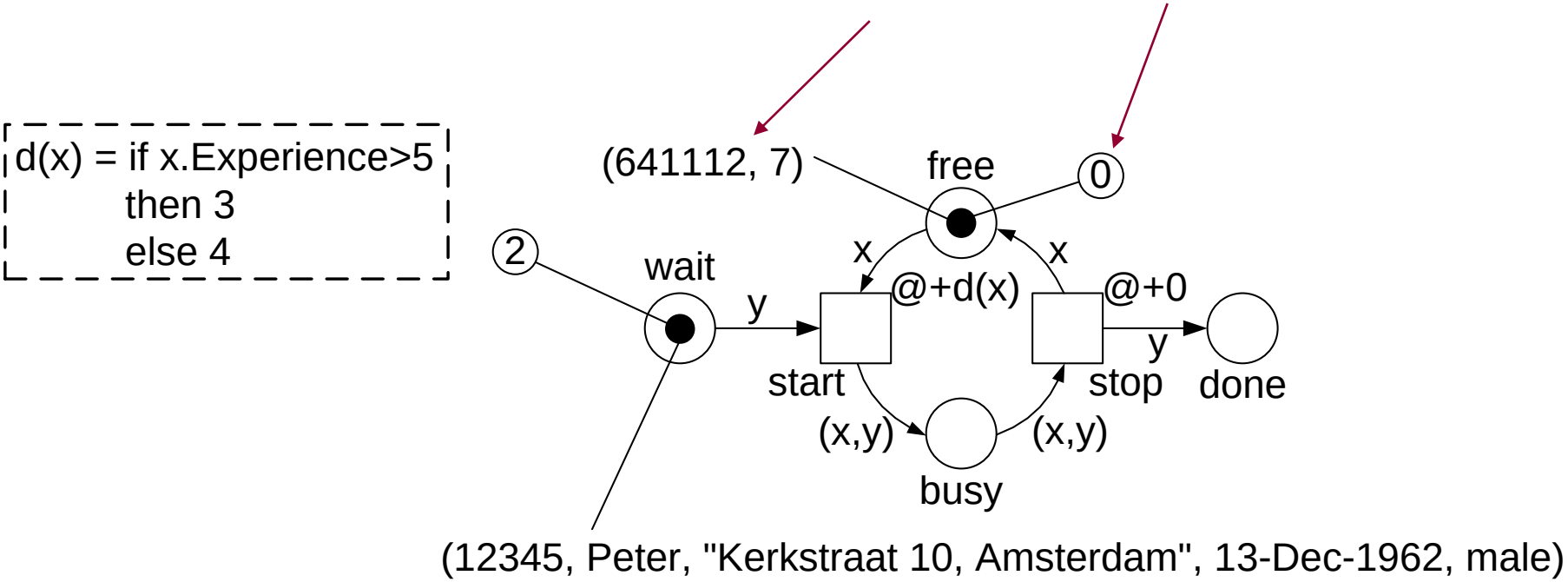
Colored Petri Nets: The Language

Dr. Massimiliano de Leoni
based on slides by prof.dr.ir. Wil van der Aalst

Technische Universiteit Eindhoven University of Technology
Where innovation starts

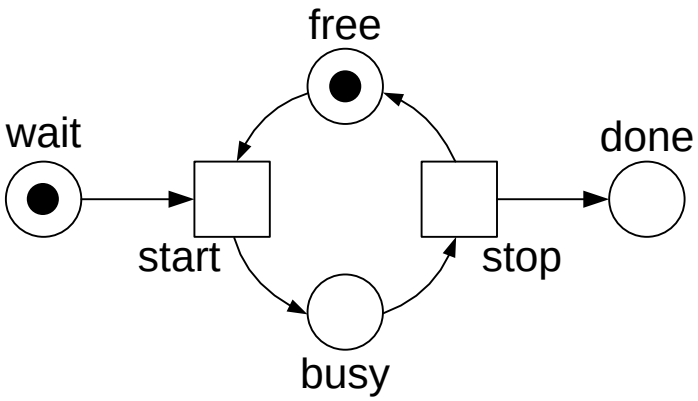
Recall

- Petri nets extended with **color** and **time**



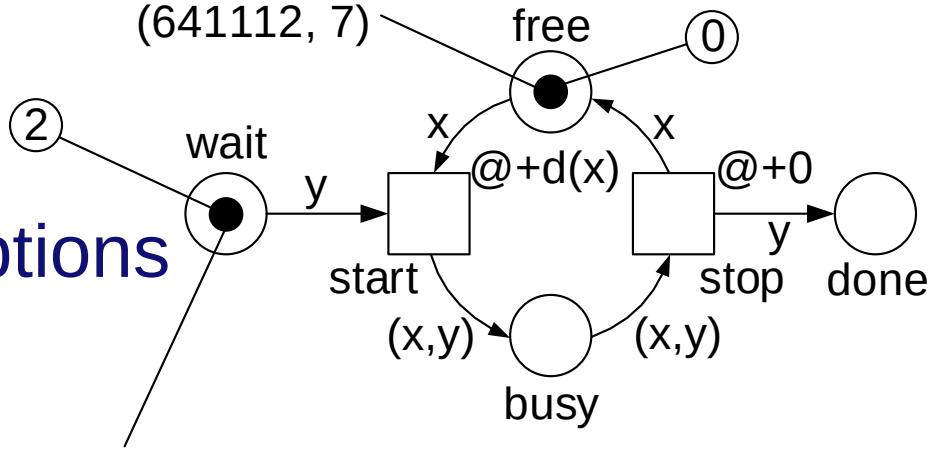
- **consuming/producing multisets** describes behavior

Petri Nets vs Colored Petri Nets



Defined by diagram of its network structure

```
d(x) = if x.Experience>5
       then 3
       else 4
```



Additional descriptions to complete specification

(12345, Peter, "Kerkstraat 10, Amsterdam", 13-Dec-1962, male)

Today: The CPN language – a concrete syntax

- Syntax needed to type places and give values (colors) to tokens
- Adopted from Standard ML (a **functional** language)
- Introduced by Kurt Jensen et al.
- Supported by CPN Tools
- In 2010, the support and further development of CPN Tools moved from Aarhus University (Denmark) to TU/e
- Version 3.0 was the first version released by TU/e
- Version 4.0.1 is used in this course.
- For more information: http://cpntools.org

Three reasons for choosing the CPN language

1. Graphical language to express the network structure and the additional descriptions in terms of a diagram
2. Formal semantics
3. Abundance of analysis methods available, some supported by CPN Tools

Outline

- Types and values
- Defining color sets
- Defining markings
- Defining arc inscriptions
- Defining guards
- Defining functions
- Defining transition priority
- Defining time

Basic types

- unit: type with just one value () (i.e., “black” token)

- Integers (int), e.g., 5 and ~32423 (i.e., -32,423)
- Reals (real), e.g., ~34.34, 7e3 (i.e., 7,000.0), and 4e~2 (i.e., 0.04)
- Strings (string), e.g., "Hello", "28-02-2003".
- Booleans (bool): true and false.

Basic operators (1): Arithmetic

- ~ for the unary minus
- + and – for reals and integers
- * (multiplication) for reals and integers
- / (division) for reals
- div and mod for integers
  (e.g., 28 div 10 = 2, 28 mod 10 = 8)
- =, >, <, >=, <=, <> for comparing things
  - >= (greater than or equal), <= (smaller than or equal), and <> (not equal)).
- ^ for strings (concatenation "AA"^"BB" = "AABB")

Basic operators (2): Logical operators

- not (for negation)
- andalso (for logical AND)
- orelse (for logical OR)
- if … then … else … (choice based on Boolean argument, the then and else part should be of the same type)

- not(1=1) results in false
- (1=1) andalso not(0>1 orelse 2>3) results in false
- if "X"="X" then 3 else 4 results in 3

Exercise: Give type and value of each result

a) if (4>=4) then ("Hello" ^ " " ^ "World") else "X"

b) if true then 20 div 8 else 20 mod 8

c) not(1=1 orelse 1=2)

d) not(1=1 andalso 1=2)

e) if ("Hello" ^ " " ^ "World" = "X") then 20 else 3

Use CPN Tools (“Create aux text” → text field → "Evaluate ML" on text field)

CPN Tools (Version 3.5.7 Koningslied Edition, April 2013)

- Tool box
  - Auxiliary
  - Create
  - Declare
  - Hierarchy
  - Monitoring
  - Net
  - Simulation
  - State space
  - Style
  - View
- Help
- Options
- basic-expressions.cpn
  - Step: 0
  - Time: 0
  - Options
  - History
  - Declarations
    - Standard declarations
    - Race
      - colset Driver = string;
      - colset Lap
      - colset TimeMS
      - colset LapTime
      - colset LapTimes
      - colset DriverResults
      - colset Race
      - val jv = "Jos Verstappen" : Driver;
        val r1jos =(1,31000) : LapTime;
        val r2jos =(2,33400) : LapTime;
        val r3jos =(3,32800) : LapTime;
        val r123jos = ((1,31000)::[(2,33400)])^^[(3,32800)] : LapTimes;
        val jos = {d=jv,r=r123jos}: DriverResults;
        val michael = {d="Michael Schumacher", r=[(1,32200),(2,31600),
        (3,30200),(4,29600)]}:DriverResults;
        val rubens = {d="Rubens Barrichello", r=[(1,34500),(2,32600),(3,3720
        val Monaco = jos :: ([michael]^^[rubens]) : Race;
  - Monitors
  - basic-expr
  - list-expr
  - race

Sim

basic-expr | list-expr | race | Binder 0

val it = false : bool

not(1=1)

val it = true : bool

(1=1) andalso not(0>1 orelse 2>3)

val it = 3 : int

if "X"="X" then 3 else 4

if (4>=4) then ("Hello" ^ " " ^ "World") else "X"

if true then 20 div 8 else 20 mod 8

not(1=1 orelse 1=2)

not(1=1 andalso 1=2)

None

Outline

- Types and values
- Defining color sets
- Defining markings
- Defining arc inscriptions
- Defining guards
- Defining functions
- Defining transition priority
- Defining time

Color set declarations (each place needs a type = color set)

- A color set is a type that is defined using a color set declaration color ... = ... ,[1] e.g.,
  - color I = int;
  - color S = string;
  - color B = bool;
  - color U = unit;
- Once declared, it may be used to type places.
- Newly defined types like I, S, B, and U may be used in other color set declarations.

[1] "color" is shown as "colset" in CPN Tools, but one can type "color"

Overview of basic color sets

- int
- real
- string
- bool
- unit
- with (subtype and enumerations)
- index (id's)

Creating subtypes and enumerations using the "with" clause

Constructing **subsets**:

- color Age = int with 0..130;
- color Temp = int with ~30..40;
- color Alphabet = string with "a".."z";

Define new, simple enumerations:

- color Human = with man | woman | child;
- color ThreeColors = with Green | Red | Yellow;

Index

- color Person = index person with 1..100;
  (values: person(1), person(2), …, person(100))
- color Mark = index m with 1..10
  (values: m(1), m(2), …, m(10))

Creating new types: product and record

- color Coordinates = product I * I * I;
  (1,2,3), (~4,66,0), ...
- color HumanAge = product Human * Age;
  (man,50), (child,3), ...
- color CoordinatesR = record x:I * y:I * z:I;
  {x=1, y=2, z=3}, {x=~4, y=66, z=0}, ...
- color CD = record artists:S * title:S * noftracks:I;
  {artists="Rammstein", title="Reise, Reise", noftracks=11}, ...
  - color I = int;
  - color S = string;
  - color B = bool;
  - color U = unit;
  - color Age = int with 0..130;
  - color Human = with man | woman | child;
  - color ThreeColors = with Green | Red | Yellow;

Creating new types: list constructors

- color Names = list S;
  ["John", "Liza", "Paul"], [ ], ...
- color ListOfColors = list ThreeColors;
  [Green], [Red, Yellow], ...

```
color I = int;
color S = string;
color B = bool;
color U = unit;
color Age = int with 0..130;
color Human = with man | woman | child;
color ThreeColors = with Green | Red | Yellow;
```

Example: Formula 1

- color Driver = string;
- color Lap = int with 1..80;
- color TimeMS = int with 0..10000000;
- color LapTime = product Lap * TimeMS;
- color LapTimes = list LapTime;
- color DriverResults = product Driver * LapTimes;
- color Race = list DriverResults;

Example: Formula 1 (cont.)

A possible value of color set Race is:

[{d="Sebastian Vettel",
  r=[(1,31000),(2,33400),(3,32800)]},

{d="Jenson Button",
  r=[(1,32200),(2,31600),(3,30200),(4,29600)]},

{d="Fernando Alonso",
  r=[(1,34500),(2,32600),(3,37200),(4,42600)]}]

```
color Driver = string;
color Lap = int with 1..80;
color TimeMS = int with 0..10000000;
color LapTime = product Lap * TimeMS;
color LapTimes = list LapTime;
color DriverResults = product Driver * LapTimes;
color Race = list DriverResults;
```

Operations on lists and records

- [ ] denotes the empty list
- ^^ concatenates two lists, e.g., [1,2,3]^^[4,5] evaluates to [1,2,3,4,5]
- :: adds an element in front of a list, e.g., "a"::["b","c"] evaluates to ["a","b","c"]
- # extracts a field of a record, e.g., #x{x=1,y=2} evaluates to 1
- # extracts an element of a product, e.g., #2(man, 50) evaluates to 50

Constants

- It is possible to define constants, e.g.,
  - val sv = "Sebastian Vettel" : Driver;
  - val lap1 = 1 : Lap;
  - val start = 0 : Time;
  - val seven = 7 : int;

```
color Driver = string;
color Lap = int with 1..80;
color TimeMS = int with 0..10000000;
color LapTime = product Lap * TimeMS;
color LapTimes = list LapTime;
color DriverResults = product Driver * LapTimes;
color Race = list DriverResults;
```

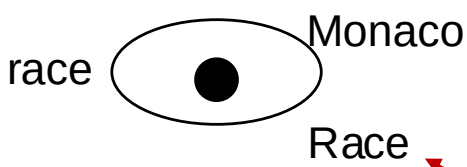
So what?

We can now type and initialize places!

declarations

```
color Driver = string;
color Lap = int with 1..80;
color Time = int with 0..10000000;
color LapTime = product Lap * Time;
color LapTimes = list LapTime;
color DriverResults = record d:Driver * r:Laptimes;
color Race = List DriverResults;
val Monaco = [{d="Sebastian Vettel", r=[(1,31000),(2,33400),(3,32800)]},
{d="Jenson Button", r=[(1,32200),(2,31600),(3,30200),(4,29600)]},
{d="Fernando Alonso", r=[(1,34500),(2,32600),(3,37200),(4,42600)]}];
```

place name



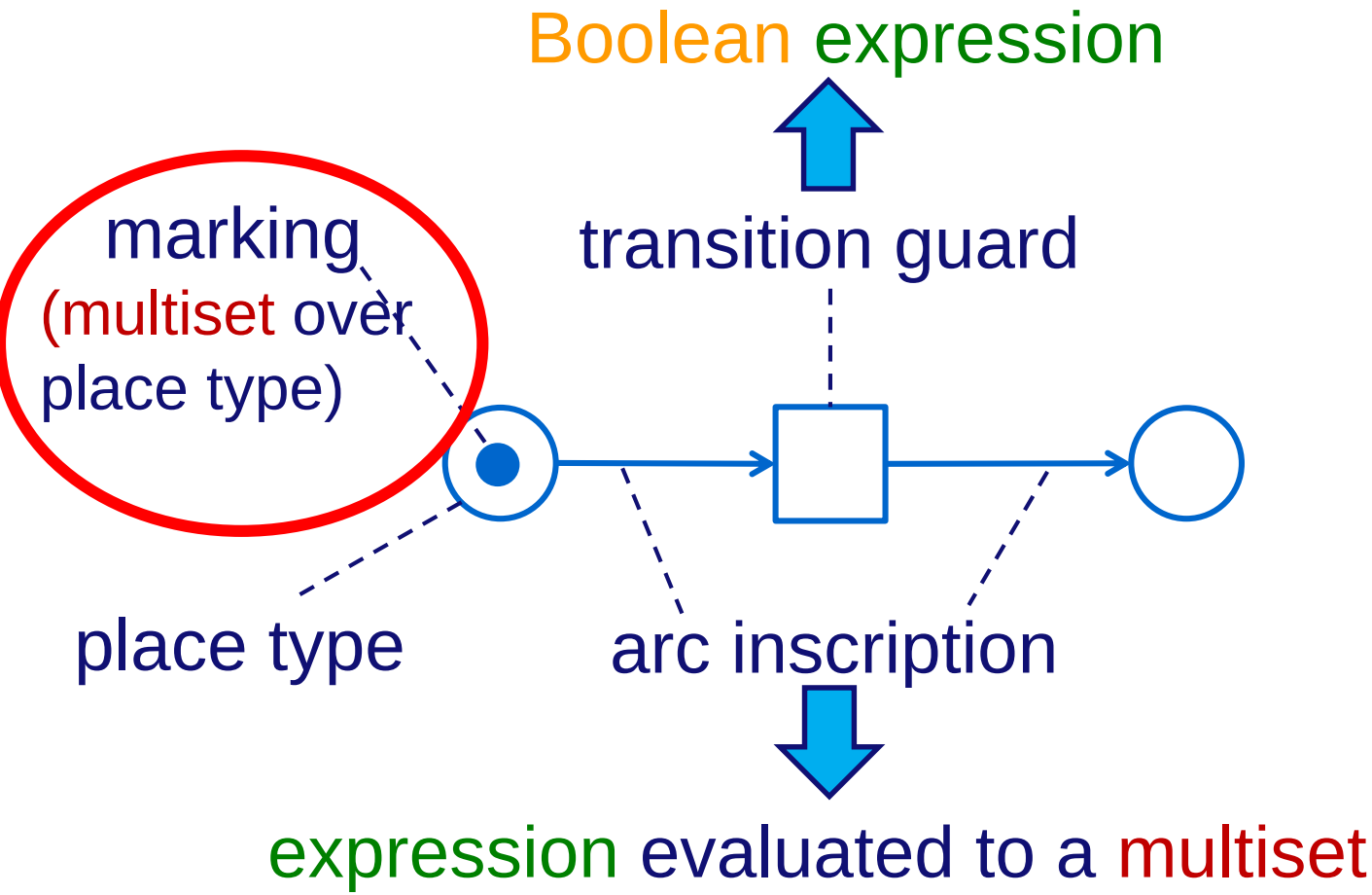
initial marking

place type

Outline

- Types and values
- Defining color sets
- Defining markings
- Defining arc inscriptions
- Defining guards
- Defining functions
- Defining transition priority
- Defining time

Markings revisited



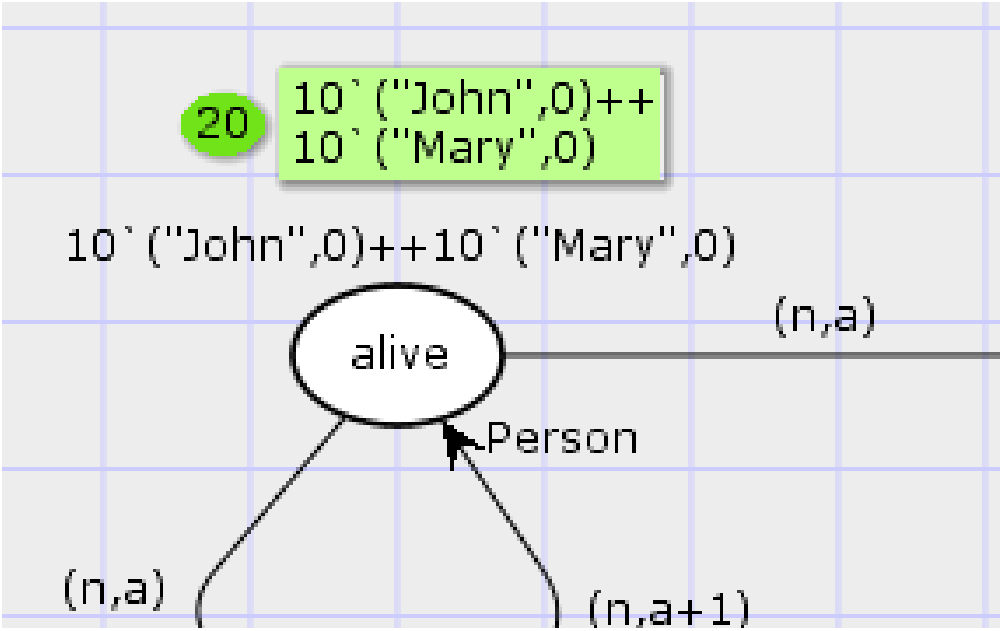
may contain constants and variables; by assigning a value to each variable, the value of this expression can be calculated

Multisets

Notation:

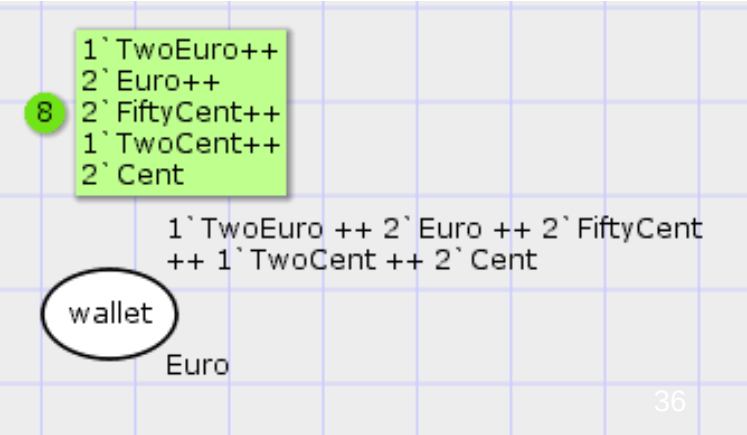
x1`v1 ++ x2`v2 ++ ... ++ xn`vn

where v1 denotes an element (i.e., value) of the multiset and x1 the multiplicity of v1 in the multiset, and so on



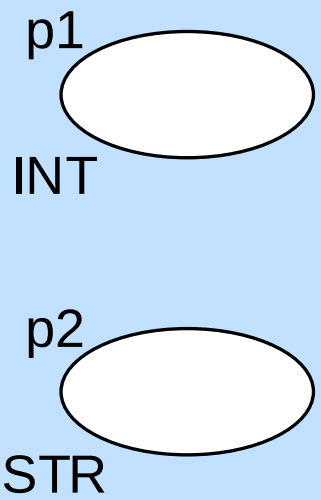
Example

- color Euro = with TwoEuro | Euro | FiftyCent | TwentyCent | TenCent | FiveCent | TwoCent | Cent;
- Multiset of type Euro:
  1`TwoEuro ++ 2`Euro ++ 2`FiftyCent ++ 1`TwoCent ++ 2`Cent

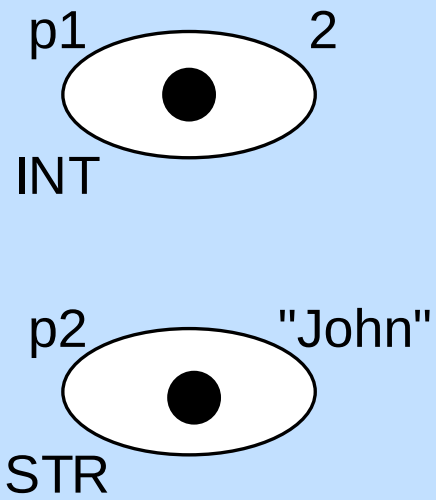


Initialization expressions

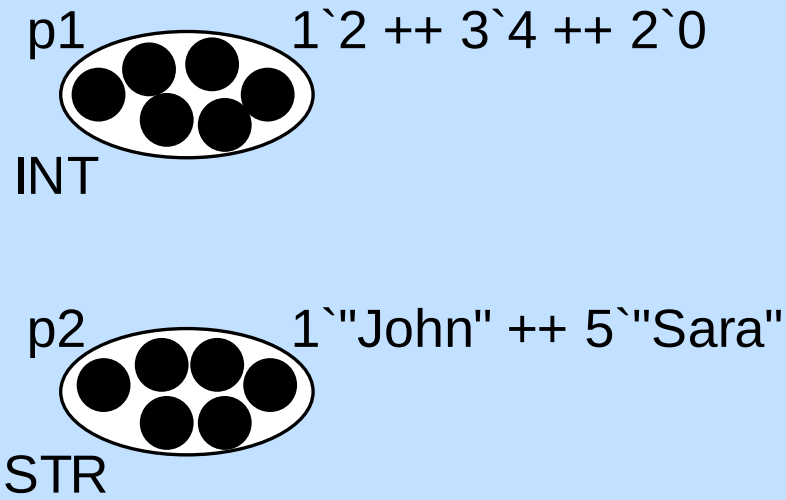
no tokens



one token



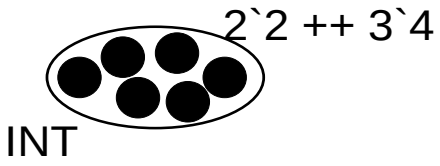
six tokens

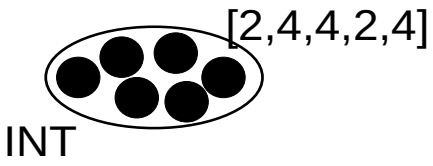


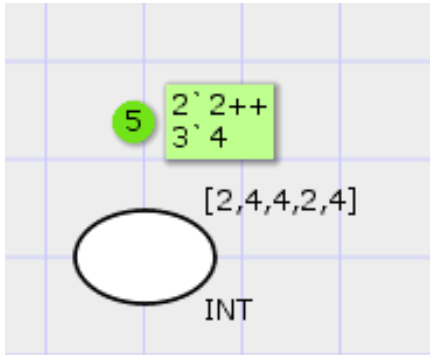
Multisets and Lists

- Multisets are implemented as lists,
- 1`"2Euro" ++ 2`"1Euro" ++ 2`"50Cent" ++ 1`"2Cent" ++ 2`"1Cent" can also be written as

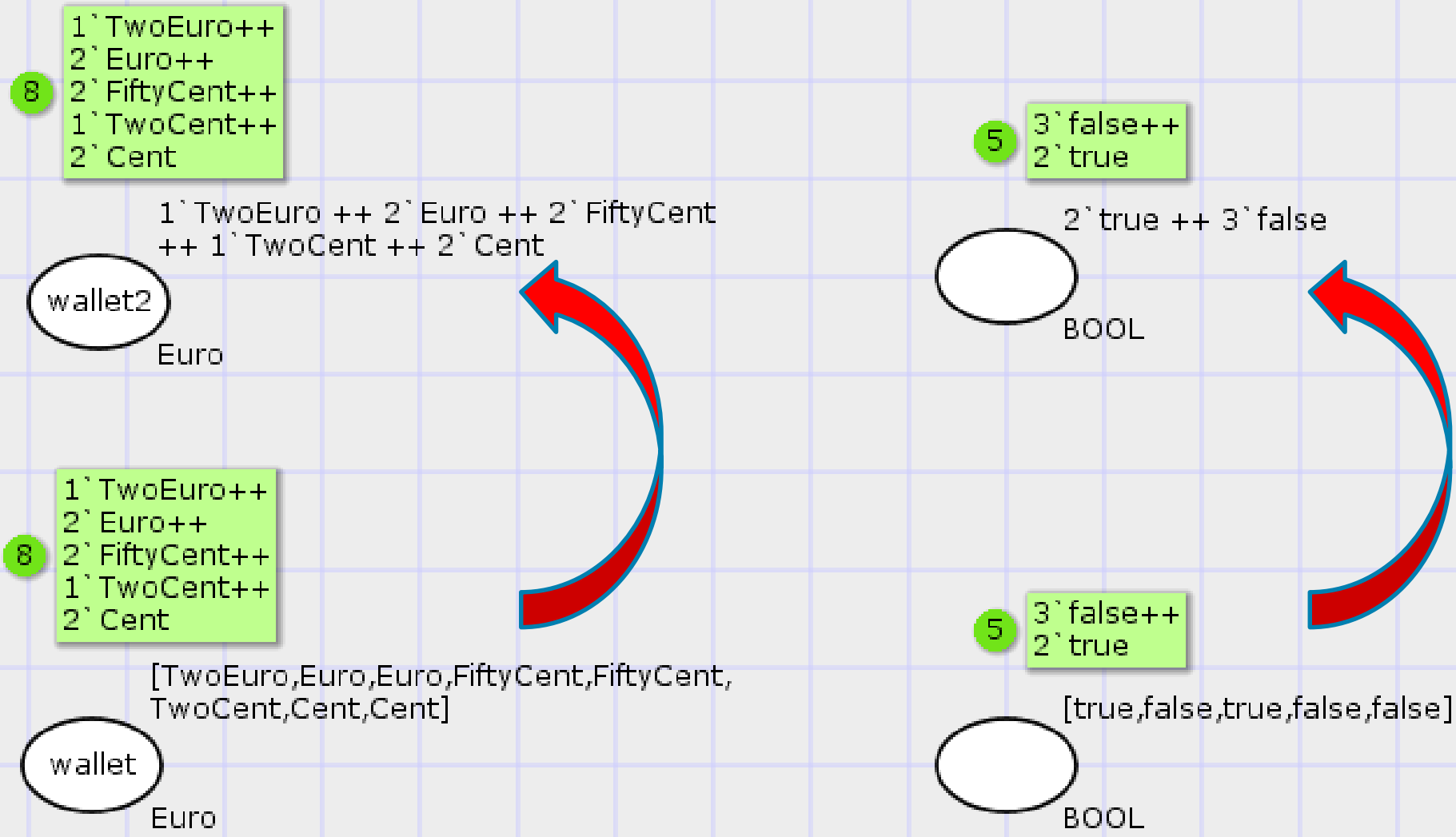
  ["2Euro", "1Euro", "1Euro", "50Cent", "50Cent", "2Cent", "1Cent", "1Cent"]
- useful when using list functions



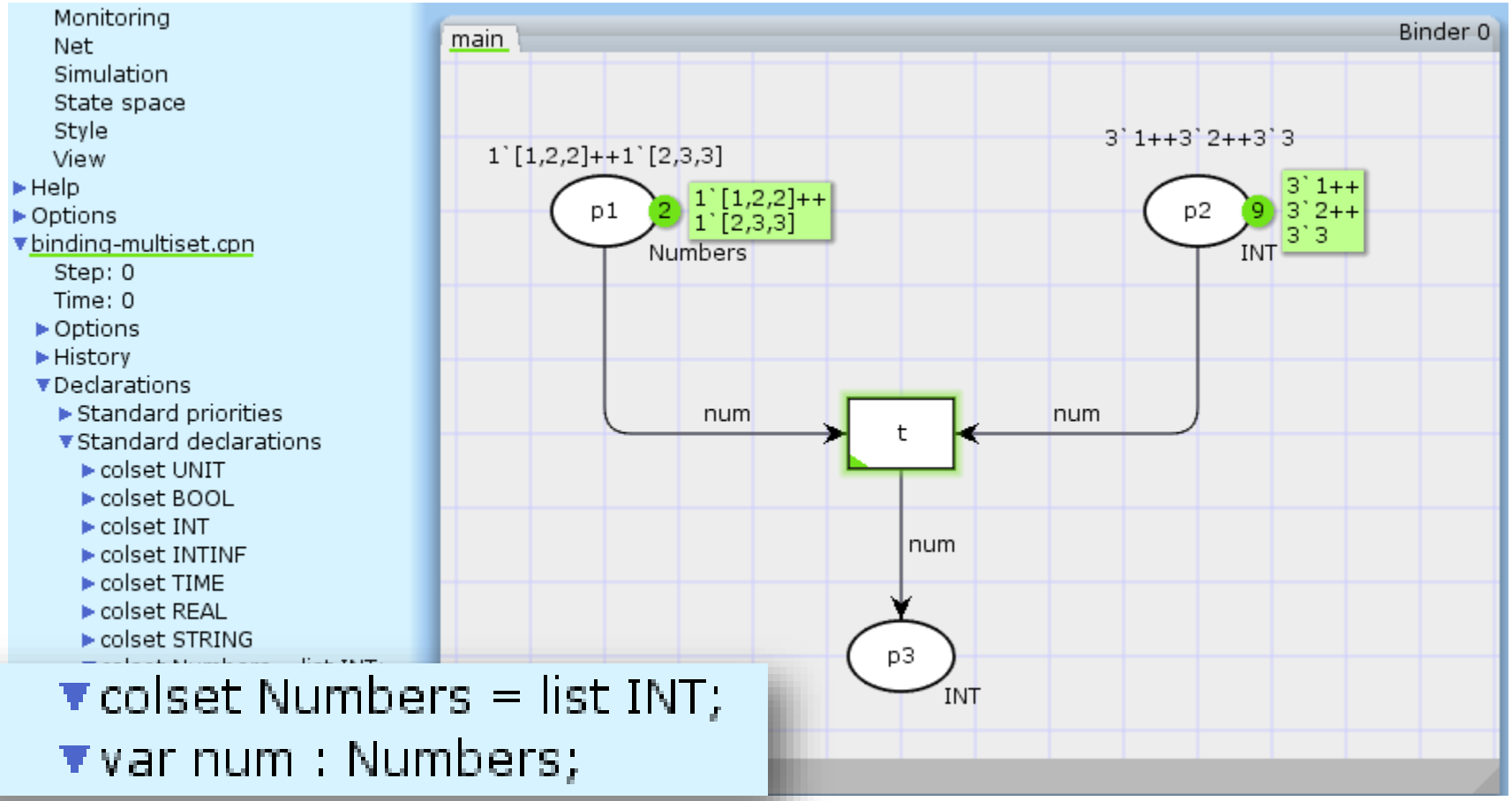




Examples: Effect is the same



Important use: consume a multiset



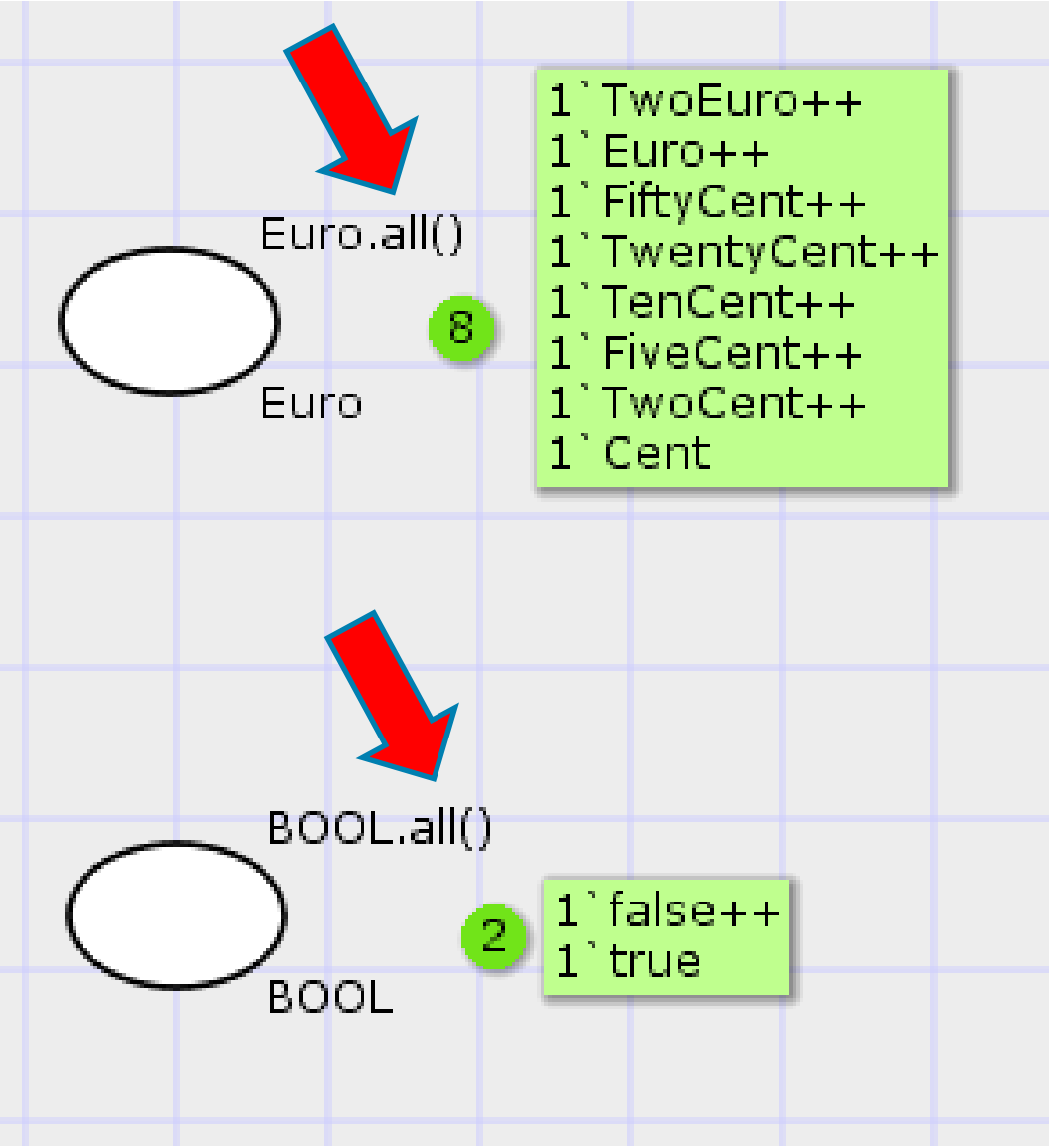
colset Numbers = list INT;

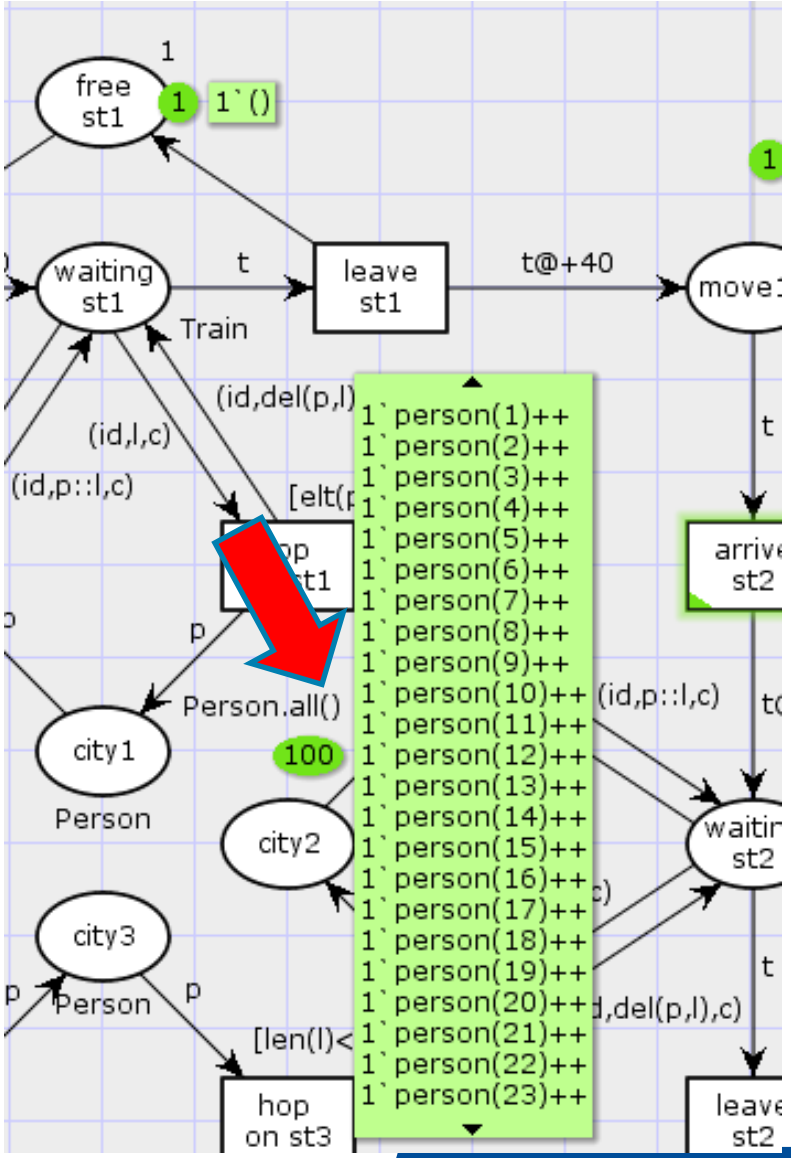
var num : Numbers;

What happens when t fires?

→ consume a multiset from p2, produce a multiset on p3

Only for "small color sets“ (index/with): Name.all()

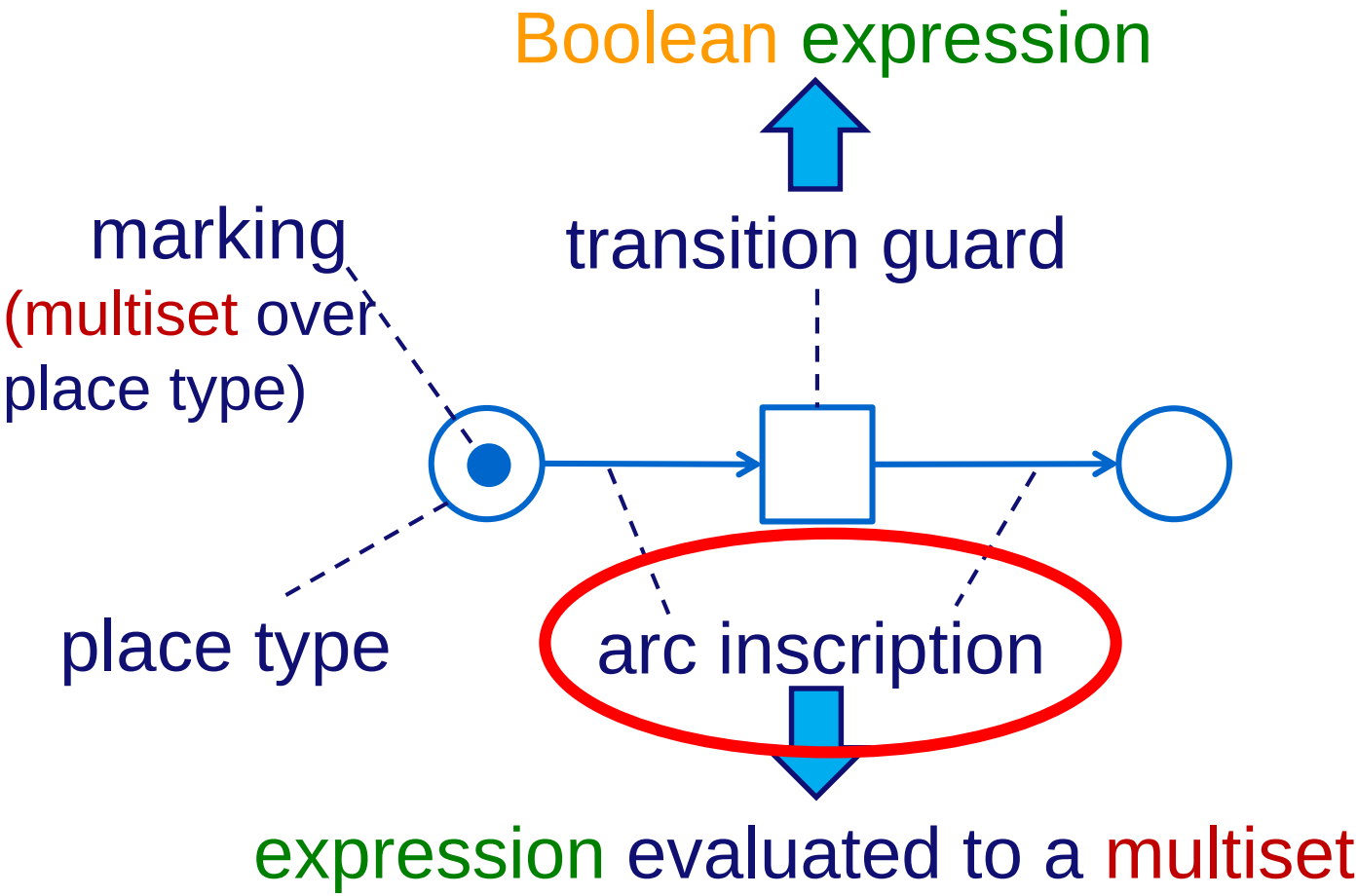




Outline

- Types and values
- Defining color sets
- Defining markings
- Defining arc inscriptions
- Defining guards
- Defining functions
- Defining transition priority
- Defining time

Arc inscription revised



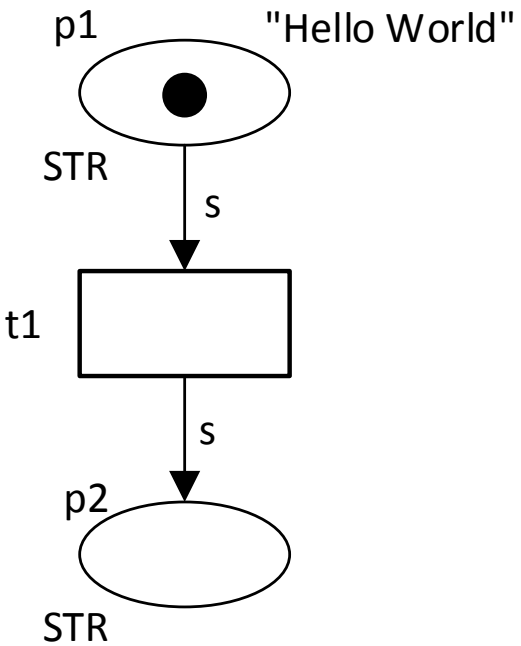
may contain constants and variables; by assigning a value to each variable, the value of this expression can be calculated

Arc inscriptions

- any expression that evaluates to singleton or multiset of the type of the adjacent place
- may contain constants (e.g., () or 2) and variables
- **variables** are typed and need to be declared

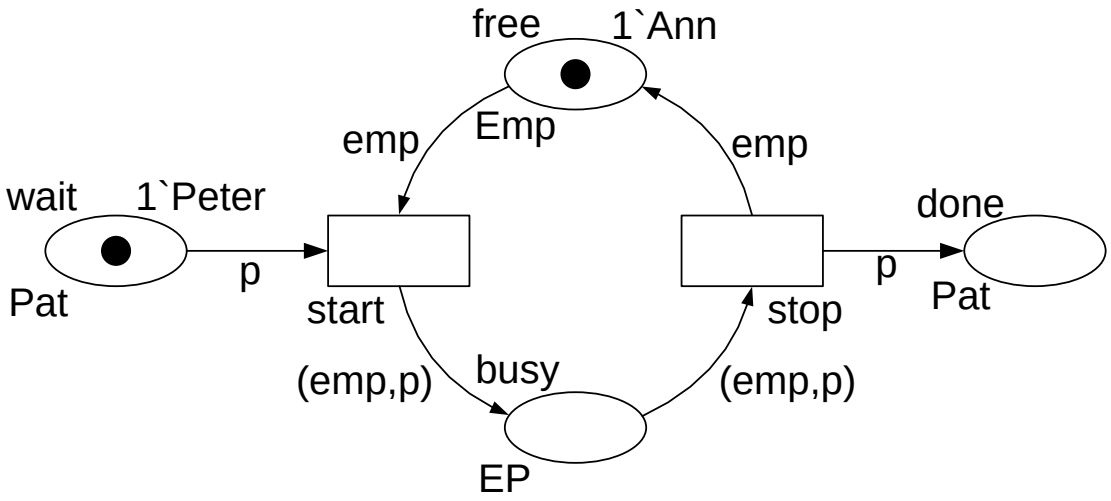
var varName:varType;

```
color STR = string;
var s:STR;
```



The service help desk

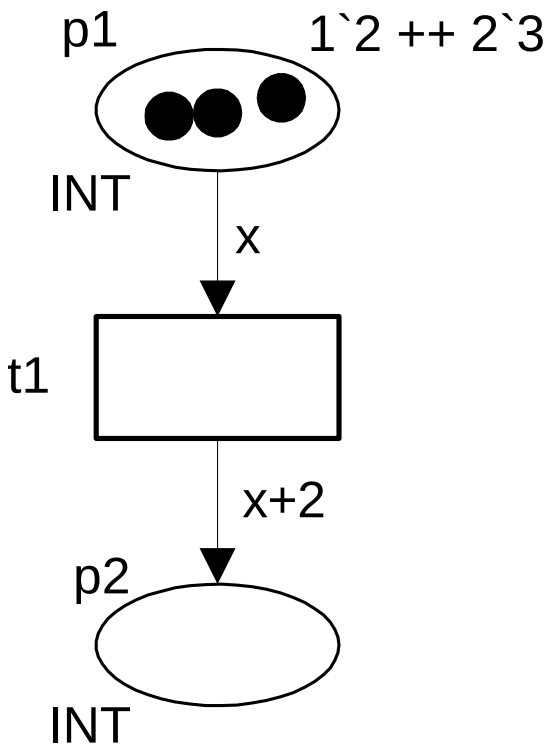
```
color Name = string;
color Address = string;
color DateOfBirth = string;
color PatientID = int;
color EmpNo = int;
color Experience = int;
color Gender = with male|female;
color Pat = product PatientID * Name * Address * DateOfBirth * Gender;
color Emp = product EmpNo * Experience;
color EP = product Emp * Pat;
var p:Pat;
var emp:Emp;
val Peter = (12345, "Peter", "Kerkstraat 10, Amsterdam", "13-Dec-1962", male);
val Ann = (641112, 7);
```



What happens when t1 fires?

- Give all bindings
- Give all enabled bindings

```
color INT = int;
var x:INT;
```

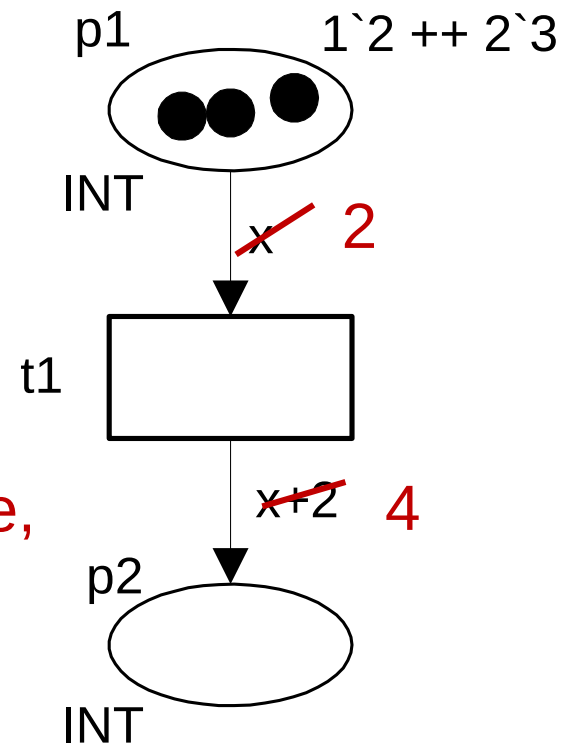


Bind variables, evaluate arc expressions → binding elements + effects

- Possible bindings of x: x=1, x=2, x=3, x=4, …
- Two bindings that enable t1:

(t1,⟨x=2⟩) and (t1,⟨x=3⟩)

```
color INT = int;
var x:INT;
```



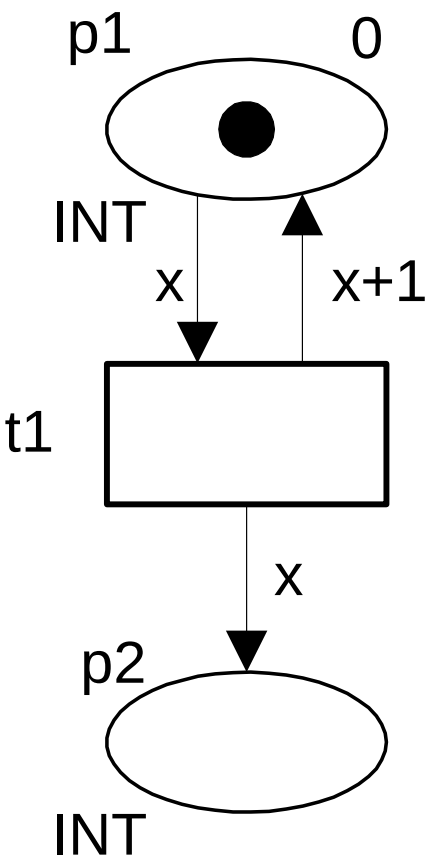
for chosen binding (x=2)

replace every occurrence of every variable with its value, evaluate expression

Example

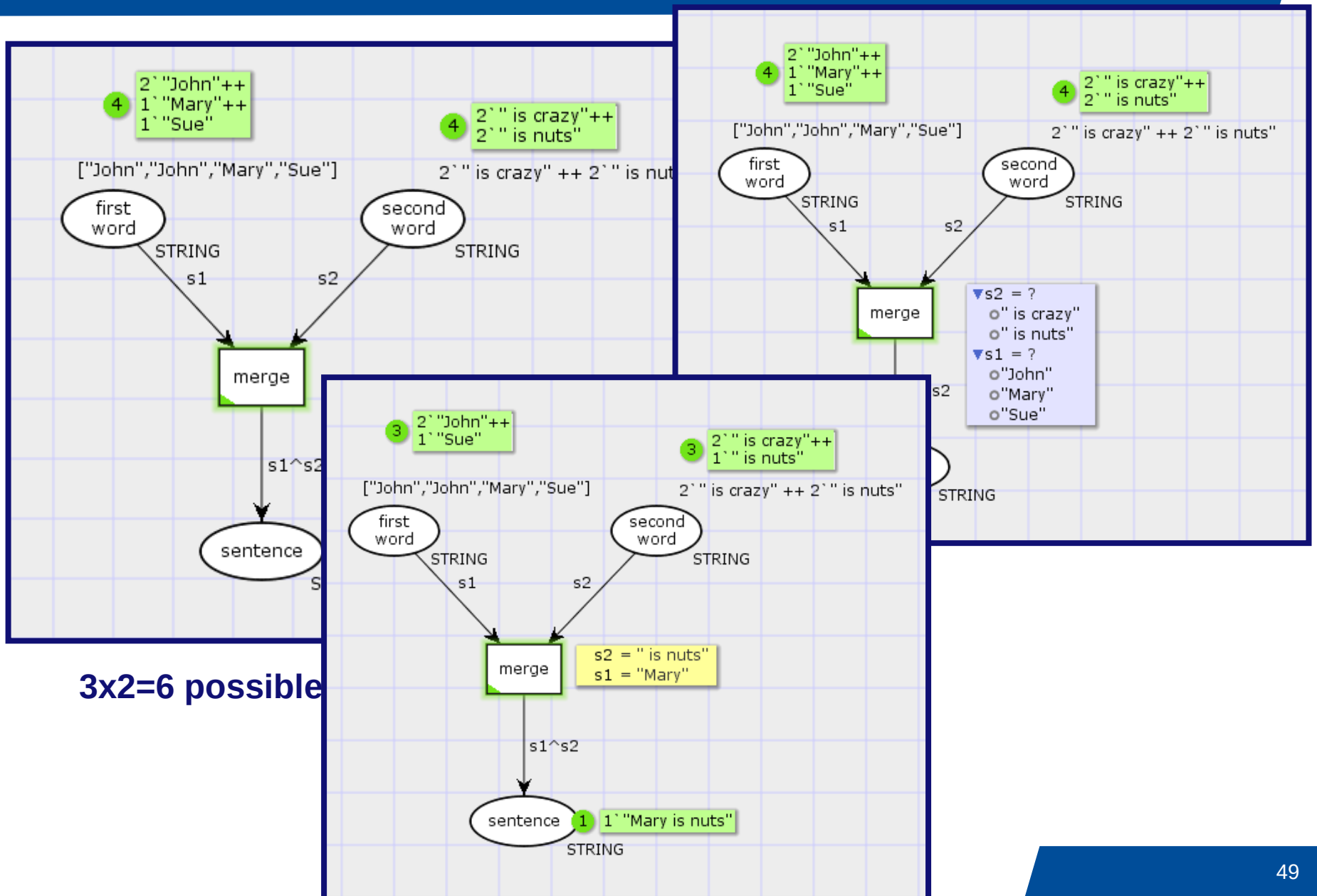
- Give all enabled bindings and the marking reached for the first 2 steps.

```
color INT = int;
var x:INT;
```



- Binding element (t1, ⟨x=0⟩).
- After it occurred (t1, ⟨x=1⟩), etc.

How many bindings are there?



3x2=6 possible

Bindings vs. Enabling Bindings

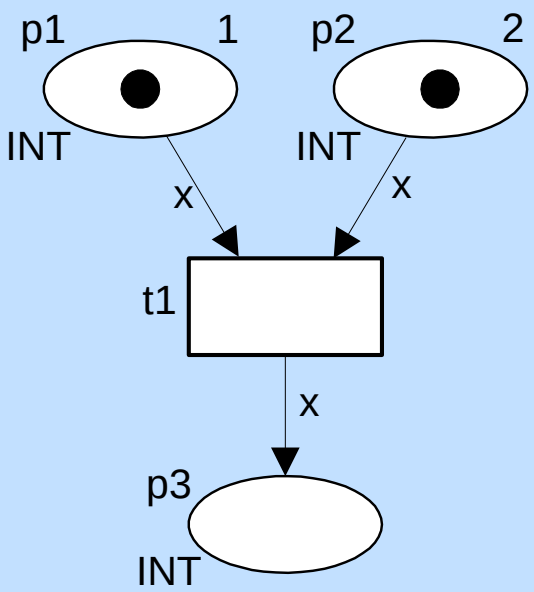
```
color INT = int;
var x:INT;
var y:INT;
```

a) Possible bindings for x and y?

(x=1,y=1), (x=2,y=1), (x=1, y=2), (x=1, y=-1), …

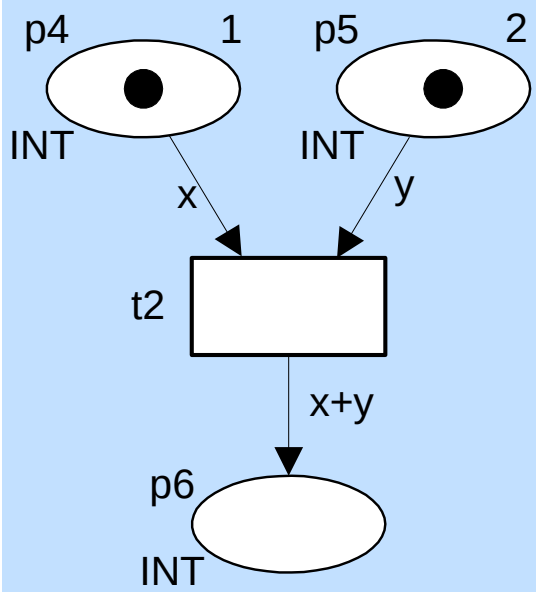
b) Possible bindings for x and y based on tokens on p1/p2?

c) Possible bindings for x and y that **enable** t1,t2,t3?



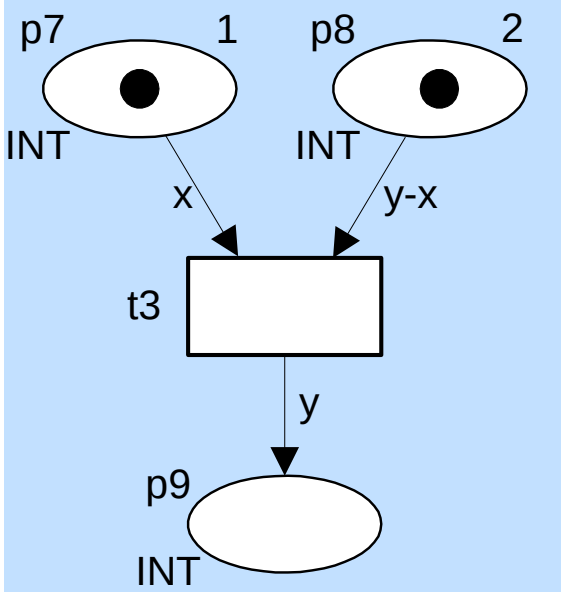
b) x=1 / x=2

c) none



b) x=1, y=2

c) x=1, y=2

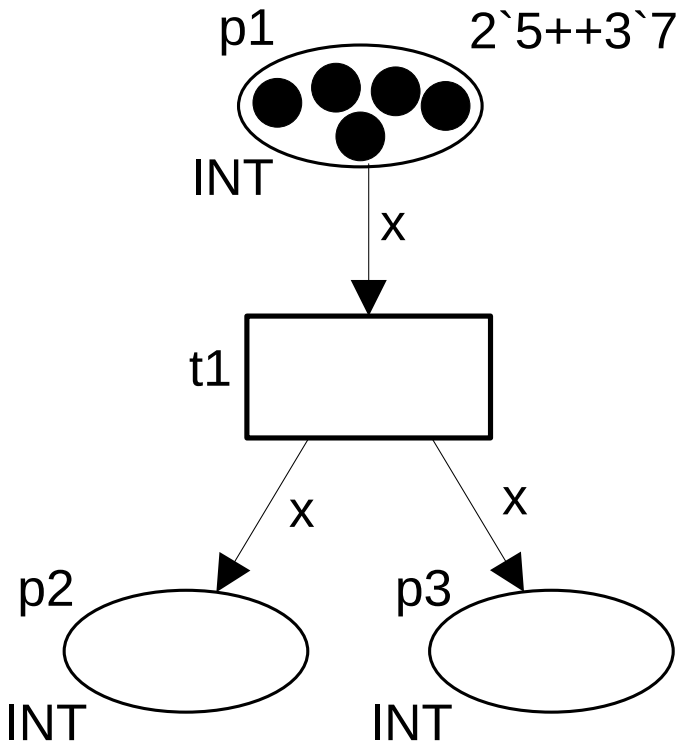


not in CPN tools!

Exercise

- Give all possible binding elements and final markings

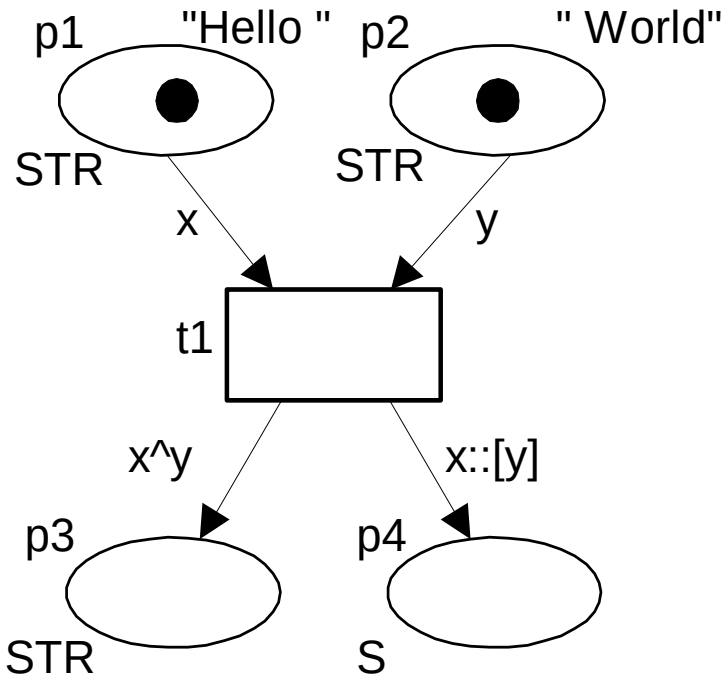
```
color INT = int;
var x:INT;
var y:INT;
```



Exercise

- Give all possible binding elements and a final marking

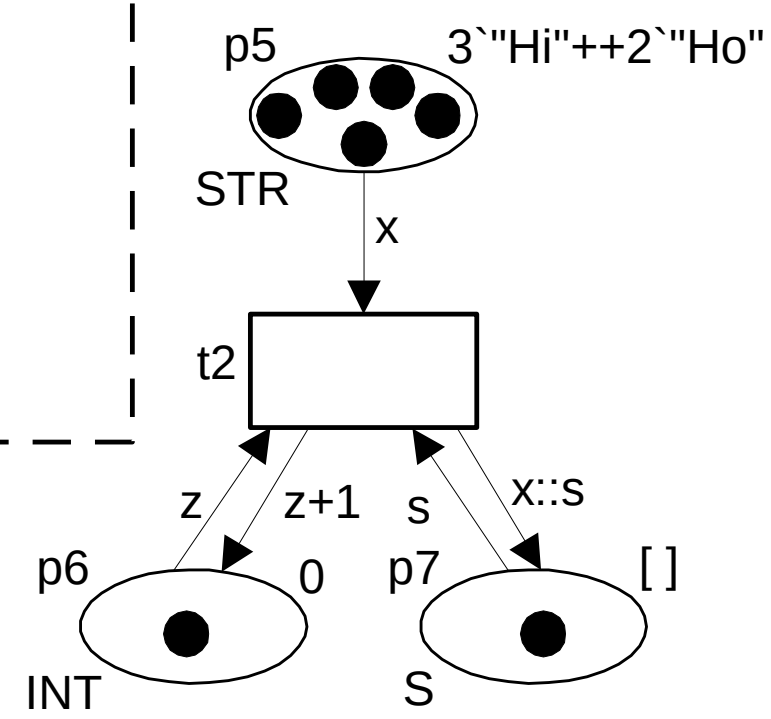
```
color STR = string;
var x:STR;
var y:STR;
color INT = int;
var z:INT;
color S = list STR;
var s:S;
```



Exercise

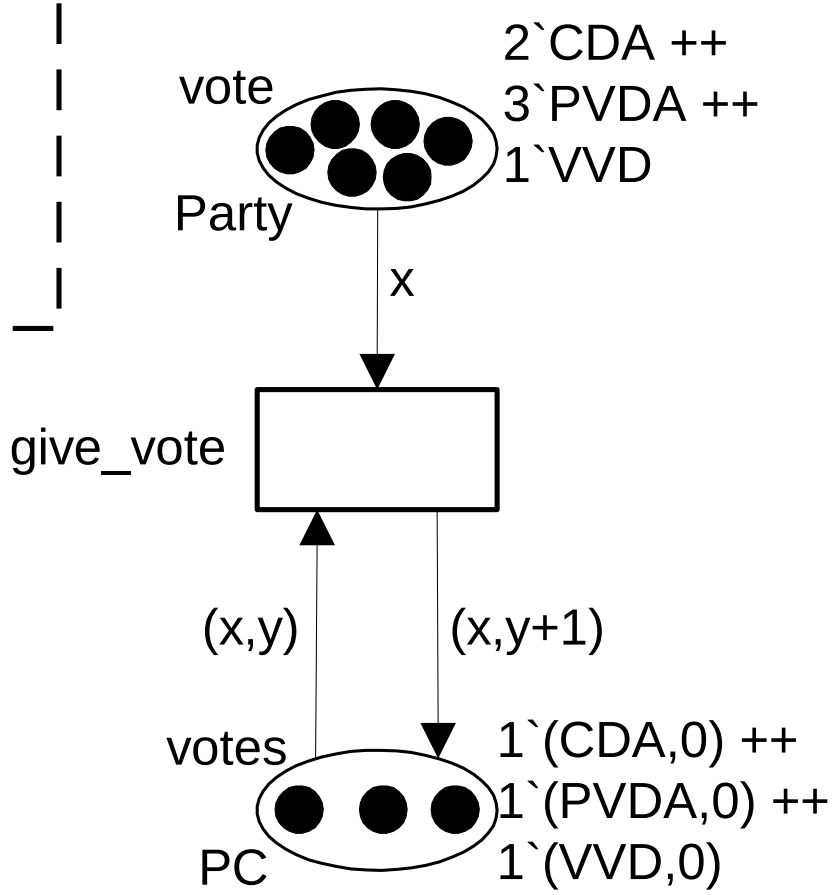
- Give all possible binding elements and a final marking

```
color STR = string;
var x:STR;
var y:STR;
color INT = int;
var z:INT;
color S = list STR;
var s:S;
```

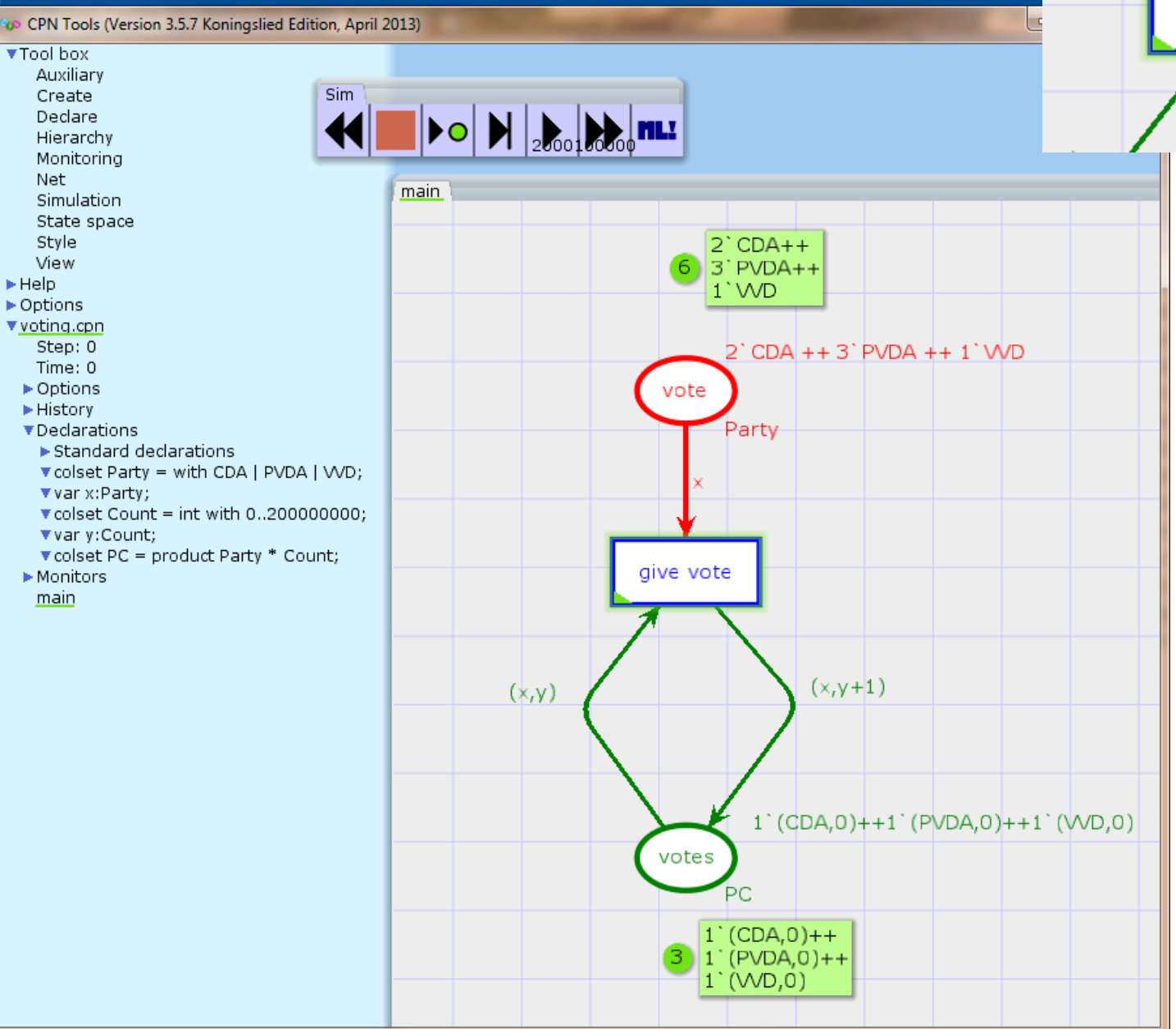


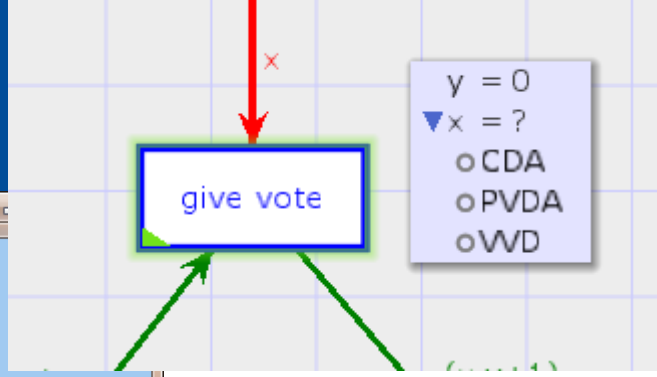
Example: Voting

```
color Party = with CDA | PVDA | VVD;
var x:Party;
color Count = int with 0..200000000;
var y:Count;
color PC = product Party * Count;
```



Voting


CPN Tools (Version 3.5.7 Koningslied Edition, April 2013)
Tool box
Auxiliary
Create
Declare
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
voting.cpn
Step: 0
Time: 0
Options
History
Declarations
Standard declarations
colset Party = with CDA | PVDA | VVD;
var x:Party;
colset Count = int with 0..200000000;
var y:Count;
colset PC = product Party * Count;
Monitors
main
Sim
main
6
2`CDA++
3`PVDA++
1`VVD
2`CDA ++ 3`PVDA ++ 1`VVD
vote
Party
x
give vote
(x,y)
(x,y+1)
1`(CDA,0)++1`(PVDA,0)++1`(VVD,0)
votes
PC
3
1`(CDA,0)++
1`(PVDA,0)++
1`(VVD,0)

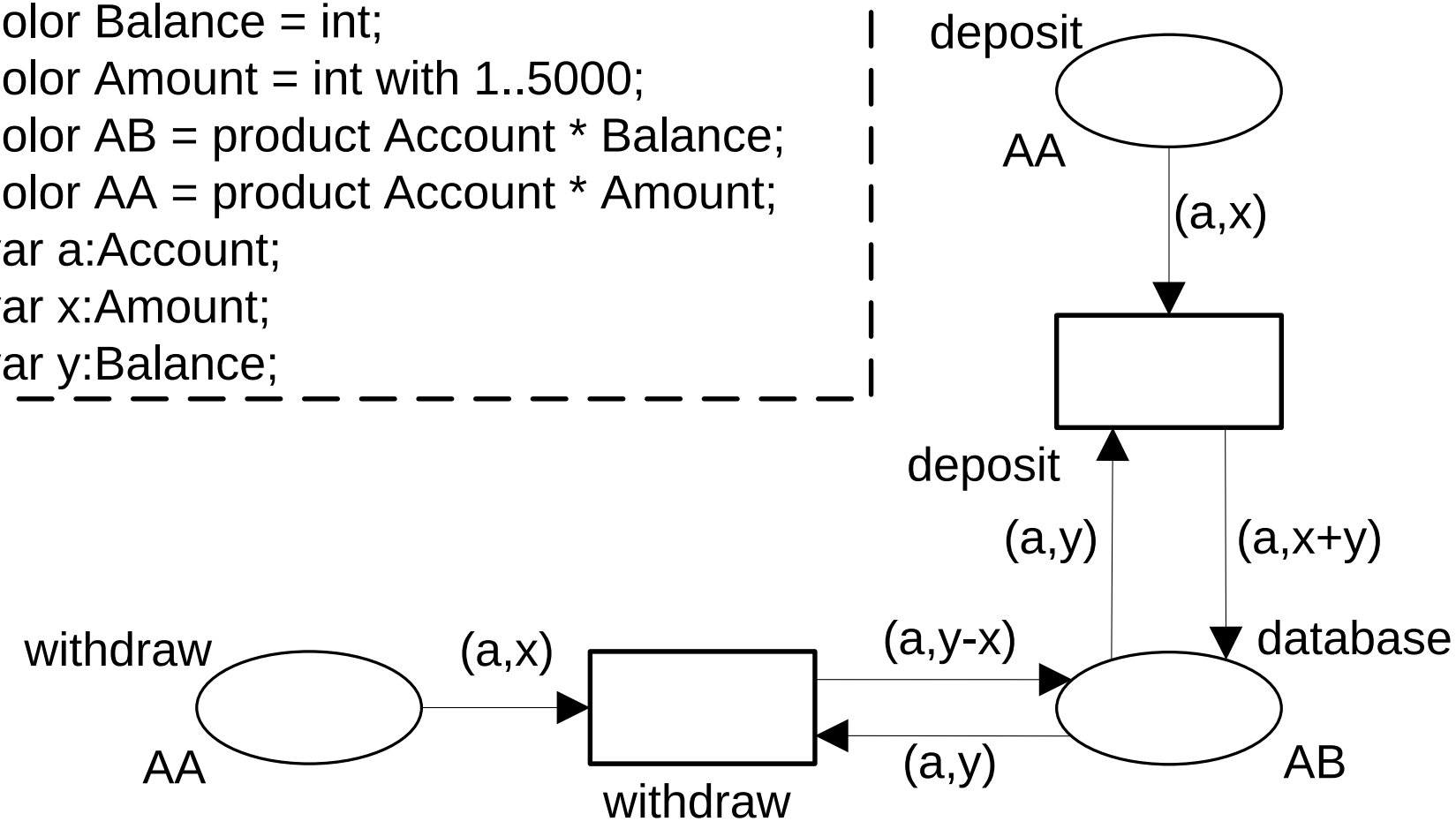

x
give vote
y = 0
x = ?
CDA
PVDA
VVD

Exercise

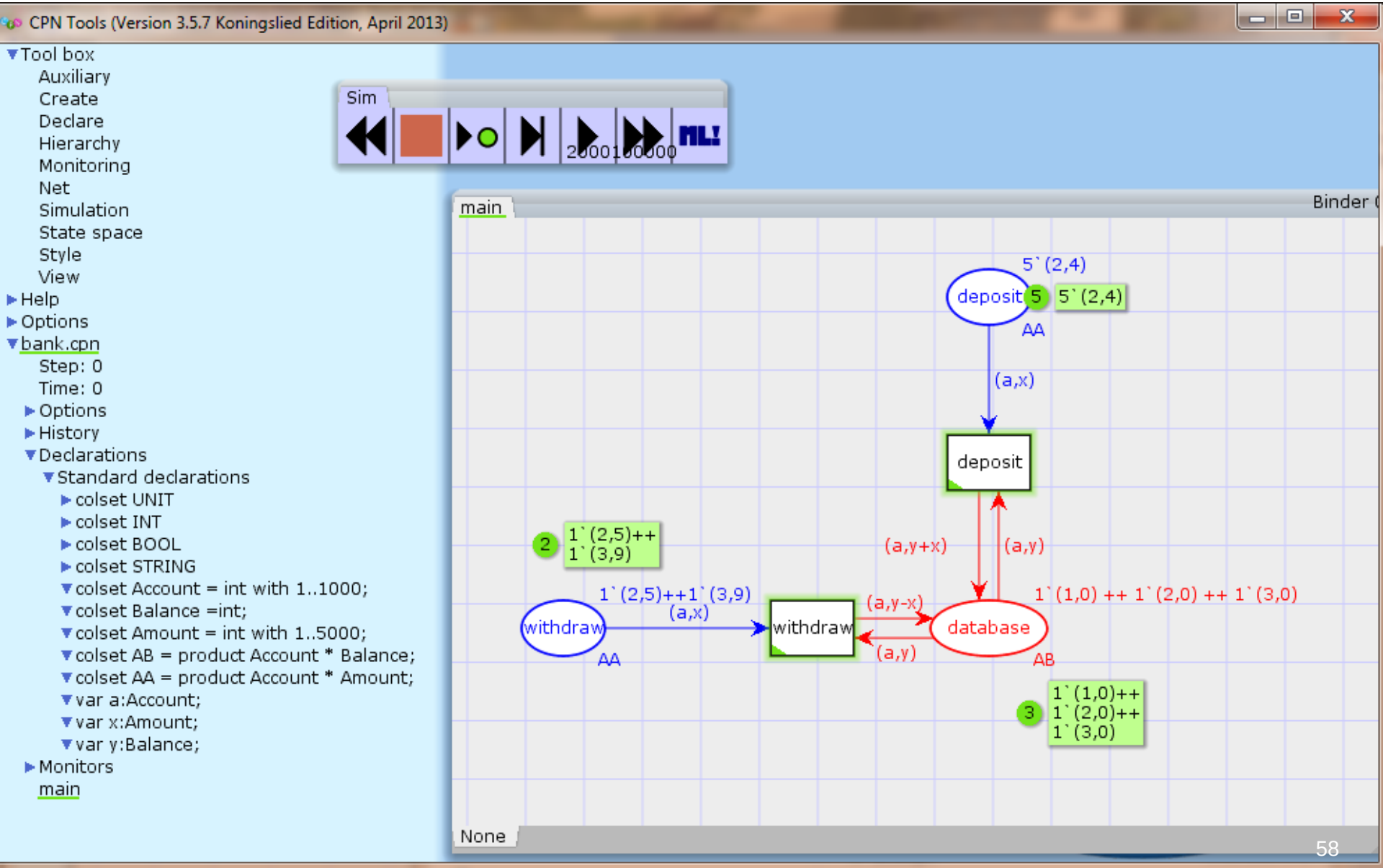
- Consider a simple banking system.
- The system manages accounts.
- Each account has an account number (1, …, 1000) and a balance.
- Account holders may deposit or withdraw money, but only amounts less than 5000 Euro.

Solution

```
color Account = int with 1..1000;
color Balance = int;
color Amount = int with 1..5000;
color AB = product Account * Balance;
color AA = product Account * Amount;
var a:Account;
var x:Amount;
var y:Balance;
```



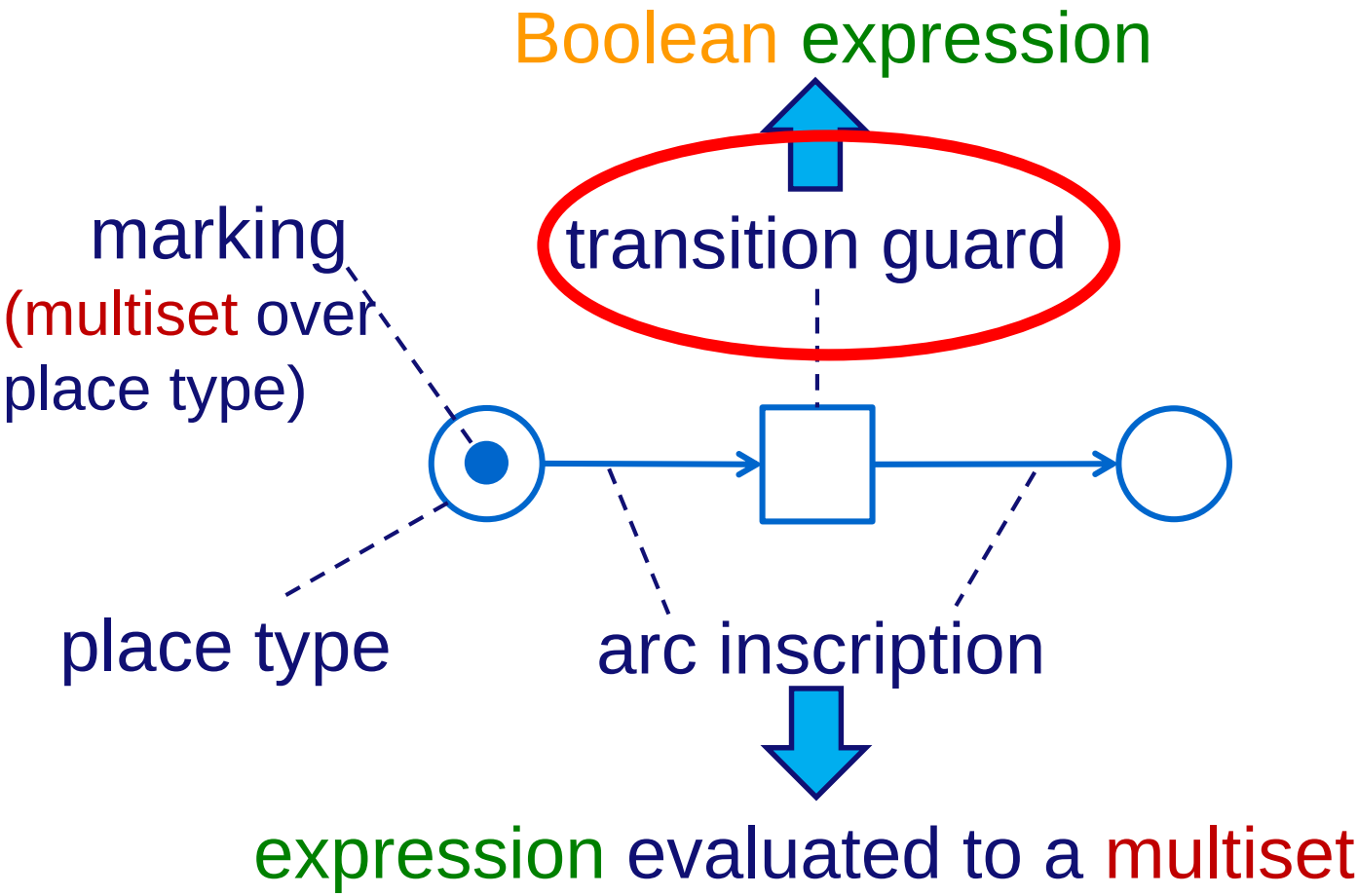
Bank in CPN Tools



Outline

- Types and values
- Defining color sets
- Defining markings
- Defining arc inscriptions
- Defining guards
- Defining functions
- Defining transition priority
- Defining time

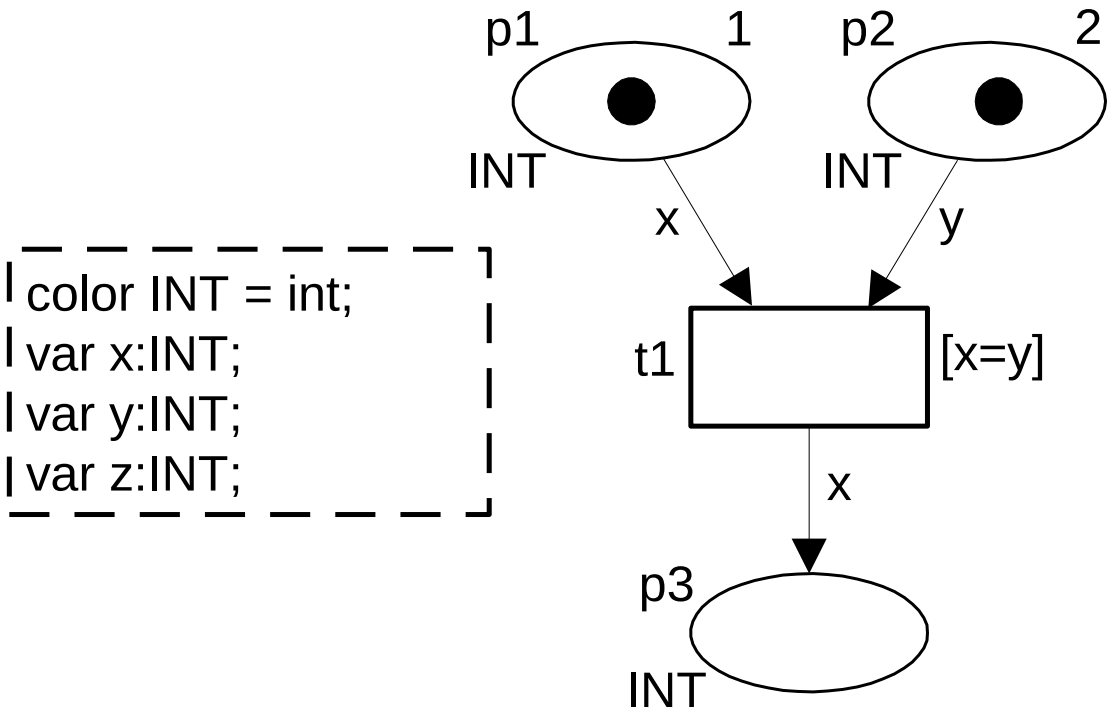
Arc inscription revisited



may contain constants and variables; by assigning a value to each variable, the value of this expression can be calculated

Guard

- a Boolean expression attached to a transition
- Notation: [Guard]

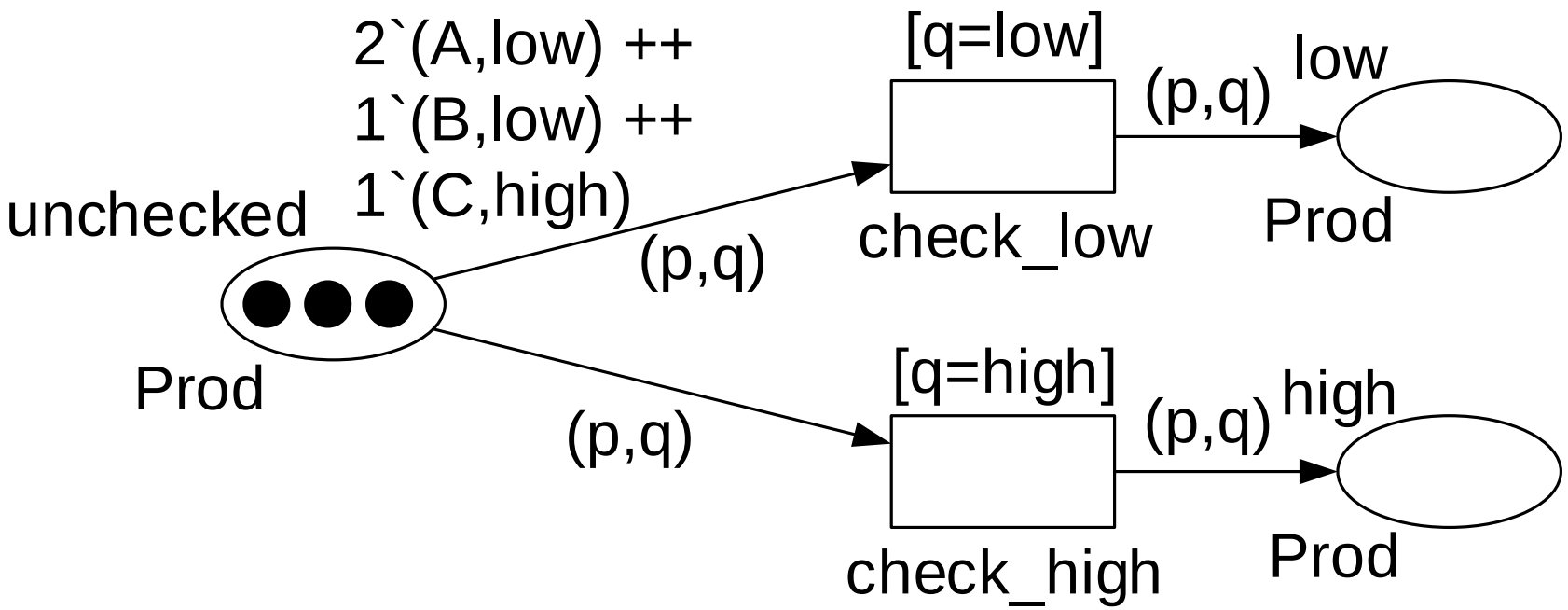


guard evaluates to false for binding (t1,⟨x=1, y=2⟩)

- transition enabled only if guard evaluates to true

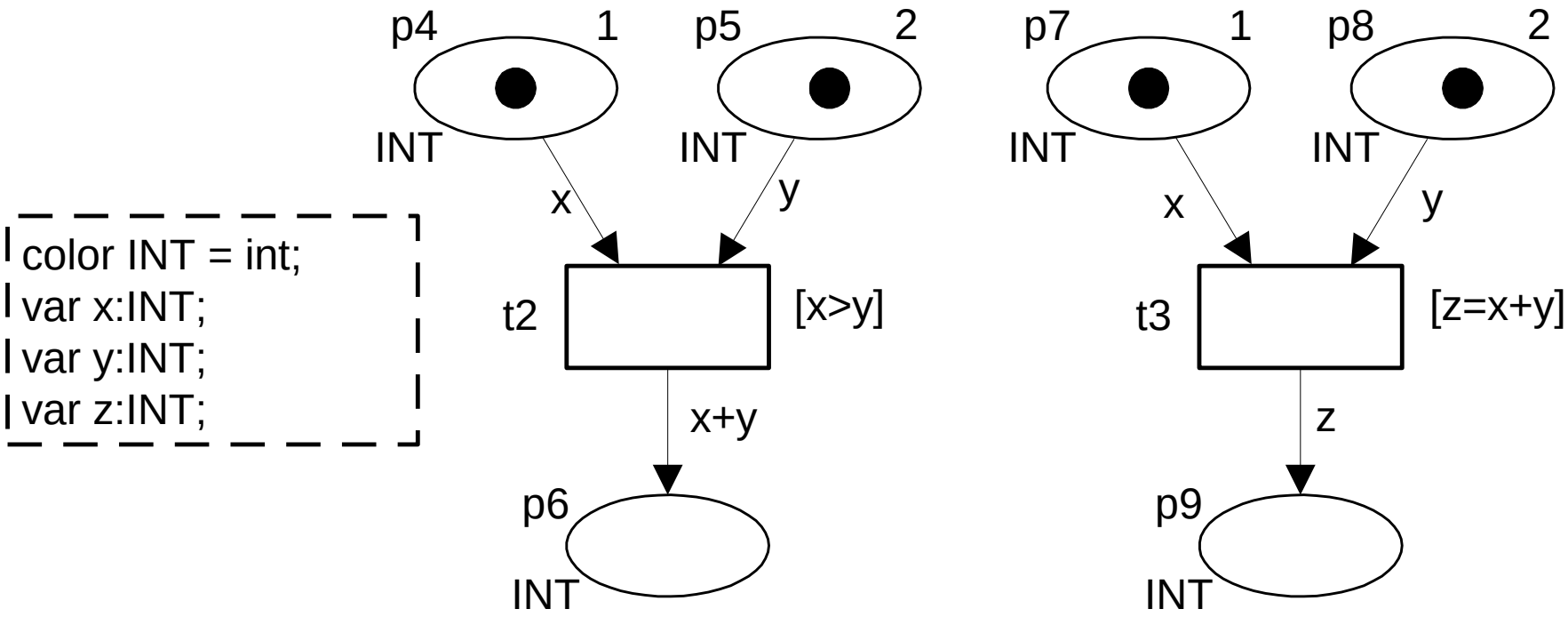
The product quality check

```
color ProdType = with A|B|C;
color Quality = with high|low;
color Prod = product ProdType * Quality;
var p:ProdType; var q:Quality;
```



Example

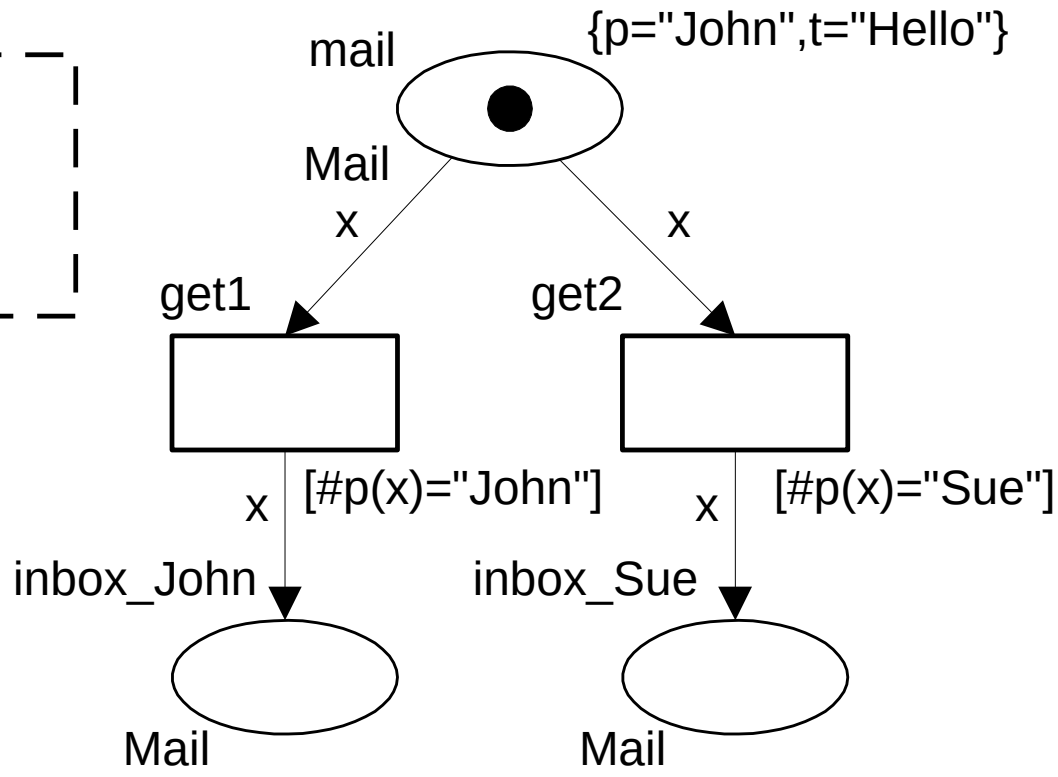
- Give all enabled bindings and the final marking.



Exercise

- Give all enabled binding elements and the final marking

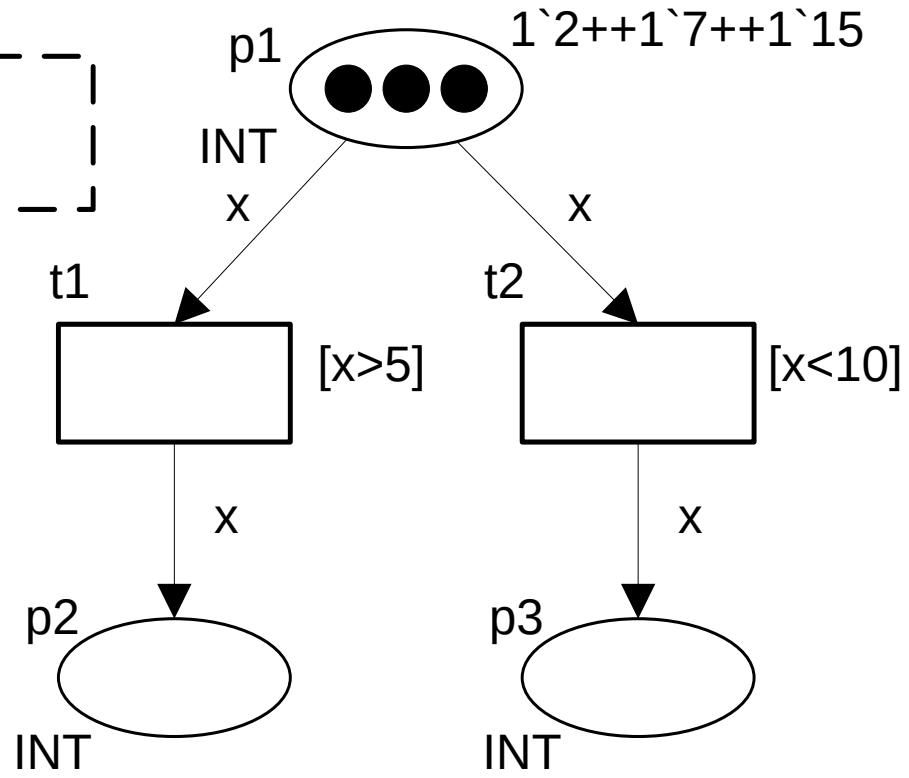
```
color Person= str;
color Text = str;
color Mail = record p:Person * t:Text;
var x:Mail;
```



Exercise

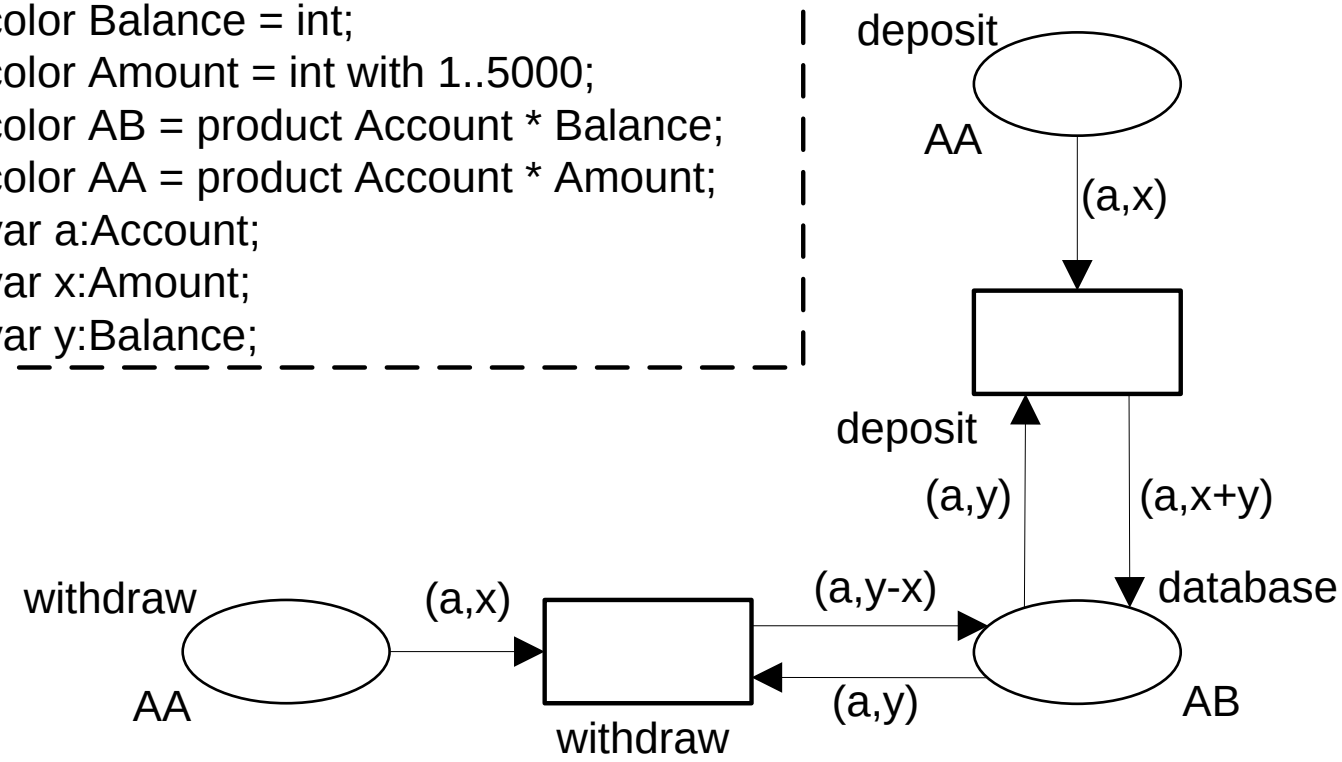
- Give all enabled binding elements and all possible final marking

```
color INT = int;
var x:INT;
```



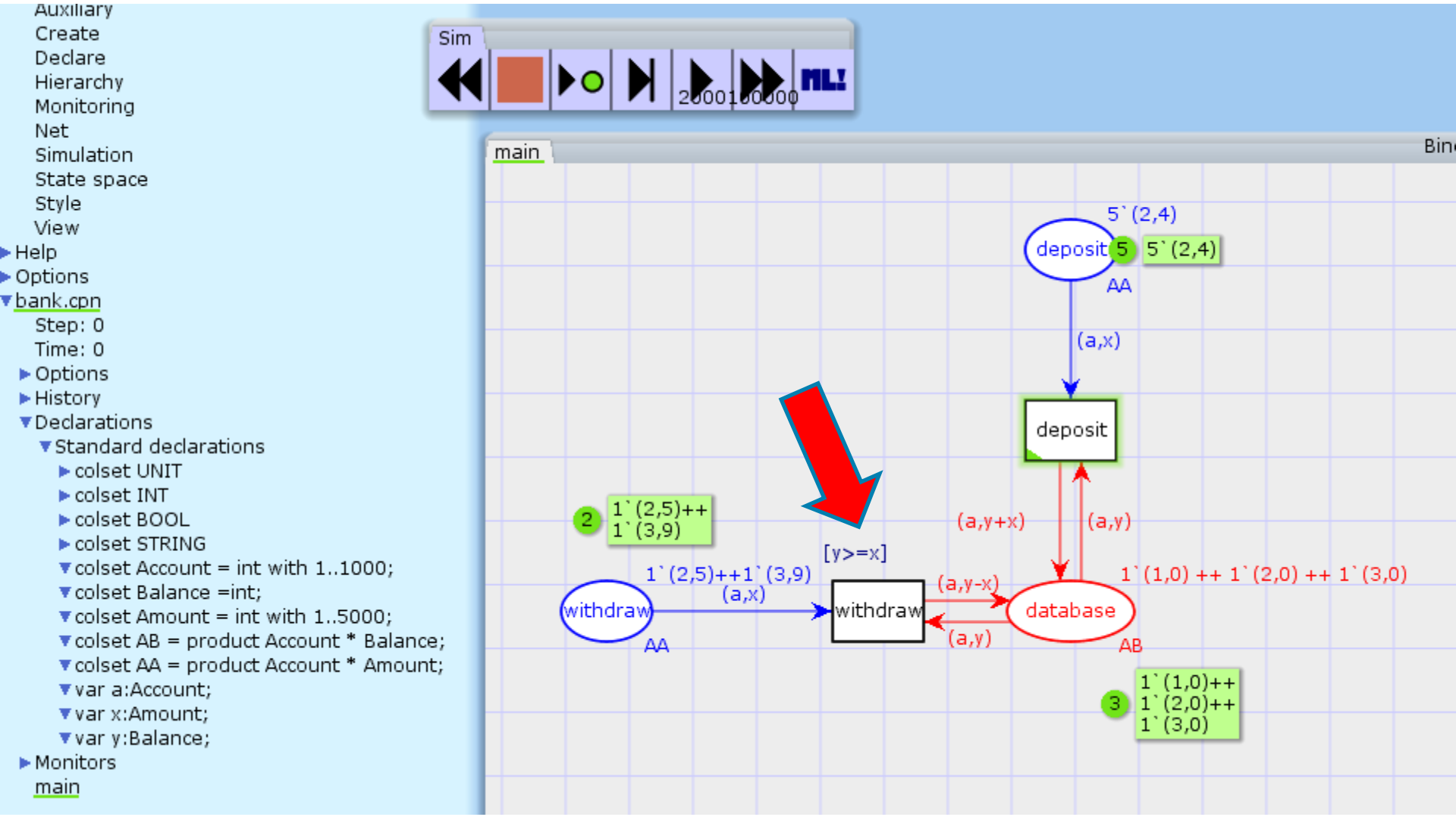
Exercise

```
color Account = int with 1..1000;
color Balance = int;
color Amount = int with 1..5000;
color AB = product Account * Balance;
color AA = product Account * Amount;
var a:Account;
var x:Amount;
var y:Balance;
```



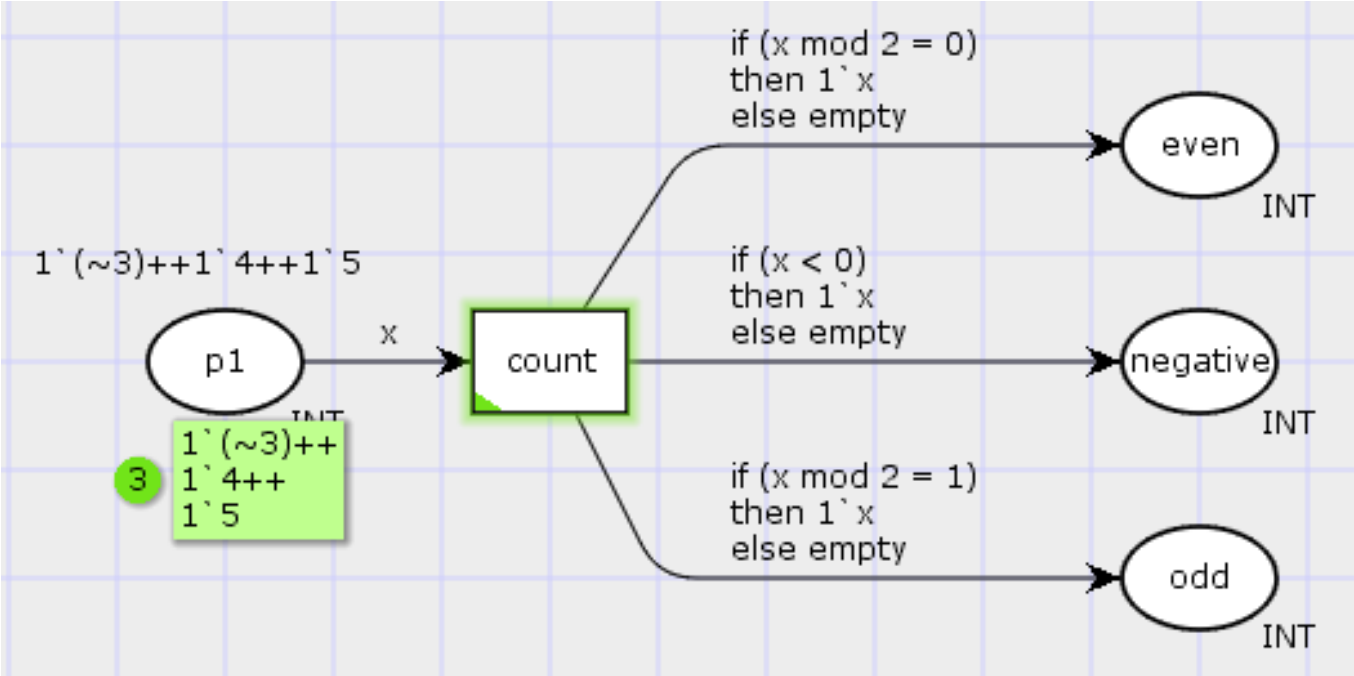
- The CPN model assumes that an account could have a negative balance. Change the model such that “withdraw” does not lead to a negative balance.

Guard


Auxiliary
Create
Declare
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
bank.cpn
Step: 0
Time: 0
Options
History
Declarations
Standard declarations
colset UNIT
colset INT
colset BOOL
colset STRING
colset Account = int with 1..1000;
colset Balance =int;
colset Amount = int with 1..5000;
colset AB = product Account * Balance;
colset AA = product Account * Amount;
var a:Account;
var x:Amount;
var y:Balance;
Monitors
main
Sim
main
5`(2,4)
deposit
5
5`(2,4)
AA
(a,x)
deposit
2
1`(2,5)++
1`(3,9)
(a,y+x)
(a,y)
[y>=x]
1`(2,5)++1`(3,9)
(a,x)
withdraw
withdraw
(a,y-x)
database
1`(1,0) ++ 1`(2,0) ++ 1`(3,0)
(a,y)
AA
AB
3
1`(1,0)++
1`(2,0)++
1`(3,0)

Guards on arcs

Combine “if … then … else …” on arcs (evaluate to multiset) with “empty” (= produce empty multiset on place)



What is the final marking?

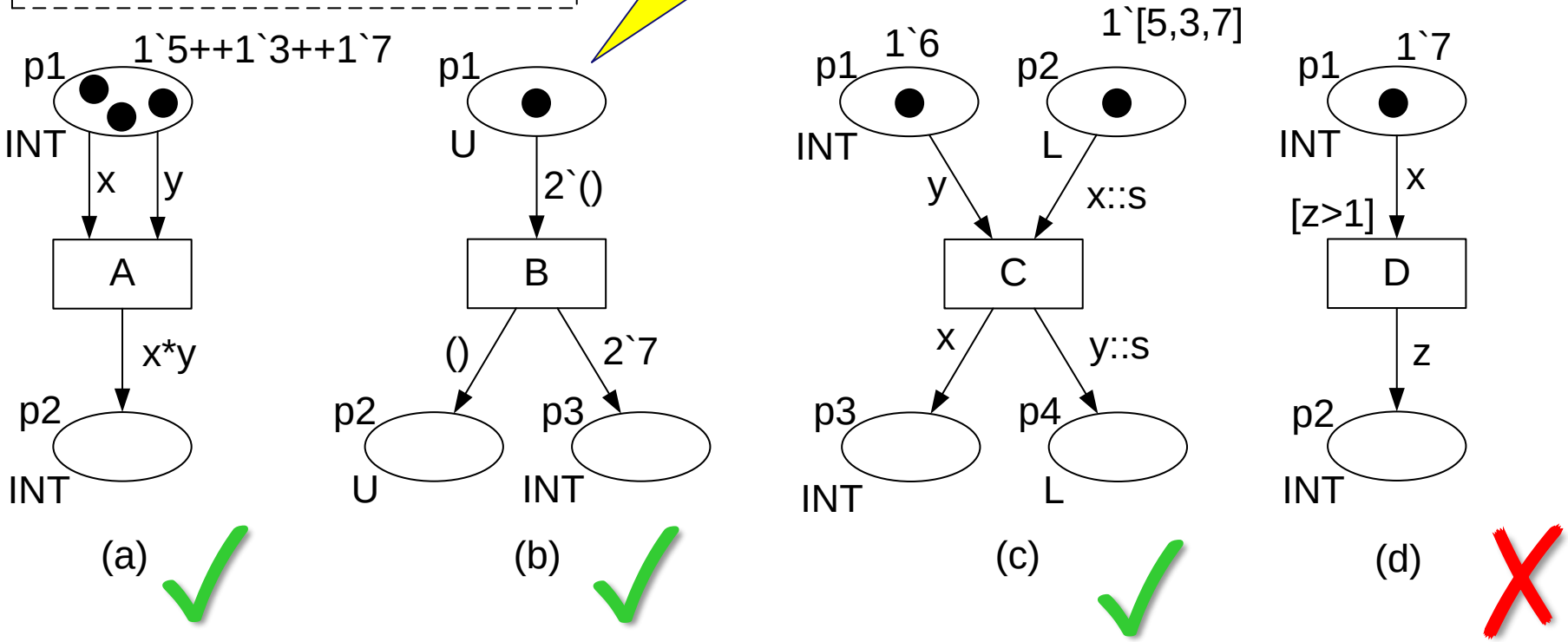
Useful when token needed on several, but not all post-places (“inclusive OR” – split)

To sum up: which bindings are (in)correct?

```
color INT = int;
color U = unit;
color L = list INT;
color R = record a:INT * b:INT;
var x,y,z:INT;
var s:L;
```

Please note that it is correct but, with this marking, B is not enabled

z is unbound



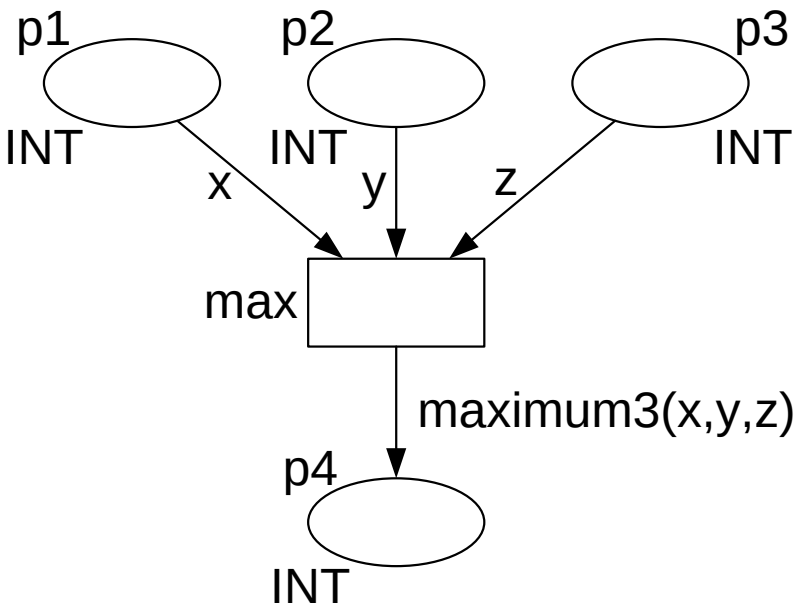
Outline

- Types and values
- Defining color sets
- Defining markings
- Defining arc inscriptions
- Defining guards
- Defining functions
- Defining transition priority
- Defining time

Functions and why do we need them

- To encode more complex calculations
- Can be used in guards, arc inscriptions, initialization expressions

```
color INT = int;
var x,y,z:INT;
fun maximum3(a:INT,b:INT,c:INT) =
   if (b>c) andalso (b>a) then b
   else if a>c then a else c;
```



Where to find standard functions?

- Sheets of the lecture
- cpntools.org, see for example http://cpntools.org/documentation/concepts/colors/declarations/colorsets/list_colour_sets and http://cpntools.org/documentation/concepts/colors/declarations/colorsets/colour_set_functions
- Standard ML
- http://www.cs.cmu.edu/~rwh/smlbook/book.pdf for book, https://www.cs.princeton.edu/~appel/smlnj/basis/ for functions, https://www.cs.princeton.edu/~appel/smlnj/basis/list.html

  for list functions, etc.

Where to find standard functions?

CPN Tools
Access/CPN
Books
Documentation
Concepts
Example nets
Graphical User Interface
Source Code
Tasks in CPN Tools
Create Declare Constraint
Installing CPN Tools
Download
FAQ
Getting Started
Grade/CPN
Licensing
Contact
Publications
Support

Google™ Custom Search

g+1 155

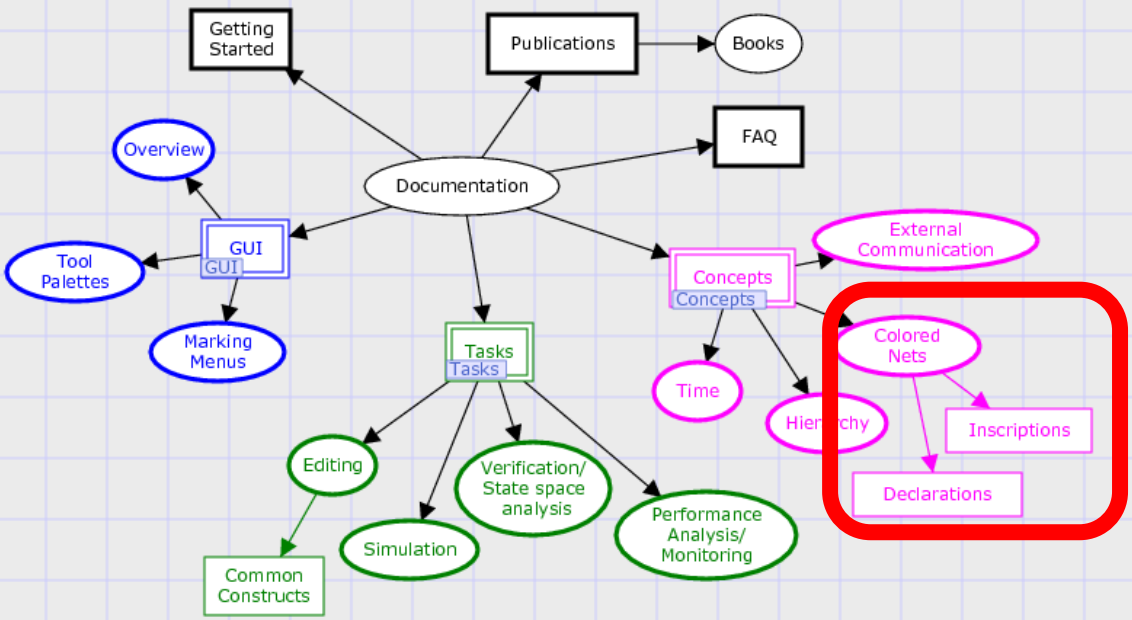
CPN Tools
Like

942 people like CPN Tools.

Facebook social plugin

Home | Download | Getting Started | **Documentation** | Support | Contact

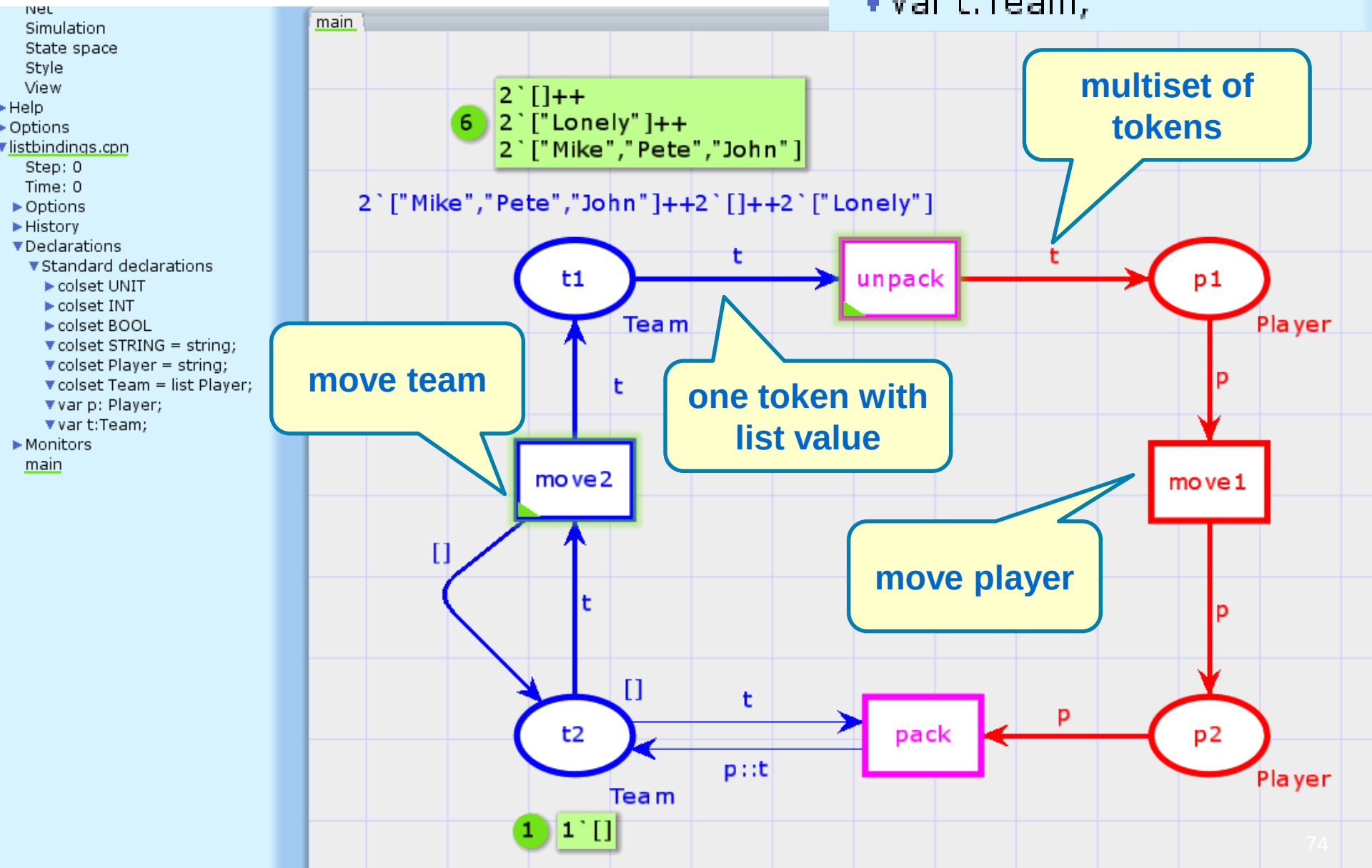
Documentation



- Getting Started
- Graphical User Interface
  - Graphical User Interface
  - Palette tools
  - Marking menus
- Tasks in CPN Tools
  - Editing a CPN
    - Common constructs
  - Simulation
  - Verification
  - Performance analysis
- Concepts
  - Timed Nets
  - Hierarchy

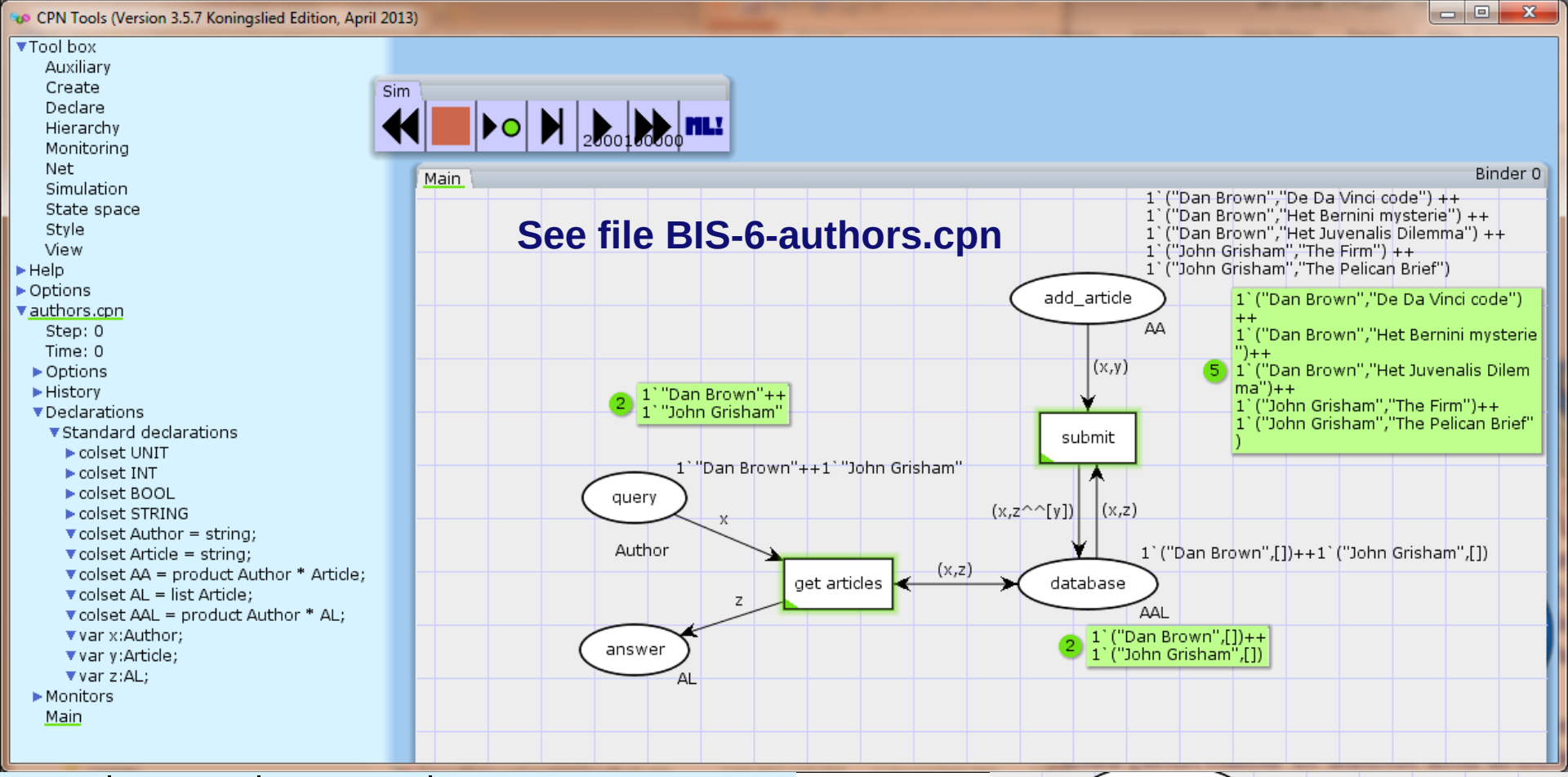
Team versus Player token

```
colset Player = string;
colset Team = list Player;
var p: Player;
var t:Team;
```

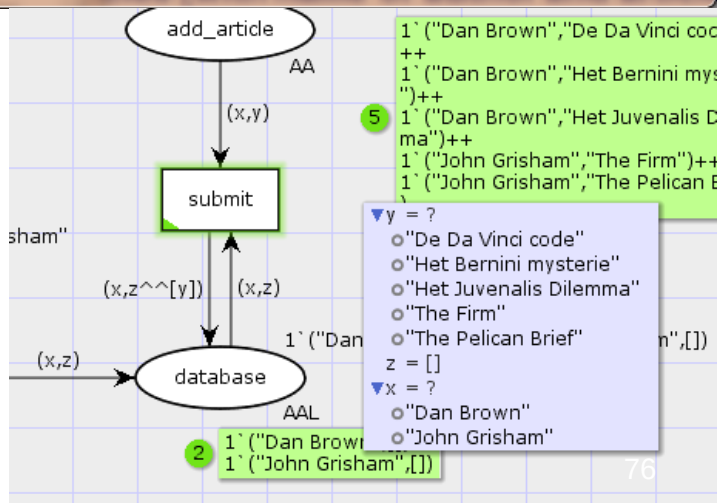


Exercise: Article database

- Consider a database system where authors can submit articles. The articles are stored in such a way that it is possible to get a sequential list of articles per author. The list is ordered in such a way that the oldest articles appear first.
- Note that the system should support two actions: submit articles (with name of author and article) and get articles of a given author.
- We assume that each article has a single author and that only authors already registered in the database can submit articles.
- Model this in terms of a CPN model.



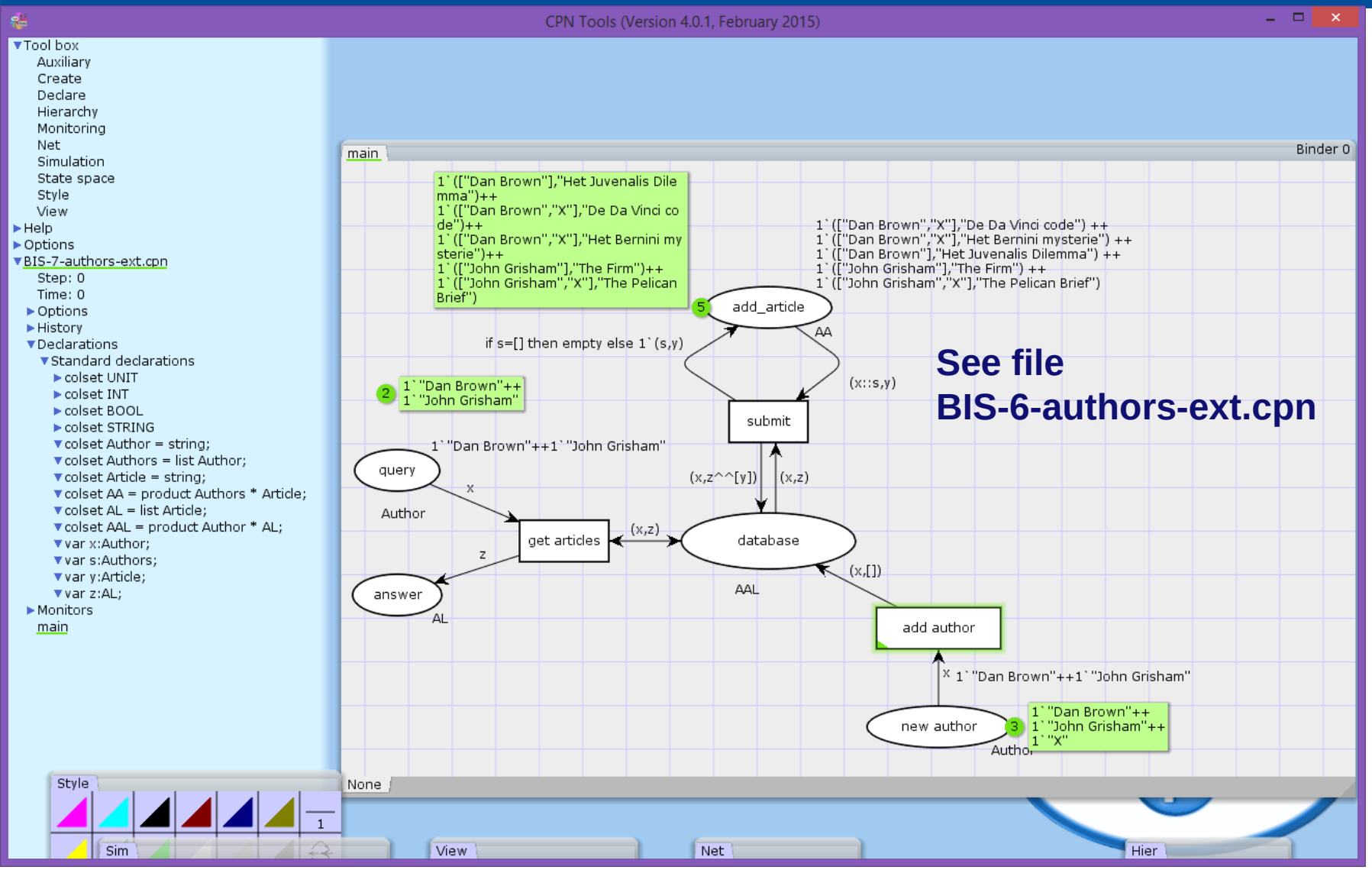
- colset Author = string;
- colset Article = string;
- colset AA = product Author * Article;
- colset AL = list Article;
- colset AAL = product Author * AL;
- var x:Author;
- var y:Article;
- var z:AL;



Exercise (2)

- Extend the CPN model such that
  1. one can add authors to the database
  2. each article can be added only if/when all authors are added to the database

Same DB type, new author information, new submit action



See file BIS-6-authors-ext.cpn