The identifier.
- The identifier is prefixed by a single quote '.
- string->symbol : gives the identifier (string character)
- Two symbols can be compared in a performant way by eq?

Examples:

```
> 'a
'a
> (symbol? 'a)
#t
> (define a 3)
> a
3
> (define b (+ a 3))
> b
6
> (+ a 7)
10
> (define (c p) (- 1 p))
> (c 3)
-2
> (c a)
-2
> (c b)
-5
```

Pairs and lists

Pair

A pair is a link between two elements:

$$(a . b)$$

- *cons* is the operator making a pair,
- *car* extracts the first element *a*,
- *cdr* extracts the second element *b*.

ExampleThe of pairs

Examples:

```
> (cons 1 2)
(1 . 2)
> (cons (cons 1 2) 3)
((1 . 2) . 3)
> (car (cons 1 2))
1
> (cdr (cons 1 2))
2
> (pair? (cons 1 2))
#t
```

the lists

List

A list is either the empty list `()`, null ou empty), either a singular pair such as the *car* is an element and the *cdr* is a list.

list is also an operator making lists with several elements.

Examples: lists

```
> null
()
> (cons 3 null)
(3)
> (list 'a 'b 'c)
(a b c)
> (define L1 (list 'a 'b 'c 'd))
> L1
(a b c d)
> (define P '(P0 P1 P2))
> P
'(P0 P1 P2)
> (car P)
'P0
> (cdr P)
'(P1 P2)
```


Examples: lists

```
> (cons P T)
'((P0 P1 P2) T0 T1 T2)
> (cons P (list T))
'((P0 P1 P2) (T0 T1 T2))
> (define PT (cons 1 (list T)))
> PT
'((P0 P1 P2) (T0 T1 T2))
> (car PT)
'(P0 P1 P2)
> (cdr PT)
'((T0 T1 T2))
> (cadr PT)
'(T0 T1 T2)
```

Functions on lists

append, reverse, member, equal?, eq? :

```
> (append P T)
'(P0 P1 P2 T0 T1 T2)
> (length PT)
2
> (list-ref P 0)
'P0
> (reverse PT)
'((T0 T1 T2) (P0 P1 P2))
> (member 'P1 P)
'(P1 P2)
> P
'(P0 P1 P2)
> (member 'P4 P)
#f
> (equal? '(p1 p2) '(p1 p2))
#t
> (eq? '(p1 p2) '(p1 p2))
#f
```

Basics functions

length, list-ref:

```
> (length P)
```

```
3
```

```
> (list-ref P 0)
```

```
'P1
```

PN structure

```

> P
' (P1 P2 P3)
> T
' (T1 T2 T3 T4)
> iP
' ((P1 T1) (P2 T2 T4) (P3 T3))
> Pi
' ((P1 T2) (P2 T1 T3) (P3 T4))
> iT
' ((T1 P2) (T2 P1) (T3 P2) (T4 P3))
> Ti
' ((T1 P1) (T2 P2) (T3 P3) (T4 P2))
> M_0
' (0 3 0)
> Arc
' ((P2 T1 1) (T1 P1 1) (P1 T2 1) (T2 P2 1) (P2 T3 3) (T3 P3 1)
(P3 T4 1) (T4 P2 3))
>

```

The functions

Function

The functions can be considered as datas or parameters. For example, it is possible to write functions which take functions as parameters and which may return functions.

Syntax : (define (f1 a b c d f2) ... Body of the function...)

The functions

```
>(define (print E)
  (for ([elt E])
    (printf "~a " elt)))
>(print P)
(P1 P2 P3)
```

The functions

Example :

sort has a a sequence and a comparaison operator in pa

```
> (sort '(1 3 4 2) <)
(1 2 3 4)
> (sort '("aardvark" "dingo" "cow" "bear") string<?)
("aardvark" "bear" "cow" "dingo")
> (sort `((9 a) (3 b) (4 c))
      (lambda (x y) (< (first x) (first y))))
((3 b) (4 c) (9 a))
> (define (twice f v)
      (f (f v)))
> (twice sqrt 16)
2
```

Lambda function

Lambda function

Lambda function is an anonymous function either at local use or unique use.

It is defined in the body of a function and its definition is then valid for the entire body of the function. In interpreted mod, the use is unique.

Lambda function

```
> ((lambda (x) x) 10)
10
> ((lambda (x y) (list y x)) 1 2)
(2 1)
> ((lambda (x [y 5]) (list y x)) 1 2)
(2 1)
> ((lambda (x [y 5]) (list y x)) 1)
(5 1)

> (twice (lambda (s) (string-append s "!"))) "hello"
"hello!!"
```

The structures

```
( define-struct pair (x y) )
```

pair is defined as a structure. This definition induce the functions:

- *pair* – *x* et *pair* – *y* which access to the members of the structures
- *make* – *pair* build
- *pair?* return a boolean

ExampThe :

```
> ( define-struct pair (x y) )
> (define C1 (make-pair 5 12))
> (pair-x C1)
5
> (pair C1)
#t
> (pair C2)
#f
```

Conditional structure

```
( if expr expr expr )  
( and expr )  
( or expr )  
( cond {[expr expr]}* )
```

IF

Example:

```
> (if (> 2 3) "greater" "lower")  
"plus petit"
```

AND

Example:

```
(define (sup a b)
  (if (and (number? a) (number? b))
      (if (< a b) (display "greater") (display "lower"))
      (display "not comparable")))
> (sup 2 5)
greater
> (sup 2 "toto")
not comparable
```

COND

```
(cond  [... ]
      [... ]
      [else ...] )
```

Example :

```
(define (analyseRdP P T)
  (cond
    [(and (empty? P) (empty? T)) (print "Null Rdp") ]
    [(or (empty? P) (empty? T)) (print "Degenerated PN")]
    [else (printf
            "there is ~a places and ~a transitions"
            (length P)
            (length T))]))
```

Recursivity

Recursive Function

The definition of f is recursive if its definition contains a call of f .

Example (EX0 2) :

```
(define (membre elt E)
  (cond
    [(empty? E) #f]
    [(equal? (car E) elt) #t]
    [else (membre elt (cdr E))]))
```