RdPHNs: Outil CPN-tool

GLSD M2

2016-2017

(2)

Réseaux de Petri Colorés

CPN-tool (1): Introduction

Spécification en CPN-tool

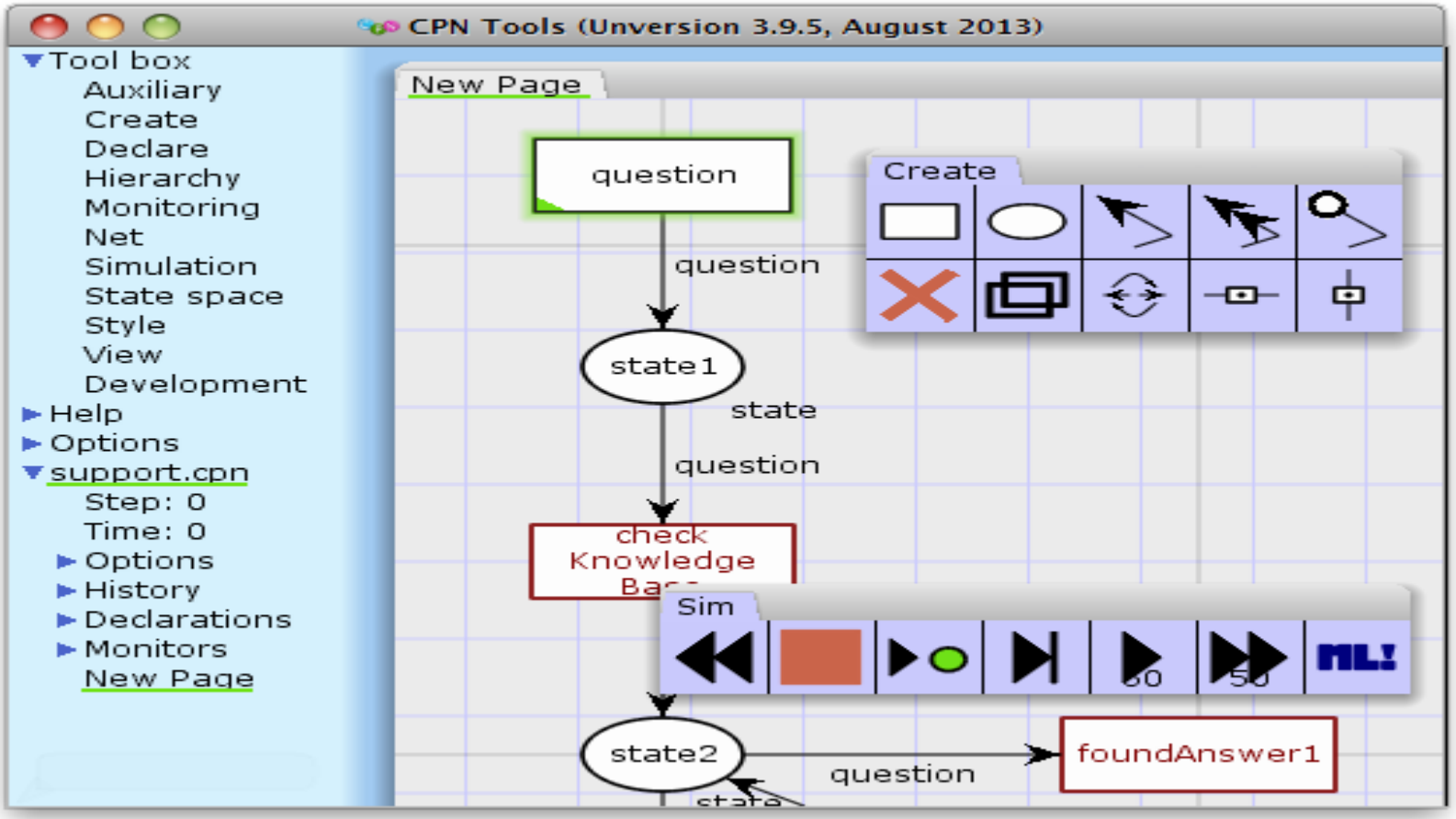
- Une spécification=

1) Le **modèle graphique**: sur une ou plusieurs **pages** (possibilité de hiérarchisation)
2) La déclaration des **types**
3) La déclaration des **variables**, **constantes**
4) La définition des **fonctions**: expressions associées aux arcs

… Il faut avoir des connaissances en **ML**

Une première vision

(last vesion 4.0.0 2013)



http://cpntools.org/

Traitements possible avec CPN-tool

- Analyse et contrôle syntaxique du RdPC
- Simulation du modèle
- Calcul et traçage de l'espace d'état du modèle;
- Écriture des algorithme d'analyse directement ou dans des fichiers séparés

CPN-tool (2)
Spécification du modèle

Des types prédéfinis

- Entier: **INT**
- Booléen: **BOOL={true, false}**
- Chaine de caractère: **STRING**

Déclaration de variables et de constantes

- Variables:

var nom_var:type;

Exemple:

var x:INT;

- Constantes:

val nom_cont=valeur;

Exemple:

val x=1;

Opérations sur les booléens

- **not** b: négation du booléen b
- b1 **andalso** b2: conjonction
- b1 **orelse** b2: disjunction (i.e., inclusive or)

Opérations sur les entiers

- **~**i: négation de l'entier i
- i1 **+** i2: addition
- i1 **-** i2: soustraction
- i1 * i2: multiplication
- i1 **div** i2: division, quotient
- i1 **mod** i2: reste de la division entière
- **abs** i: valeur absolue de i
- **Int.min**(i1,i2): minimum entre i1 et i2
- **Int.max**(i1,i2): maximum entre i1 et i2

Opérations sur les STRING

- s1**^**s2: concaténer s1 and s2
- **String.size** s: nombre de caractères de s
- **substring** (s,i,len): extraire une sous chaine de longueur len à partir de la position i de s, la position initiale est égale à 0
- **explode** s: convertir s en une liste de caractères
- **implode** l: convertire une liste l de caractère en une STRING

Déclaration d'ensemble de couleurs (color sets)

Declarations

- **Color sets**
  - Alias color sets
  - Boolean color set
  - Color set functions
  - Declare clause
  - Enumeration color set
  - Definition of list functions
  - Index color sets
  - Integer color sets
  - Large Integer color sets
  - List color sets
  - Product color sets
  - Real color sets
  - Record color sets
  - Size and complexity of color sets
  - String color sets
  - Subset color sets
  - Time color sets
  - Union color sets
  - Unit color set

Toujours commencer par le mot clé

coloset

Déclaration d'ensemble de couleurs (color sets): **Alias**

Alias (ou re-nomage de type prédéfini):

1. On peut utiliser **l'égalité** directe entre nouveau type et type prédifini;

Exemple**:**

colset WholeNumber = INT;

Déclaration d'ensemble de couleurs (color sets): **Alias**

2. Pour créer des alias, on peut utiliser le mot clé **with** afin de rassembler (**énumérer** des valeurs) des valeurs:

Exemple:

colset Answer = **bool with** (no, yes);

colset binaire= **int with** 0 .. 1;

Ou bien de manière général:

colset name = **with** id0 | id1 | … | idn;

Déclaration d'ensemble de couleurs (color sets): **Alias**

Exemple:

colset Day = **with** Mon | Tues | Wed | Thurs | Fri | Sat | Sun ;

Déclaration d'ensemble de couleurs (color sets): **Indexed CS**

- L'objectif est de créer un type indexé:

colset name = **index** id **with** int-exp1..int-exp2;

int-exp1..int-exp2 sont des **expressions entières**.

Exemple:

Colset PH= **index** ph **with** 1..5;

Colset Bag= **index** bg **with** 1..5;

Déclaration d'ensemble de couleurs (color sets): **List CS**

- Pour créer des listes de valeurs (entières):

colset nom= **list** type_base **with** val1..val2;

type_base est souvent **INT**

- Exemple:

colset list_int= **list** INT **with** 1..5;

Opérations sur les listes (1)

- **nil:** empty list (same as [])
- e**::**l: prepend element e as head of list l
- **hd** l: head, the first element of the list l
- **tl** l: tail, list with exception of first element
- **length** l: length of list l
- **rev** l: reverse list l
- **map** f l: use function f on each element in list l and returns a list with all the results

Opérations sur les listes (2)

- **app** f l: use function f on each element in list l and returns ()
- **foldr** f z l: returns f(e1, f(e2, …,f(en, z) …)) where l = [e1, e2,…, en]
- **foldl** f z l: returns f(en, …,f(e2,f(e1, z)) …) where l = [e1, e2,…, en]
- **List.nth**(l,n): nth element in list l, where 0 ⇐ n < length l

Opérations sur les listes (3)

- **List.take**(l, n): returns first n elements of list l
- **List.drop**(l, n): returns what is left after dropping first n elements of list l
- **List.exists** p l: returns true if p is true for some element in list l
- **List.null** l: returns true if list l is empty

Déclaration d'ensemble de couleurs (color sets): **Product CS**

- Pour créer des produit cartésiens (types complexes) :

colset type= **product** type1 * type2 * … * type_n;

Exemple:

colset couple_entier=**product** INT*INT;

Opérations sur les Produits

- **#**i x:

  extraire l’élément d’ordre i dans le tuple x

Déclaration d'ensemble de couleurs (color sets): **real CS**

- **Pour créer une intervalle de valeurs réelles**

colset name = **real with** real-exp1..real-exp2;

Exemple:

colset SomeReal = **real** with 1.0..3.5;

Opérations sur les réels (1) (color sets): **real CS**

- **~**r: negation of the integer value r
- r1 **+** r2: addition
- r1 **-** r2: subtraction
- r1 * r2: multiplication
- r1 **/** r2: division
- **Real.==**(r1,r2): returns true if and only if neither r1 nor r2 is NaN, and r1 and r2 are equal

Opérations sur les réels (2) (color sets): **real CS**

- **abs** r: absolute value of r
- **floor** r: return largest integer not larger than r
- **ceil** r: return smallest integer not less than r
- **trunc** r: round r towards zero
- **round** r: return integer nearest to r

Opérations sur les réels (3) (color sets): **real CS**

- **real** i: convert integer i to real value
- **Real.min**(r1,r2): minimum of r1 and r2
- **Real.max**(r1,r2): maximum of r1 and r2
- **Math.sqrt** r: square root of r
- **Math.ln** r: natural logarithm
- **Math.exp** r: exponential
- **Math.sin** r: sine
- **Math.cos** r: cosine
- **Math.tan** r: tangent
- **Math.arctan** r: arc tangent

Déclaration d'ensemble de couleurs (color sets): **record CS**

- **Pour créer des enregistrement:**

colset type = **record** id1: type1 * id2: type2 * … * idn: type_n;

Exemple

colset PACK = **record** se: SITES * re: SITES * no: INT;

Opérations sur les « **records** »

- **#idi** rec: extract the id_i element from the record rec

Déclaration d'ensemble de couleurs (color sets): **subset CS**

- **Créer un sous type d'un type prédifini**

1. Les valeurs du sous type doivent satifsaire une fonction booléenne:

colset name = **subset** name0 **by** subset-function;

Exemple:

fun even i = (i **mod** 2)=0;

colset EvenInt = **subset** INT **by** even;

Déclaration d'ensemble de couleurs (color sets): **subset CS**

2. Les valeurs doivent faire partie d'une liste:

colset name = **subset** name0 **with** subset-list;

Exemple:

colset Weekend = **subset** Day **with** [Sat, Sun];

Autres opérations sur les « **colset** » ??

Opérations (1): applicables à tous les types

Pour un type donné cs, les fonctions suivantes sont applicables:

- **cs.lt**(c1,c2) : c1 < C2
- **cs.legal**(v): vérifier si v est dans cs
- **cs.mkstr**(c): donner une représentation en string de c
- **cs.mkstr_ms**(ms): donner une représentation en string du multi-set ms de cs

Opérations (2): Petit nombre de valeurs

- **cs.all**(): donne un multi-set avec une occurrence de chaque valeur dans cs;
- **cs.ran**(): donne une valeur élatoire de cs;

~~**INT.ran()**~~

- **cs.size**(): la cardinalité de l'ensemble cs;
- **cs.ord(c)**: donne l'ordre de c dans l'ensemble cs. (les positions commencent de 0 à cs.size()-1);
- **cs.col**(i): donne la valeur de position i;

Fonctions,
structures de contrôles,
expressions d’arcs et de transitions

Déclaration de fonctions

```
fun id pat1 = exp1
    | id pat2 = exp2
    | ...
    | id patn = expn;
```

Example:

```
fun listMult (c, x::xs) = (c * x)::listMult(c, xs)
     | listMult (_, nil) = nil
```

Structures de contrôle

- **if** bool-exp **then** exp1 **else** exp2;

Exemple:

if x=p **then** 1`e **else** 2`e

Structures de contrôle (2)

- **case** exp **of**
  pat1 => exp1 |
  pat2 => exp2 |
  ... |
  patn => expn;

- Exemple:

```
case x of
    p=> 1`e |
    q=> 2`e |
   _=> 4`e;
```

Expressions sur des arcs (1)

Des expressions qui doivent évaluer pour un multi-ensemble qui correspond à la place attachée à cet arc:

- **Valeurs**;
- **Variables**: sur des arcs sortants d'une transitions, on peut avoir **des variables libres** à condition qu'elles appartiennent à de **petits ensembles**;
- **fonctions**

Inscriptions des transitions

- Noms
- Gardes
- Inscription de **temps** (à revoir une autre fois)
- Segments de code

Inscriptions des transitions (2)
Gardes:

- Une liste d'expressions booléennes [b-exp1, b-exp2, …, b-expn]

- Les opérateurs suivants sont utilisés:

  = <> <= >= < > andalso orelse

Inscriptions des transitions (2)
Code

- Le segment de code a toujours la forme:

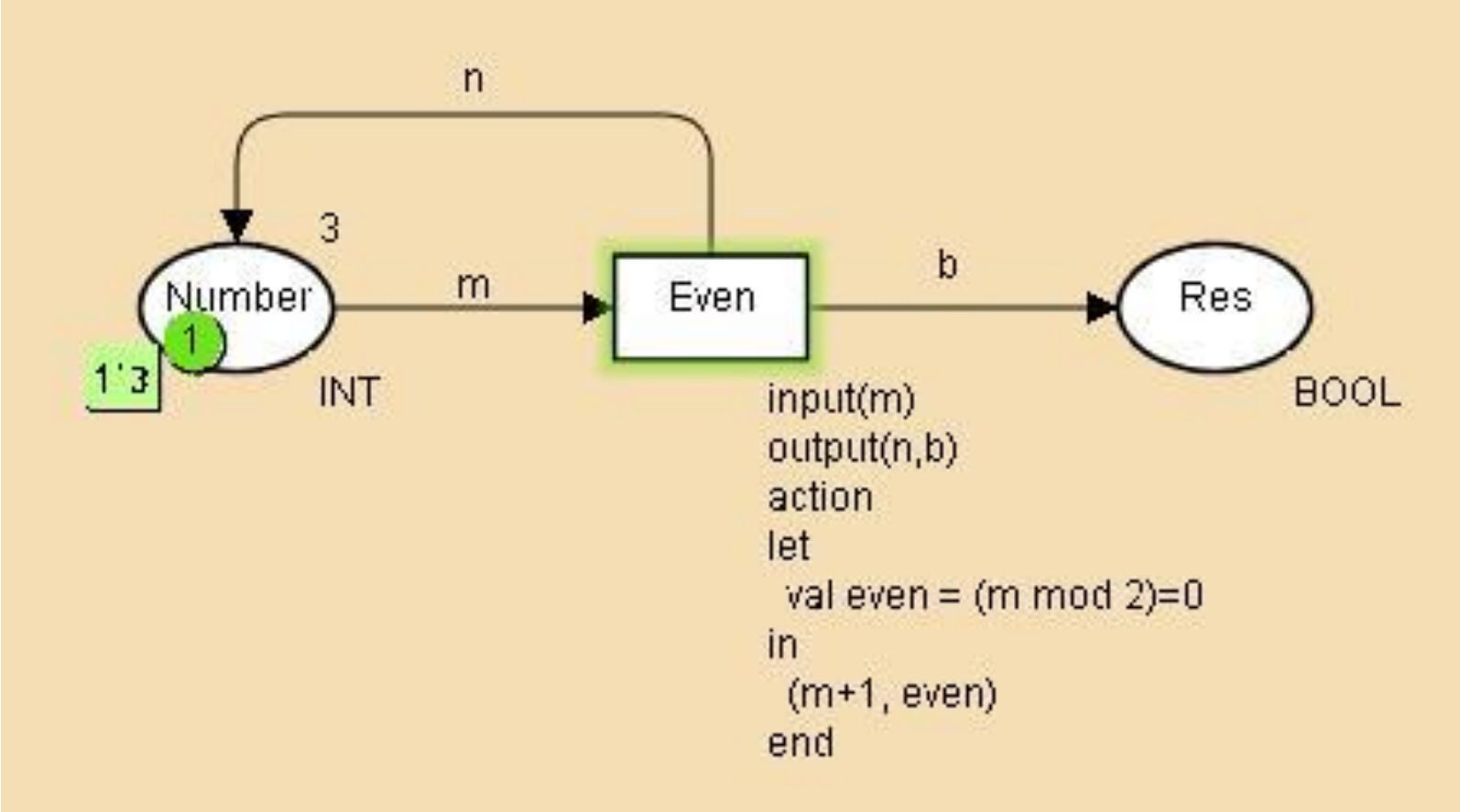
input ();

output ();

action ();

- Input doit contenir les paramètres d'entrée; output: les paramètre de sortie et enfin une action à exécuter.

Inscriptions des transitions (3)



Inscriptions des transitions (4)
Code

- Dans la partie action, on utilise souvent les déclarations suivantes:

```
let val pat1 = exp1
      val pat2 = exp2
      ...
     val patn = expn
 in
exp
end;
```

Opérations sur les multi-ensembles

- i`c: the constructor of a multiset where i is an integer and c is a color

- **empty:** the empty constant constructs an empty multiset that is identical for all kinds of multisets

- ms1 == ms2 : multiset equality
- ms1 <><> ms2 : multiset inequality
- ms1 >> ms2 : multiset greater than
- ms1 >>= ms2 : multiset greater than or equal to
- ms1 << ms2 : multiset less than
- ms1 <⇐ ms2 : multiset less than or equal to

Opérations sur les multi-ensembles (2)

- ms1 ++ ms2 : multiset addition
- ms1 -- ms2 : multiset subtraction (ms2 must be less than or equal to ms1),
  raises Subtract exception if ms2 is not less than or equal to ms1.
- i ** ms : scalar multiplication

Opérations sur les multi-ensembles(3)

Dans les ces fonctions, l'usage des parenthèses "()" est obligatoire:

- **size** ms: size of multiset ms
- **random** ms: returns a pseudo-random color from ms
- **cf**(c,ms): returns the number of appearances of color c in ms
- **filter** p ms: takes a predicate p and a multiset ms and produces the multiset of all the appearances in ms satisfying the predicate

Opérations sur les multi-ensembles(4)

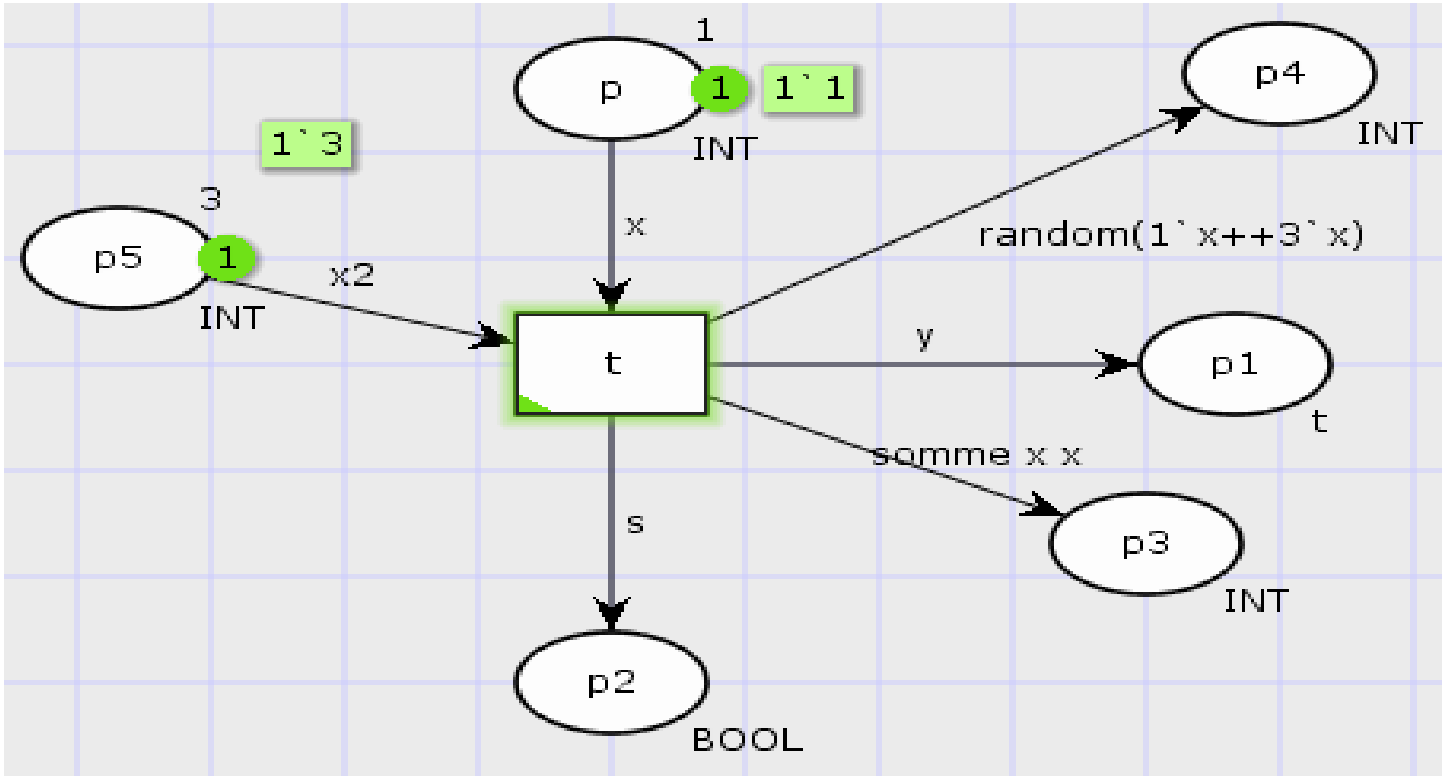
Dans les ces fonctions, l'usage des parenthèses "()" est obligatoire:

- **ext_col** f ms: takes a function f and a multiset c1`s1++c2`s2++…++cn`sn and produces the multiset c1`f(s1)++c2`f(s2)++…++cn`f(sn)

- **ext_ms** f ms: takes a function f and a multiset c1`s1++c2`s2++…++cn`sn and produces the multiset c1*f(s1)++c2*f(s2)++…++cn*f(sn)

- **ms_to_col** ms: converts a multiset of size 1 into the single element in the multiset, raises no_singleton exception if(size ms) is not equal to 1.

Un autre exemple

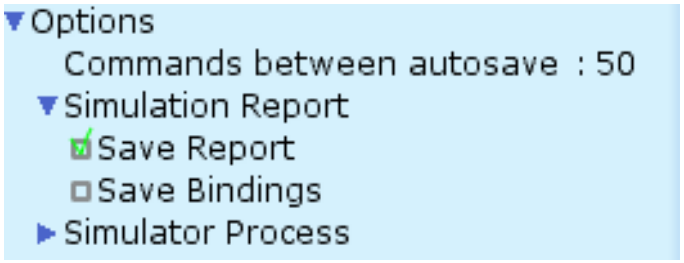


CPN-tool (3)
Simulation

Simulation du modèle (1)

- Il est possible de simuler un modèle de CPN en utilisant la boite à outil de simulaiton

- Il est possible de générer un rapport de simulation final, si on choisit ceci:



Simulation du modèle (2) générer des fichiers

- Il est possible de stocker certaines informations liées au modèle et à son exécution dans des fichiers particuliers

- Ceci peut être fait en associant des actions aux transitions du modèle

- Ces actions doivent créer les fichiers, enregistrer des données dans ces fichiers, enfin fermer ces fichier

Simulation du modèle (3) générer des fichier

- Déclaration d'un fichier de sortie:

globref nom_fichier=**TextIO**.**stdOut**;

- Usage du fichier dans une action:

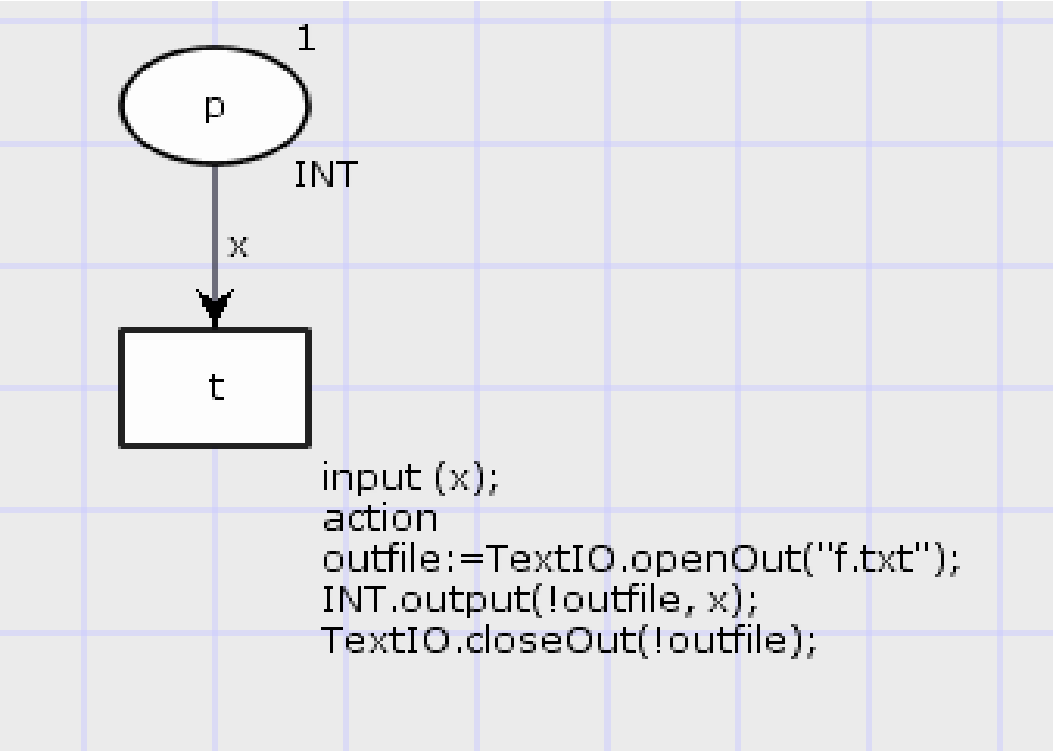
-ouverture:
nom_fichier:=**TextIO**.**openOut**(''nom physique'');

-écriture:
Type_donnée.**output**(!nom_fichier, donnée);

-fermeture:
TextIO.**closeOut**(!nom_fichier);

Simulation du modèle (4) générer des fichier: exemple

- On stocke la valeur de x après l'exécution de t, dans un fichier nomé: f.txt



Simulation du modèle (5) importer des déclarations de fichiers

- Il est possible d'importer des déclarations stockés dans d'autres fichiers;

- On doit utiliser un texte associé au modèle:

use filename;

Exp: **use** "valueDeclarations.sml";

- Lors de la simulation du modèle, on doit commencer par exécuter l'importation avec ML!

Simulation du modèle (6) importer des déclarations de fichiers

- Le fichier f3.sml contient la définition de fact