CPN
A concrete language for high-level Petri nets

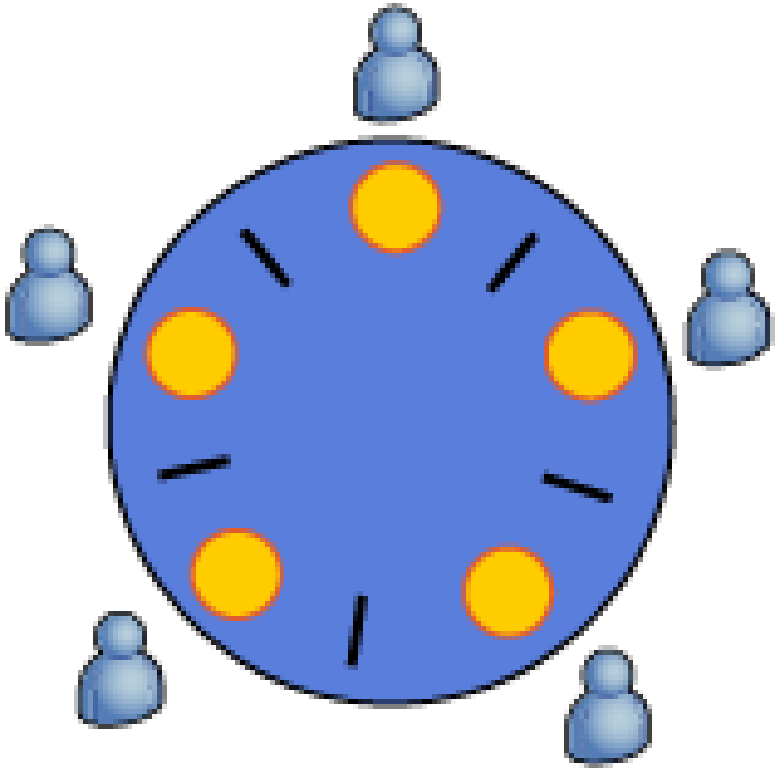
prof.dr.ir. Wil van der Aalst

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Last week ..

5 philosophers



CPN (Colored Petri nets)

- CPN is the language developed by Kurt Jensen et al.
- CPN supports the extensions with time, color and hierarchy.
- CPN is based on standard ML.
- CPN is supported by Design/CPN and CPN Tools.
- In 2010, the support and further development of CPN Tools moved from Aarhus University (Denmark) to TU/e.
- Version 3 was the first version released by TU/e.
- For more information: http://cpntools.org

Values and types

- Syntax is needed to type places and give values (colors) to tokens.
- Adopted from Standard ML

Outline:

- Basic types: int, string, bool, (real), and unit.
- Type constructors: with, product, record, list.
- Defining constants.

Basic types

- Integers (int), e.g., 5, 34234, ~32423.
- Reals (real), e.g., 34.34, ~23.0, 7e3, 4e~2.
- Strings (string), e.g., "Hallo", "28-02-2003".
- Booleans (bool): true and false.
- unit: type with just one value ()

- ~32423 means -32423
- ~23.0 means –23, 7e3 means 7000.0, and 4e~2 means 0.04
- unit is used to represent black (i.e., uncolored) tokens
- Reals are supported in ML but cannot be used as a color set because equality is undefined and hence bindings cannot be calculated

Basic operators

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

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 true
- 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

Color set declarations

- 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,U may be used in other color set declarations.

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

Creating subtypes using the "with" clause

- color Age = int with 0..130;
- color Temp = int with ~30..40;
- color Alphabet = string with "a".."z";
- color YN = bool with (no,yes);
- color BlackToken = unit with null;

Creating new types using the "with" clause

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

Creating new types using product, record, and list constructors

- color Coordinates = product I * I * I;
- color HumanAge = product Human * Age;
- color CoordinatesR = record x:I * y:I * z:I;
- color CD = record artists:S * title:S * noftracks:I;
- color Names = list S;
- color ListOfColors = list ThreeColors;

Possible values (colors)

- **Coordinates: (1,2,3), (~4,66,0), ...**
- **HumanAge: (man,50), (child,3), ...**
- **CoordinatesR: {x=1, y=2, z=3}, {x=~4, y=66, z=0}, {y=2, x=1, z=3}, ...**
- **CD: {artists="Havenzangers", title="La La", noftracks=10}, ...**
- **Names: ["John", "Liza", "Paul"], [ ], ...**
- **ListOfColors = [Green], [Red, Yellow], ...**

Note the difference between products and records.

Example

- 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 = record d:Driver * r:LapTimes;
- color Race = list DriverResults;

Example (2)

A possible color of type Race is:

```
[{d="Jos Verstappen",
   r=[(1,31000),(2,33400),(3,32800)]},
{d="Michael Schumacher",
   r=[(1,32200),(2,31600),(3,30200),(4,29600)]},
{d="Rubens Barrichello",
   r=[(1,34500),(2,32600),(3,37200),(4,42600)]}]
```

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 #x{x=1,y=2} evaluates to 1

Constants

- **It is possible to define constants, e.g.,**
  - **val jv = "Jos Verstappen" : Driver;**
  - **val lap1 = 1 : Lap;**
  - **val start = 0 : Time;**
  - **val seven = 7 : int;**

Example

- **Determine the value of constant Monaco:**
  - **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),(3,37200), (4,42600)]}:DriverResults;**
  - **val Monaco = jos :: ([michael]^^[rubens]) : Race;**

Exercise

- **Determine the value of the following constants:**
  - **val e1 = r1jos::[ ];**
  - **val e2 = #d(michael);**
  - **val e3 = (#r(jos))^^(#r(rubens));**

So what?

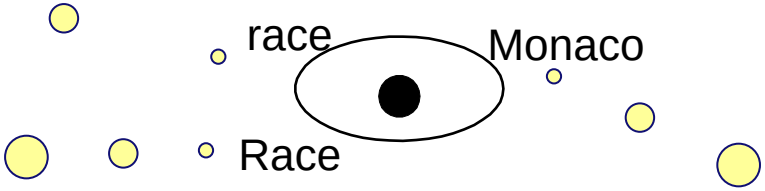
We can now type and initialize places!

declarations

```
color Driver = string;
color Lap = int with 1..80;
color Time = real with 0.0..1000.0;
color LapTime = product Lap * Time;
color LapTimes = list LapTime;
color DriverResults = record d:Driver * r:Laptimes;
color Race = List DriverResults;
val Monaco = [{d="Jos Verstappen", r=[(1,31000),(2,33400),(3,32800)]},
{d="Michael Schumacher", r=[(1,32200),(2,31600),(3,30200),(4,29600)]},
{d="Rubens Barrichello", r=[(1,34500),(2,32600),(3,37200),(4,42600)]}];
```

name of place

type of place

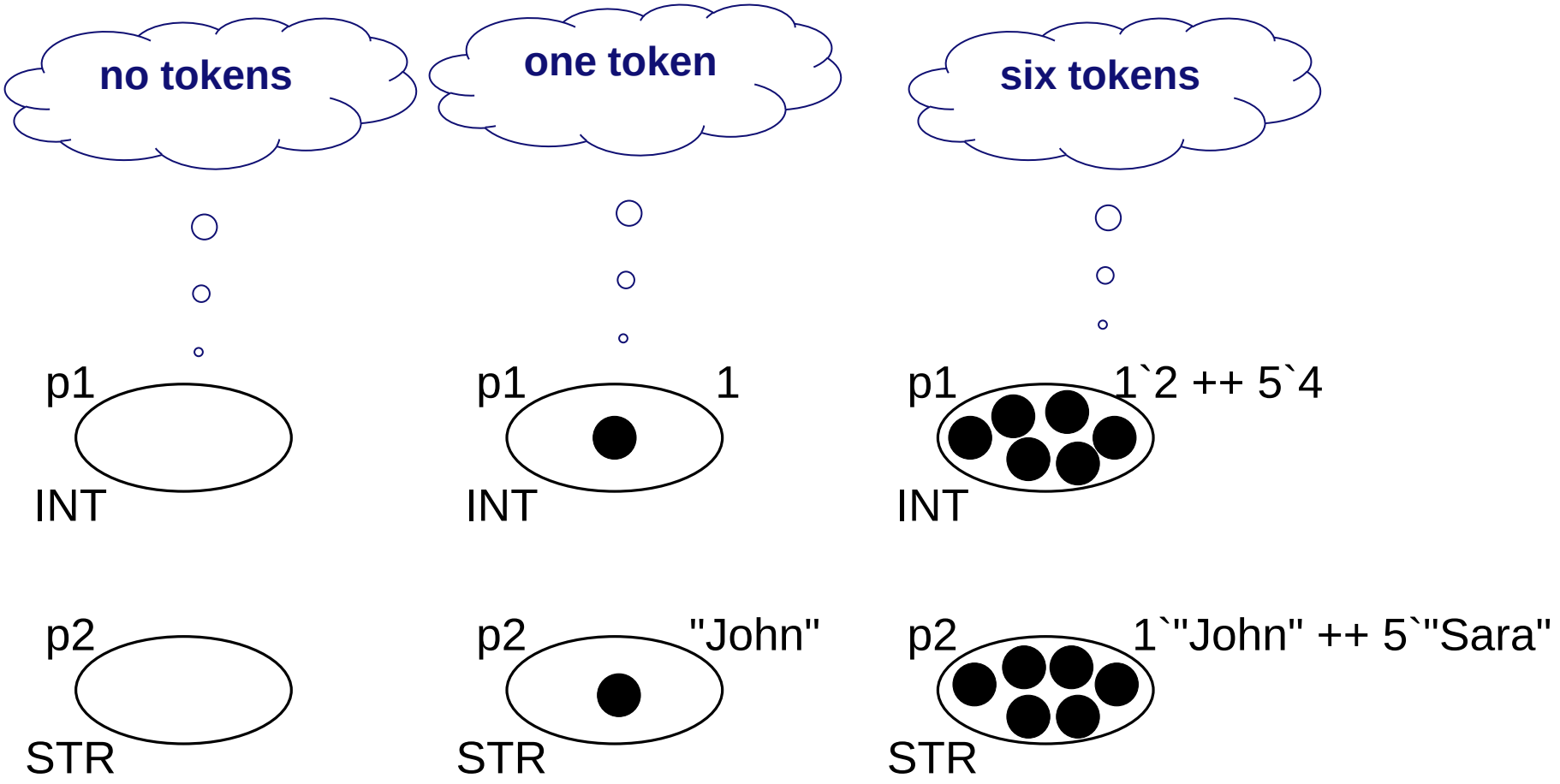


initial marking

Multi-sets

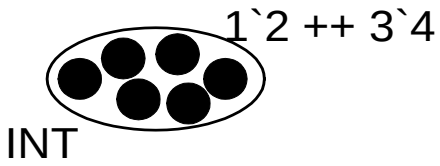
- **To initialize places with multiple tokens but also for various other purposes we need multi-sets also referred to as bags.**
- **In CPN multi-sets are denoted using the following notation: x1`v1 ++ x2`v2 ++ ... ++ xn`vn**
  where v1 is a value and x1 the number of times this element appears in the multi-set, etc.
- **E.g., 4`"Red" ++ 2`"Green" ++ 1`"Blue" is a multi-set containing 7 elements**

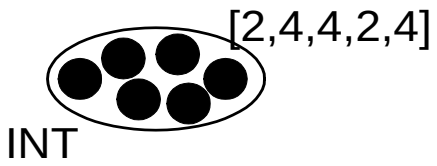
Initialization expressions

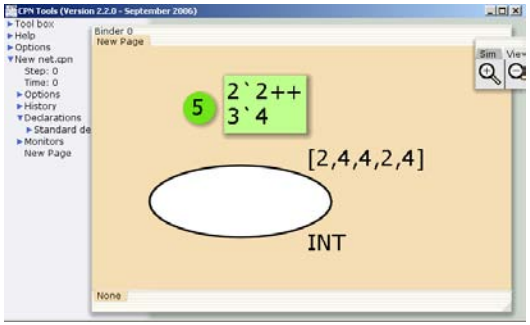


Trick

- **Multi-sets are implemented as lists, i.e., 4`"Red" ++ 2`"Green" ++ 1`"Blue" can also be written as e.g. ["Red","Red","Red","Red","Green","Green", "Blue"].**
- **This is useful when using list functions.**



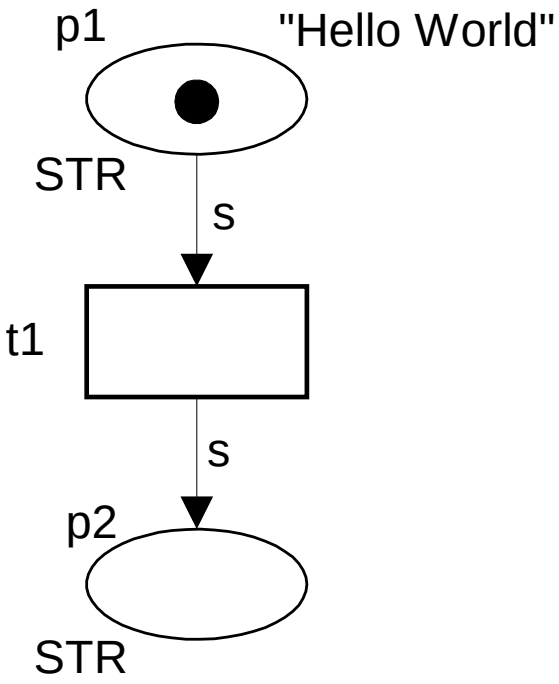




Arc inscriptions

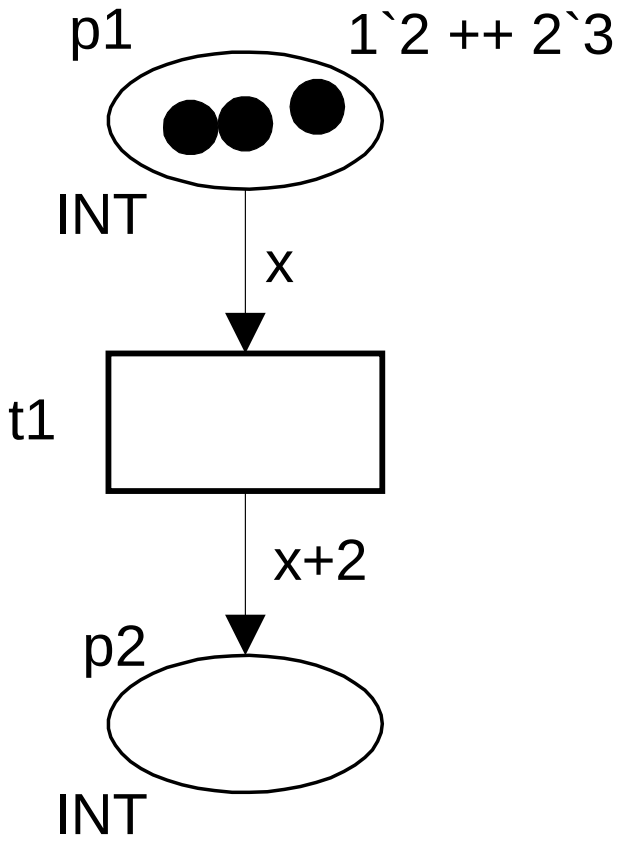
- **Arc inscriptions are used to define input-output behavior.**
- **Arc inscriptions may use variables.**
- **Variables are typed and need to be declared**

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



Example

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



- **Give final marking.**

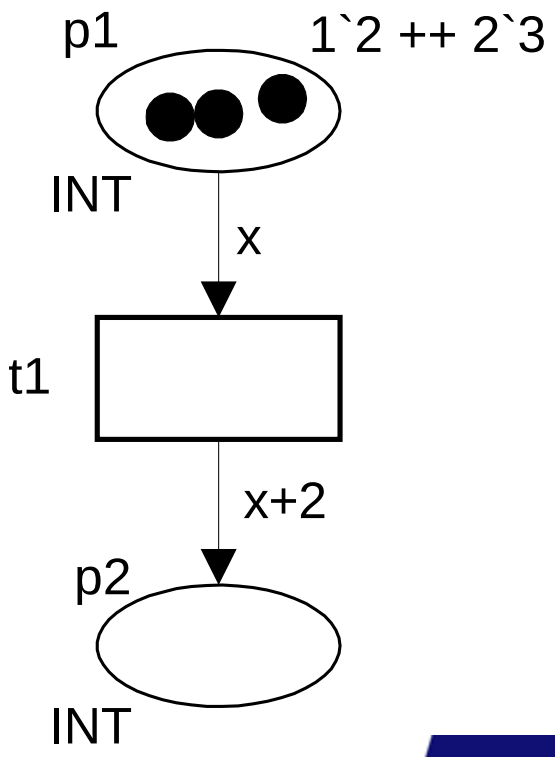
Binding

- Given a transition t with variables x1, x2, ..., xn on its input and output arcs, a **binding** of t allocates a concrete value to each of these variables. These values should be of the corresponding type.
- A **binding is enabled** if there are tokens matching the values of the arc inscriptions.
- If a binding is enabled, it can **occur**, i.e., the transition fires while consuming and producing the corresponding tokens.
- The pair (t1,<x1=v1,x2=v2, ...xn=vn>) is called a **binding element**.

Example

- **Two binding elements: (t1,<x=2>) and (t1,<x=3>)**

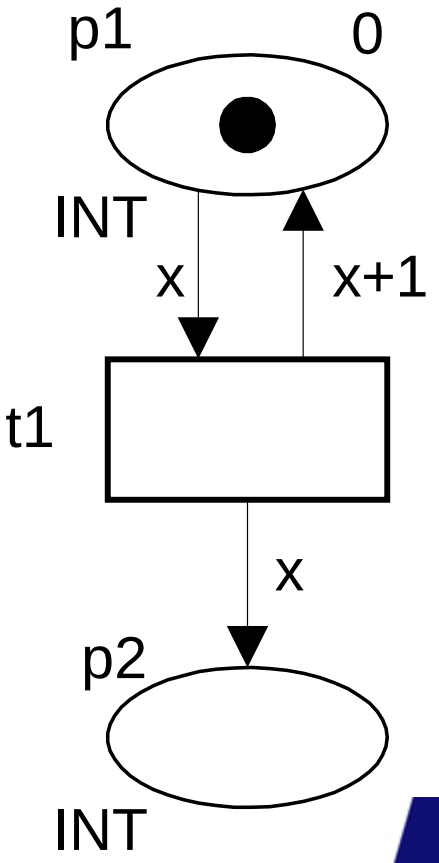
```
color INT = int;
var x:INT;
```



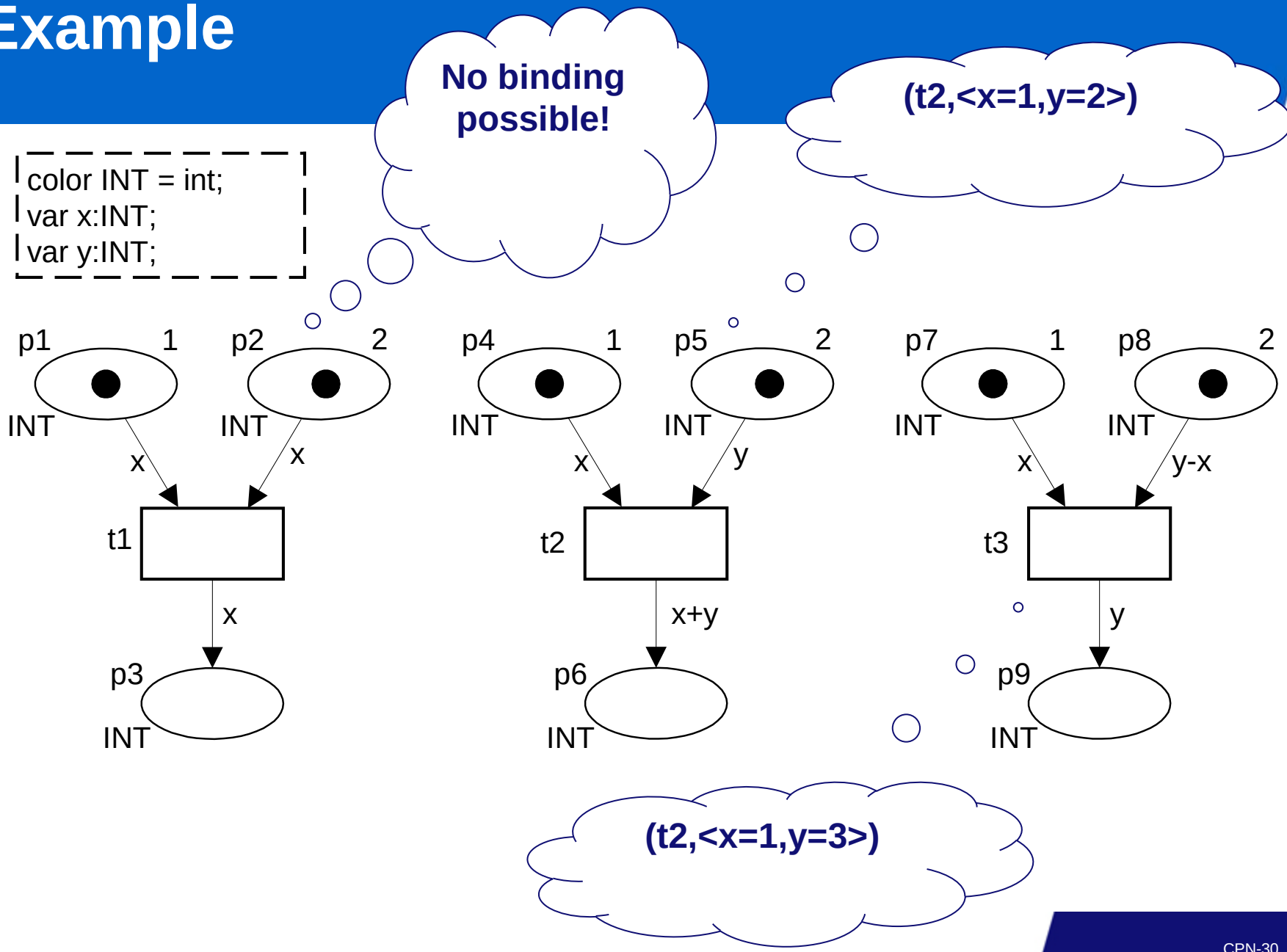
Example

- **Binding element (t1,<x=0>). After it occurred (t1,<x=1>), etc.**

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



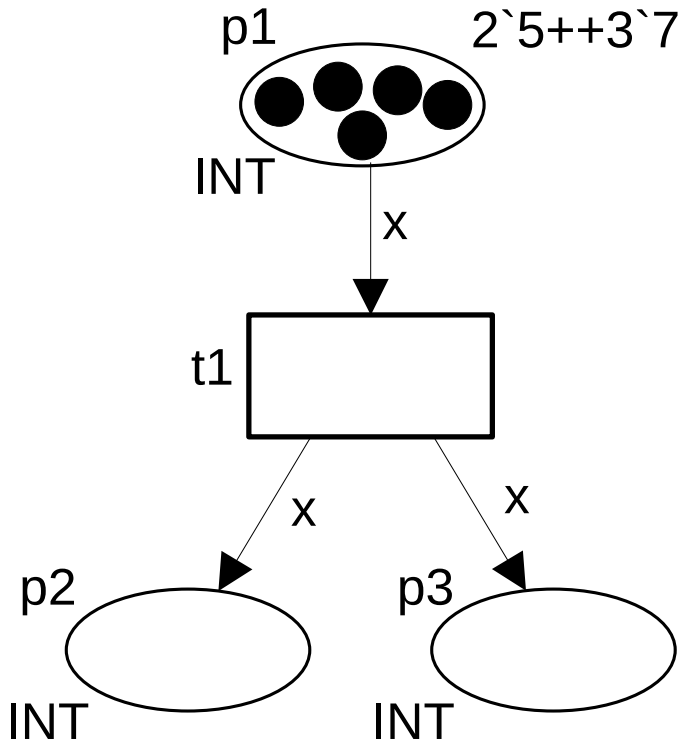
Example


No binding possible!
(t2,<x=1,y=2>)
color INT = int;
var x:INT;
var y:INT;
p1
1
INT
p2
2
INT
x
x
t1
x
p3
INT
p4
1
INT
p5
2
INT
x
y
t2
x+y
p6
INT
p7
1
INT
p8
2
INT
x
y-x
t3
y
p9
INT
(t2,<x=1,y=3>)

Exercise

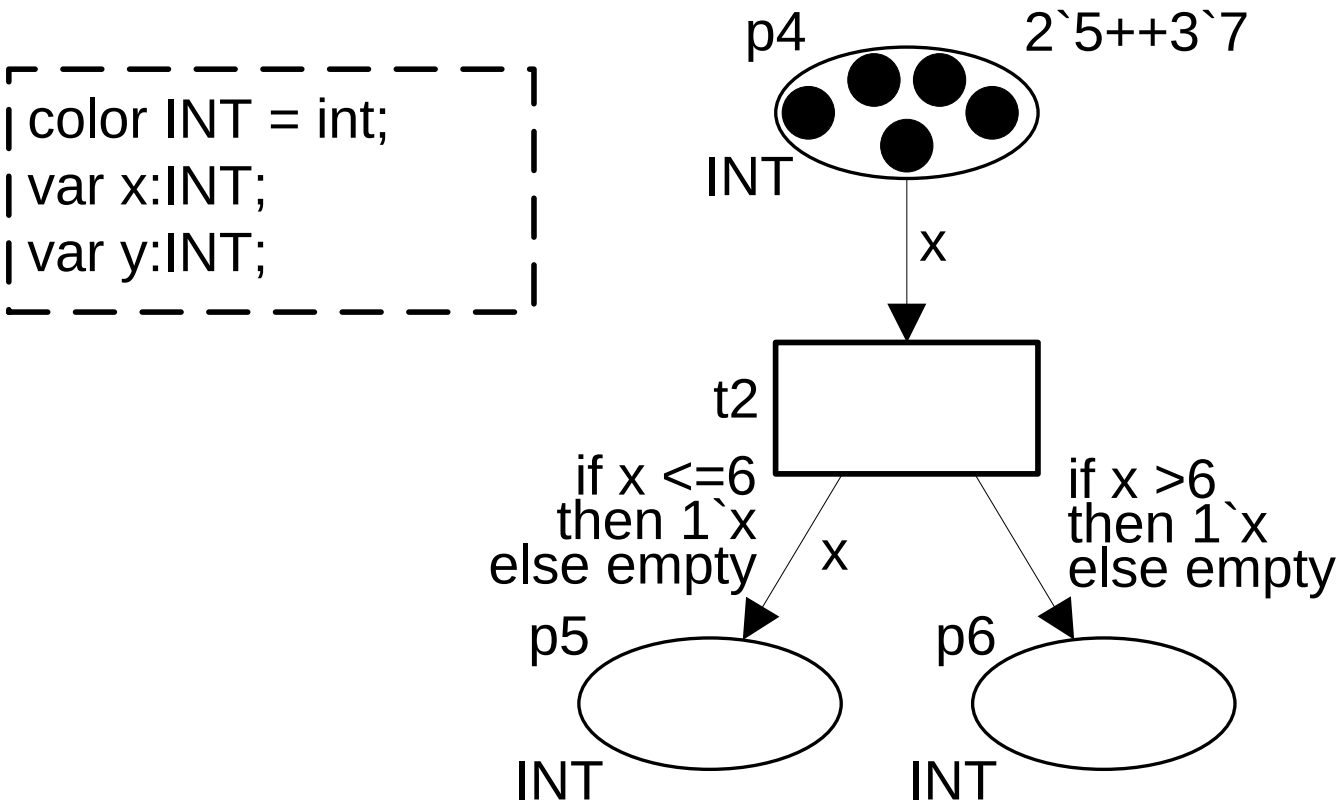
- **Give all possible binding elements and final markings**

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



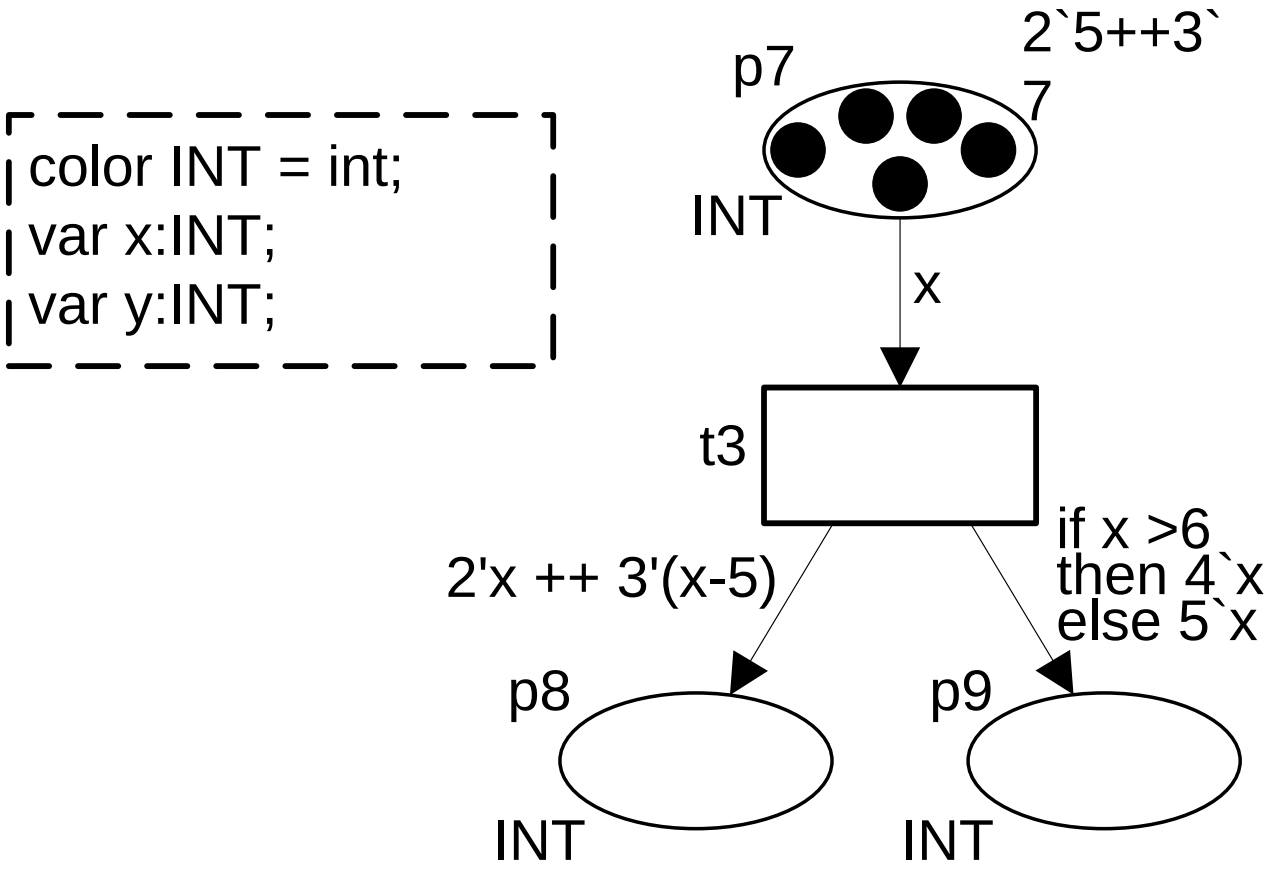
Exercise

- **Give all possible binding elements and final markings**



Exercise

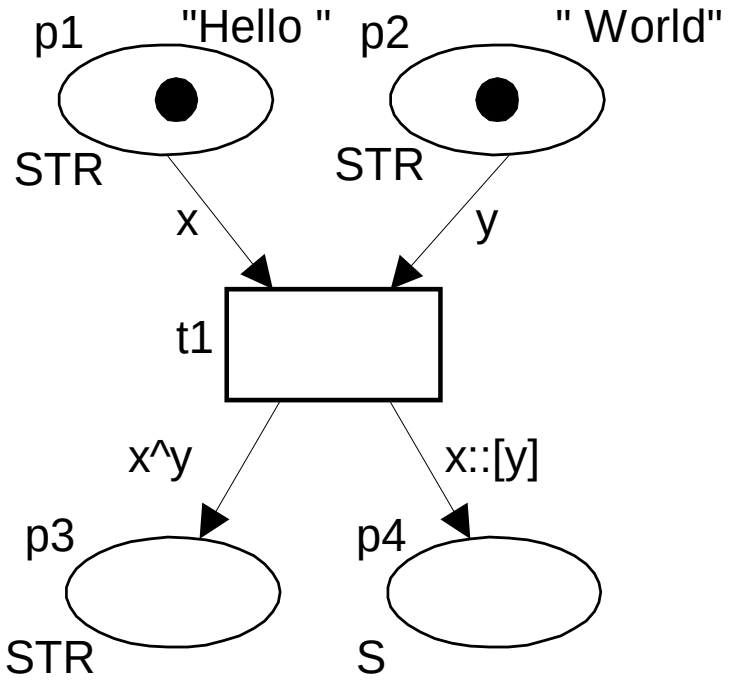
- **Give all possible binding elements and final markings**



Exercise

- **Give all possible binding elements and a final marking**

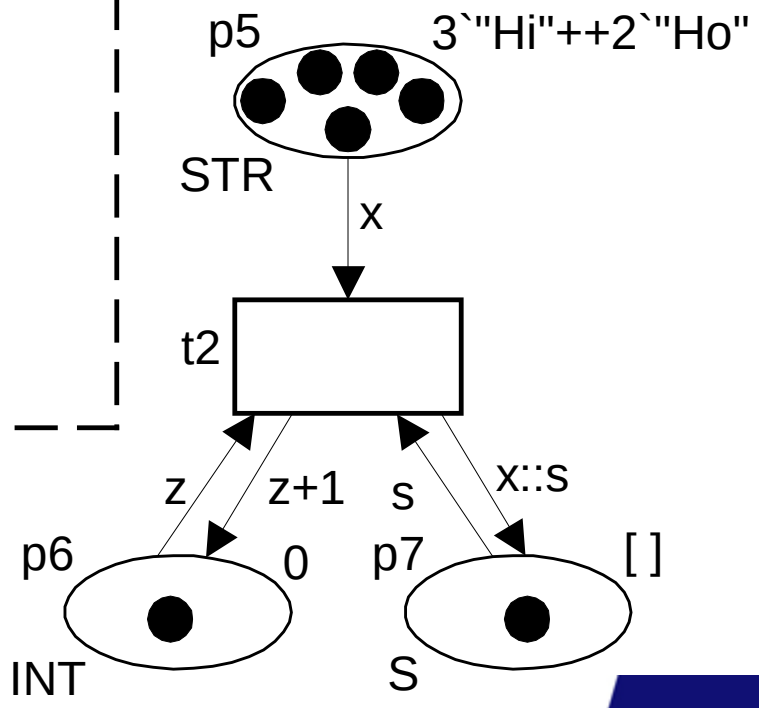
```
color STR = string;
var x:STR;
var y:STR;
color INT = int;
var z:INT;
color S = list STR;
var s:S;
color R = record a:STR * b:S;
var r:R;
```



Exercise

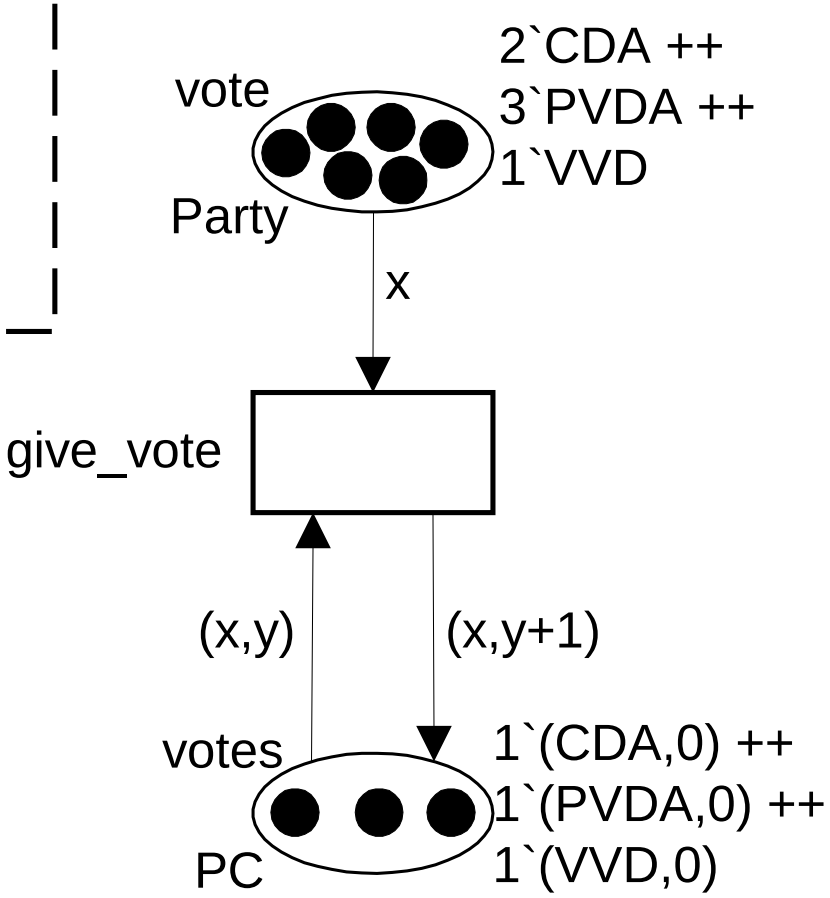
- **Give all possible binding elements and a final marking**

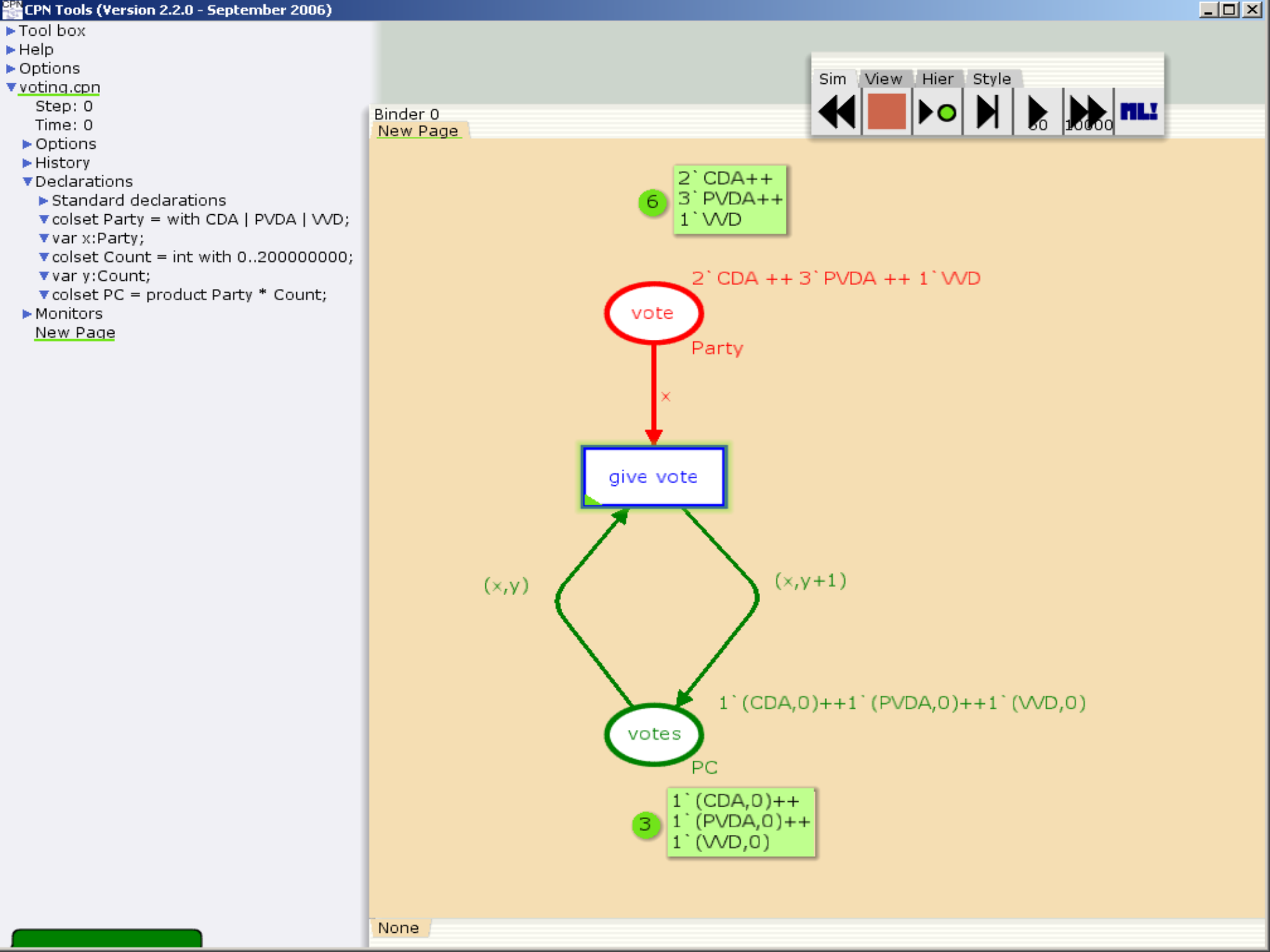
```
color STR = string;
var x:STR;
var y:STR;
color INT = int;
var z:INT;
color S = list STR;
var s:S;
color R = record a:STR * b:S;
var r:R;
```



Example: Voting

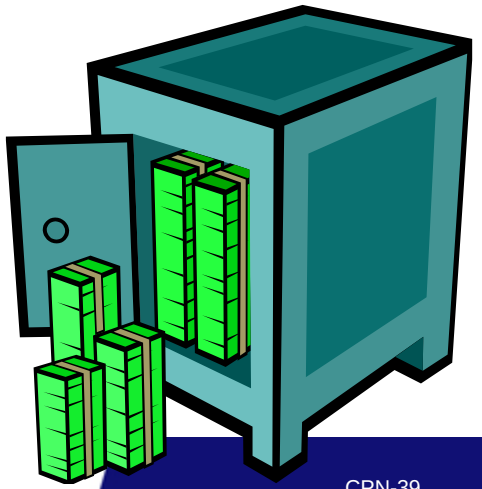
```
color Party = with CDA | PVDA | VVD;
var x:Party;
color Count = int with 0..200000000;
var y:Count;
color PC = product Party * Count;
```




CPN Tools (Version 2.2.0 - September 2006)
Tool box
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
New Page
Sim
View
Hier
Style
Binder 0
New Page
2`CDA++
3`PVDA++
1`VVD
6
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
1`(CDA,0)++
1`(PVDA,0)++
1`(VVD,0)
3
None

Exercise: Bank

- **Consider a simple banking system. There are 1000 accounts numbered from 1 to 1000. People can deposit or withdraw money. Only amounts between 1 EURO and 5000 EURO can be deposited or withdrawn. The account may have a negative balance.**
- **Model this in terms of a CPN model.**



Exercise: Article database

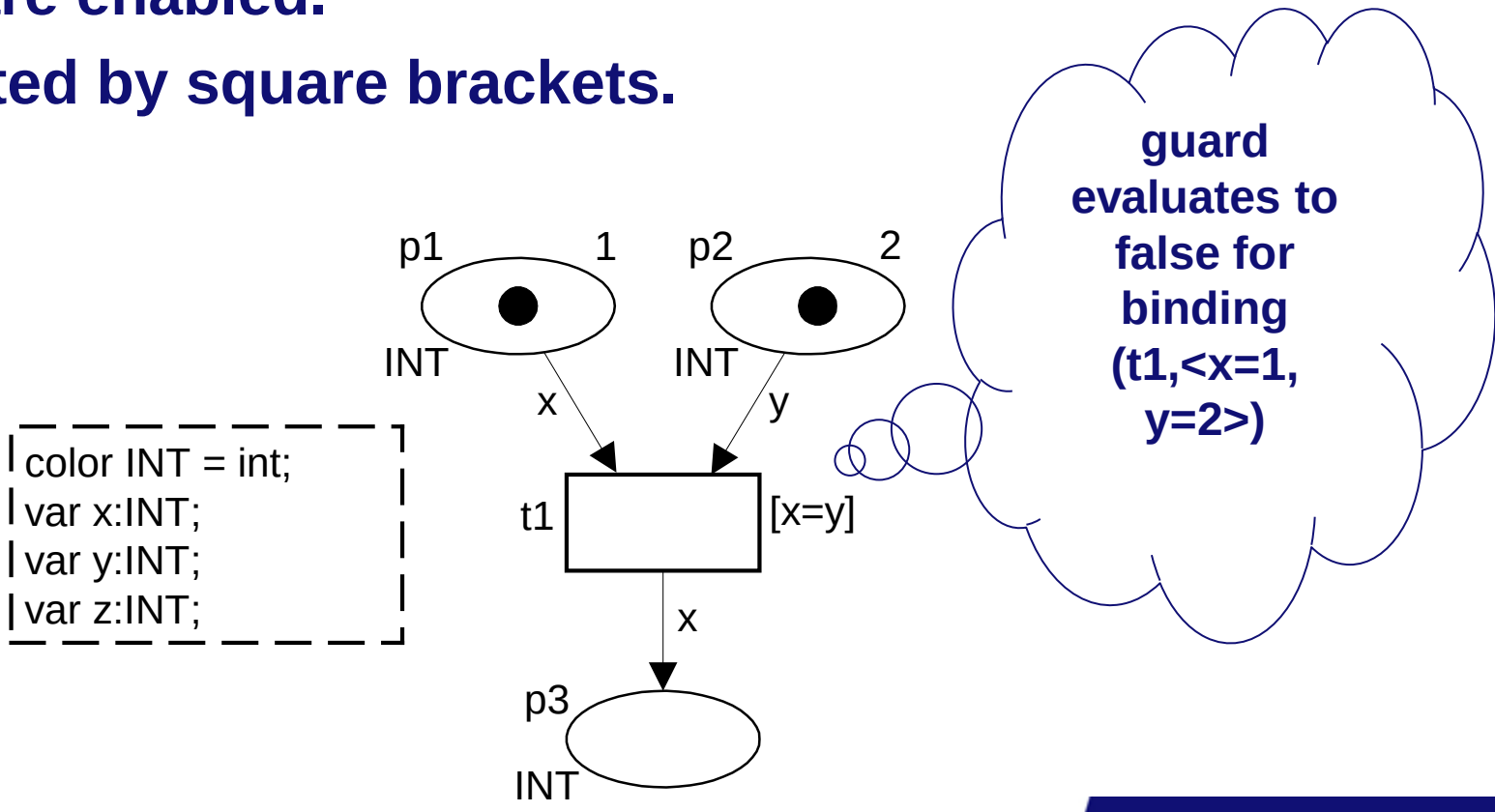
- Consider a database system where authors can submit articles. The articles are stored in such a way that it is possible to get a sequential list of articles per author. The list is ordered in such a way that the oldest articles appear first.
- Note that the system should support two actions: submit 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.

Exercise (2)

- **Extend the CPN model such that each article can have multiple authors, i.e., the article is stored once for each author, and that there is an explicit action to add authors to the database.**

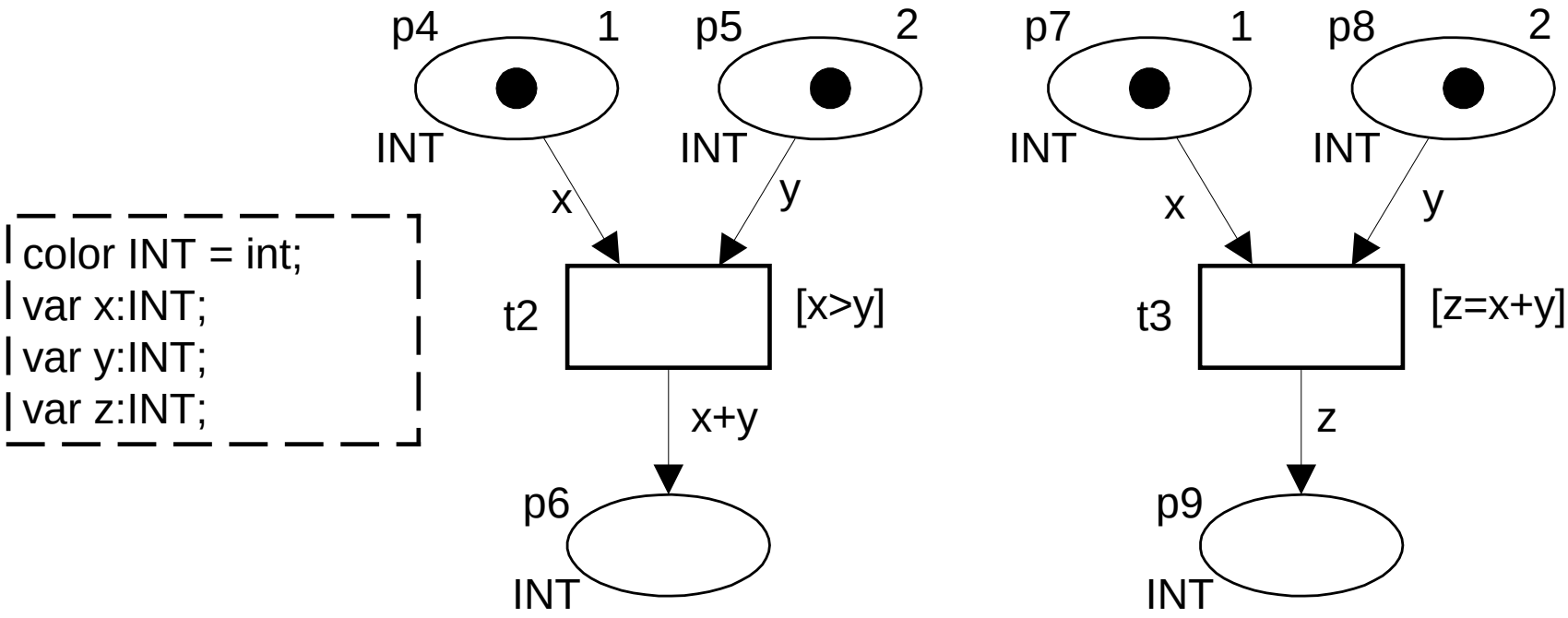
Guard

- **A guard is a Boolean expression attached to a transition. Only binding elements which evaluate to true are enabled.**
- **Denoted by square brackets.**



Example

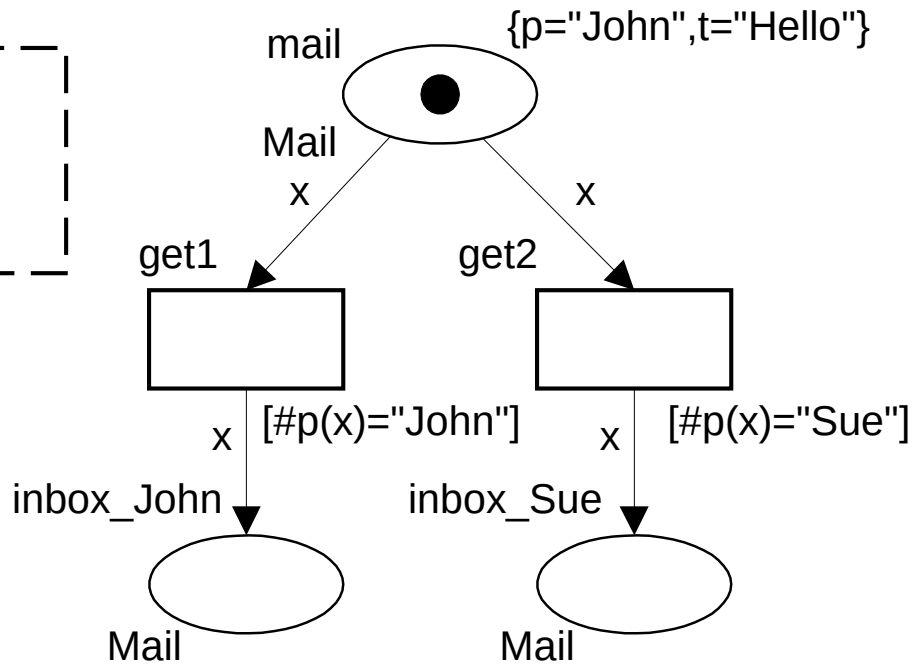
- **Give all enabled binding elements and the final marking**



Exercise

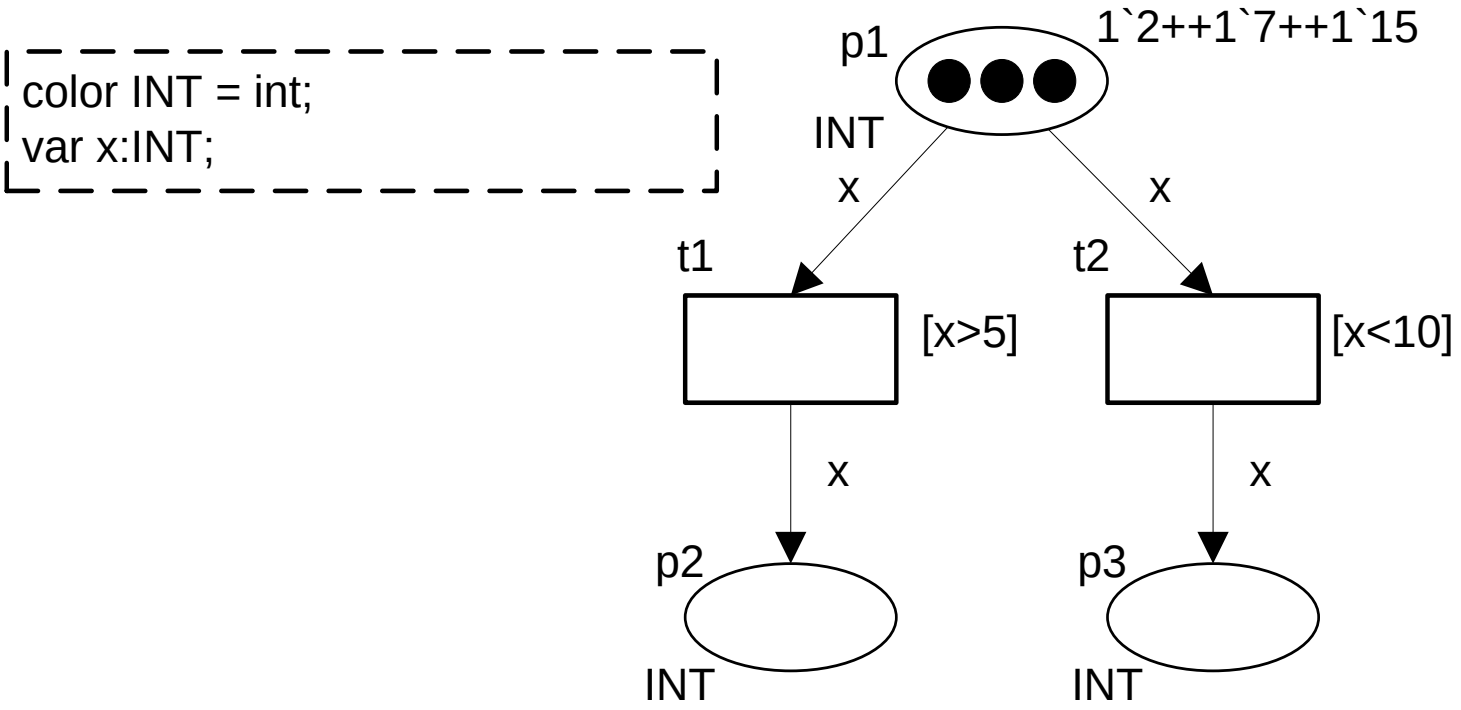
- **Give all enabled binding elements and the final marking**

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



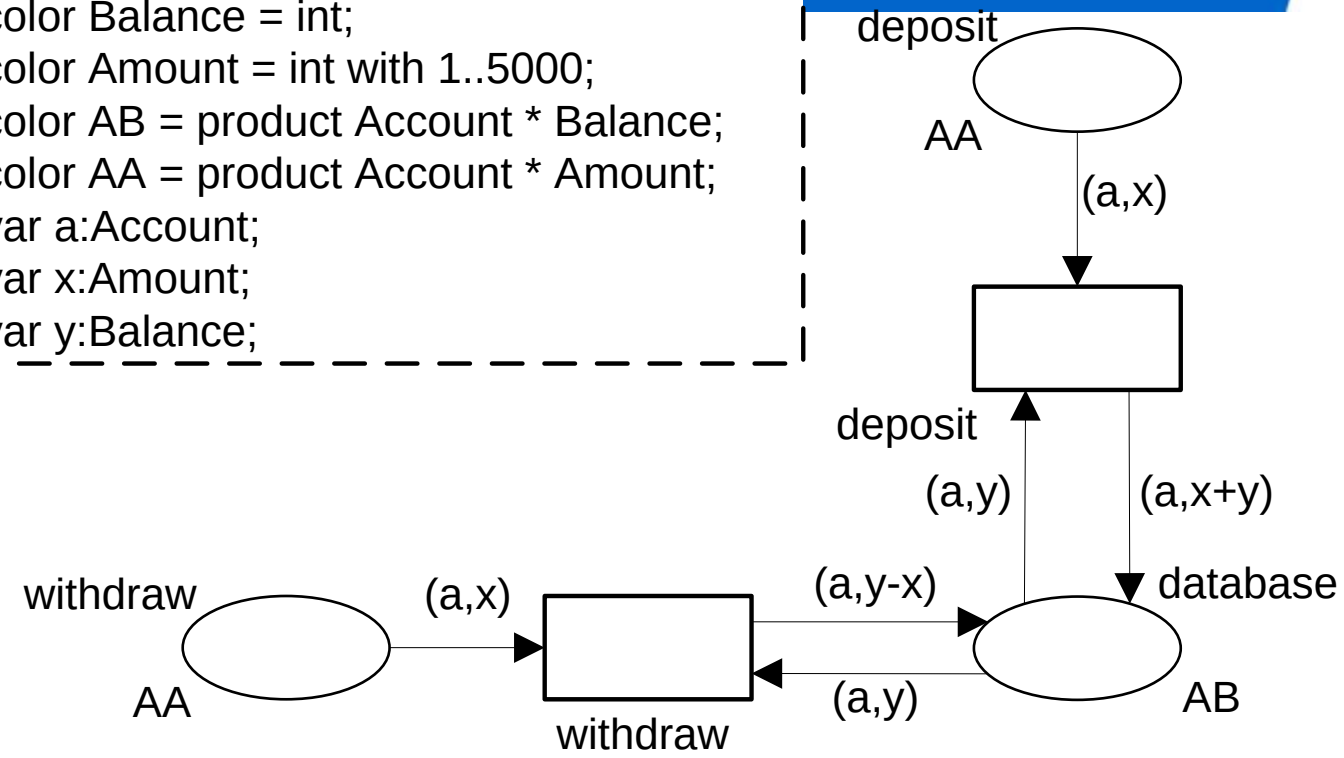
Exercise

- **Give all enabled binding elements and all possible final marking**



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 the balance cannot become negative, i.e., do not accept transactions which lead to a negative balance.**

Function declarations

```
color INT = int;
fun fac(x:INT) = if x>1 then x*fac(x-1) else 1;
fun fib(x:INT) = if x<2 then 1 else fib(x-1) + fib(x-2);
color L = list INT;
fun sum(x:L) = if x=[ ] then 0 else hd(x)+sum(tl(x));
fun odd(x:L) = if x=[ ] then [ ] else hd(x)::(if tl(x)=[ ] then [ ]
   else odd(tl(tl(x))));
```

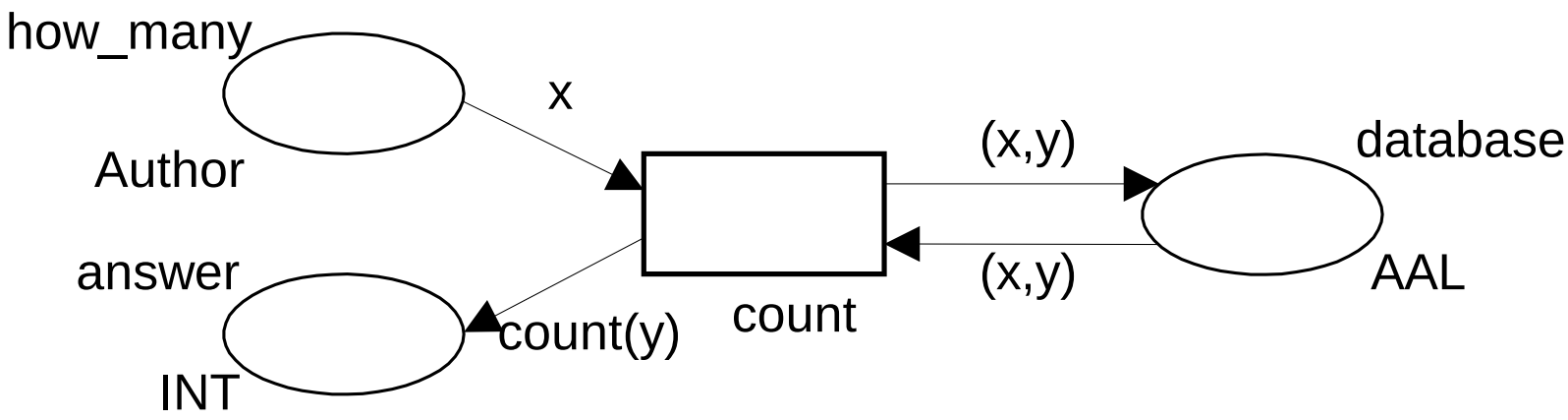
- Calculate fac(fib(3)) and modify odd such that the odd lines are returned in reversed order.

Where to find standard functions?

- **These sheets.**
- **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**
- **www.standardml.org, see for example http://www.standardml.org/Basis/list.html#LIST:SIG:SPEC for list functions, http://www.standardml.org/Basis/integer.html#INTEGER:SIG:SPEC for integer functions, http://www.standardml.org/Basis/string.html#STRING:SIG:SPEC for string functions, etc.**

Example

```
color INT:int;
color Author = string;
color Article = string;
color AL = list Article;
color AAL = product Author * AL;
var x:Author;
var y:AL;
fun count(z:AL) = if z=[] then 0 else 1+tl(z)
```



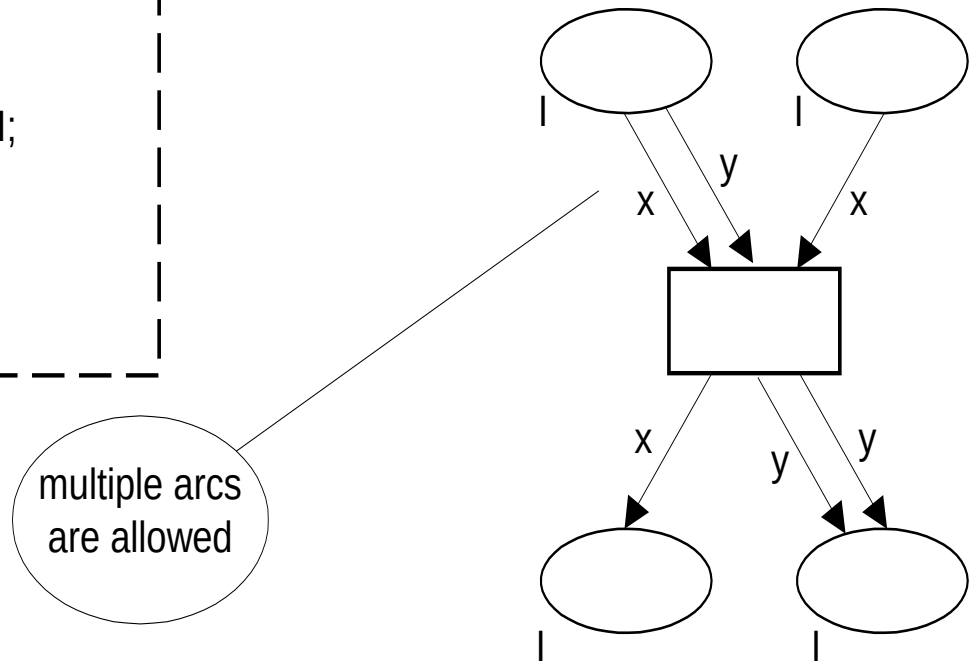
Questions

- **Is it possible to have multiple arcs connecting a place and a transitions?**
- **Is it possible to have multi-sets as arc inscriptions on input arcs?**
- **Is it possible to use constants or other expressions without variables as arc inscriptions?**
- **Is it possible to use records, lists, etc. with variables (e.g., {a=x,b=y} and x::y) in arc inscriptions?**

4x

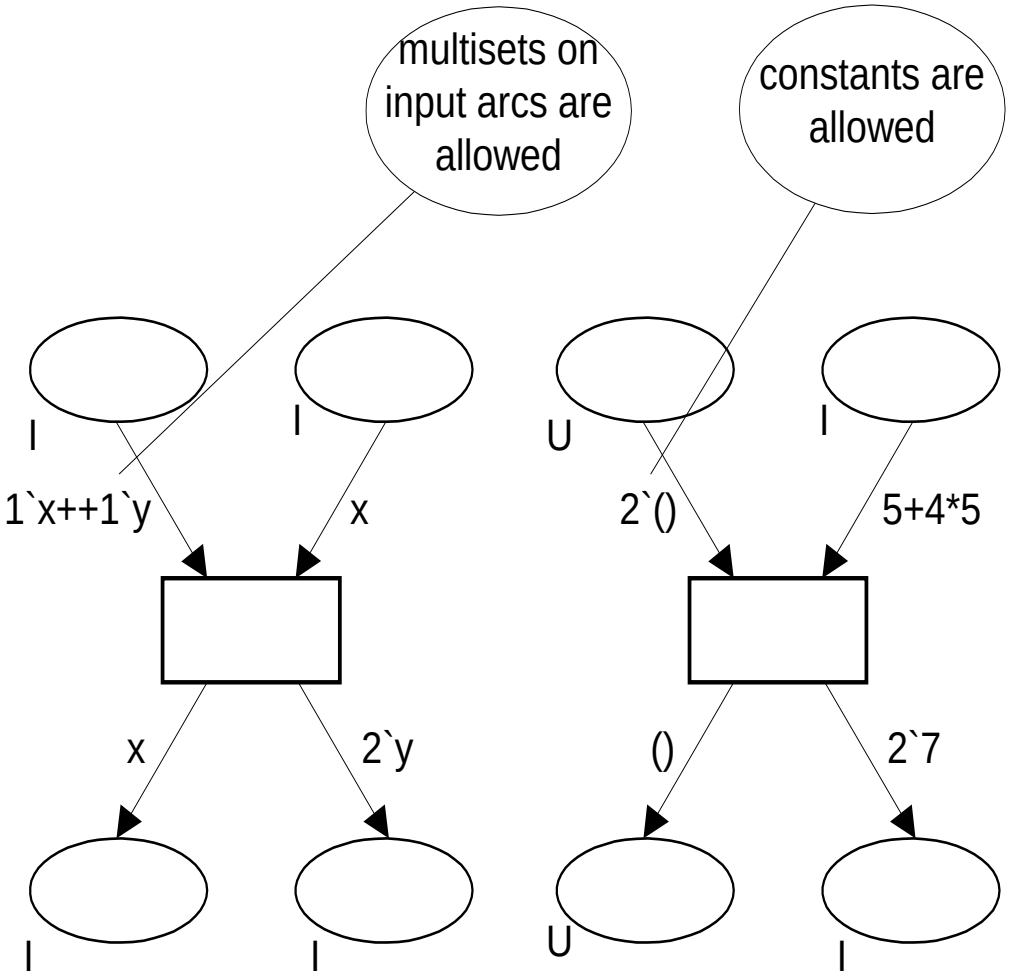
Example: Multiple arcs

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



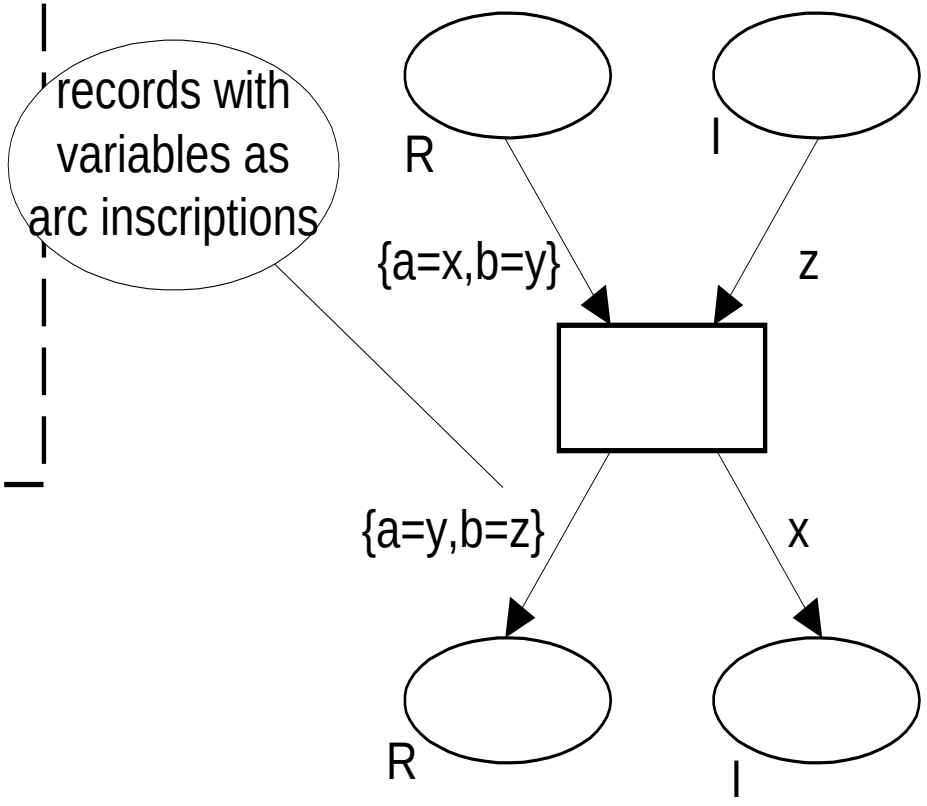
Example: Multi-sets and constants

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



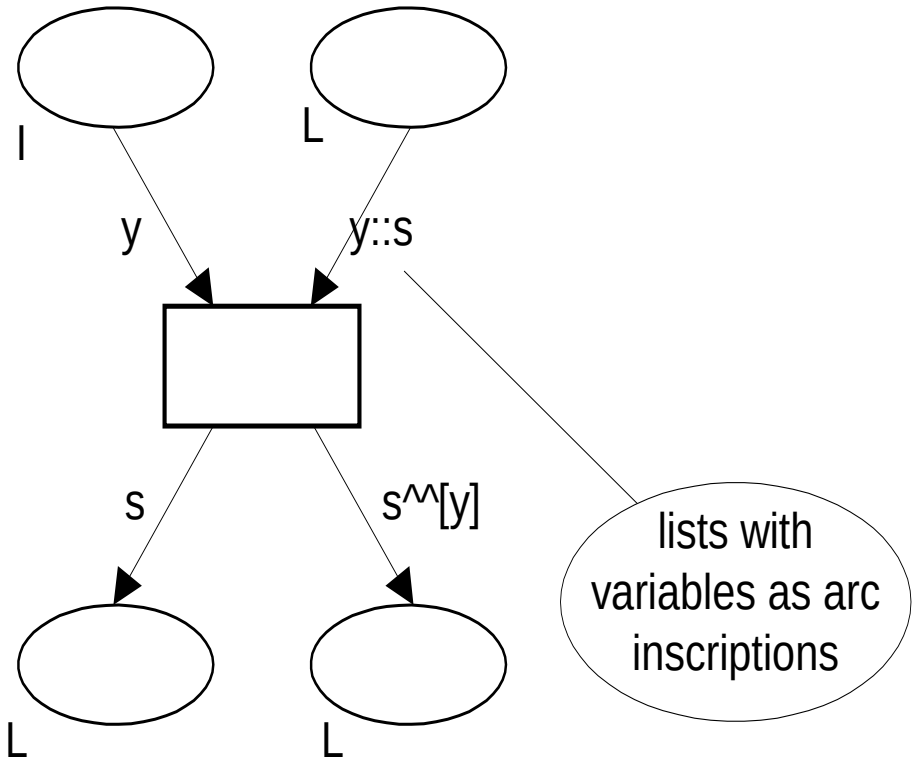
Example: Records

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



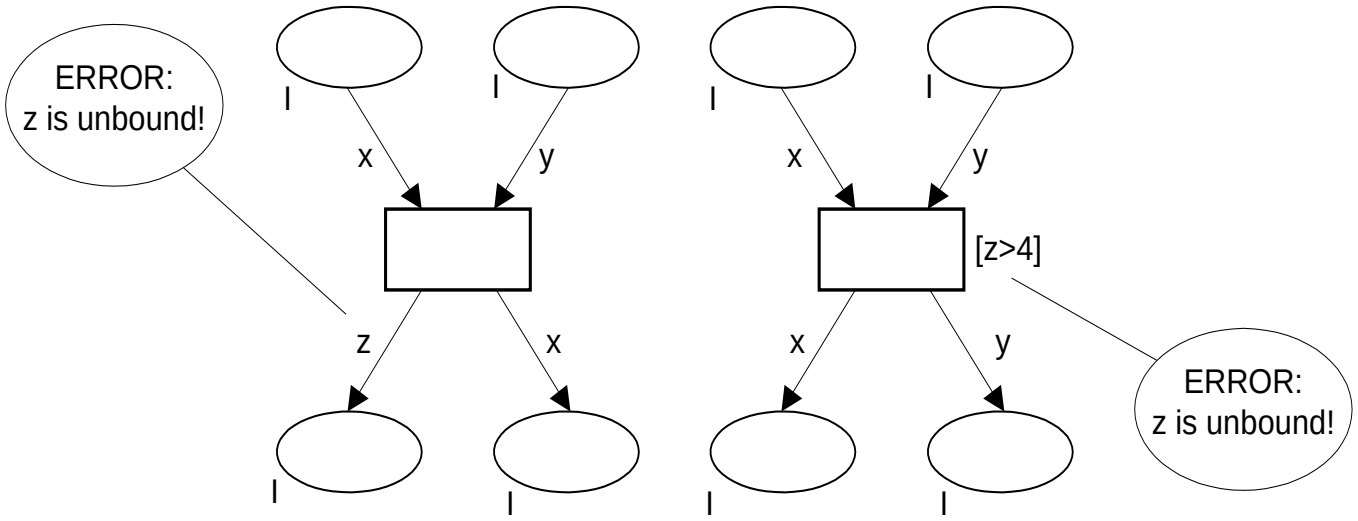
Example: Lists

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

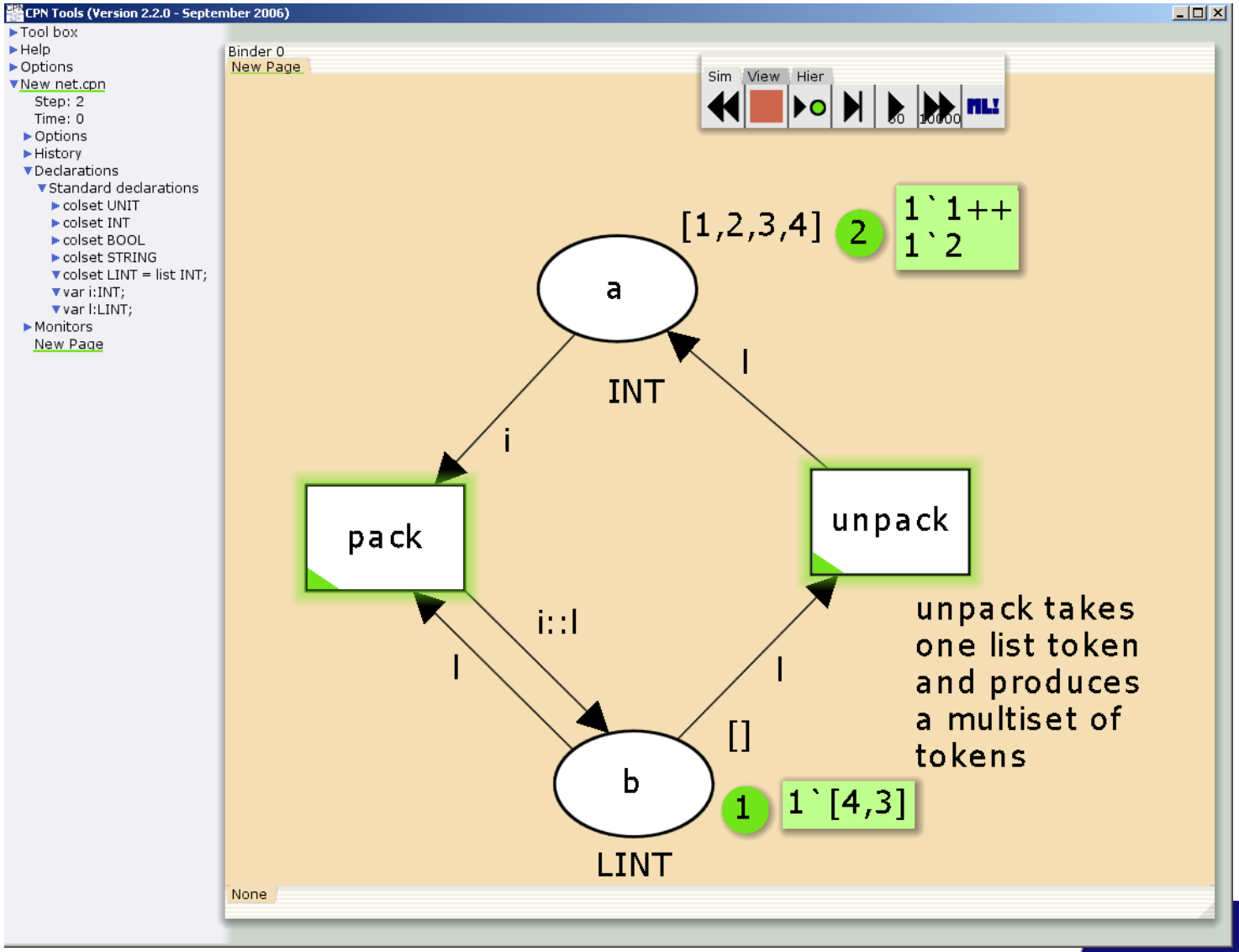


Requirement

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

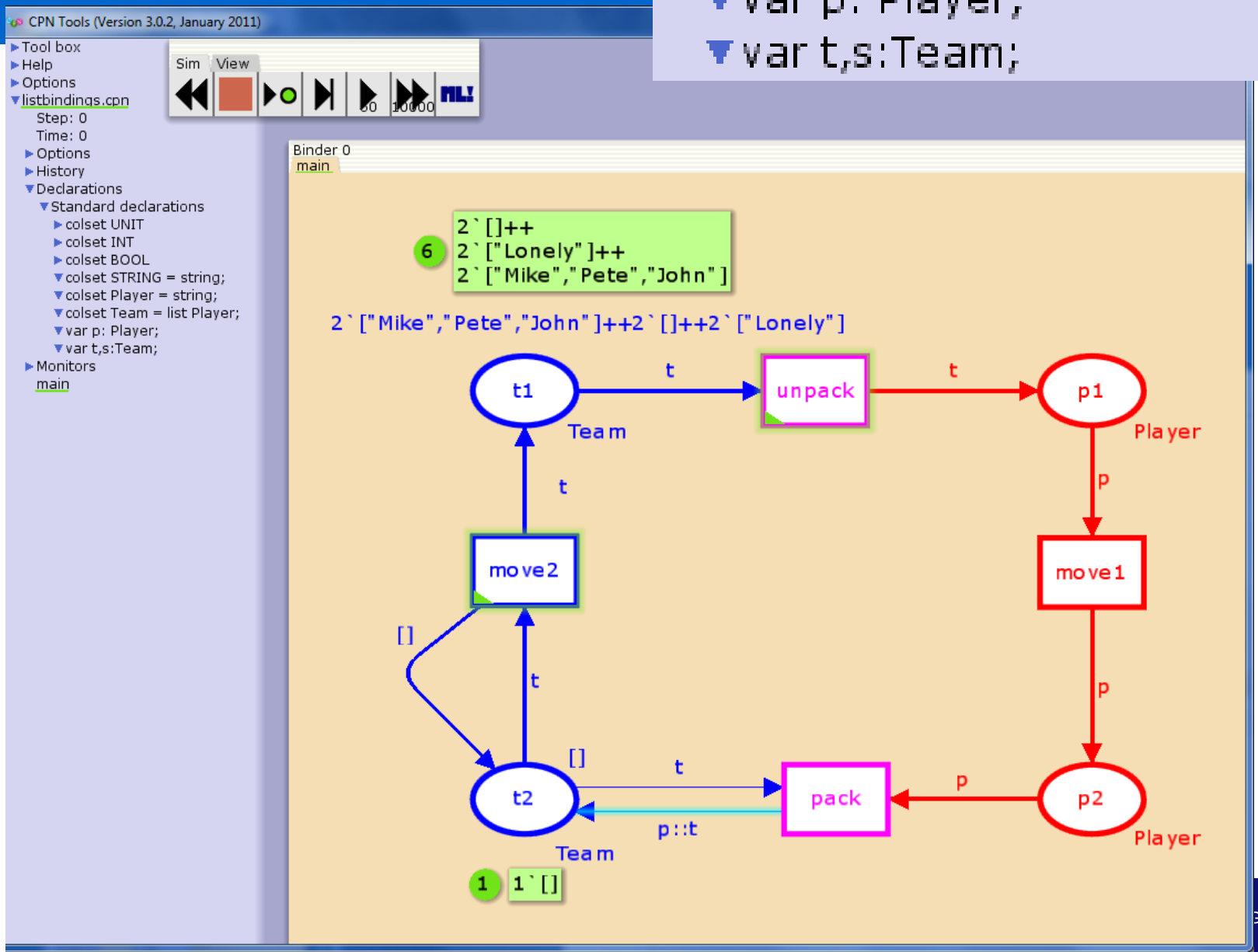


Trick: use lists on arcs to produce/consume multi-sets of tokens

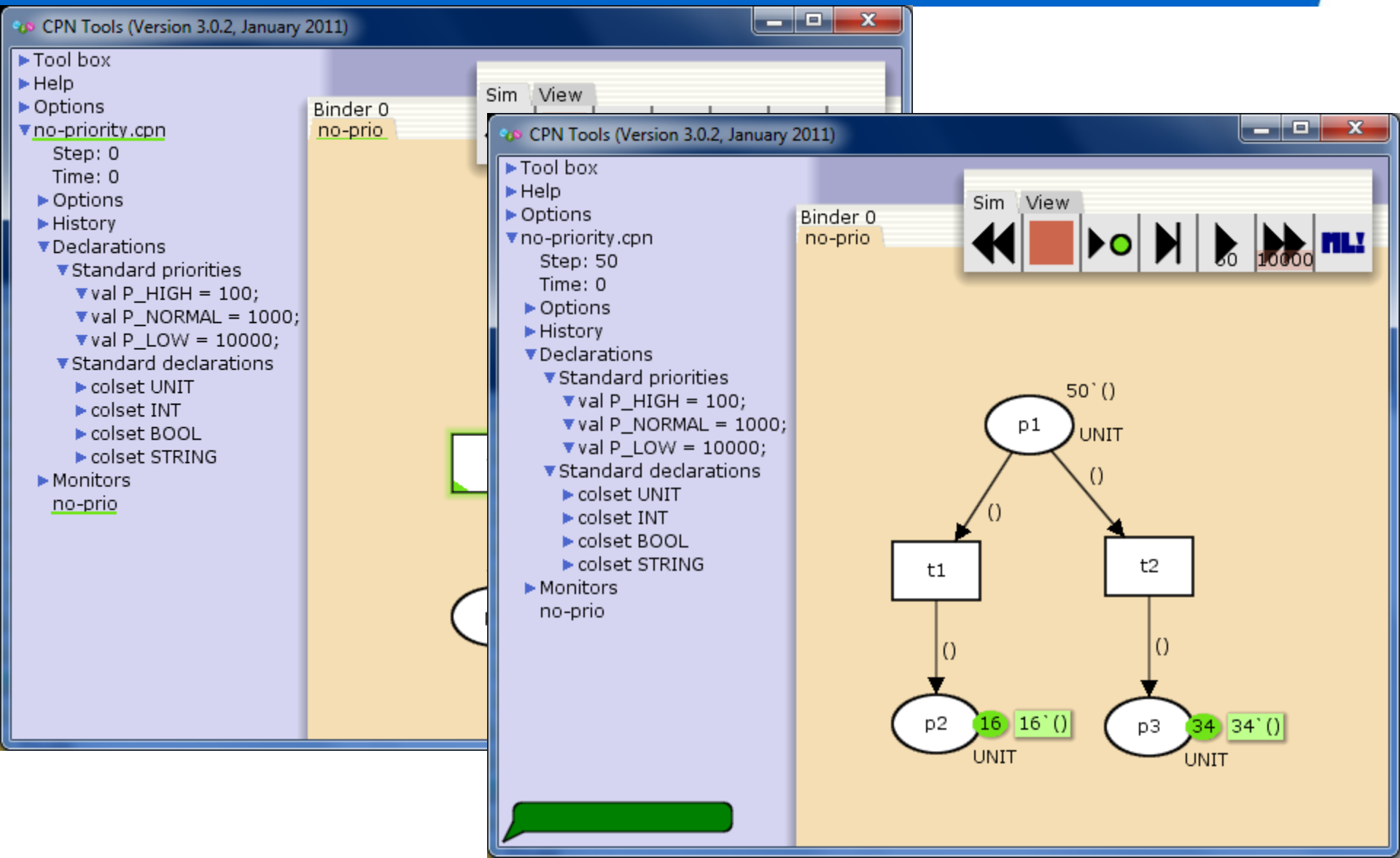


Another example

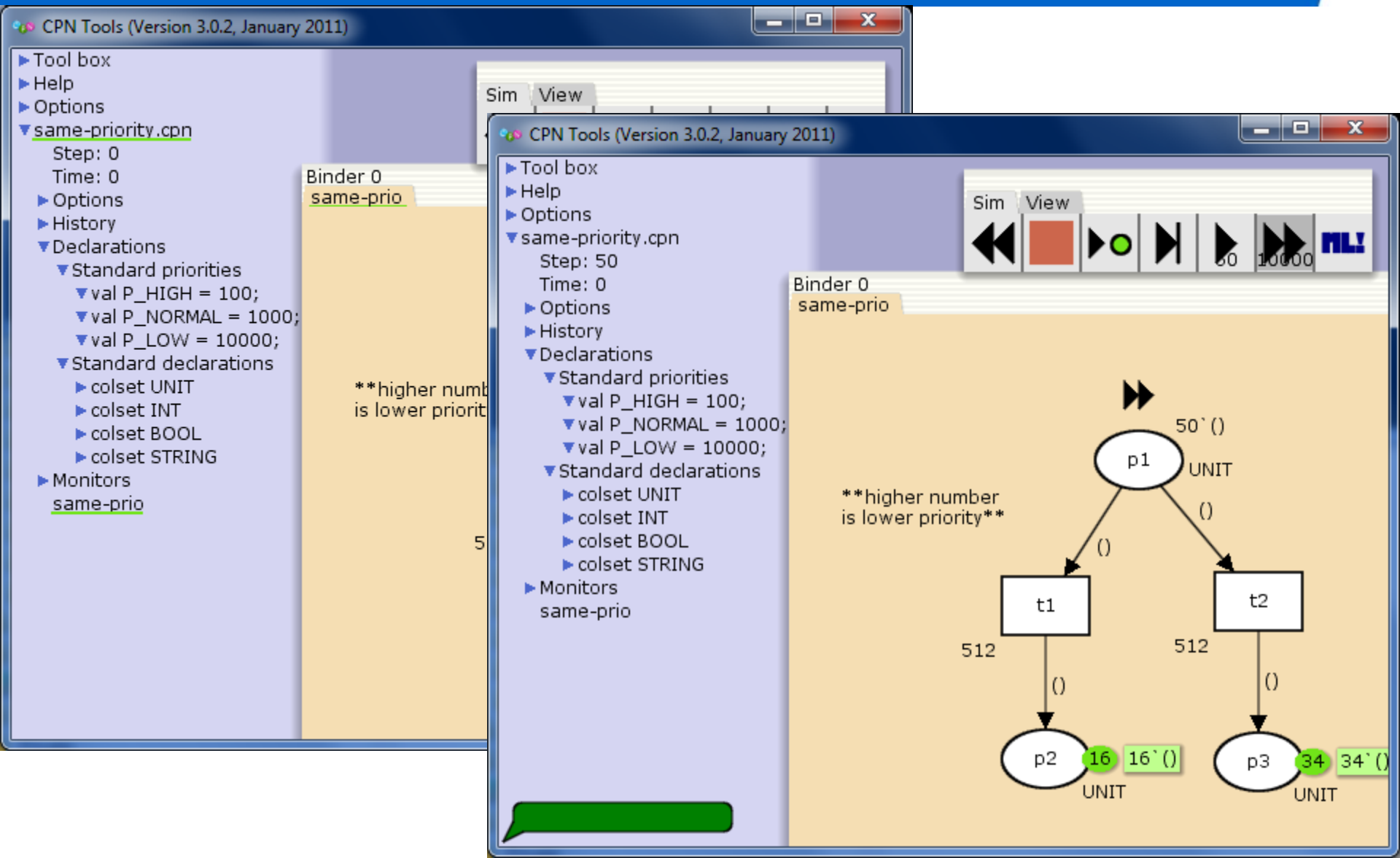
- colset Player = string;
- colset Team = list Player;
- var p: Player;
- var t,s:Team;



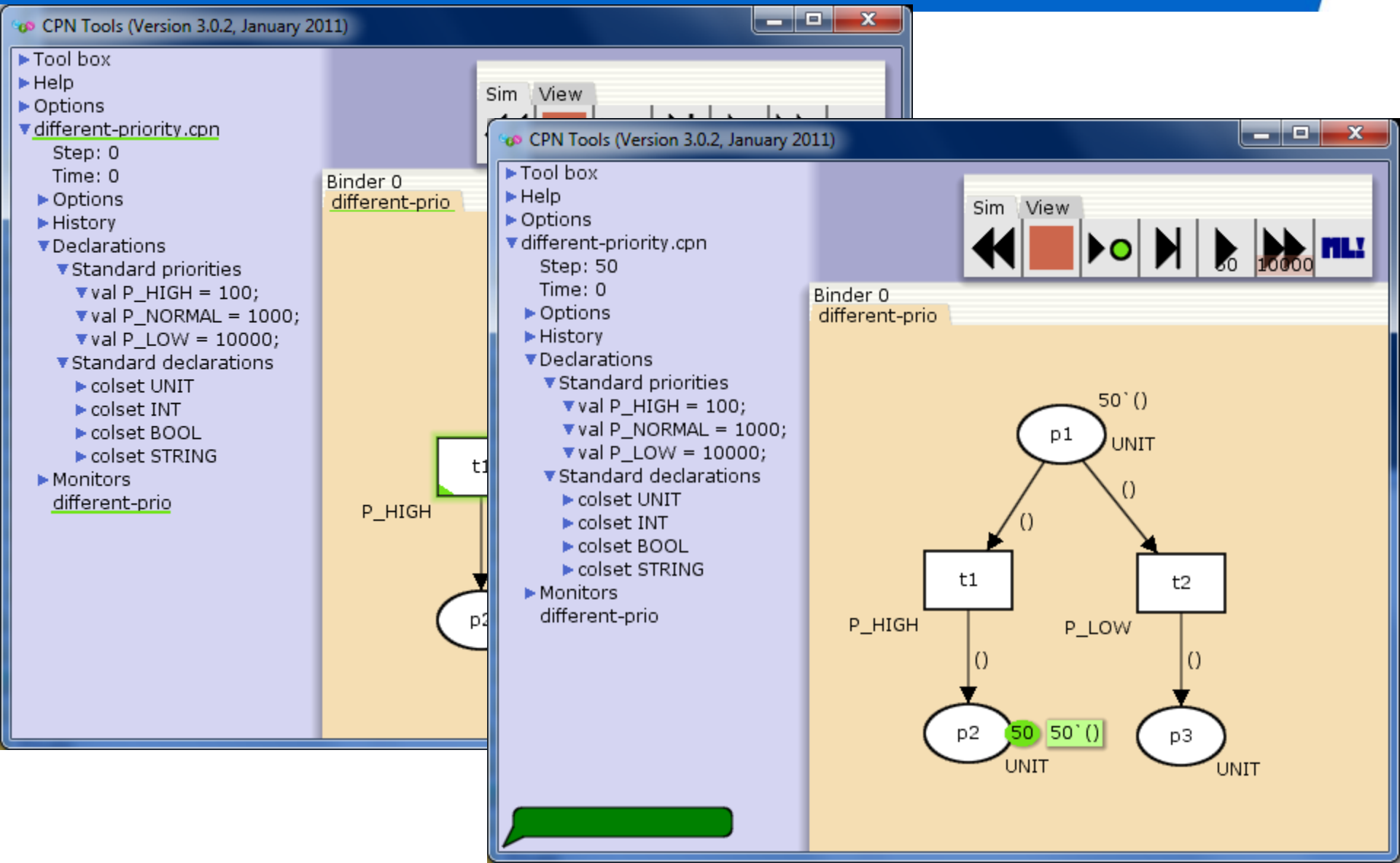
Priority (no priority P_NORMAL = 1000)



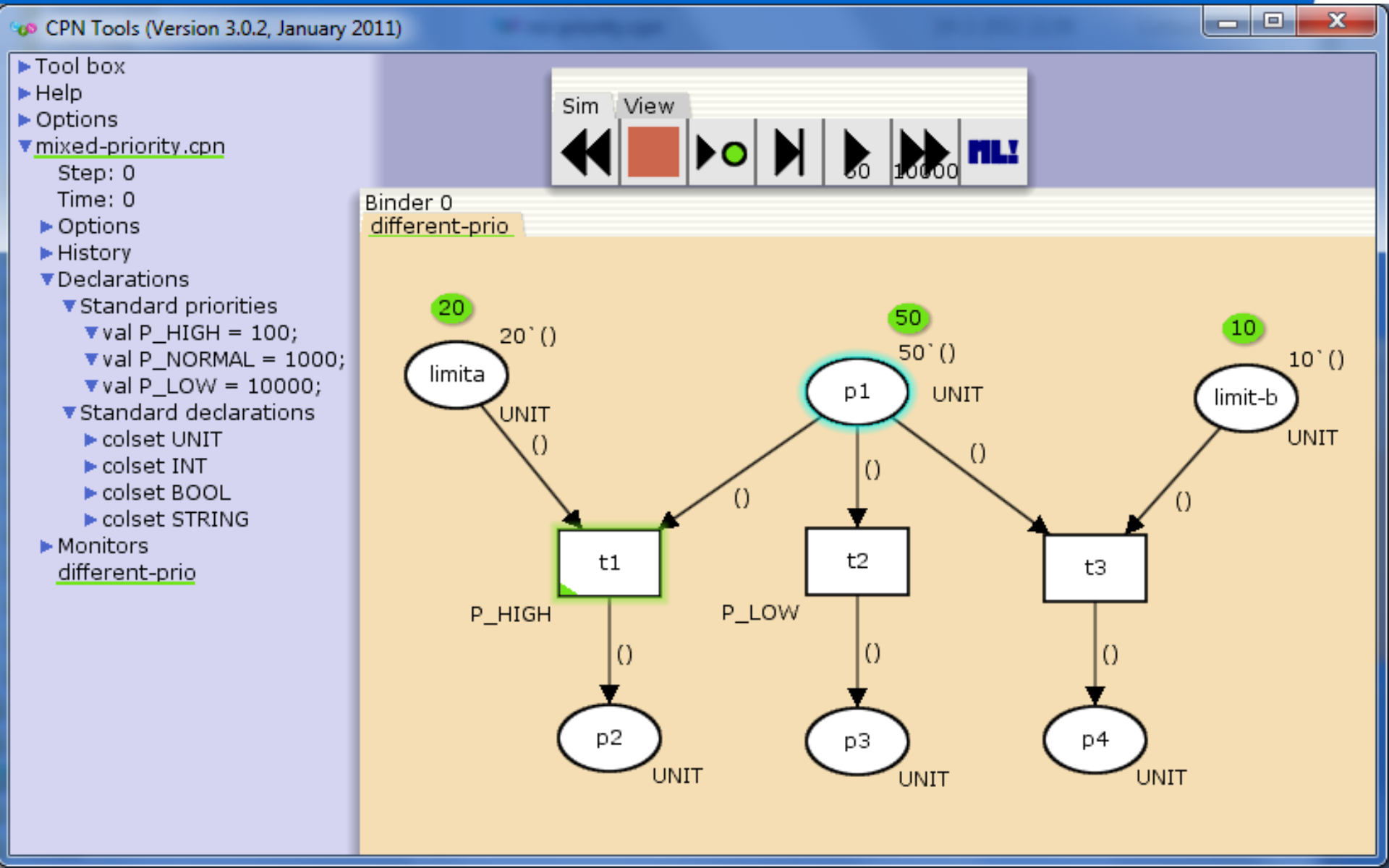
Priority (same)



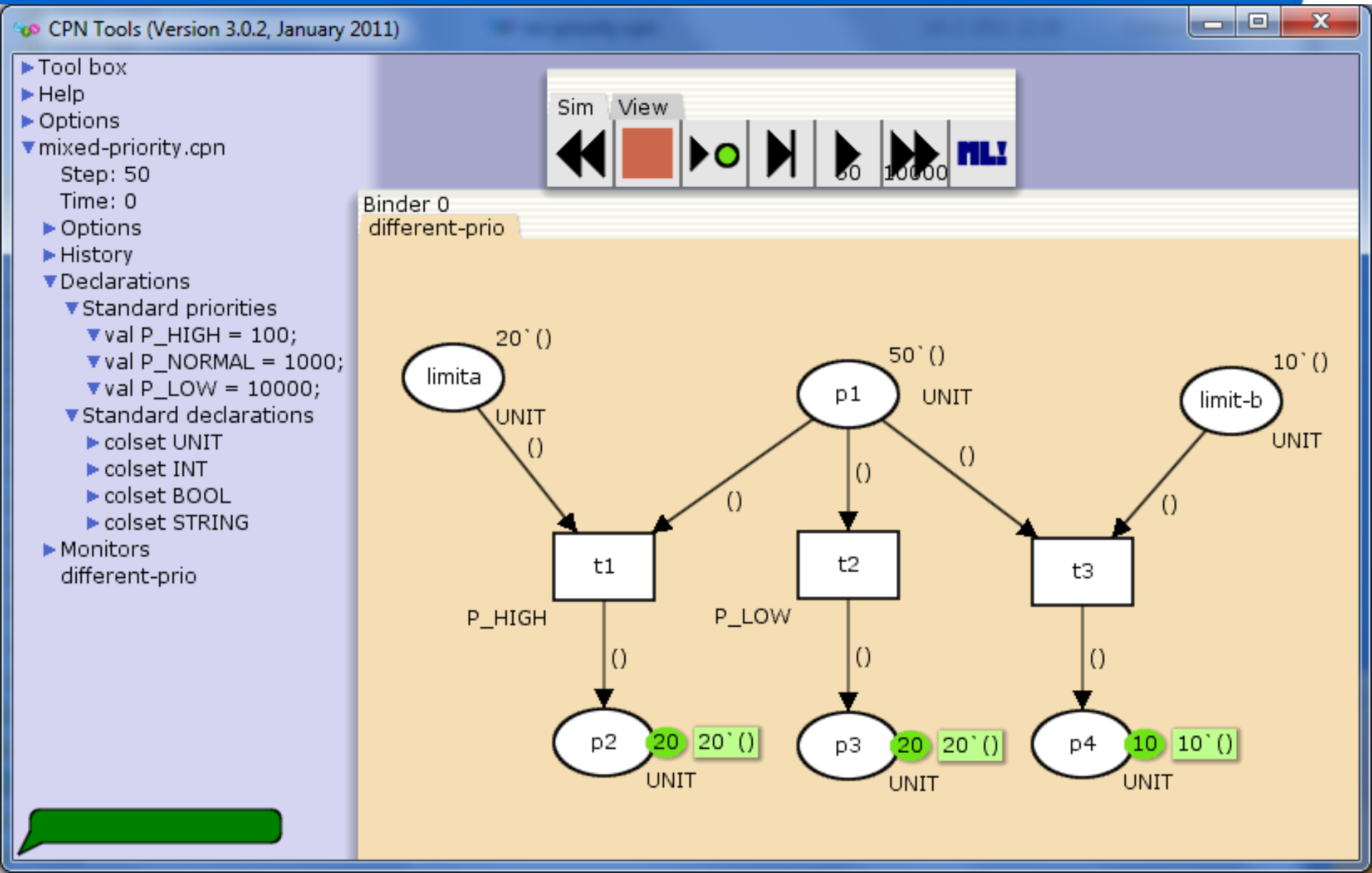
Priority (P_HIGH wins)



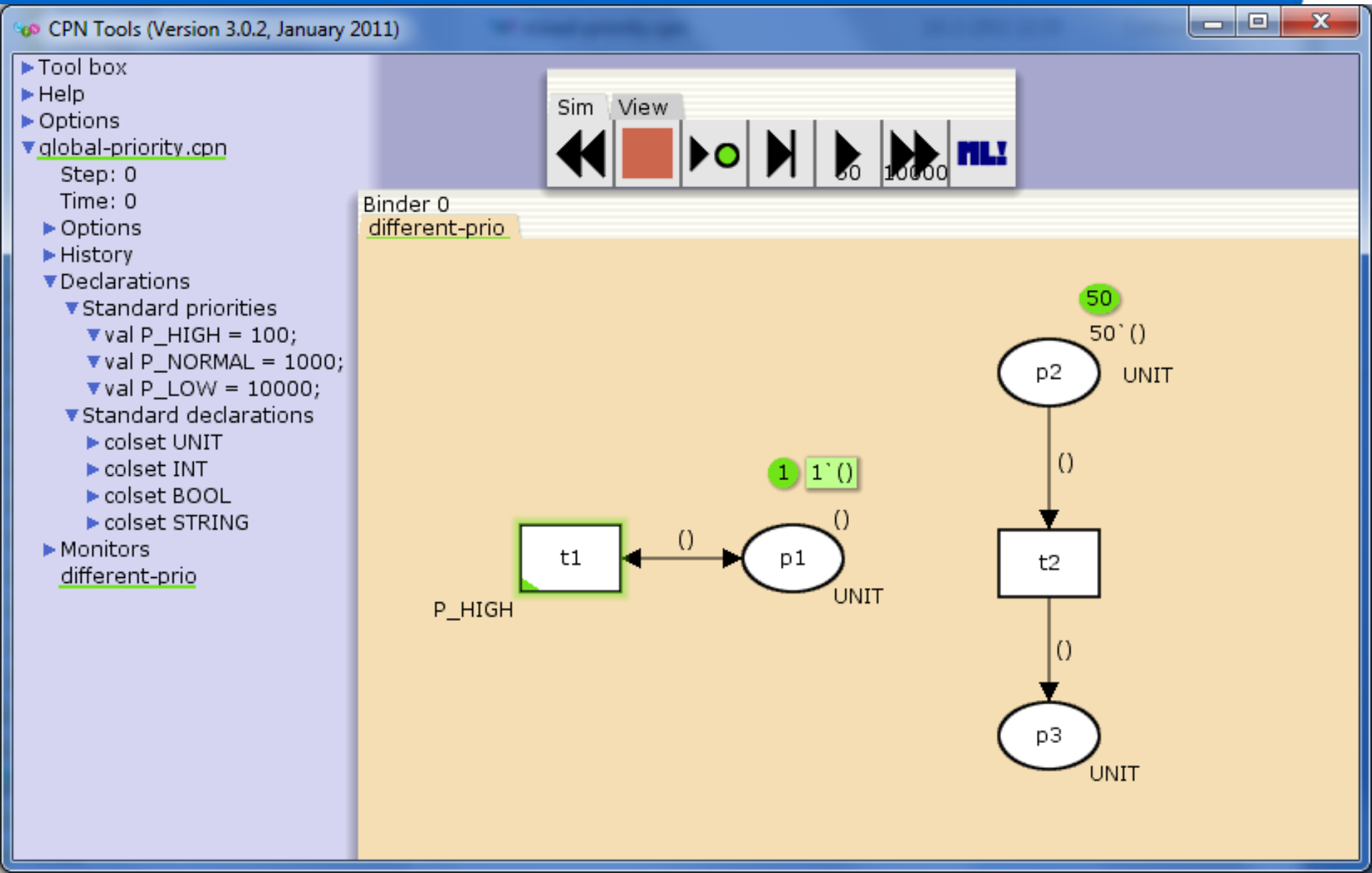
Priority: Guess final state



Result


CPN Tools (Version 3.0.2, January 2011)
Tool box
Help
Options
mixed-priority.cpn
Step: 50
Time: 0
Options
History
Declarations
Standard priorities
val P_HIGH = 100;
val P_NORMAL = 1000;
val P_LOW = 10000;
Standard declarations
colset UNIT
colset INT
colset BOOL
colset STRING
Monitors
different-prio
Sim
View
Binder 0
different-prio
limita
20`()
UNIT
()
p1
50`()
UNIT
limit-b
10`()
UNIT
t1
t2
t3
P_HIGH
P_LOW
p2
20
20`()
UNIT
p3
20
20`()
UNIT
p4
10
10`()
UNIT

Global property (t2 never fires)

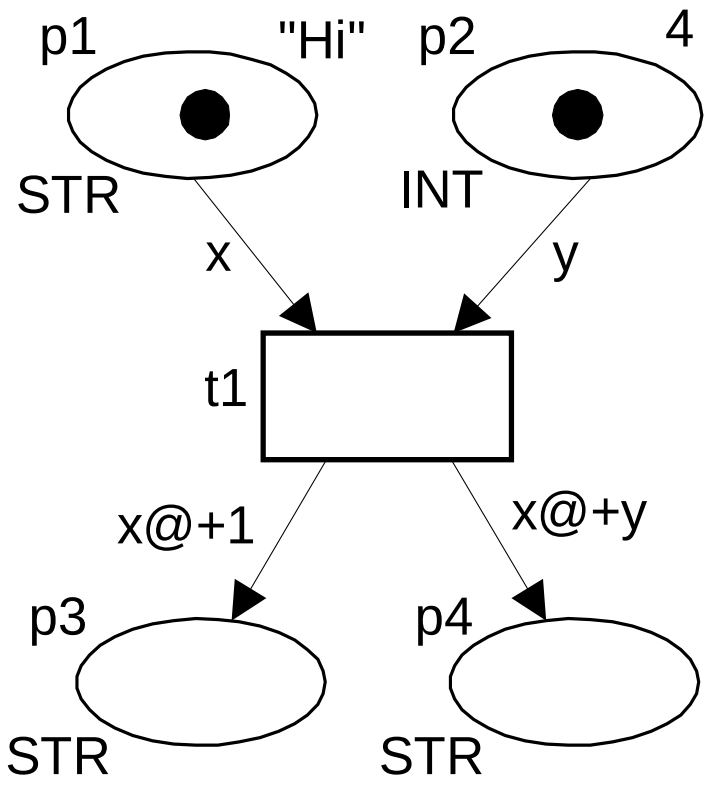


Time in CPN

- Tokens are either **untimed** (are always available) or **timed** (bear a timestamp).
- Color sets can be made timed color sets by adding the keyword **timed**.
- A **delay** is modeled by **v@+d** as arc expression on an outgoing arc where v is the value and d is the delay of the produced token.
- Delays may depend on the values of tokens to be consumed (i.e., through the binding of variables).

Example

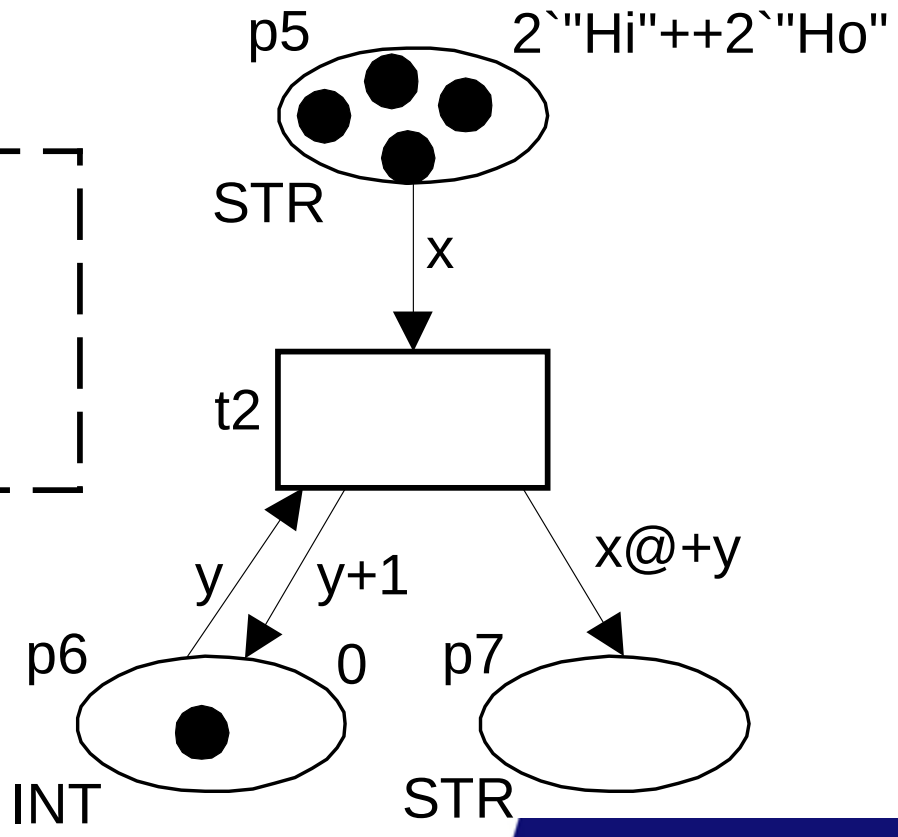
```
color STR = string timed;
var x:STR;
color INT = int;
var y:INT;
```



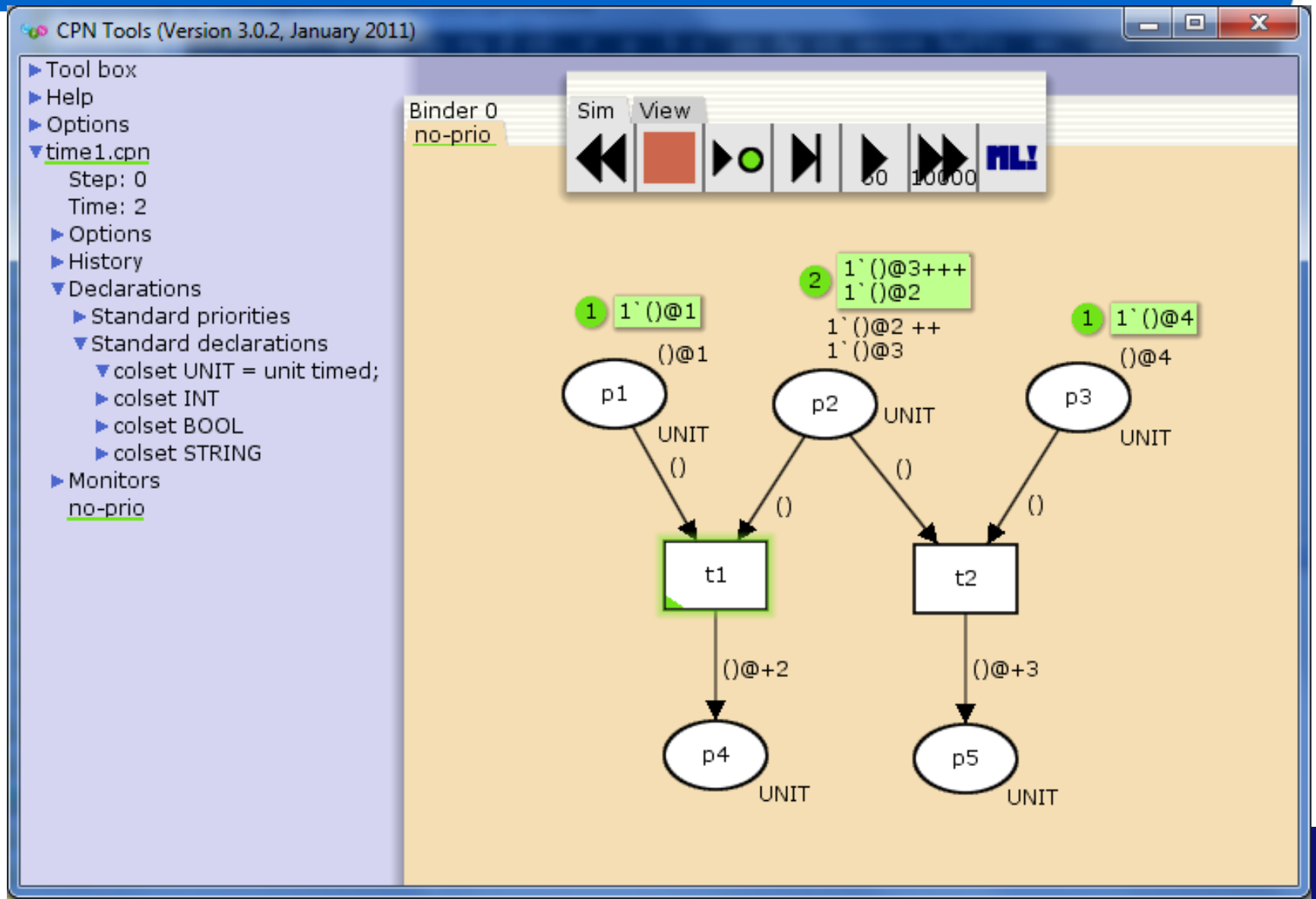
Exercise

- **Determine a possible final state.**

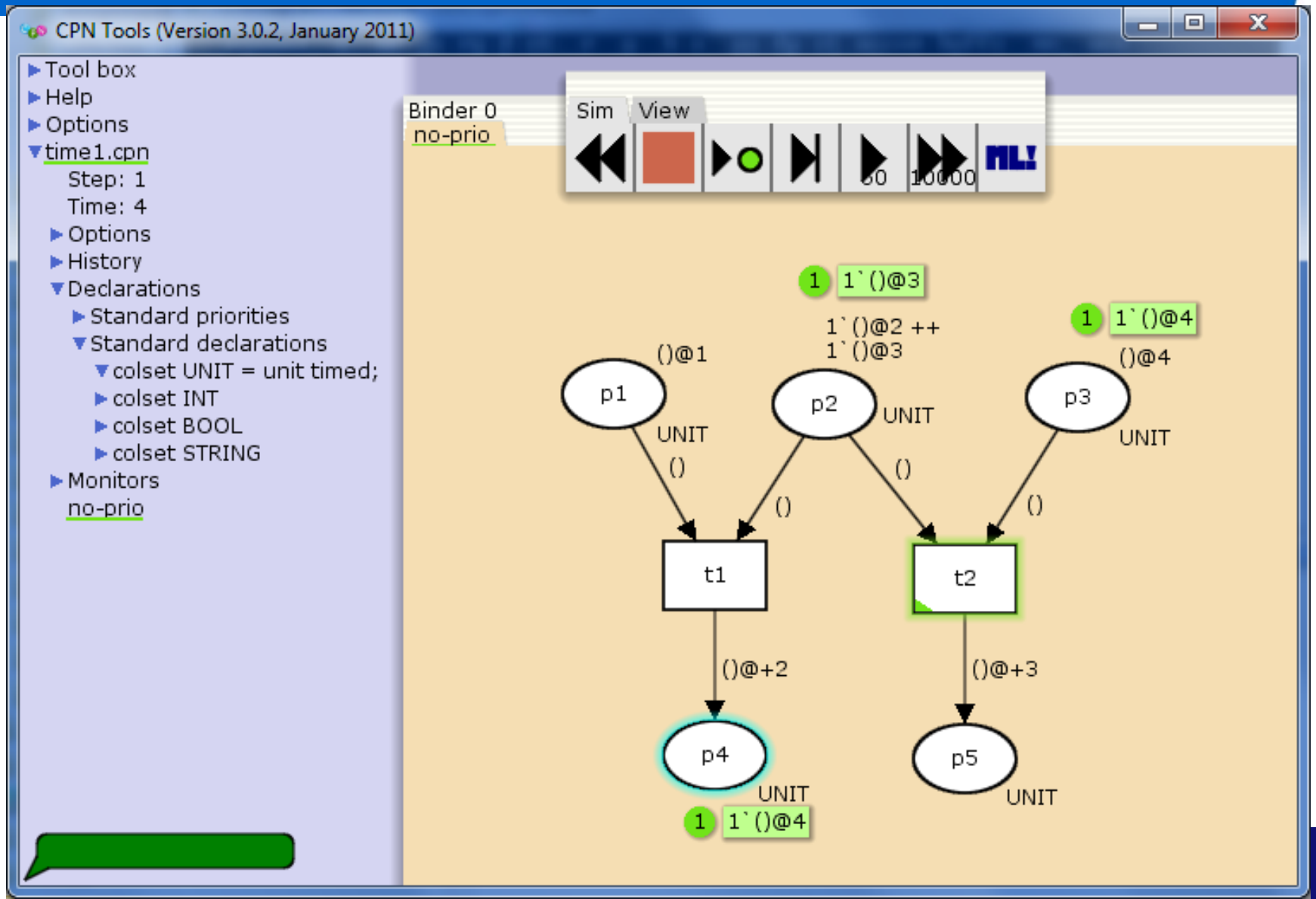
```
color STR = string timed;
var x:STR;
color INT = int;
var y:INT;
```



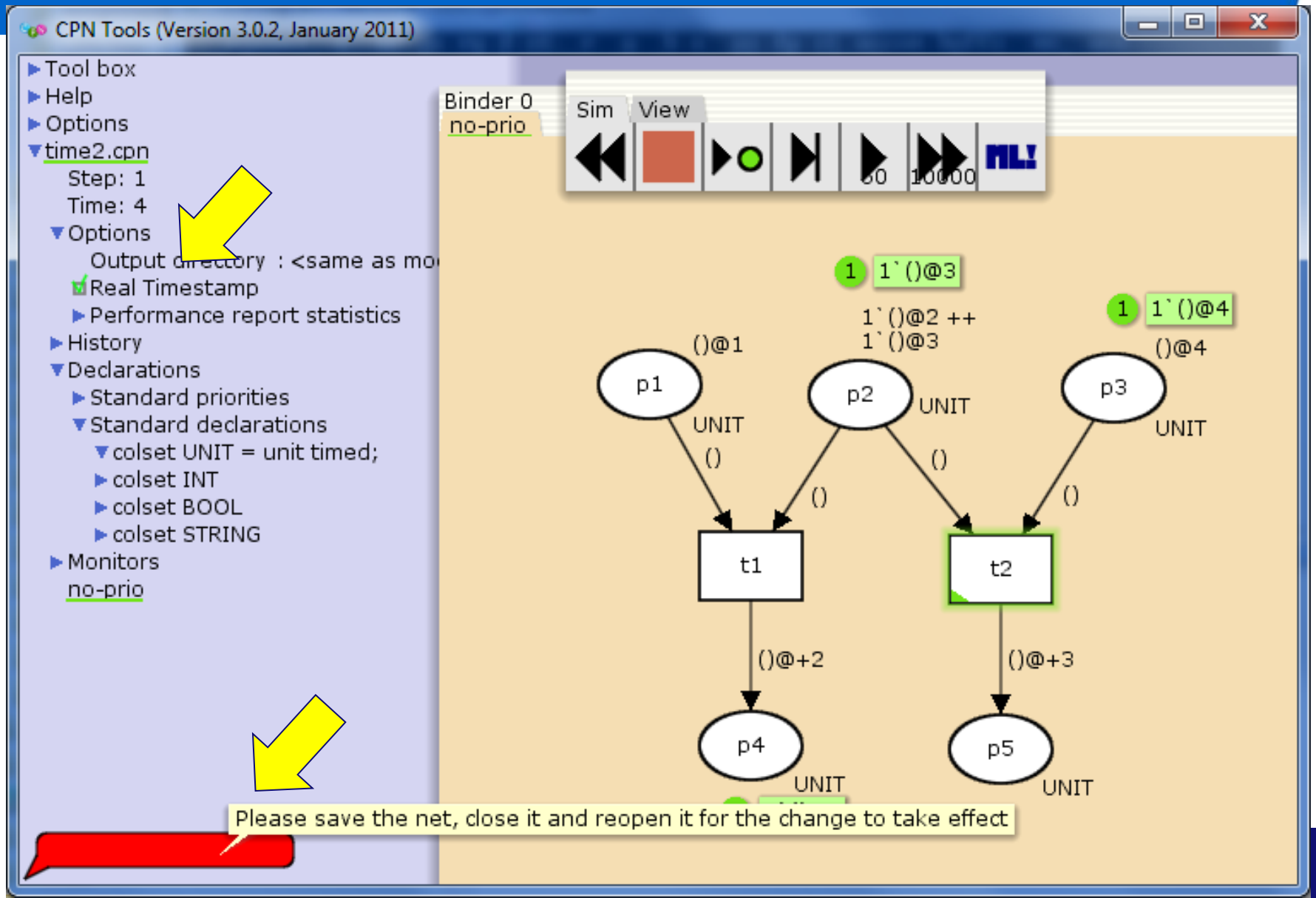
Time (t1 is enabled at time 2)



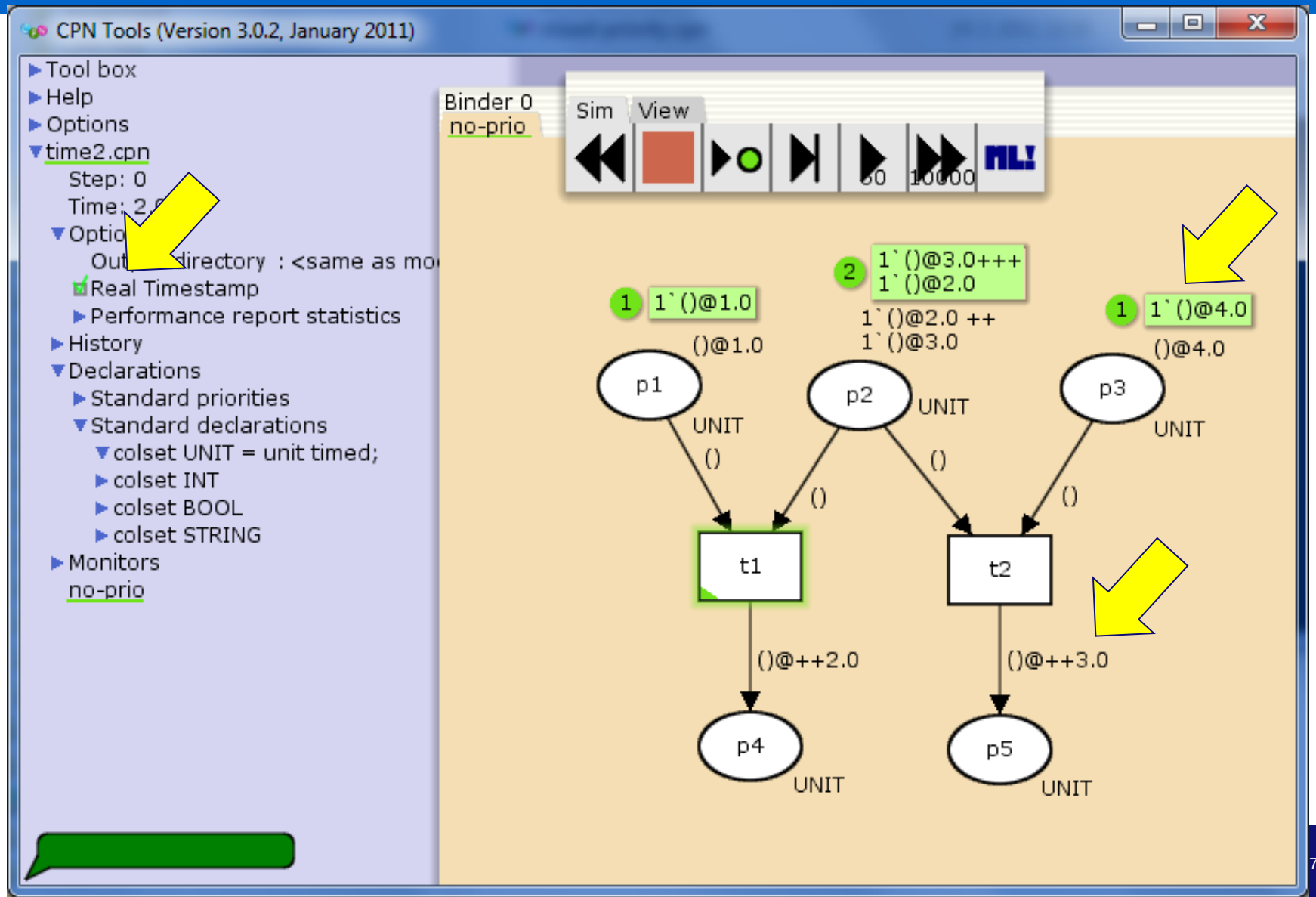
t1 fired at time 2; t2 is enabled at time 4



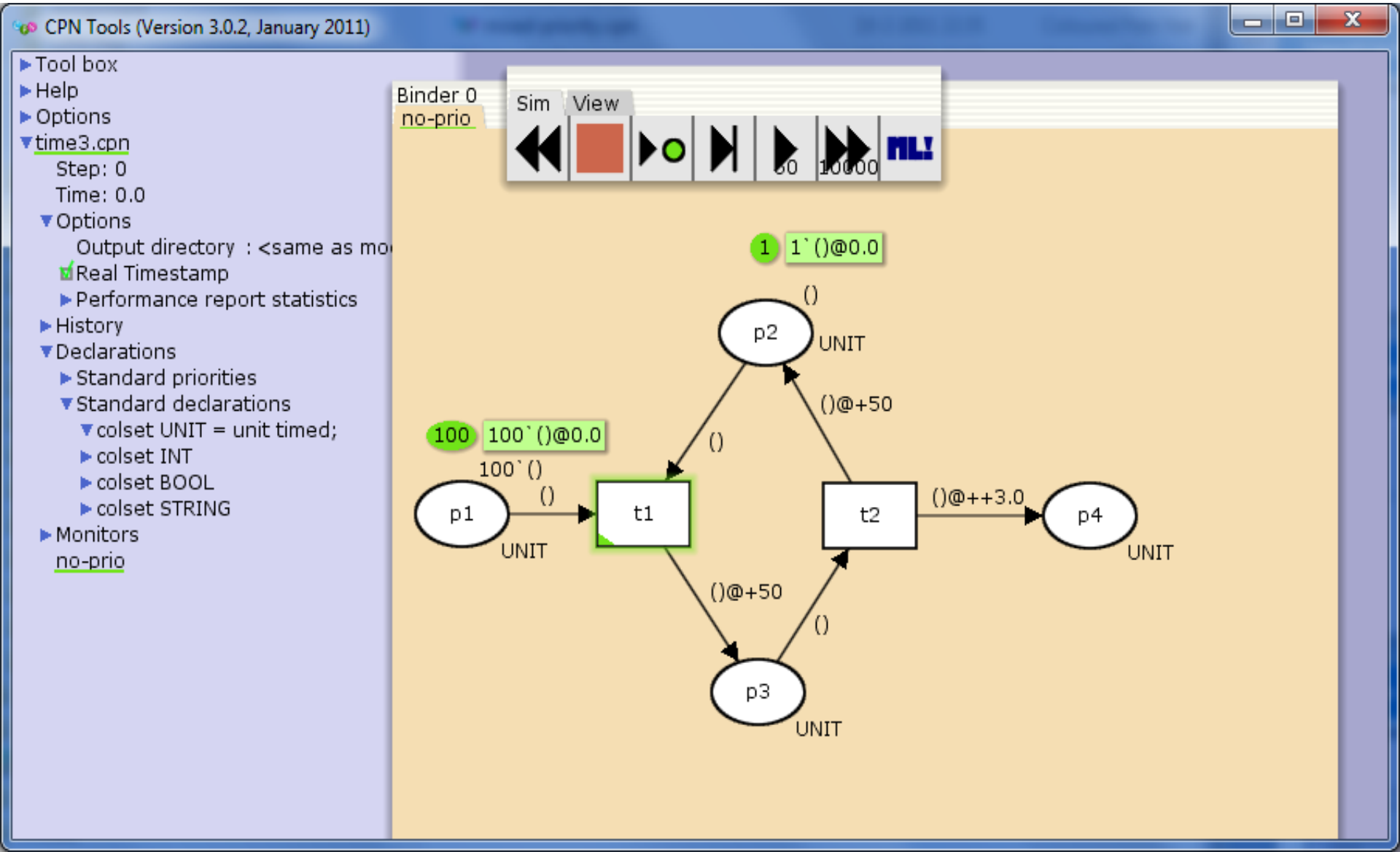
“Real” time


CPN Tools (Version 3.0.2, January 2011)
Tool box
Help
Options
time2.cpn
Step: 1
Time: 4
Options
Output directory : <same as mo
Real Timestamp
Performance report statistics
History
Declarations
Standard priorities
Standard declarations
colset UNIT = unit timed;
colset INT
colset BOOL
colset STRING
Monitors
no-prio
Binder 0
no-prio
Sim
View
1 1`()@3
1`()@2 ++
1`()@3
1 1`()@4
()@1
()@4
p1
p2
p3
UNIT
UNIT
UNIT
()
()
()
()
t1
t2
()@+2
()@+3
p4
p5
UNIT
UNIT
Please save the net, close it and reopen it for the change to take effect

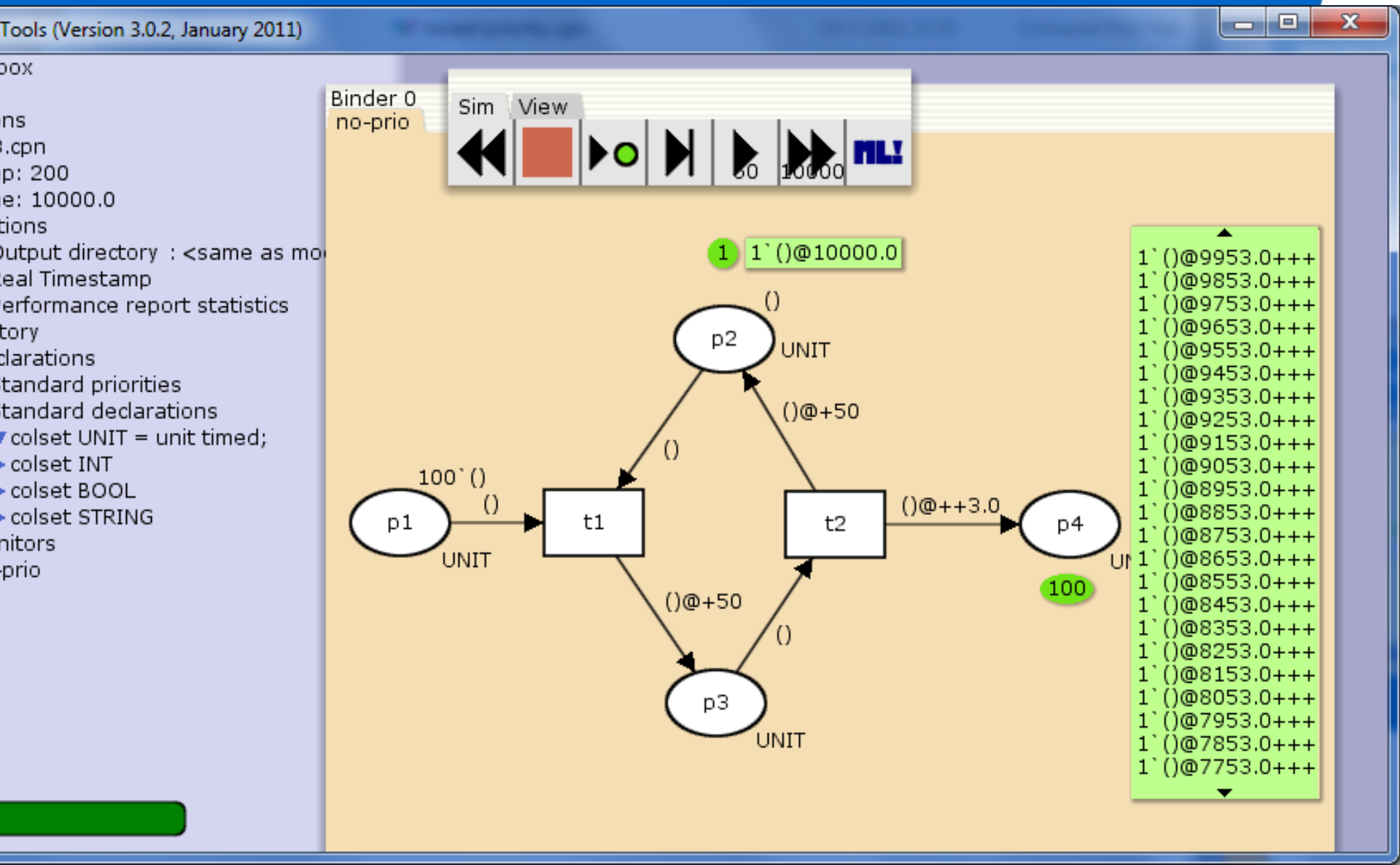
Note the types and the @++



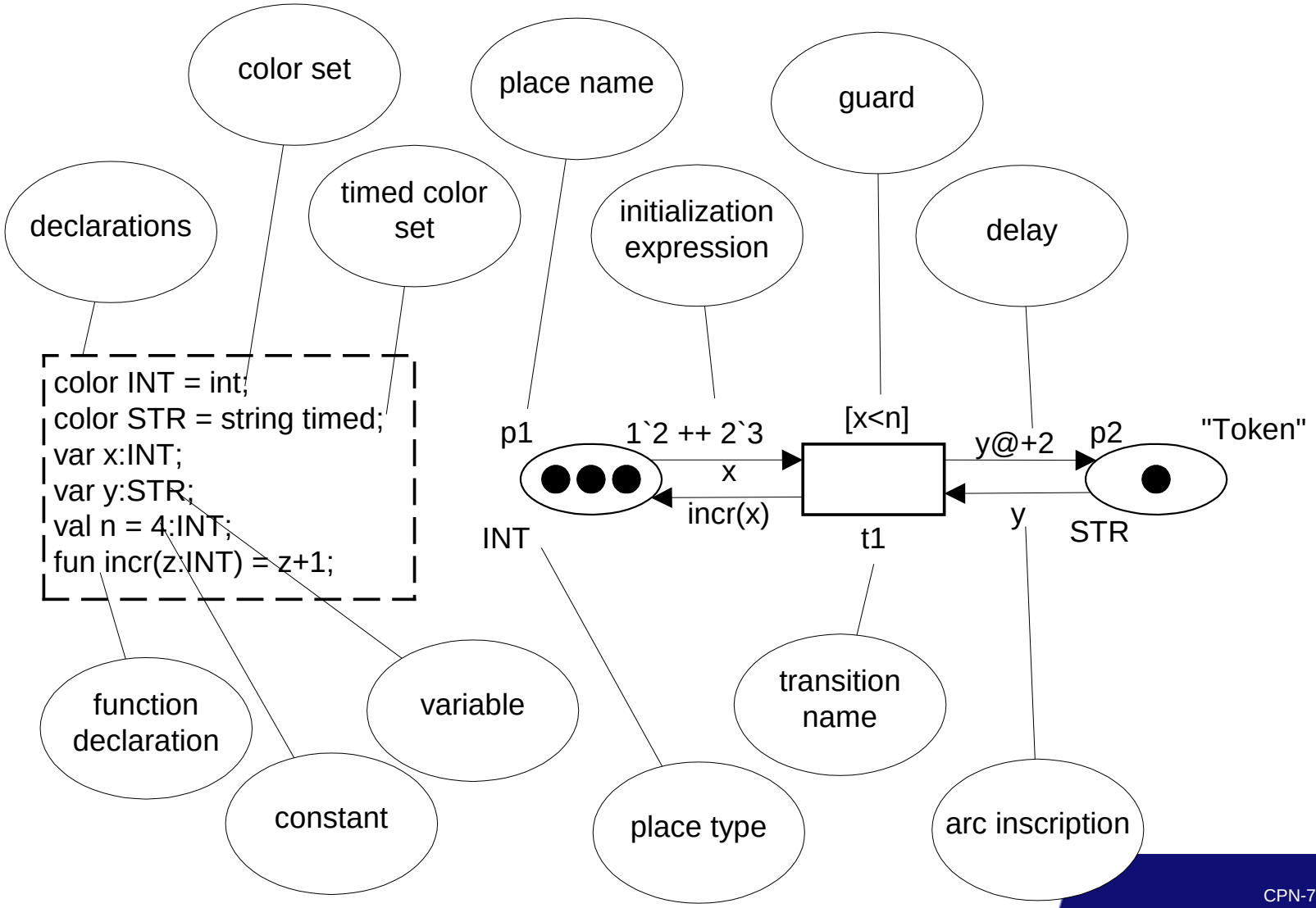
Determine final state



Final state (time = 10000)



Overview of CPN (with color and time)



Coffee and tea example (1)

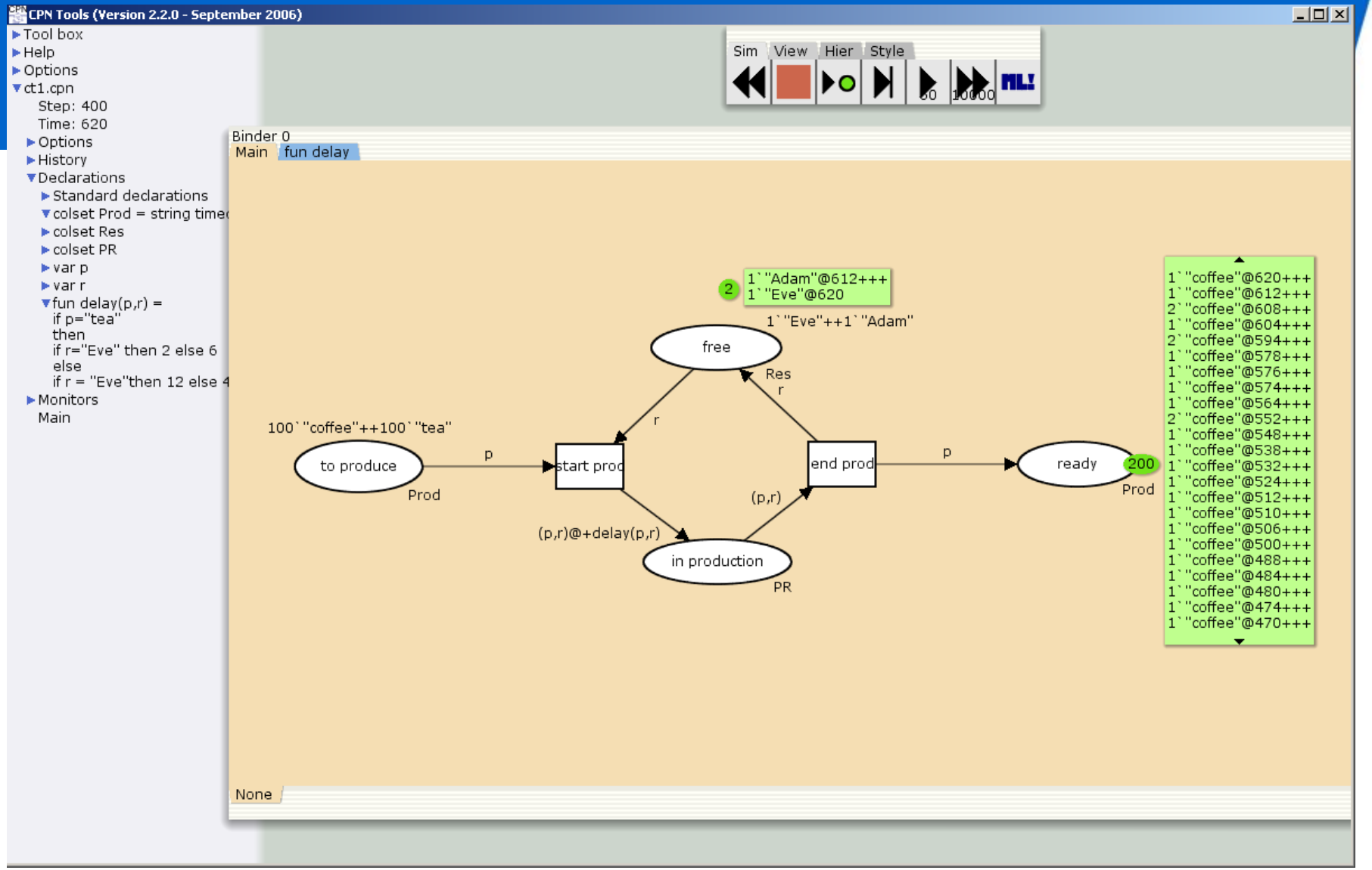
- **We need to produce 100 cups of tea and 100 cups of coffee.**
- **There are two persons able to manufacture these drinks: Adam and Eve.**
- **Assume "random allocation".**
- **Production times:**

	Eve	Adam
tea	2	6
coffee	12	4

Coffee and tea example (2)

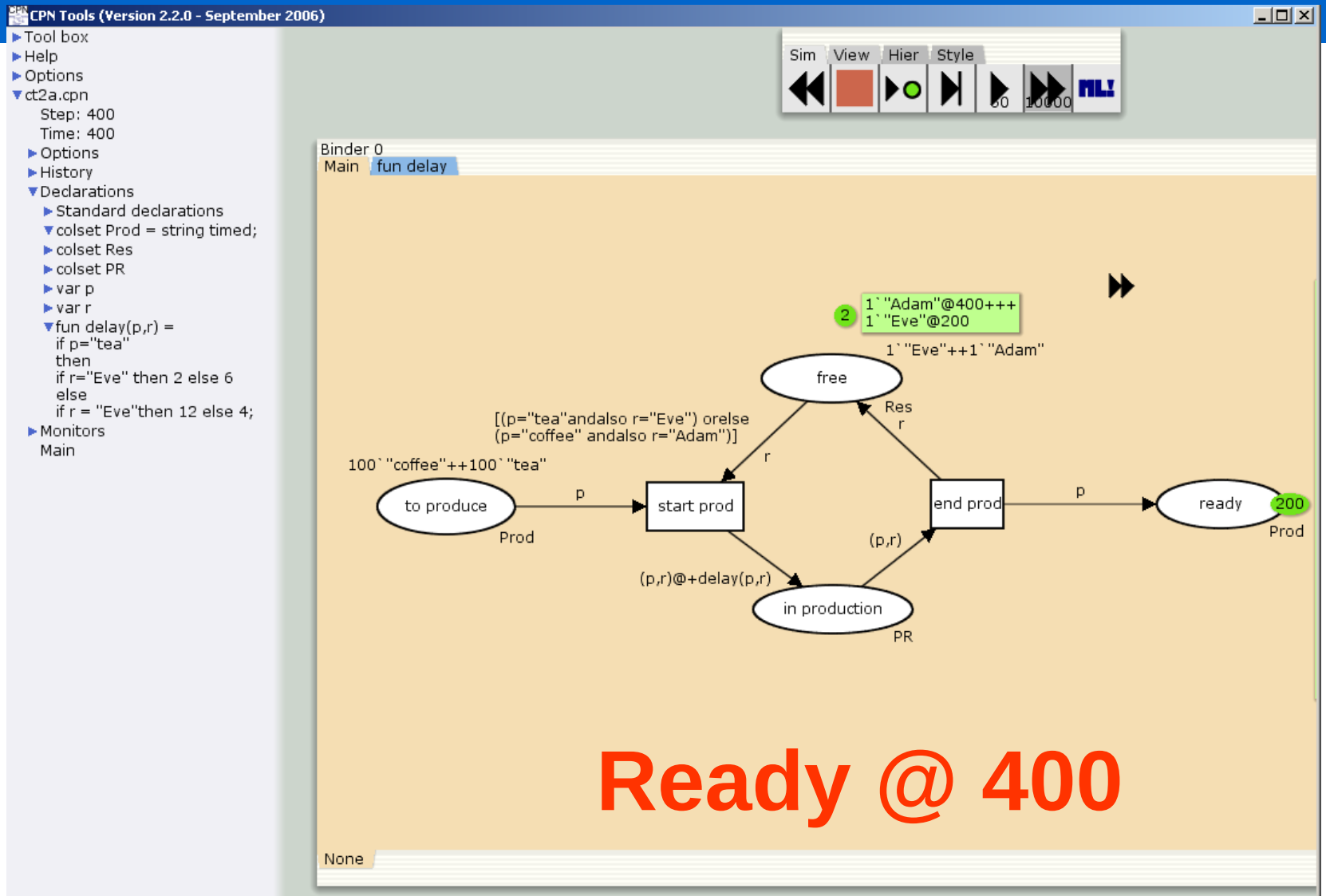
- Simulate the model a couple of times and record the makespan.
- Evaluate two control strategies:
  - Eve just makes tea and Adam just makes coffee.
  - Adam makes coffee and Eve can make both.
  - Eve makes tea and Adam can make both.
- Why is it diffcult to model priorities/preferences?

	Eve	Adam
tea	2	6
coffee	12	4

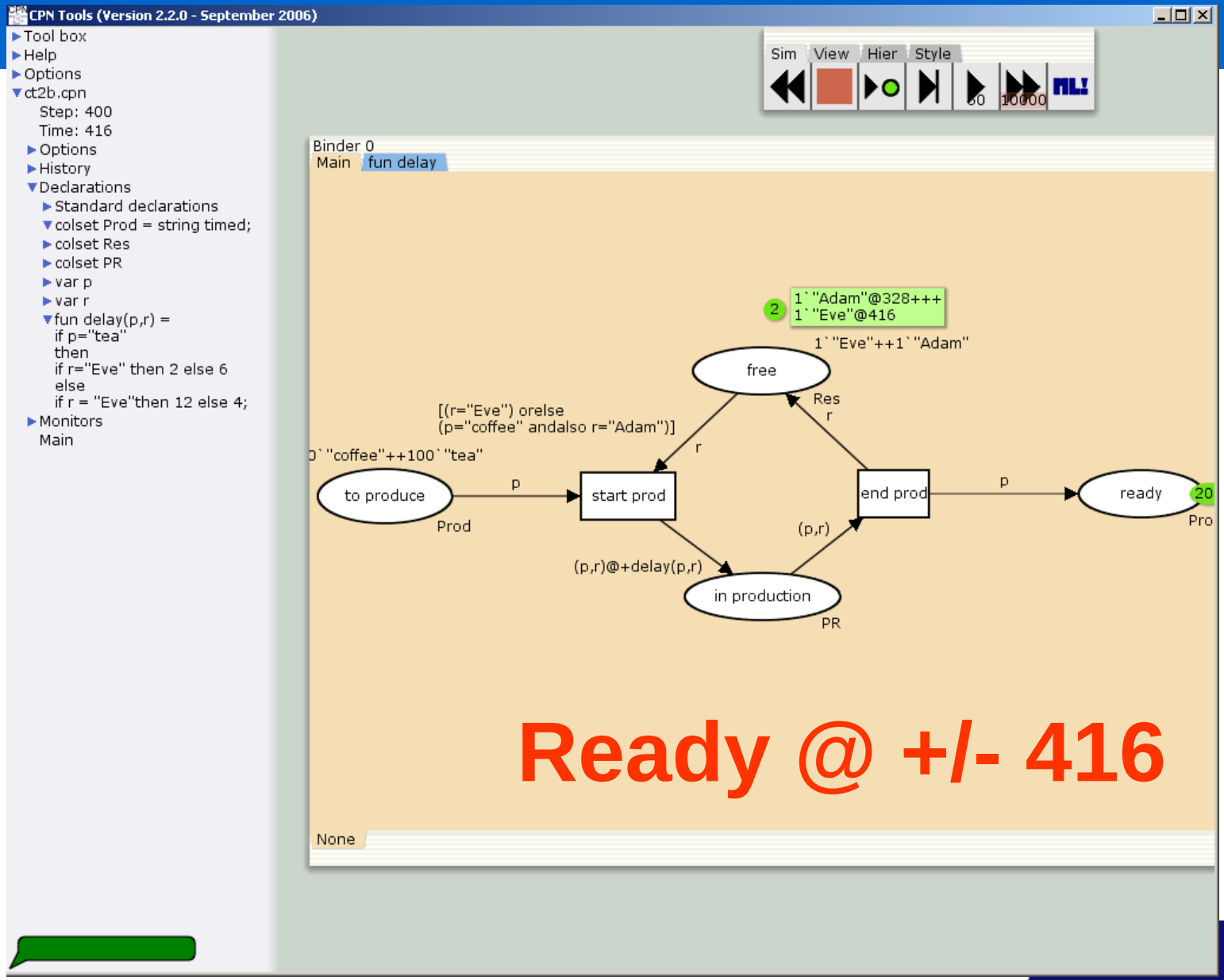

CPN Tools (Version 2.2.0 - September 2006)
Tool box
Help
Options
ct1.cpn
Step: 400
Time: 620
Options
History
Declarations
Standard declarations
colset Prod = string time
colset Res
colset PR
var p
var r
fun delay(p,r) =
if p="tea"
then
if r="Eve" then 2 else 6
else
if r = "Eve"then 12 else 4
Monitors
Main
Sim
View
Hier
Style
Binder 0
Main
fun delay
2
1`"Adam"@612+++
1`"Eve"@620
1`"Eve"++1`"Adam"
free
Res
r
r
100`"coffee"++100`"tea"
to produce
Prod
p
start prod
end prod
p
ready
200
Prod
(p,r)
(p,r)@+delay(p,r)
in production
PR
1`"coffee"@620+++
1`"coffee"@612+++
2`"coffee"@608+++
1`"coffee"@604+++
2`"coffee"@594+++
1`"coffee"@578+++
1`"coffee"@576+++
1`"coffee"@574+++
1`"coffee"@564+++
2`"coffee"@552+++
1`"coffee"@548+++
1`"coffee"@538+++
1`"coffee"@532+++
1`"coffee"@524+++
1`"coffee"@512+++
1`"coffee"@510+++
1`"coffee"@506+++
1`"coffee"@500+++
1`"coffee"@488+++
1`"coffee"@484+++
1`"coffee"@480+++
1`"coffee"@474+++
1`"coffee"@470+++
None

Ready @ +/- 620

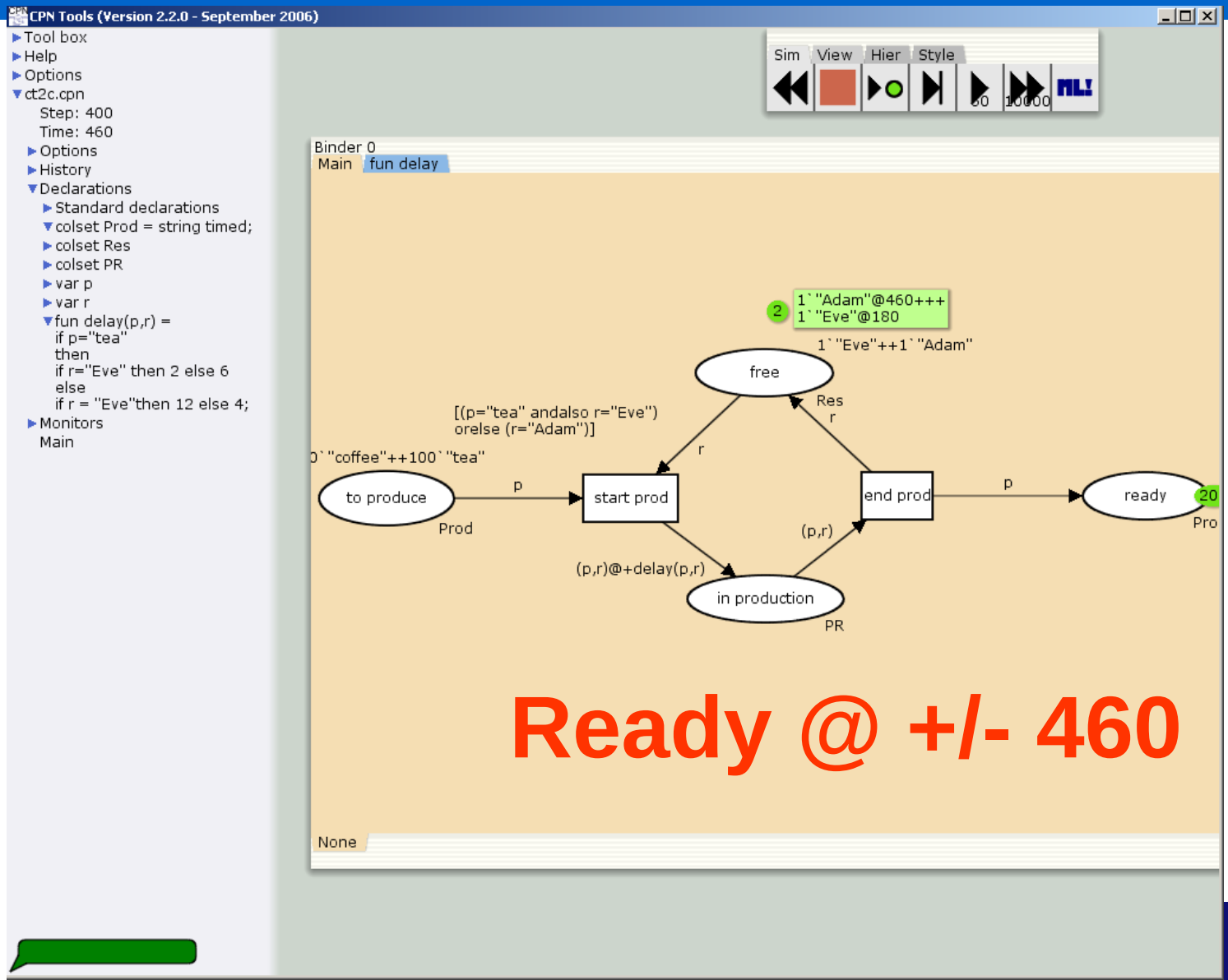
Eve just makes tea and Adam just makes coffee.



Adam makes coffee and Eve can make both



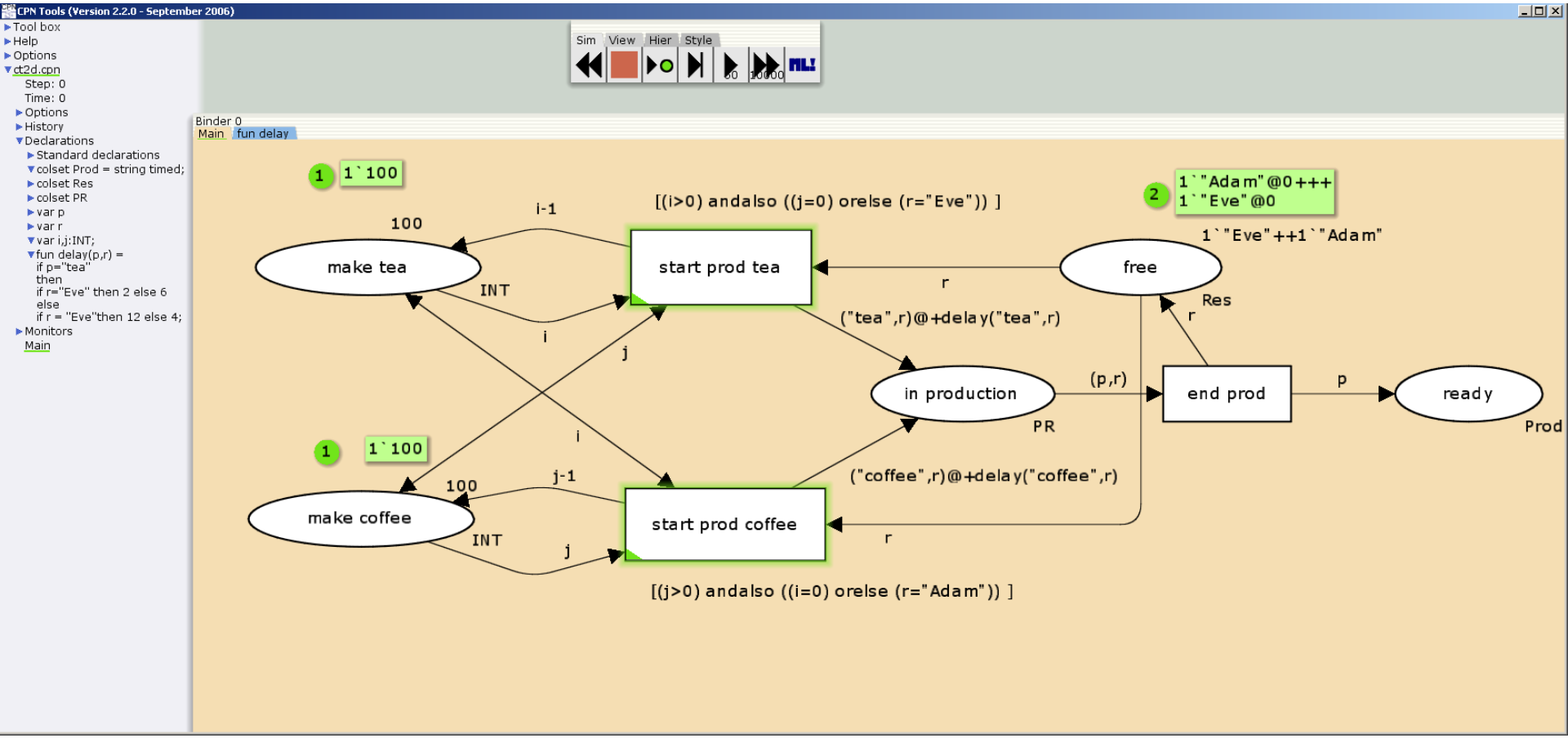
Eve makes tea and Adam can make both



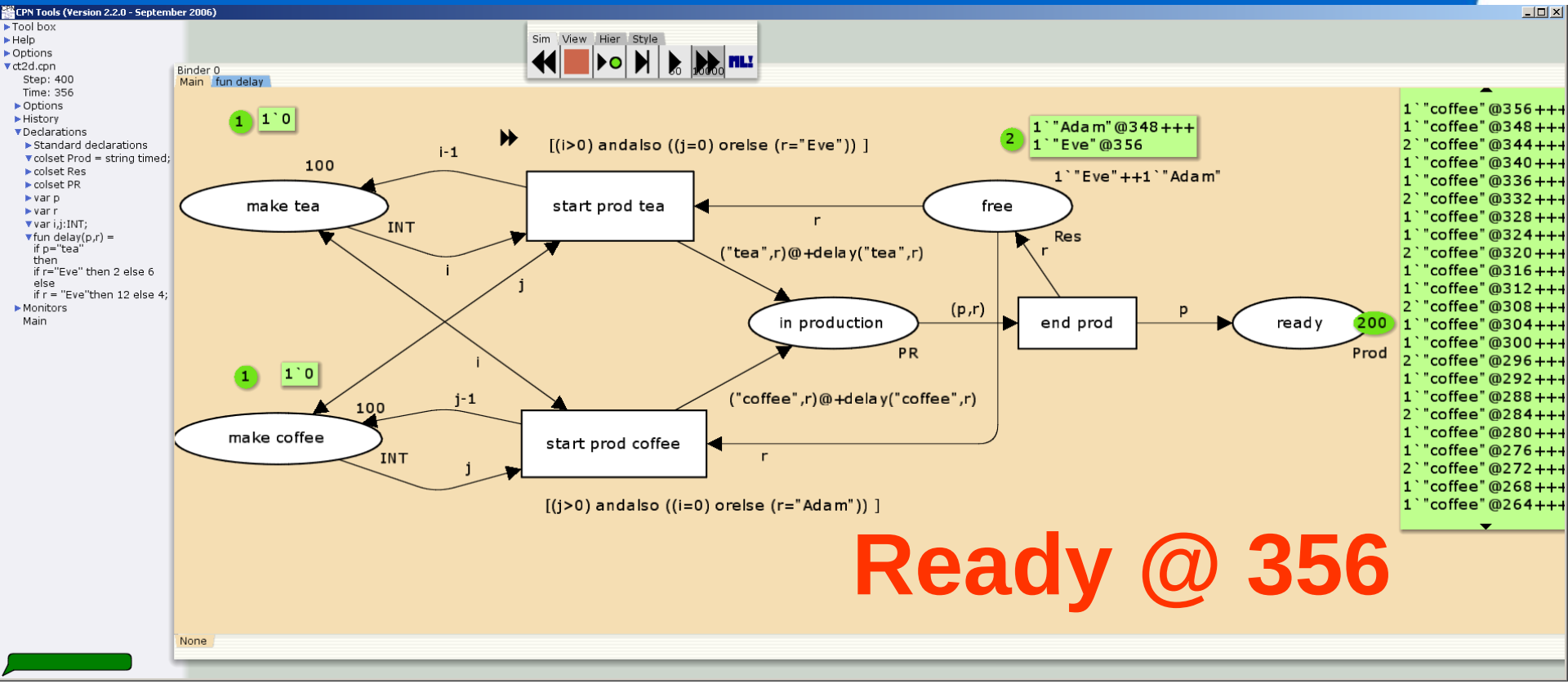
A smarter strategy

- **Eve just makes tea and Adam just makes coffee unless ...**
  - **Eve can make coffee if there are no tea orders left.**
  - **Adam can make tea if there are no coffee orders left.**
- **Why is it diffcult to model priorities/preferences?**
- **Let us look at an intermediate solution using counters rather than lists.**

CPN model with counters



Almost optimal makespan ...



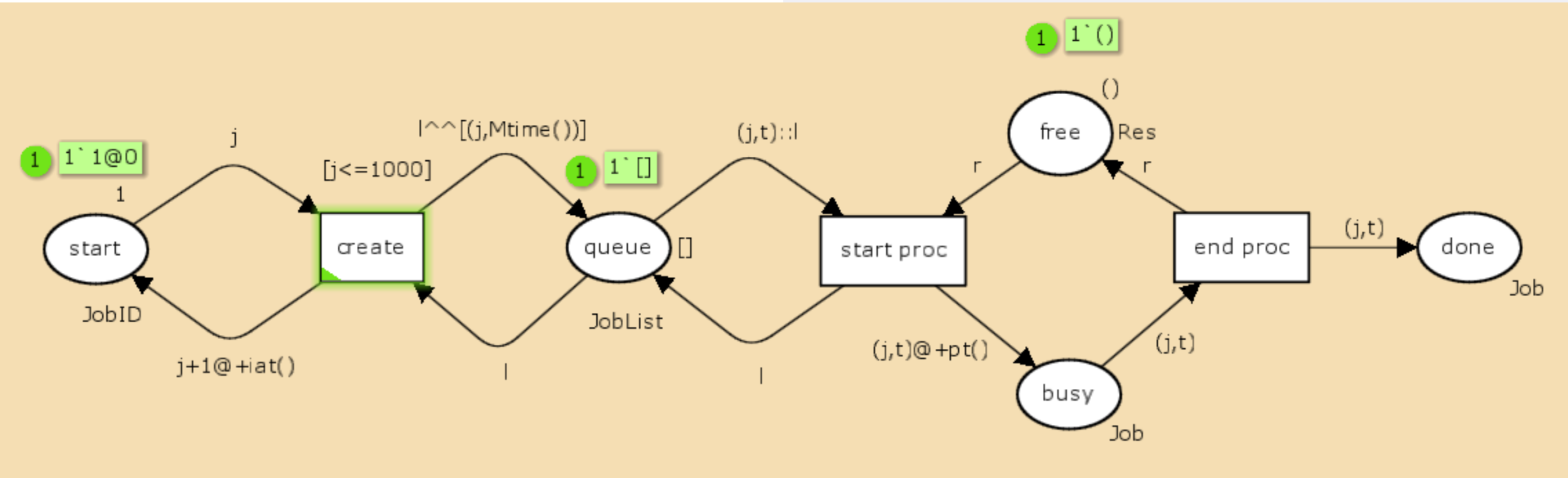
Adam: 87 coffee
Eve: 100 tea and 13 coffee
Makespan = 356

Optimal
Adam: 88 coffee
Eve: 100 tea and 12 coffee
Makespan = 352

intermezzo

M/G/1 queue

```
Declarations
  Standard declarations
  colset JobID = int timed;
  colset Timestamp = int;
  colset Job = product JobID*Timestamp timed;
  colset JobList = list Job;
  colset Res = unit;
  var j:JobID;
  var r:Res;
  var l:JobList;
  var t:Timestamp;
  fun iat() = round(exponential(0.05));
  fun pt() = round(normal(10.0,1.0));
  fun Mtime() = IntInf.toInt(time()):int;
```



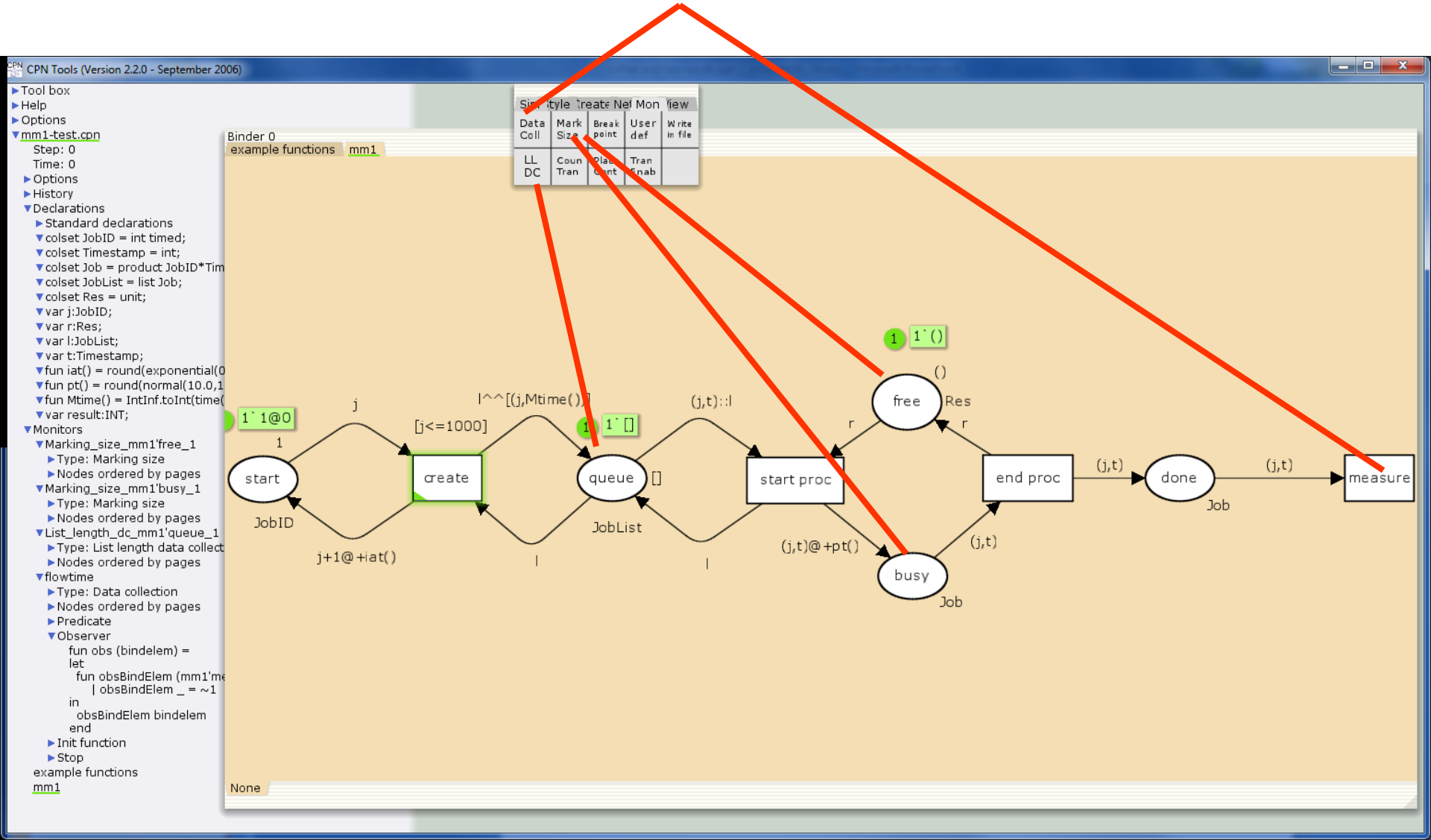
Poisson arrival process (expected average interarrival time is 1/0.05=20). Expected service time is 10 (Normal distribution)

To measure results: CPN monitors

Mon

Data Coll	Mark Size	Break point	User def	Write in file
LL DC	Coun Tran	Place Cont	Tran Enab	

Create monitors



- Monitors
  - Marking_size_mm1'free_1
    - Type: Marking size
    - Nodes ordered by pages
  - Marking_size_mm1'busy_1
    - Type: Marking size
    - Nodes ordered by pages
  - List_length_dc_mm1'queue_1
    - Type: List length data collection
    - Nodes ordered by pages
  - flowtime
    - Type: Data collection
    - Nodes ordered by pages
    - Predicate
    - Observer

```
fun obs (bindelem) =
let
  fun obsBindElem (mm1'measure (1, {j,t})) = Mtime() - t
    | obsBindElem _ = ~1
in
  obsBindElem bindelem
end
```

    - Init function
    - Stop

CPN Tools (Version 2.2.0 - September 2006)

- Tool box
- Help
- Options
- mm1-test.cpn
  - Step: 0
  - Time: 0
  - Options
  - History
  - Declarations
    - Standard declarations
    - colset JobID = int timed;
    - colset Timestamp = int;
    - colset Job = product JobID*Timestamp timed;
    - colset JobList = list Job;
    - colset Res = unit;
    - var j:JobID;
    - var r:Res;
    - var l:JobList;
    - var t:Timestamp;
    - fun iat() = round(exponential(0.05));
    - fun pt() = round(normal(10.0,1.0));
    - fun Mtime() = IntInf.toInt(time()):int;
    - var result:INT;
  - Monitors
    - Marking_size_mm1'free_1
      - Type: Marking size
      - Nodes ordered by pages
    - Marking_size_mm1'busy_1
      - Type: Marking size
      - Nodes ordered by pages
    - List_length_dc_mm1'queue_1
      - Type: List length data collection
      - Nodes ordered by pages
    - flowtime
      - Type: Data collection
      - Nodes ordered by pages
      - Predicate
      - Observer

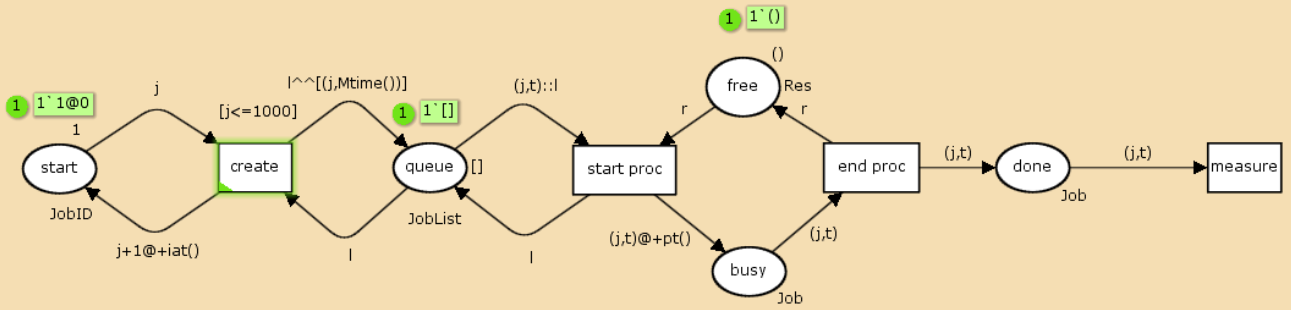
```
fun obs (bindelem) =
let
  fun obsBindElem (mm1'measure (1, {j,t})) = Mtime() - t
    | obsBindElem _ = ~1
in
  obsBindElem bindelem
end
```

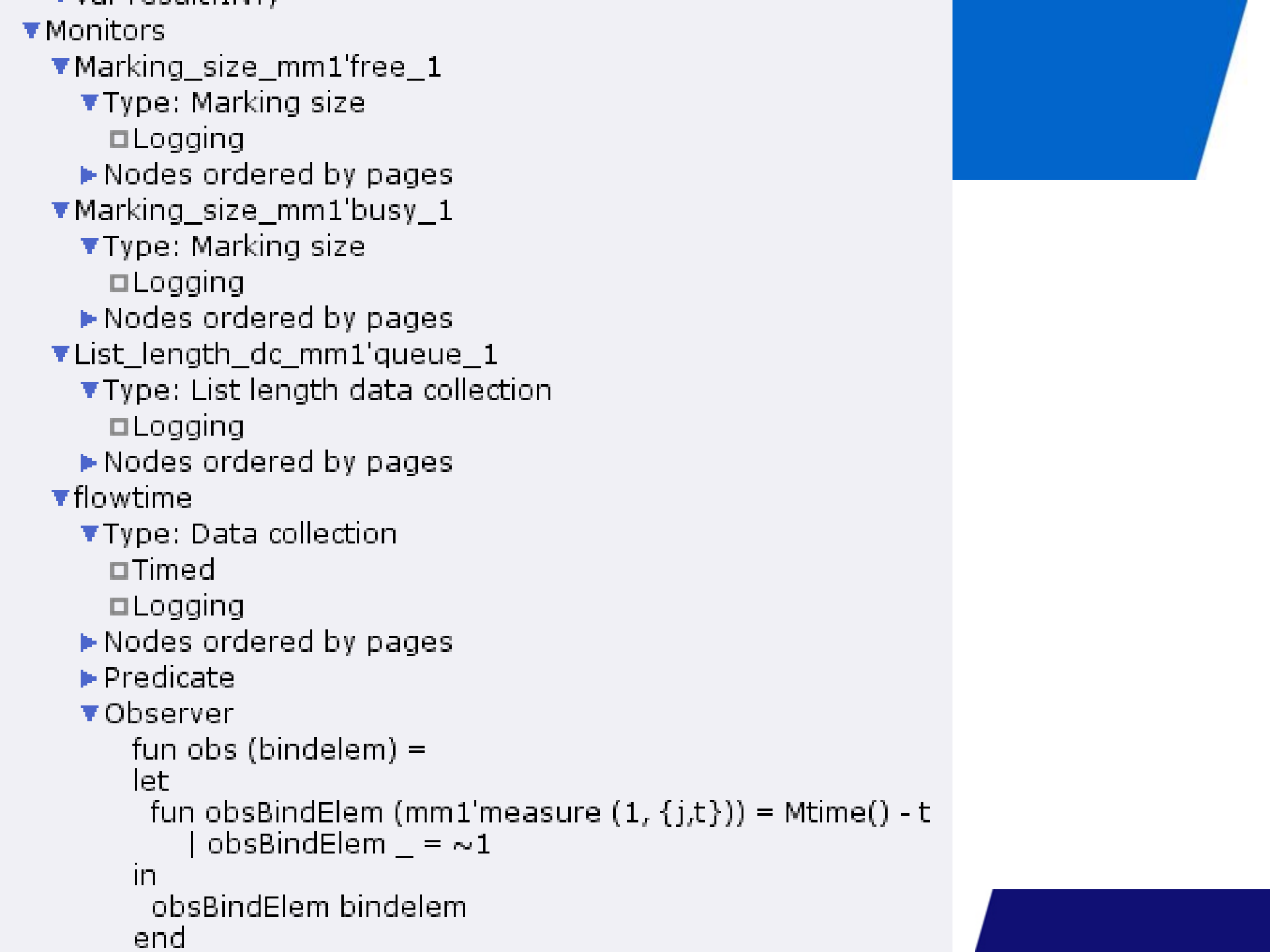
      - Init function
      - Stop
  - example functions
  - mm1

Sim Style Create Iel Mor View

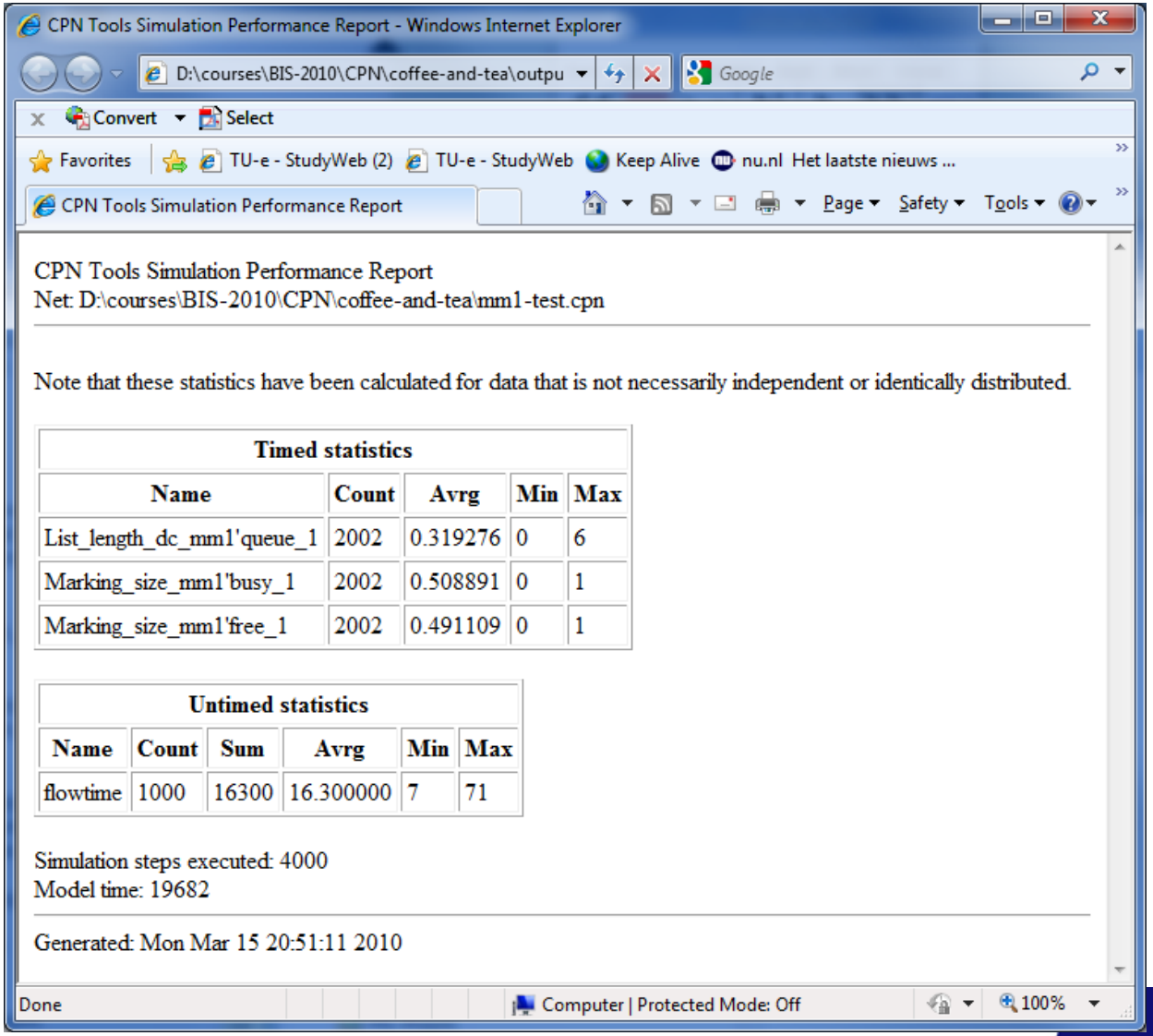
Binder 0

example functions | mm1

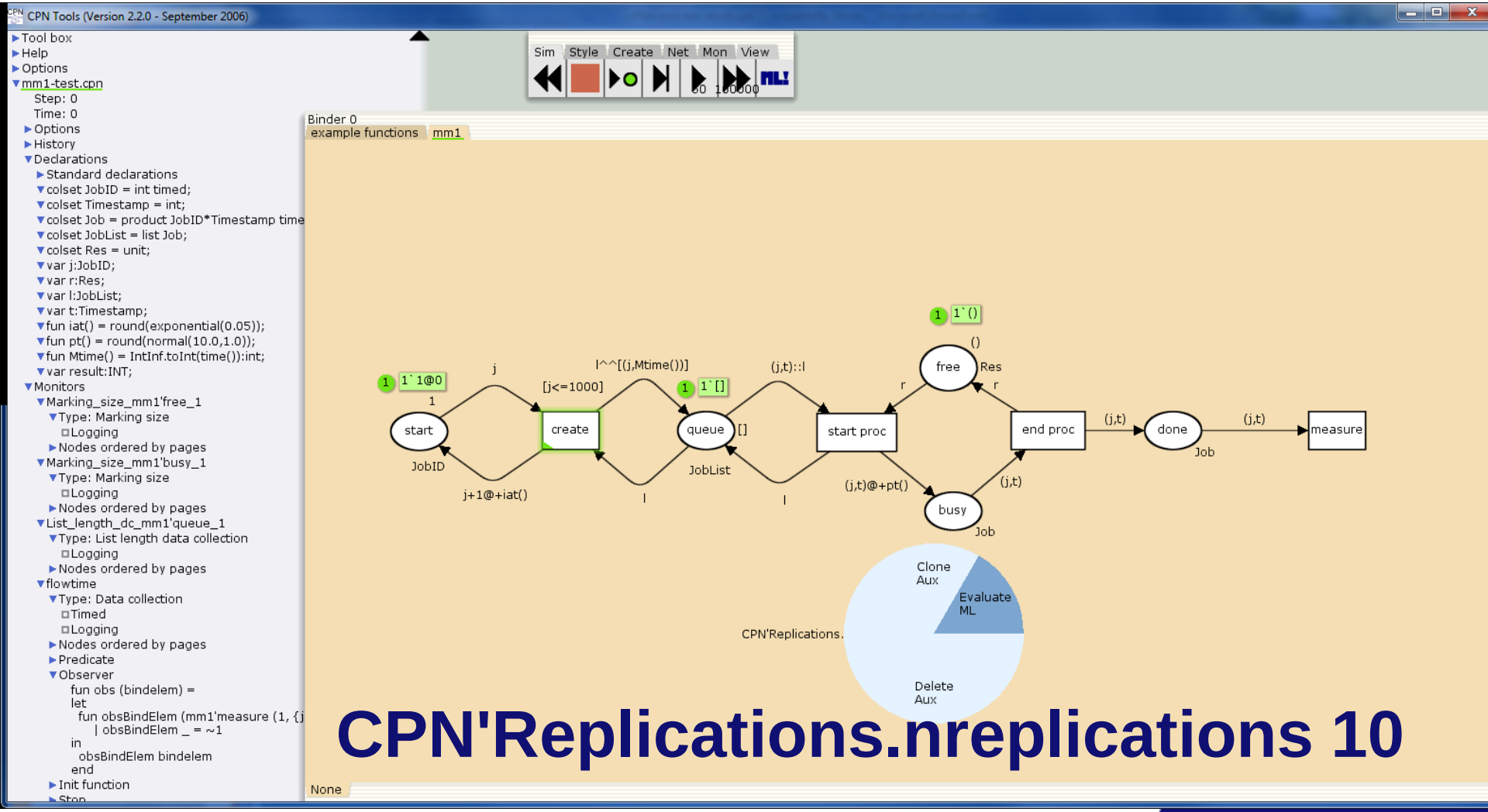



Monitors
Marking_size_mm1'free_1
Type: Marking size
Logging
Nodes ordered by pages
Marking_size_mm1'busy_1
Type: Marking size
Logging
Nodes ordered by pages
List_length_dc_mm1'queue_1
Type: List length data collection
Logging
Nodes ordered by pages
flowtime
Type: Data collection
Timed
Logging
Nodes ordered by pages
Predicate
Observer
fun obs (bindelem) =
let
fun obsBindElem (mm1'measure (1, {j,t})) = Mtime() - t
| obsBindElem _ = ~1
in
obsBindElem bindelem
end

One run

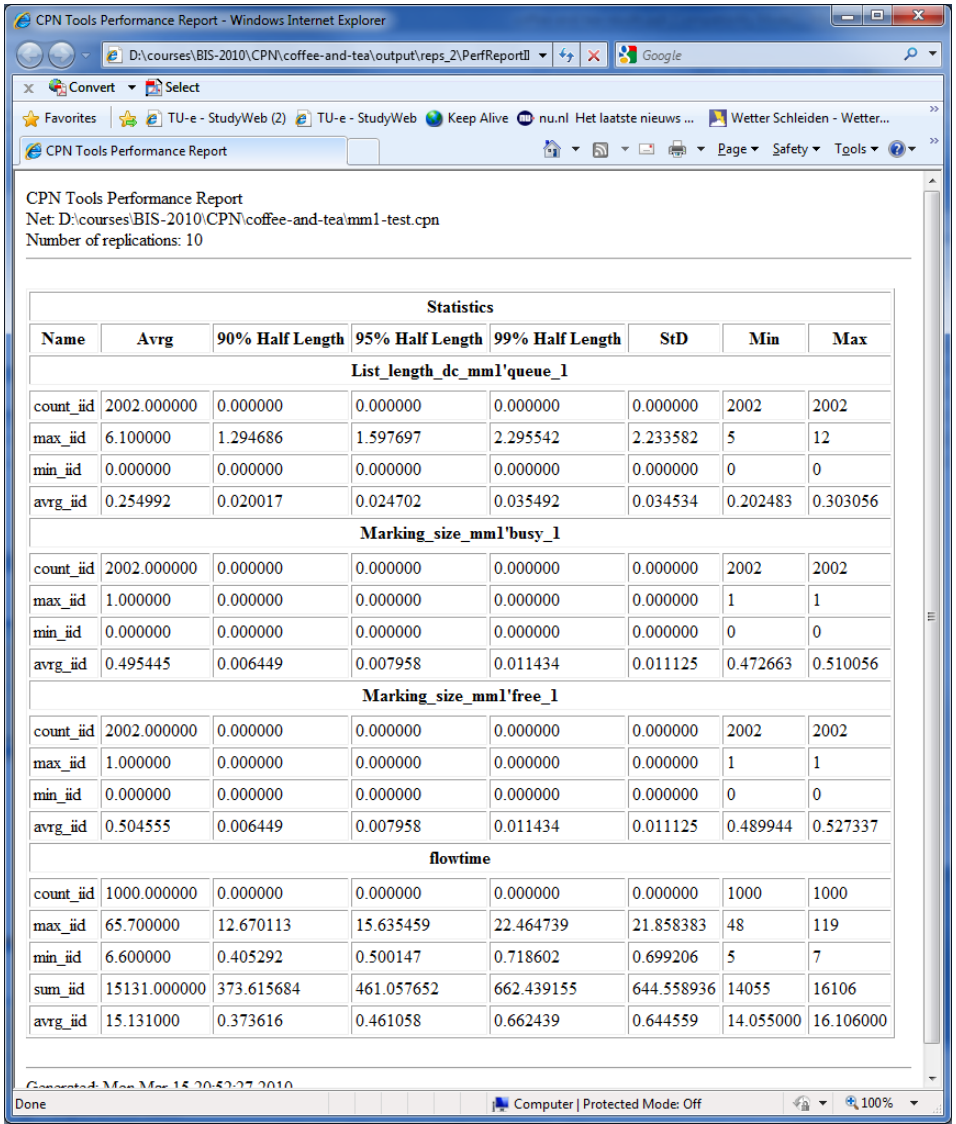

CPN Tools Simulation Performance Report - Windows Internet Explorer
D:\courses\BIS-2010\CPN\coffee-and-tea\outpu
Google
Convert
Select
Favorites
TU-e - StudyWeb (2)
TU-e - StudyWeb
Keep Alive
nu.nl Het laatste nieuws ...
CPN Tools Simulation Performance Report
Page
Safety
Tools
CPN Tools Simulation Performance Report
Net: D:\courses\BIS-2010\CPN\coffee-and-tea\mm1-test.cpn
Note that these statistics have been calculated for data that is not necessarily independent or identically distributed.
Timed statistics
Name | Count | Avrg | Min | Max
List_length_dc_mm1'queue_1 | 2002 | 0.319276 | 0 | 6
Marking_size_mm1'busy_1 | 2002 | 0.508891 | 0 | 1
Marking_size_mm1'free_1 | 2002 | 0.491109 | 0 | 1
Untimed statistics
Name | Count | Sum | Avrg | Min | Max
flowtime | 1000 | 16300 | 16.300000 | 7 | 71
Simulation steps executed: 4000
Model time: 19682
Generated: Mon Mar 15 20:51:11 2010
Done
Computer | Protected Mode: Off
100%

Multiple subruns



CPN'Replications.nreplications 10

Results with confidence intervals



CPN Tools Performance Report
Net: D:\courses\BIS-2010\CPN\coffee-and-tea\mm1-test.cpn
Number of replications: 10

Statistics

Name	Avrg	90% Half Length	95% Half Length	99% Half Length	StD	Min	Max
List_length_dc_mm1'queue_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	6.100000	1.294686	1.597697	2.295542	2.233582	5	12
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
avrg_iid	0.254992	0.020017	0.024702	0.035492	0.034534	0.202483	0.303056
Marking_size_mm1'busy_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	1.000000	0.000000	0.000000	0.000000	0.000000	1	1
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
avrg_iid	0.495445	0.006449	0.007958	0.011434	0.011125	0.472663	0.510056
Marking_size_mm1'free_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	1.000000	0.000000	0.000000	0.000000	0.000000	1	1
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
avrg_iid	0.504555	0.006449	0.007958	0.011434	0.011125	0.489944	0.527337
flowtime							
count_iid	1000.000000	0.000000	0.000000	0.000000	0.000000	1000	1000
max_iid	65.700000	12.670113	15.635459	22.464739	21.858383	48	119
min_iid	6.600000	0.405292	0.500147	0.718602	0.699206	5	7
sum_iid	15131.000000	373.615684	461.057652	662.439155	644.558936	14055	16106
avrg_iid	15.131000	0.373616	0.461058	0.662439	0.644559	14.055000	16.106000

Generated: Mon Mar 15 20:52:27 2010

CPN Tools Performance Report
Net: D:\courses\BIS-2010\CPN\coffee-and-tea\mm1-test.cpn
Number of replications: 10

Statistics

Name	Avrg	90% Half Length	95% Half Length	99% Half Length	StD	Min	Max
List_length_dc_mm1'queue_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	6.100000	1.294686	1.597697	2.295542	2.233582	5	12
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
avrg_iid	0.254992	0.020017	0.024702	0.035492	0.034534	0.202483	0.303056
Marking_size_mm1'busy_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	1.000000	0.000000	0.000000	0.000000	0.000000	1	1
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
avrg_iid	0.495445	0.006449	0.007958	0.011434	0.011125	0.472663	0.510056
Marking_size_mm1'free_1							

max_iid	6.100000	1.294686	1.597697	2.295542	2.233582	5	12
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
avrg_iid	0.254992	0.020017	0.024702	0.035492	0.034534	0.202483	0.303056
Marking_size_mm1'busy_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	1.000000	0.000000	0.000000	0.000000	0.000000	1	1
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
avrg_iid	0.495445	0.006449	0.007958	0.011434	0.011125	0.472663	0.510056
Marking_size_mm1'free_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	1.000000	0.000000	0.000000	0.000000	0.000000	1	1
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000	0	0
avrg_iid	0.504555	0.006449	0.007958	0.011434	0.011125	0.489944	0.527337
flowtime							
count_iid	1000.000000	0.000000	0.000000	0.000000	0.000000	1000	1000
max_iid	65.700000	12.670113	15.635459	22.464739	21.858383	48	119
min_iid	6.600000	0.405292	0.500147	0.718602	0.699206	5	7
sum_iid	15131.000000	373.615684	461.057652	662.439155	644.558936	14055	16106
avrg_iid	15.131000	0.373616	0.461058	0.662439	0.644559	14.055000	16.106000

Generated: Mon Mar 15 20:52:27 2010

Will be explained in detail …

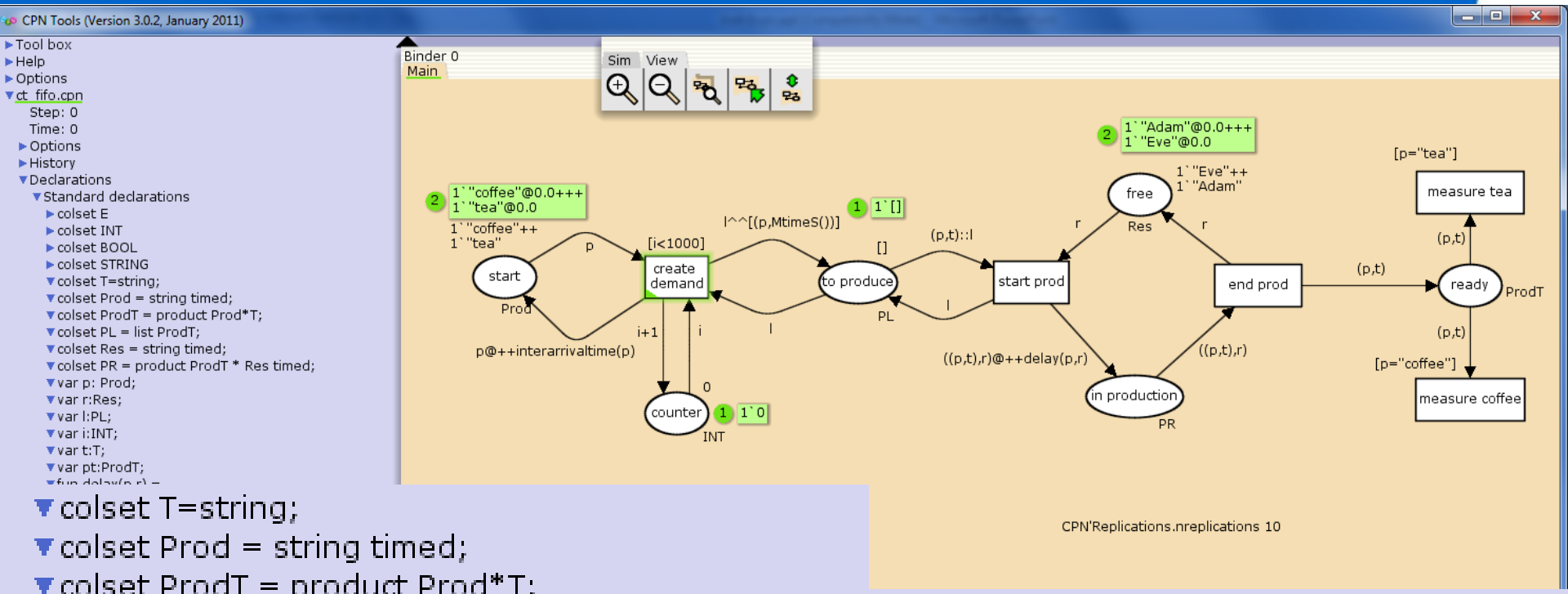
Coffee and tea example (3)

- Assume a continuous flow of tea and coffee drinker, i.e., every 5 minutes there is request for tea and every 10 minutes there is a request of coffee.
- There are two persons able to manufacture these drinks (Adam and Eve) and the production times are as before.
- Process the requests in FIFO (first-in-first-out) order.

Flow

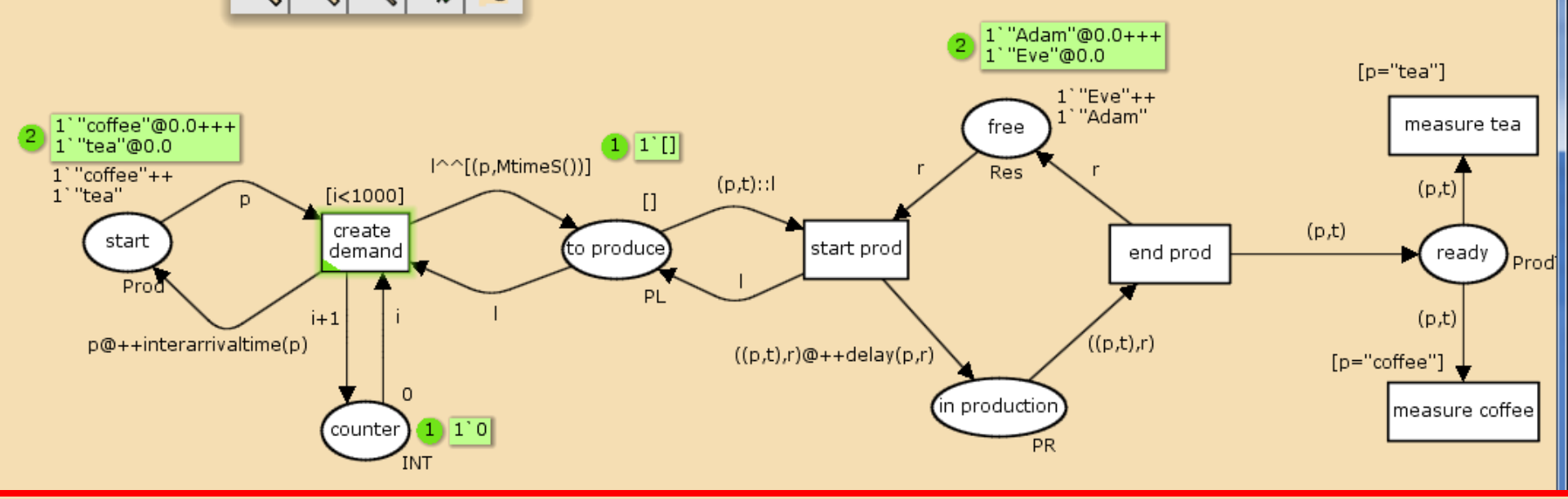
arrival process **processing** **departure process**

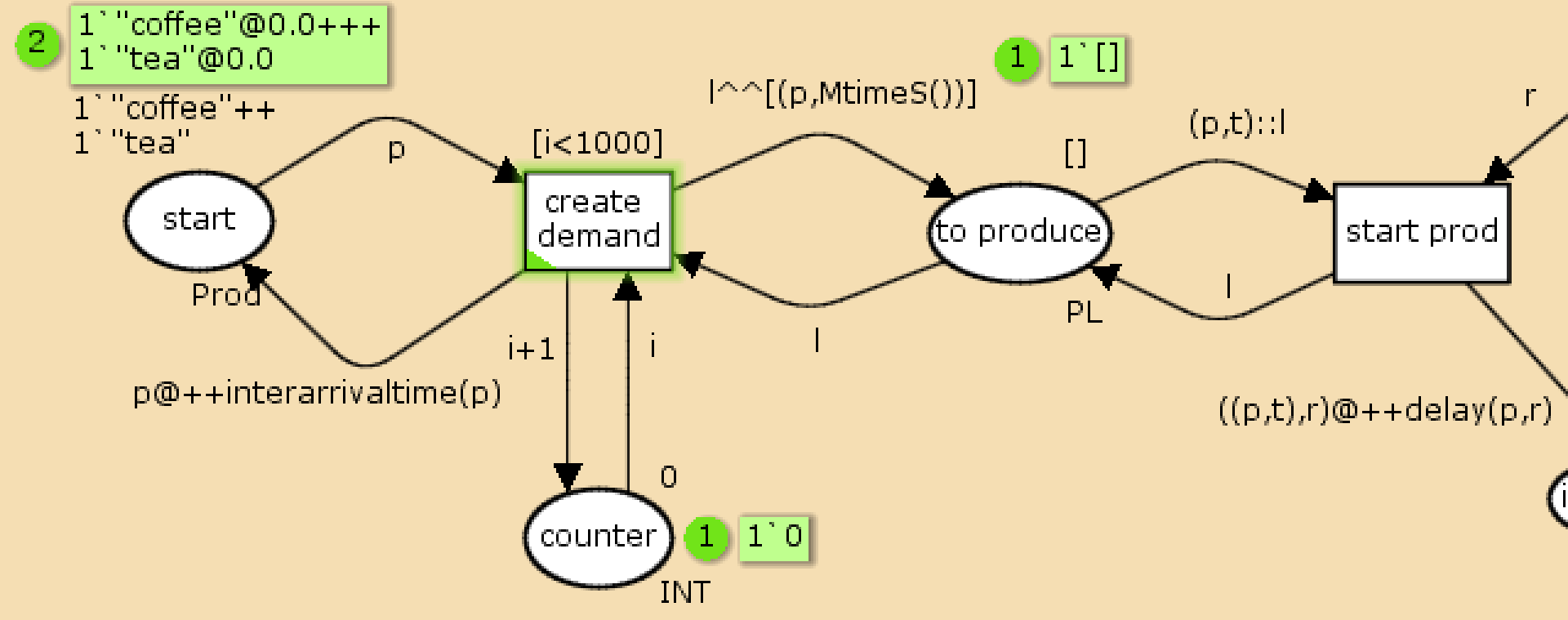
FIFO

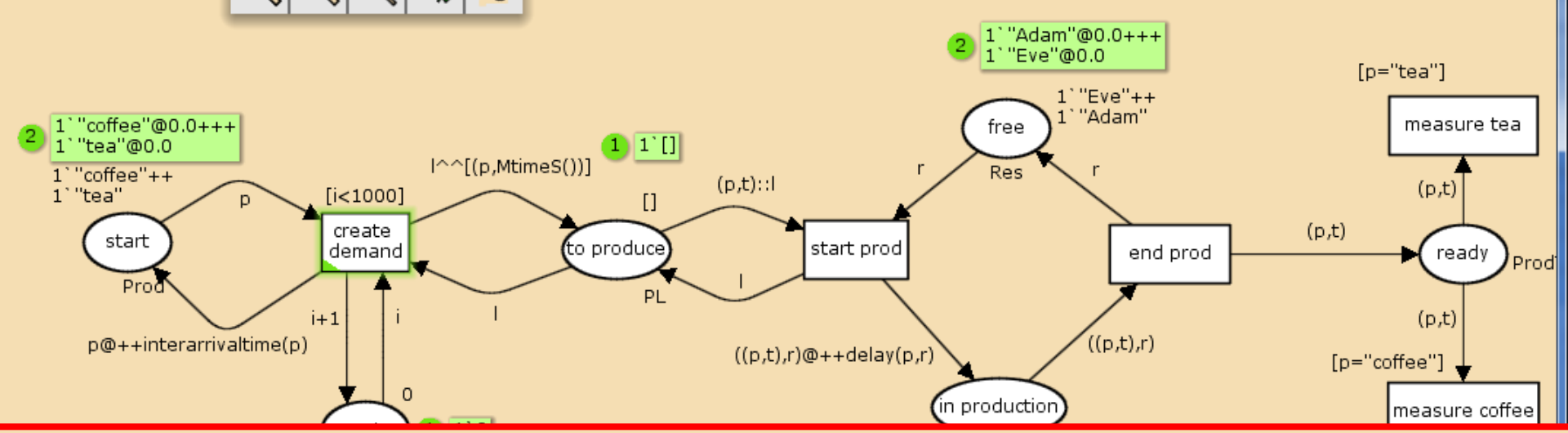


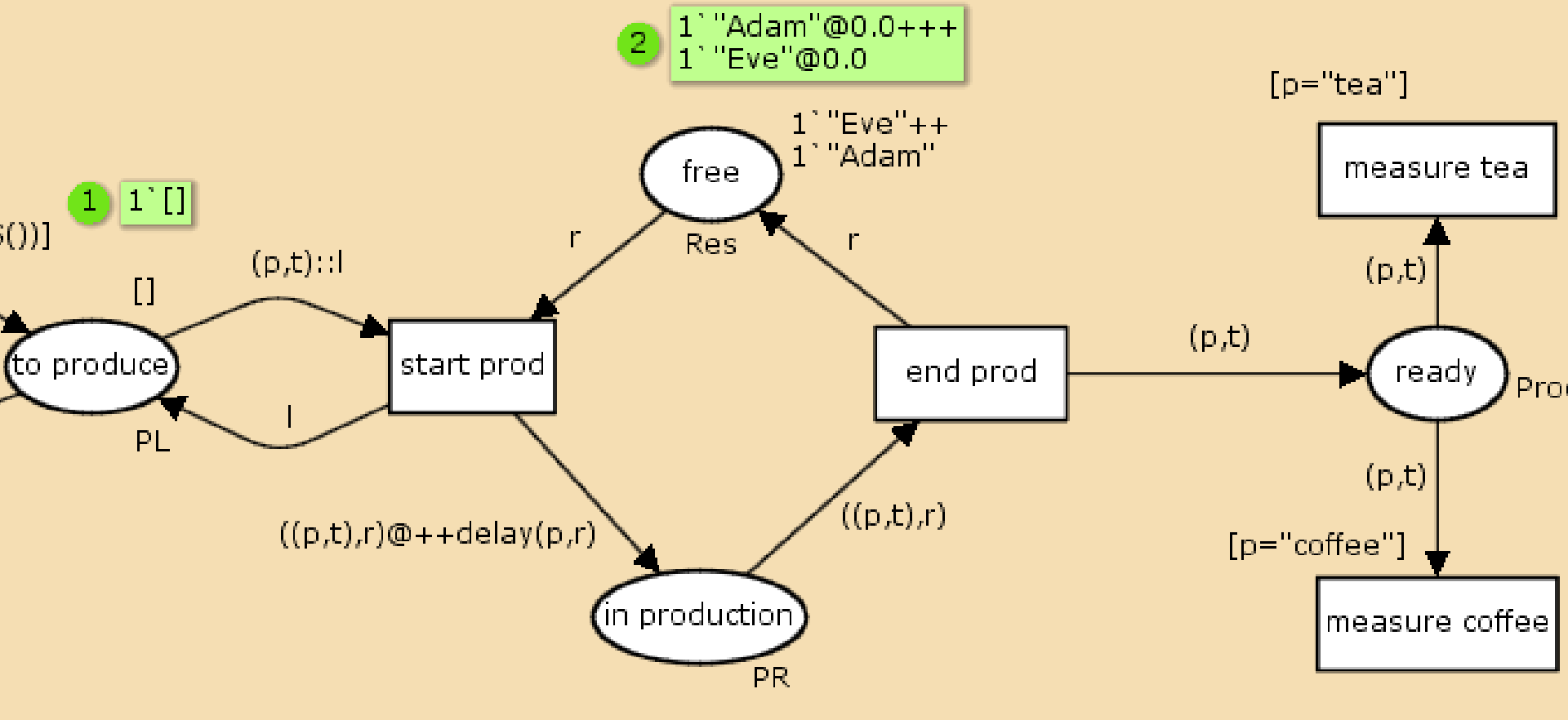

```
colset T=string;
colset Prod = string timed;
colset ProdT = product Prod*T;
colset PL = list ProdT;
colset Res = string timed;
colset PR = product ProdT * Res time
var p: Prod;
var r:Res;
var l:PL;
var i:INT;
var t:T;
var pt:ProdT;
```

```
fun delay(p,r) =
  if p="tea" then
  (if r="Eve" then 2.0 else 6.0) else
  (if r="Eve" then 12.0 else 4.0) ;
fun interarrivaltime(p)=
  if p = "tea" then 5.0 else 10.0;
fun Mtime() = ModelTime.time():time;
fun MtimeS() = ModelTime.toString(Mtime()):string;
fun StoR(t:string) = ModelTime.maketime(t): time;
```


2
1`"coffee"@0.0+++
1`"tea"@0.0
1`"coffee"++
1`"tea"
start
Prod
p
[i<1000]
create
demand
p@++interarrivaltime(p)
i+1
i
0
counter
1
1`0
INT
I^^[(p,MtimeS())]
1
1`[]
[]
to produce
PL
I
(p,t)::I
start prod
r
free
Res
2
1`"Adam"@0.0+++
1`"Eve"@0.0
1`"Eve"++
1`"Adam"
((p,t),r)@++delay(p,r)
in production
PR
((p,t),r)
end prod
(p,t)
[p="tea"]
measure tea
ready
Prod
(p,t)
[p="coffee"]
measure coffee


2
1`"coffee"@0.0+++
1`"tea"@0.0
1`"coffee"++
1`"tea"
start
Prod
p
[i<1000]
create
demand
p@++interarrivaltime(p)
i+1
i
0
counter
1
1`0
INT
I^^[(p,MtimeS())]
1
1`[]
[]
to produce
PL
I
(p,t)::I
start prod
r
((p,t),r)@++delay(p,r)


2
1`"Adam"@0.0+++
1`"Eve"@0.0
2
1`"coffee"@0.0+++
1`"tea"@0.0
1`"coffee"++
1`"tea"
p
[i<1000]
start
create demand
Prod
p@++interarrivaltime(p)
i+1
i
0
l^^[(p,MtimeS())]
l
1
1`[]
[]
to produce
PL
(p,t)::l
l
free
r
Res
r
1`"Eve"++
1`"Adam"
start prod
((p,t),r)@++delay(p,r)
in production
((p,t),r)
end prod
(p,t)
[p="tea"]
measure tea
(p,t)
ready
Prod
(p,t)
[p="coffee"]
measure coffee


2
1`"Adam"@0.0+++
1`"Eve"@0.0
[p="tea"]
1`"Eve"++
1`"Adam"
free
1
1`[]
r
Res
r
measure tea
(p,t)::l
[]
(p,t)
to produce
start prod
end prod
(p,t)
ready
PL
l
((p,t),r)@++delay(p,r)
((p,t),r)
(p,t)
[p="coffee"]
in production
PR
measure coffee

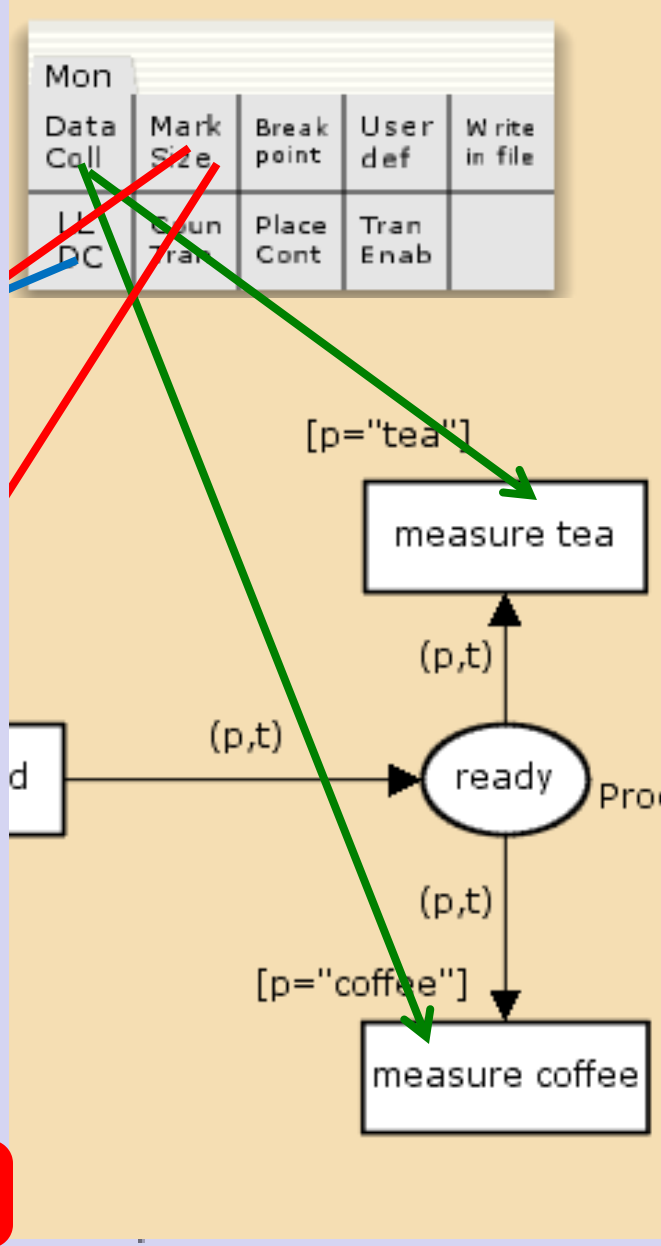
- Monitors
  - Marking_size_Main'free_1
    - Type: Marking size
    - Nodes ordered by pages
  - Marking_size_Main'in production_1
    - Type: Marking size
    - Nodes ordered by pages
  - List_length_dc_Main'to produce_1
    - Type: List length data collection
    - Nodes ordered by pages
  - tea flow time
    - Type: Data collection
    - Nodes ordered by pages
    - Predicate
    - Observer

```
fun obs (bindelem) =
let
  fun obsBindElem (Main'measure_tea (1, {p,t})) = Mtime() - StoR(t)
    | obsBindElem _ = ~1.0
in
  obsBindElem bindelem
end
```

    - Init function
    - Stop
  - coffee flow time
    - Type: Data collection
    - Nodes ordered by pages
    - Predicate
    - Observer

```
fun obs (bindelem) =
let
  fun obsBindElem (Main'measure_coffee (1, {p,t})) = Mtime() - StoR(t)
    | obsBindElem _ = ~1.0
in
  obsBindElem bindelem
end
```

    - Init function
    - Stop




```
fun Mtime() = ModelTime.time():time;
fun MtimeS() = ModelTime.toString(Mtime()):string;
fun StoR(t:string) = ModelTime.maketime(t): time;
```

CPN Tools Performance Report

file:///D:/courses/BIS-2011/CPN%20files/coffee-and-tea/output/reps_2/Perf

Apple Yahoo! Google Maps YouTube Wikipedia News (20) Popular

Statistics

Name	Avrg	90% Half Length	95% Half Length	99% Half Length	StD	Min	Max
List_length_dc_Main'to_produce_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	6.400000	0.680381	0.839619	1.206349	1.173788		
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000		
avrg_iid	0.739548	0.094939	0.117159	0.168332	0.163789	0.589205	1.103943
Marking_size_Main'free_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	2.000000	0.000000	0.000000	0.000000	0.000000		
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000		
avrg_iid	0.368205	0.027995	0.034547	0.049637	0.048297	0.277180	0.413070
Marking_size_Main'in_production_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	2.000000	0.000000	0.000000	0.000000	0.000000		
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000		
avrg_iid	1.631795	0.027995	0.034547	0.049637	0.048297	1.586930	1.722820
coffee_flow_time							
count_iid	333.200000	0.244400	0.301600	0.433333	0.421637	333	334
avrg_iid	10.485863	0.459849	0.567474	0.815336	0.793329		
max_iid	27.600000	2.102408	2.594461	3.727674	3.627059		
min_iid	4.000000	0.000000	0.000000	0.000000	0.000000	4.000000	4.000000
sum_iid	3494.000000	154.343970	190.467026	273.659521	266.273043	3260.000000	4061.000000
tea_flow_time							
count_iid	666.800000	0.244400	0.301600	0.433333	0.421637	666	667
avrg_iid	6.632286	0.386403	0.476837	0.685111	0.666619		
max_iid	23.000000	2.116566	2.611933	3.752777	3.651484		
min_iid	2.000000	0.000000	0.000000	0.000000	0.000000	2.000000	2.000000
sum_iid	4422.300000	256.812836	316.917968	455.341908	443.051552	4024.000000	5403.000000

FIFO

average queue length = 0.74+/-0.09

average utilization = (2-(0.37+/-0.03))/2

average utilization = (1.63+/-0.03)/2

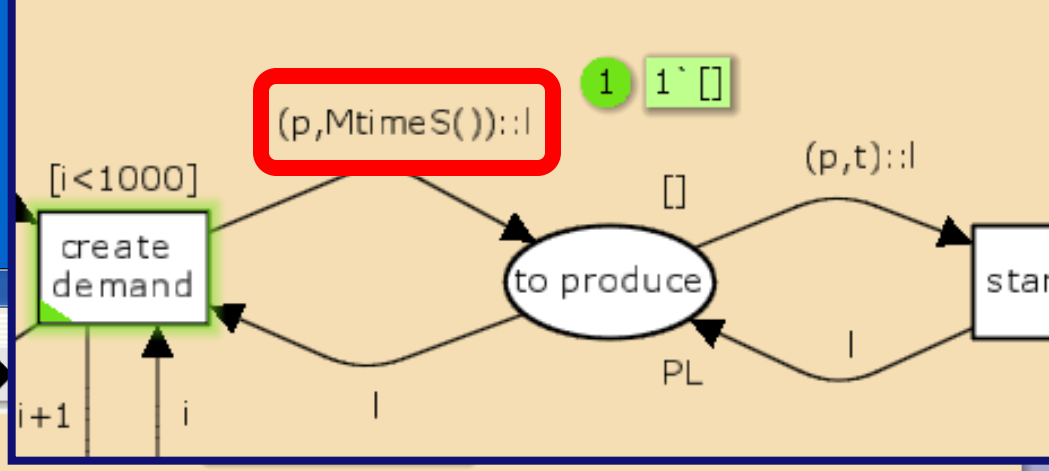
average flow time coffee= 10.49+/-0.46

average flow time tea= 6.63+/-0.39

Coffee and tea example (4)

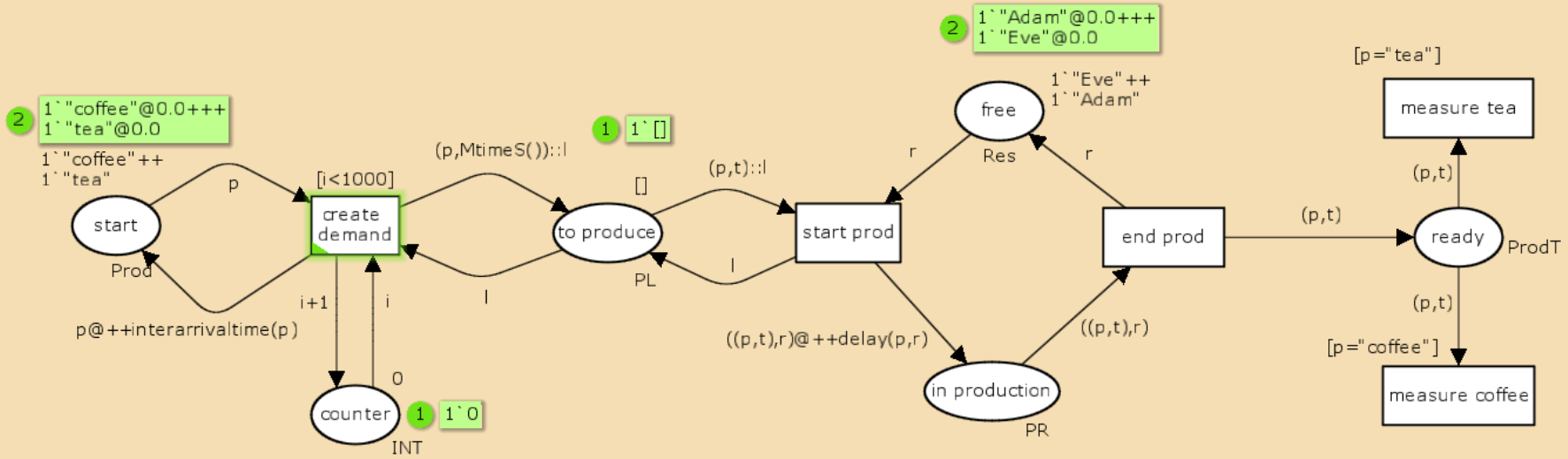
- **Assume a continuous flow of tea and coffee drinker, but now evaluate the following alternatives:**
  - **LIFO (last-in-first-out) order**
  - **SPT (tea before coffee) order**
  - **FIFO with Eve preferably working on tea and Adam on coffee.**
- **Test also your own strategy.**

LIFO



CPN Tools (Version 3.0.2, January 2011)

Tool box
Help
Options
ct_lifo
Binder 0
Main
Sim
View



CPN'Replications.nreplications 10

None

CPN Tools Performance Report

file:///D:/courses/BIS-2011/CPN%20files/coffee-and-tea/output/reps_3/Perf

Google

Apple Yahoo! Google Maps YouTube Wikipedia News (10) Popular

Statistics

Name	Avrg	90% Half Length	95% Half Length	99% Half Length	StD	Min	Max
List_length_dc_Main'to_produce_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	4.900000	0.794300	0.980200	1.408333	1.370320		
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000		
avrg_iid	0.654748	0.082484	0.101788	0.146248	0.142300	0.489820	1.002095
Marking_size_Main'free_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	2.000000	0.000000	0.000000	0.000000	0.000000		
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000		
avrg_iid	0.353071	0.019841	0.024484	0.035178	0.034229	0.311191	0.421557
Marking_size_Main'in_production_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	2.000000	0.000000	0.000000	0.000000	0.000000		
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000		
avrg_iid	1.646929	0.019841	0.024484	0.035178	0.034229	1.578443	1.688809
coffee_flow_time							
count_iid	333.700000	0.279995	0.345526	0.496446	0.483046	333	334
avrg_iid	10.194212	0.391368	0.482965	0.693916	0.675186		
max_iid	91.500000	51.205482	63.189744	90.789862	88.339308		
min_iid	4.000000	0.000000	0.000000	0.000000	0.000000	4.000000	4.000000
sum_iid	3401.900000	131.594550	162.393274	233.323670	227.025916	3099.000000	3921.000000
tea_flow_time							
count_iid	666.300000	0.279995	0.345526	0.496446	0.483046	666	667
avrg_iid	6.433586	0.307211	0.379112	0.544701	0.529998		
max_iid	114.500000	47.711433	58.877938	84.594739	82.311401		
min_iid	2.000000	0.000000	0.000000	0.000000	0.000000	2.000000	2.000000
sum_iid	4286.700000	204.666040	252.566602	362.883049	353.088296	3809.000000	5072.000000

LIFO

average queue length = 0.65+/-0.08

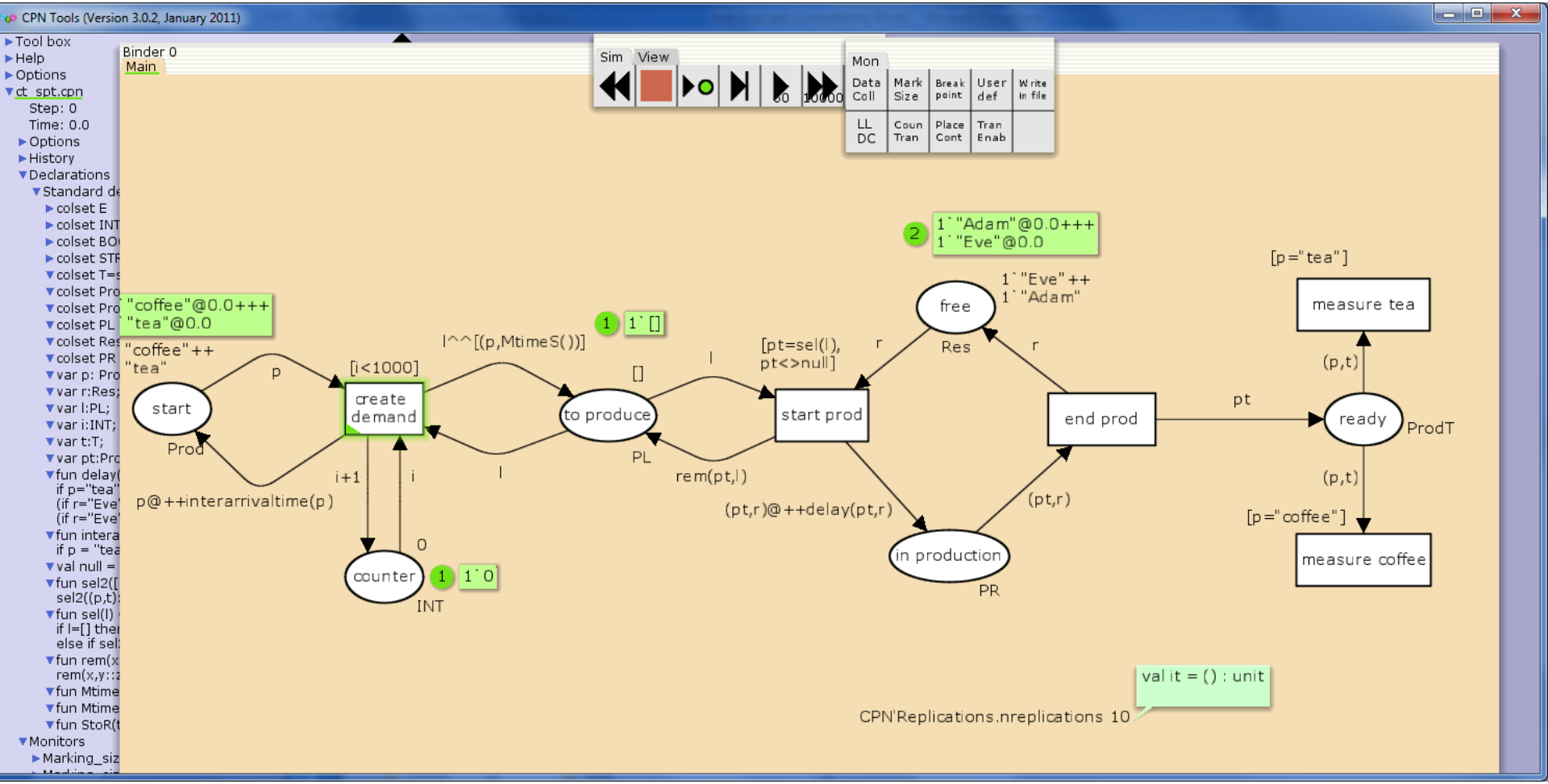
average utilization = (2-(0.35+/-0.02))/2

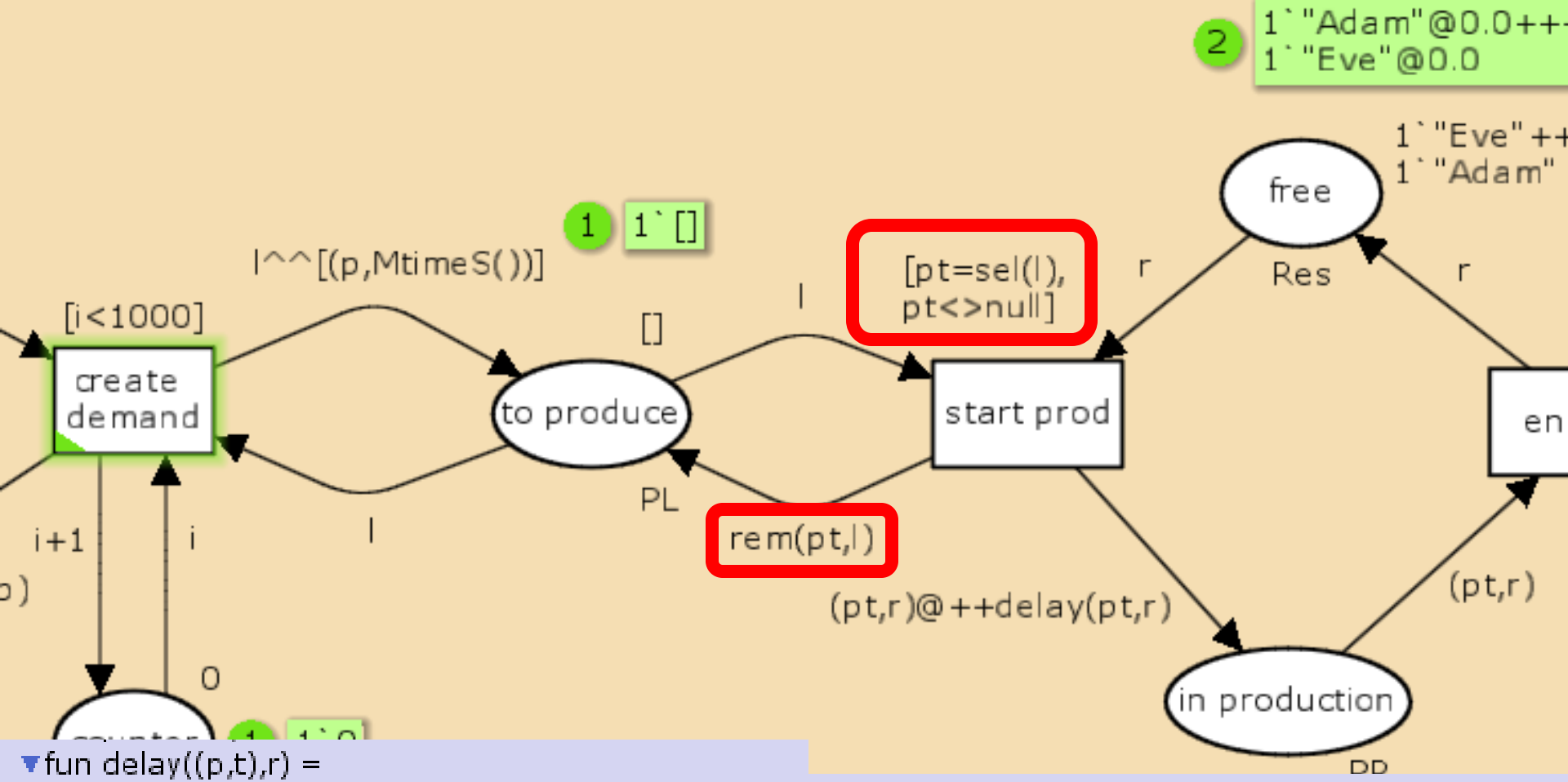
average utilization = (1.65+/-0.02)/2

average flow time coffee= 10.19+/-0.39

average flow time tea= 6.43+/-0.31

SPT


CPN Tools (Version 3.0.2, January 2011)
Tool box
Help
Options
ct_spt.cpn
Step: 0
Time: 0.0
Options
History
Declarations
Monitors
Binder 0
Main
Sim
View
Mon
Data Coll
Mark Size
Break point
User def
Write in file
LL DC
Coun Tran
Place Cont
Tran Enab
"coffee"@0.0+++
"tea"@0.0
"coffee"++
"tea"
start
Prod
p
[i<1000]
create demand
i+1
i
0
counter
1`0
INT
p@++interarrivaltime(p)
l^^[(p,MtimeS())]
1`[]
[]
to produce
PL
l
[pt=sel(l), pt<>null]
start prod
rem(pt,l)
(pt,r)@++delay(pt,r)
in production
PR
(pt,r)
2
1`"Adam"@0.0+++
1`"Eve"@0.0
free
Res
r
1`"Eve"++
1`"Adam"
end prod
pt
[p="tea"]
measure tea
(p,t)
ready
ProdT
(p,t)
[p="coffee"]
measure coffee
val it = () : unit
CPN'Replications.nreplications 10


2
1`"Adam"@0.0++
1`"Eve"@0.0
1`"Eve"++
1`"Adam"
free
Res
r
r
1
1`[]
l^^[(p,MtimeS())]
[i<1000]
create demand
[]
to produce
l
[pt=sel(l), pt<>null]
start prod
en
PL
rem(pt,l)
(pt,r)@++delay(pt,r)
(pt,r)
i+1
i
l
0
in production
fun delay((p,t),r) =
if p="tea" then
(if r="Eve" then 2.0 else 6.0) else
(if r="Eve" then 12.0 else 4.0) ;
fun interarrivaltime(p)=
if p = "tea" then 5.0 else 10.0;
val null = ("error","error"):ProdT;
fun sel2([]) =null |
sel2((p,t)::l) = if p = "tea" then (p,t) else sel2(l);
fun sel(l) =
if l=[] then null
else if sel2(l) = null then hd(l) else sel2(l);
fun rem(x,[]) = [] |
rem(x,y::z) = if x=y then z else x::rem(x,z);
fun Mtime() = ModelTime.time():time;
fun MtimeS() = ModelTime.toString(Mtime()):string;
fun StoR(t:string) = ModelTime.maketime(t): time;

SPT

Tools Performance Report

file:///D:/courses/BIS-2011/CPN%20files/coffee-and-tea/output/reps_4/Perf

Apple Yahoo! Google Maps YouTube Wikipedia News (13) Popular

Statistics

ne	Avrg	90% Half Length	95% Half Length	99% Half Length	StD	Min	Max
List_length_dc_Main'to_produce_1							
_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
iid	3.000000	0.000000	0.000000	0.000000	0.000000	3	
id	0.000000	0.000000	0.000000	0.000000	0.000000	0	
iid	0.234327	0.017611	0.021733	0.031225	0.030383	0.183508	0.296529
Marking_size_Main'free_1							
_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
iid	2.000000	0.000000	0.000000	0.000000	0.000000	2	
id	0.000000	0.000000	0.000000	0.000000	0.000000	0	
iid	0.589351	0.021186	0.026144	0.037564	0.036550	0.528426	0.668666
Marking_size_Main'in_production_1							
_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
iid	2.000000	0.000000	0.000000	0.000000	0.000000	2	
id	0.000000	0.000000	0.000000	0.000000	0.000000	0	
iid	1.410649	0.021186	0.026144	0.037564	0.036550	1.331334	1.471574
coffee_flow_time							
_iid	221.400000	5.264537	6.496663	9.334286	9.082339	201	233
iid	8.437801	0.266928	0.329400	0.473277	0.460502	7.50000	
iid	16.000000	0.000000	0.000000	0.000000	0.000000	16.0000	
id	4.000000	0.000000	0.000000	0.000000	0.000000	4.000000	4.000000
iid	1866.200000	54.113218	66.778013	95.945421	93.355712	1710.000000	2052.000000
tea_flow_time							
_iid	778.600000	5.264537	6.496663	9.334286	9.082339	767	799
iid	4.634149	0.088778	0.109556	0.157408	0.153160	4.32253	
iid	12.000000	0.000000	0.000000	0.000000	0.000000	12.0000	
id	2.000000	0.000000	0.000000	0.000000	0.000000	2.000000	2.000000
iid	3608.300000	75.660478	93.368250	134.149784	130.528881	3337.000000	3817.000000

average queue length = 0.23+/-0.02

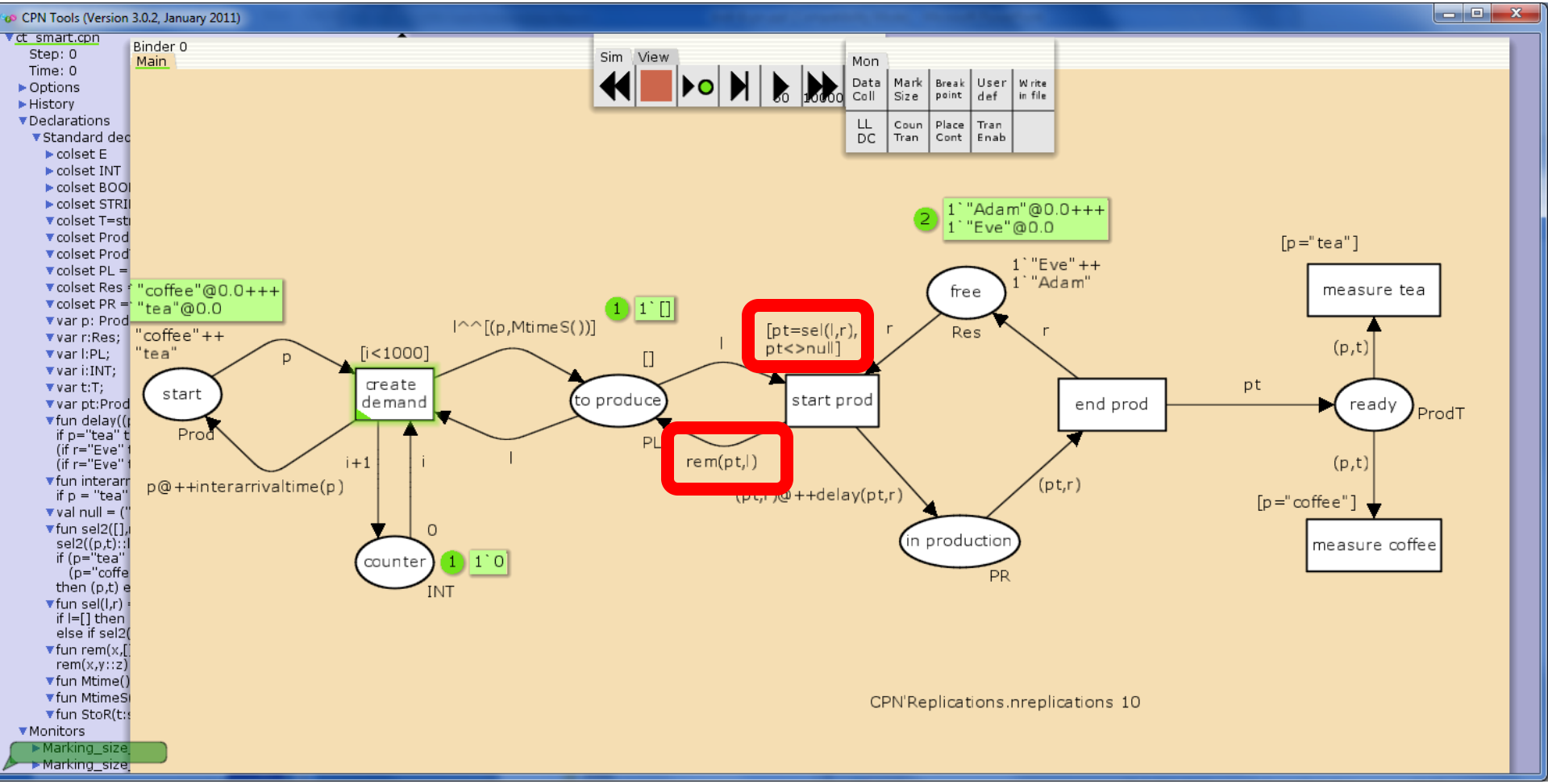
average utilization = (2-(0.59+/-0.02))/2

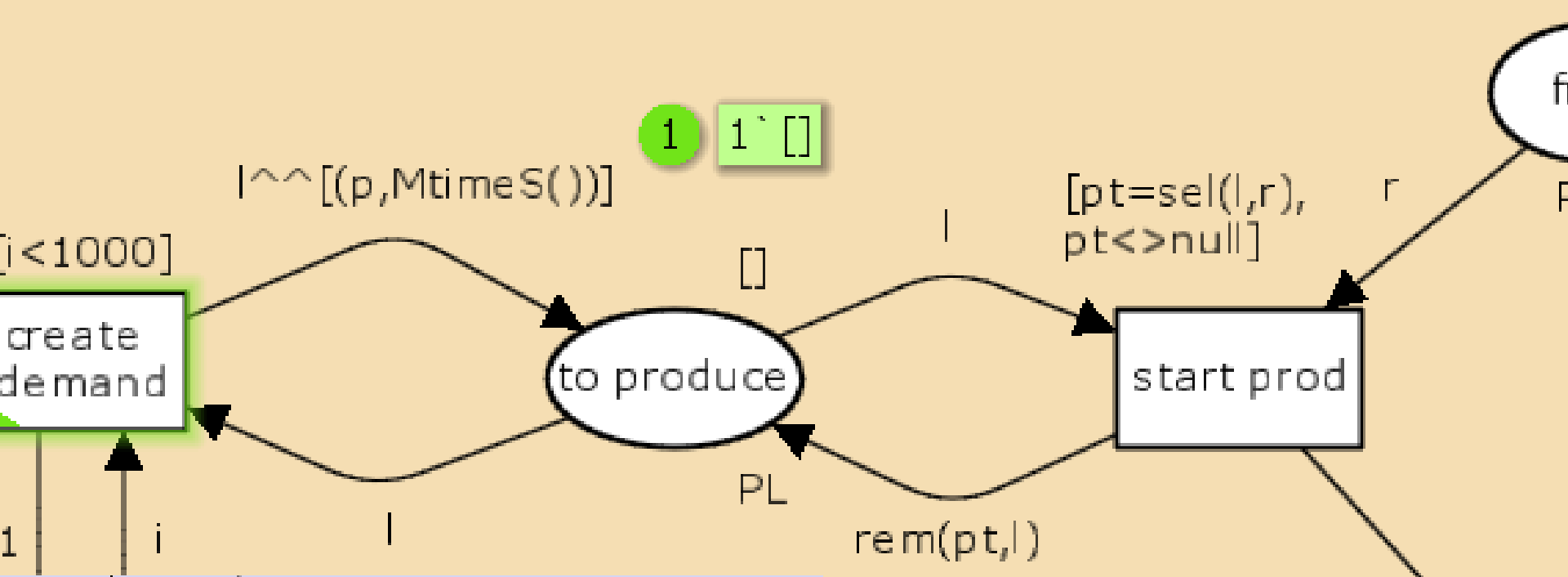
average utilization = (1.41+/-0.02)/2

average flow time coffee= 8.44+/-0.27

average flow time tea= 4.63+/-0.09

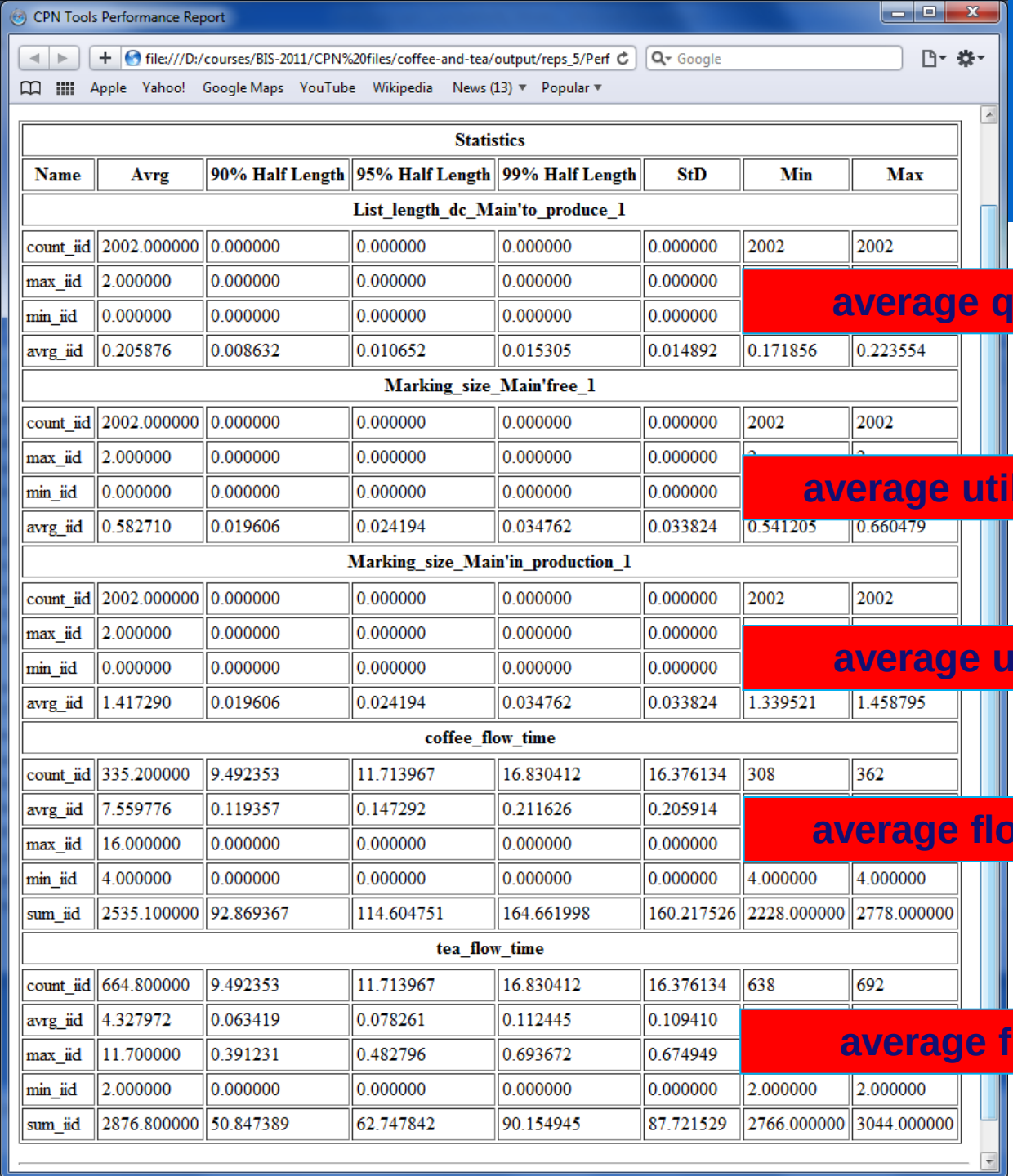
SMART


CPN Tools (Version 3.0.2, January 2011)
Binder 0
Main
Sim
View
Mon
Data Coll
Mark Size
Break point
User def
Write in file
LL DC
Coun Tran
Place Cont
Tran Enab
"coffee"@0.0+++
"tea"@0.0
"coffee"++
"tea"
start
Prod
p
[i<1000]
create demand
i+1
i
p@++interarrivaltime(p)
I^^[(p,MtimeS())]
1 1`[]
[]
to produce
I
PL
[pt=sel(I,r), pt<>null]
rem(pt,I)
start prod
r
free
Res
2 1`"Adam"@0.0+++
1`"Eve"@0.0
1`"Eve"++
1`"Adam"
end prod
pt
ready
ProdT
(p,t)
[p="tea"]
measure tea
[p="coffee"]
measure coffee
(pt,r)
(pt,r)@++delay(pt,r)
in production
PR
counter
1 1`0
0
INT
CPN'Replications.nreplications 10




```
fun delay((p,t),r) =
  if p="tea" then
  (if r="Eve" then 2.0 else 6.0) else
  (if r="Eve" then 12.0 else 4.0) ;
fun interarrivaltime(p)=
  if p = "tea" then 5.0 else
val null = ("error","error"
fun sel2([],r) =null |
  sel2((p,t)::l,r) =
  if (p="tea" andalso r="E
     (p="coffee" andalso r=
  then (p,t) else sel2(l,r);
fun sel(l,r) =
  if l=[] then null
  else if sel2(l,r) = null the
fun rem(x,[]) = [] |
  rem(x,y::z) = if x=y then
```

```
fun sel2([],r) =null |
  sel2((p,t)::l,r) =
  if (p="tea" andalso r="Eve") orelse
     (p="coffee" andalso r="Adam")
  then (p,t) else sel2(l,r);
fun sel(l,r) =
  if l=[] then null
  else if sel2(l,r) = null then hd(l) else sel2(l,r);
```



Statistics

Name	Avrg	90% Half Length	95% Half Length	99% Half Length	StD	Min	Max
List_length_dc_Main'to_produce_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	2.000000	0.000000	0.000000	0.000000	0.000000		
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000		
avrg_iid	0.205876	0.008632	0.010652	0.015305	0.014892	0.171856	0.223554
Marking_size_Main'free_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	2.000000	0.000000	0.000000	0.000000	0.000000		
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000		
avrg_iid	0.582710	0.019606	0.024194	0.034762	0.033824	0.541205	0.660479
Marking_size_Main'in_production_1							
count_iid	2002.000000	0.000000	0.000000	0.000000	0.000000	2002	2002
max_iid	2.000000	0.000000	0.000000	0.000000	0.000000		
min_iid	0.000000	0.000000	0.000000	0.000000	0.000000		
avrg_iid	1.417290	0.019606	0.024194	0.034762	0.033824	1.339521	1.458795
coffee_flow_time							
count_iid	335.200000	9.492353	11.713967	16.830412	16.376134	308	362
avrg_iid	7.559776	0.119357	0.147292	0.211626	0.205914		
max_iid	16.000000	0.000000	0.000000	0.000000	0.000000		
min_iid	4.000000	0.000000	0.000000	0.000000	0.000000	4.000000	4.000000
sum_iid	2535.100000	92.869367	114.604751	164.661998	160.217526	2228.000000	2778.000000
tea_flow_time							
count_iid	664.800000	9.492353	11.713967	16.830412	16.376134	638	692
avrg_iid	4.327972	0.063419	0.078261	0.112445	0.109410		
max_iid	11.700000	0.391231	0.482796	0.693672	0.674949		
min_iid	2.000000	0.000000	0.000000	0.000000	0.000000	2.000000	2.000000
sum_iid	2876.800000	50.847389	62.747842	90.154945	87.721529	2766.000000	3044.000000

SMART

average queue length = 0.21+/-0.01

average utilization = (2-(0.58+/-0.02))/2

average utilization = (1.42+/-0.02)/2

average flow time coffee= 7.56+/-0.12

average flow time tea= 4.33+/-0.06

Compare (1/2)

FIFO
LIFO
SPT
SMART

average queue length = 0.74+/-0.09

average queue length = 0.65+/-0.08

average queue length = 0.23+/-0.02 ++

average queue length = 0.21+/-0.01 ++

average utilization = (1.63+/-0.03)/2

average utilization = (1.65+/-0.02)/2

average utilization = (1.41+/-0.02)/2

average utilization = (1.42+/-0.02)/2

Compare (2/2)

FIFO
LIFO
SPT
SMART

average flow time coffee= 10.49+/-0.46

average flow time coffee= 10.19+/-0.39

average flow time coffee= 8.44+/-0.27 +

average flow time coffee= 7.56+/-0.12 ++

average flow time tea= 6.63+/-0.39

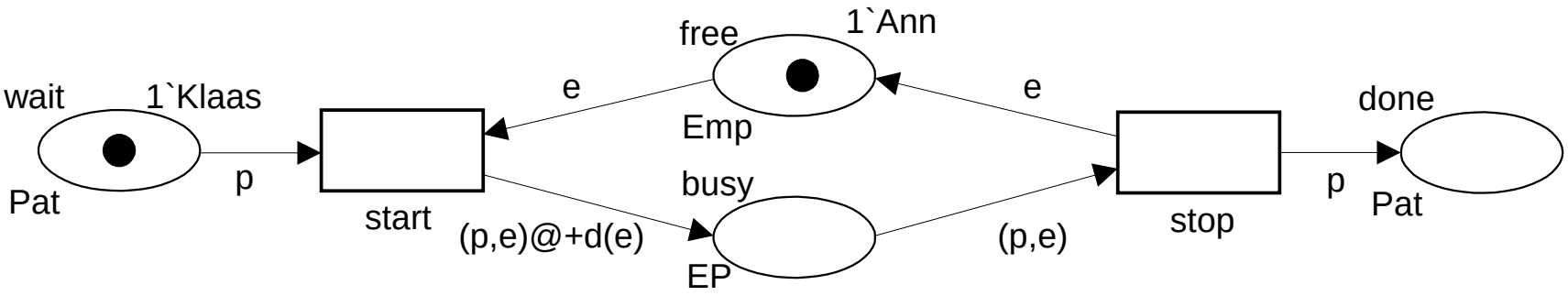
average flow time tea= 6.43+/-0.31

average flow time tea= 4.63+/-0.09 +

average flow time tea= 4.33+/-0.06 ++

Example revisited: Punch card desk

```
color STR = string;
color INT = int;
color Pat = record Name:STR * Address:STR *
                   DateOfBirth:STR * Gender:STR;
color Emp = record EmpNo:INT * Experience:INT;
color EP = product Pat * Emp;
var p:Pat;
var e:Emp;
val Klaas = {Name="Klaas", Address="Plein 10",
                    DateOfBirth="13-Dec-1962", Gender="M"};
val Ann = {EmpNo=641112, Experience=7};
fun d(e:Emp) = if #Experience(e) > 5 then 3 else 4;
```



Improved color sets

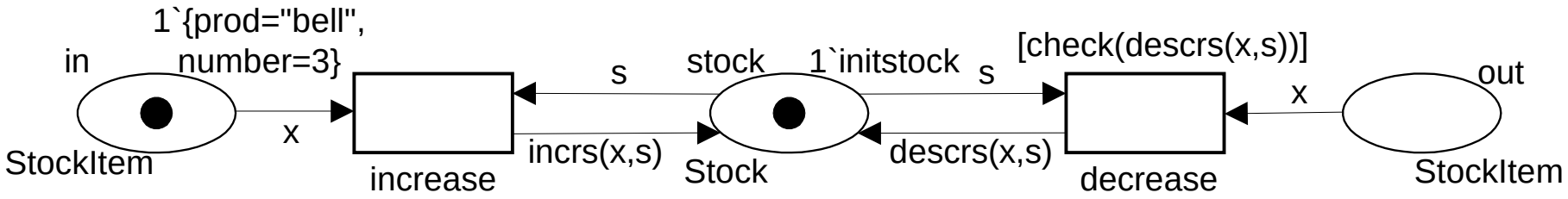
```
color Name = string;
color Street = string;
color Number = int;
color Town = string;
color Address = record s:Street * n:Number *
   t:Town;
color Day = int with 1..31;
color Month = with Jan | Feb | Mar | Apr | May | Jun |
                  Jul | Aug | Sep | Oct | Nov | Dec;
color Year = int with 0..2100;
color Date = record d:Day * m:Month * y:Year;
color Gender = with male | female;
color Pat = record name:Name * address:Address *
                  birthdate:Date * gender:Gender timed
```

Improved color sets (2)

```
color EmpNo = int with 100000..999999;
color Emp = record empno:EmpNo *
   experience:Year timed;
color EP = product Pat * Emp timed;
var p:Pat;
var e:Emp;
val Klaas = {name="Klaas",
            address={s="Plein",n=10,t="Unknown"},
            birthdate={d=13,m=Dec,y=1962},
            gender=male}:Pat;
val Ann = {empno=641112, experience=7}:Emp;
fun d(x:Emp) = if #experience(x) > 5 then 3 else 4;
```

Example revisited: Stock keeping system

```
color Product = string;
color Number = int;
color StockItem = record prod:Product * number:Number;
color Stock = list StockItem;
var x:StockItem;
var s:Stock;
fun incrs(x:StockItem,s:Stock) = if s=[] then [x] else (if (#prod(hd(s)))=(#prod(x))
   then {prod=(#prod(hd(s))),number=((#number(hd(s)))+(#number(x)))}::tl(s)
   else hd(s):: incrs(x,tl(s)));
fun decrs(x:StockItem,s:Stock)= incrs({prod=(#prod(x)),number=(~(#number(x)))},s);
fun check(s:Stock)= if s=[] then true else if (#number(hd(s)))<0 then false
  else check(tl(s));
val initstock = [{prod="bike", number=4},{prod="wheel", number=2},
  {prod="bell", number=3}, {prod="steering wheel", number=3},
  {prod="frame", number=2}];
```



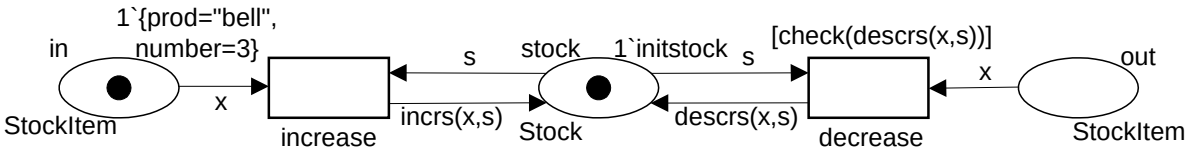
Store

- Place stock is a so-called store, i.e., it will always contain a single token.
- Only the value of the token matters (not its presence).
- Stores that aggregate elements are always of type list.
- Drawback: complex functions/inscriptions
- Advantage: easy to query the individual items as a whole, e.g., taking the sum of things ...

Function "totalstock"

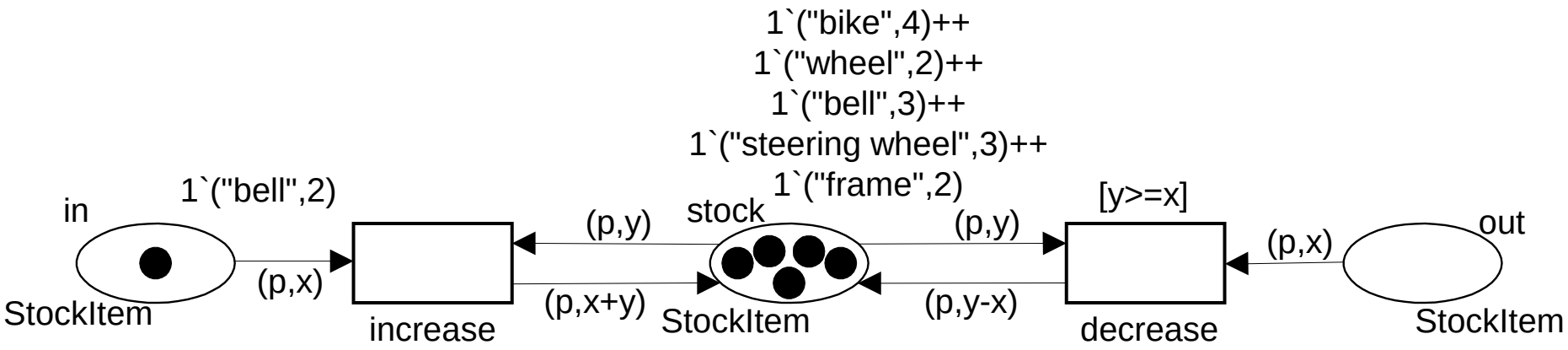
```
fun totalstock(s:Stock) =
  if s=[ ]
    then 0
    else (#number(hd(s)))+totalstock(tl(s));
```

```
color Product = string;
color Number = int;
color StockItem = record prod:Product * number:Number;
color Stock = list StockItem;
var x:StockItem;
var s:Stock;
fun incrs(x:StockItem,s:Stock) = if s=[] then [x] else (if (#prod(hd(s)))=(#prod(x))
   then {prod=(#prod(hd(s))),number=((#number(hd(s)))+(#number(x)))}::tl(s)
   else hd(s):: incrs(x,tl(s)));
fun decrs(x:StockItem,s:Stock)= incrs({prod=(#prod(x)),number=(~(#number(x)))},s);
fun check(s:Stock)= if s=[] then true else if (#number(hd(s)))<0 then false
  else check(tl(s));
val initstock = [{prod="bike", number=4},{prod="wheel", number=2},
  {prod="bell", number=3}, {prod="steering wheel", number=3},
  {prod="frame", number=2}];
```



Alternative model

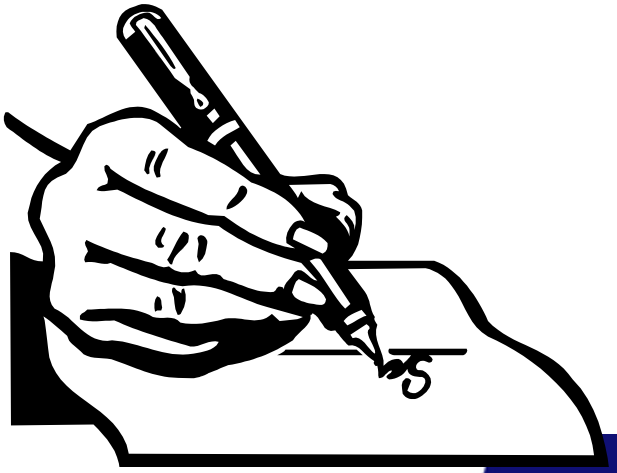
```
color Product = string;
color Number = int;
color StockItem = product Product*Number;
var p:Product;
var x:Number;
var y:Number;
```



Note the simplicity/elegance of the arc inscriptions.

Example: Signing documents

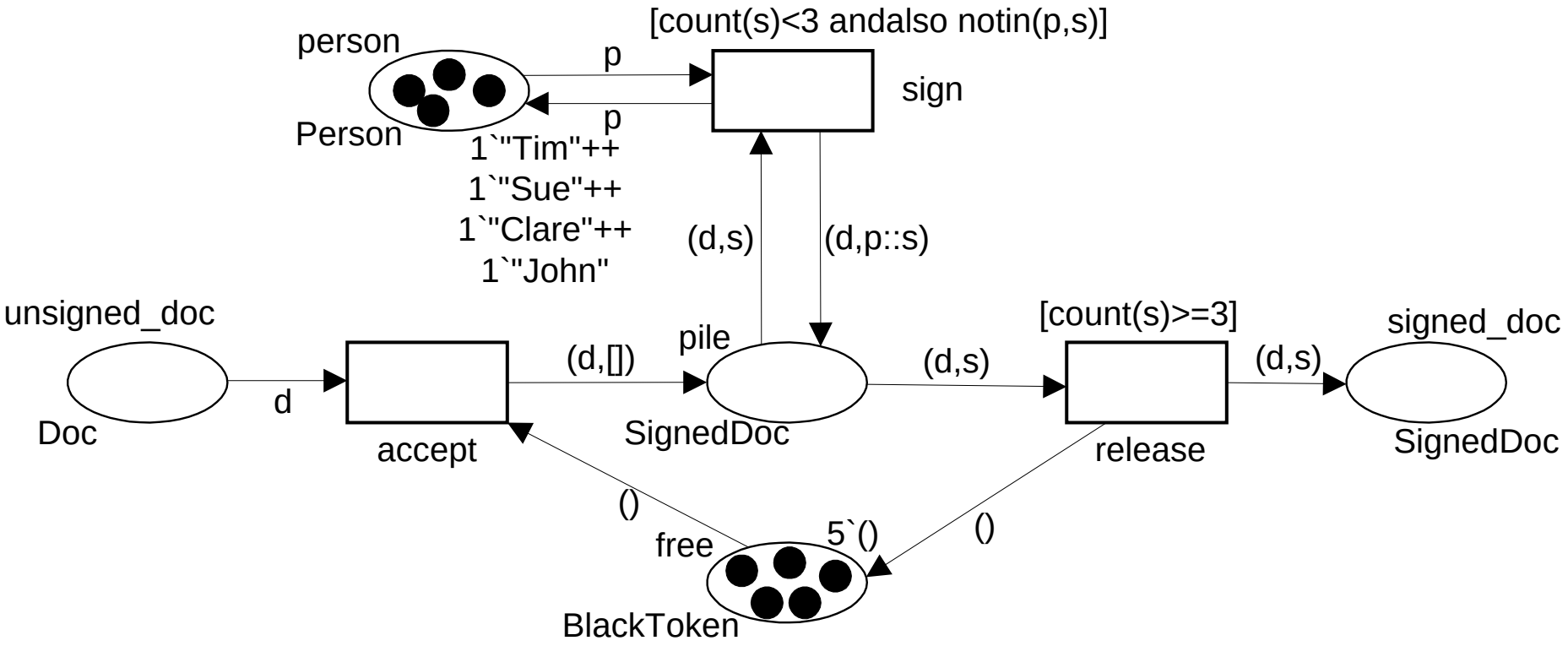
- Documents need to be signed by persons.
- Four persons: Tim, Sue, Clare and John.
- Each document requires three signatures.
- No two signatures of the same person.
- Work in progress is limited to five documents.



Signing documents: Declarations

```
color Doc = string;
color Person = string;
color Signatures = list Person;
color SignedDoc = product Doc * Person;
color BlackToken = unit;
var d:Doc;
var p:Person;
var s:Signatures;
fun notin(p:Person,s:Signatures) =
if s=[] then true else if p=hd(s) then false else notin(p,tl(s));
fun count(s:Signatures) = if s=[] then 0 else 1+count(tl(s));
```

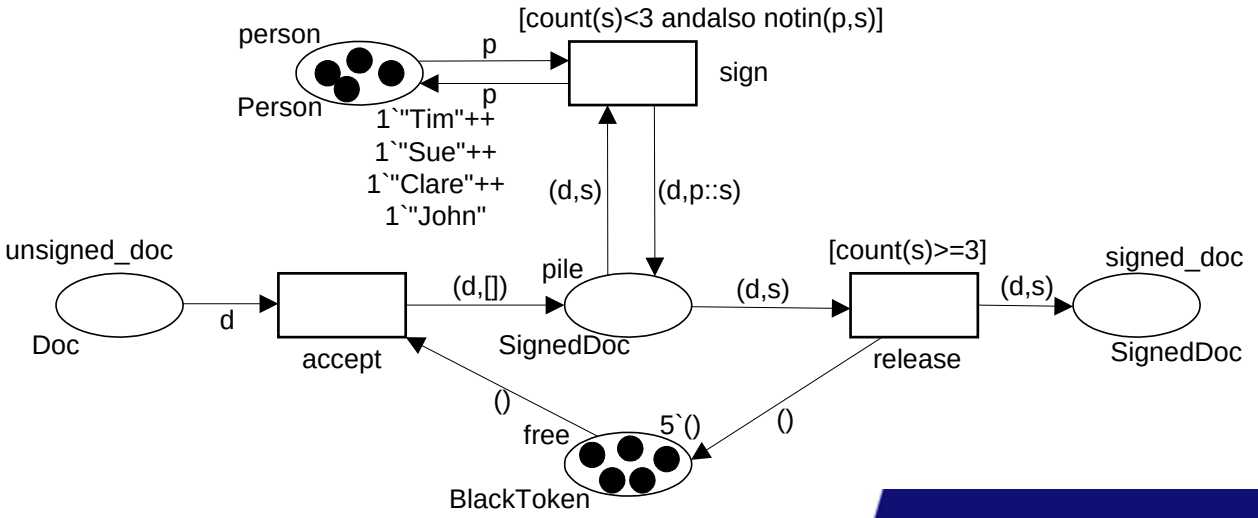
Signing documents: Network structure



Exercise

- **Replace place free by a place always holding one token.**

```
color Doc = string;
color Person = string;
color Signatures = list Person;
color SignedDoc = product Doc * Person;
color BlackToken = unit;
var d:Doc;
var p:Person;
var s:Signatures;
fun notin(p:Person,s:Signatures) =
if s=[] then true else if p=hd(s) then false else notin(p,tl(s));
fun count(s:Signatures) = if s=[] then 0 else 1+count(tl(s));
```

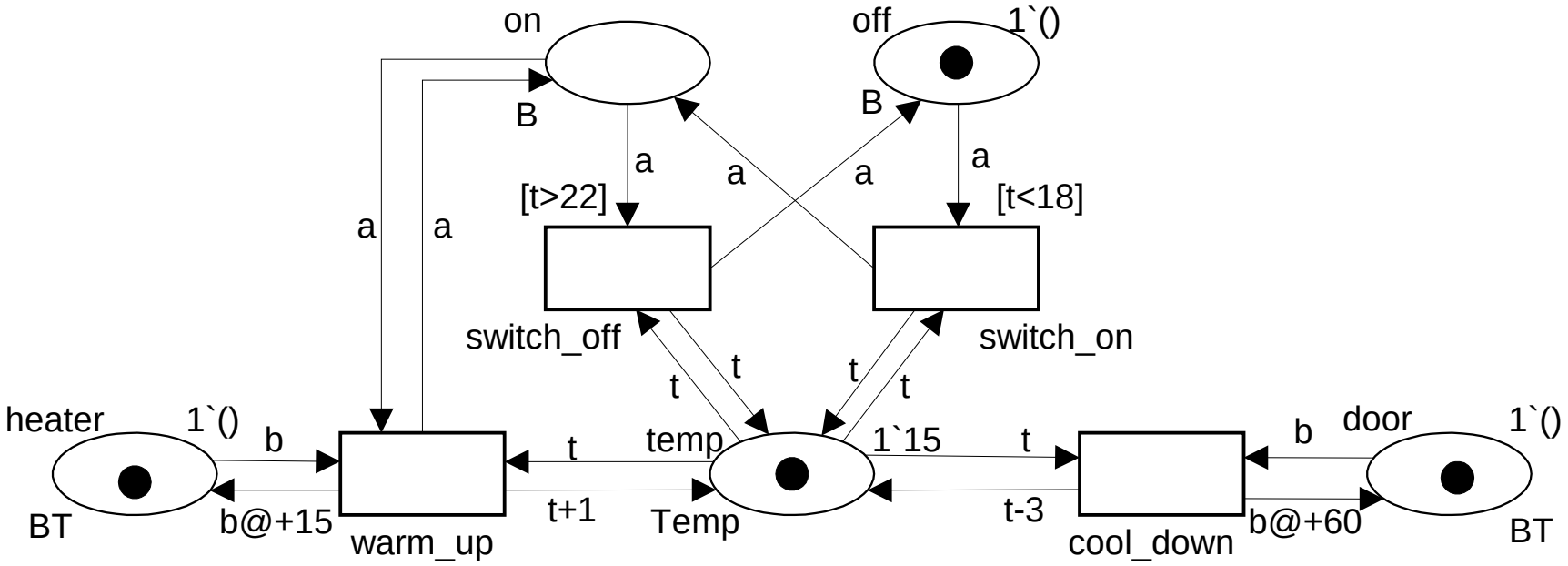


Example: Thermostat system

- At any point the room has a temperature (initially 15 degrees centigrade).
- There is a heater to warm up the house and there is a door which opens every hour such that part of the warmth escapes.
- When the door opens the temperature in the room suddenly drops by three degrees centigrade.
- The heater has a capacity of heating the room 1 degree centigrade every 15 minutes.
- When the heater would be switched on the whole time the temperature would continue to rise by 1 degree per hour. Therefore, there is a control system, i.e., the thermostat, which switches off the heater. The thermostat uses the following rules.
- If the temperature drops below 18, the heater is switched on.
- If the temperature rises above 22, the heater is switched off.

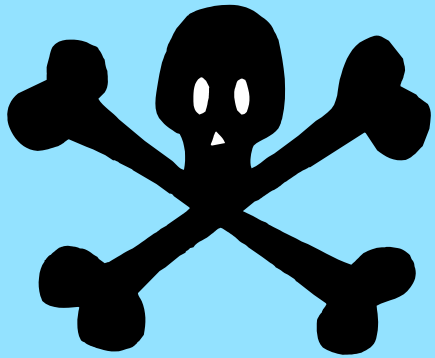
CPN model of thermostat system

```
color Temp = string;
color B = unit;
color BT = B timed;
var t:Temp;
var a:B;
var b:BT;
```



Exercise

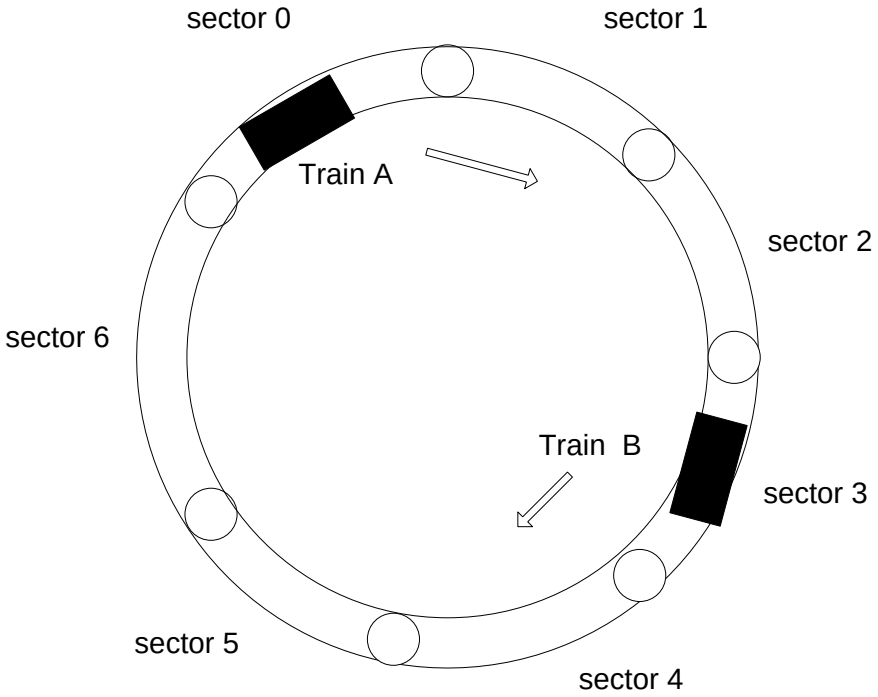
- **Describe the room temperature in time starting in the initial state shown, i.e., play a timed, colored ``token game''.**
- **Extend the model such that there is a day program and a night program. From midnight to 8am, the thermostat tries to keep the temperature between 14 and 18 degrees centigrade. (If the temperature drops below 14 the heater is switched on. If the temperature rises above 18 the heater is switched off.) From 8am to midnight, the temperature is kept between 18 and 22 degrees, like before.**



WARNING

It is not sufficient to understand the (process) models. You have to be able to design them yourself !

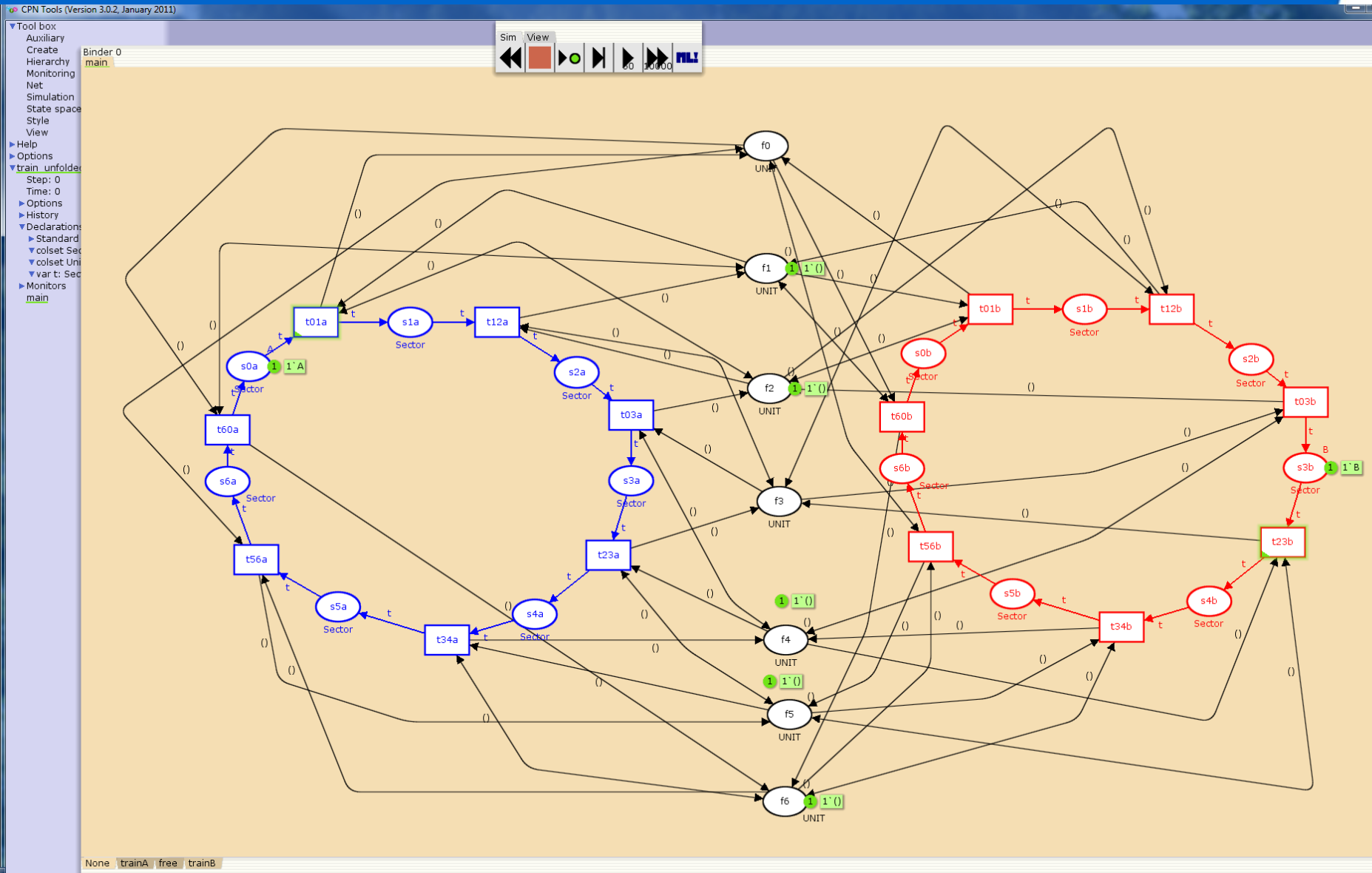
Exercise: Train system



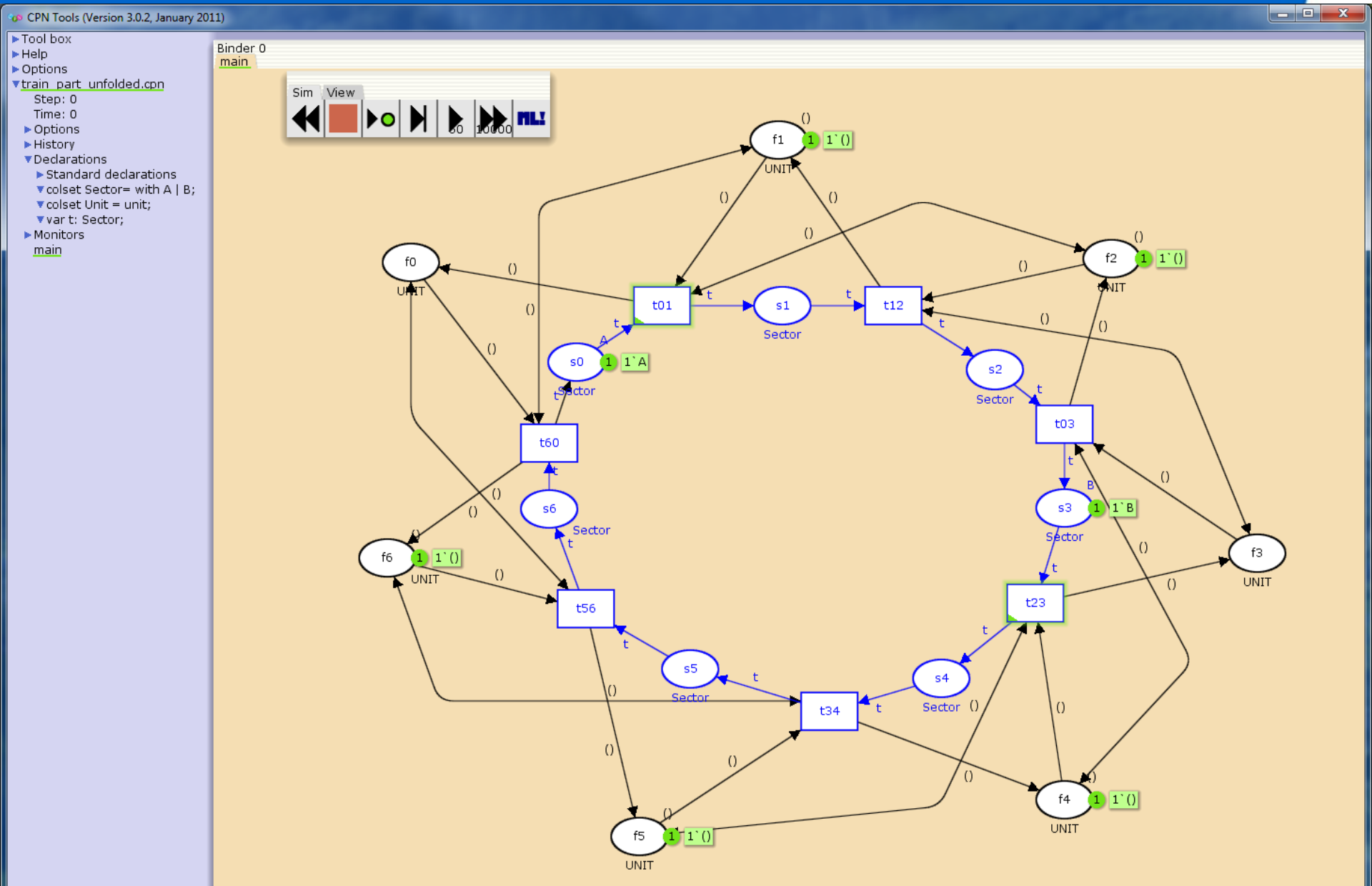
- 7 sectors (tracks)
- 2 trains: A and B
- When moving to a new sector both this sector and the next one should be empty.
- Trains drive in one direction.

- Model as a classical Petri net.
- Model in terms of CPN without folding the tracks.
- Model as a CPN with folding the tracks (i.e., only two places).

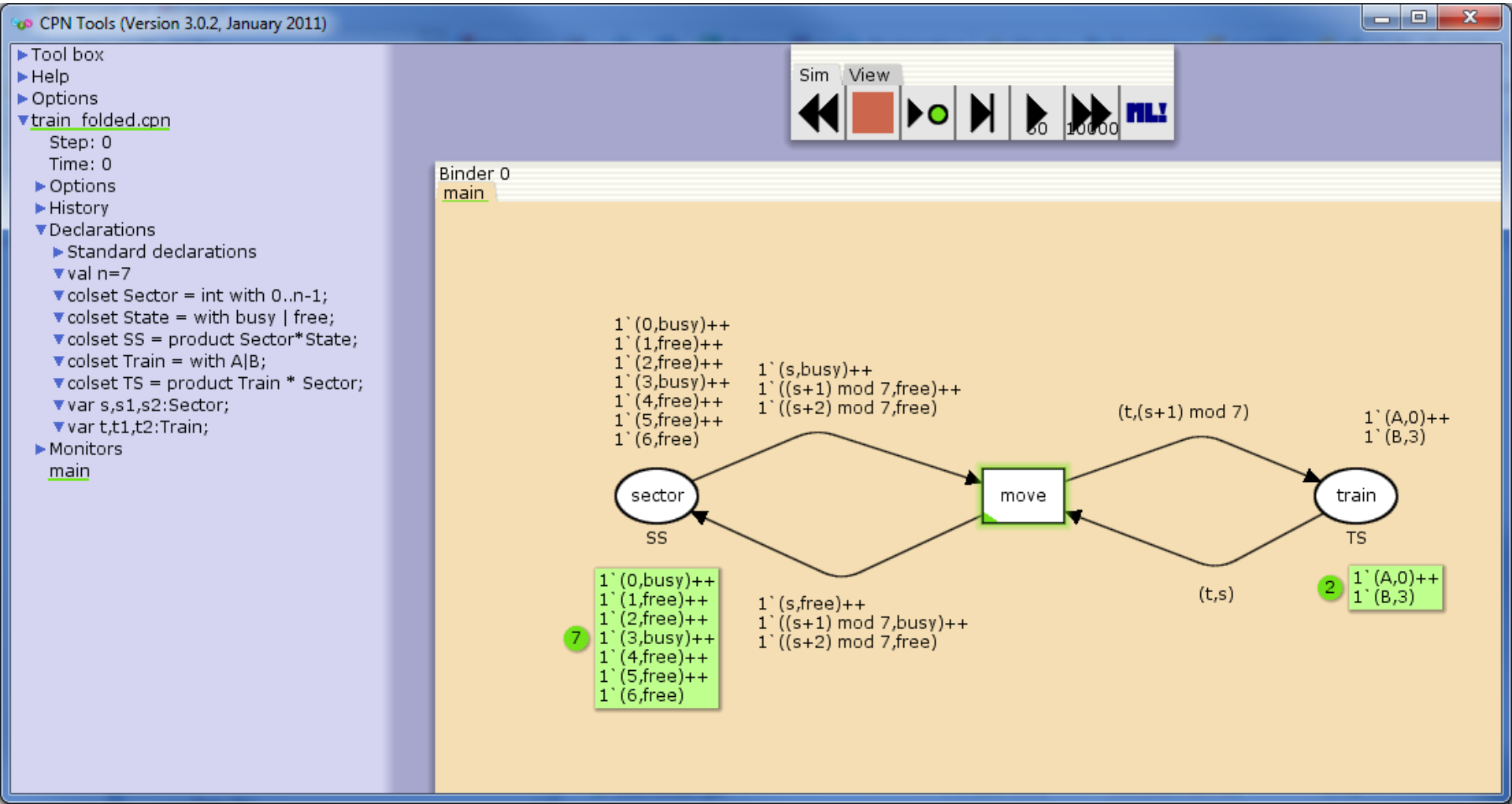
Unfolded


CPN Tools (Version 3.0.2, January 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
train_unfolded
Step: 0
Time: 0
Options
History
Declarations
Standard
colset Sec
colset Uni
var t: Sec
Monitors
main
Sim
View
Binder 0
main
f0
UNIT
f1
UNIT
f2
UNIT
f3
UNIT
f4
UNIT
f5
UNIT
f6
UNIT
t01a
s1a
Sector
t12a
s2a
Sector
t03a
s3a
Sector
t23a
s4a
Sector
t34a
s5a
Sector
t56a
s6a
Sector
t60a
s0a
Sector
t01b
s1b
Sector
t12b
s2b
Sector
t03b
s3b
Sector
t23b
s4b
Sector
t34b
s5b
Sector
t56b
s6b
Sector
t60b
s0b
Sector
None
trainA
free
trainB

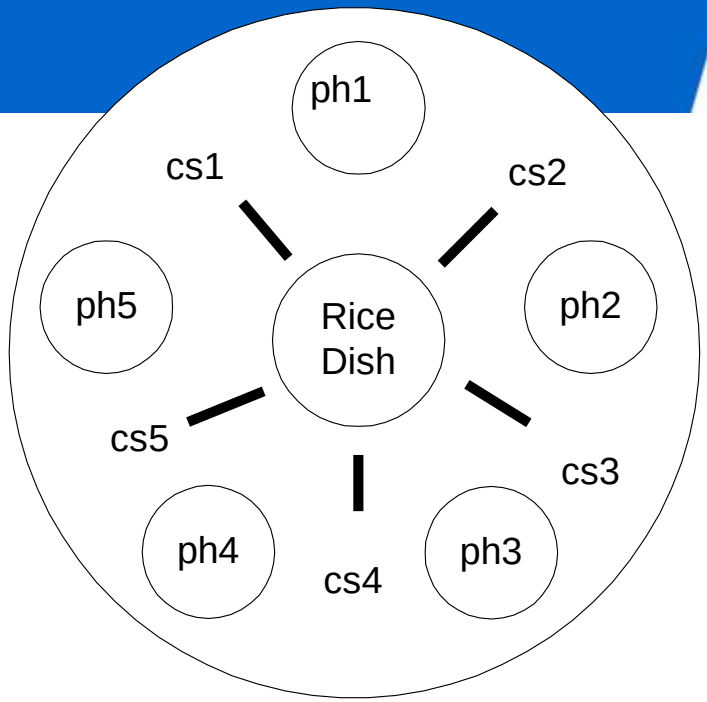
Partially folded



Trains and tracks folded

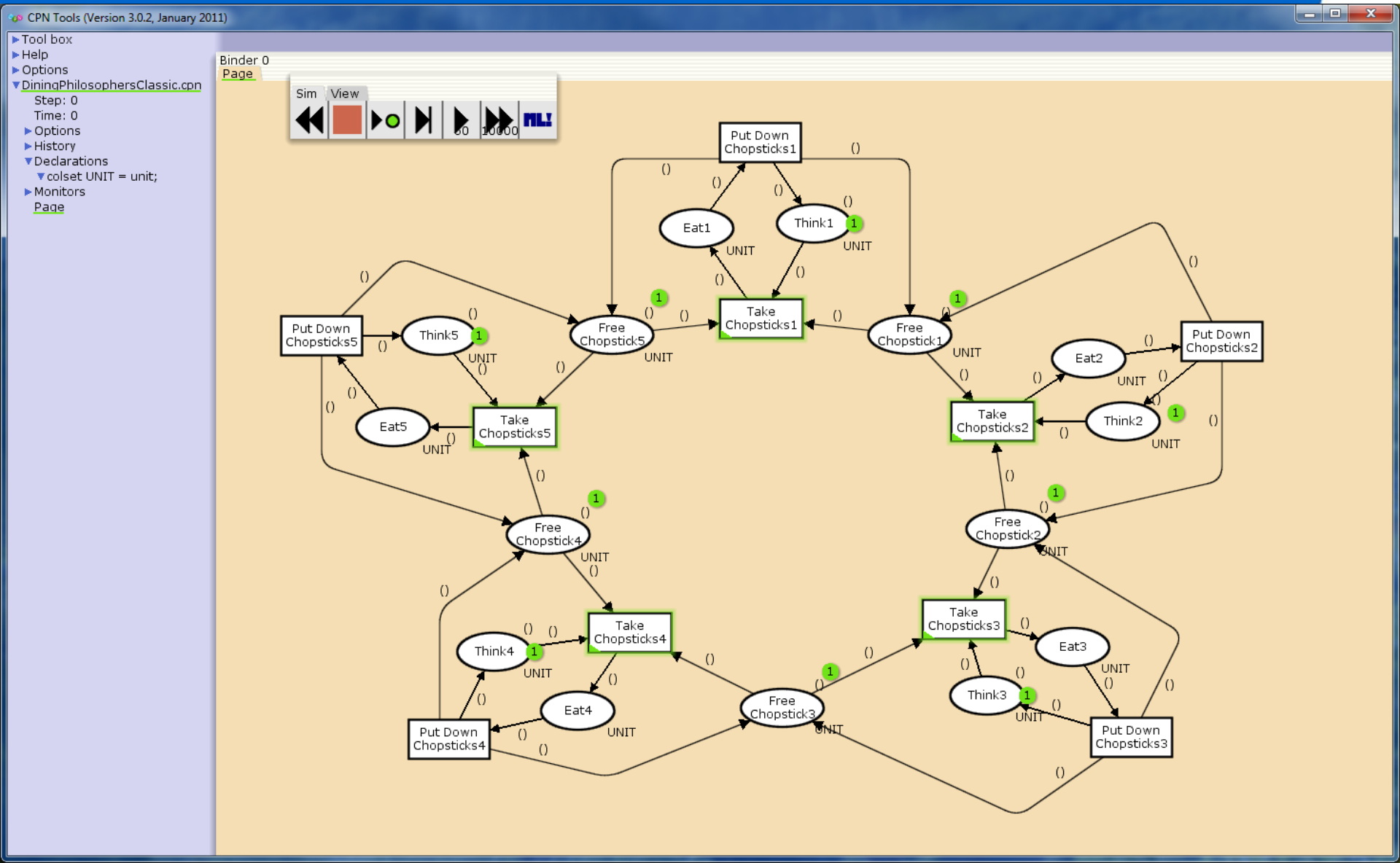


Exercise: Philosophers

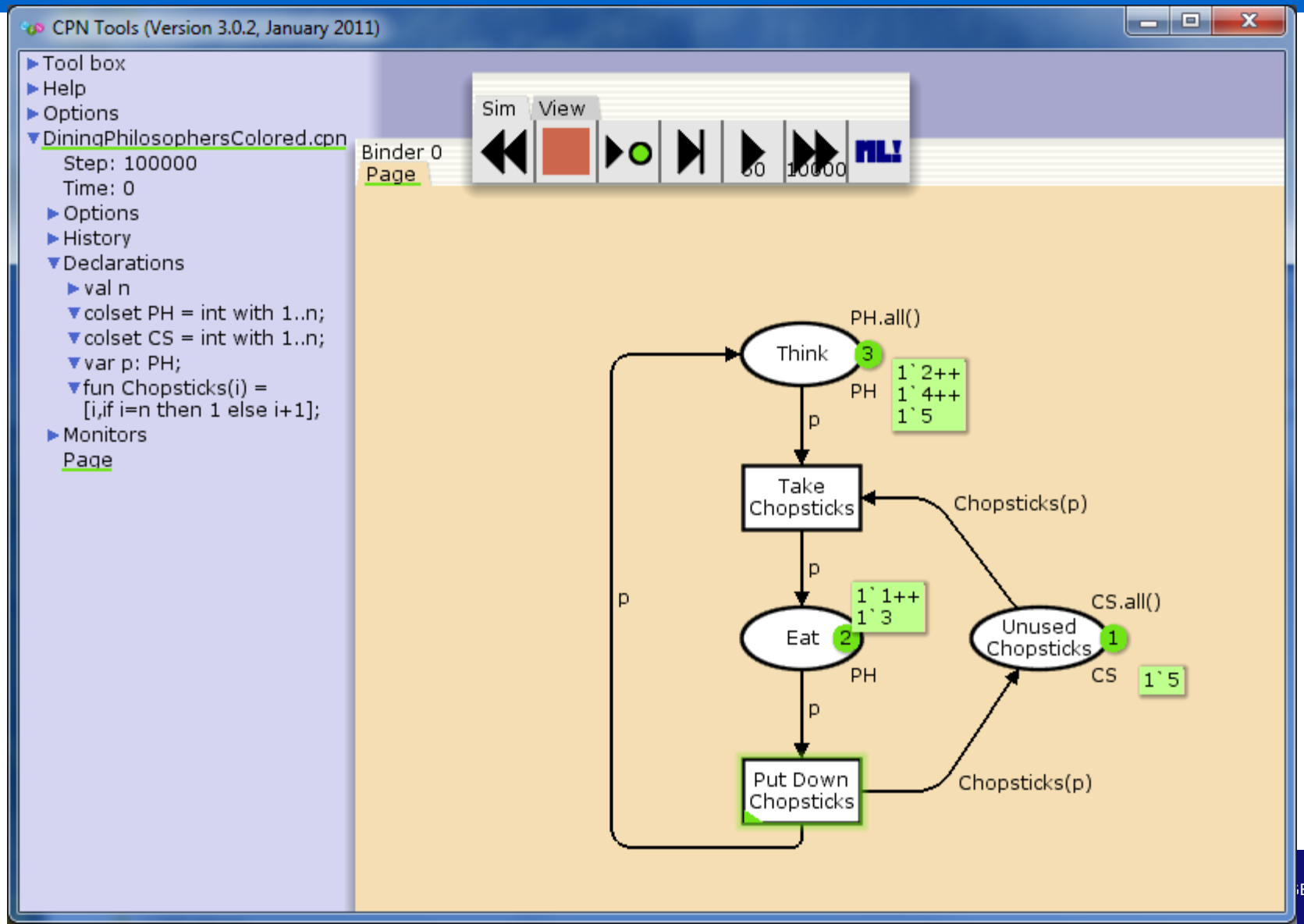


- 5 philosophers
- 5 chopsticks
- Each philosopher is either thinking or eating.
- For eating two chopsticks are needed.
- Chopsticks need to be shared among neighbors.
- Both chopsticks are taken and released at the same time.
- Model as a classical Petri net.
- Model in terms of CPN using only three places and two transitions.

Classical Petri net

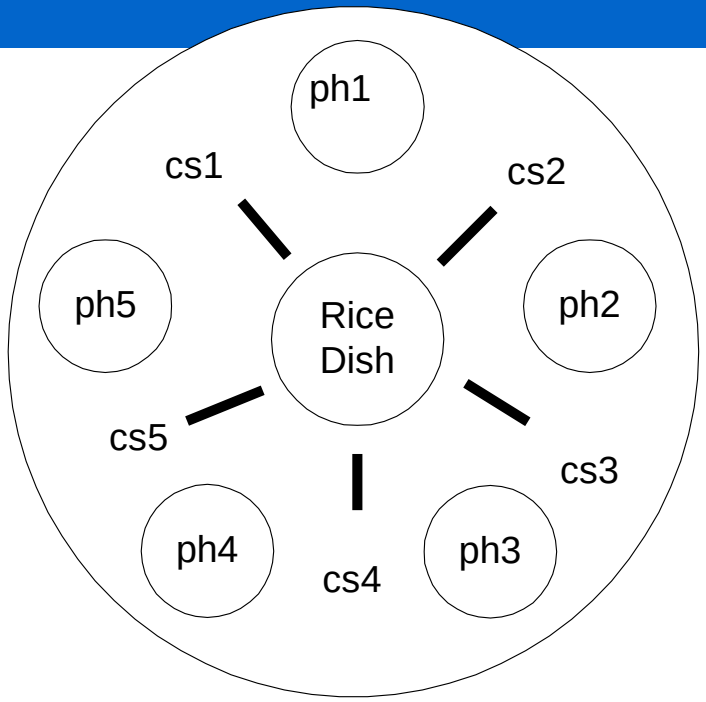


Folded


CPN Tools (Version 3.0.2, January 2011)
Tool box
Help
Options
DiningPhilosophersColored.cpn
Step: 100000
Time: 0
Options
History
Declarations
val n
colset PH = int with 1..n;
colset CS = int with 1..n;
var p: PH;
fun Chopsticks(i) =
[i,if i=n then 1 else i+1];
Monitors
Page
Sim
View
Binder 0
Page
PH.all()
Think
3
PH
1`2++
1`4++
1`5
p
Take
Chopsticks
Chopsticks(p)
p
1`1++
1`3
Eat
2
PH
CS.all()
Unused
Chopsticks
1
CS
1`5
p
Put Down
Chopsticks
Chopsticks(p)

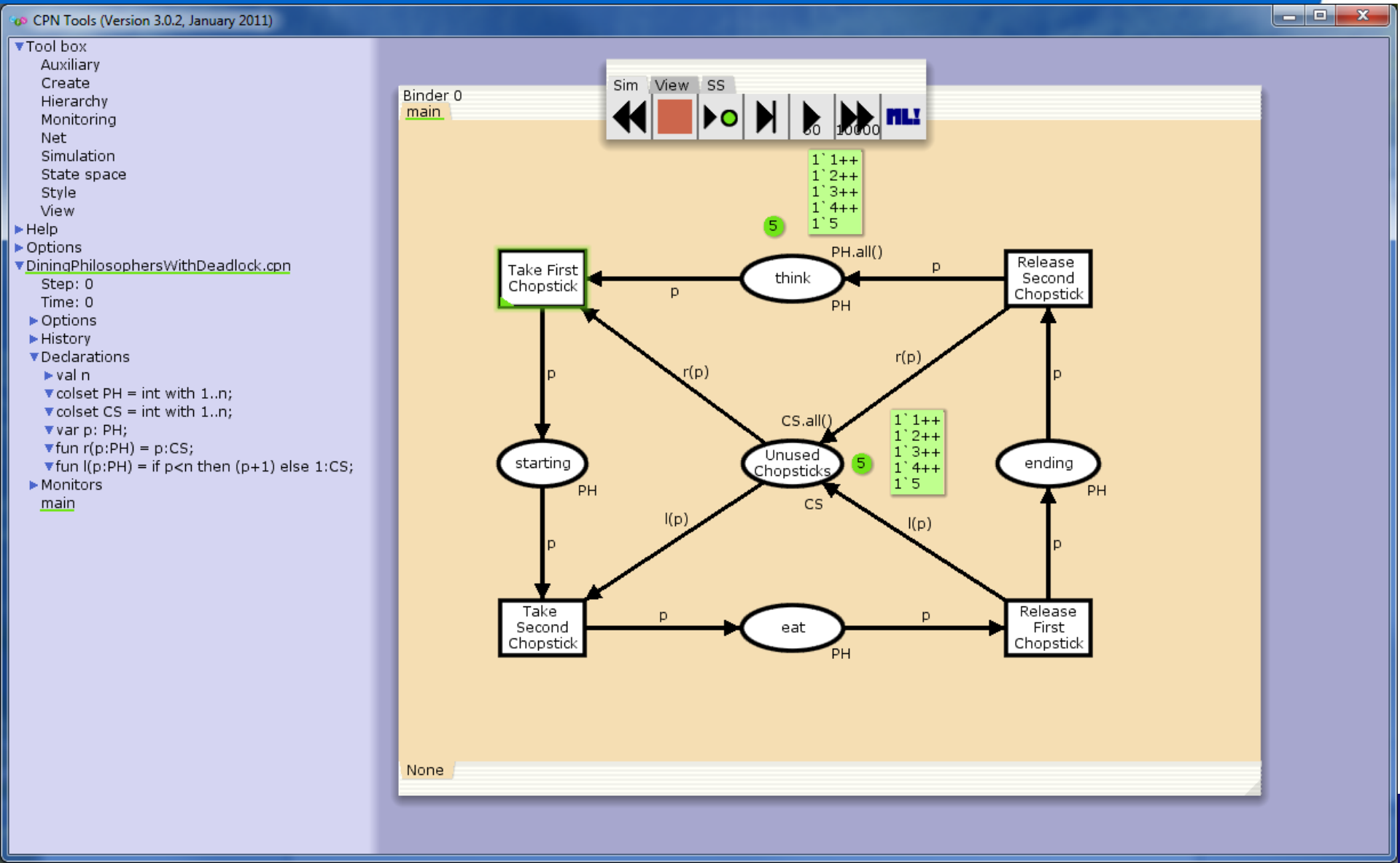
Exercise: Philosophers (2)

- **5 philosophers**
- **5 chopsticks**
- **Each philosopher is either thinking or eating.**
- **For eating two chopsticks are needed.**
- **Chopsticks need to be shared among neighbors.**
- ***First the right chopstick is taken. Then the second one is taken.***
- ***The two chopstick are released in reversed order.***

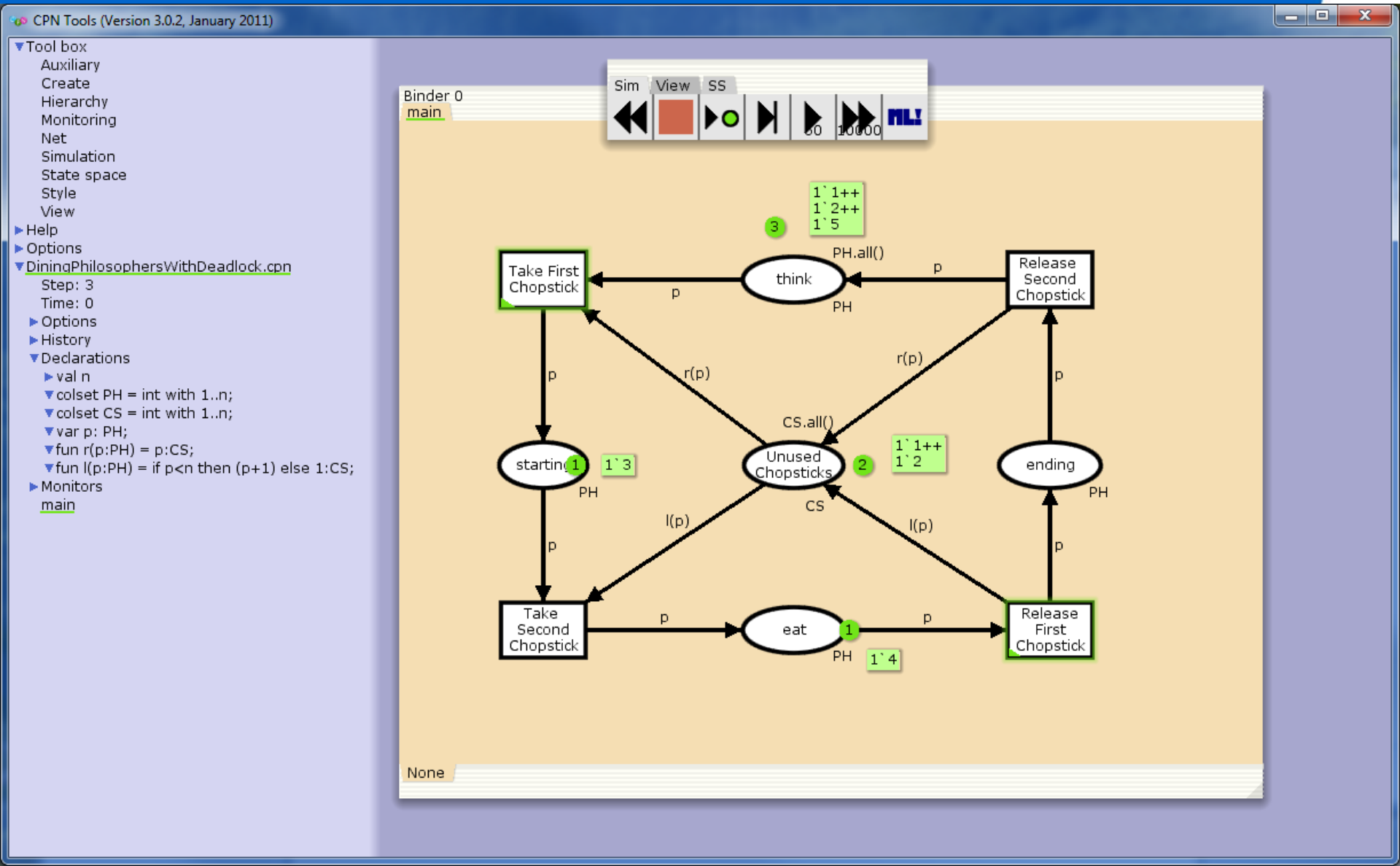


- **Model in terms of CPN.**
- **Are deadlocks possible?**

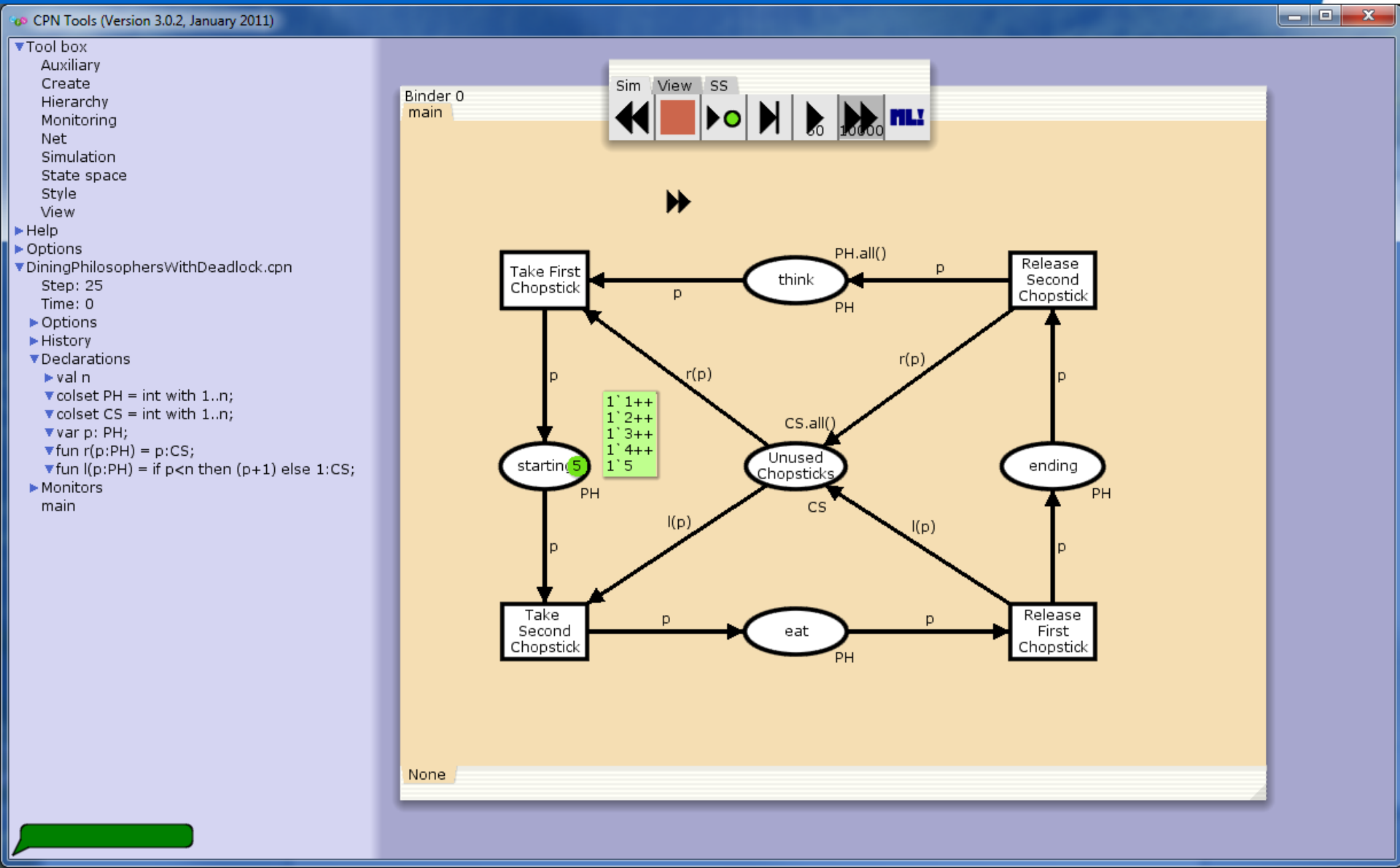
Initial state



4 is eating, 3 took his right chopstick

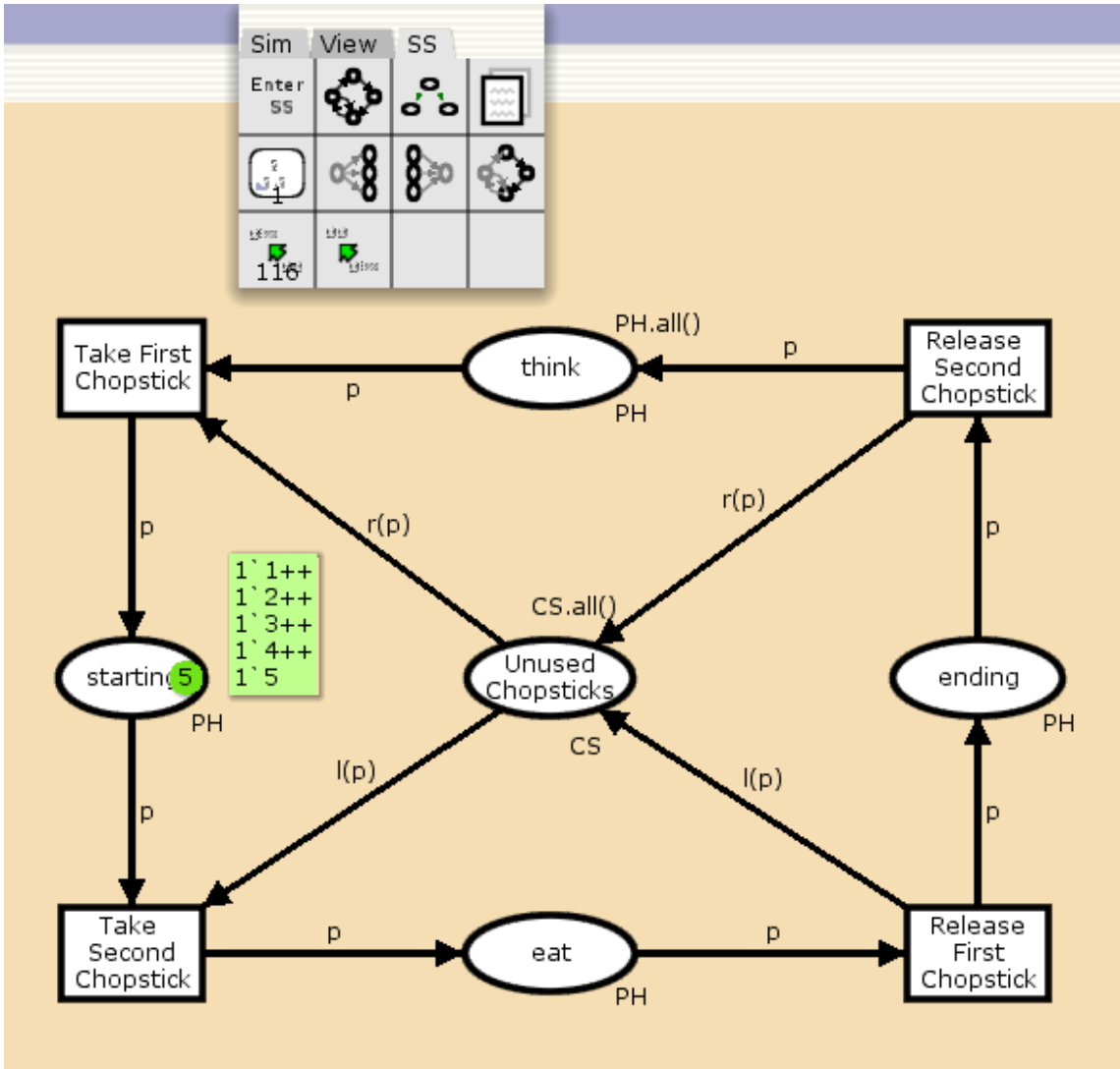


Deadlock


CPN Tools (Version 3.0.2, January 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
DiningPhilosophersWithDeadlock.cpn
Step: 25
Time: 0
Options
History
Declarations
val n
colset PH = int with 1..n;
colset CS = int with 1..n;
var p: PH;
fun r(p:PH) = p:CS;
fun l(p:PH) = if p<n then (p+1) else 1:CS;
Monitors
main
Binder 0
main
Sim
View
SS
ML!
Take First Chopstick
think
PH.all()
Release Second Chopstick
r(p)
l(p)
1`1++
1`2++
1`3++
1`4++
1`5
CS.all()
starting
5
Unused Chopsticks
CS
ending
Take Second Chopstick
eat
Release First Chopstick
None

From state space report

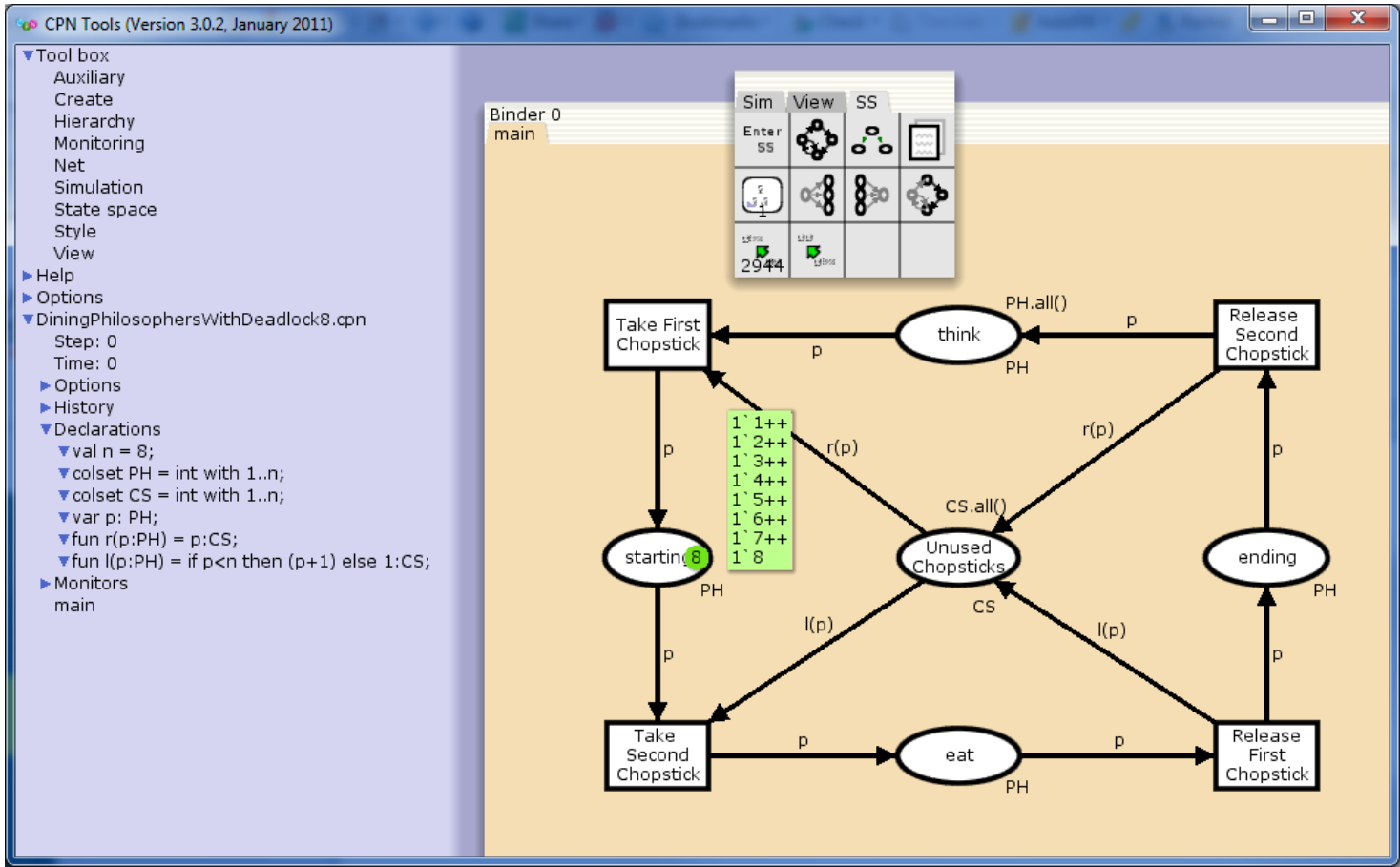
- **State Space**
  - **Nodes: 392**
  - **Arcs: 1415**
- **One home marking**
- **One dead marking**



Adding philosophers (n=8)

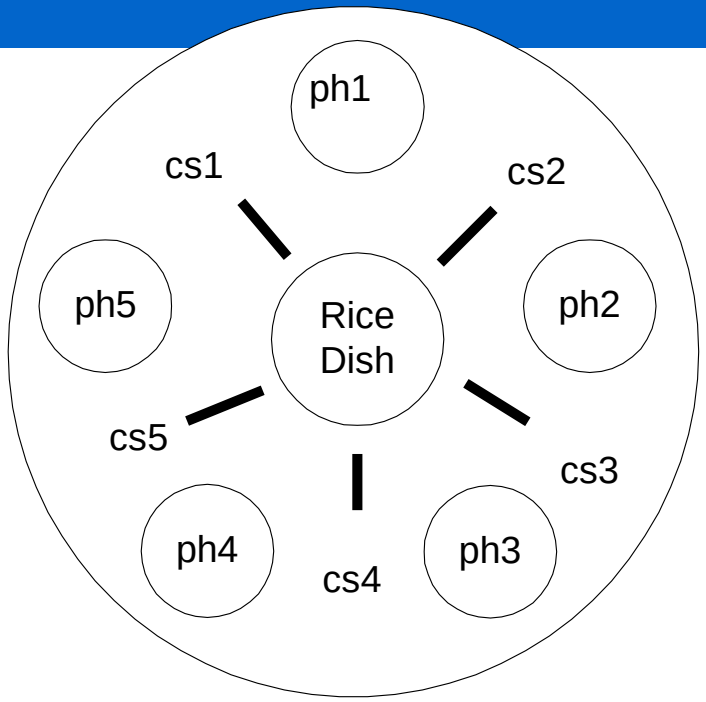
Nodes: 14158

Arcs: 81848



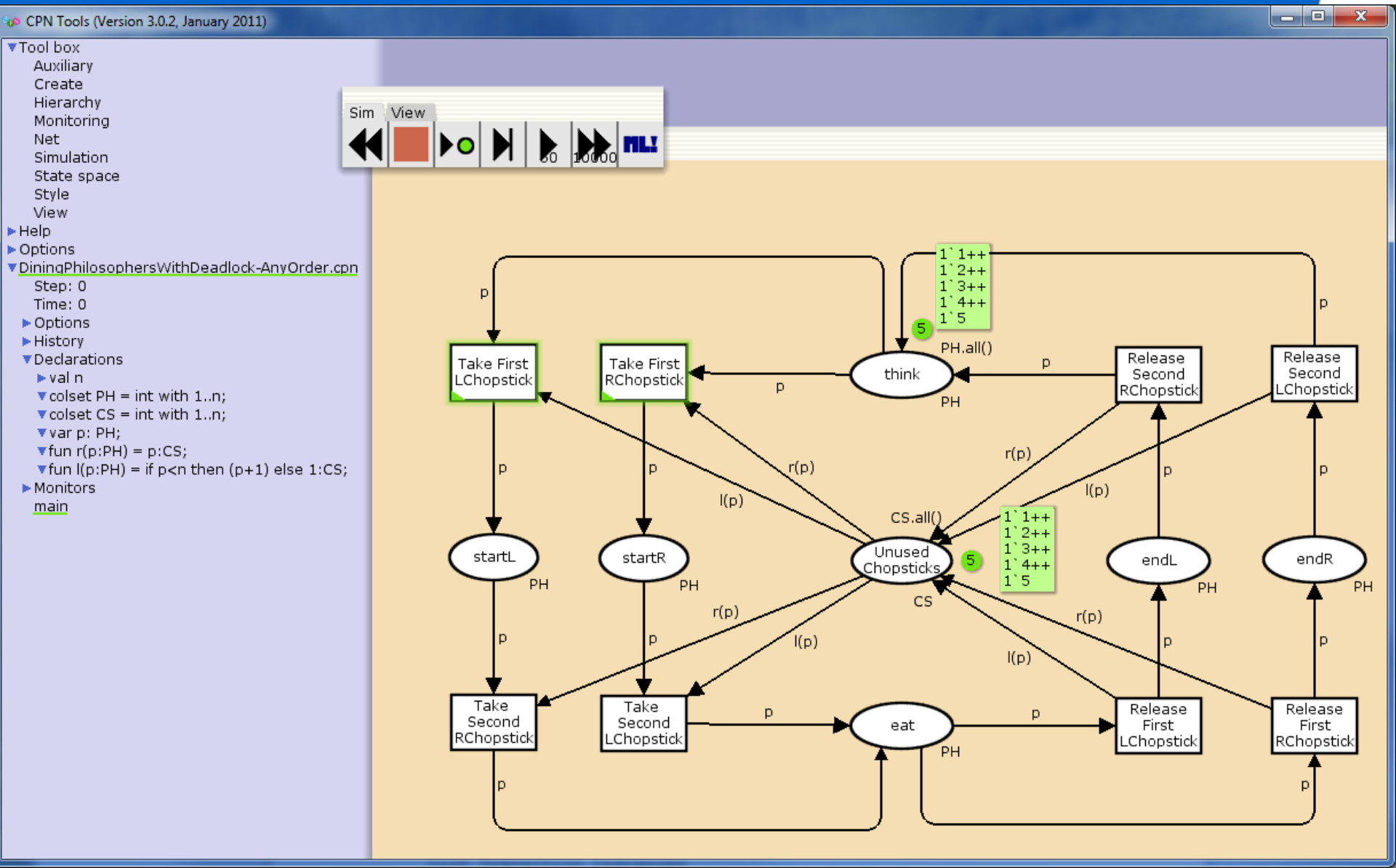
Exercise: Philosophers (3)

- **5 philosophers**
- **5 chopsticks**
- **Each philosopher is either thinking or eating.**
- **For eating two chopsticks are needed.**
- **Chopsticks need to be shared among neighbors.**
- ***First the one chopstick (either left or right) is taken. Then the other one is taken.***
- ***Also released in arbitrary order.***



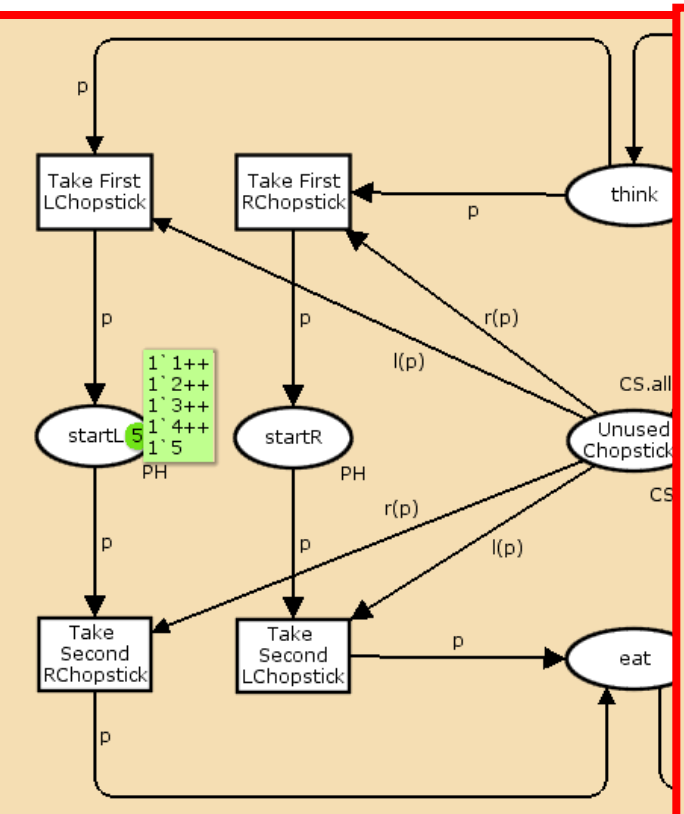
- **Model in terms of CPN.**
- **Are deadlocks possible?**

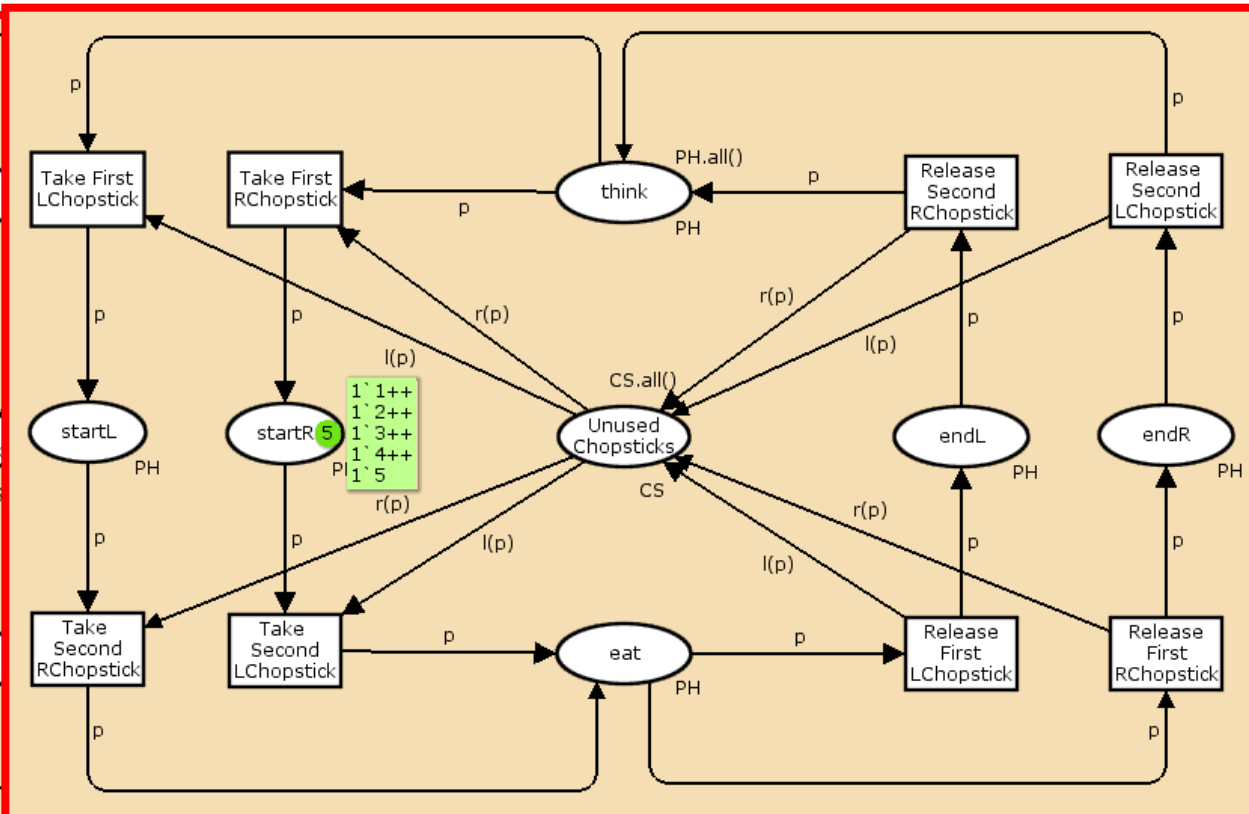
Model


CPN Tools (Version 3.0.2, January 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
DiningPhilosophersWithDeadlock-AnyOrder.cpn
Step: 0
Time: 0
Options
History
Declarations
val n
colset PH = int with 1..n;
colset CS = int with 1..n;
var p: PH;
fun r(p:PH) = p:CS;
fun l(p:PH) = if p<n then (p+1) else 1:CS;
Monitors
main
Sim
View
1`1++
1`2++
1`3++
1`4++
1`5
PH.all()
think
Take First LChopstick
Take First RChopstick
Release Second RChopstick
Release Second LChopstick
startL
startR
Unused Chopsticks
CS.all()
CS
endL
endR
Take Second RChopstick
Take Second LChopstick
eat
Release First LChopstick
Release First RChopstick
PH
p
r(p)
l(p)

State space analysis

- 1473 states
- 6270 transitions
- two dead markings





Tradeoff

- **More information in tokens**
  - color sets, functions, etc.
  - behavior may be hidden in “code”
  - extreme case: all behavior folded into one place and one transition
- **More information in network**
  - possibly spaghetti networks to encode simple things
  - behavior may be incomprehensible
  - cannot be parameterized
  - extreme case: (infinite) classical Petri net

More on functions: Recursion

- **“fun fac(x:INT) = if x>1 then x*fac(x-1) else 1” is a recursive function since the function is expressed in terms of itself.**
- **Two cases:**
  - **fac(x) = x*fac(x-1)**
  - **fac(1) = 1**
- **fac(10)=10*fac(9)=10*9*fac(8)=10*9*8*fac(7)= … = 10*9*8*7*6*5*4*3*2*1 = 3628800**

Recursion (1)

```
color Product = string;
color Number = int;
color StockItem = record prod:Product * number:Number;
color Stock = list StockItem;
fun totalstock(s:Stock) =
  if s = [ ]
  then 0
  else (#number(hd(s)))+totalstock(tl(s));
```

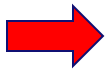
Also see http://cpntools.org/documentation/concepts/colors/declarations/colorsets/record_colour_sets

Recursion (2)

Instead of sum the maximum is taken

```
fun maxstock(s:Stock) =
  if s = [ ]
  then 0
  else if (#number(hd(s))) >= maxstock(tl(s)) then #number(hd(s))
                                              else maxstock(tl(s));
```

Prod:Product	Number:number
"apple"	301
"orange"	504
"pear"	423
“banana"	134
…	…



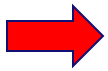
504

Recursion (3)

Function calls other function

```
fun maxstockname(s:Stock) =
  if s = [ ]
  then "no product found"
  else if (#number(hd(s)))=maxstock(tl(s)) then #prod(hd(s))
                                  else maxstockname(tl(s));
```

Prod:Product	Number:number
"apple"	301
"orange"	504
"pear"	423
“banana"	134
…	…



"orange"

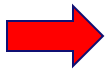
Recursion (4)

Function has two arguments

```
fun enoughstock(s:Stock,n:Number) =
  if s = [ ]
  then [ ]
  else if (#number(hd(s)))>= n then hd(s)::enoughstock(tl(s),n)
                               else enoughstock(tl(s),n);
```

Prod:Product	Number:number
"apple"	301
"orange"	504
"pear"	423
“banana"	134
…	…

n=400



Prod:Product	Number:number
"orange"	504
"pear"	423
…	…

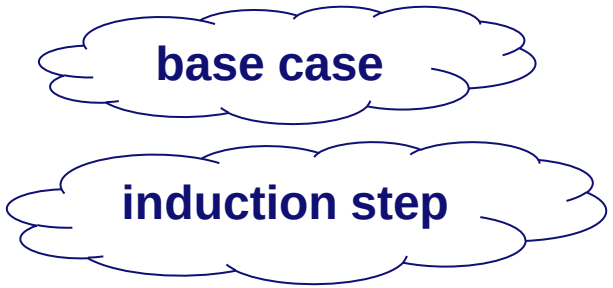
Recursion (5)


```
fun enoughstockn(s:Stock,n:Number) =
  if s = [ ]
  then 0
  else if (#number(hd(s)))>= n then 1+enoughstockn(tl(s),n)
                               else enoughstockn(tl(s),n);
```

More on functions: Pattern matching

```
fun lenlist1(s:Stock) =
  if s = [ ]
  then 0
  else 1+lenlist(tl(s));
```

```
fun lenlist2([ ]) = 0 |
 lenlist2(si::s) = 1+lenlist2(s);
```



No explicit typing!!!

Pattern matching (1)

```
fun totalstock(s:Stock) =
if s = [ ]
then 0
else (#number(hd(s)))+totalstock(tl(s));
```

```
fun totalstock([ ] : Stock) = 0 |
  totalstock(si::s) = (#number(si))+totalstock(s);
```

Pattern matching (2)

```
fun maxstock(s:Stock) =
  if s=[ ]
  then 0
  else if (#number(hd(s))) >= maxstock(tl(s)) then #number(hd(s))
                                              else maxstock(tl(s));
```

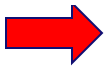
```
fun maxstock([ ]:Stock) = 0 |
 maxstock(si::s) = if (#number(si))>maxstock(s) then #number(si)
                                                else maxstock(s);
```

Pattern matching (3)

```
fun     incrs(x:StockItem,[ ]:Stock) = [x]        |
        incrs (x,(si::s)) =
                if  (#prod(si))=(#prod(x))
                then    {prod=(#prod(si)),
                         number=((#number(si))+(#number(x)))}
                         ::incrs(x,s)
                else (si::incrs(x,s));
```

Prod:Product	Number:number
"apple"	301
"orange"	504
"pear"	423
“banana"	134
…	…

x={prod="apple", number=20}



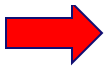
Prod:Product	Number:number
"apple"	321
"orange"	504
"pear"	423
“banana"	134
…	…

Pattern matching (3)

```
fun     incrs(x:StockItem,[ ]:Stock) = [x]      |
        incrs (x,(si::s)) =
                if  (#prod(si))=(#prod(x))
                then    {prod=(#prod(si)),
                         number=((#number(si))+(#number(x)))}
                         ::incrs(x,s)
                else (si::incrs(x,s));
```

Prod:Product	Number:number
"apple"	301
"orange"	504
"pear"	423
“banana"	134
…	…

x={prod=“XX", number=20}



Prod:Product	Number:number
"apple"	301
"orange"	504
"pear"	423
“banana"	134
…	…
"XX"	20

Pattern matching (4)

```
fun reverse([ ]) = [ ] | reverse(x::y) = reverse(y)^^[x];

fun elt([ ], a) = false | elt((x::xs), a) =  a=x orelse elt(xs, a);


fun del(a,[ ]) = [ ] | del(a,(x::xs)) = if a=x then xs else x::(del(a,xs));

fun intersect([ ], ys) = [ ] |
intersect(xs, [ ]) = [ ] |
intersect ((x::xs), ys) = if elt(ys,x)
                          then x::(intersect(xs,(del(x,ys))))
                          else intersect(xs, ys);
```

Example: Sudoku

```
colset Index = int with 0..8;
colset Cel = int with ~1..9;
colset Cels = list Cel;
colset Pos = product Index * Index;
colset Val = product Pos * Cel;
colset Sudoku = list Val;
```

for 9 rows and columns

~1 and 0 have a technical reason, normal values are 1..9

Write an ML function to solve a Sudoku assuming that in each step there is a "deterministic candidate", i.e., no backtracking needed.

Input

0 values are not inserted

```
val v4 = [
[6,0,0, 0,8,0, 0,0,9],
[0,7,0, 4,0,6, 0,8,0],
[0,0,0, 5,0,1, 0,0,0],

[0,1,7, 2,0,9, 8,5,0],
[2,0,0, 0,0,0, 0,0,1],
[0,8,4, 1,0,3, 6,7,0],

[0,0,0, 3,0,8, 0,0,0],
[0,4,0, 9,0,5, 0,1,0],
[8,0,0, 0,7,0, 0,0,5]
];
```

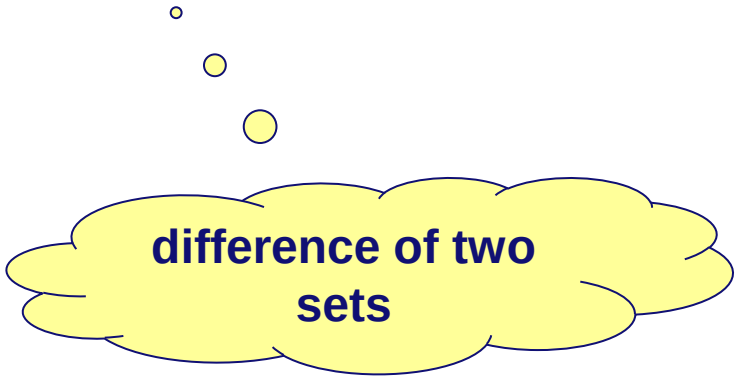
```
fun readcell(x,i,j) = if x=[ ] then [ ] else if
   hd(x) = 0 then readcell(tl(x),i,j+1) else
   ((i,j),hd(x))::readcell(tl(x),i,j+1);
fun readrow(x,i) = if x=[ ] then [ ] else
   readcell(hd(x),i,0)^^readrow(tl(x),i+1);
fun read(x) = readrow(x,0):Sudoku;
```

to map “string” (list of lists) representation to list of ((i,j),c) values

0 values are empty

Useful functions

```
fun dom([ ]) = [ ] | dom((x,y)::l) = x::dom(l);
fun elt([ ], a) = false | elt((x::xs), a) =  a=x orelse elt(xs,
   a);
fun fmap([ ],z) = 0 | fmap((x,y)::l,z) = if x=z then y else
   fmap(l,z);
fun sdiff([ ],z) = [ ] | sdiff(x::y,z) = if elt(z,x) then
   sdiff(y,z) else x::sdiff(y,z);
infix sdiff;
```



Basic functions

same block

values in the block containing (i,j)

```
fun row([ ],k) = [ ] | row(((i,j),c)::s,k) = if i=k then
   c::row(s,k) else row(s,k) : Cels;
fun column([ ],k) = [ ] | column(((i,j),c)::s,k) = if j=k then
   c::column(s,k) else column(s,k) : Cels;
fun de(i,j) = (i div 3) = (j div 3);
fun block([ ],i,j) = [ ] | block(((i1,j1),c)::s,i,j) = if de(i,i1) andalso de(j,j1) then c::block(s,i,j) else block(s,i,j) :
   Cels;
val uni = [1,2,3,4,5,6,7,8,9]: Cels;
fun free(s,i,j) = ((uni sdiff row(s,i)) sdiff column(s,j)) sdiff
   block(s,i,j) : Cels;
```

all cell values in row k

all cell values in column k

all values

remaining options

Possible moves

all possible values of type Pos, i.e., list of all cells

given an s of type Sudoku, all undefined positions are returned

```
fun allpos() = Pos.all();
fun undef(s) = allpos() sdiff dom(s);
fun analyze1(s,[ ]) = [ ] | analyze1(s,(i,j)::l) =
   ((i,j),free(s,i,j))::analyze1(s,l);
fun analyze(s) = analyze1(s,undef(s));
```

possible moves per position

Sol

add error entry
(no options left)

add entry with just
one possible move c

```
fun new([ ]) = [ ] |
   new(((i,j),[ ])::s) = ((i,j),~1)::new(s) |
   new(((i,j),[c])::s) = ((i,j),c)::new(s) |
   new(((i,j),c::cs)::s) = new(s);
fun solve(s) =
   if new(analyze(s)) = [ ]
         then s
         else solve(new(analyze(s))^^s);
```

skip if multiple
moves possible

repeatedly call solve until no
entries can be added (done or
non-deterministic choice needed

Sort results

```
fun sord(((x1,y1),z1),((x2,y2),z2)) = (x1 < x2) orelse
   (x1=x2 andalso y1 < y2);
fun solver(s) =  sort sord (solve(s));
```

sort function is built in
“sort lt_fun l” sorts list l using the function lt_fun to determine when one element in the list is less than another.

See http://cpntools.org/documentation/concepts/colors/declarations/colorsets/list_colour_sets

Generate string (just for presentation)

```
fun result1(s,i) = if i>= 81 then "-------------------\n" else
   (Int.toString(fmap(s,(i div 9, i mod 9))) ^ (if i mod 9 = 8
   then "\n"  else " " )^ result1(s,i+1));
fun result(s) =
   "\n-------------------\n"^result1(solver(s),0);
```

CPN Tools (Version 3.0.2, January 2011)

- Tool box
- Help
- Options
- sudoku.cpn
  - Step: 0
  - Time: 0
  - Options
  - History
  - Declarations
    - Standard declarations
    - types
      - colset Index = int with 0..8;
      - colset Cel = int with ~1..9;
      - colset Cels = list Cel;
      - colset Pos = product Index * Index;
      - colset Val = product Pos * Cel;
      - colset Sudoku = list Val;
      - val uni = [1,2,3,4,5,6,7,8,9]: Cels;
    - values
    - functions

```
fun dom([]) = [] | dom((x,y)::l) = x::dom(l);
fun elt([], a) = false |
  elt((x::xs), a) =  a=x orelse elt(xs, a);
fun fmap([],z) = 0 | fmap((x,y)::l,z) = if x=z then y else fmap(l,z);
fun sdiff([],z) = [] | sdiff(x::y,z) = if elt(z,x) then sdiff(y,z) else x::sdiff(y,z);
infix sdiff;
fun readcell(x,i,j) = if x=[] then [] else if hd(x) = 0 then readcell(tl(x),i,j+1) else ((i,j),hd(x))::readcell(tl(x),i,j+1);
fun readrow(x,i) = if x=[] then [] else readcell(hd(x),i,0)^^readrow(tl(x),i+1);
fun read(x) = readrow(x,0):Sudoku;
fun row([],k) = [] | row(((i,j),c)::s,k) = if i=k then c::row(s,k) else row(s,k) : Cels;
fun column([],k) = [] | column(((i,j),c)::s,k) = if j=k then c::column(s,k) else column(s,k) : Cels;
fun de(i,j) = (i div 3) = (j div 3);
fun block([],i,j) = [] | block(((i1,j1),c)::s,i,j) = if de(i,i1) andalso de(j,j1) then c::block(s,i,j) else block(s,i,j) : Cels;
fun free(s,i,j) = ((uni sdiff row(s,i)) sdiff column(s,j)) sdiff block(s,i,j) : Cels;
fun allpos() = Pos.all();
fun undef(s) = allpos() sdiff dom(s);
fun analyze1(s,[]) = [] | analyze1(s,(i,j)::l) = ((i,j),free(s,i,j))::analyze1(s,l);
fun analyze(s) = analyze1(s,undef(s));
fun new([]) = [] |
  new(((i,j),[])::s) = ((i,j),~1)::new(s) |
  new(((i,j),[c])::s) = ((i,j),c)::new(s) |
  new(((i,j),c::cs)::s) = new(s);
fun solve(s) = if new(analyze(s)) = [] then s else solve(new(analyze(s))^^s);
fun sord(((x1,y1),z1),((x2,y2),z2)) = (x1 < x2) orelse (x1=x2 andalso y1 < y2);
fun solver(s) =  sort sord (solve(s));
fun result1(s,i) = if i>= 81 then "----------\n" else (Int.toString(fmap(s,(i div 9, i mod 9))) ^
  (if i mod 9 = 8 then "\n"  else " " )^ result1(s,i+1));
fun result(s) = "\n----------\n"^result1(solver(s),0);
```

  - Monitors
  - main

Sim View SS

Create

Binder 0

main | val v5

*** replace v1 by v2, v3, v4 or your own sudoku ***

result(read(v5))

calculate

*** eva

STRING

1

```
1`"
----------
1 2 6 5 8 4 9 3 7
4 7 5 2 3 9 8 6 1
8 9 3 1 6 7 5 4 2
7 5 1 9 4 6 3 2 8
3 4 8 7 2 5 1 9 6
9 6 2 3 1 8 4 7 5
2 3 7 8 9 1 6 5 4
6 1 9 4 5 2 7 8 3
5 8 4 6 7 3 2 1 9
----------
"
```

read(v1

read(v2

read(v3

read(v4

[1,2] sdiff [2,3]

None

CPN Tools (Version 3.0.2, January 2011)

- Tool box
- Help
- Options
- sudoku.cpn
  - Step: 0
  - Time: 0
  - Options
  - History
  - Declarations
    - Standard declarations
    - types
      - colset Index = int with 0..8;
      - colset Cel = int with ~1..9;
      - colset Cels = list Cel;
      - colset Pos = product Index * Index;
      - colset Val = product Pos * Cel;
      - colset Sudoku = list Val;
      - val uni = [1,2,3,4,5,6,7,8,9]: Cels;
    - values
      - val v1 = [
        [9,0,3, 5,0,0, 1,0,0],
        [0,0,6, 7,0,0, 0,0,0],
        [0,0,0, 8,0,4, 0,0,0],

        [0,0,5, 0,0,8, 4,0,0],
        [7,6,0, 0,0,0, 0,2,1],
        [0,0,4, 6,0,0, 3,0,0],

        [0,0,0, 3,0,5, 0,0,0],
        [0,0,0, 0,0,9, 2,0,0],
        [0,0,1, 0,0,2, 7,0,8]
        ];
      - val v2 = [
        [0,7,2, 6,0,1, 5,4,0],
        [8,0,0, 7,0,4, 0,0,2],
        [6,0,0, 2,9,5, 0,0,7],

        [2,3,8, 0,1,0, 7,5,6],
        [0,0,6, 8,0,2, 1,0,0],
        [4,1,7, 0,5,0, 9,2,8],

        [1,0,0, 5,6,8, 0,0,9],
        [7,0,0, 1,0,3, 0,0,5],
        [0,8,5, 9,0,7, 4,6,0]
        ];
      - val v3 = [
        [1,2,3, 0,0,0, 7,8,9],
        [0,0,0, 4,9,0, 0,0,0],
        [0,0,0, 5,0,0, 0,0,0],

        [0,0,0, 6,0,0, 0,0,0],
        [0,0,0, 7,0,0, 0,0,0],
        [0,0,0, 8,0,0, 0,0,0],

        [0,0,0, 0,0,0, 0,0,0],
        [0,0,0, 0,0,0, 0,0,0],
        [0,0,0, 0,0,0, 0,0,0]
        ];
      - val v4 = [
        [6,0,0, 0,8,0, 0,0,9],
        [0,7,0, 4,0,6, 0,8,0]

Sim View SS

Create

Binder 0
main

*** replace v1 by v2, v3, v4 or your own sudoku ***

result(read(v5))

calculate

STRING

1

1`"
---
1 2 6 5 8 4 9 3 7
4 7 5 2 3 9 8 6 1
8 9 3 1 6 7 5 4 2
7 5 1 9 4 6 3 2 8
3 4 8 7 2 5 1 9 6
9 6 2 3 1 8 4 7 5
2 3 7 8 9 1 6 5 4
6 1 9 4 5 2 7 8 3
5 8 4 6 7 3 2 1 9

"

[1,2] sdiff [2,3]

None

Binder 0
val v5

```
val v5 = [
[0,2,0, 5,0,0, 0,3,0],
[0,0,5, 2,0,9, 0,6,1],
[0,9,3, 0,0,7, 5,0,0],

[7,0,0, 9,4,0, 3,0,0],
[3,4,0, 0,0,0, 0,9,6],
[0,0,2, 0,1,8, 0,0,5],

[0,0,7, 8,0,0, 6,5,0],
[6,1,0, 4,0,2, 7,0,0],
[0,8,0, 0,0,3, 0,1,0]
];
```

note the two 1's in middle block

More information

- **About Standard ML:**
  - Robin Milner, Mads Tofte, Robert Harper, and David MacQueen.The Definition of Standard ML: Revised 1997. The MIT Press, 1997.
  - J. D. Ullman. Elements of ML Programming (ML 97 edition). Prentice-Hall, 1998.
  - http://www.standardml.org/Basis/ (for functions)
- **About CPN:**
  - K. Jensen and L.M. Kristensen. Coloured Petri Nets: Modelling and Validation of Concurrent Systems, Springer-Verlag, 2009.
  - W. van der Aalst and C. Stahl. Modeling Business Processes: A Petri Net-Oriented Approach. MIT Press, 2011.
  - K. Jensen: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag, 1997.
  - K. Jensen: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods. Monographs in Theoretical Computer Science, Springer-Verlag, 1997.
  - K. Jensen: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3, Practical Use. Monographs in Theoretical Computer Science, Springer-Verlag, 1997.
  - K. Jensen and G. Rozenberg (eds.): High-level Petri Nets. Theory and Application. Springer-Verlag, 1991.