Coloured Petri Nets

Modelling and Validation of Concurrent Systems

Chapter 4: Formal Definition of CP-nets

Kurt Jensen &
Lars Michael Kristensen

{kjensen,lmkristensen}@daimi.au.dk

Syntax

$$CPN = (P, T, A, \Sigma, V, C, G, E, I)$$

Semantics

$$\sum_{MS\,(t,b)\in Y}^{++} E(p,t)\langle b\rangle \ll= M(p) \text{ for all } p \in P$$

Why do we need a formal definition?

- The formal definition is unambiguous.
- It provides a more precise and complete description than an informal explanation.

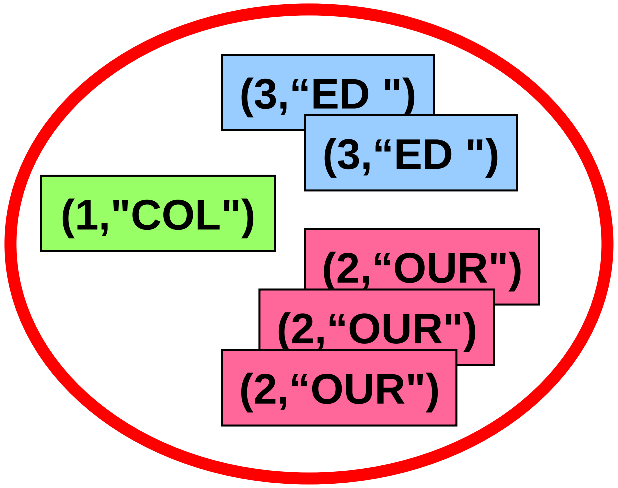
- Users who are satisfied with the informal explanation can skip the formal definition.
- Only few programmers know the formal definition of the programming language they are using.

- We define:
  - Multi-sets.
  - Syntax of Coloured Petri Nets.
  - Semantics of Coloured Petri Nets.

Multi-set

- Similar to a set but with multiple occurrences of elements.

m



Elements in multi-set

Non-negative integers

- **Function NOxDATA $\rightarrow \mathbb{N}$:**

$$m(s) = \begin{cases} 1 & \text{if } s = (1,\text{"COL"}) \\ 3 & \text{if } s = (2,\text{"OUR"}) \\ 2 & \text{if } s = (3,\text{"ED "}) \\ 0 & \text{otherwise} \end{cases}$$

- **Sum:** m = 1‘(1,"COL") ++ 3‘(2,"OUR") ++ 2‘(3,"ED ")

Number of appearances (coefficient)

Elements (from NOxDATA)

Formal definition of multi-sets

- Let $S = \{s_1, s_2, s_3, \ldots\}$ be a non-empty set.

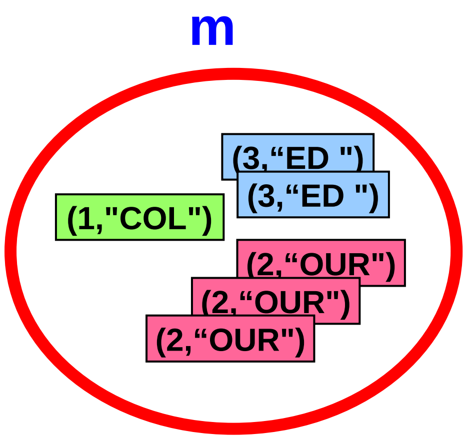
- A multi-set over S is a function $m : S \rightarrow \mathbb{N}$ mapping each element $s \in S$ into a non-negative integer $m(s) \in \mathbb{N}$ called the number of appearances (or coefficient) of s in m.

- A multi-set m is also written as a sum:

$$\sum_{s \in S}^{++} m(s)`s = m(s_1)`s_1 {++}\ m(s_2)`s_2 {++}\ m(s_3)`s_3 {++}\ m(s_4)`s_4 {++} \ldots$$

- Notation:
  - S_{MS} is the set of all multi-sets over S.
  - $\emptyset_{MS}$ is the empty multi-set (polymorphic).

Membership of multi-set



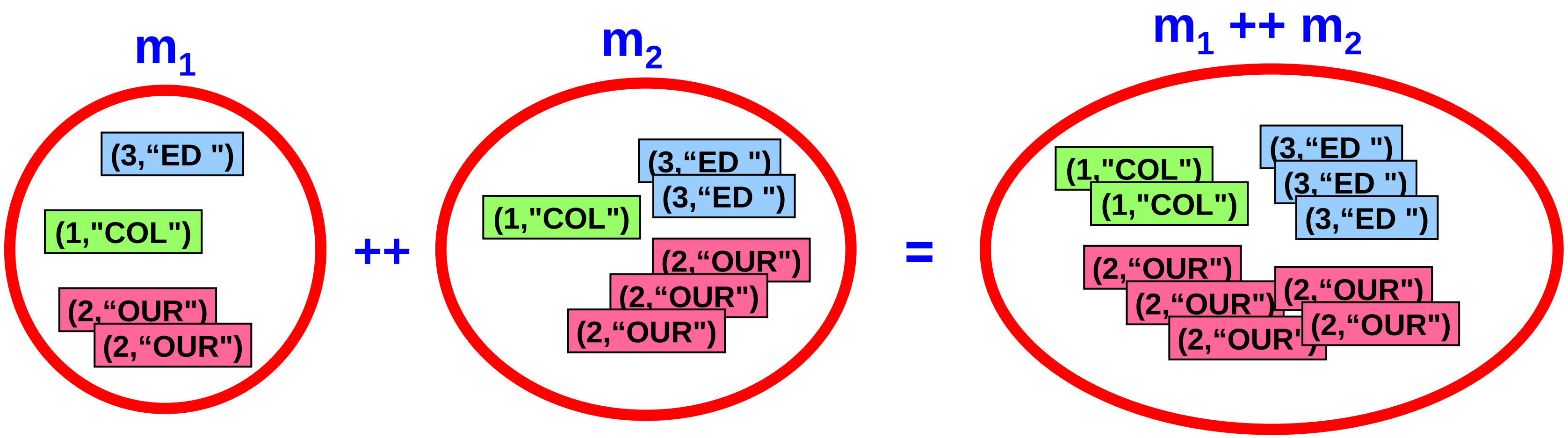
- (1,”COL”), (2,”OUR”) and (3,”ED ”) are members of the multi-set m.
- (4,”PET”) and (17,”CPN”) are not members.

$$\forall s \in S: s \in m \Leftrightarrow m(s) > 0.$$

Membership of multi-set

Comparison of integers

Addition of multi-sets

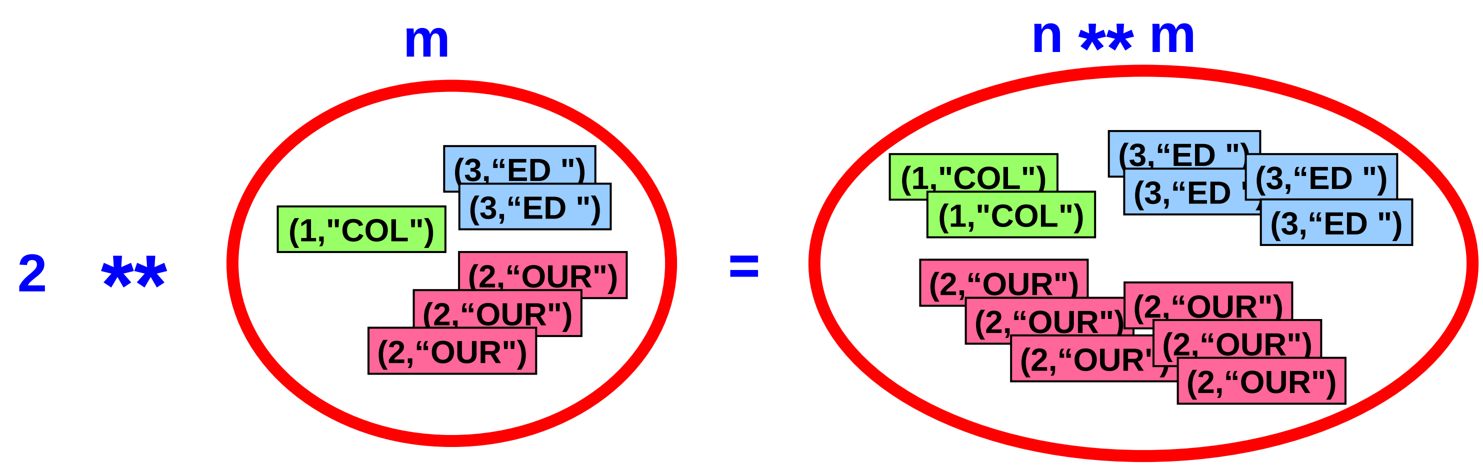


$$\forall s \in S: (m_1 \mathbin{++} m_2)(s) = m_1(s) + m_2(s).$$

Addition of multi-sets

Addition of integers

Scalar multiplication of multi-sets

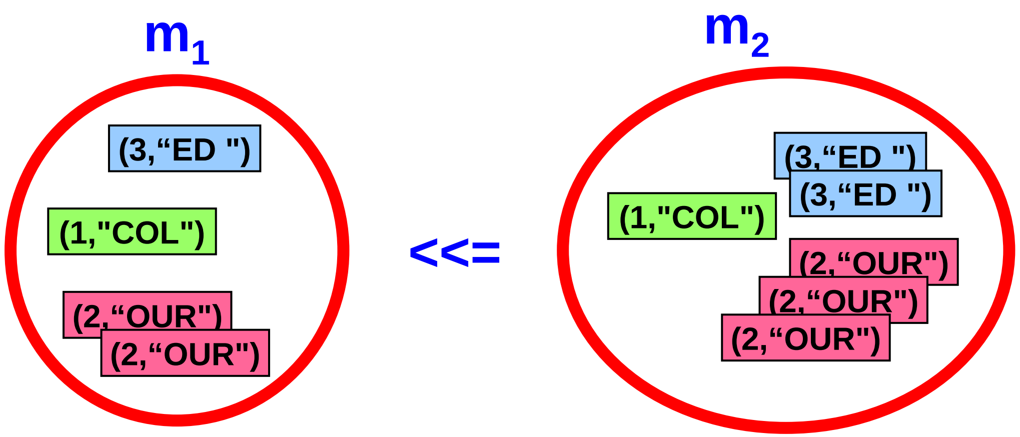


$$\forall s \in S: (n ** m)(s) = n * m(s).$$

Scalar multiplication of multi-set

Multiplication of integers

Comparison of multi-sets

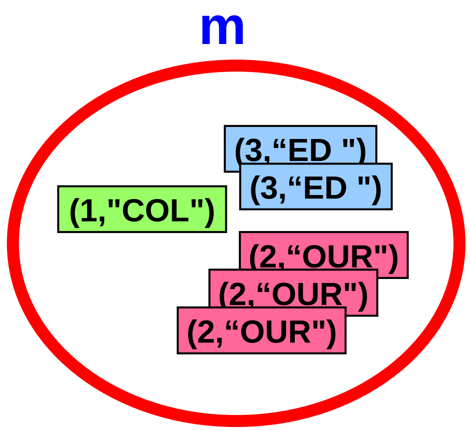


$$m_1 \ll= m_2 \quad \Leftrightarrow \quad \forall s \in S: m_1(s) \leq m_2(s).$$

Smaller than or equal for multi-sets

Smaller than or equal for integers

Size of multi-set



- This multi-set contains six elements.

$$|m| = \sum_{s \in S} m(s).$$

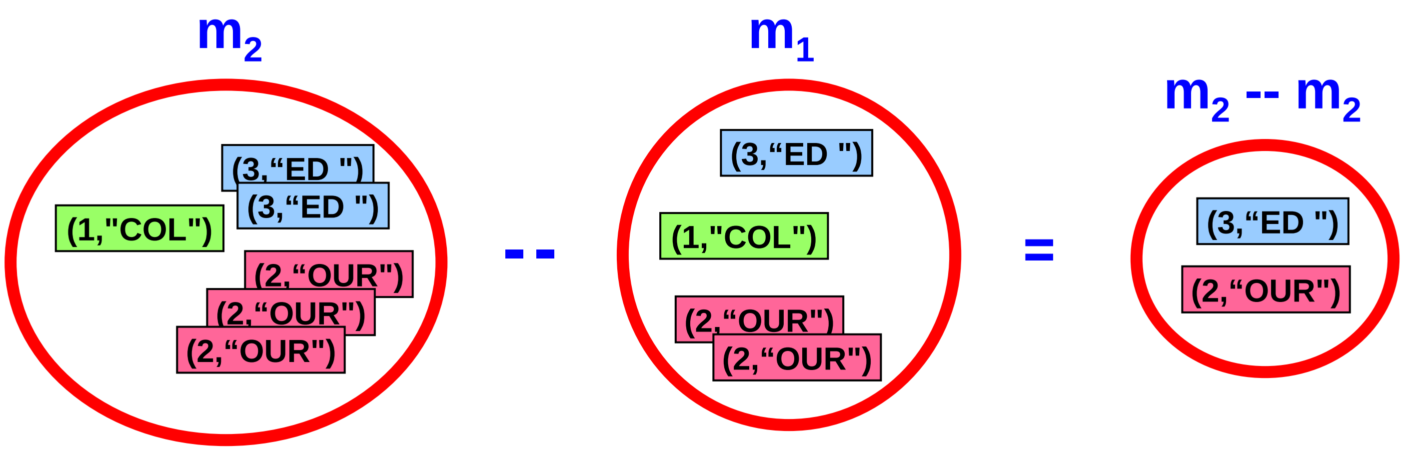
Size of multi-set

Summation of integers

When $|m| = \infty$ we say that m is infinite.

Subtraction of multi-sets

- When m_1 <<= m_2 we also define subtraction:



$$\forall s \in S: (m_2 -- m_1)(s) = m_2(s) - m_1(s).$$

Subtraction of multi-sets

Subtraction of integers

Formal definition of Coloured Petri Nets

A Coloured Petri Net is a nine-tuple CPN = (P, T, A, Σ, V, C, G, E, I).

- P set of places.
- T set of transitions.
- A set of arcs.

Net structure

- Σ set of colour sets.
- V set of variables.

Types and variables

- C colour set function (assigns colour sets to places).
- G guard function (assigns guards to transitions).
- E arc expression function (assigns arc expressions to arcs).
- I initialisation function (assigns initial markings to places).

Net inscriptions

Places and transitions

- A finite set of places P.

  P = { PacketsToSend, A, B, DataReceived, NextRec, C,D, NextSend }.

- A finite set of transitions T.
- We demand that $P \cap T = \emptyset$.

A node is either a place or a transition – it cannot be both

T = { SendPacket,TransmitPacket, ReceivePacket,TransmitAck, ReceiveAck }.

Arcs

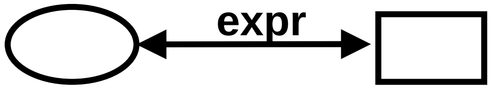
- A set of directed arcs A.
- We demand that $A \subseteq P \times T \cup T \times P$.

Each arc starts in a place and ends in a transition – or it starts in a transition and ends in a place

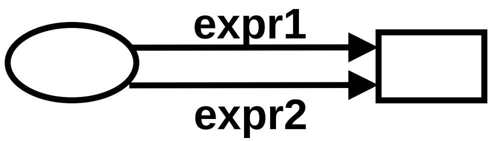
A = { (PacketsToSend, SendPacket), (SendPacket, PacketsToSend),
(SendPacket, A), (A,TransmitPacket), (TransmitPacket, B),
(B, ReceivePacket), (NextRec, ReceivePacket), (ReceivePacket, NextRec),
(DataReceived, ReceivePacket), (ReceivePacket, DataReceived),
(ReceivePacket, C), (C, TransmitAck), (TransmitAck, D), (D, ReceiveAck),
(ReceiveAck, NextSend), (NextSend, ReceiveAck),
(NextSend, SendPacket), (SendPacket, NextSend) }.

Arcs

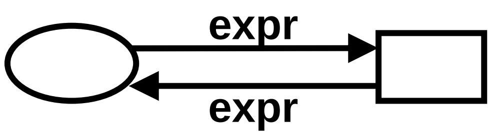
- In the formal definition we do <u>not</u> have:
  - double-headed arcs:



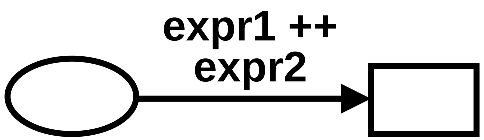
  - parallel arcs:



- CPN Tools allow these and consider them to be short-hands for:
  - two oppositely directed arcs with the same arc expression:



  - addition of the two arc expressions:



Colour sets and variables

- A finite set of non-empty colour sets Σ.

  Σ = { NO, DATA, NOxDATA, BOOL }.

- A finite set of typed variables V.

  We demand that Type[v]$\in\Sigma$ for all v$\in$V.

  V = { n : NO, k : NO, d : DATA, data : DATA, success : BOOL }.

Type of variable must be one of those that is defined in Σ

Colour sets for places

- A colour set function $C : P \rightarrow \Sigma$.

  Assigns a colour set to each place.

$$
C(p) = \begin{cases} \text{NO} & \text{if } p \in \{\text{NextSend, NextRec, C, D}\} \\ \text{DATA} & \text{if } p = \text{DataReceived} \\ \text{NOxDATA} & \text{if } p \in \{\text{PacketsToSend, A, B}\} \end{cases}
$$

Guard expressions

All variables must belong to V

- A guard function $G : T \to EXPR_V$.

  Assigns a guard to each transition.

  We demand that $Type[G(t)] = Bool$ for all $t \in T$.

  G(t) = true for all $t \in T$.

The guard expression must evaluate to a boolean

- In the formal definition we demand all transitions to have a guard.
- CPN Tools consider a missing guard to be a short-hand for the guard expression true which always evaluates to true.
- Hence we have omitted all guards in the protocol example.

Coloured Petri Nets Group
Department of Computer Science

Guard expressions

- CPN Tools consider a list of Boolean expressions:

  $[expr_1, expr_2, \ldots, expr_n]$

  to be a short-hand for:

  $expr_1 \wedge expr_2 \wedge \ldots \wedge expr_n$

- We recommend to write all guards as a list even if they only have a single Boolean expression:

  [expr]

- In this way it is easy to distinguish guards from other kinds of net inscriptions.

Arc expressions

- An arc expression function $E : A \rightarrow EXPR_V$

  Assigns an arc expression to each arc.

  We demand that $Type[E(a)] = C(p)_{MS}$ for all $a \in A$, where p is the place connected to the arc a.

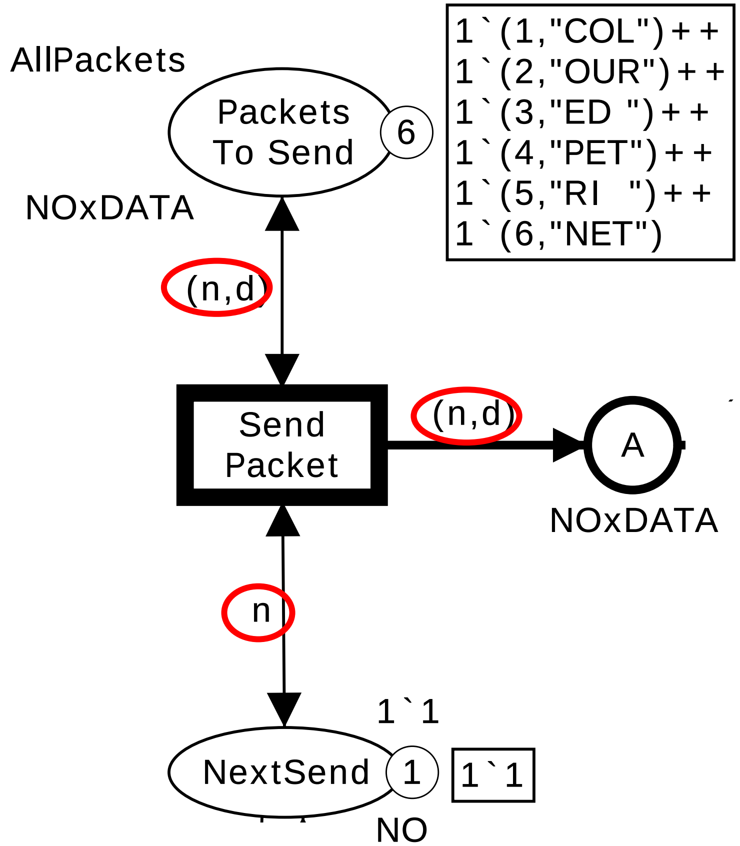
All variables must belong to V

Arc expression must evaluate to a multi-set of tokens belonging to the colour set of the connected place

$$\mathbf{E(a)} = \begin{cases} 1`(n,d) & \text{if } a \in \{ (PacketsToSend,\ SendPacket), \ldots \} \\ 1`n & \text{if } a \in \{ (C,\ TransmitAck), (D,\ ReceiveAck) \ldots \} \\ 1`data & \text{if } a = (DataReceived,\ ReceivePacket) \} \\ \ldots\ldots\ldots & \end{cases}$$

Arc expressions

- In the formal definition we demand all arc expressions to evaluate to multi-sets.
- CPN Tools consider an arc expression expr which evaluates to a single value to be a short-hand for 1`expr.
- Hence we can write n and (n,d) instead of 1`n and 1`(n,d).



Initialisation expressions

- An initialisation function $I : P \to EXPR_{\emptyset}$.
  Assigns an initial marking to each place.
  We demand that
  - $Type[I(p)] = C(p)_{MS}$ for all $p \in P$.

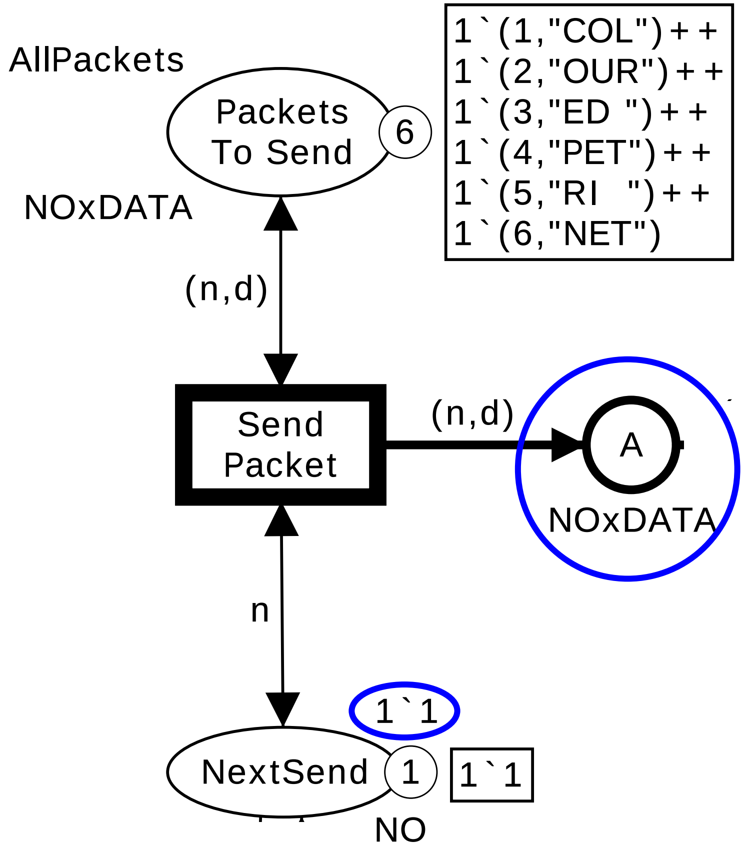
Initialisation expression is not allowed to contain any variables

Initialisation expression must evaluate to a multi-set of tokens belonging to the colour set of the place

$$
\mathbf{I(p)} = \begin{cases} \text{AllPackets} & \text{if } p = \text{PacketsToSend} \\ 1`1 & \text{if } p \in \{ \text{NextSend, NextRec} \} \\ 1`\text{""} & \text{if } p = \{ \text{DataReceived} \} \\ \emptyset_{MS} & \text{otherwise} \end{cases}
$$

Initialisation expressions

- In the formal definition we demand all places to have an initialisation expression and that these evaluate to multi-sets.
- CPN Tools consider a missing initialisation expression to be a short-hand for $\emptyset_{MS}$.
- Hence we are allowed to omit the initialisation expression for place A.
- CPN Tools consider an initialisation expression expr which evaluates to a single value to be a short-hand for 1`expr.
- Hence we could have written 1 instead of 1`1.



Questions about CPN syntax

A. Can a node be both a place and a transition?

B. Can we have an infinite number of places?

C. Can we have an arc from a place to another place?

D. Can a transition have two guards?

E. Can a guard evaluate to an integer?

F. Can an arc expression evaluate to a multi-set of booleans?

G. Can we have a variable in an initial marking expression?

H. Can an arc expression always evaluate to empty?

Find those where the answer is YES?

Markings

- A marking is a function M mapping each place p into a multi-set of tokens $M(p) \in C(p)_{MS}$.

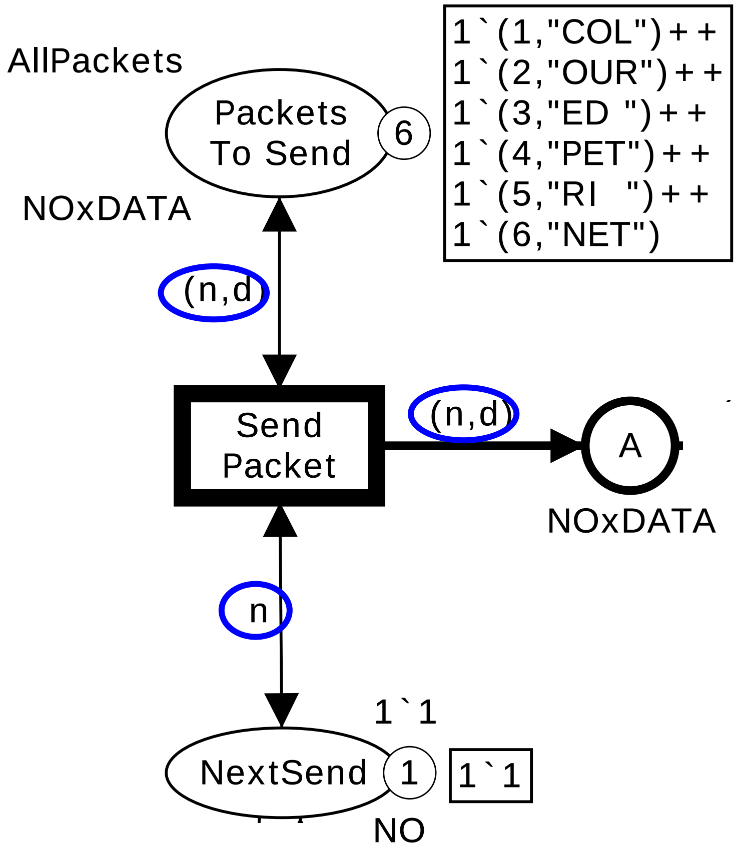
All token values must belong to the colour set of the place

- The initial marking M_0 is defined by $M_0(p) = I(p)\langle\rangle$ for all $p \in P$.

Initialisation expression has no variables. Hence it is evaluated in the empty binding

Variables of a transition

- The variables of a transition are those that appear in the guard or in an arc expression of an arc connected to the transition.
- The set of variables is denoted $Var(t) \subseteq V$.
- Var(SendPacket) = {n,d}.



Coloured Petri Nets Group
Department of Computer Science
Kurt Jensen and Lars M. Kristensen

Bindings and binding elements

- A binding of a transition t is a function b mapping each variable $v \in Var(t)$ into a value $b(v) \in Type[v]$.
- Bindings are written in: brackets: <n=1,d="COL">.
- The set of all bindings for a transition t is denoted B(t).

Each variable must be bound to a value in its type

Transition **Binding for t**

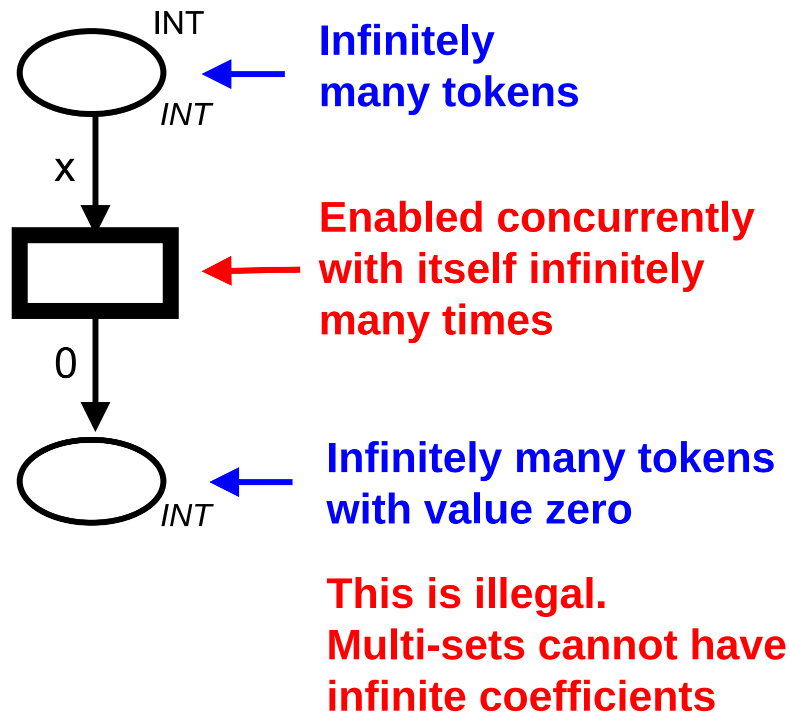
- A binding element is a pair (t,b) such that t is a transition and $b \in B(t)$.
- The set of all binding elements of a transition t is denoted BE(t).
- The set of all binding elements in CPN is denoted BE.

Steps

- A step $Y \in BE_{MS}$ is a non-empty and finite multi-set of binding elements.

- Why forbid empty steps?
  - We would have steps with no effect.
  - It would be impossible to reach a dead marking, i.e., a marking without enabled steps.

- Why forbid infinite steps?
  - We would be able to produce markings which are not multi-sets.



Evaluation of guards and arc expressions

- The rules for enabling and occurrence are based on evaluation of guards and arc expressions.

G(t)<b>	**Evaluation of the guard expression for t in the binding b**
E(a)<b>	**Evaluation of the arc expression for a in the binding b**
E(p,t)<b>	**Evaluation of the arc expression on the arc from p to t in the binding b. If no such arc exists E(p,t) = $\emptyset_{MS}$**
E(t,p)<b>	**Evaluation of the arc expression on the arc from t to p in the binding b. If no such arc exists E(t,p) = $\emptyset_{MS}$**

Enabling of single binding element

- A binding element $(t,b) \in BE$ is enabled in a marking M if and only if the following two properties are satisfied:

  - $G(t)\langle b \rangle = true$.

    Guard must evaluate to true

  - $E(p,t)\langle b \rangle \ll= M(p)$ for all $p \in P$.

    Smaller than or equal for multi-sets

    The tokens demanded by the input arc expressions must be present in the marking M

Occurrence of single binding element

- When the binding element $(t,b) \in BE$ is enabled in a marking M, it may occur leading to a new marking M' defined by:

  - $M'(p) = (M(p) -- E(p,t)\langle b\rangle) ++ E(t,p)\langle b\rangle$ for all $p \in P$.

New marking

Old marking

Tokens consumed by input arcs

Tokens produced by output arcs

Enabling of step

- A step $Y \subseteq BE_{MS}$ is enabled in a marking M if and only if the following two properties are satisfied:

  - $G(t)\langle b\rangle = true$ for all $(t,b) \in Y$.

    All guards must evaluate to true

  - $$\sum_{MS\,(t,b)\in Y}^{++} E(p,t)\langle b\rangle \ll= M(p) \quad \text{for all } p \in P.$$

    The tokens demanded by the input arc expressions must be present in the marking M

Smaller than or equal for multi-sets

Summation over a multi-set Y

Occurrence of step

- When the step $Y \subseteq BE_{MS}$ is enabled in a marking M, it may occur leading to a new marking M' defined by:
  - $M'(p) = (M(p) \;{-\!-}\; {}_{MS}\sum^{++}_{(t,b)\in Y} E(p,t)\langle b\rangle) \;{+\!+}\; {}_{MS}\sum^{++}_{(t,b)\in Y} E(t,p)\langle b\rangle$ for all $p \in P$.

New marking | **Old marking** | **Tokens consumed by input arcs** | **Tokens produced by output arcs**

Notation for occurrence and enabling

$M_1 \xrightarrow{Y} M_2$

Step Y occurs in marking M_1 leading to marking M_2

$M_1 \longrightarrow M_2$

Marking M_2 can be reached from marking M_1 (by the occurrence of an unknown step)

$M_1 \xrightarrow{Y}$

Step Y is enabled in marking M_1 (leading to an unknown marking)

Finite occurrence sequence

$$M_1 \xrightarrow{Y_1} M_2 \xrightarrow{Y_2} M_3 \; \ldots\ldots \; M_n \xrightarrow{Y_n} M_{n+1}$$

- Length $n \geq 0$.
- All markings in the sequence are reachable from M_1.
- An arbitrary marking is reachable from itself by the trivial occurrence sequence of length 0.

Infinite occurrence sequence

$$M_1 \xrightarrow{Y_1} M_2 \xrightarrow{Y_2} M_3 \xrightarrow{Y_3} \ldots\ldots$$

- $\Re(M)$ **The set of markings reachable from M**
- $\Re(M_0)$ **The set of reachable markings**

Diamond property

Y_1 can occur followed by Y_2

$$
\begin{tikzcd}
& M'' \arrow[dr, "Y_2"] & \\
M \arrow[ur, "Y_1"] \arrow[rr, "Y"] \arrow[dr, "Y_2"'] & & M' \\
& M''' \arrow[ur, "Y_1"'] &
\end{tikzcd}
$$

Y_2 can occur followed by Y_1

Y can be divided into two substeps:

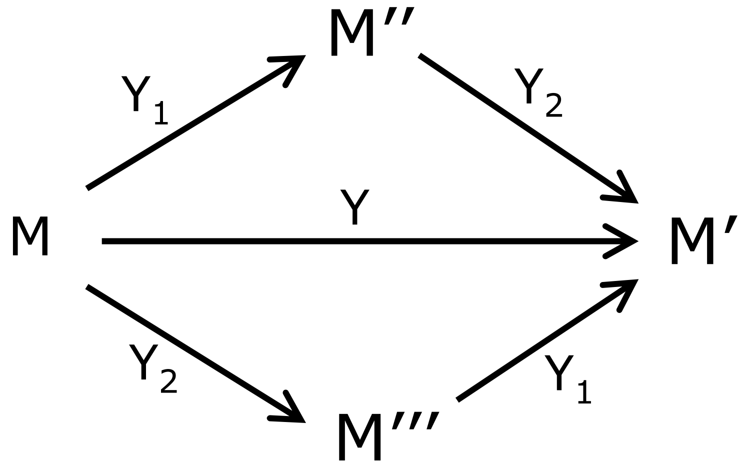
$$Y = Y_1 ++ Y_2$$

In all three cases we reach the same marking

- This is called the diamond property.
- It can be proved from the definition of enabling and occurrence.
- It plays an important role in Petri net theory.

Diamond property

- The diamond property follows from the fact that the effect of a step is independent of the marking in which it occurs.
- The diamond property is <u>not</u> satisfied by ordinary programming languages.



x := x+1; x := 0;	x := 0; x := x+1;

- Repeated use of diamond property:
  - When a step Y is enabled in a marking M, the binding elements of Y can occur one by one in any order.
  - The order has no influence on the total effect.

Questions about CPN semantics

A. Can a transition change the marking of places that are neither input nor output places?

B. Can a transition occur concurrently with itself?

C. Can a binding element occur concurrently with itself?

D. Can two transitions that “reads” the value of the same token occur concurrently?

E. Can a marking be reachable from itself?

F. Can we have more than one initial marking?

G. Can we have an infinite number of reachable markings?

Find those where the answer is YES?

Coloured Petri Nets Group
Department of Computer Science
Kurt Jensen and Lars M. Kristensen