INTRODUCTORY TUTORIAL

Petri Nets

Organised by

Gianfranco Balbo
Jörg Desel
Kurt Jensen
Wolfgang Reisig
Grzegorz Rozenberg
Manuel Silva

June 2000
# Table of Contents

*Wolfgang Reisig*
An Informal Introduction to Petri Nets ........................................ 1

*Grzegorz Rozenberg*
Elementary Net Systems .............................................................. 5

*Jörg Desel*
Place/Transition Nets I ............................................................... 85

*Kurt Jensen*
Coloured Petri Nets ........................................................................ 111

*Grzegorz Rozenberg*
Behaviour of Elementary Net Systems .............................................. 161

*Manuel Silva*
Place/Transition Nets II ................................................................. 199

*Gianfranco Balbo*
An Introduction to Generalised Stochastic Petri Nets ............................ 217
An Informal Introduction
to
Petri Nets

Running Example:
A Producer/Consumer System

W. Reisig
Humboldt University of Berlin

A Programming Notation

P1: do forever
    if buffer = empty then
        buffer := filed
    end
P2: do forever
    if buffer = filed then
        buffer := empty
    end.

An Action Based Representation

\[
\begin{align*}
\text{producer} &= \text{p.d.producer} \\
\text{consumer} &= \text{r.c.consumer} \\
\text{buffer} &= \text{d.r.buffer} \\
\text{producer} \parallel \text{buffer} \parallel \text{consumer}
\end{align*}
\]
Neither State Based
Nor Action Based
But Well Ballanced

Two Consumers

Two Buffer Cells

What About 100 Buffer Cells?

so far: elementary
net
systems

... a place/transition net
Arc Weights of Place/Transition Nets

so far: place/transition nets

Producing and Consuming Objects of Sort a

... a high-level net

Producing and Consuming Objects of Sort a or b

Producing and Consuming Any Kind of Items
How Long Does It Take to Produce One Item?

... a timed Petri net

Stochastic Conflict Resolution

... a stochastic Petri net

hence the structure of the tutorial:

G. Rozenberg:
Elementary Net Systems
I today
II tomorrow

J. Desel / M. Silva:
Place / Transition Nets
I today
II tomorrow

K. Jensen:
High Level Nets
I today
II tomorrow

S. Donatelli:
Timed and Stochastic Nets
I tomorrow
II tomorrow
The area of Petri Nets was initiated by C.A. Petri in early 60's.
The chief attraction of this area is the way in which the basic aspects of distributed systems are identified both conceptually and mathematically.

In our lecture we will illustrate this point using the most fundamental class of Petri nets called elementary net systems.
The guiding principles of net theory in formulating the basic notions of states and changes-of-states (called transitions) are:

1. States and transitions are two intertwined but distinct notions that deserve an even-handed treatment.
2. Both states and transitions are distributed entities.
3. The extent of change caused by a transition is fixed; it does not depend on the state at which it occurs.
4. A transition is enabled to occur at a state iff the fixed extent of change associated with the transition is possible at that state.

BASIC LEVEL OF SYSTEM DESCRIPTION:

• atomic (local) states
• atomic (local) transitions
• conditions
• events

\[ B \cap E = \emptyset : \text{states and transitions are distinct entities} \]

• distributed (global) state case
  set of conditions holding concurrently
• distributed (global) transition step
  set of events occurring concurrently

• transition relation
  specifies how cases are transformed into cases by the occurrences of steps
Key questions:
(1) When can a step occur (concurrently) at a case?
(2) What is the resulting case when a step occurs at a case?

The answers within net theory are given by postulating fixed neighbourhood relationship, flow relation, between the conditions and the events: "structural" transition relation relating potential cases to potential cases via potential steps.

by adding an initial case:
potential $\rightarrow$ actual

NETS
**Definition**

A net is a triple \( N = (S, T, F) \)

1. \( S \cup T \neq \emptyset \) and \( S \cap T = \emptyset \)
2. \( F \subseteq (S \times T) \cup (T \times S) \)
3. \( \text{dom}(F) \cup \text{ran}(F) = S \cup T \)

\[ \sim \sim \]

\[ \text{dom}(F) = \{x \in S \cup T : (x, y) \in F \text{ for some } y \in S \cup T\} \]

\[ \text{ran}(F) = \{y \in S \cup T : (x, y) \in F \text{ for some } x \in S \cup T\} \]

\[ \sim \sim \]

\( S_N \quad S \quad S\text{-elements (of } N) \)

\( T_N \quad T \quad T\text{-elements (of } N) \)

\( X_N \quad S \cup T \quad \text{elements (of } N) \)

\( F_N \quad F \quad \text{flow relation (of } N) \)

---

A net is an ordered bipartite directed graph without isolated nodes.

Hence there is a nice graphical notation to represent nets:

- \( S \)-elements drawn as circles \( \bigcirc \)
- \( T \)-elements drawn as boxes \( \square \)
- flow relation drawn as edges \( \rightarrow \)
\[
N = (S, T, F)
\]

\[
S = \{s_1, \ldots, s_5\}, T = \{t_1, \ldots, t_4\}
\]

\[
F = \{(s_1, t_2), (s_2, t_2), (s_3, t_1), (s_5, t_4), (s_4, t_3), (t_2, s_3), (t_2, s_2), (t_4, s_5), (t_4, s_4), (t_3, s_1)\}
\]

\[
x = \{y \in X_N : (y, x) \in F_N\}
\]

\[
x' = \{y \in X_N : (x, y) \in F_N\}
\]

\[
Y = \bigcup_{x \in Y} x', Y' = \bigcup_{x \in Y} x',
\]

\[
t_2 = \{s_4, s_2\}, t_2' = \{s_3\},
\]

\[
t_3 = \{s_4\}, t_3' = \{s_4\}
\]

\[
\{t_2, t_3\} = \{s_4, s_2, s_4\},
\]

\[
\{t_2, t_3\}' = \{s_4, s_3\}\]
**Definition**

Nets $N_1 = (S_1, T_1, F_1)$, $N_2 = (S_2, T_2, F_2)$ are isomorphic, $N_1 \equiv N_2$, iff there exist bijections $\alpha : S_1 \to S_2$, $\beta : T_1 \to T_2$ such that

\[ \forall p \in S_1 \quad \forall t \in T_1 \]

\[ (p, t) \in F_1 \iff (\alpha(p), \beta(t)) \in F_2 \]

\[ (t, p) \in F_1 \iff (\beta(t), \alpha(p)) \in F_2 \]

\[ \forall t \in T_1 \]

\[ \alpha(t^*) = \beta(t) \quad \& \quad \alpha(t) = \beta(t^*) \]

---

**Definition**

A net $N$ is:

- pure iff $\forall x \in X_N \quad [x \cap x^* = \emptyset]$

- simple iff $\forall x, y \in X_N$

\[ (x = y \land x^* = y^*) \Rightarrow (x = y) \]

---

pure:

---

simple:
Depending on applications various interpretations can be given to the elements of a net. We will use a net to represent the (static) underlying structure of a distributed system.

Conditions represented by $S$-element (local states)

Events represented by $T$-element (local transitions)

Neighbourhood represented by $F$ relationship

Accordingly:

$N = (B, E, F)$

$e \in E$: pre-conditions of $e$

$e'$: post-conditions of $e$
Note that if $C \subseteq \mathbb{N}$ then $\text{ene} = \emptyset$.

Hence we often consider only pure nets (no loops).

Q: When can e occur at C?

A: e can occur at C if all pre-conditions hold at C(\{e\} \subseteq C) and no post-conditions hold at C(\{e\} \cap C = \emptyset).

$N = (B, E, F), \ C \subseteq B, \ e \in E$

$\{e\} \subseteq \mathbb{N}$
N = (B, E, F)  C ∈ B  e ∈ E

- e can occur at C

Q: What is the result of e occurring at C?

A: When e occurs at C, the pre-conditions of e cease to hold and the post-conditions of e begin to hold; the remaining part of the case remains unaffected (hence the resulting case C' is (C ∖ e) ∪ e').

\[ C[e] > C' \text{ iff } C - C' = e \text{ and } C' - C = e' \]

\[ C \xrightarrow{H(e)} e' \]

C' uniquely determined by C and e

\[ C_1 \xrightarrow{H_1(e)} e' \]

Hence the change of state produced by e does not depend on a global state in which it occurs!!!
$N = (B, E, F), C \subseteq B, U \subseteq E$

Q: When can the events in $U$ occur concurrently at $C$?

(When can the step $U$ occur at $C$?)

A: $U$ can occur at $C$ iff the events in $U$ can individually occur at $C$ without interfering with each other. $C \subseteq U \subseteq N$

Since the effect of an occurrence of the event $e$ is confined to $e \cup e^*$, the "non-interfering" can be formalized as:

$(\forall e_1, e_2)_U \left[ (e_1 \neq e_2) \Rightarrow \left( \cdot e_1 \cup e_1^* \right) \cap \left( \cdot e_2 \cup e_2^* \right) = \emptyset \right]$
because \( \{e_{1}, e_{2}, e_{3}\} \cap \emptyset = \emptyset \)

\( e_{3} \) and \( e_{6} \) interfere with each other
N = (B, E, F), C \subseteq B, U \subseteq T

U is enabled to occur at C

Q: What is the result of U occurring at C?

A: The result is the "sum" of the results of the events in U occurring individually at C (hence when U occurs at C the resulting case C' is given by:

\[ C' = (C - \cdot U) \cup U \]

1. \[ \{b_1, b_3, b_6\} [\{e_4, e_2\}]_N \{b_2, b_4, b_6\} \]

2. \[ \{b_2, b_4, b_9\} [\{e_3, e_6\}]_N \{b_5, b_6, b_7\} \]

---

**Definition**

Let N be a net and let U \subseteq E_N.

1. U is independent, \( \text{ind}_N (U) \), \( \iff \)
   \((\forall e_1, e_2) \_ U [\text{if } e_1 \neq e_2 \text{ then } (e_1 \cup e_1) \cap (e_2 \cup e_2) = \emptyset] \).

2. Let \( C \subseteq B_N \).
   U is a step enabled at C, \( C [U >_N, \text{iff } \text{ind}_N (U), U \equiv C \text{ and } U \cap C = \emptyset \).

3. Let \( C_1, C_2 \subseteq B_N \).
   U is a step leading from \( C_1 \) to \( C_2 \), \( C_1 [U >_N C_2, \quad \text{iff } \quad C_1 [U >_N \quad \text{and } \quad C_2 = (C_1 - \cdot U) \cup U] \).
**Theorem**

\[ N = (B, E, F), C, D \subseteq B, U \subseteq T. \]
Let \( \{U_1, U_2\} \) be a partition of \( U \) such that
\( (U_1, U_2 \neq \emptyset, U_1 \cap U_2 = \emptyset, U_1 \cup U_2 = U) \).

If \( C[U_N > D \}, then \exists Q \subseteq B \)
such that
\( C[U_1]_N > Q \) and \( Q[U_2]_N > D \)

\[
\begin{array}{c}
\text{C} \\
\downarrow \text{u} \\
\downarrow \text{D}
\end{array} \quad \Rightarrow \quad \begin{array}{c}
\text{Q} \\
\downarrow \text{u} \\
\downarrow \text{D}
\end{array}
\]
\( u_1 \rightarrow C \rightarrow u_2 \rightarrow u_1 \)

---

**Definition**

Let \( N \) be a net and let \( C \subseteq B_N \).
The forward case class generated by \( C, \{C\} \_N \}, is the smallest
subset of \( 2^{B_N} \) such that:

1) \( C \subseteq \{C\} \_N \).
2) If \( C_1 \subseteq \{C\} \_N \), and \( C_2 \subseteq B_N \)
is such that \( C_1 \cup U \subseteq C_2 \) for
some \( U \subseteq E_N \),
then \( C_2 \subseteq \{C\} \_N \).
\[ N = (B, E, F), \ C \in B, \]
\[ \tau = e_1 e_2 \ldots e_n \in E^+, \ n \geq 1 \]

Q: When can \( \tau \) occur at \( C \)?
A: \( \tau \) can occur at \( C \) iff
the events in \( \tau \) can individually occur in the order determined by \( \tau \)

\[ C[\tau]_N \]
\[ \exists C_0, C_1, \ldots, C_n \in B \text{ such that } \]
\[ C_0 = C \text{ and } \]
\[ \forall i \in \{1, \ldots, n\} \quad C_{i-1}[e_i]_N C_i \]

we write:
\[ C_0[e_1]_N C_1[e_2]_N C_2 \ldots C_{n-1}[e_n]_N C_n \]

Note that:
\( C_2 C_2 \ldots C_n \) uniquely determined by \( C \) and \( \tau \)
because
\[ C_0 - C_1 = e_1, \ C_1 - C_0 = e_1^* \]
\[ C_1 - C_2 = e_2, \ C_2 - C_1 = e_2^* \]
\[ \vdots \]
\[ C_{n-1} - C_n = e_n, \ C_n - C_{n-1} = e_n^* \]
\[ N = (B, E, F), C \subseteq B, e_1, e_2 \in E \]

If \( C \subseteq e_2 \) and \( C \subseteq e_2 \), then \( C \subseteq (e_1, e_2) \).

Q: What is the result of \( C \) occurring at \( e_2 \)?

A: The unique \( C_n \) such that \( C_n \subseteq (e_1, e_2) \), \( C_n \subseteq (e_1, e_2) \)
\[ \begin{align*}
. e_1 \cap e_2 &\subseteq C, \quad e_1 \cap C = \emptyset, \quad e_2 \cap C = \emptyset \\
\therefore (e_1 \cup e_2) \cap (e_2 \cup e_2) &= \emptyset \quad \text{iff} \\
. e_1 \cap e_2 &= \emptyset \\
. e_1 \cap Q_1 &= \emptyset \quad \text{and} \quad . e_2 \subseteq Q_1 \\
. e_1 \cap e_2 &= \emptyset \\
. e_1 \subseteq Q_1 \quad \text{and} \quad . e_2 \subseteq Q_1 \\
\therefore . e_1 \cap e_2 &= \emptyset \\
(1) \quad (2) \quad (3) \quad \Rightarrow \quad \text{ind}_N \{e_1, e_2\} \\
C[e_1] \quad \& \quad C[e_2] \quad \& \quad (4) \quad \Rightarrow \quad C\{e_1, e_2\}.
\end{align*} \]
THEOREM

$N = (B, E, F)$, $C, D \subseteq B$,
$\emptyset \neq U \subseteq E$.

(1) $C[U \triangleright_N D$ iff
$\forall$ ordering $e_1, ..., e_n$ of $U$
$C[e_1 ... e_n]_N > D$.

(2) $C[U \triangleright_N D$ iff
$\forall$ ordering $e_1, ..., e_n$ of $U$
$C[e_1 ... e_n]_N > D$.

sequentialization
property

ELEMENTARY NET SYSTEMS
**Definition**

An elementary net system (EN system for short) is a 4-tuple \( N^0 = (B, E, F, C_{in}) \) where 
\( (B, E, F) \) is a net called the underlying net of \( N^0 \), \( \text{und}(N) \), 
\( C_{in} \in B \) is the initial case of \( N^0 \), \( \text{inc}(N^0) \).

\[ C_{in} \]

We carry over to EN systems the notation and the terminology concerning nets 
\( B_{N^0} \), \( E_{N^0} \), \( F_{N^0} \).

- EN system as an abstract model of a distributed system:

\[ N^0 = (B, E, F, C_{in}) \]

- underlying static structure
- dynamic behaviour (actual state space)

\[ C_{in} \] the set of cases of \( N^0 \)

\[ U_{N^0} = \{ U \in E : (\exists C_1, C_2) \in C_{in} \} \]

\[ [ C_1 U \geq_{N^0} C_2 ] \]

the set of steps of \( N^0 \)
Graphical notation for an EN system consists of the graphical notation for the underlying net, and the marking of C in by tokens. Playing the token game we can compute $c_{Po}$ and $U$. 
Let $\mathcal{N}$ be an EN system, let $C \in \mathcal{E}_\mathcal{N}$, and let $e_1, e_2 \in \mathcal{E}_\mathcal{N}$.

$e_1, e_2$ can be related to each other at $C$ in (at least) three ways. 

**Sequence**

$e_1$ can occur at $C$ but not $e_2$.

However, after $e_1$ has occurred, $e_2$ can occur.

$e_1, e_2$ are in sequence at $C$ iff $C \leq e_1 \succ C \leq e_2 \succ$, and $C \leq e_2 \succ$, where $C \leq e_1 > C'$.

**Choice** (conflict)

$e_1$ and $e_2$ can occur individually at $C$ but they cannot occur together at $C$: $\{e_1, e_2\}$ is not a step at $C$. (Whether $e_1$ or $e_2$ will occur at $C$ is left unspecified: in this way $\mathcal{N}$ exhibits nondeterminism.)

$e_1, e_2$ are in conflict at $C$ iff $C \leq e_1 \succ$, $C \leq e_2 \succ$, and $\neg(C \leq \{e_1, e_2\})$.

---

![Diagram](image)
Concurrency

e₁ and e₂ can occur at C without interfering with each other. Moreover no order is specified over their occurrences. Hence, in general, the occurrences of events and the resulting holdings of conditions will be partially ordered: in this way P exhibits non-sequential behaviour.

e₁, e₂ can occur concurrently at C

\[ C \models \{e₁, e₂\} \]

A mixture of concurrency and conflict may result in a situation called confusion.

Confusion

\[ \{b₁, b₂, b₃\} \models \{e₁, e₂\} > \{b₅, b₆\} \]

\( Θ₁: \) first e₁ with no conflict, then e₂

\( Θ₂: \) first e₂ , then conflict between e₁, e₃, the conflict resolved in favour of e₃ which then occurred.
Let $N^0$ be an EN system, let $C \in C_{N^0}$.

- Let $e \in E$ be such that $C[e]$.
- The conflict set of $e$ (at $C$), $\text{cfl}(e, C)$, is the set
  \[ \{ e' \in E : C[e'] \text{ and } \neg (C[\{e, e'\}]) \} \].

- For $e_1, e_2 \in E$ such that $C[\{e_1, e_2\}]$, the triplet $(C, e_1, e_2)$ is a confusion (at $C$) if $\text{cfl}(e_1, C) \neq \text{cfl}(e_2, C_2)$, where $C[e_2] > C_2$.

- $N^0$ is confused at $C$ iff there is a confusion at $C$.

---

\[ C = \{b_1, b_2, b_3\} \]
\[ \text{cfl}(e_1, C) = \emptyset \]
\[ (C, e_1, e_2) \text{ is a confusion} \]
\[ \text{cfl}(e_1, C) = \emptyset \neq \{e_3\} = \text{cfl}(e_2, C_2) \]
\[ C_2 = \{b_2, b_3, b_4\} \]
One may have then conflict increasing confusions, conflict decreasing confusions, and confusions that are neither conflict increasing nor conflict decreasing.

\[ C = \{b_2, b_3\} \] is a confusion

\[ C_2 = \{b_2, b_3\} \]
STATE SPACES OF EN SYSTEMS
\[ N = (B, E, F), \ C \subseteq B \]

Let \[ F_S^N (C) = \{ \tau \in E^* : C[\tau] > N \} \]

Then
\[ [C > N = \{ D \in B : \exists \tau \in F_S^N (C) \text{ such that } C[\tau] > D \} \].

In particular, for \( \mathcal{N} = (B, E, F, C_{in}) \)
\[ C_{in} = [C_{in} > \mathcal{N} = \{ D \in B : \exists \tau \in F_S^\mathcal{N} (C_{in}) \text{ such that } C_{in}[\tau] > D \} \].

Thus \( SCG(\mathcal{N}) \) is strongly connected (connected from \( v_{in} \)).

The difference between \( SCG(\mathcal{N}) \) and \( CG(\mathcal{N}) \) is in labelled edges only.
\textbf{\textit{\textit{\Diamond}}-rule (for edge-labeled graphs where labels are sets)}:

\begin{itemize}
\item \textbf{\textit{\Diamond}}-rule
\end{itemize}

\textbf{\textit{\Diamond}}(g), is the graph h resulting from g by applying the \textbf{\textit{\Diamond}}-rule as long as possible.

\textbf{THEOREM}

\((\forall N) \left[ \text{\textit{\Diamond}}^*(SCG(N)) = CCG(N) \right] \)

The case graph of \(N\) can be "syntactically" recovered from the sequential case graph of \(N\) !!!
An isomorphism from $g_2$ onto $g_1$ is a pair of bijections $\phi: V_1 \rightarrow V_2$ and $\psi: \Delta_1 \rightarrow \Delta_2$ such that 

$$\phi(v_1) = v_2, \quad \phi(v'_1) = v'_2, \quad \phi(e_1) = e_2, \quad \phi(e'_1) = e'_2.$$ 

The problem at hand is to find such an isomorphism $g_1 \cong g_2$. 

**Diagram:**

- $g_1$: Graph with vertices 1, 2, 3, 4, 5, 6, 7, 8, and edges connecting them.
- $g_2$: Graph with vertices A, B, C, D, E, and edges connecting them.

For each vertex $v$ in $g_1$, there is a corresponding vertex $\phi(v)$ in $g_2$. Similarly, for each edge $e$ in $g_1$, there is a corresponding edge $\phi(e)$ in $g_2$. 

**Notes:**

- $\phi$ and $\psi$ are the bijections that map the vertices and edges of $g_1$ to those of $g_2$.
- The specific mapping is shown in the diagrams for $g_1$ and $g_2$. 

**Example:**

- $\phi(1) = A$, $\phi(2) = B$, etc.
- $\psi(A) = 1$, $\psi(B) = 2$, etc.
**DEFINITION**

EN systems $N_1, N_2$ are state space similar, $N_1 \cong N_2$, iff $CG(N_1) \text{lisom} CG(N_2)$.

**THEOREM**

Let $N_1, N_2$ be EN systems.

1) $CG(N_1) \text{lisom} CG(N_2)$
   iff
   $SCG(N_1) \text{lisom} SCG(N_2)$.

2) $N_1 \cong N_2$ iff $SCG(N_1) \text{lisom} SCG(N_2)$.

$inc(N_1) = \{3, 4, 5, 6\}$
SCG($N_2$)

$\text{inc}(N_2) = \{1, 2\}$
$\mathcal{N}_1 \cong \mathcal{N}_2$

$\varphi(\{3, 4, 5, 6\}) = \{1, 2\}$
$\varphi(\{1, 5, 6\}) = \{2, 3\}$
$\varphi(\{2, 3, 4, 6\}) = \{1, 4\}$
$\varphi(\{5, 7\}) = \{2, 5\}$
$\varphi(\{1, 2, 6\}) = \{3, 4\}$
$\varphi(\{3, 4, 6, 8\}) = \{1, 6\}$
$\varphi(\{2, 7\}) = \{4, 5\}$
$\varphi(\{1, 6, 8\}) = \{3, 6\}$
$\varphi(\{7, 8\}) = \{5, 6\}$

$\psi:
\begin{array}{cccccccc}
e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\
e_4 & e_3 & e_2 & e_6 & e_1 & e_5 \\
\end{array}$
SIMPLE EN SYSTEMS

An EN system $N^0$ is simple iff $N = \text{und}(N^0)$ is simple

\[(\forall x, y \in X_N) [\cdot x = \cdot y \land \cdot x^* = \cdot y^* \implies x = y] \]

---

In an EN system $N^0$ the change caused by an event occurrence is the same in every context:

\[(\forall e) E [\text{if } (C_1, e, C_2), (C_3, e, C_4) \in \mathcal{C}_G(H) \text{ then } (C_1 - C_2, C_2 - C_1) = (C_3 - C_4, C_4 - C_3)]\]
On the other hand we may have

\[ C \xrightarrow{?} Y \rightarrow D \]

and different events \( e_1, e_2 \) with \( e_1 = e_2 = X \) & \( e_1^* = e_2^* = Y \).

This cannot happen in an \( \leq \) (event) simple EN system.

**Theorem**

\( \mathcal{N} = (B, E, F, C_{\text{in}}) \) simple EN syst

\[ \forall e_1, e_2 \in B \quad \forall C_1, D_1, C_2, D_2 \in B \]

If \( C_1 \equiv e_1 > D_1 \) & \( C_2 \equiv e_2 > D_2 \),

then \( e_1 = e_2 \) iff

\[ C_1 - D_1 = C_2 - D_2 \quad \& \quad D_1 - C_1 = D_2 - C_2. \]

extensionality principle

Note that

\[ C_1 - D_1 = e_1^* \quad \& \quad D_1 - C_1 = e_1^* \]

\[ C_2 - D_2 = e_2^* \quad \& \quad D_2 - C_2 = e_2^* \]

Thus

\( e_1 = e_2 \) iff \((e_1, e_1^*) = (e_2, e_2^*)\)

For an event \( e \)

\((e, e^*)\) characteristic pair of \( e \)
**Reduced EN Systems**

$N$: $b_1 \rightarrow e_1 \rightarrow e_2 \rightarrow b_2 \rightarrow b_3 \rightarrow b_4$

$C_{in} = \{b_3\}$

$e_1$ useless, $e_2$ useful

$e \in E$ is useful iff

$\exists e \in E \left( C[e] \right)$,

otherwise $e$ is useless.

$N$ is reduced iff

$\forall e \in E$ e is useful.

Note that e is useful iff e is an edge-label in $SCG(N)$

"Mostly" it is assumed that an EN system is reduced.

**Theorem**

$(\forall e \in E) (\exists e \in E')$

$[N' is reduced \& N \approx N']$.
Various degrees of usefulness

\[ e \text{ is useful but not live} \]

\[ \text{e} \in E \text{ is live iff} \]
\[ \forall C \in \mathcal{C}_N \exists e \in E \exists D \in E^N \]
\[ C \cap D \land D \subseteq \{e\} \]

CONTACT-FREE EN SYSTEMS

In an EN system \( N \) an event \( e \) is enabled at a case \( C \) if \( e \in C \) (input concession) and \( e \cup C = \emptyset \) (output concession).

In a contact-free EN system the input concession suffices for an event to be enabled.

An EN system \( N \) is contact-free if \( (\forall e) (\forall C) \)
\[ N \]
\[ \left[ e \in C \text{ implies } e \cup C \neq \emptyset \right] \]
Conditions $b_1, b_2$ are complements of each other, i.e., $b_1 \equiv b_2$ if $b_1 \in \overline{E}$ and $b_2 \in \overline{E}$. If $N$ is (condition) simple and each $b \in \mathcal{B}$ has at most one complement.

Hence in a contact-free EN system, $\text{inc}(\mathcal{N}) = \{b_2, b_3\}$, $\mathcal{N}$ is not contact free.
CONSTRUCTION

Let \( N = (B, E, F, C_{in}) \) be an EN system.
Let \( \overline{B} \) be a set disjoint with \( B \cup E \),
and let \( \varphi: B \rightarrow \overline{B} \) be a bijection.
The \( S \)-complementation of \( N \) (relative to \( (\overline{B}, \varphi) \)) is the EN system
\( N' = (B', E', F', C'_{in}) \) such that
\[
\begin{align*}
B' &= B \cup \overline{B}, \quad E' = E, \\
F' &= F \cup \{(e, \varphi(b)) : e \in E \land (b, e) \in F\} \\
&\quad \cup \{(\varphi(b), e) : e \in E \land (e, b) \in F\}, \\
C'_{in} &= C_{in} \cup \varphi(B - C_{in}).
\end{align*}
\]

If you want to be thrifty,
you add \( \varphi(b) \) only to such \( b \in B \) that do not have \( \overline{b} \in B \).

It is easier to understand
the basic idea of this construction
if we assume
that no \( b \) in \( B \) has a complement in \( B \).

\[ \begin{array}{c}
b_1 \circ \quad \circ b_2 \\
\downarrow \quad \downarrow \\
b_3 \circ \\
\end{array} \sim \quad \begin{array}{c}
b_1 \circ \quad \circ b_2 \circ \circ \varphi(b_3) \\
\downarrow \quad \downarrow \quad \downarrow \\
\varphi(b_1) \quad \varphi(b_2) \\
\end{array} \]
Basic Feature:

∀c′ ∈ C′, ∀b ∈ B

[ beC′ iff b ∈ C′ ]

But

∀c′ ∈ C′, ∀b ∈ B

[ beC′ ]

and so

∀c′ ∈ C′, ∀e ∈ E

[ if e ∈ C′ then e ∩ C′ = φ ]

Hence

input concession implies

output concession

Thus

N′ is contact-free.
\[ N_2, N'_2 \text{ EN systems.} \]
\[ N_2 \text{ is a complementation of } N'_1 \iff N'_2 \text{ is the } S \text{-complementation of } N'_1 \text{ relative to } (B, \varphi) \]
for some \( B, \varphi \).

**THEOREM**

Let \( N'_1, N'_2 \) be EN systems such that \( N'_2 \) is a complementation of \( N'_1 \).

1. \( N'_2 \) is contact-free.
2. \( N'_1 \cong N'_2 \)

**THEOREM**

For each EN system there exists a state space similar EN system that is contact free.
BEHAVIOUR OF ELEMENTARY NET SYSTEMS

EN SYSTEM

\[ N = (B, E, F, C_{in}) \]

- underlying structure
- initial state (dynamic state space)
- static
- potential dynamics
- actual dynamics

\[ C_N \]

\[ U_N \]
ALL IS FINITE!

EN SYSTEMS ARE CONTACT-FREE

WHAT IS THE BEHAVIOUR?

HOW CAN THE BEHAVIOUR BE OBSERVED?
Sequential Observations

Observed are:

Occurrences of single events

$q \in E^*$ is a firing sequence iff
$q = \Lambda$
OR
$q = e_1 \ldots e_n \quad n \geq 1$
\[ e_1, \ldots, e_n \in E \]

WHERE
\[ (\exists C_0, C_1, \ldots, C_n \in \mathcal{C}_N ) \]

\[ C_0 \{ e_1 \} C_1 \{ e_2 \} \ldots \{ e_n \} C_n \]
\[ \equiv \]

$C_{in}$
To analyze $FS(n_p) = FS_{n_r}(c_{in})$, we need the sequential graph of $SCG(n_r)$ obtained by deleting all edges labeled by non-singleton steps from $CG(n_p)$. $C_{in} = \{b_1, b_2, b_3, b_4\}$.
\[ SCG(\mathcal{N}_1) \]

\[ FS(\mathcal{N}_1): \]

\[
e_1, e_4, e_2, e_5, e_3 \in \]

\[
e_1, e_3, e_5, e_4, e_1 \in \]

\[
e_1, e_2, e_4 \notin \]
(1) \( FS(N) \) is prefix closed
\[
\forall q_1, q_2 \in FS(N) : \quad q_1 \leq q_2 \Rightarrow q_1 \in FS(N)
\]
(2) \( SCG(N) \) is finite

**THEOREM**
\[
(\forall N) \quad [FS(N) \text{ is a prefix closed regular language}]
\]

**PROBLEMS !!!**
..... \( e_1 e_3 \) ..... \( e \)

is it a causal order?
is it only an observational order?
Is \( \{e_1, e_3\} \) a step?
HOW TO EXTRACT (RECOVER) CAUSAL ORDERS FROM SEQUENTIAL OBSERVATIONS?

THEORY OF TRACES (MAZURKIEWICZ)
\[ \varphi = \ldots e_1 e_2 \ldots \in E^*_{\mathcal{N}} \]
\[ \mu = \ldots e_2 e_1 \ldots \]
\[ (e_1, e_2) \in I_{\mathcal{N}}^\varphi \]
\[ \varphi \equiv_{I_{\mathcal{N}}} \mu \]
\[ \varphi \equiv_{I_{\mathcal{N}}} \mu \equiv_{I_{\mathcal{N}}} \gamma \ldots \equiv_{I_{\mathcal{N}}} \delta \]
\[ \varphi \equiv_{I_{\mathcal{N}}} \delta \]

\[ \equiv_{I_{\mathcal{N}}} \] is an equivalence relation on \( E^*_{\mathcal{N}} \).
$Z \leq E^*_W$ is $I^*_W$-consistent

IFF $Z$ is a union of equivalence classes of $\frac{*}{I_W}$.

An equivalence class is called a trace.
THEOREM

$(\forall \mathcal{M}, \mathcal{N}) \in \text{FS}(\mathcal{N})$, is $I^\mathcal{M}$-consistent.

IF $\mathcal{N}$ OR $\mathcal{M}$ OBSERVABLE IN $\mathcal{N}$, THEN EACH ELEMENT OF $\mathcal{N}$ OBSERVABLE IN $\mathcal{N}$.

Each equivalence class $[x]_{I^\mathcal{M}}$ is either included in $\text{FS}(\mathcal{M})$ or disjoint with $\text{FS}(\mathcal{N})$.

Those that are included are called (FIRING) TRACES $\mathcal{F}^\mathcal{M}$.

This $Z$ is not $I^\mathcal{M}$-consistent.
Sequential observation

- Firing sequences
  linear - difficult to interpret
  break them down to
  dependence graphs
  acyclic directed graphs
  ↓
  partial orders

7. \( \mathcal{N} \quad q \in \mathcal{FS}(\mathcal{N}) \)

The canonical dependence graph of \( q \) \( \langle q \rangle_{\mathcal{D}_{\mathcal{N}}} \)

(i) \( q = \bot \quad \rightarrow \quad \langle q \rangle_{\mathcal{D}_{\mathcal{N}}} \) is empty

(ii) \( q = e_1 \ldots e_n \), \( n \geq 1 \), \( e_1, \ldots, e_n \in \mathcal{E}_{\mathcal{N}} \)

\( \langle q \rangle_{\mathcal{D}_{\mathcal{N}}} \) is the \( \mathcal{E}_{\mathcal{N}} \)-lab. graph \((V, Y, \psi)\)

\( V = \{1, \ldots, n\} \),

\( (\forall i \in \{1, \ldots, n\}) \) \[ \psi(i) = e_i \]

\( (\forall i, j \in \{1, \ldots, n\}) \)

\[ (i, j) \in Y \quad \text{iff} \quad (i < j) \land (e_i, e_j) \in \mathcal{D}_{\mathcal{N}} \]
\( q = a \ b \ c \ a \ d \)

\( (c, a) \in D \quad (c, b) \in I \)

\( (a, b) \in D \)
The canonical dependence graph of $g$

$g = a,b,c,d$

$(d,c) \in D$

$(d,a) \in I$

$(c,b) \in I$

$(d,a) \in D$

$(c,a) \in D$

$(a,b) \in D$

$(a,b) \in D$
\( \varphi = \begin{array}{cccc} a & b & c & a \\ hline b & c & a & d \end{array} \) \quad (b, c) \in I

\( \varphi' = \begin{array}{cccc} a & c & b & a \\ hline b & c & a & d \end{array} \)

\[ \langle \varphi' \rangle_D \text{ isom } \langle \varphi \rangle_D \]

So

\[ \langle \varphi' \rangle_D = \langle \varphi \rangle_D \]

**THEOREM**

\( J^\mathcal{P} = (B, E, F, C_{in}) \) EN system,

\( \varphi_1, \varphi_2 \in E^x \).

\( \mathcal{T} \varphi \equiv \mathcal{T} \varphi' \text{ iff } \langle \varphi_1 \rangle_D \text{ isom } \langle \varphi_2 \rangle_D \)

\( \text{iff } \langle \varphi_1 \rangle_D = \langle \varphi_2 \rangle_D \)

\( \text{iff } [\varphi_1]_I = [\varphi_2]_I \)
Thus, if \( t \) is a firing trace, i.e., \( t = [q_1, \ldots, q_n] \) for some \( q \in FS(N) \), then each firing sequence from \( t \) is a sequential observation of (in general nonsequential) \( \langle \text{run ("process") } \rangle \) of \( N \).

Hence each firing trace is the set of all sequential observations of the same run of \( N \).

\[
E^*/E\overset{*}{=} I
\]

Every string from \( t \) will yield the same abstract dependence graph of \( t \), \( \text{adj}(t) \).
POSETS

\( (A, R) \)  
ANTISYM.
TRANSIT.
REFLEX.

\( g = (V, E) \)  
DIR.
ACYCL.
GRAPH

TRANS. \&
REFL. CLOS.

\( \leq_g = (V, E^*) \)

\( \varphi \in \Sigma^* \)

\(<\varphi>_D \quad <<\varphi>>_D \quad \overline{<\varphi>_D} \quad \overline{<<\varphi>>_D} \quad \text{adg}(t) \quad \text{alp}(t) \quad \text{ADG}(T) \quad \text{ALP}(T) \)
\[ \text{FS}(N) \subseteq E^* \]

\[ \text{FT}(N) \]

\[ \text{ADG(FT}(N)\text{))} \]

\[ \text{ALP(FT}(N)\text{))} \]

\[ \text{C_{in} = \{b_1, b_3\}} \]
I_\mathcal{N} = \{(e_1, e_5), (e_5, e_1),
           (e_1, e_4), (e_4, e_1),
           (e_3, e_5), (e_5, e_3)\}

D_\mathcal{N} = E \times E - I_\mathcal{N}

q = e_1 e_5 e_3 e_4 e_5
   \in \mathbf{FS}(\mathcal{N})

[q] =
   \begin{align*}
   &I_\mathcal{N} \\
   &\{ e_1 e_5 e_3 e_4 e_5 , \\
   &e_5 e_1 e_3 e_4 e_5 , \\
   &e_1 e_3 e_5 e_4 e_5 \} \\
   \end{align*}
$e_1 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5$

$\in AFLP(N)$

NON-SEQUENTIAL OBSERVATIONS
no conflicts in a run!
no cycles in a run
A net \( N = (S, T, F) \) is an occurrence net iff
\[
(\forall s \in S) \left[ |s|^2 \leq 1 \text{ and } |s|^2 \leq 1 \right]
\]
S-non-branching
\[
(\forall x, y \in X) \left[ (x, y) \in F^+ \implies (y, x) \notin F^+ \right]
\]
acyclic
\[
(\forall t \in T) \left[ t^* \neq \emptyset \right]
\]
A node-labeled occurrence net
\[ N = (S, T, F, \varphi) \]
\((S, T, F)\) occurs.
Net
\[ \varphi: S \cup T \rightarrow \Sigma \]
\[ \varphi(S) \cap \varphi(T) = \emptyset \]
\[ N^0 \text { EN SYSTEM } \]

\[ N = (S, T, H, \varphi) \text { NODE-LAB. } \]

\[ N \text { IS A PROCESS OF } N^0 \text { IF } \]

1. \( \varphi(S) \subseteq B_{N^0} \text { AND } \varphi(T) \subseteq E_{N^0} \).
2. \( (\forall s_1, s_2 \in S) [\varphi(s_1) = \varphi(s_2) \Rightarrow (s_1 \leq_N s_2) \lor (s_2 \leq_N s_1)] \)
3. \( (\forall t \in T) [\varphi(t^*) = \varphi(t) \text { AND } \varphi(t^*) = \varphi(t)] \).
4. \( \varphi(0N) \subseteq C_{in} \).

\[ P(N^0) \]
THEOREM
\( N \in \text{EN SYSTEM} \)
\( N = (S, T, F, \phi) \in P(N) \)
\( S' \subseteq S \) A SLICE OF N

\[ (\exists C \in \mathcal{C}_N)[\phi(S') \leq C]. \]
EN SYSTEM $\mathcal{NP}$ IS REDUCED IFF ALL EVENTS OF $\mathcal{NP}$ "VISIBLE" IN SCG($\mathcal{NP}$)

\[ E_{\mathcal{NP}} = \bigcup_{U \in \mathcal{U}} \mathcal{U} \]

63)
\[ G_1 \xrightarrow{\text{lisom}} G_2 \]
\[ \exists \delta : \Sigma_1 \rightarrow \Sigma_2 \]
\[ \exists \gamma : G_1 \rightarrow G_2 \]

EN SYSTEMS \( N_1^0, N_2^0 \)
STRUCTURALLY SIMILAR
\( N_1^0 \equiv N_2^0 \)
\( \text{und}(N_1^0) \text{ isom } \text{und}(N_2^0) \)

\( C_{in}^1, C_{in}^2 \) RELATED ACCORDINGLY
A bipartite graph $g = (V, W, F)$

$\text{W-contraction of } g$

$\text{ctr}_W(g) = (V, E_f)$

Diagram:

**Theorem**

$\mathcal{N}_2$, $\mathcal{N}_2^*$ reduced ENSSs $\mathcal{P}(\mathcal{N}_2^*)$ isom. $\mathcal{N}_2$

iff

$\mathcal{N}_2 = \mathcal{N}_2^*$

Process representation of the behavior of an EN system is too detailed.
A system $N$ with $N \in P(N)$ and $S = S_N$.

The $S$-contracted version of $N$ is a contracted process of $N$.

$CP(N)$
THEOREM
\[ (\exists N_1, N_2) \quad \text{EN REDUCED} \]
\[ [ \text{CP}(N_1) \overset{\text{LiSom}}{\cong} \text{CP}(N_2) ] \]
\[ N_1 \not\equiv N_2 \]
I. Introduction to place/transition-nets

An example
- Different features of place/transition-nets
- Place/transition-nets vs en-systems

Formal definitions
- Place/transition-nets
- Occurrence sequences and reachability
- Marking graphs

Behavioral properties
- Deadlock-freedom and liveness
- Boundedness and 1–safety
- Reversibility

 Capacities and complements
- Weak and strong capacities
- Weak and strong complements
- Inhibitor arcs
An example: a vending machine

Control structure of a vending machine

Adding concurrency: a vending machine with capacity 1 ...

... and its behaviour
Adding bounded storages: a vending machine with capacity 4 ...

... and its behaviour

Adding unbounded counters: the control part with a counter ...

... and its behaviour
Adding arc weights: the vending machine selling pairs ...

Adding limited capacities: replacing the place "request for refill" ...

... by a capacity restriction
Marked place/transition-nets generalize en-systems

Each contact-free en-system is a 1-safe marked place/transition-net

Terminology:

<table>
<thead>
<tr>
<th>en-system</th>
<th>marked p/t-net</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>place</td>
</tr>
<tr>
<td>event</td>
<td>transition</td>
</tr>
<tr>
<td>case / state</td>
<td>marking</td>
</tr>
<tr>
<td>$c \subseteq$ conditions</td>
<td>$m$: places $\rightarrow {0, 1}$</td>
</tr>
<tr>
<td>sequential case graph</td>
<td>marking graph</td>
</tr>
<tr>
<td></td>
<td>(reachability graph, state graph)</td>
</tr>
</tbody>
</table>

Formal definition of marked place/transition-nets

A **marked place/transition-net (p/t-net)** is a tuple $(S, T, F, k, w, m_0)$ where

$(S, T, F)$ is a net with

- $S$ – set of **places** (Stellen), nonempty, finite (often $P$ is used)
- $T$ – set of **transitions**, nonempty, finite
- $F \subseteq (S \times T) \cup (T \times S)$ – **flow relation**
- $k : S \rightarrow \{1, 2, 3, \ldots\} \cup \{\infty\}$ – **partial capacity restriction** (default: $\infty$)
- $w : F \rightarrow \{1, 2, 3, \ldots\}$ – **weight function** (default: 1)
- $m_0 : S \rightarrow \{0, 1, 2, \ldots\}$ – a **marking** satisfying
  \[
  \forall s \in S : k(s) = \infty \lor m_0(s) \leq k(s)
  \]

(initial marking )
The occurrence rule

A transition $t$ is **enabled** at a marking $m$ if

- every place $s \in t^\bullet$ satisfies $m(s) \geq w(s, t)$ and
- every place $s \in t^\bullet$ satisfies $m(s) + w(t, s) \leq k(s)$

The occurrence of $t$ leads to the **successor marking** $m'$, defined by

$$m'(s) = \begin{cases} m(s) & \text{if } s \not\in t^\bullet \text{ and } s \not\in t^\circ \\ m(s) - w(s, t) & \text{if } s \in t^\bullet \text{ and } s \not\in t^\circ \\ m(s) + w(t, s) & \text{if } s \not\in t^\bullet \text{ and } s \in t^\circ \\ m(s) - w(s, t) + w(t, s) & \text{if } s \in t^\bullet \text{ and } s \in t^\circ \end{cases}$$

**Notation:** $m \xrightarrow{t} m' (m[t]m')$

---

Occurrence sequences and reachability

A finite sequence $\sigma = t_1 t_2 \ldots t_n$ of transitions is a **finite occurrence sequence** leading from $m_0$ to $m_n$ if

$$m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} m_n$$

A marking $m$ is **reachable** (from $m_0$) if there is an occurrence sequence leading from $m_0$ to $m$

**Notation:** $[m_0]$ is the set of all reachable markings

An infinite sequence $\sigma = t_1 t_2 t_3 \ldots$ is an **infinite occurrence sequence** enabled at $m_0$ if

$$m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \xrightarrow{t_3} \cdots$$

[Diagram showing transitions and markings]
Marking graphs

The **marking graph** of a marked p/t-net is an edge-labeled graph with initial vertex

- **initial vertex** – initial marking $m_0$ (denoted $\bullet$)
- **vertices** – set of reachable markings $[m_0]$
- **labeled edges** – set of triples $(m, t, m')$ such that $m \xrightarrow{t} m'$

**Example:**

```
  s1 ---- s2 ---- s3
    \   \     /     /
     \  t3  t1
   t2
```

**Lemma** Each occurrence sequence corresponds to the labels of a directed path of the marking graph starting with the initial vertex, and vice versa.

Behavioral properties of marked p/t-nets

A marked p/t-net is

- **terminating** – if there is no infinite occurrence sequence
- **deadlock-free** – if each reachable marking enables a transition
- **live** – if each reachable marking enables an occurrence sequence containing all transitions
- **bounded** – if, for each place $s$, there is a bound $b(s)$ such that $m(s) \leq b(s)$ for every reachable marking $m$
- **1-safe** – if $b(s) = 1$ is a bound for each place $s$
- **reversible** – if $m_0$ is reachable from each other reachable marking

**Example** The vending machines are deadlock-free and live.

Some are 1-safe, some are bounded, some are unbounded.

The bounded vending machines are reversible.
A marked p/t-net which is not deadlock-free and its marking graph

Proposition  A marked p/t-net is deadlock-free if and only if its marking graph has no vertex without successor

Proposition  No deadlock-free marked p/t-net is terminating (but the converse does not necessarily hold)

A deadlock-free marked p/t-net which is not live

Proposition  Every live marked p/t-net is deadlock-free (this does not hold for nets without transitions)

Proposition  A marked p/t-net is live if and only if at no reachable marking a transition is dead (cannot become enabled again)

Example  Some transitions are dead at a reachable marking
**Proposition** A marked p/t-net is bounded if and only if  
its set of reachable markings is finite  
(its marking graph is finite)

**Proof**

\((\Leftarrow)\) The maximal number of tokens on a place can be taken as its bound.

\[(\Rightarrow)\] If a place \(s\) is bounded by \(b(s)\) then it can be in at most \(b(s) + 1\) different states, vic.

\[m(s) = 0, m(s) = 1, \ldots, m(s) = b(s).\]

So the number of reachable markings does not exceed

\[(b(s_1) + 1) \cdot (b(s_2) + 1) \cdot \cdots (b(s_n) + 1)\]

where \(\{s_1, s_2, \ldots, s_n\}\) is the (finite !) set of places

**Corollary** A 1–safe marked p/t-net with \(n\) places has

at most \(2^n\) reachable markings

---

**Proposition** A marked p/t-net is reversible if and only if  
its marking graph is strongly connected

**Example** a 1–safe non-live marked p/t-net which is not reversible

**Example** an unbounded marked p/t-net which is not reversible

**Example** a live and 1–safe marked p/t-net which is not reversible
Substituting capacities ...

... by complement places

Weak capacities

... guarantee bounds of places

weak enabling condition:

A transition $t$ is enabled at a marking $m$ if

- every place $s \in \bullet t$ satisfies $m(s) \geq w(s, t)$ and
- every place $s \in t^* \setminus \bullet t$ satisfies $m(s) + w(t, s) \leq k(s)$ and
- every place $s \in t^* \cap \bullet t$ satisfies $m(s) - w(s, t) + w(t, s) \leq k(s)$

Proposition If $k(s)$ is finite then $s$ is $k(s)$-bounded

Replacing a weak capacity restriction by a weak complement
Strong capacities

... generalize contact of en-systems

**strong enabling condition:**

a transition $t$ is enabled at a marking $m$ if

- every place $s \in \cdot t$ satisfies $m(s) \geq w(s, t)$ and
- every place $s \in t^\bullet$ satisfies $m(s) + w(t, s) \leq k(s)$

**Proposition** each en-system is equivalent to a

marked p/t-net without arc weights and

with the strong capacity restriction $k(s) = 1$ for every place $s$

Replacing a strong capacity restriction by a **strong complement**

Inhibitor arcs for null tests

**inhibitor enabling condition:** If $(s, t)$ is an inhibitor arc then

$t$ is only enabled at a marking $m$ if $m(s) = 0$

Replacing an inhibitor arc at a bounded place by a weak complement
II. Basic Linear-algebraic techniques

analysis

The marking equation

Place invariants

Transition invariants

Structural techniques

Siphons

Traps

The siphon/trap property

Restricted net classes

State machines

Marked graphs

Free-choice nets

Causal Semantics

Occurrence nets

Process nets

Linear-algebraic representation of markings and transitions

vector representation of the marking $m_0$: $\vec{m}_0 = (4, 0, 0, 0, 1)$

vector representation of the transition $t_2$: $\vec{t}_2 = (-1, 1, 1, 0, -1)$

\[ m_0 \xrightarrow{t_2} m_1 \Rightarrow \vec{m}_0 + \vec{t}_2 = \vec{m}_1 = (3, 1, 1, 0, 0) \]
Matrix representation of a net

![Diagram of a net with transitions and places]

**incidence matrix of the net:**

\[
\begin{pmatrix}
  \vec{t}_1 & \vec{t}_2 & \vec{t}_3 & \vec{t}_4 & \vec{t}_5 \\
  \vec{s}_1 & 1 & -1 & 0 & 0 & 0 \\
  \vec{s}_2 & -1 & 1 & 0 & 0 & 0 \\
  \vec{s}_3 & 0 & 1 & -1 & 0 & 1 \\
  \vec{s}_4 & 0 & 0 & 1 & -1 & -1 \\
  \vec{s}_5 & 0 & -1 & 0 & 1 & 0 \\
\end{pmatrix} = \{N\}
\]

The marking equation

\[
m_0 \xrightarrow{t_2 t_3 t_5 t_1 t_3} m \quad \Rightarrow \quad m_0 + \vec{t}_2 + \vec{t}_3 + \vec{t}_5 + \vec{t}_1 + \vec{t}_3 = \vec{m}
\]

\[
m_0 + (1 \cdot \vec{t}_1) + (1 \cdot \vec{t}_2) + (2 \cdot \vec{t}_3) + (0 \cdot \vec{t}_4) + (1 \cdot \vec{t}_5) = \vec{m}
\]

Parikh vector of \( t_2 t_3 t_5 t_1 t_3 \)

\[
\text{Parikh vector of } t_2 t_3 t_5 t_1 t_3
\]

**The Marking Equation**

If \( m_0 \xrightarrow{\sigma} m \) and \( \mathcal{P}(\sigma) \) denotes the Parikh vector of \( \sigma \) then

\[
m_0 + \{N\} \cdot \mathcal{P}(\sigma) = \vec{m}
\]

... yields a necessary condition for reachability of a marking:

A marking \( m \) is only reachable from \( m_0 \) if

\[
m_0 + \{N\} \cdot \vec{x} = \vec{m} \]

has a solution for \( \vec{x} \) in \( \mathbb{N}^* \).
Example: a live and 1-safe marked p/t-net

```
Example: mutual exclusion
```

Every reachable marking $m$ satisfies $m(s_2) + m(s_4) \leq 1$

1) $m(s_2) + m(s_3) + m(s_4) = 1$ holds initially
2) $m(s_2) + m(s_3) + m(s_4) = 1$ is stable — will be shown by a place invariant
3) $m(s_2) + m(s_3) + m(s_4) = 1 \Rightarrow m(s_2) + m(s_4) \leq 1$
Place invariants

Three equivalent definitions:

A **place invariant** of a net \( N \) is a vector \( \vec{i} \) satisfying

\[
\begin{align*}
(1) & \quad \sum_{s \in t^*} \vec{i}_s = \sum_{s \in t^*} \vec{i}_s \\
(2) & \quad \vec{i} \cdot \vec{t} = 0 \\
(3) & \quad \vec{i} \cdot (\mathcal{J}(N)) = (0, 0, \ldots, 0)
\end{align*}
\]

The token conservation law for a place invariant \( \vec{i} \):

If \( m \) is reachable from \( m_0 \) then \( \vec{i} \cdot m_0 = \vec{i} \cdot m \)

Proof: \( m_0 \xrightarrow{\sigma} m \Rightarrow m_0 + [N] \cdot \mathcal{P}[\sigma] = m \)

\[
\Rightarrow \vec{i} \cdot m_0 + \vec{i} \cdot [N] \cdot \mathcal{P}[\sigma] = \vec{i} \cdot m
\]

\[
\Rightarrow \vec{i} \cdot m_0 = \vec{i} \cdot m
\]

Proving stability

The number of tokens on \( \{s_2, s_3, s_4\} \) is not changed by transition occurrences.

\[
\Rightarrow \vec{i} = (0, 1, 1, 1, 0) \text{ is a place invariant.}
\]

\( \vec{i} \cdot m_0 = 1 \) implies \( \vec{i} \cdot m = 1 \) for each reachable marking \( M \).

\[
\Rightarrow m(s_2) + m(s_3) + m(s_4) = 1 \text{ is stable.}
\]
Further place invariants

(0, 1, 1, 0) mutual exclusion

(0, 1, 1, 0, -1) $m(s_2) + m(s_3) = m(s_5)$
if $s_2$ is marked then $s_5$ is marked

(1, 1, 0, 0, 0) $m(s_1) + m(s_2) = 1$
$m(s_1), m(s_2) \leq 1$, the places $s_1$ and $s_2$ are bounded

A necessary condition for liveness

Proposition: In a live marked p/t-net without isolated places,
each place invariant $\mathbf{i}$ without negative entries and
with some positive entry $i_s$ satisfies $\mathbf{i} \cdot \bar{m}_0 > 0$.

Proof: otherwise transitions in $s \cup s^\bullet$ are dead.

Examples: place invariants $(1, 1, 0, 0, 0), (0, 0, 0, 1, 1), (0, 1, 1, 1, 0)$
A sufficient condition for boundedness:

Proposition: Each marked p/t-net with a place invariant $\vec{i}$ satisfying $\vec{i}_s > 0$ for each place $s$ is bounded.

Proof: $m$ is reachable $\Rightarrow \vec{i} \cdot \vec{m} = \vec{i} \cdot \vec{m}_0$. 
$\Rightarrow \vec{i}_s \cdot \vec{m}_s \leq \vec{i} \cdot \vec{m} = \vec{i} \cdot \vec{m}_0$. 
$\Rightarrow m(s) = \vec{m}_s \leq \frac{\vec{i} \cdot \vec{m}_0}{\vec{i}_s}$

Example: place invariant $(1, 2, 1, 2, 1)$

Place invariants and the marking equation

Proposition There is a place invariant $\vec{i}$ satisfying $\vec{i} \cdot \vec{m}_0 \neq \vec{i} \cdot \vec{m}$ if and only if $\vec{m}_0 + (\mathcal{N}) \cdot \vec{x} = \vec{m}$ has no rational-valued solution for $\vec{x}$.

Example: 

$\vec{m}_0 + (\mathcal{N}) \cdot (1, 0, 1, \frac{1}{2}, \frac{1}{2}) = \vec{m} = (1, 0, 1, 0, 1, 1, 0, 0)$

$\Rightarrow$ no place invariant proves the non-reachability of $m$.

But the marking equation has no solution in $\mathbb{N}^*$

$\rightarrow \text{modulo place invariants}$
Transition invariants

A *transition invariant* of a net $N$ is a vector $\vec{j}$ satisfying
\[
[N] \cdot \vec{j} = (0, 0, \ldots, 0)
\]

Example:

Transition invariants: $(1, 1, 0, 0), (0, 0, 1, 1), (2, 2, 1, 1)$

**Proposition** Let $m_0 \xrightarrow{\sigma} m$ be an occurrence sequence.

$m_0 = m$ if and only if $P[\sigma]$ is a transition invariant

**Proof:** follows immediately from $m_0 + [N] \cdot P[\sigma] = m$

A necessary condition for liveness and boundedness

**Proposition:** Each live and bounded marked p/t-net

has a transition invariant $\vec{j}$ satisfying

$\vec{j}(t) > 0$ for each transition $t$.

**Proof:** By liveness, there exist occurrence sequences

$m_0 \xrightarrow{\sigma_1} m_1 \xrightarrow{\sigma_2} m_2 \xrightarrow{\sigma_3} \cdots$

such that all transitions occur in every $\sigma_i$.

By boundedness, $m_i = m_j$ for some $i < j$.

$\Rightarrow m_i \xrightarrow{\sigma_{i+1}} \cdots \xrightarrow{\sigma_j} m_j = m_i$.

$\Rightarrow \vec{j} = P[\sigma_{i+1} \cdots \sigma_j]$ is a suitable transition invariant.
Structural Techniques

A siphon is a set of places which, once unmarked, never gains a token again

$S$ is a siphon if $\cdot S \subseteq S^\bullet$, i.e. if $t^\bullet \cap S \not= \emptyset$ implies $\cdot t \cap S \not= \emptyset$.

A trap is a set of places which, once marked, never looses all tokens

$S$ is a trap if $S^\bullet \subseteq \cdot S$, i.e. if $\cdot t \cap S \not= \emptyset$ implies $t^\bullet \cap S \not= \emptyset$.

If a marking $m$ satisfies $m(s_1) = m(s_2) = 0$ then so do all follower markings.

If a marking $m$ satisfies $m(s_3) + m(s_4) > 0$ then so do all follower markings.

Example for the use of a trap

${s_1, s_4, s_5}$ is an initially marked trap

$\Rightarrow$ the marking $(0, 1, 1, 0, 0)$ is not reachable.
Siphons and traps, liveness and deadlock-freedom

Proposition: In a live marked p/t-net without isolated places, each nonempty siphon contains an initially marked place.

Proof: otherwise, for each place $s$ of the siphon, all transitions in $\cdot s \cup s $ are dead.

Proposition: Assume a marked p/t-net with some transition, without capacity restrictions and arc weights. If each nonempty siphon includes an initially marked trap then the marked p/t-net is deadlock-free.

Proof: the set of unmarked places at a dead marking is a nonempty siphon. This siphon contains no marked trap. $\Rightarrow$ It contains no initially marked trap.

Restricted net classes

State machines are marked p/t-nets without branched transitions, i.e. $|\cdot t| = |t\cdot| = 1$ for each transition, without arc weights and without capacity restrictions.

Example

Proposition Each marked state machine is bounded.

Proposition A marked state machine is live if and only if it is strongly connected and some place is initially marked.
Marked graphs are marked \( p/t \)-nets without branched places, i.e. \( |s^*| = |s^| = 1 \) for each place, without arc weights and without capacity restrictions.

Example

```
\[\begin{array}{c}
\text{\textbf{Example}} \\
\end{array}\]
```

Proposition A marked graph is live if and only if each cycle carries a token initially.

Proposition It is moreover 1–safe if and only if each place belongs to a cycle with exactly one token.

Free-choice nets are marked \( p/t \)-nets without arc weights and capacity restrictions satisfying

\[
(s, t) \in F \Rightarrow s^t s^* \subseteq F \text{ for each place } s \text{ and transition } t
\]

Example

```
\[\begin{array}{c}
\text{\textbf{Example}} \\
\end{array}\]
```

\{s_1, s_2, s_5\} is a siphon which includes no nonempty trap

\[\Rightarrow\text{ this free-choice net is not live.}\]
Causal semantics of marked p/t-nets

Example: a producer / consumer system

A causal run of the producer / consumer system

Causal runs

A causal run of a marked p/t-net is given by a labeled Petri net \((B, E, K)\)

Interpretation of causal runs

<table>
<thead>
<tr>
<th>net element</th>
<th>name</th>
<th>symbol</th>
<th>interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>places</td>
<td>conditions</td>
<td>(B)</td>
<td>tokens on system places</td>
</tr>
<tr>
<td>transitions</td>
<td>events</td>
<td>(E)</td>
<td>system transition occurrences</td>
</tr>
<tr>
<td>arcs</td>
<td>causal relation</td>
<td>(K)</td>
<td>flow of tokens</td>
</tr>
</tbody>
</table>
Occurrence nets

An occurrence net is a net \((B, E, K)\) with the following properties

- it has no cycles (i.e. \(K^+\) is a partial order \(\prec\))
- it has no branched places, i.e.
  \[ |\bullet b|, |\bullet^\ast b| \leq 1 \text{ for each condition } b \]
- events have finite fan-in and fan-out, i.e.
  \(\bullet e\) and \(e^\ast\) are finite sets for each event \(e\)
- it has neither input nor output-events, i.e.
  \[ |\bullet e|, |\bullet^\ast e| \geq 1 \text{ for each event } e \]
- no node has infinitely many predecessors, i.e.
  the set \(\{x \in (B \cup E) \mid x \prec y\}\) is finite for each node \(y\)

Process nets of marked p/t-nets represent causal runs

Assume a marked p/t-net \((S, T, F, k, w, m_0)\) without capacity restrictions

An occurrence net \((B, E, K)\) together with
labels \(\pi : (B \cup E) \rightarrow (S \cup T)\) is a process net of \(N\) if

- sorts of nodes are respected by \(\pi\), i.e.
  \(\pi(B) \subseteq S\) and \(\pi(E) \subseteq T\)
- \(m_0\) agrees with \(\min(B)\), i.e.
  \(m_0(s) = |\{b \in B \mid \bullet b = \emptyset \text{ and } \pi(b) = s\}|\) for every place \(s \in S\)
- transition vicinities are preserved, i.e.
  \(\pi(\bullet e) = \bullet(\pi(e)), |\{b \in \bullet e \mid \pi(b) = s\}| = w(s, \pi(e))\) for each event \(e\)
  \(\pi(e^\ast) = (\pi(e))^\ast, |\{b \in e^\ast \mid \pi(b) = s\}| = w(\pi(e), s)\) for each event \(e\)
Occurrence sequences versus process nets

**occurrence** provide total orders of events that respect causality but add arbitrary interleavings of independent events. Information about causal relationships can get lost.

**process nets** provide partial orders reflecting causality.

**Example:**

\[
\begin{align*}
\text{maximal occurrence sequences} & \quad \text{maximal process nets} \\
a b c & \quad \text{d a f c h} \\
b a c & \quad \text{e b f g h} \\
a c b & \quad \text{d a f c h} \\
b c a & \quad \text{e b f g h}
\end{align*}
\]

Two process nets corresponding to \( t_1 t_3 t_5 t_2 t_3 t_5 t_2 t_3 t_4 \)
Two process nets without a common occurrence sequence

![Diagram of two process nets without a common occurrence sequence]
Part 1: Introduction to CP-nets

An ordinary Petri net (PT-net) has *no types and no modules*:
- Only one kind of tokens and the net is flat.

With Coloured Petri Nets (CP-nets) it is possible to use *data types* and complex *data manipulation*:
- Each token has attached a data value called the *token colour*.
- The token colours can be *investigated* and *modified* by the occurring transitions.

With CP-nets it is possible to make *hierarchical* descriptions:
- A large model can be obtained by *combining* a set of *submodels*.
- Well-defined *interfaces* between the submodels.
- Well-defined *semantics* of the combined model.
- *Submodels* can be reused.
Declarations:

- type U = with p | q;
- type I = int;
- type P = product U * I;
- type E = with e;
- var x : U;
- var i : I;

**Resource allocation example**

```
if x=q then 1`e
else empty
```

**Occurrence of enabled binding**

```
if x=p then 1`e
else empty
```
Binding which is not enabled

Binding: 
<x=q, i=2>

case x of
p => 2 e
| q => 1 e

(x,i)

case x of
p => 2 e
| q => 1 e

(x,i)

Binding cannot occur

A more complex example

Binding: 
<x=p, i=2>

case x of
p => 2 e
| q => 1 e

(x,i)

Binding: 
<x=q, i=3>

case x of
p => 2 e
| q => 1 e

(x,i)
Concurrency

- The two bindings may occur *concurrently*.
- This is possible because they use *different tokens*.

Conflict

- These two bindings cannot occur *concurrently*.
- The reason is that they need the *same tokens*.
Resource allocation system

Two kinds of processes:
- Three cyclic q-processes (states A,B,C,D and E).
- Two cyclic p-processes (states B,C,D and E).

Three kinds of resources:
- Represented by the places R, S and T.

During a cycle a process reserves some resources and releases them again:
- Tokens are removed from and added to the resource places R, S and T.

A cycle counter is increased each time a process completes a full cycle.

It is rather straightforward to prove that the resource allocation system cannot deadlock.
- What happens if we add an additional token to place S – i.e., if we start with four S-resources instead of three?

Coloured Petri Nets

Declarations:
- Types, functions, operations and variables.

Each place has the following inscriptions:
- Name (for identification).
- Colour set (specifying the type of tokens which may reside on the place).
- Initial marking (multi-set of token colours).

Each transition has the following inscriptions:
- Name (for identification).
- Guard (boolean expression containing some of the variables).

Each arc has the following inscriptions:
- Arc expression (containing some of the variables). When the arc expression is evaluated it yields a multi-set of token colours.
Enabling and occurrence

A binding assigns a colour (i.e., a value) to each variable of a transition.

A binding element is a pair (t,b) where t is a transition while b is a binding for the variables of t. Example: (T2,<x=p, i=2>).

A binding element is enabled if and only if:
• There are enough tokens (of the correct colours on each input-place).
• The guard evaluates to true.

When a binding element is enabled it may occur:
• A multi-set of tokens is removed from each input-place.
• A multi-set of tokens is added to each output-place.

A binding element may occur concurrently to other binding elements – iff there are so many tokens that each binding element can get its "own share".

Main characteristics of CP-nets

Combination of text and graphics.

Declarations and net inscriptions are specified by means of a formal language, e.g., a programming language.

• Types, functions, operations, variables and expressions.

Net structure consists of places, transitions and arcs (forming a bi-partite graph).

• To make a CP-net readable it is important to make a nice graphical layout.
• The graphical layout has no formal meaning.

CP-nets have the same kind of concurrency properties as Place/Transition Nets.
**Formal definition of CP-nets**

**Definition:** A Coloured Petri Net is a tuple \( CPN = (\Sigma, P, T, A, N, C, G, E, I) \) satisfying the following requirements:

(i) \( \Sigma \) is a finite set of non-empty types, called **colour sets**.
(ii) \( P \) is a finite set of **places**.
(iii) \( T \) is a finite set of **transitions**.
(iv) \( A \) is a finite set of **arcs** such that:
   - \( P \cap T = P \cap A = T \cap A = \emptyset \).
(v) \( N \) is a **node** function. It is defined from \( A \) into \( P \times T \cup T \times P \).
(vi) \( C \) is a **colour** function. It is defined from \( P \) into \( \Sigma \).
(vii) \( G \) is a **guard** function. It is defined from \( T \) into expressions such that:
   - \( \forall t \in T: \text{Type}(G(t)) = \text{Bool} \land \text{Type}(\text{Var}(G(t))) \subseteq \Sigma \).
(viii) \( E \) is an **arc expression** function. It is defined from \( A \) into expressions such that:
   - \( \forall a \in A: \text{Type}(E(a)) = C(p(a))_{MS} \land \text{Type}(\text{Var}(E(a))) \subseteq \Sigma \) where \( p(a) \) is the place of \( N(a) \).
(ix) \( I \) is an **initialization** function. It is defined from \( P \) into closed expressions such that:
   - \( \forall p \in P: \text{Type}(I(p)) = C(p)_{MS} \).

**Formal definition of behaviour**

**Definition:** A step is a multi-set of binding elements.

A step \( Y \) is **enabled** in a marking \( M \) iff the following property is satisfied:

\[
\forall p \in P: \sum_{(t,b) \in Y} E(p,t)<b> \leq M(p).
\]

When a step \( Y \) is enabled in a marking \( M_1 \) it may **occur**, changing the marking \( M_1 \) to another marking \( M_2 \), defined by:

\[
\forall p \in P: M_2(p) = (M_1(p) - \sum_{(t,b) \in Y} E(p,t)<b>) + \sum_{(t,b) \in Y} E(t,p)<b>.
\]

The first sum is called the **removed** tokens while the second is called the **added** tokens. Moreover we say that \( M_2 \) is **directly reachable** from \( M_1 \) by the occurrence of the step \( Y \), which we also denote: \( M_1 [Y \triangleright M_2 \].

An **occurrence sequence** is a sequence of markings and steps:

\( M_1 [Y_1 \triangleright M_2 [Y_2 \triangleright M_3 \ldots M_n [Y_n \triangleright M_{n+1} \]

such that \( M_i [Y_i \triangleright M_{i+1} \) for all \( i \in 1..n \). We then say that \( M_{n+1} \) is **reachable** from \( M_1 \). We use \([M \triangleright \) to denote the set of markings which are reachable from \( M \).
Formal definition

The existence of a *formal definition* is very important:

- It is the basis for *simulation*, i.e., execution of the CP-net.
- It is also the basis for the *formal verification* methods (e.g., state spaces and place invariants).
- Without the formal definition, it would have been impossible to obtain a *sound* net class.

It is *not necessary* for a *user* to know the formal definition of CP-nets:

- The correct *syntax* is checked by the CPN editor, i.e., the computer tool by which CP-nets are constructed.
- The correct use of the *semantics* (i.e., the enabling rule and the occurrence rule) is guaranteed by the CPN simulator and the CPN tools for formal verification.

High-level contra low-level nets

The relationship between CP-nets and Place/Transition Nets (PT-nets) is *analogous* to the relationship between high-level programming languages and assembly code.

- In theory, the two levels have exactly the same *computational power*.
- In practice, high-level languages have much more *modelling power* – because they have better structuring facilities, e.g., types and modules.

Each CP-net has an *equivalent* PT-net – and vice versa.

- This equivalence is used to derive the definition of *basic properties* and to establish the *verification methods*.
- In practice, we *never* translate a CP-net into a PT-net – or vice versa.
- Description, simulation and verification are done *directly* in terms of CP-nets.
Other kinds of high-level nets

CP-nets have been developed from Predicate/Transition Nets.

- **Hartmann Genrich & Kurt Lautenbach.**
- With respect to description and simulation the two models are nearly identical.
- With respect to formal verification there are some differences.

Several other kinds of high-level Petri Nets exist.

- They all build upon the same basic ideas, but use different languages for declarations and inscriptions.
- A detailed comparison is outside the scope of this talk.

Simple protocol

```
1. Model
2. 'g and An
3. 'y Means
4. 'of Colour
5. 'red Pet'
6. 'Nets
7. ':
8. "#######"
```

```
type INT = int;
type BOOL = bool;
type DATA = string;
type INTxDATA = product INT * DATA;
var n, k : INT;
var p, str : DATA;
val stop = "########";
type Int_0_10 = int with 0..10;
type Int_1_10 = int with 1..10;
var s : Int_0_10;
var r : Int_1_10;
fun Ok(s : Int_0_10, r : Int_1_10) = (r <= s);```

```
type INT = int;
type BOOL = bool;
type DATA = string;
type INTxDATA = product INT * DATA;
var n, k : INT;
var p, str : DATA;
val stop = "########";
type Int_0_10 = int with 0..10;
type Int_1_10 = int with 1..10;
var s : Int_0_10;
var r : Int_1_10;
fun Ok(s : Int_0_10, r : Int_1_10) = (r <= s);
```
Only the binding \(<n=1, p= \text{"Modellin"}>) is enabled.

- When the binding occurs it adds a token to place A. The token represents that the packet (1,"Modellin") is sent to the network.
- The packet is not removed from place Send and the NextSend counter is not changed.

There are now 10 enabled bindings:

- They are all of the form \(<n=1, p= \text{"Modellin"}, s=8, r=\ldots>)
- The variable \(r\) can take 10 different values, because the type of \(r\) is defined to contain the integers 1..10.

The function \(\text{Ok}(s,r)\) checks whether \(r \leq s\).

- For \(r \in 1..8\), \(\text{Ok}(s,r) = \text{true}\). This means that the token is moved from A to B, i.e., that the packet is successfully transmitted over the network.
- For \(r \in 9..10\), \(\text{Ok}(s,r) = \text{false}\). This means that no token is added to B, i.e., that the packet is lost.
- The CPN simulator makes random choices between enabled bindings. Hence there is 80% chance for successful transfer.
Receive packet

It is checked whether the number of the incoming packet \( n \) matches the number of the expected packet \( k \).

Correct packet number

- The data in the packet is *concatenated* to the data already received.
- The \( \text{NextRec} \) counter is increased by one.
- An *acknowledgement message* is sent. It contains the number of the next packet which the receiver wants to get.
Wrong packet number

- The data in the packet is *ignored*.
- The *NextRec* counter is unchanged.
- An *acknowledgement message* is sent. It contains the number of the next packet which the receiver wants to get.

Transmit acknowledgement

This transition works in a similar way as *Transmit Packet*.

- The token on place *RA* determines the success rate for transmission of acknowledgements.
- When *RA* contains a token with value 8, the success rate is 80%.
- When *RA* contains a token with value 10, *no* acknowledgements are lost.
- When *RA* contains a token with value 0, *all* acknowledgements are lost.
Receive acknowledgement

When an acknowledgement arrives to the Sender it is used to update the NextSend counter.

- In this case the counter value becomes 2, and hence the Sender will begin to send packet number 2.

Intermediate state

- The Receiver is expecting package no. 6. This means that it has successfully received the first 5 packets.

- The Sender is still sending packet no. 5. In a moment it will receive an acknowledgement containing a request for packet no. 6.

- When the acknowledgement is received the NextSend counter is updated and the Sender will start sending packet no. 6.
Final state

When the last packet, i.e., packet no. 8 reaches the Receiver an acknowledgement with value 9 is sent.

When this acknowledgement reaches the Sender the NextSend counter is updated to 9.

This means that the Send Packet transition no longer can occur, and hence the transmission stops.

Part 2: Hierarchical CP-nets

A hierarchical CP-net contains a number of interrelated subnets—called pages.

Simple Protocol

Sender

Receive Acknow.  

Network

Transmit Acknow.

Receiver

NextSend

NextRec

• When the last packet, i.e., packet no. 8 reaches the Receiver an acknowledgement with value 9 is sent.

• When this acknowledgement reaches the Sender the NextSend counter is updated to 9.

• This means that the Send Packet transition no longer can occur, and hence the transmission stops.
Substitution transitions

A page may contain one or more substitution transitions.

- Each substitution transition is related to a page, i.e., a subnet providing a more detailed description than the transition itself.
- The page is a subpage of the substitution transition.

There is a well-defined interface between a substitution transition and its subpage:

- The places surrounding the substitution transition are socket places.
- The subpage contains a number of port places.
- Socket places are related to port places – in a similar way as actual parameters are related to formal parameters in a procedure call.
- A socket place has always the same marking as the related port place. The two places are just different views of the same place.

Substitution transitions work in a similar way as the refinement primitives found in many system description languages – e.g., SADT diagrams.

Pages can be used more than once

There are two different instances of the Receiver page. Each instance has its own marking.
Ring network

Formal definition of hierarchical CP-nets

The syntax and semantics of hierarchical CP-nets have formal definitions – similar to the definitions for non-hierarchical CP-nets.

Each hierarchical CP-net has an equivalent non-hierarchical CP-net – and vice versa.

- The two kinds of nets have the same computational power – but hierarchical CP-nets have much more modelling power.
- The equivalence is used for theoretical purposes.
- In practice, we never translate a hierarchical CP-net into a non-hierarchical CP-net – or vice versa.
CP-nets may be large

A typical industry application of CP-nets contains:

• 10-200 pages.
• 50-1000 places and transitions.
• 10-200 colour sets.

This corresponds to thousands/millions of nodes in a Place/Transition Net.

Most of the industrial applications would be totally impossible without:

• Colours.
• Hierarchies.
• Computer tools.

Protocol for telephone network

Transport layer of a protocol for digital telephone communication.

Overview of the hierarchy structure:

• Each node represents a page, i.e., a subnet.
• Each arc represents a transition substitution.
Two of the most abstract pages

This page describes the possible actions that can happen when the user site is in state U8:

- From the network five different kinds of messages may be received.
- In addition there is one kind of internal user request.
- In three of the cases a new message is sent to the network site.
Typical transition

This transition describes the actions that are taken when a Status Enquiry message is received in state U8:

- The guard checks that the message is a Status Enquiry message. It also checks that the Call Reference is correct (i.e., matches the one in the User State token at place U8).
- A Status message is sent to the network site. It tells that the user site is in state U8.

SDL description of user page

Each vertical string of SDL symbols describes a sequence of actions – which is translated into a single CPN transition.

- The translation from SDL to CPN was done manually.
- The translation is straightforward and it could easily be automated.

The graphical shape of a node has a well-defined meaning in SDL.
- In the CP-net the shape is retained – to improve the readability. It has no formal meaning.
Typical page for the network site

SDL description of network page

Similar structure as for the user page – but slightly more complex.

It is easy to see that there is a very straightforward relationship between the SDL page and the corresponding CPN page.
Some pages are used many times

- 43 pages with more than 100 page instances.
- The entire modelling of this – fairly complex protocol – was made in only 3 weeks (by a single person).
- According to engineers at the participating telecommunications company, the CPN model was the most detailed behavioural model that they had ever seen for such protocols.

Practical use of CP-nets

CP-nets are used in many different areas. A few selected examples are:

- Communication protocols (BRI, DQDB, ATM).
- VLSI chips (clocked and self-timed).
- Banking procedures (check processing and funds transfer).
- Correctness of ADA programs (rendezvous structure).
- Teleshopping systems.
- Military systems (radar control post and naval vessel).
- Security systems (intrusion alarms, etc.).
- Flexible manufacturing.
Summary of practical experiences

*Graphical representation* and *executability* are extremely important.

Most practical models are *large*.
- They cannot be constructed without the *hierarchy concepts*.
- Neither can they be constructed or verified without the *computer tools*.

CP-nets are often used *together* with other graphical description languages, such as SADT, SDL and block diagrams.
- This means that the user does not have to learn a completely *new language*.

CP-nets are well-suited for *verification* of existing designs – in particular concurrent systems.
- CP-nets can also be used to *design* new systems.
- Then it is possible to use the *insight* gained through the modelling, simulation and verification activities – to *improve* the design itself.

Part 3: Construction and Simulation of CP-nets

CP-nets have an *integrated* set of *robust* computer tools with *reliable support*:
- *Construction* and *modification* of CPN models.
- *Syntax checking* (e.g., types and module interfaces).
- *Interactive simulation*, e.g., to gain additional understanding of the modelled system. Can also be used for *debugging*.
- *Automatic simulations*, e.g., to obtain performance measures. Can also be used for *prototyping*.
- *Verification* to *prove* behavioural properties.
  - *State spaces* (also called reachability graphs and occurrence graphs).
  - *Place invariants*.

The computer tools are available on:
- Sun Sparc with Solaris.
- HP with HPUX
- Intel PCs with Linux.
- Macintosh with Mac OS.
CPN editor

Each page is shown in its own window.

The user performs an operation by selecting an object and a command for it, e.g.:
- Select a transition (by pointing with the mouse).
- Select the desired command (by pointing in the corresponding drop-down menu).

Commands can be performed on a set of objects.

Editor knows syntax of CP-nets

Some kinds of errors are impossible, e.g.:
- An arc between two places or two transitions.
- A place with two colour sets or an arc with two arc expressions.
- A transition with a colour set.
- Port assignment involving a place which is a non-socket or a non-port.
- A cyclic set of substitution transitions.

The editor behaves intelligently:

- When a node is repositioned or resized the surrounding arcs and inscriptions are automatically adjusted.
- When a node is deleted the surrounding arcs are automatically deleted.
Attributes

Each graphical object has its own *attributes*.

They determine how the object appears on the screen/print-outs:

- **Text attributes**
  - Transition
  - Transition
  - Transition

- **Graphical attributes**
  -
  -

- **Shape attributes**
  -
  -

Each *kind of objects* has its own *defaults*:

**Initial Marking**

- Place Name
- Colour Set
- Arc Expression

**Transition**

- Name
- [Guard]

Defaults can be *changed* and they can be *overwritten* (for individual objects).

---

Easy to experiment

Can we improve the *layout* of this page?
We want to move the four selected nodes to a new page – and replace them by a substitution transition:

- This is done by a single command – called Move to Subpage.
Result of Move to Subpage

Move to Subpage is complex

The Move to Subpage command is complex. The command:

- Checks the legality of the selection (all border nodes must be transitions).
- Creates the new page.
- Moves the subnet to the new page.
- Prompts the user to create a new transition which becomes the supernode for the new subpage.
- Creates the port places by copying those places which were next to the selected subnet.
- Calculates the port types (in, out or in/out).
- Creates the corresponding port inscriptions.
- Constructs the necessary arcs between the port nodes and the selected subnet.
- Draws the arcs surrounding the new transition.
- Creates a hierarchy inscription for the new transition.
- Updates the hierarchy page.

All these things are done in a few seconds.
Top-down and bottom-up

*Move to Subpage* supports top-down development. However, it is also possible to work bottom-up – or use a mixture of top-down and bottom-up.

The *Substitution Transition* command is used to relate a substitution transition to an existing page.

The command:

- Makes the hierarchy page active.
- Prompts the user to select the desired subpage; when the mouse is moved over a page node it blinks, unless it is illegal (because selecting it would make the page hierarchy cyclic).
- Waits until a blinking page node has been selected.
- Tries to deduce the port assignment by means of a set of rules which looks at the port/socket names and the port/socket types.
- Creates the hierarchy inscription with the name and number of the subpage and with those parts of the port assignment which could be automatically deduced.
- Updates the hierarchy page.

Syntax checking

When a CPN diagram has been constructed it can be syntax checked.

The most common errors are:

- Syntax errors in the declarations.
- Syntax errors in arc expressions or guards.
- Type mismatch between arc expressions and colour sets.

Syntax checking is incremental:

- When a colour set, guard or an arc expression is changed, it is sufficient to recheck the nearest surroundings.
- Analogously, if an arc is added or removed.

All CPN diagrams in this set of lecture notes are made by means of the CPN editor.
CPN simulator

When a syntactical correct CPN diagram has been constructed, the CPN tool generates the necessary code to perform simulations.

The simulation code:

- Calculates whether the individual transitions and bindings are enabled.
- Calculates the effect of occurring transitions and bindings.

The code generation is incremental. Hence it is fast to make small changes to the CPN diagram.

We distinguish between two kinds of simulations:

- In an interactive simulation the user is in control, but most of the work is done by the system.
- In an automatic simulation the system does all the work.

Simulation results are shown directly on the CP-net:

- The user can see the enabled transitions and the markings of the individual places.

To execute a step, the user:

- Selects one of the enabled transitions.
- Then he either enters a binding or asks the simulator to calculate all the enabled bindings, so that he can select one.
Execution of a step

The simulator:

- Checks the *legality and enabling* of the binding.
- Calculates the *result of the execution*.

The *user determines* whether the simulator displays the tokens which are added/removed:

Interactive simulation with random selection of steps

The simulator *chooses* between conflicting transitions and bindings (by means of a *random number generator*):

- The user can *observe* all details, e.g., the markings the enabling and the added/removed tokens.
- The simulator *shows the page* on which each step is executed – by moving the corresponding window to the top of the screen.
- The user can set *breakpoints* so that he has the necessary time to inspect markings, enablings, etc.

A simulation with this amount of graphical feedback is *slow* (typically a few transitions per minute):

- It takes a lot of time to update the graphics.
- A user has no chance to follow a fast simulation.

It is possible to *turn off* selected parts of the *graphical feedback*, e.g.:

- Added and removed *tokens*.
- Observation of *uninteresting pages*.
Automatic simulation

The simulator *chooses* between conflicting transitions and bindings (by means of a *random number generator*).

The user does *not* intend to follow the simulation:

- The simulation can be *very fast* – several hundred steps per second.
- The user specifies some *stop criteria*, which determine the duration of the simulation.
- When the simulation stops the graphics of the CP-net is *updated*.
- Then the user can inspect all details of the graphics, e.g., the *enabling* and the *marking*.
- Automatic simulations can be *mixed* with interactive simulations.

To find out what happens during an *automatic simulation* the user has a number of choices.

Simulation report

1. `SendPack@(1:Top#1) {n = 1, p = "Modellin"}`
2. `TranPack@(1:Top#1) {n = 1, p = "Modellin", r = 6, s = 8}`
3. `SendPack@(1:Top#1) {n = 1, p = "Modellin"}`
4. `TranPack@(1:Top#1) {n = 1, p = "Modellin", r = 3, s = 8}`
5. `RecPack@(1:Top#1) {k = 1, n = 1, p = "Modellin", str = ""}`
6. `SendPack@(1:Top#1) {n = 1, p = "Modellin"}`
7. `SendPack@(1:Top#1) {n = 1, p = "Modellin"}`
8. `TranPack@(1:Top#1) {n = 2, r = 2, s = 8}`
9. `TranPack@(1:Top#1) {n = 1, p = "Modellin", r = 7, s = 8}`
10. `RecPack@(1:Top#1) {k = 2, n = 1, p = "Modellin", str = "Modellin"}`
11. `RecAck@(1:Top#1) {k = 1, n = 2}`
12. `RecPack@(1:Top#1) {k = 2, n = 1, p = "Modellin", str = "Modellin"}`
13. `TranAck@(1:Top#1) {n = 2, r = 7, s = 8}`
14. `TranPack@(1:Top#1) {n = 1, p = "Modellin", r = 6, s = 8}`
15. `RecAck@(1:Top#1) {k = 2, n = 2}`
16. `SendPack@(1:Top#1) {n = 2, p = "g and An"}`
17. `TranAck@(1:Top#1) {n = 2, r = 6, s = 8}`
18. `RecPack@(1:Top#1) {k = 2, n = 1, p = "Modellin", str = "Modellin"}`
19. `RecAck@(1:Top#1) {k = 2, n = 2}`
20. `SendPack@(1:Top#1) {n = 2, p = "g and An"}`

The *simulation report* shows the *transitions* which have occurred. The user determines whether he also wants to see the *bindings*. 
These charts are used to show the *progress* of a simulation of the simple protocol:

- The upper chart is updated each time a new packet is *successfully received*.
- The lower chart is updated for *each 50 steps*.

This graphic is used to display the state of a *simple telephone system*. The graphics is updated each time one of the telephones changes to a new state:

- Telephones u(7) and u(8) are *connected*.
- Telephone u(2) is calling u(6) which is *ringing*.
- Telephone u(10) is calling u(2). This call will *not succeed* because u(2) already is engaged.
Code segments

Each transition may have a code segment, i.e., a sequence of program instructions which are executed each time the transition occurs.

- The instructions in code segment are used to update charts and graphics.
- This is done by calling a number of library functions.
- Usually, the code segment does not influence the behaviour of the CP-net (i.e., the enabling and occurrence).
- However, a code segment may read and write from files.
- In this way it is possible to input values to be used during the simulation, or to output simulation results.

Standard ML

Declarations, net inscriptions and code segments are specified in a programming language called Standard ML.

- Strongly typed, functional language.
- Data types can be:
  - Atomic (integers, reals, strings, booleans and enumerations).
  - Structured (products, records, unions, lists and subsets).
- Arbitrary complex functions and operations can be defined (polymorphism and overloading).
- Computational power of expressions are equivalent to lambda calculus (and hence to Turing machines).
- Developed at Edinburgh University by Robin Milner and his group.
- Standard ML is well-known, well-tested and very general. Several text books are available.
Time analysis

CP-nets can be extended with a *time concept*. This means that the *same language* can be used to investigate:

- **Logical correctness.** Desired functionality, absence of deadlocks, etc.
- **Performance.** Remove bottlenecks. Predict mean waiting times and average throughput. Compare different strategies.

In a timed CP-net each token carries a *colour* (data value) and a *time stamp* (telling when it can be used).

Time stamps are specified by expressions:

- Time stamps can depend upon *colour values*.
- Time stamps can be specified by *probability distributions*.
- This means that we, e.g., can specify *fixed* delays, *interval* delays and *exponential* delays.

A timed CP-net for protocol

For the three *Send* and *Receive* operations we specify a fixed delay.

For the *network* we specify an interval delay, i.e., random delay between 25 and 75 time units.

The token colour on place *Wait* specifies the delay between two retransmissions of the same packet.

The computer tools for CP-nets also support simulation of timed CP-nets.
Timed simulation of protocol

Time: 570

- Model time is now 570.
- Send Packet has sent a copy of packet no. 2 at time 570.
- If no acknowledgement arrives another copy of packet no. 2 will be sent at time 670.
- The only transition which is enabled at time 570 is Transmit Packet.

Timed simulations

Timed simulations have the same facilities as untimed simulations, e.g.:

- We can switch between interactive and automatic simulation.
- Simulation reports tell the time at which the individual transitions occurred.
- We can use charts and other kinds of reporting facilities.

It is easy to switch between a timed and an untimed simulation.
Charts for a timed simulation

Part 4: Verification of CP-nets

In this part of the talk we describe the two most important methods for verification of CP-nets:

- **State spaces** (also called reachability graphs and occurrence graphs).
- **Place invariants**.

We also describe how the verification methods are supported by computer tools.

- **Short interval** between retransmissions implies fast transmission with heavy use of the network.

- **Long interval** between retransmissions implies slow transmission with less use of the network.

- To get reliable results it is necessary to make a large number of lengthy simulation runs.
State space analysis

```plaintext
type U = with p | q;
type E = with e;
var x : U;

if x=q then 1`q else empty
if x=p then 1`p else empty

To obtain a finite state space we remove the cycle counters. Otherwise there would be an infinite number of reachable markings.
```

State space for resource allocation

Directed graph with:
- A node for each reachable marking (i.e., state).
- An arc for each occurring binding element.
Some questions that can be answered from state spaces

**Boundedness** properties:

- What is the *maximal number* of tokens on the different places?
- What is the *minimal number* of tokens on the different places?
- What are the *possible token colours*?

**Home** properties:

- Is it *always* possible to *return* to the initial marking?

**Liveness** properties:

- Are all transitions live, i.e., can they *always* become enabled *again*?

---

State space report for resource allocation system

**Statistics**

<table>
<thead>
<tr>
<th>Occurrence Graph</th>
<th>Scc Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes: 13</td>
<td>Nodes: 1</td>
</tr>
<tr>
<td>Arcs: 20</td>
<td>Arcs: 0</td>
</tr>
<tr>
<td>Secs: 1</td>
<td>Secs: 1</td>
</tr>
<tr>
<td>Status: Full</td>
<td></td>
</tr>
</tbody>
</table>

**Boundedness Properties**

<table>
<thead>
<tr>
<th>Upper Integer Bounds</th>
<th>Lower Integer Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 3</td>
<td>A: 1</td>
</tr>
<tr>
<td>B: 3</td>
<td>B: 1</td>
</tr>
<tr>
<td>C: 1</td>
<td>C: 0</td>
</tr>
<tr>
<td>D: 1</td>
<td>D: 0</td>
</tr>
<tr>
<td>E: 1</td>
<td>E: 0</td>
</tr>
<tr>
<td>R: 1</td>
<td>R: 0</td>
</tr>
<tr>
<td>S: 3</td>
<td>S: 0</td>
</tr>
<tr>
<td>T: 2</td>
<td>T: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Upper Multi-set Bounds</th>
<th>Lower Multi-set Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 3q</td>
<td>A: 1q</td>
</tr>
<tr>
<td>B: 2q+ 1q</td>
<td>B: 1p</td>
</tr>
<tr>
<td>C: 1q+ 1q</td>
<td>C: empty</td>
</tr>
<tr>
<td>D: 1q+ 1q</td>
<td>D: empty</td>
</tr>
<tr>
<td>E: 1q+ 1q</td>
<td>E: empty</td>
</tr>
<tr>
<td>R: 1e</td>
<td>R: empty</td>
</tr>
<tr>
<td>S: 3e</td>
<td>S: empty</td>
</tr>
<tr>
<td>T: 2e</td>
<td>T: empty</td>
</tr>
</tbody>
</table>
State space report (continued)

Home Properties

Home Markings: All

Liveness Properties

Dead Markings: None
Live Transitions: All

Fairness Properties

T1: No Fairness
T2: Impartial
T3: Impartial
T4: Impartial
T5: Impartial

Generation of the state space report takes only a few seconds.

- The report contains a lot of useful information about the behaviour of the CP-net.
- The report is excellent for locating errors or to increase our confidence in the correctness of the system.

Strongly connected components

- Subgraph where all nodes are reachable from each other.
- Maximal subgraph with this property.
Strongly connected components are very useful

There are often much fewer strongly connected components than nodes:

- A cyclic system has only one strongly connected component.
- This is, e.g., the case for the resource allocation system.
- The strongly connected components can be determined in linear time, e.g., by Tarjan’s algorithm.

Strongly connected components can be used to answer questions about home properties and liveness properties.

State space for simple protocol

To obtain a finite state space we limit the number of tokens on the “buffer” places A, B, C and D. Otherwise there would be an infinite number of reachable markings.

Moreover, we now only have 4 packets and a binary choice between success and failure.
State space report for protocol

Statistics

Occurrence Graph

Nodes: 4298
Arcs: 15887
Secs: 53
Status: Full

ScC Graph

Nodes: 2406
Arcs: 11677
Secs: 17

Boundedness Properties

Upper Integer Bounds

A: 1
B: 2
C: 1
D: 2
NextRec: 1
NextSend: 1
RA: 1
RP: 1
Received: 1
Send: 4

Lower Integer Bounds

A: 0
B: 0
C: 0
D: 0
NextRec: 1
NextSend: 1
RA: 1
RP: 1
Received: 1
Send: 4

State space report (continued)

Upper Multi-set Bounds

A: \(1 \cdot (1,"\text{Coloured}) + 1 \cdot (2,"\text{Petri N}) + 1 \cdot (3,"\text{ets}) + 1 \cdot (4,"\text{ets})\)

B: \(2 \cdot (1,"\text{Coloured}) + 2 \cdot (2,"\text{Petri N}) + 2 \cdot (3,"\text{ets}) + 2 \cdot (4,"\text{ets})\)

C: \(1 \cdot 2 + 1 \cdot 3 + 1 \cdot 4 + 1 \cdot 5\)

D: \(2 \cdot 2 + 2 \cdot 3 + 2 \cdot 4 + 2 \cdot 5\)

NextRec: \(1 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 + 1 \cdot 4 + 1 \cdot 5\)

NextSend: \(1 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 + 1 \cdot 4 + 1 \cdot 5\)

RA: 1
RP: 1
Received: 1
Send: 4

Lower Multi-set Bounds

A: empty
B: empty
C: empty
D: empty

NextRec: empty
NextSend: empty
RA: 1
RP: 1
Received: empty
Send: \(1 \cdot (1,"\text{Coloured}) + 1 \cdot (2,"\text{Petri N}) + 1 \cdot (3,"\text{ets}) + 1 \cdot (4,"\text{ets})\)
State space report (continued)

Home Properties

Home Markings: 1 [452]

Liveness Properties

Dead Markings: 1 [452]
Live Transitions: None

Fairness Properties

Send Packet: Impartial
Transmit Packet: Impartial
Receive Packet: No Fairness
Transmit Acknow: No Fairness
Receive Acknow: No Fairness

Generation of the state space report takes only a few seconds.

• The report contains a lot of useful information about the behaviour of the CP-net.
• The report is excellent for locating errors or to increase our confidence in the correctness of the system.

Investigation of dead marking

We ask the system to display marking number 452.

Marking no. 452 is the desired final marking (all packets has been received in the correct order)

Marking 452 is dead:
• This implies that the protocol is partially correct (if execution stops it stops in the desired final marking).

Marking 452 is a home marking:
• This implies that we always have a chance to finish correctly (it is impossible to reach a state from which we cannot reach the desired final marking).
Investigation of shortest path

We ask the system to calculate one of the *shortest paths* from the initial marking to the dead marking:

```
val path = NodesInPath(1,452);
> val path = [1,2,3,5,8,11,15,20,27,38,50, 64,80,102,133,164,199,243, 301,375,452] : Node list
```

```
Length(path);
> 20 : int
```

The calculated path contains **20 transitions**.

- This is as expected because there are 4 *packets* which each needs 5 *transitions* to occur.

Drawing of shortest path

We ask the system to draw the *first six nodes* in the calculated shortest path:

```
DisplayNodePath; [1,2,3,5,8,11];
```

```
1
NextSend = 1
NextRec = 1
Received = ""

1:1
SendPack
(p="Coloured",n=1)

2
NextSend = 1
NextRec = 1
Received = ""

2:2
TranPack
(s=1,r=1,p="Coloured",n=1)

3
NextSend = 1
NextRec = 1
Received = ""

3:3
RecPack
(str="",p="Coloured",n=1,k=1)

5
NextSend = 1
NextRec = 2
Received = "Coloured"

5:1
TranAck
(s=1,r=1,n=2)

8
NextSend = 1
NextRec = 2
Received = "Coloured"

8:2
RecAck
(n=2,k=1)

11
NextSend = 2
NextRec = 2
Received = "Coloured"

11:5
RecAck
(n=2,k=1)
Non-standard questions

We ask the system to search all arcs in the entire graph and return the first 10 arcs where NextSend has a larger value in the source marking than it has in the destination marking.

```
PredArcs
(EntireGraph,
 fn a => ((ms_to_col(Mark.NextSend 1 (SourceNode a))) >
 (ms_to_col(Mark.NextSend 1 (DestNode a)))),
 10)
end;
```

>\[10179,10167,10165,10159,10055,10052,10035,10031,10019,10007\] : Arc list
Temporal logic

It is also possible to make questions by means of a CTL-like temporal logic.

Usually CTL focuses on queries about state properties, e.g.:
- Inv(Pos(M)) checks whether M is a home marking.
- Ev(dead) checks whether there are any infinite occurrence sequences.

Our version of CTL also allows queries about transitions and binding elements.
- Inv(Pos(t in Arc)) checks whether transition t is live.

Timed CP-nets

The computer tools for CP-nets also support state space analysis of timed CP-nets.

State spaces – pro/ contra

State spaces are powerful and easy to use.
- The main drawback is the state explosion, i.e., the size of the state space.
- The present version of our tool handles graphs with 250,000 nodes and 1,000,000 arcs. For many systems this is not sufficient.

Fortunately, it is sometimes possible to construct much more compact state spaces – without losing information.
- This is done by exploiting the inherent symmetries of the modelled system.
- We define two equivalence relations (one for markings and one for binding elements).
- The condensed state spaces are often much smaller (polynomial size instead of exponential).
- The condensed state spaces contain the same information as the full state spaces.
**Place invariants analysis**

The basic idea is similar to the use of *invariants* in *program verification*.

- A place invariant is an *expression* which is satisfied for all reachable markings.
- The expression *counts* the tokens of the marking – using a specified set of weights.

We first *construct* a set of place invariants.

Then we check whether they are *fulfilled*.

- This is done by showing that each occurring binding element *respects* the invariants.
- The *removed* set of tokens must be identical to the *added* set of tokens – when the weights are taken into account.

Finally, we use the place invariants to *prove* behavioural properties of the CP-net.

- This is done by a *mathematical proof*.

---

**Example of place invariants**

- **INVARIANTS**
  1: $2p+3q : U$
  2: $e : E$
  3: $3e : E$
  4: $2e : E$

---

**INV**

1: PR1
2: Q
3: Q

---

**INV**

1: PR1
2: Q
3: 2P+2Q
4: P

---

**INV**

1: PR1
2: Q
3: 2P+2Q
4: P+Q
Place invariants for resource allocation system

To specify the weights we use *three functions*:

- $PR_1$ is a *projection* function: $(x,i) \rightarrow x$.
- $P$ is an *indicator* function: $(p,i) \rightarrow 1^e; (q,i) \rightarrow 0$.
- $Q$ is an *indicator* function: $(p,i) \rightarrow 0; (q,i) \rightarrow 1^e$.
- $P$ and $Q$ “counts” the number of $p$ and $q$ tokens.

\[ PR_1(M(A)+M(B)+M(C)+M(D)+M(E)) = 2^p + 3^q \]
\[ M(R) + Q(M(B)+M(C)) = 1^e \]
\[ M(S) + Q(M(B)) + (2P + 2Q)(M(C)+M(D)+M(E)) = 3^e \]
\[ M(T) + P(M(D)) + (2P + Q)(M(E)) = 2^e \]

A more readable version of the place invariants

\[ PR_1(A+B+C+D+E) = 2^p + 3^q \]
\[ R + Q(B+C) = 1^e \]
\[ S + Q(B) + (2P + 2Q)(C+D+E) = 3^e \]
\[ T + P(D) + (2P + Q)(E) = 2^e \]

The place invariants can be used to prove properties of the resource allocation system, e.g., that it is *impossible to reach a dead marking*. 
Tool support for place invariants

**Check** of place invariants:
- The *user* proposes a set of weights.
- The *tool* checks whether the weights constitute a place invariant.

**Automatic calculation** of all place invariants:
- This is possible, but it is a very *complex* task.
- Moreover, it is difficult to represent the results on a *useful form*, i.e., a form which can be used by the system designer.

**Interactive calculation** of place invariants:
- The *user* proposes some of the weights.
- The *tool* calculates the *remaining weights* – if possible.

Interactive calculation of place invariants is *much easier* than a fully automatic calculation.

How to use place invariants

Invariants in ordinary *programming languages*:
- No one would construct a large program – and then expect *afterwards* to be able to calculate invariants.
- Instead invariants are constructed *together* with the program.

For *CP-nets* we should do the same:
- During the system specification and modelling the designer gets a lot of *knowledge* about the system.
- Some of this knowledge can easily be formulated as *place invariants*.
- The invariants can be *checked* and in this way it is possible to find *errors*.
- It can be seen *where* the errors are.

Some *prototypes* of computer tools for invariants analysis do exist. However, none of them are at a state where they can be widely used.
Place invariants – pro/contra

From place invariants it is possible to prove many kinds of *behavioural properties*.

- Invariants can be used to make *modular verification* – because it is possible to combine invariants of the individual pages.
- Invariants can be used to verify *large systems* – without computational problems.
- The user needs some ingenuity to *construct* invariants. This can be supported by *computer tools* – interactive process.
- The user also needs some ingenuity to *use* invariants. This can also be supported by *computer tools* – interactive process.
- Invariants can be used to verify a system – without fixing the *system parameters* (such as the number of sites in the data base system).

Conclusion

One of the main reasons for the success of CP-nets is the fact that we – *simultaneously* – have worked with:

**TOOLS**
- editing
- simulation
- verification

**THEORY**
- models
- basic concepts
- verification methods

**PRACTICAL USE**
- specification
- investigation
- verification
- implementation
More information on CP-nets

The following WWW pages contain a lot of information about CP-nets and their computer tools:

http://www.daimi.au.dk/CPnets/

A detailed introduction to CP-nets can be found in the following papers/books:


Some of the most important papers on high-level nets, their verification methods and applications have been reprinted in:


A list of papers that describe industrial use of CP-nets and their tools can be found on:

http://www.daimi.au.dk/CPnets/intro/example_indu.html
BEHAVIOUR
OF
ELEMENTARY
NET SYSTEMS

G. ROZENBERG

EN SYSTEM
\( \mathcal{N} = (B, E, F, C_{in}) \)

underlying
structure

initial state
(dynamic state
space)

potential
dynamics

actual
dynamics

\( \mathcal{N} \)

\( \mathcal{V} \)
ALL IS FINITE!

EN SYSTEMS ARE CONTACT-FREE

WHAT IS THE BEHAVIOUR?

HOW CAN THE BEHAVIOUR BE OBSERVED?
- Observations

5) Their records

- Behaviour

6) Observations

Sequential

Non-Sequential
SEQUENTIAL OBSERVATIONS

SEQUENTIAL OBSERVATIONS

\( q \in E^* \) is a firing sequence iff
\( q = \lambda \)
or
\( q = e_1 \ldots e_n \quad n \geq 1 \)
\( e_1, \ldots, e_n \in E \)

WHERE
\( (\exists C_0, C_1, \ldots, C_n \in \mathcal{C}_W) \)
\( C_0 [e_1] > C_1 [e_2] > \ldots > [e_n] C_n \parallel C_{in} \)

OBSERVED ARE:

OCCURRENCES OF SINGLE EVENTS
To analyze $FS(N^p)$, we need the sequential case graph of $N^p$, $SCG(N^p)$, obtained by deleting from $CG(N^p)$ all edges labeled by non-singleton steps.

$N_1^p$:

$C_{in} = \{ b_2, b_2 \}$
1) $\mathbf{FS}(N)$ is prefix closed

$\varrho \in \mathbf{FS}(N)$ \quad \{ \varrho_1 \in \mathbf{FS}(N) \}

$\varrho = \varrho_1 \varrho_2$

2) $\mathbf{SCG}(N)$ is finite

**Theorem**

$(\forall N) \quad \lim_{N \to \infty} \mathbf{FS}(N)$ is a prefix closed regular language
HOW TO EXTRACT (RECOVER) CAUSAL ORDERS FROM SEQUENTIAL OBSERVATIONS?

THEory of TRACES (MAZURKIEWICZ)

\[ \text{\# EN SYSTEM} \]

THE INDEPENDENCE RELATION INDUCED BY \( N \): \( I_N \)

\[ (\forall e_1, e_2 \in E_N) \]

\[ [(e_1, e_2) \in I_N \iff (e_1 u e_1^I) \cap (e_2 u e_2^I) = \emptyset] \]

THE DEPENDENCE RELATION INDUCED BY \( N \): \( D_N \)

\[ D_N = (E \times E) - I_N \]

\[ I_N \text{ is symmetric & irreflexive} \]

\[ D_N \text{ is symmetric & reflexive} \]
$q = \ldots e_1 e_2 \ldots \in FS(\mathcal{N})$

$\mu = \ldots e_2 e_1 \ldots$

$(e_1, e_2) \in I_{\mathcal{N}^0}$

\[
\frac{q \vdash_{I_{\mathcal{N}^0}} \mu}{q \equiv_{I_{\mathcal{N}^0}} \mu}
\]

$q \equiv_{I_{\mathcal{N}^0}} \mu \equiv_{I_{\mathcal{N}^0}} \gamma \ldots \equiv_{I_{\mathcal{N}^0}} \delta$

\[
\frac{q \equiv_{I_{\mathcal{N}^0}} \delta}{q \equiv_{I_{\mathcal{N}^0}} \delta}
\]

$\equiv_{I_{\mathcal{N}^0}}$ an equivalence relation on $FS(\mathcal{N})$
$Z \leq E^*_{\mathcal{N}}$ is $I_{\mathcal{N}_0}$-consistent
IFF $Z$ is a union of
equivalence classes
of $\equiv_{\mathcal{N}_0}$.

An equivalence class is called
a trace.

This $Z$ is $I_{\mathcal{N}_0}$-consistent.
This $Z$ is NOT $I_{\mathbb{N}}$-consistent.

(5) THEOREM

$(\forall \mathbb{N}) [FS(\mathbb{N})$ is $I_{\mathbb{N}}$-consistent.

IF $\eta$ OBSERVABLE IN $\mathbb{N}$, THEN EACH ELEMENT OF

$[\eta]_{I_{\mathbb{N}}}$ OBSERVABLE IN $\mathbb{N}$.

Each equivalence class $[\alpha]_{I_{\mathbb{N}}}$ is either included in $FS(\mathbb{N})$ or disjoint with $FS(\mathbb{N})$.

Those that are included are called (FIRING) TRACES $FT(\mathbb{N})$. 
SEQUENTIAL OBSERVATIONS

1. FIRING SEQUENCES
   Linear - difficult to interpret
   Break them down to
   DEPENDENCE GRAPHS
   Acyclic directed graphs

2. PARTIAL ORDERS

7. $\mathcal{N} \in \mathcal{FS}(\mathcal{N})$
   THE CANONICAL DEPENDENCE GRAPH
   OF $\varphi < \varphi >_{D_{\mathcal{N}}}$
   (i) $\varphi = \lambda$ $\rightarrow$ $< \varphi >_{D_{\mathcal{N}}}$ is empty
   (ii) $\varphi = e_1 \ldots e_n$, $n \geq 1$, $e_1, \ldots, e_n \in E_{\mathcal{N}}$
        $< \varphi >_{D_{\mathcal{N}}}$ is the $E_{\mathcal{N}}$-lab. graph $(V, Y, \varphi)$
        $V = \{1, \ldots, n\}$
        $(\forall i \in \{1, \ldots, n\}) [\varphi(i) = e_i]$
        $(\forall i, j \in \{1, \ldots, n\})$
        $[(i, j) \in Y \iff (i < j) \& (e_i, e_j) \in D_{\mathcal{N}}]$
3) \( \langle q \rangle_p \quad e = a \cdot b \cdot a \cdot d \)

(abstract dependence graph)

\[ \text{abstract dependence graph} \]

\[ \text{abstract dependence graph} \]

\[ \text{abstract dependence graph} \]
\[ q = abca \quad q' = acba \quad (b, c) \in I \]

\[ \langle q' \rangle_D \overset{\text{isom}}{=} \langle q \rangle_D \]

\[ \overline{\langle q' \rangle_D} = \overline{\langle q \rangle_D} \]

**THEOREM.** Let \( Z = (\Sigma, I, D) \) and let \( q, q' \in \Sigma^* \).

\[ q \overset{*}{\in} I \iff \overline{\langle q \rangle_D} = \overline{\langle q' \rangle_D} \]

\[ [q]_I = [q']_I \iff \overline{\langle q \rangle_D} = \overline{\langle q' \rangle_D} \]

\[ t \in \Theta(I) \rightarrow \overline{\langle t \rangle_D} \]

abstract dependence graph of \( t \)

\[ \text{adg}(t) \]
POSETS

$(A, R)$ \text{ ANTiSYM.}

\text{TRANSIT.}

\text{REFLEX.}

$g = (V, E)$ \text{ Dir.}

\text{ACYCL.}

\text{GRAPH}

\text{TRANS.}$\varphi$

\text{REFL. CLOS.}

$
\leq_g = (V, E^*)
$

$q \in \Sigma^*$

$\langle q \rangle_D$

$\langle \langle q \rangle \rangle_D$

$\langle q \rangle_D$

$\langle \langle q \rangle \rangle_D$

$\text{adg}(t)$

$\text{alp}(t)$

$\text{ADG}(T)$

$\text{ALP}(T)$
\[
\begin{align*}
\text{FS}(N) & \subseteq E^* \\
[\text{FS}(N)] & \text{ FIRING} \\
\text{I}_N & \text{ TRACES OF } N \\
\text{FT}(N) & \\
\text{ADG(FT}(N)) & \text{ ABSTRACT} \\
& \text{ FIRING DEP.} \\
& \text{GRAPHS OF } N \\
\text{AFD}(N) & \\
\text{ALP(FT}(N)) & \text{ ABSTRACT} \\
& \text{ FIRING LAB.} \\
& \text{ POSTSETS OF } N \\
\text{AFLP}(N) & 
\end{align*}
\]
\[ \mathcal{F}(\mathcal{W}) = e_1 e_5 e_3 e_4 e_5 \]

\[
[ e_1 e_5 e_3 e_4 e_5 ] = \begin{bmatrix}
I_5 & \mathcal{I} \\
-\mathcal{E} \times \mathcal{E} & -I_5
\end{bmatrix}
\]

\[ I_5 = \{(e_1, e_5), (e_5, e_1), (e_4, e_1), (e_3, e_5), (e_5, e_3)\} \]

\[ D_5 = \mathcal{E} \times \mathcal{E} - I_5 \]
$e_1 \quad e_3 \quad e_4 \quad e_5$

$e_5$

$\langle \langle \circ \rangle \rangle_D \quad N$

$\in \text{AFLP}(N)$

NON-SEQUENTIAL OBSERVATIONS
A net \( N = (S, T, F) \) is an occurrence net iff

\[(\forall \delta \in S) \left[ |\cdot \delta| \leq 1 \text{ AND } |S| \leq 1 \right] \]

\[S - \text{non-branching} \]

\[(\forall x, y \in X) \left[ (x, y) \in F^+ \Rightarrow (y, x) \notin F^+ \right] \]

\[\text{acyclic} \]

\[(\forall t \in T) \left[ t \cdot \neq \emptyset \right] \]
A NODE-LABELLED OCCURRENCE NET
\[ N = (S, T, F, \varphi) \]
\( (S, T, F) \) OCCURR. NET
\[ \varphi : \text{SUT} \to T \]
\( \varphi(S) \cap \varphi(T) = \emptyset \)
$N^p$ EN SYSTEM

$N=(S,T,H,\varphi)$ NODE-LAB.

OCCUR. NET $N$ IS A PROCESS OF $N^p$ IF

i) $\varphi(S) \subseteq B_{N^p}$ AND $\varphi(T) \subseteq E_N$.

ii) $(\forall s_1, s_2 \in S) \ [\varphi(s_1) = \varphi(s_2) \implies (s_1 \leq_N s_2) \lor (s_2 \leq_N s_1)]$

iii) $(\forall t \in T) \ [\varphi(t^*) = \varphi(t) \land \varphi(t^*) = \varphi(t)]$.

(iv) $\varphi(0_N) \subseteq C_{in}.$

$P(N^p)$
THEOREM

\[ N = (s, t, F, \Phi) \in P(N) \]

\[ S \subseteq S, \text{ a slice of } N \]

\[ \Phi(s) = C \]

\[ (s, t, F) \subseteq C \]

\[ \{ b_1, b_2 \} \]

\[ \{ b_1, b_2 \} \]

\[ b_2 \]

\[ b_1 \]

\[ e_2 \]

\[ b_3 \]

\[ o_4 \]

\[ b_1 \]

\[ b_2 \]

\[ b_5 \]

\[ o_7 \]

\[ D^1 \]
EN SYSTEM $\mathcal{N}$ IS REDUCED IFF ALL EVENTS OF $\mathcal{N}$ "VISIBLE" IN $\text{INSCG}(\mathcal{N})$

$E_{\mathcal{N}} = \bigcup_{u \in \mathcal{U}_{\mathcal{N}}} E_u$
EN SYSTEMS $N_1^2, N_2$

STRUCTURALLY SIMILAR

$N_1^2 \cong N_2$

und (N) isom und (N)

accordingly

$C_{in}^2, C_{in}^2$ RELATED

$g_2

$G_1 \xrightarrow{L_{isom}} G_2$

$\Sigma_1 \rightarrow \Sigma_2 \rightarrow G_1 \rightarrow G_2$

$E \xrightarrow{\Sigma_1 \rightarrow G_1 \rightarrow G_2}$
THEOREM

$\mathcal{N}_1^0, \mathcal{N}_2^0$ REDUCED ENS's

$P(\mathcal{N}_1^0) \cong P(\mathcal{N}_2^0)$

iff

$\mathcal{N}_1^0 \equiv \mathcal{N}_2^0$

PROCESS REPRESENTATION OF THE BEHAVIOUR OF AN EN SYSTEM IS TOO DETAILED

$g$ A BIPARTITE GRAPH

$g = (\mathcal{V}, \mathcal{W}, F)$

$W$-CONTRACTION OF $g$

$\overline{\text{ctr}}_W(g) = (\mathcal{V}, F')$

$F : \mathcal{W} \rightarrow$
NP an en system
N e P(N) S = S^n

\text{The } S\text{-contracted version of } N \text{ is a contracted process of } NP
CP(NP)

\text{The labeled poset } E \text{S}(NP) \text{ is an elementary event structure of } NP
THEOREM

\[ \frac{\not\exists N^2 \perp C \in E}{C P(N^2) \not\equiv L \text{ ISO } C P(N^2)} \]

AND

\[ N^1 \neq N^2 \]

\[ N^1 \neq N^2 \]
Diagram showing the process of sequence observations (SEQ, OBS) leading to EN NP, which further transitions to CP(NP) and subsequently to P(N). The CP(NP) also connects to EES(NP) and SE-CONTR. This diagram illustrates a model involving EN, NP, CP(NP), P(N), and EES(NP).
**Place/Transition Nets II**

M. Silva  
Universidad de Zaragoza

---

**OUTLINE**

1. **Introduction and motivation**
   
   1.1 Basic approaches: lead to (possibly weak) necessary or sufficient conditions.  
   
   1.2 On net subclasses: (sometimes) lead to necessary and sufficient conditions.  
   
   1.3 Nets and net systems: graph and convex geometry/linear algebra perspectives.

2. **Reduction**
   
   2.1 Basic concept: Net and marking transformation. Properties preserved by a rule.  
   
   2.2 A very simple kit of reduction rules preserving liveness and boundedness.  
   
   2.3 Implicit places: Basic concept and search technique.  
   
   2.4 Reduction on net subclasses can be complete: The free choice example.
1 Introduction and Motivation

1.1 Motivation and basic approaches

- State enumeration analysis.
- Reachability graph (for bounded systems).
- Coverability graph (leading to decision algorithms for unbounded systems).
- Problems: computationally complex (may be intractable).
- Valid only for a given initial marking.

New idea: Bridge behaviour and structure.

The basic approaches (considered here):

- Reduction: looking for “simpler” systems preserving the properties under study.
- Global structural approaches: Nets as graphs, nets as non-negative integer linear equations.

3 Structure Theory: A convex geometry/linear algebra perspective

3.1 Structural computation of the bound of a place.
3.2 Structural boundedness and structural liveness.

3.2.1 Conservation and Consistency
3.2.2 The Rank property

3.3 Boundedness and liveness for free choice systems. Some consequences.
Reachability graph: Behaviour Sequentialization

Idea: Exhaustive sequential state exploration

EXAMPLE: Adding p_6 does not change the reachability graph, but b and c cannot be concurrently fired.

- Reduction may be not complete for a given kit of reduction rules and a given net system. Therefore other analysis techniques are needed.
- Global structural analysis can be unable to decide on easy properties like boundedness (while decisions can be obtained for non-liveness for unbounded systems, for example).

Therefore: The practical analysis of general net system may need the cooperative use of several analysis techniques.

Nevertheless, necessary and sufficient conditions in the characterization of some properties are available for net subclasses (i.e. simpler nets).
1.2 **On net subclasses: (sometimes) lead to necessary and sufficient conditions**

For "simpler" systems, the analysis problem became "easier".

(e.g. continuous time, constant coefficient linear differential equations are easy to solve).

Simpler systems?

Are defined constraining the interleaving of concurrency and conflicts.

Only the most basic/classical net subclasses will be considered. They define the constraints in a syntactical way.
1.3 Nets and net systems

Net system ≡ Net structure +
+ distributed initial state (marking)

Discrete-event-dynamic-system:
* States → state variables: Places, P
* Events → state-transitions: Transitions, T

\[ N \] describes a net structure (weighted-bipartite directed graph):

\[ N = < P, T, F, W > \]

arc inscriptions:
* bulk arrivals
* bulk services

Ordinary net: weights equal to 1.

A second perspective for net structures:

\[ N = < P, T, Pre, Post > \]

Pre-incidence function, \( Pre(p,t) : P \times T \rightarrow N^+ (\{0,1\} \text{ for ordinary}) \)

Post-incidence function, \( Post(t,p) : P \times T \rightarrow N^+ (\{0,1\} \text{ for ordinary}) \)

Incidence matrices:

- Pre-incidence : \( C^-_{num} = Pre_{num} \)
- Post-incidence : \( C^+_{num} = Post_{num} \)
- Incidence : \( C_{num} = C^+ - C^- \)

where: \( n = |P| \)
\( m = |T| \)
\[ C^- = \begin{pmatrix}
  a & b & c & d & e & f \\
  p1 & 1 & 0 & 0 & 0 & 0 \\
  p2 & 0 & 1 & 0 & 0 & 0 \\
  p3 & 0 & 0 & 0 & 1 & 0 \\
  p4 & 0 & 0 & 1 & 0 & 0 \\
  p5 & 0 & 0 & 0 & 0 & 0 \\
  p6 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix} \]

\[ C^+ = \begin{pmatrix}
  a & b & c & d & e & f \\
  p1 & 0 & 0 & 0 & 1 & 0 \\
  p2 & 1 & 0 & 0 & 0 & 0 \\
  p3 & 0 & 1 & 0 & 0 & 0 \\
  p4 & 1 & 0 & 0 & 0 & 0 \\
  p5 & 0 & 0 & 1 & 0 & 0 \\
  p6 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix} \]

- \( N \) is the static structure
- \( M \) allows to represent the dynamic on \( N \)

The State Equation:

- \( M_{k-1} (t > M_k \Leftrightarrow M_k = M_{k-1} + C(t) = M_{k-1} + C^+ (t) - \bar{C}(t) \geq 0 \)
- Integrating along an execution (firing sequence):
  \[ M_0 \frac{\sigma}{\sigma} > M_k \Rightarrow M_k = M_0 + C \cdot \bar{\sigma} \]
  where \( \bar{\sigma} \) is the firing count vector of \( \sigma \).
- Very important: Unfortunately
  \[ M_k = M_0 + C \cdot \bar{\sigma} \geq 0 \quad \bar{\sigma} \geq 0 \quad \Rightarrow \quad M_0 \frac{\sigma}{\sigma} > M_k \]
- Therefore: convex geometry/linear algebra based analysis techniques cannot conclude always in general: semidecision.
2 Reduction of net models

2.1 Basic concept

\[ <N_0^i, M_0^i> \rightarrow <N_0^{i+1}, M_0^{i+1}> \]

while:

- properties under study (e.g. boundedness, liveness, ...) are preserved
- \( <N_0^{i+1}, M_0^{i+1}> \) is simpler to analyse (e.g. smaller reachability graph).

RULE:

- Structural precondition
- Marking precondition
- Structural change
- Marking change

APPLICATION: if preconditions are true (thus properties are preserved)

then make changes

PROBLEM:

- \( \exists \) irreducible systems given a reduction kit.

- Tradeoff: kit reduction power versus kit complexity
2.2 A very simple kit of reduction rules preserving liveness and boundedness

An illustrative example: A Producer-Consumer system with a Store operated under Mutual Exclusion
Two producers and the consumer with bounded buffer and mutual exclusion *(Note: The initialization is not considered).*

<table>
<thead>
<tr>
<th>PRODUCER 1</th>
<th>PRODUCER 2</th>
<th>CONSUMER</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>loop</em></td>
<td><em>loop</em></td>
<td><em>loop</em></td>
</tr>
<tr>
<td>produce 1</td>
<td>produce 2</td>
<td>P(OC)</td>
</tr>
<tr>
<td>P (Mutex)</td>
<td>P (Mutex)</td>
<td>P (OC)</td>
</tr>
<tr>
<td>P (EC)</td>
<td>P (EC)</td>
<td>V (EC)</td>
</tr>
<tr>
<td>deposit 1</td>
<td>deposit 2</td>
<td>withdraw</td>
</tr>
<tr>
<td>V (OC)</td>
<td>V (OC)</td>
<td>consume</td>
</tr>
<tr>
<td>V (Mutex)</td>
<td>V (Mutex)</td>
<td><em>endloop</em></td>
</tr>
<tr>
<td><em>endloop</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Petri net shown in Figure 2.7 is linear, bounded, and reversible.
2.3 Implicit places: Basic concept and search technique

Definition: A place $p$ is implicit in $<N, M_0>$ if never is the unique to constraint the firing of its output transitions.

Therefore: Removing implicit places does not change the set of firable sequences.

thus: removing implicit places preserve liveness, synchronic properties (lead, distance, slack, fairness, ...)

\[ 
\begin{array}{c}
\text{Mutex} \\
11 \\
prod 1 \\
12 \\
P(Mutex) \\
13 \\
P(EC) \\
14 \\
dep 1 \\
15 \\
V(OC) \\
16 \\
\end{array}
\]

\[ 
\begin{array}{c}
\text{Mutex} \\
21 \\
prod 2 \\
22 \\
P(Mutex) \\
23 \\
P(EC) \\
24 \\
dep 2 \\
25 \\
V(OC) \\
26 \\
\end{array}
\]

\[ 
\begin{array}{c}
\text{Mutex} \\
\end{array}
\]
Searching implicit places

\begin{align*}
(1) \quad v &= \min Y^T M_0 \\
& \text{s.t.} \quad C(p) = Y^T C \\
& \quad Y \geq 0
\end{align*}

(2) if $M_0(p) \geq v$ then $p$ is implicit \\
else ???

(a sufficient but not necessary condition)

---

Searching implicit places

Place $p$ is not structurally implicit

---

Remark. Any structurally implicit place $p$ (i.e. $C(p) = Y^T C$, $Y \geq 0$) can be made implicit just adding tokens (sooner or later $v$ will be smaller or equal than $Y^T M_0$)
2.4 Reduction on net subclasses can be complete: the free choice example

Two reduction rules with orthogonal rôles:

R1) Reduce some sequential (SM) subnets into places
R2) Remove some places defining concurrency.

R1) Is a particular case of the macroplace rule
R2) Is a particular case of the structurally implicit place rule

R1 & R2 lead to:

- SOUNDEDNESS
- COMPLETENESS: All LBFC can be reduced to a marked seed net (single place-with a selfloop transition).

RULE 1: Places with \( |p| = 1 \) can be refined into FLIPPER-SUBNETS (i.e. sequential subnets with one single way-out).

| * Liveness &
* The bound of the net (\( \Rightarrow \) boundedness) are preserved. |
**MARKING IMPLICIT**

\[ p \text{ is MSIP } \iff \exists Y \geq 0 \quad Y^T C = l_p \]

where:

* C is the incidence matrix
* \( l_p \) is the row representing \( p \)

RULE 2: MSIP with \( |p^*|=1 \) (to preserve FC?) can be added if no new \( P \)-semiflow of the new net is unmarked.

Now: SL is an interesting property, but what about liveness for a given \( M_0 \)?

### A SL & SB (i.e. lively and boundedly markable) FC net is

\[ \text{live for } M_0 \iff \text{all } p\text{-invariants are marked by } M_0 \]

**Equivalently:**

\[ \text{non live } \iff \exists Y \geq 0 \quad Y^T C = 0 \quad \& \quad Y^T M_0 = 0 \]

Therefore, for FC nets:

1. SL & SB can be characterized in CG/LA terms.
2. Liveness in SL & SB FC nets can also be characterized in CG terms.

As \( B \Leftrightarrow SB \) for live FC then:

\( L \& B \) can be decided in polynomial time.
3 Structure Theory: A Convex Geometry/Linear Algebra perspective

The key point: *The net state equation*

- $M_0 \{ \sigma > M \Rightarrow M = M_0 + C \cdot \sigma \}$

$M \in \mathbb{N}^n$, $\sigma \in \mathbb{N}^m$

- Unfortunately the reverse is *not* true:

*SPURIOUS SOLUTIONS*

What does we understand (today) about the behaviour of P/T systems using CG/LA?

- Some simple explanations & answers
- Some semidecision & decision techniques

Structural:

- Boundedness: boundedness $\forall M_0$
- Liveness: $\exists M_0$ making live the system
3.1 Structural computation of the bound of a place. Structural boundedness characterization

\[ SB(p) = \max M(p) \]
\[ \text{subject to } M = M_0 + C \cdot \bar{\sigma} \]
\[ M \geq 0, \quad \bar{\sigma} \geq 0 \]

(Primal) LPP

\[ SB(p) \] can be computed in polynomial time

The dual of the above problem: \( M(p) = e_p \cdot M \)

\[ SB(p) = \min Y^T \cdot M_0 \]
\[ \text{subject to } Y^T \cdot C \leq 0 \]
\[ Y \geq e_p \]

(Dual) LPP

Duality in LP:

- The primal LPP has always a solution: \( M = 0, \bar{\sigma} = 0 \)
- By duality the Dual LPP is:
  
  * bounded if the Primal LPP is bounded or
  * non feasible if the primal problem is unbounded.

\[ p \text{ is SB iff } \exists Y \geq e_p, \quad Y^T \cdot C \leq 0 \quad (\Rightarrow Y^T \cdot M \leq Y^T \cdot M_0) \]

\[ N \text{ is SB iff } \exists Y \geq 1, \quad Y^T \cdot C \leq 0 \]
3.2 Structural boundedness and structural liveness

3.2.1 Conservativeness and Consistency

\[ \text{SR}(t) = \max e_t \cdot \bar{\sigma} \]

subject to \( M = M_0 + C \cdot \bar{\sigma} \)

\[ M \geq 0, \bar{\sigma} \geq 0 \]

Thus (through duality & boundednes in LP):

\[ t \text{ is } \text{SR iff } \exists X \geq e_t \text{ such that } C \cdot X \geq 0 \]

\[ N \text{ is SR iff } \exists X \geq 1 \text{ such that } C \cdot X \geq 0 \]

And it is possible to write

\[ N \text{ is } SB \land SL \implies N \text{ is } SB \land SR \iff N \text{ is } Cv \land Cv \]

where:

\[ N \text{ is } Cv \iff \exists Y \geq 1, Y^T \cdot C = 0 \]

\[ N \text{ is } Ct \iff \exists X \geq 1, C \cdot X = 0 \]
3.2.2 The Rank Property

It is possible to improve the knowledge about SL in SB nets?

Definition: $t_a$ and $t_b$ are in equality conflict relation (ECR) iff $\text{Pre}(t_a) = \text{Pre}(t_b)$

ECR is an Equivalence Relation:

Let:

- $D_i$ be an equivalence class
- $\delta_i = |D_i| - 1$
- $\delta = \sum \delta_i$

Property: If $N$ is SB & SL then:

- $N$ is $C_r$ & $C_f$
- $\text{rank}(C) \leq |T| - 1 - \delta$

\[
\begin{align*}
\text{rank}(C) &= 4 \\
|T| - 1 - \delta &= 5 - 1 - 1 = 3 \\
\text{rank}(C) &= 4 \\
|T| - 1 - \delta &= 7 - 1 - 2 = 4
\end{align*}
\]

SL? $N$ is not struct-Live

SL? No answer
3.3 Boundedness and liveness for free choice systems. Some consequences.

Property: FC Nets: \( L \& B \not\Rightarrow SL \& SB \)

Remark: For general live P/T nets \( B \) does not imply \( SB \).

\[ SL \& SB \] is characterizable for FC nets

The FC net \( N \) is \( SL \& SB \) iff
- \( N \) is \( C_t \& C_v \)
- \( \text{rank}(C) = m - 1 - \delta = m - 1 - (a - n) \)

\[ \text{where } a = \# \text{ arcs in Pre} \]

Some consequences for FC nets:

1. \( SL \& SB \) can be computed in polynomial time
2. Hack's Duality Theorem:
   Let \( N \) be a FC net and \( N_{rd} \) its reverse-dual
   \[ N \text{ is } SL \& SB \Leftrightarrow N_{rd} \text{ is } SL \& SB \]

3. Soundedness of the primal and dual complete reduction kits for LBFC

Concluding Remarks

A number of analysis techniques for P/T nets have been presented:
- property preserving reduction of nets
- convex geometry / linear algebra
- graph theoretical arguments

For distinguished net subclasses efficient analysis algorithms are available.

In general: The analysis problem should be considered using different analysis techniques in a COOPERATIVE way.

Interleaving of FUNCTIONAL and PERFORMANCE (structural) analysis.
AN INTRODUCTION TO
GENERALIZED STOCHASTIC PETRI NETS

Gianfranco Balbo
Dipartimento di Informatica
Università di Torino

Outline

TIMED PETRI NETS
- Timed places, tokens, arcs, transitions
- Race and preselection
- Memory
- Single and multiple server semantics

STOCHASTIC PETRI NETS
- The exponential distribution
- Markov chains
- Isomorphism between SPNs and MCs
- Example

GENERALIZED SPNs
- Immediate transitions and priority
- GSPN definition
- Extended conflict sets
- Isomorphism between GSPNs and MCs
- Performance indices
- Example

AN APPLICATION OF GSPNS

CONCLUSIONS
**Prerequisites**

- The basic definitions of Petri net theory
  - places, transitions, arcs, tokens
  - marking
  - enabling, firing, reachability
  - (enabling degree)
  - conflict, confusion
  - (invariants)
- Some elementary notions of probability theory
  - random variable
  - stochastic process
  - pdf, PDF
  - state space
  - averages
  - sojourn times
  - (ergodicity)
  - (Little’s formula)

Items in parentheses are optional.
**Timing specifications**

Time is introduced in Petri nets to model the interaction among several activities considering their starting and completion times.

The introduction of time specifications corresponds to an interpretation of the model by means of

- observation of the autonomous (untimed) model
- definition of a non-autonomous model

Time specifications should provide

- consistency among autonomous and non-autonomous models
- non-determinism reduction on the basis of time considerations
- support for the computation of performance indices

**Timed places**

Several approaches are possible for the introduction of temporal specifications in PN models:

- time may be associated with places (TPPN):
  - tokens generated in an output place become available to fire a transition only after a delay has elapsed; the delay is an attribute of the place
Timed tokens

- time may be associated with tokens:
  - tokens carry a time stamp that indicates when they are available to fire a transition; this time stamp can be incremented at each transition firing.

![Timed tokens diagram](image)

\[ \theta_1 = \theta_1 + \Delta \theta_1 \]

Timed arcs

- time may be associated with arcs:
  - a travelling delay is associated with each arc; tokens are available for firing only when they reach a transition.

![Timed arcs diagram](image)
Timed transitions

- time may be associated with transitions (TTPN);
  transitions represent activities
  - activity start corresponds to transition enabling,
  - activity end corresponds to transition firing

Different firing policies may be assumed:
- three-phase firing
  1. tokens are consumed from input places when the transition is enabled
  2. the delay elapses
  3. tokens are generated in output places
- atomic firing
  tokens remain in input places for the transition delay; they are consumed from input places and generated in output places when the transition fires

Atomic firing

We shall consider TTPN with atomic firing.

TTPN with atomic firing can preserve the basic behaviour of the underlying untimed model.

It is thus possible to qualitatively study TTPN with atomic firing exploiting the theory developed for untimed (autonomous) PN (reachability set, invariants, etc.).

Timing specifications may affect the qualitative behaviour of the PN when they describe constant and interval firing delays.
Internal timer

We can explain the behaviour of one timed transition with atomic firing by assuming that it incorporates a timer.

- When the transition is enabled, its timer is set to the current delay value
- Then, the timer is decremented at constant speed, until it reaches the value zero
- At this point the transition fires

Conflicts

When more than one timed transition with atomic firing is enabled, the behaviour is similar, but a problem arises:

*Which one of the enabled transitions is going to fire?*
Selection rules

Two alternative selection rules:

- preselection:
  the enabled transition that will fire is chosen when
  the marking is entered, according to some metric
  (priority, probability, ...)

- race:
  the enabled transition that will fire is the one whose
  firing delay is minimum

Memory policies

When a timed transition is disabled by a conflicting transition, a problem arises:

*How is the transition timer set when the transition will again become enabled?*

*How does the transition keep memory of its past enabling time?*
Basic mechanisms

Two basic mechanisms can be defined:

- Continue:
The timer associated with the transition holds the present value and will continue later on the countdown.

- Restart:
The timer associated with the transition is restarted, i.e., its present value is discarded and a new value will be generated when needed.

Transition memory policies

From the two basic mechanisms it is possible to construct several transition memory policies; the usual ones are:

- Resampling:
  - At each and every transition firing, the timers of all timed transitions in the timed PN system are discarded (restart mechanism).
  - No memory of the past is recorded.
  - After discarding all timers, new values of the timers are set for the transitions that are enabled in the new marking.
• Enabling memory:
  - At each transition firing, the timers of all timed transitions that become disabled are restarted, whereas the timers of all timed transitions that remain enabled hold their present value (continue mechanism).

  - The memory of the past is recorded with an enabling memory variable associated with each transition.

  - The enabling memory variable accounts for the work performed by the activity associated with the transition since the last instant of time when its timer was set.

  The enabling memory variable measures the enabling time of the transition since the last instant of time it became enabled.

• Age memory:
  - At each transition firing, the timers of all timed transitions hold their present values (continue mechanism).

  - The memory of the past is recorded with an age memory variable associated with each timed transition.

  The age memory variable accounts for the work performed by the activity associated with the transition since the time of its last firing.

  - The age memory variable measures the cumulative enabling time of the transition since the last instant of time when it fired.
The *enabling degree* of a transition is the number of times the transition could fire in the given marking before becoming disabled.

When the enabling degree of a transition is $> 1$, attention must be paid to the timing semantics.

Three cases are common:

- Single-server semantics
- Infinite-server semantics
- Multiple-server semantics
**Single-server semantics:**

A firing delay is set when the transition is first enabled, and new delays are generated upon transition firing if the transition is still enabled in the new marking.

Enabling sets of tokens are processed *serially* and the temporal specification associated with the transition is independent of the enabling degree;

**Infinite-server semantics:**

Every enabling set of tokens is processed as soon as it forms in the input places of the timed transition.

Its corresponding firing delay is generated at this time, and the timers associated with all these enabling sets run down to zero concurrently.

Multiple enabling sets of tokens are thus processed *in parallel*.

The overall temporal specifications of transitions with this semantics depend directly on their enabling degrees.
Multiple-server semantics:

Enabling sets of tokens are processed as soon as they form in the input places of the transition up to a maximum degree of parallelism (say $K$).

For larger values of the enabling degree, the timers associated with new enabling sets of tokens are set only when the number of concurrently running timers decreases below the value $K$.

The overall temporal specifications of transitions with this semantics depend directly on their enabling degrees up to a threshold value $K$.

---

Example of server semantics

Consider a timed transition with enabling degree equal to 3.

The three enablings are associated with firing delays equal to 3, 2, and 4 time units.
Queueing policies

Upon firing of a transition, input tokens are removed at random.

Specific queueing policies must be explicitly represented at model level.

Firing and selection rules

We consider TTPN with atomic firing and race selection rule.

Transitions within one TTPN can use

- Resampling
- Enabling memory
- Age memory

and

- Single-server semantics
- Infinite-server semantics
- Multiple-server semantics

in any combination.
Timed Transition PN with atomic firing in which all transition delays are *random variables* with negative exponential distributions are called *Stochastic PN (SPN)*.

The dynamic behaviour of a SPN is described through a *stochastic process*.
Definitions

A random variable is a real function defined over a probability space.

Stochastic processes are mathematical models useful for the description of phenomena of a probabilistic nature as a function of a parameter that usually has the meaning of time.

A stochastic process \( \{X(t), t \in T\} \) is a family of random variables defined over the same probability space, indexed by the parameter \( t \) and taking values in the state space \( S \).

Stochastic processes

A sample path (or realization) of a stochastic process is a function of time.
The probabilistic description of a random variable $X$ is given by its probability density function (pdf)

$$f_X(x) = \frac{d}{dx} P\{X \leq x\} \quad -\infty < x < \infty$$

The probabilistic description of a random process is given by the joint pdf of any set of random variables extracted from the process.

$$P\{X(t_1) \leq x_1, X(t_2) \leq x_2, ..., X(t_n) \leq x_n\}$$

In the general case the complete probabilistic description of a random process is not feasible.

MARKOVIAN processes are one special class of stochastic processes for which the probabilistic description is simpler and of particular relevance.

A process that satisfies the Markov property:

$$P\{X(t) \leq x | X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, \cdots\}$$

$$X(t_0) = x_0 = P\{X(t) \leq x | X(t_n) = x_n\}$$

with $t > t_n > t_{n-1} > \cdots > t_0$ is called a Markovian process.

If the state space is denumerable, the process is a Markov chain.

If the parameter $t$ is continuous, the process is a continuous-time Markov chain (CTMC).
A continuous-time Markov chain (CTMC) is a stochastic process where

- sojourn times in states are exponentially distributed random variables
- the future evolution depends only on the present state, not on the past history

The exponential pdf

\[ f_X(x) = \lambda e^{-\lambda x} \quad (x \geq 0) \]

is the only continuous pdf for which the memoryless property holds:

\[ P\{X > x + \alpha | X > \alpha\} = P\{X > x\} \]
The exponential pdf

\[ f_X(x) = \lambda e^{-\lambda x} \quad (x \geq 0) \]

is defined only by its rate \( \lambda \), which is the inverse of its average value:

\[ E[X] = \frac{1}{\lambda} \]

Given two random variables \( X \) and \( Y \) with exponential pdf

\[ f_X(x) = \lambda e^{-\lambda x} \quad (x \geq 0) \]
\[ f_Y(y) = \mu e^{-\mu y} \quad (y \geq 0) \]

the new random variable \( Z = \min(X, Y) \) also has an exponential pdf

\[ f_Z(z) = (\lambda + \mu)e^{-(\lambda + \mu)z} \quad (z \geq 0) \]

In fact,

\[
F_Z(z) = 1 - Pr\{Z > z\} = 1 - Pr\{X > z, Y > z\} = 1 - e^{-\lambda z}e^{-\mu z} = 1 - e^{-(\lambda + \mu)z} \quad (z \geq 0)
\]
The residual sojourn time in a state of a Markov chain is a random variable with the same distribution as the whole sojourn time.

A CTMC can be described through a state transition rate diagram, or equivalently with a state transition rate matrix, also called infinitesimal generator, denoted by $Q$.

$$Q = \begin{bmatrix} -\lambda & \lambda \\ \mu & -\mu \end{bmatrix}$$
The solution of a CTMC at time $t$ is the probability distribution over the set of states:

$$\pi(t) = (\pi_1(t), \pi_2(t), \pi_3(t), \cdots)$$

with

$$\pi_i(t) = P\{X(t) = i\}$$

It can be proven that

$$\frac{d\pi(\tau)}{d\tau} = \pi(\tau)Q$$

whose solution can be formally written as

$$\pi(t) = \pi(0)H(t)$$

with

$$H(t) = e^{Qt}$$

This is a very elegant solution that is however usually very expensive to compute since the matrix exponentiation is defined by the following infinite sum

$$e^{Qt} = \sum_{k=0}^{\infty} \frac{(Qt)^k}{k!}$$

The solution of a CTMC at steady-state is the probability distribution over the set of states.

The steady-state distribution exists for ergodic CTMCs.

The steady-state distribution

$$\pi = (\pi_1, \pi_2, \pi_3, \cdots)$$

with

$$\pi_i = \lim_{t \to \infty} P\{X(t) = i\}$$

is computed as the solution of the linear system of equations

$$\pi Q = 0$$

with the normalizing condition

$$\sum_i \pi_i = 1$$
Definition of stochastic Petri nets

Formally, an SPN is defined through an 8-tuple:

\[ \text{SPN} = (P, T, I(\cdot), O(\cdot), H(\cdot), W(\cdot), M_0) \]

where

- \( \text{PN} = (P, T, I(\cdot), O(\cdot), H(\cdot), M_0) \) is the marked PN underlying the SPN
- \( W(\cdot) \) is a function defined on the set of transitions that associates a rate with each transition. This rate is the inverse of the average firing time of the transition

SPNs can be proved to be isomorphic to CTMCs: the reachability graph of the SPN corresponds to the state transition rate diagram of the MC.

This can be easily seen in the case of simple subclasses of Petri nets such as: Finite State Machines and Marked Graphs.
SPNs without choices and synchronizations

- The net has the structure of both a Finite State Machine (no transition has more than one input and one output place) and of a Marked Graph (no place has more than one input and one output transition);

- the initial marking contains only one token

Each place of the net univocally identifies a state of the net.

Each place of the net maps into a state of the corresponding probabilistic model.

The time spent by the token in each of its places is completely determined by the characteristics of the only transition that can withdraw it from that place.

The probabilistic model that represents the behaviour of the net (Marking Process) is a CTMC
SPNs with choices

- The net has the structure of a Finite State Machine (no transition has more than one input and one output place);

- the initial marking contains only one token

Conflicts arise when several transitions share a common input place.

A race starts among the simultaneously enabled transitions.

The race is won by one of the transitions and the way of dealing with the partially completed activities of the transitions that were interrupted becomes an issue (in general).

When the firing times of the transitions of the net have negative exponential distributions, their memoryless property makes the distinction among

- resampling
- enabling memory
- age memory

irrelevant and the CTMC corresponding to the SPN is obtained from the net in a straightforward manner.

\[
\begin{array}{cccc}
\lambda & \mu & 0 & 0 \\
T_1 & p_1 & 1 & 0 \\
T_2 & p_2 & 0 & 1 \\
T_3 & p_3 & 0 & 0 \\
T_4 & p_4 & 0 & 0 \\
T_5 & p_5 & 0 & 0 \\
\end{array}
\]
More complex situations arise already when several tokens are allowed in the initial markings of these simple models.

- Service semantics adopted when the input place of a transition contains several tokens;
- Queueing policy assumed with respect to the tokens residing in the input place of a transition.

**A simple example**

![Diagram](attachment:image.png)

- **IS (on both transitions)**
  - Transition $T_1$ with input place $p_1$ and output place $p_2$.
  - Transition $T_2$ with input place $p_2$ and output place $p_1$.

- **SS (on both transitions)**
  - Transition $T_1$ with input place $p_1$ and output place $p_2$.
  - Transition $T_2$ with input place $p_2$ and output place $p_1$. 
Queueing policy

It is possible to show that when the firing times are exponentially distributed and the performance figures of interest are only related to the moments of the number of tokens in the input place of a transition many queueing policies yield the same results and thus the random order (that is the most natural in the Petri net context) can be assumed.

In general, the CTMC associated with a given SPN system is obtained by applying the following simple rules:

1. The CTMC state space $S = \{s_i\}$ corresponds to the reachability set $RS(m_0)$ of the PN associated with the SPN $(m_i \rightarrow s_i)$.

2. The transition rate from state $s_i$ (corresponding to marking $m_i$) to state $s_j$ ($m_j$) is obtained as the sum of the firing rates of the transitions that are enabled in $m_i$ and whose firings generate marking $m_j$.

Assuming that all the transitions of the net operate with a single-server semantics and marking-independent speeds, and denoting with

- $Q$ the infinitesimal generator,
- $w_k$ the firing rate of $T_k$,
- $e_j(m_i) = \{ h : T_h \in e(m_i) \land m_i[T_h] = m_j \}$ the set of transitions that bring the net from $m_i$ to $m_j$,

the components of $Q$ are:

$$q_{ij} = \begin{cases} 
\sum_{T_k \in e_J(m_i)} w_k & i \neq j \\
-q_i & i = j 
\end{cases}$$

where

$$q_i = \sum_{T_k \in e(m_i)} w_k$$
**Performance indices**

The steady-state distribution $\pi$ is the basis for a quantitative evaluation of the behaviour of the SPN expressed in terms of performance indices.

These results can be computed using a unifying approach in which proper index functions (also called *reward functions*) are defined over the markings of the SPN and an average reward is derived using the steady-state probability distribution of the SPN.

Assuming that $r(m)$ represents one of such reward functions, the average reward can be computed using the following weighted sum:

$$ R = \sum_{m_i \in RS(m_0)} r(m_i) \pi_i $$

**Probability of a particular condition $\Upsilon(m)$ of the SPN.**

Define the following reward function:

$$ r(m) = \begin{cases} 1 & \Upsilon(m) = true \\ 0 & \text{otherwise} \end{cases} $$

The desired probability is computed using the following expression:

$$ P\{\Upsilon\} = \sum_{m_i \in RS(m_0)} r(m_i) \pi_i = \sum_{m_i \in A} \pi_i $$

where $A = \{m_i \in RS(m_0) : \Upsilon(m_i) = true\}$. 
Expected value of the number of tokens in a given place.

In this case the reward function is:

\[ r(m) = n \text{ iff } m(p_j) = n \]

The expected value of the number of tokens in \( p_j \) is given by:

\[ e[m(p_j)] = \sum_{m_i \in RS(m_0)} r(m_i) \pi_i = \sum_{n>0} [n \ P\{A(j,n)\}] \]

where \( A(j,n) = \{m_i \in RS(m_0) : m_i(p_j) = n\} \) and the sum is obviously limited to values of \( n \leq k \); if the place is \( k \)-bounded.

Mean number of firings per unit of time of a given transition.

A transition may fire only when it is enabled, thus the reward function assumes the following form:

\[ r(m) = \begin{cases} w_j & T_j \in e(m) \\ 0 & \text{otherwise} \end{cases} \]

The mean number of firings of \( T_j \) per unit of time is then given by:

\[ f_j = \sum_{m_i \in RS(m_0)} r(m_i) \pi_i = \sum_{m_i \in A_j} w_j \pi_i \]

where \( A_j = \{m_i \in RS(m_0) : T_j \in e(m_i)\} \).
The average steady-state delay spent in traversing a subnet can be computed from Little’s formula

\[ E[T] = \frac{E[N]}{E[S]} \]

where \( E[N] \) is the average number of (equivalent) tokens in the subnet, and \( E[S] \) is the average input rate into the subnet.

Delay distributions are in general difficult to compute.
Computation of $\mu_1$, $\mu_2$:

- Total rate out of $\text{P3} + \text{P4}$ is:
  \[ W(T_{\text{par1}}) + W(T_{\text{par2}}) \]

- With what probability $T_{\text{par1}}$ is the first to fire?
  \[ \frac{W(T_{\text{par1}})}{W(T_{\text{par1}}) + W(T_{\text{par2}})} \]

- Therefore:
  \[ \mu_1 = \frac{W(T_{\text{par1}}) + W(T_{\text{par2}})}{W(T_{\text{par1}}) + W(T_{\text{par2}})} \]
  \[ = W(T_{\text{par1}}) \]
Computation of $\lambda_{OK}$, $\lambda_{KO}$:

same computation as before:

\[ \lambda_{OK} = W(T_{OK}) \]
\[ \lambda_{KO} = W(T_{KO}) \]

but...

what is the meaning of $W(T_{OK})$ and $W(T_{KO})$?

→ check activity: 0.0001

→ rate of 10,000

→ probability of $OK/KO$ is 99% vs. 1%

→ $W(T_{OK}) = 9,900$ and $W(T_{KO}) = 100$

Parameter specifications:

<table>
<thead>
<tr>
<th>transition</th>
<th>rate</th>
<th>value</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{newdata}$</td>
<td>$\lambda$</td>
<td>1</td>
<td>infinite-server</td>
</tr>
<tr>
<td>$T_{start}$</td>
<td>$\tau$</td>
<td>1000</td>
<td>single-server</td>
</tr>
<tr>
<td>$T_{par1}$</td>
<td>$\mu_1$</td>
<td>10</td>
<td>single-server</td>
</tr>
<tr>
<td>$T_{par2}$</td>
<td>$\mu_2$</td>
<td>5</td>
<td>single-server</td>
</tr>
<tr>
<td>$T_{syn}$</td>
<td>$\sigma$</td>
<td>2500</td>
<td>single-server</td>
</tr>
<tr>
<td>$T_{OK}$</td>
<td>$\alpha$</td>
<td>9900</td>
<td>single-server</td>
</tr>
<tr>
<td>$T_{KO}$</td>
<td>$\beta$</td>
<td>100</td>
<td>single-server</td>
</tr>
<tr>
<td>$T_{I/O}$</td>
<td>$\nu$</td>
<td>25</td>
<td>single-server</td>
</tr>
<tr>
<td>$T_{check}$</td>
<td>$\theta$</td>
<td>0.5</td>
<td>single-server</td>
</tr>
</tbody>
</table>

The consistency check operation has an average duration 0.0001 time units, and results in a success 99% of the times, and in a failure 1% of the times.
Performance indices:

- Throughput of transition $T_{I/O}$:
  - 1.504 success/time units

- Average number of items under test:
  - 0.031

- Average production time:
  - 0.33 time units
Two classes of transitions exist in GSPNs:

- **timed** transitions, whose delays are exponentially distributed random variables (like in SPNs)
- **immediate** transitions, whose delays are deterministically zero

Immediate transitions have been introduced in the model

- to account for instantaneous actions (e.g. choice among classes of clients);
- to implement specific modelling features (e.g. to empty a place);
- to account for time scale differences (e.g. bus arbitration and I/O accesses).

Immediate transitions have priority over timed transitions.

Several priority levels for immediate transitions can be defined. Immediate transitions at priority level $n$ are called $n$-immediate.

The autonomous model associated with a GSPN is a *Petri net with priorities*.

A transition $t$ is said to have *concession* in marking $M$ iff $M \geq I(t) \land M < H(t)$.

A transition $t_j$ is defined to be enabled in marking $M$ iff it has concession in $M$ and $\forall t_k \in T$ that have concession in $M$, $\pi_j \geq \pi_k$. 
Effects induced by the presence of priorities

\[ \Sigma \iff \Sigma_\pi \]

- Properties
  - safety (invariant): must hold in all states
  - eventuality (progress): must hold in some state

- \( RS(\Sigma) \supseteq RS(\Sigma_\pi) \)
  - safety properties are maintained (absence of deadlocks, boundedness, mutual exclusion, …)
  - eventuality properties are not necessarily maintained (reachability, liveness, …)

Effects induced by the presence of priorities

\[ \Sigma \iff \Sigma_\pi \]

- Reachability
  - \( M \in RS(\Sigma) \not\implies M \in RS(\Sigma_\pi) \)
  - but \( M \in RS(\Sigma_\pi) \implies M \in RS(\Sigma) \)

- Boundedness
  - \( \Sigma \) bounded \( \implies \Sigma_\pi \) bounded
  - but \( \Sigma \) not bounded \( \not\implies \Sigma_\pi \) not bounded

- Liveness - home states
  - priority can introduce or remove liveness
Some fine points:

1. Need for priority

2. Irrelevance of distinction between resampling - enabling - age due to the memoryless property of exponential distributions

3. Impossibility of two timers to expire at the same time
   probability of extracting a specific sample $x$ is equal to zero

Markings that enable timed transitions only are said to be \textit{tangible}, whereas markings that enable $n$-immediate transitions are said to be \textit{vanishing}.
Formally, a GSPN is an 8-tuple:

$$\text{GSPN} = \langle P, T, \Pi(\cdot), I(\cdot), O(\cdot), H(\cdot), W(\cdot), M_0 \rangle$$

where

- $\text{PN}_\pi = \langle P, T, \Pi(\cdot), I(\cdot), O(\cdot), H(\cdot), M_0 \rangle$ is the marked PN with priority underlying the GSPN
- $W(\cdot)$ is a function defined on the set of transitions

The subnets formed by $n$-immediate transitions must be confusion-free.

The function $W(\cdot)$ allows the definition of the stochastic component of a GSPN model. In particular, it maps transitions into real positive numbers.

The quantity $W(t_k) = w_k$ is called

- the “rate” of transition $t_k$ if $t_k$ is timed
- the “weight” of transition $t_k$ if $t_k$ is $n$-immediate

Rates are used like in SPNs.

Weights are used for the probabilistic resolution of conflicts of immediate transitions.
When a tangible marking is entered, the timed transitions that become enabled for the first time since their last firing, sample a firing delay instance and set their timer to the sampled value.

Then, all timers of the enabled (timed) transitions are decremented at equal speed, until one of them reaches the value zero.

At this point the transition whose timer reached zero fires.

All the transitions that did not fire keep their timer readings, and their timers will be again decremented in the next marking in which the transition is enabled.

(Enabling memory was used in the description, but is irrelevant)

When a vanishing marking is entered, the weights of the enabled $n$-immediate transitions are used to probabilistically select the ($n$-immediate) transition(s) to fire. The time spent in any vanishing marking is deterministically equal to zero.
The definition of weights requires the identification of the sets of immediate transitions that can be simultaneously enabled in conflict.

Such sets of transitions are called Extended Conflict Sets (ECSs).

When all the ECSs in a GSPN are known, the association of weights to transitions is easy, provided that no confusion exists.

The structural and behavioural analysis of the marked PN with priority underlying the GSPN allows the qualitative study of the GSPN behaviour, and in particular the identification of

- ECS
- confusion

\[
\begin{align*}
W(t_1) &= 10 \\
W(t_2) &= 20 \\
W(t_3) &= 44 \\
\end{align*}
\]
Confusion destroys the locality of conflicts

Confusion produces different transition probabilities
In the case of vanishing markings, the weights of the enabled $n$-immediate transitions can be used to determine which one will actually fire in a marking $M$ that enables more than one conflicting $n$-immediate transitions.

When several transitions belonging to the same ECS are the only ones enabled in a given marking, one of them, say transition $t_i$, is selected as a candidate to fire with probability:

$$\mathbf{P}\{ t_i \mid M \} = \frac{w_i}{W_I(M)}$$

where $W_I(M)$ is the weight of $ECS(t_i)$ in marking $M$, and is defined as follows:

$$W_I(M) = \sum_{k: t_k \in ECS(t_i) \cap E(M)} w_k$$

It may however happen that several ECSs comprising transitions of the same priority level are simultaneously enabled in a vanishing marking.

The characteristic of the subnets of $n$-immediate transitions of being confusion-free guarantees that the way in which this choice is performed is irrelevant with respect to the resulting stochastic model.
GSPNs can be proved to be isomorphic to Semi-Markov processes.

The analysis of a GSPN can be performed by studying a CTMC.

The state transition rate diagram of the MC corresponds to the tangible reachability graph of the GSPN.

The memoryless property of the exponential distribution makes the distinction among

- resampling
- enabling memory
- age memory

irrelevant.

The sojourn time in a tangible marking is exponentially distributed with a parameter that is the sum of the rates of all enabled timed transitions, so that the average time spent in marking $M$ is given by:

$$E[SJ(M)] = \left[ \sum_{t \in E(M)} W(t) \right]^{-1}$$
Numerical solution of GSPN models

An embedded Markov chain (EMC) can be recognized disregarding the concept of time and focusing the attention on the set of states of the semi-Markov process.

The specifications of a GSPN system are sufficient for the computation of the transition probabilities of such a chain.

Several techniques have been devised for restricting the computation to reduced models accounting for the tangible markings only.

Three different approaches can be used:

- Identify a reduced Embedded Markov Chain defined over the set of tangible markings only;
- Compute the transition probabilities among tangible markings directly by applying (on-the-fly) a depth-first algorithm that explores all complete vanishing paths emanating from each tangible state. The method assumes that no loops among vanishing states exist and memory saving is traded-off with (possible) repeated computations;
- Reduce the GSPN to an equivalent SPN obtained by fusing immediate transitions with preceding timed transitions using an algorithm that in the simple cases produces the following reductions
Computational considerations

The mathematically elegant solution of the model using the REMC suffers in practice of the difficulties deriving from the size of the CTMC and from time-scale differences that may exist among the firing rates of the transitions of a model.

Approaches that can be used to overcome these difficulties are the following:

- Transient solution
  - Uniformization method

- Steady state solution
  - Time scale decomposition
  - Tensor algebras and compositionality
  - Symmetries and exact lumping

- Simulation

Performance indices

From the steady-state probability distribution of markings it is possible to obtain several performance indices that are the basis for a quantitative evaluation of the behaviour of the GSPN.

As in the case of SPNs, these results can be computed using the unifying approach based on the definition of reward functions.
The applicability of the [(G)S]PN approach to anything but the smallest toy examples rests on the availability of efficient tools for the

- model construction (**top-down, bottom-up, compositionality**)
- model debugging (**structural analysis**)
- definition of performance indices
- model solution (**analysis and/or simulation**)
- computation of aggregate results
- display of results

Good software tools are a must.

The user-friendliness and the graphical capabilities of the tool are of paramount importance.
The consistency check operation results in a success 99% of the times, and in a failure 1% of the times.

<table>
<thead>
<tr>
<th>transition</th>
<th>rate</th>
<th>value</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{newdata}$</td>
<td>$\lambda$</td>
<td>1</td>
<td>infinite-server</td>
</tr>
<tr>
<td>$T_{par1}$</td>
<td>$\mu_1$</td>
<td>10</td>
<td>single-server</td>
</tr>
<tr>
<td>$T_{par2}$</td>
<td>$\mu_2$</td>
<td>5</td>
<td>single-server</td>
</tr>
<tr>
<td>$T_{I/O}$</td>
<td>$\nu$</td>
<td>25</td>
<td>single-server</td>
</tr>
<tr>
<td>$T_{check}$</td>
<td>$\theta$</td>
<td>0.5</td>
<td>single-server</td>
</tr>
</tbody>
</table>

The GSPN model generates

- 20 tangible markings
- 18 vanishing markings

As an example, with the numerical values chosen for the model parameters, the probability of at least one process waiting for synchronization is computed to be 0.238.
A case study

a kanban system

A kanban cell and the parts and cards that flow into and out of it

The basic model

GSPN model of a single Kanban cell
A $n$-cell Kanban model has $2n$ minimal P-semiflows, whose associated P-invariants are:

\[ \forall i, 1 \leq i \leq n: \]
\[ M(BB_i) + M(IB_i) + M(busyM_i) + M(OB_i) = K_i \]
\[ M(idleM_i) + M(busyM_i) = 1 \]

It follows that:
- The number of parts in cell $i$ is at most $K_i$, the number of cards in the cell;
- Each machine can process only one part at a time;
- Places $idleM_i$ and $busyM_i$ are mutually exclusive.
All transitions of the GSPN model are covered by a single minimal T-semiflow: it represents the deterministic flow of the unique type of parts processed by the system.

The net behaviour is deterministic: no structural conflicts exist, hence neither effective conflicts nor confusion can ever arise.

We consider $K$ cards and $n = 5$ cells of equal machine time (the rate of transitions $outM_i$ is 4.0)

First case: Input and output inventory in the cells

<table>
<thead>
<tr>
<th></th>
<th>Input buffer inventory</th>
<th>Output buffer inventory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell</td>
<td>1 Card</td>
<td>2 Cards</td>
</tr>
<tr>
<td>1</td>
<td>0.486</td>
<td>1.041</td>
</tr>
<tr>
<td>2</td>
<td>0.486</td>
<td>1.040</td>
</tr>
<tr>
<td>3</td>
<td>0.486</td>
<td>1.047</td>
</tr>
<tr>
<td>4</td>
<td>0.486</td>
<td>1.056</td>
</tr>
<tr>
<td>5</td>
<td>0.486</td>
<td>1.073</td>
</tr>
</tbody>
</table>

The input inventory is fairly constant, while the output inventory decreases as the cell position increases.
Second case: fault free versus failure prone systems

- Cells can fail independently;
- Failure rate is 0.02;
- Repair rate is 0.4.

Model of a Kanban cell that can fail

In a perfectly balanced Kanban system the cell performance is position-dependent.
Stochastic Petri net techniques are attractive because they provide a performance evaluation approach based on a formal description.

This allows the use of the same language for the

- specification
- validation
- performance evaluation
- implementation
- documentation

of a system.
Two are the main directions of the research being presently conducted in the field of GSPN-based performance evaluation.

1. Extensions of the GSPN analysis approach to environments in which tokens possess an identity has already been proposed by several authors, and more work is being performed to obtain an environment with a high descriptive power in which the model specification is simple.

2. Various approaches are being pursued for the reduction of the complexity of the solution computation with stochastic techniques, possibly producing only partial or approximate results.

Successes in these two fields would make GSPN a prominent modeling technique in the whole area of distributed systems.