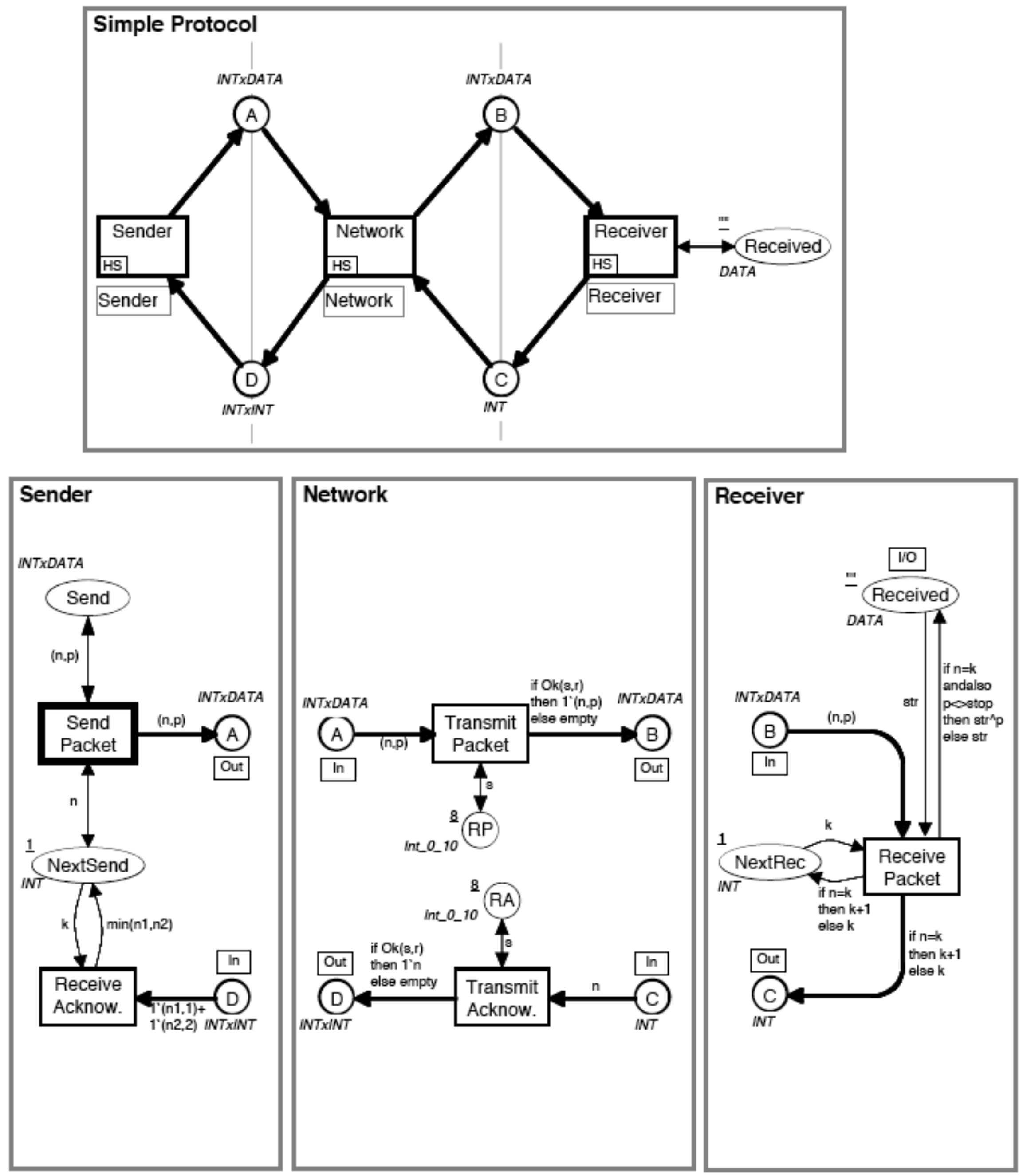
PMR 5237
Modelagem e Design de Sistemas Discretos em Redes de Petri

Aula 11: Modelagem e análise em Redes Coloridas

Prof. José Reinaldo Silva
reinaldo@poli.usp.br

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



A page may contain one ore more *substitution transitions*.

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

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

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

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

Dobramentos (folding)

Um dobramento seria plenamente justificável se, para este exemplo, tivermos agora diferentes tipos de peças para fabricar onde cada tipo delas denota uma receita diferente, isto é, uma sucessão diferente de operações. Neste caso os conflitos da rede deveriam ser resolvidos com o tipo da peça.

Segundo a norma ISO/IEC 15.909-1

A **HLPN** is a structure $HLPN = (P, T, D; Type, Pre, Post, M_0)$ where

- P is a finite set of elements called Places.
- T is a finite set of elements called Transitions disjoint from P ($P \cap T = \emptyset$).
- D is a non-empty finite set of non-empty domains where each element of D is called a type.
- $Type : P \cup T \longrightarrow D$ is a function used to assign types to places and to determine transition modes.
- $Pre, Post : TRANS \longrightarrow \mu PLACE$ are the pre and post mappings with

$$TRANS = \{(t, m) \mid t \in T, m \in Type(t)\}$$

$$PLACE = \{(p, g) \mid p \in P, g \in Type(p)\}$$

. . .

- $M_0 \in \mu PLACE$ is a multiset called the initial marking of the net.

NOTE: $\mu PLACE$ is the set of multisets over the set, $PLACE$ (see Annex A, section A.2).

Exploring CPN Tools

Teoria:
modelagem
req. formais
análise

Ferramentas:
editor
simulador
verificador

Aplicação:
especificação
validação
verificação
implementação

Theory

- The Coloured Petri Nets formalism
  - Introduction of hierarchies
- Modelling primitives
  - Channels, inhibitor arcs, test arcs, ...
  - Monitoring framework
- Analysis methods
  - Symmetry methods
  - Sweep-line method

August 27, 2002 Søren Christensen, MOCA'02

Ferramentas para Redes de Alto nível

1990-2002

Design CPN

2003-…

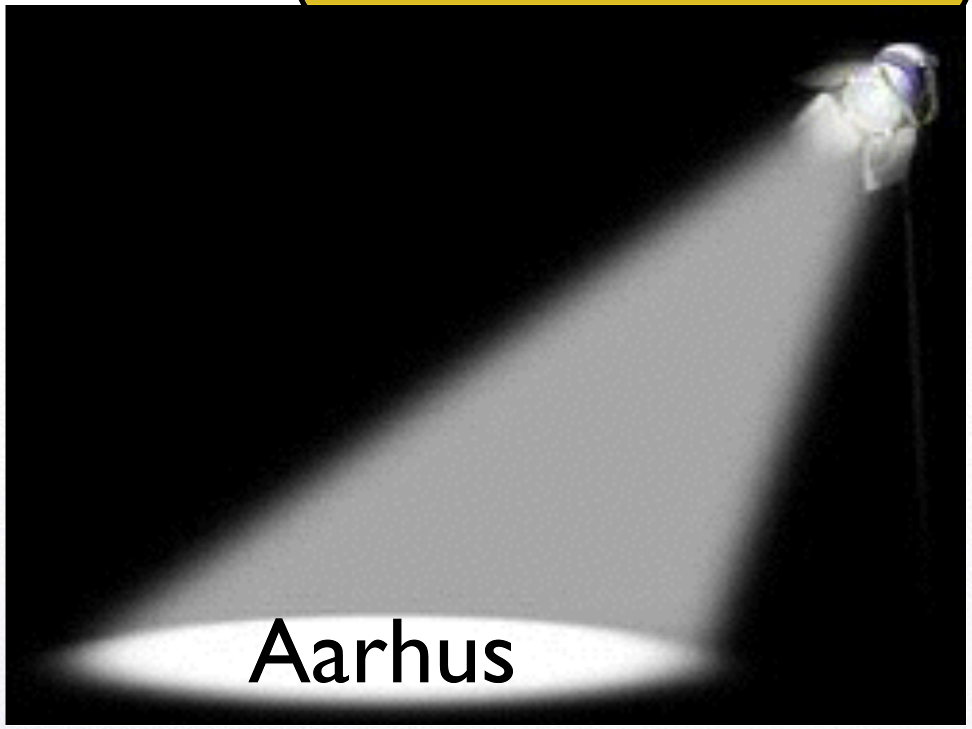
CPN Tools

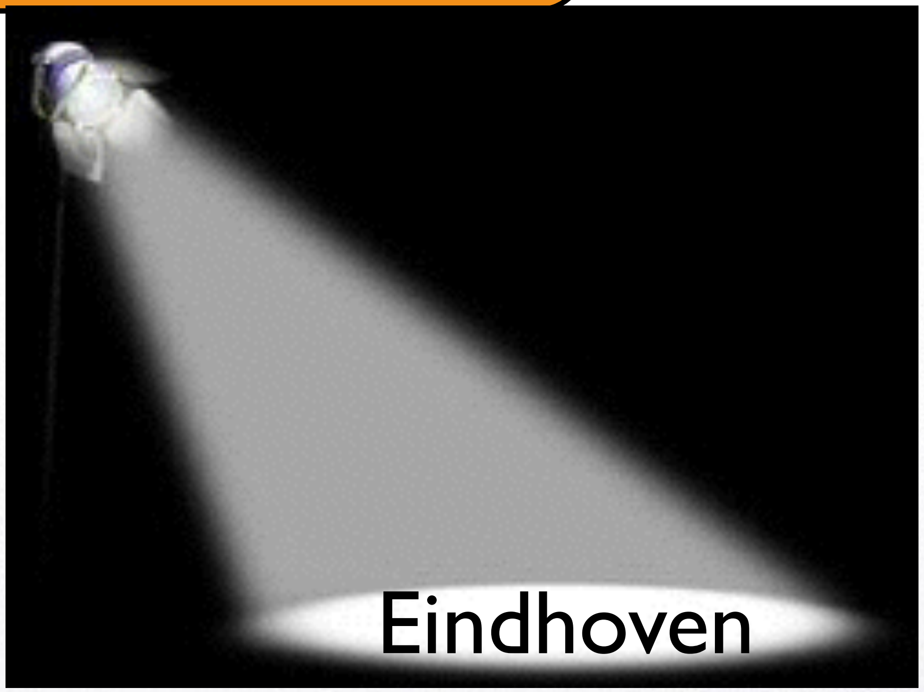
1990-2002

Design CPN

2003-…

CPN Tools





Aarhus Univ., Denmark

Eindhoven Univ., Netherlands

cpntools.org

CPN Tools
- Access/CPN
- Books
- Documentation
- Download
- FAQ
- Getting Started
- Grade/CPN
- Knowledge Base
- Licensing
- Support
- Contact
- Publications

Search

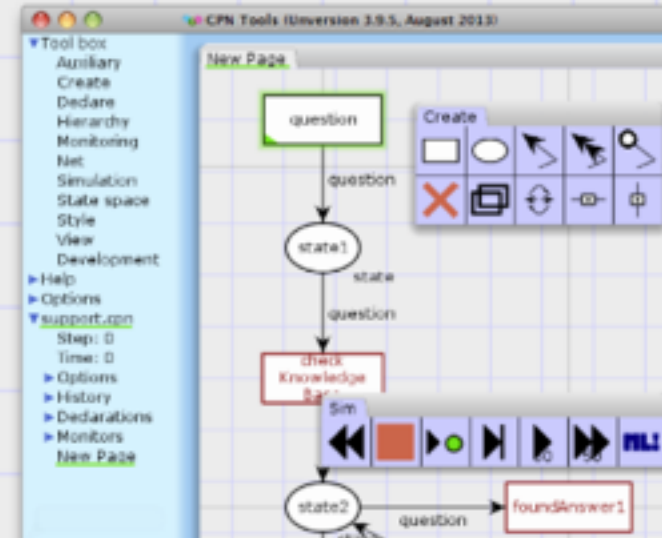
G+1 158

CPN T... 1.1K likes
Like Page
Be the first of your friends to like this

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

CPN Tools is a tool for *editing*, *simulating*, and *analyzing* Colored Petri nets.

The tool features incremental syntax checking and code generation, which take place while a net is being constructed. A fast simulator efficiently handles untimed and timed nets. Full and partial state spaces can be generated and analyzed, and a standard state space report contains information, such as boundedness properties and liveness properties.



New Features in Version 4.0

- Declarative constraints
- 3rd part extensions
- Simplified use of non-colored nets
- Support for export to PNML
- Support for real and time colorsets
- Improved support for time (time intervals and state-space reduction)
- Simplified state-space analysis
- Fresh new look

CPN Tools is originally developed by the CPN Group at Aarhus University from 2000 to 2010. The main architects behind the tool are Kurt Jensen, Søren Christensen, Lars M. Kristensen, and Michael Westergaard. From the autumn of 2010, CPN Tools is transferred to the AIS group, Eindhoven University of Technology, The Netherlands.

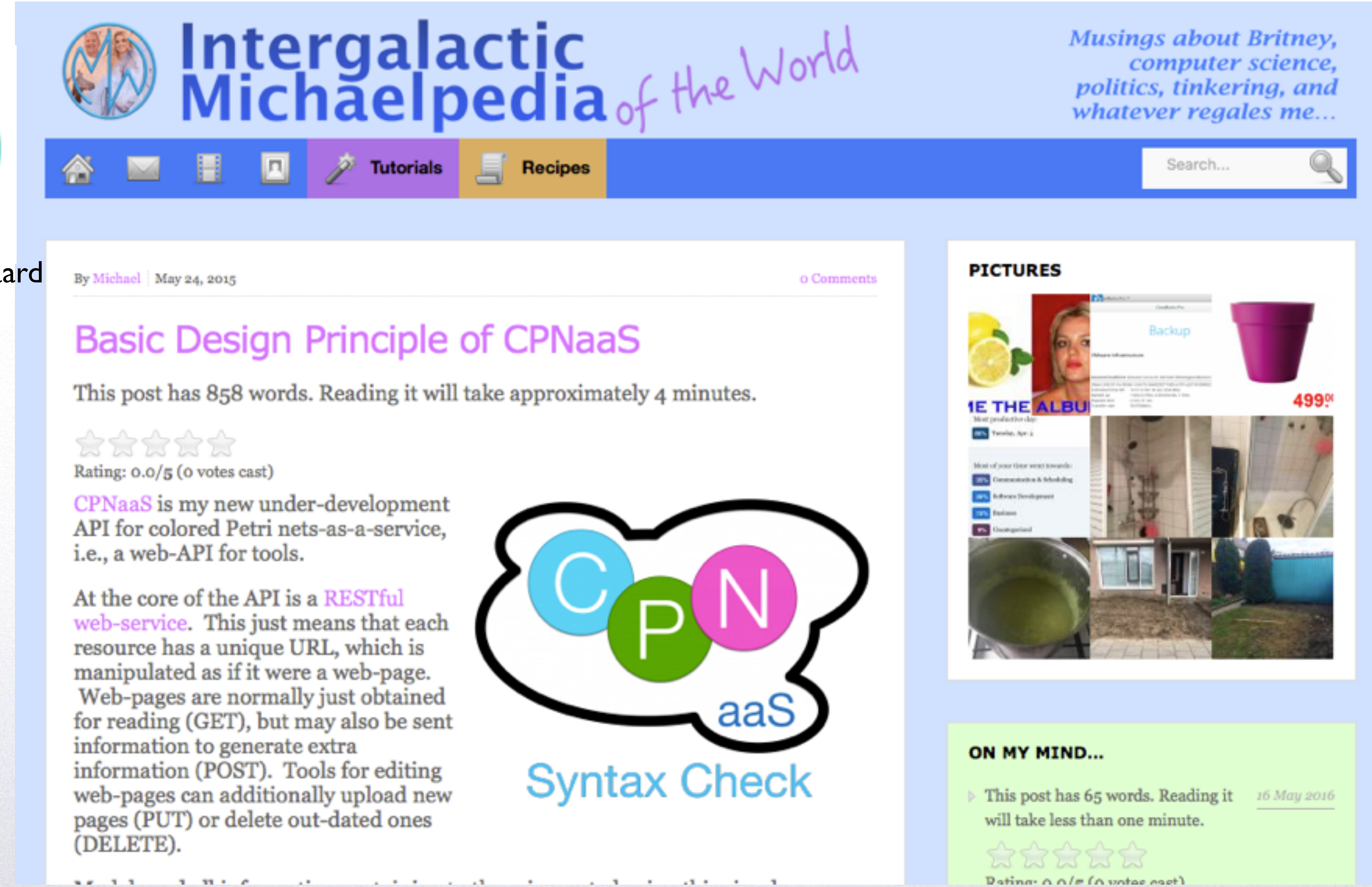
Latest Version of CPN Tools

The latest released version is version **4.0.1** from February 2015. Get it from the Download page. For a full list of new features for each version of CPN Tools, Access/CPN, and Grade/CPN refer to the Whats New?-list.

Michael's blog on CPN Tools

- Is Google's Go-bot a Breakthrough for Artificial Intelligence? (2016/03/25 22:12)
- Improved Paradigm for Analysis, Verification

Michael Westergaard



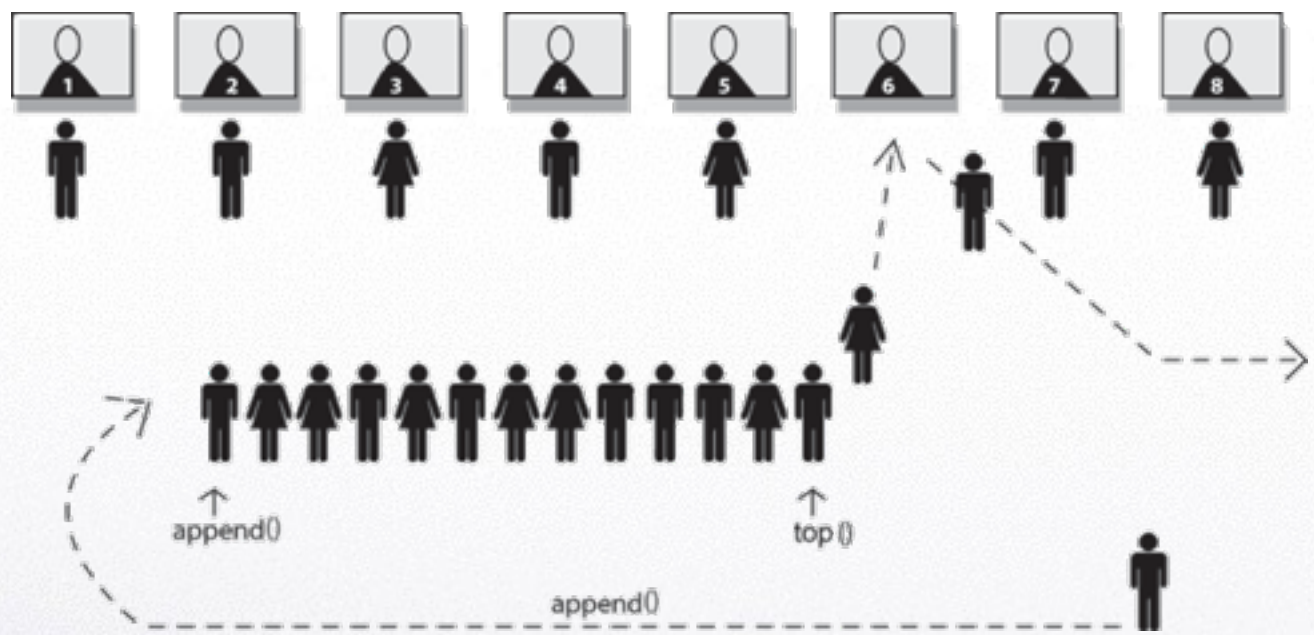
What is a model?

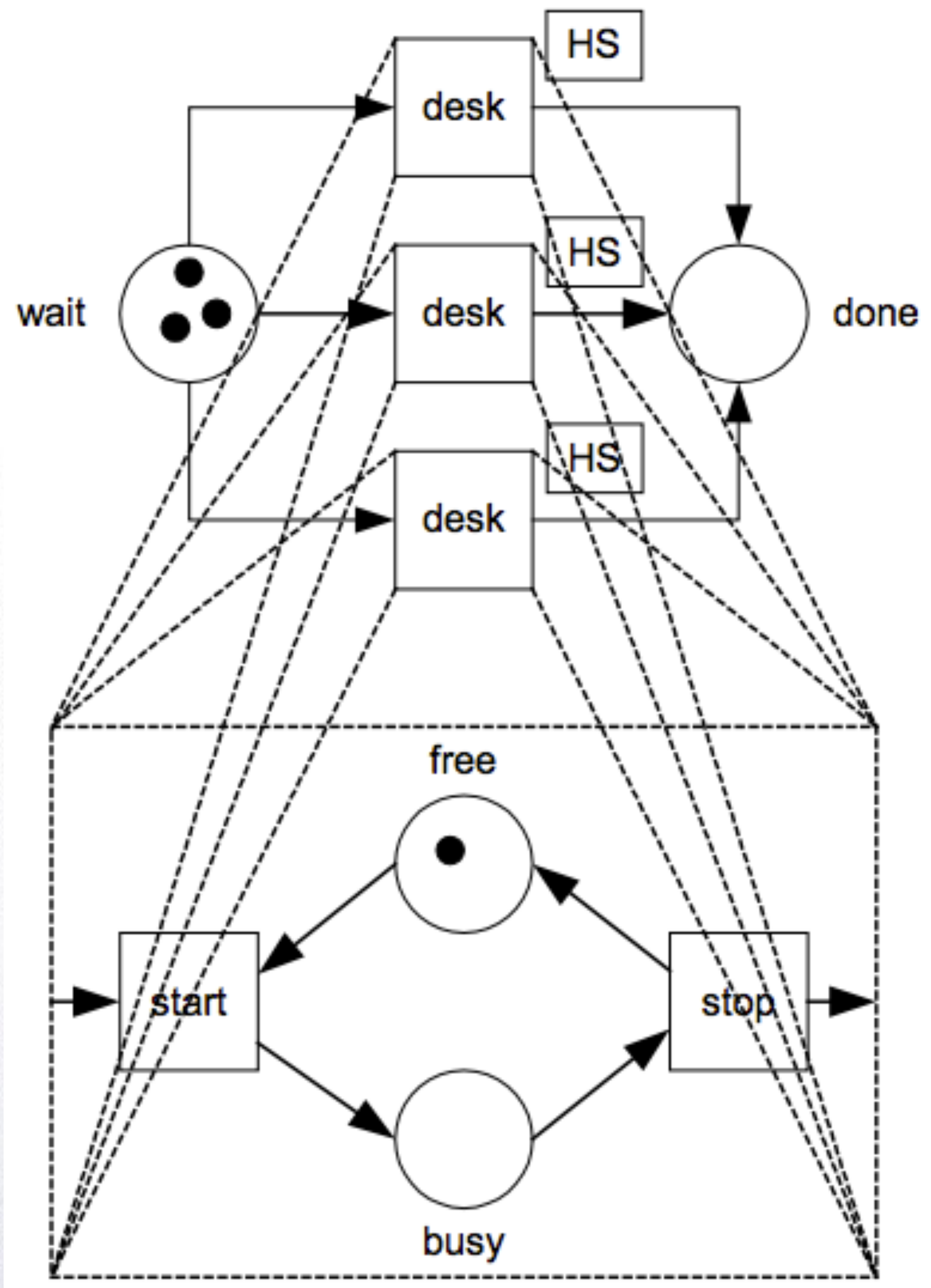
model and modelling

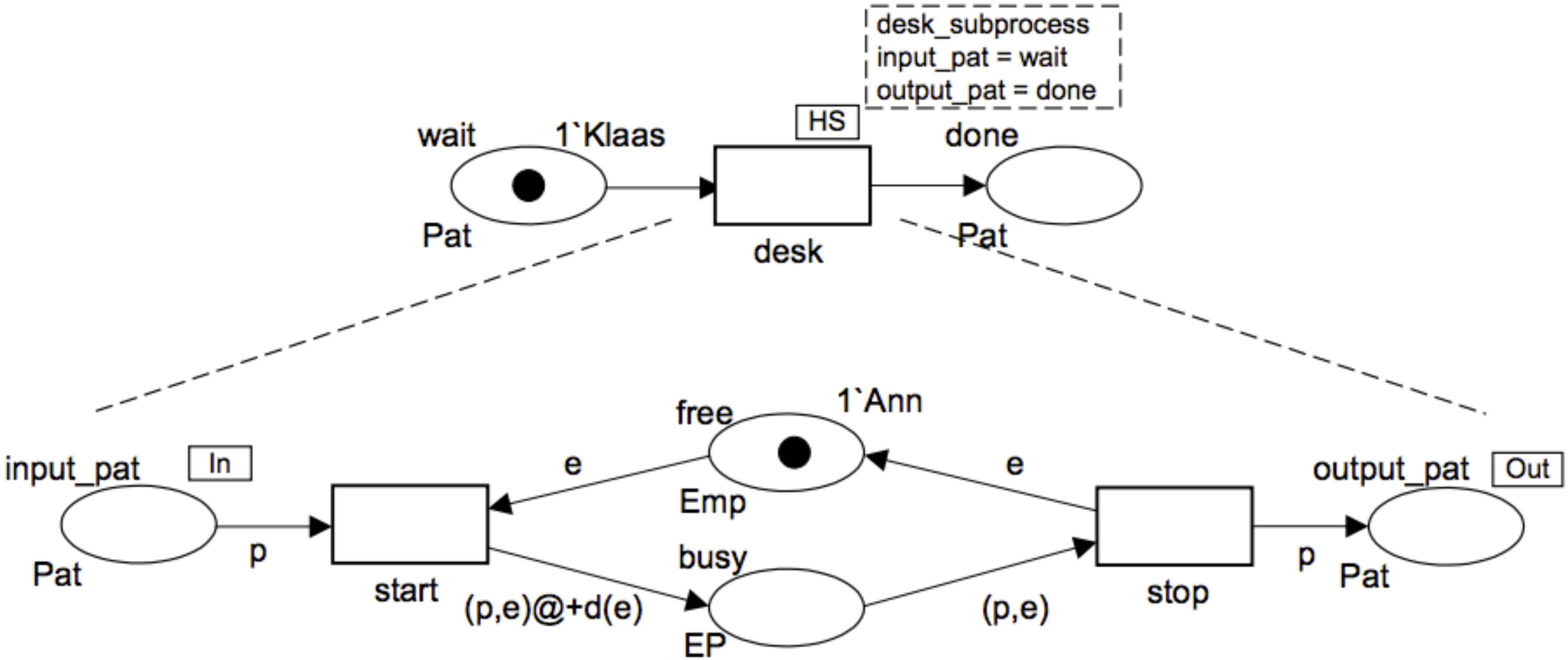
in painting, the *use of light and shade to simulate volume in the representation of solids*. In sculpture the terms denote a technique involving the use of a pliable material such as clay or wax. As opposed to carving, modelling permits addition as well as subtraction of material and lends itself to freer handling and change of intention. The technique is exemplified also by those works in cast metal and plaster that are made from the mold of a clay original. The mold is made by the process of cire perdue. The noun model is used to describe such an original and also *any three-dimensional scale model for a larger or more elaborate project in architecture, landscaping, or industry*. It also denotes a person or object used as an aid to representation in painting.

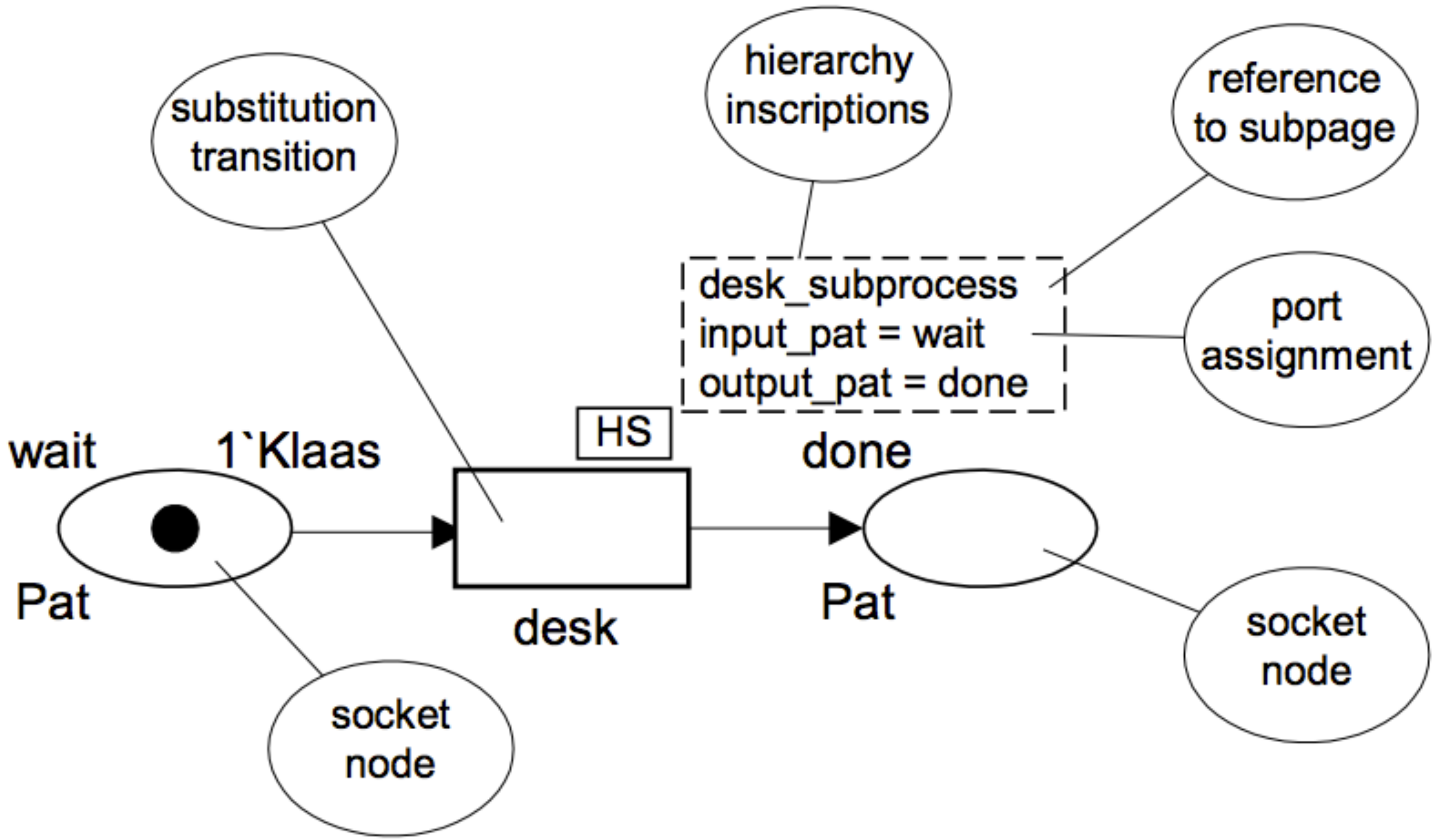
The Columbia Encyclopaedia, Sixth Edition. 2001.

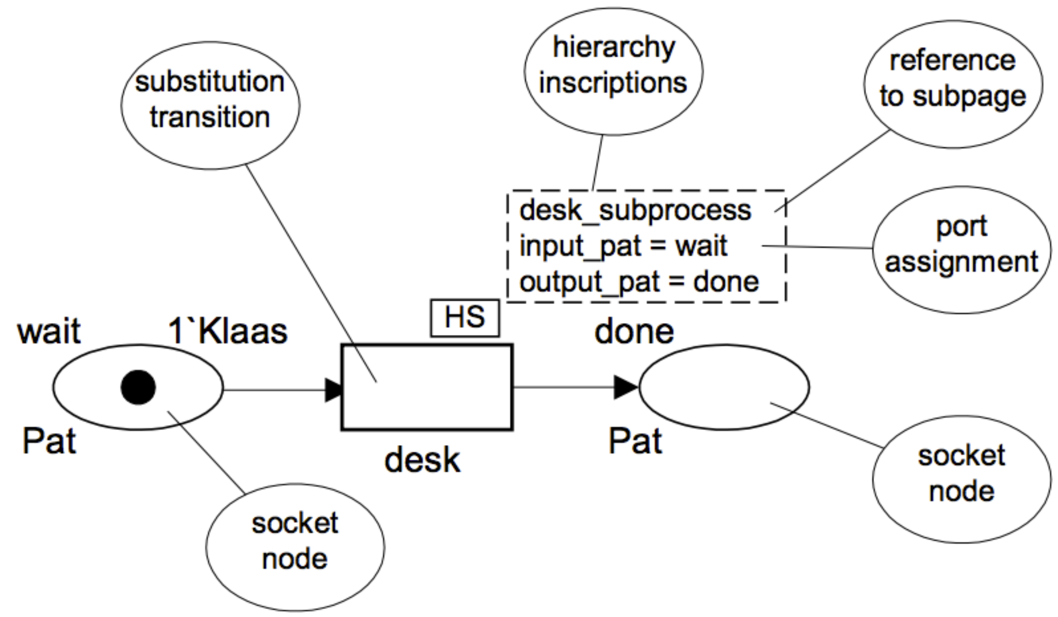
Abstract representation, scale model of future design

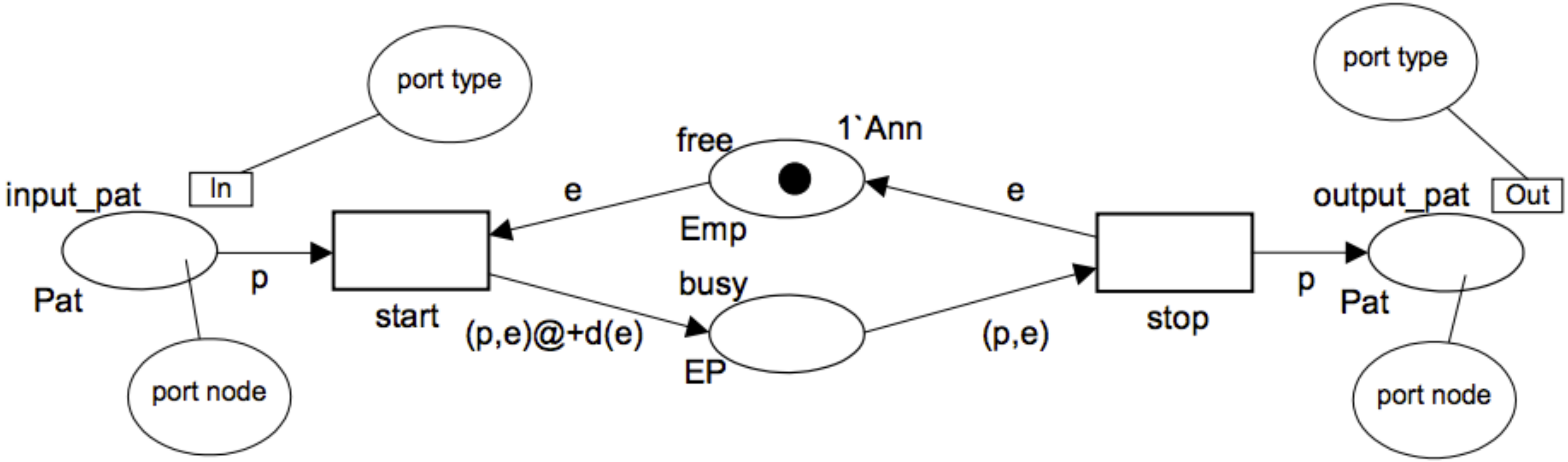

append()
top ()
append()


HS
desk
HS
wait
desk
done
HS
desk
free
start
stop
busy

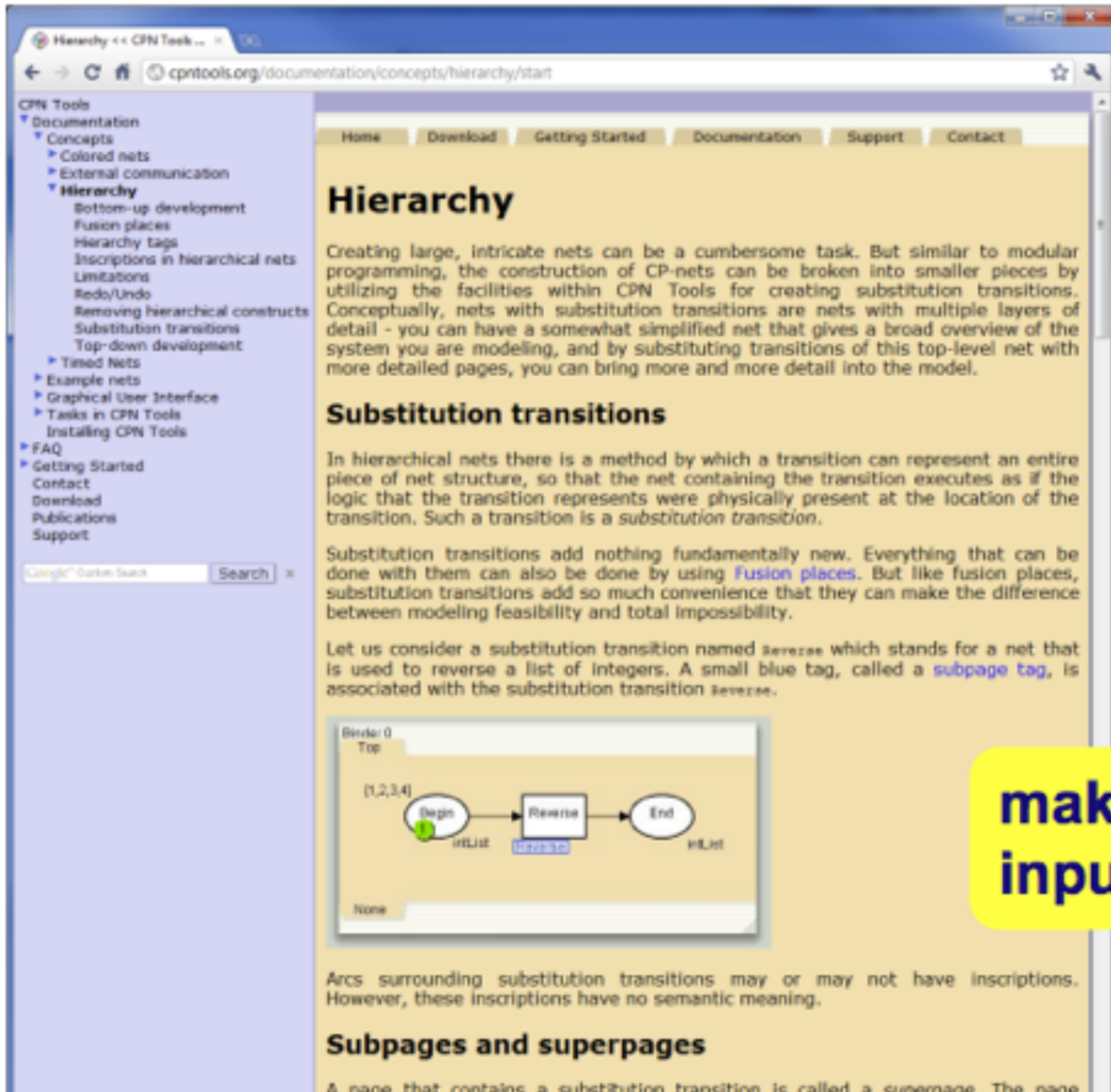

desk_subprocess
input_pat = wait
output_pat = done
HS
wait
1`Klaas
done
Pat
desk
Pat
free
1`Ann
input_pat
In
e
e
output_pat
Out
Emp
Pat
p
busy
p
Pat
start
stop
(p,e)@+d(e)
(p,e)
EP


hierarchy
inscriptions
substitution
transition
reference
to subpage
desk_subprocess
input_pat = wait
output_pat = done
port
assignment
HS
wait
1`Klaas
done
Pat
desk
Pat
socket
node
socket
node


substitution transition
hierarchy inscriptions
reference to subpage
desk_subprocess
input_pat = wait
output_pat = done
port assignment
HS
wait
1`Klaas
Pat
desk
done
Pat
socket node
socket node


port type
In
input_pat
Pat
p
port node
start
free
1`Ann
e
Emp
busy
(p,e)@+d(e)
EP
e
(p,e)
stop
p
port type
output_pat
Out
Pat
port node

CPN Tools


Hierarchy << CPN Tools
cpntools.org/documentation/concepts/hierarchy/start

CPN Tools
- Documentation
  - Concepts
    - Colored nets
    - External communication
    - **Hierarchy**
      - Bottom-up development
      - Fusion places
      - Hierarchy tags
      - Inscriptions in hierarchical nets
      - Limitations
      - Redo/Undo
      - Removing hierarchical constructs
      - Substitution transitions
      - Top-down development
    - Timed Nets
  - Example nets
  - Graphical User Interface
  - Tasks in CPN Tools
  - Installing CPN Tools
- FAQ
- Getting Started
- Contact
- Download
- Publications
- Support

Search

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

Hierarchy

Creating large, intricate nets can be a cumbersome task. But similar to modular programming, the construction of CP-nets can be broken into smaller pieces by utilizing the facilities within CPN Tools for creating substitution transitions. Conceptually, nets with substitution transitions are nets with multiple layers of detail - you can have a somewhat simplified net that gives a broad overview of the system you are modeling, and by substituting transitions of this top-level net with more detailed pages, you can bring more and more detail into the model.

Substitution transitions

In hierarchical nets there is a method by which a transition can represent an entire piece of net structure, so that the net containing the transition executes as if the logic that the transition represents were physically present at the location of the transition. Such a transition is a *substitution transition*.

Substitution transitions add nothing fundamentally new. Everything that can be done with them can also be done by using Fusion places. But like fusion places, substitution transitions add so much convenience that they can make the difference between modeling feasibility and total impossibility.

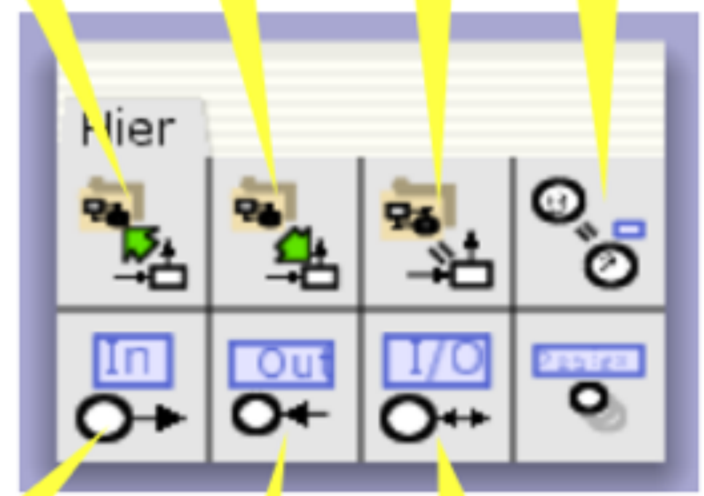
Let us consider a substitution transition named `Reverse` which stands for a net that is used to reverse a list of integers. A small blue tag, called a subpage tag, is associated with the substitution transition `Reverse`.

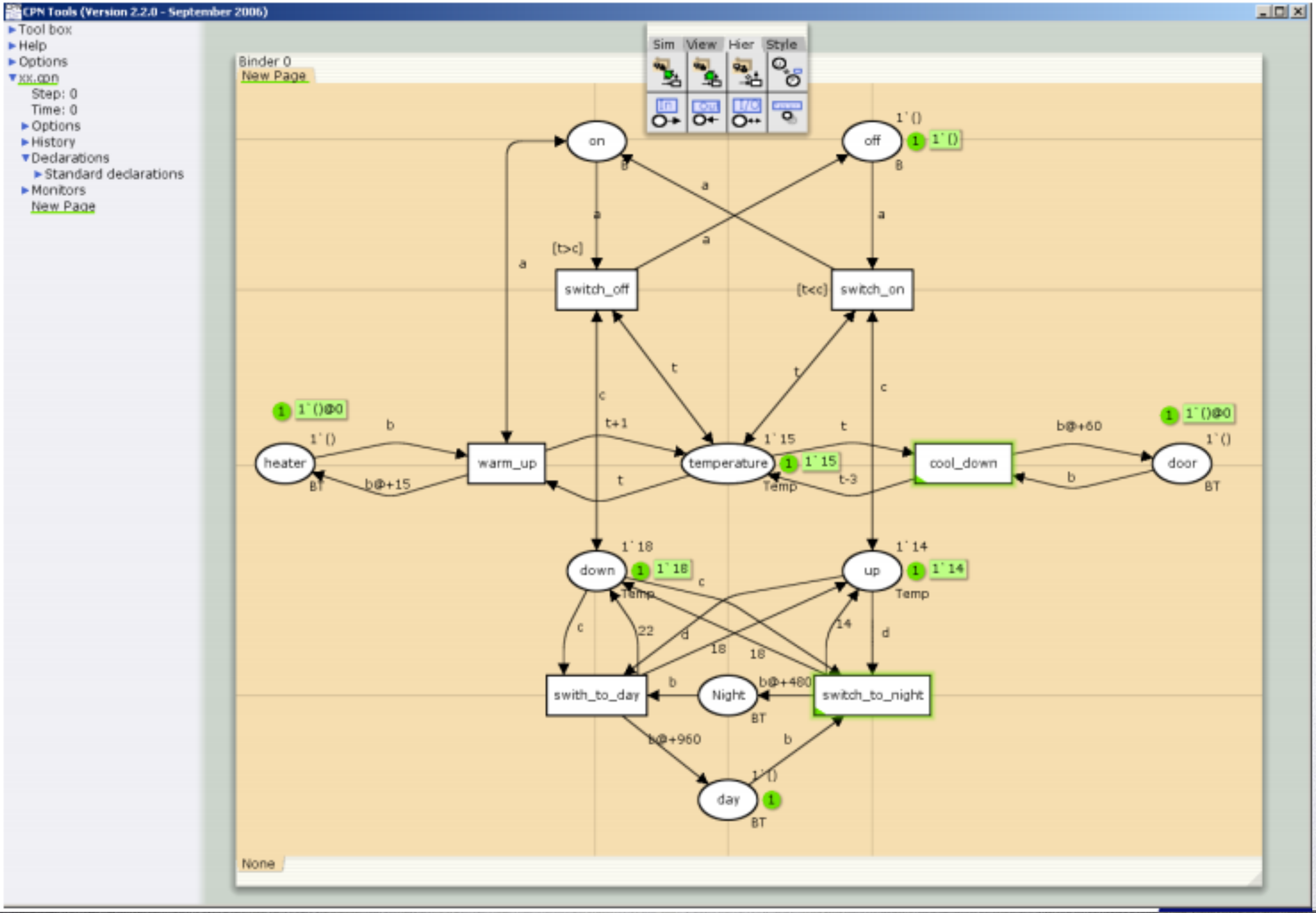
Binder 0
Top
[1,2,3,4]
Begin
Reverse
End
IntList
IntList
None

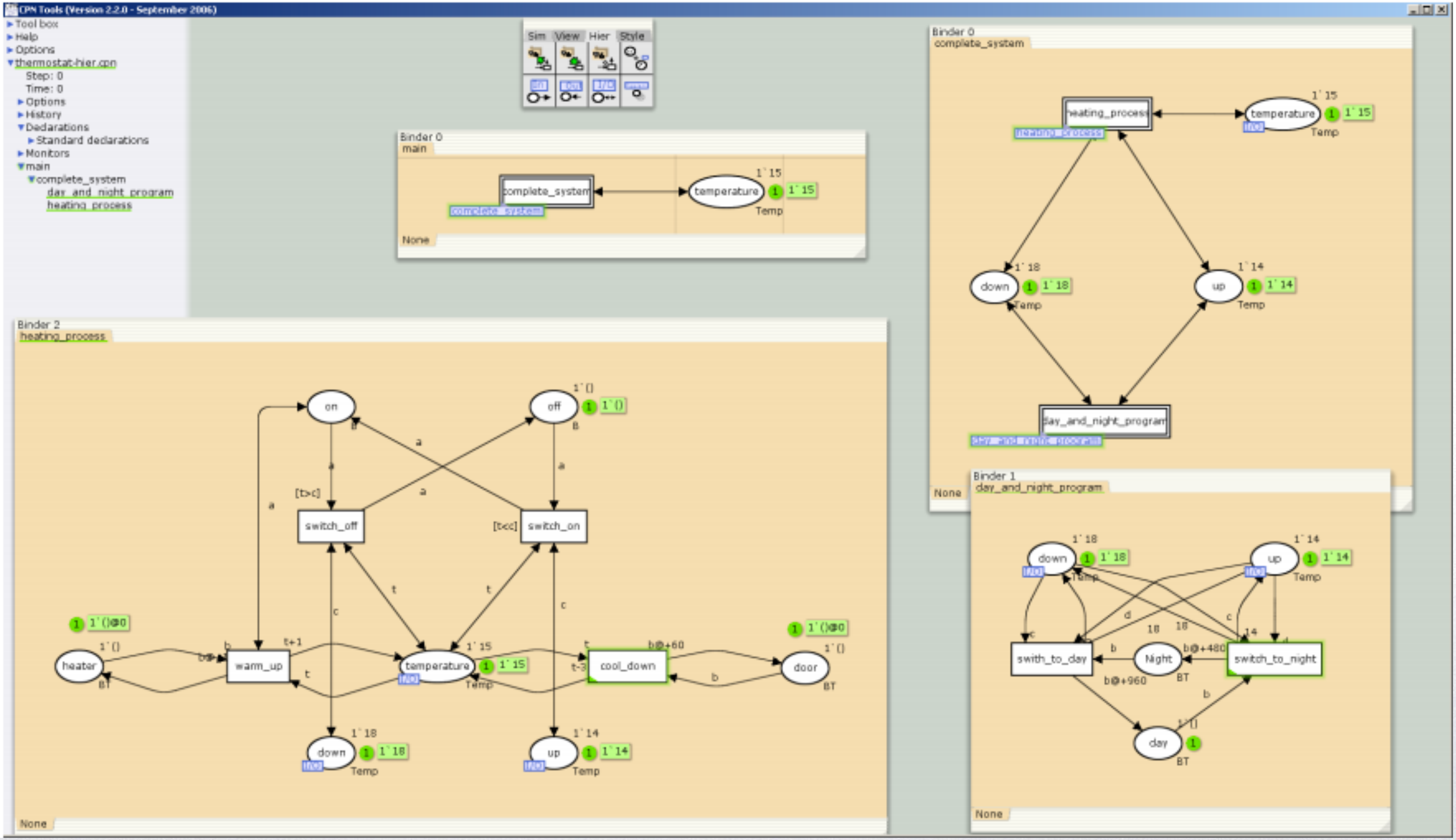
Arcs surrounding substitution transitions may or may not have inscriptions. However, these inscriptions have no semantic meaning.

Subpages and superpages

A page that contains a substitution transition is called a *superpage*. The page

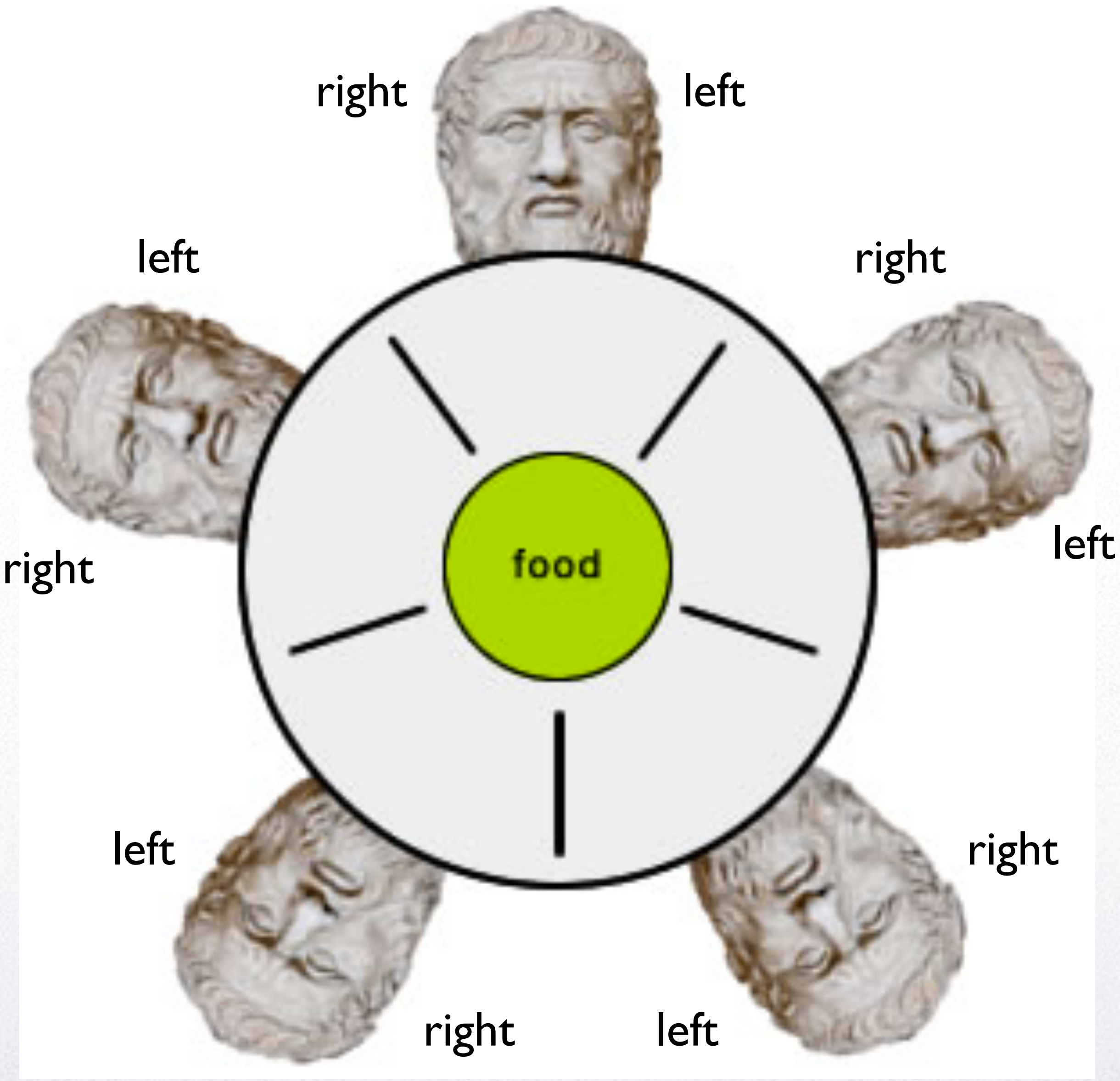



CPN Tools (Version 2.2.0 - September 2006)
Tool box
Help
Options
xx.cpn
Step: 0
Time: 0
Options
History
Declarations
Standard declarations
Monitors
New Page
Binder 0
New Page
Sim
View
Hier
Style
on
off
switch_off
switch_on
heater
warm_up
temperature
cool_down
door
down
up
swith_to_day
Night
switch_to_night
day
None

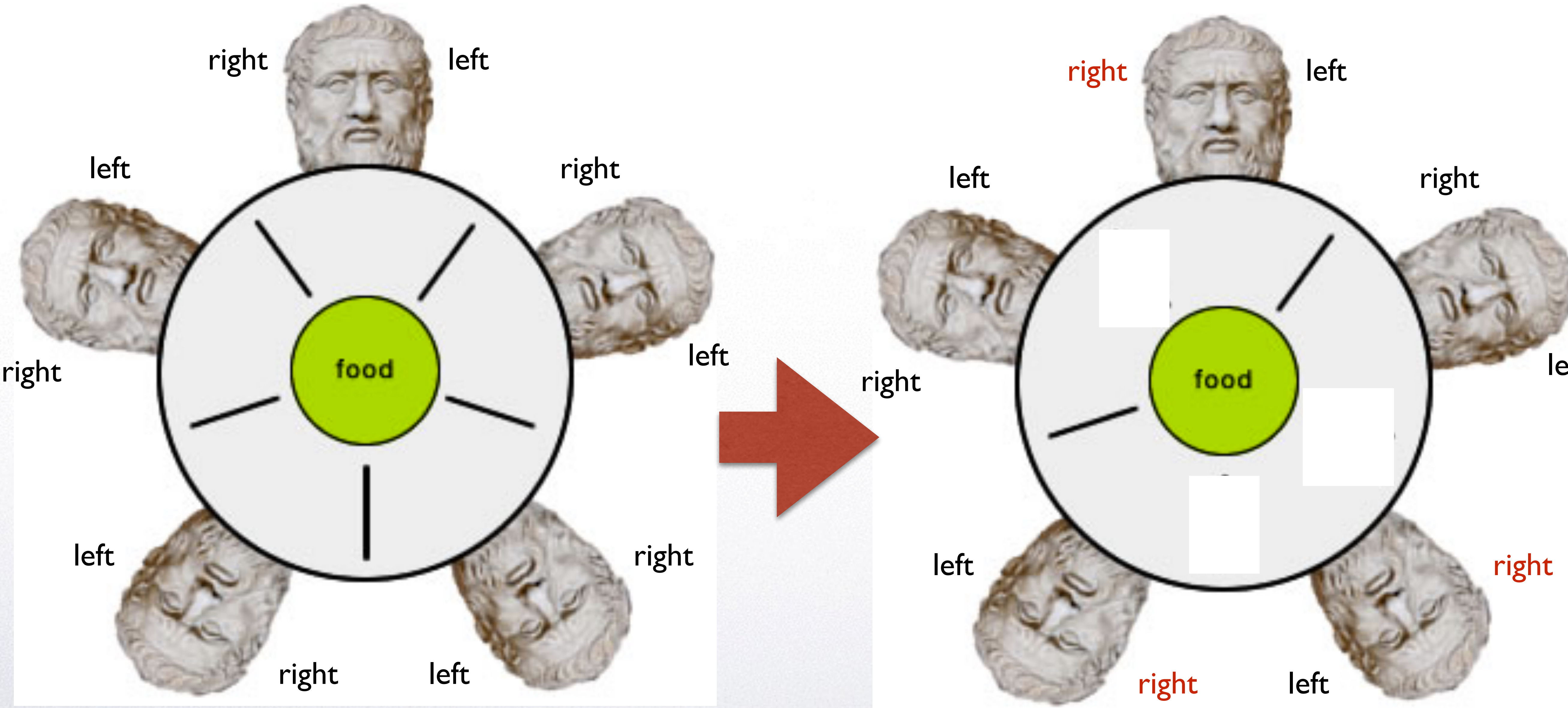

CPN Tools (Version 2.2.0 - September 2006)
Tool box
Help
Options
thermostat-hier.cpn
Step: 0
Time: 0
Options
History
Declarations
Standard declarations
Monitors
main
complete_system
day_and_night_program
heating_process
Binder 0
main
complete_system
temperature
1`15
Temp
None
Binder 0
complete_system
heating_process
temperature
down
up
day_and_night_program
Binder 2
heating_process
on
off
switch_off
switch_on
heater
warm_up
temperature
cool_down
door
down
up
Binder 1
day_and_night_program
swith_to_day
Night
switch_to_night
day

- Make a hierarchical CPN model of five Chinese philosophers alternating between states thinking and eating. To eat two chopsticks are needed. In total there are five chopsticks. The philosophers are sitting in a circle, and need to complete for chopsticks with their direct neighbors (left and right). Assume that both chopsticks need to be taken at the same time.
  Model this using a hierarchical CPN model. Make sure to model the behavior of a philosopher only once and just use the color set BlackToken of type unit.
- Change the model such that philosophers can take one chopstick at a time but avoid deadlocks and a fixed ordering of philosophers.
- Flatten the hierarchical CPN model.

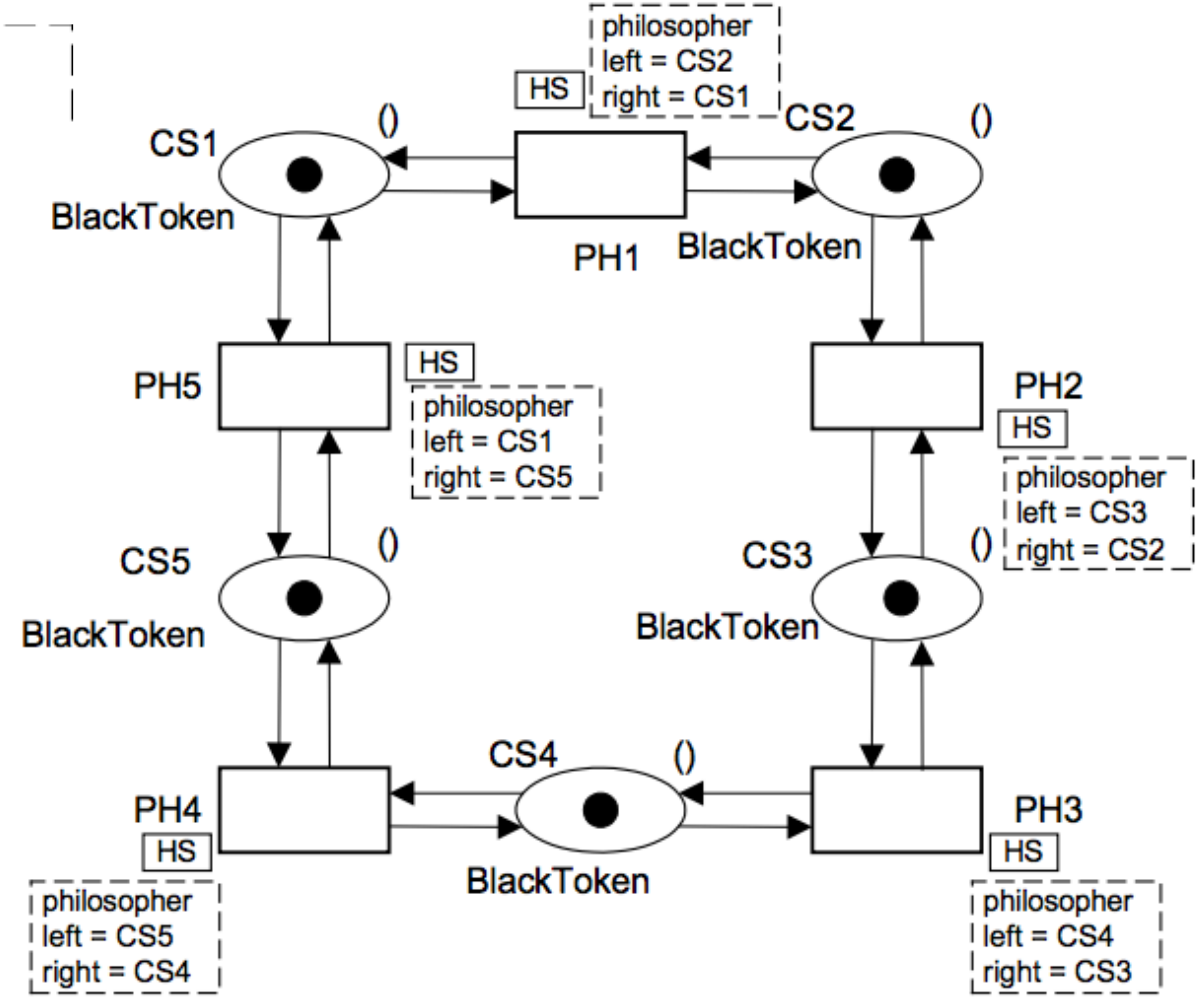
HC


right
left
left
right
right
left
food
left
right
right
left

deadlock


right
left
left
right
right
food
left
left
right
right
left
right
left
left
right
right
food
left
left
right
right
left

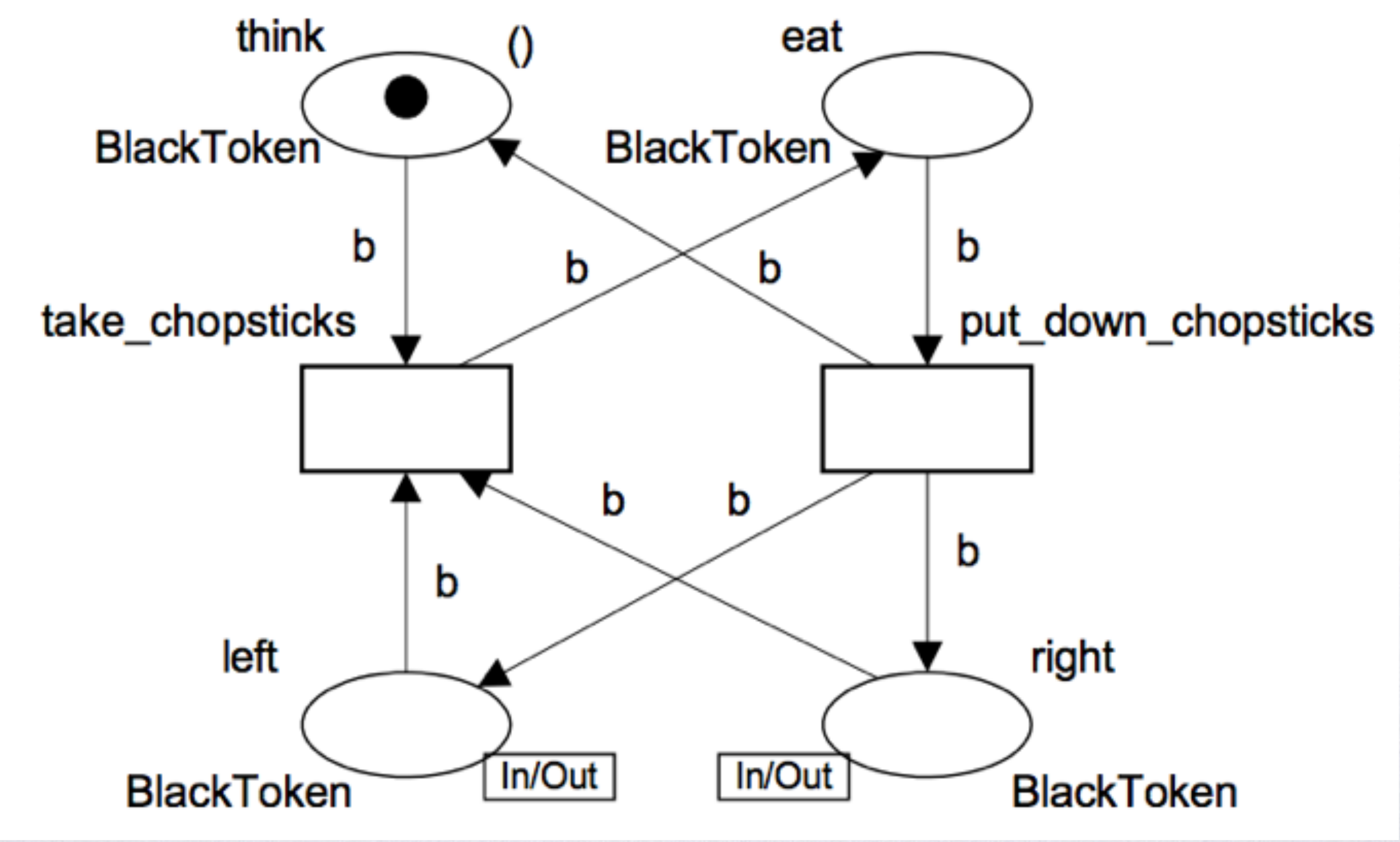
color BlackToken = unit;
var b:BackToken

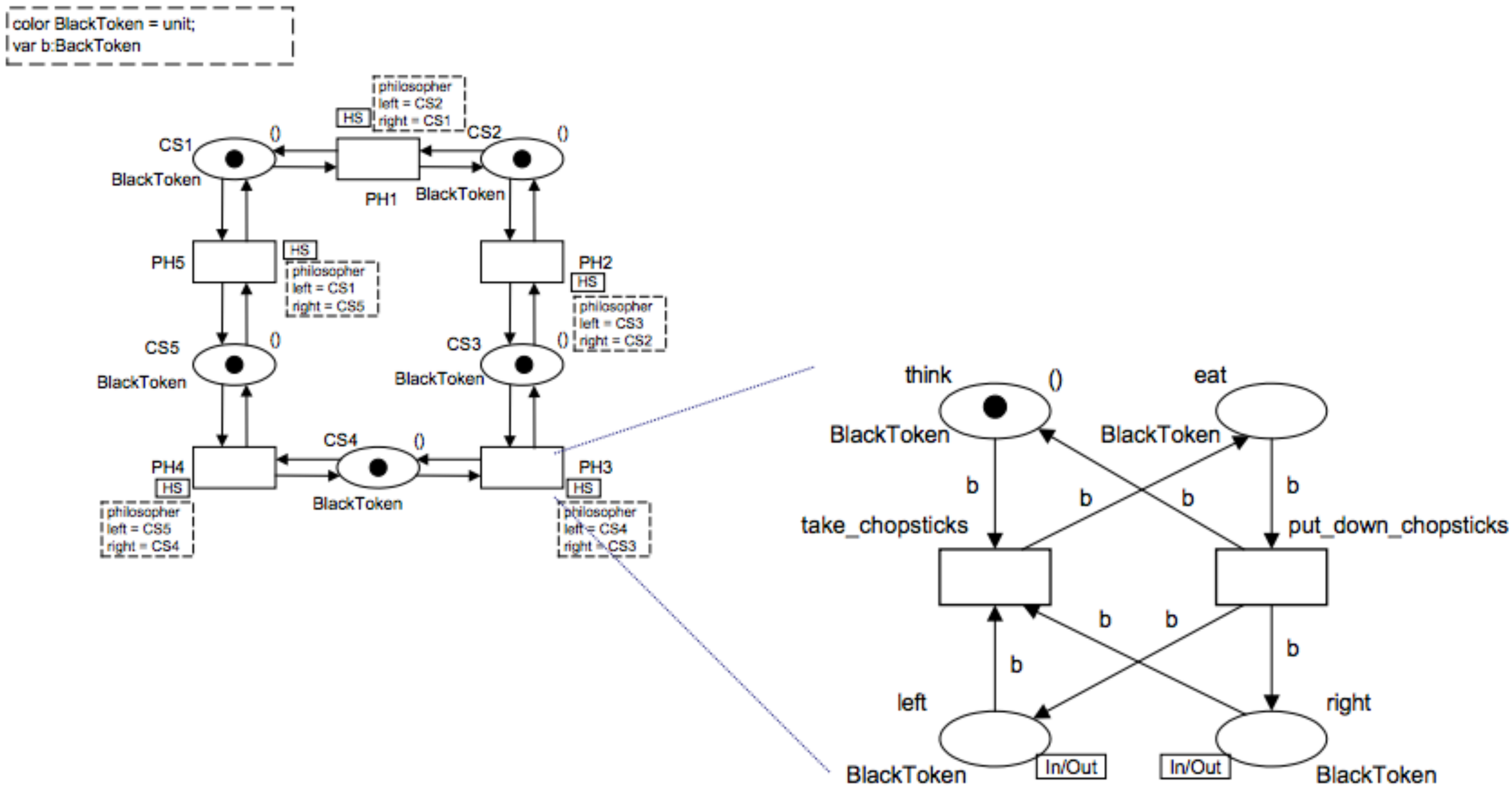


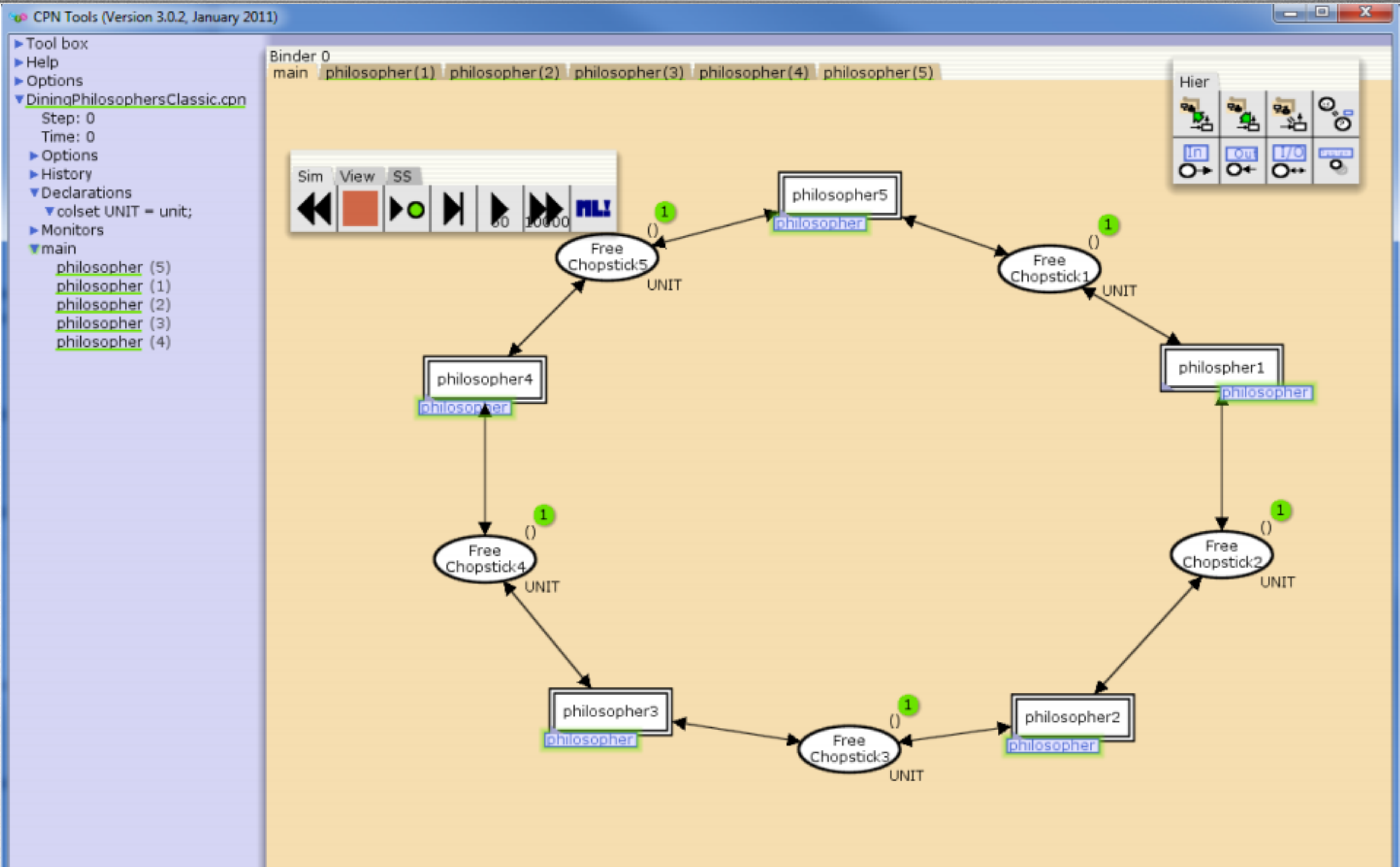
Esta modelagem tem dois aspectos interessantes:

a) não tem muito sentido pegar os sticks e devolver logo em seguida, exceto se retiver os sticks por algum tempo;
b) o estado “comendo" não aparece explicitamente, o que indica que deve ocorrer para cada filósofo em um nível hierárquico mais baixo, dentro das transições.

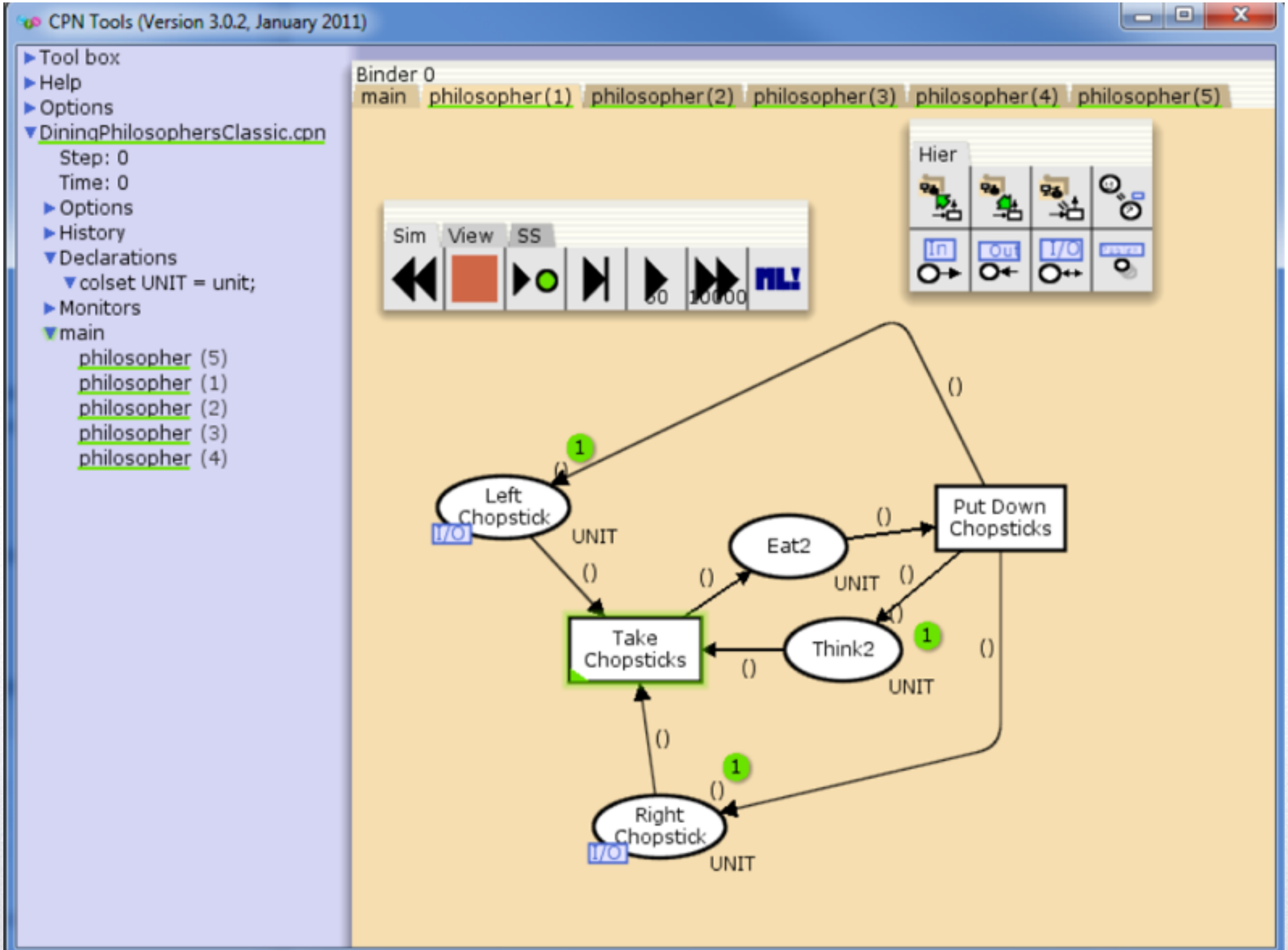
Portanto cada filósofo usa os chopsticks para passar para o estado comendo, e, expandindo cada transição associada a ele temos:

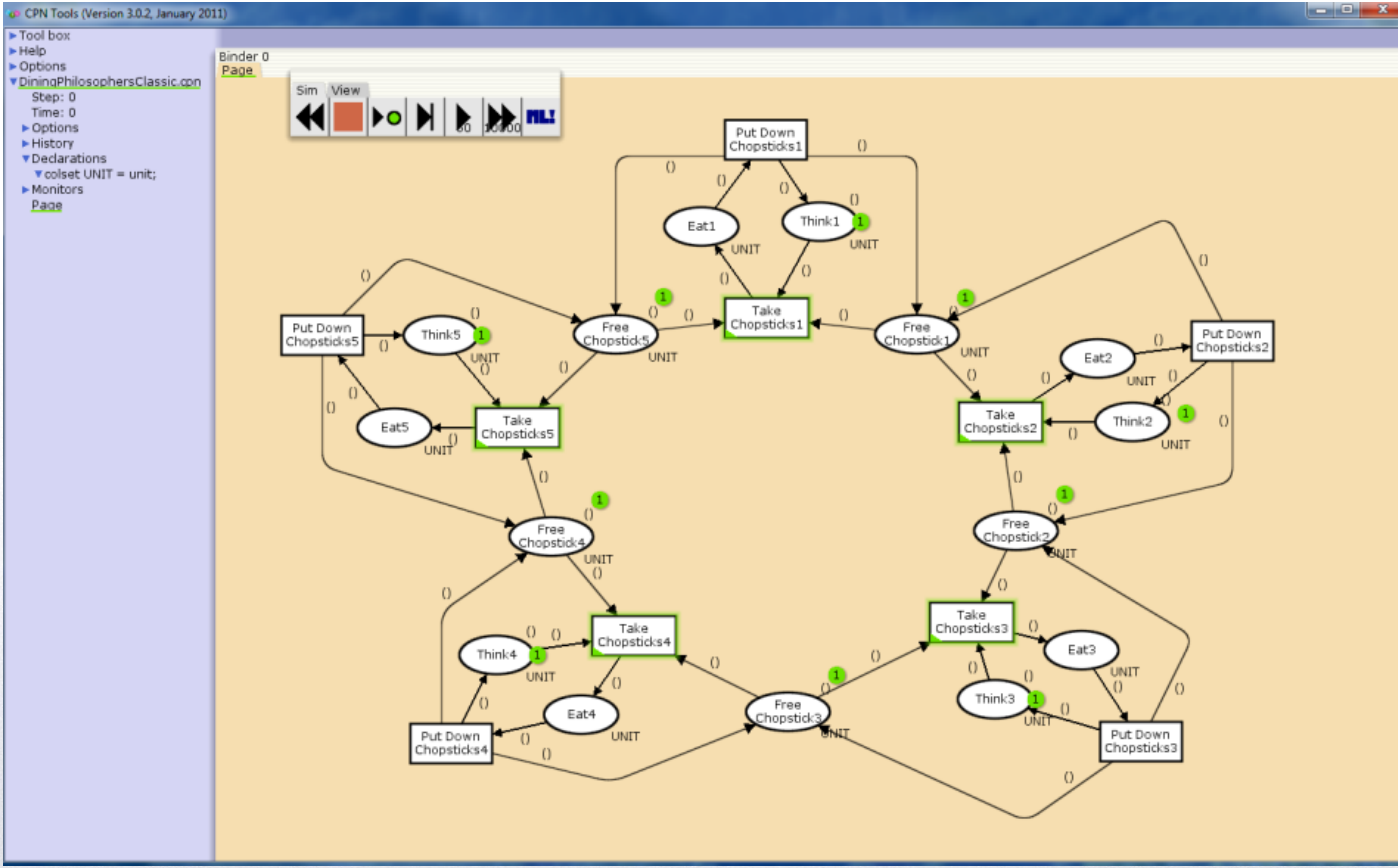



color BlackToken = unit;
var b:BackToken
philosopher
left = CS2
right = CS1
HS
CS1
()
CS2
()
BlackToken
PH1
BlackToken
PH5
HS
philosopher
left = CS1
right = CS5
PH2
HS
philosopher
left = CS3
right = CS2
CS5
()
CS3
()
BlackToken
BlackToken
CS4
()
PH4
HS
philosopher
left = CS5
right = CS4
BlackToken
PH3
HS
philosopher
left = CS4
right = CS3
think
()
eat
BlackToken
BlackToken
b
b
b
b
take_chopsticks
put_down_chopsticks
b
b
b
b
left
right
BlackToken
In/Out
In/Out
BlackToken



CPN Tools (Version 3.0.2, January 2011)
Tool box
Help
Options
DiningPhilosophersClassic.cpn
Step: 0
Time: 0
Options
History
Declarations
colset UNIT = unit;
Monitors
main
philosopher (5)
philosopher (1)
philosopher (2)
philosopher (3)
philosopher (4)
Binder 0
main
philosopher(1)
philosopher(2)
philosopher(3)
philosopher(4)
philosopher(5)
Sim
View
SS
Hier
philosopher5
philosopher1
philosopher2
philosopher3
philosopher4
Free Chopstick1
Free Chopstick2
Free Chopstick3
Free Chopstick4
Free Chopstick5
UNIT


CPN Tools (Version 3.0.2, January 2011)
Tool box
Help
Options
DiningPhilosophersClassic.cpn
Step: 0
Time: 0
Options
History
Declarations
colset UNIT = unit;
Monitors
main
philosopher (5)
philosopher (1)
philosopher (2)
philosopher (3)
philosopher (4)
Binder 0
main
philosopher(1)
philosopher(2)
philosopher(3)
philosopher(4)
philosopher(5)
Sim
View
SS
Hier
In
Out
I/O
Left Chopstick
UNIT
Eat2
Put Down Chopsticks
Take Chopsticks
Think2
Right Chopstick
()
1


CPN Tools (Version 3.0.2, January 2011)
Tool box
Help
Options
DiningPhilosophersClassic.cpn
Step: 0
Time: 0
Options
History
Declarations
colset UNIT = unit;
Monitors
Page
Binder 0
Page
Sim
View
Put Down Chopsticks1
Eat1
Think1
Take Chopsticks1
Free Chopstick1
Free Chopstick5
Put Down Chopsticks5
Think5
Eat5
Take Chopsticks5
Free Chopstick4
Take Chopsticks4
Think4
Eat4
Put Down Chopsticks4
Free Chopstick3
Take Chopsticks3
Eat3
Think3
Put Down Chopsticks3
Free Chopstick2
Take Chopsticks2
Eat2
Think2
Put Down Chopsticks2
UNIT

Inserindo funções

Uma maneira de incrementar as inscrições é incluir elementos de “ordem" superior à que temos até agora (variáveis, expressões, elementos lógicos básicos, if-then-else, etc.).

Uma função é associada a um identificador e uma assinatura que especifica o tipo dos argumentos e o tipo do valor de retorno.

Introduziremos também o conceito de **lista**.

Uma **lista** é uma estrutura homogênea (todos os componentes são do mesmo tipo), composta de uma sucessão de elementos.

Exemplo: [a, b, c, d, e, f, g]
[1, 2, 4, 6, 7, 10]

Uma lista também pode ser definida de forma recursiva, como sendo composta de dois elementos básicos. Para isso definiremos em primeiro lugar a lista vazia [ ].

Uma lista não vazia é composta por dois elementos: head, que é o primeiro elemento da lista e tail que é uma lista (necessáriamente menor que a lista original), e se representa como [head | tail]. No exemplo dado anteriormente,

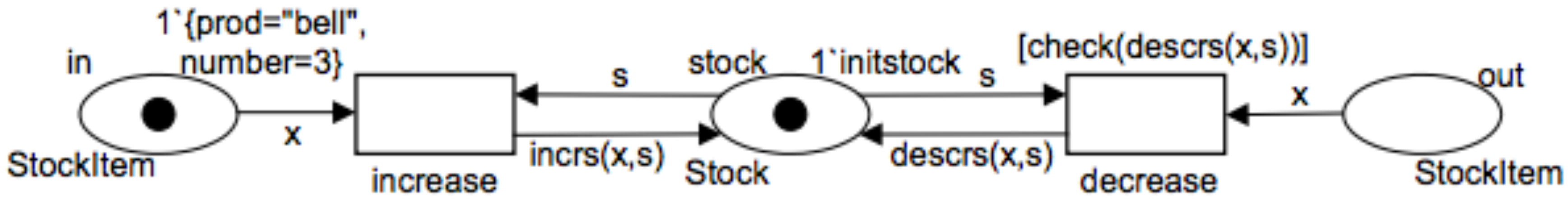
[a, b, c, d, e, f, g] → [a | [b, c, d, e, f, g]]

[1, 2, 4, 6, 7, 10] → [1 | [2, 4, 6, 7, 10]]

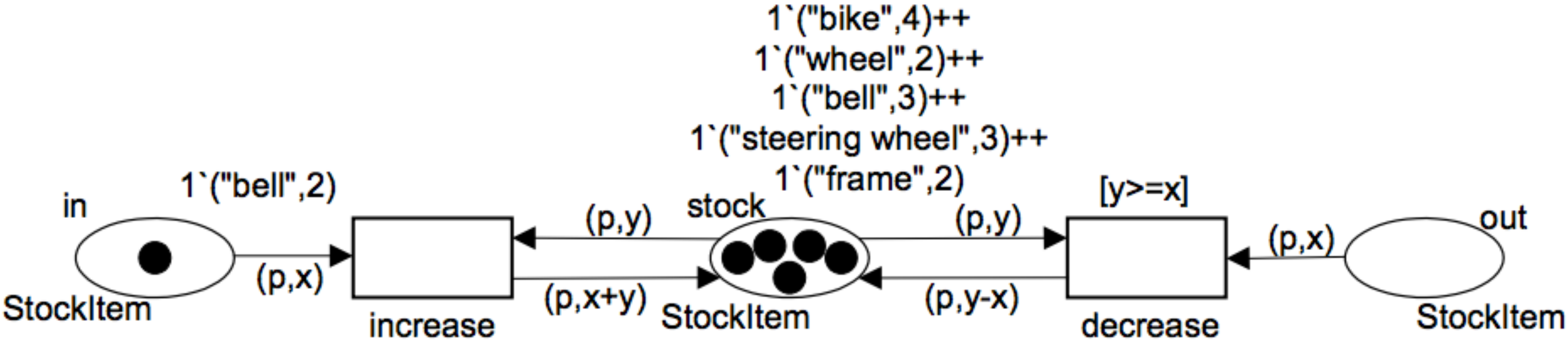
fun totalstock(s:Stock) =

```
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}];
```



```
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.

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
...	...

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

CPN Tools
- Access/CPN
- Books
- Documentation
- Download
- FAQ
- **Getting Started**
  - First steps
- Grade/CPN
- Knowledge Base
- Licensing
- Support
- Contact
- Publications

Search

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

Getting Started

Modeling With Coloured Petri Nets

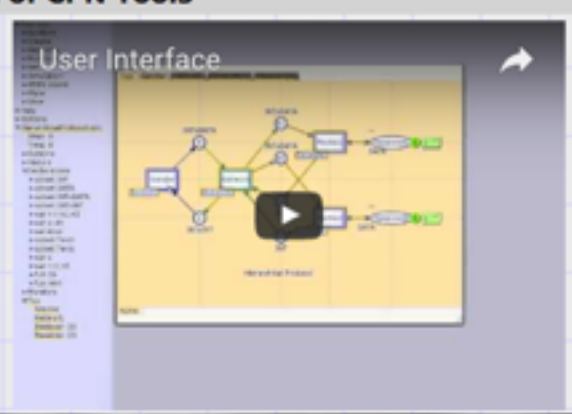
In all of the following and on all of these pages, we assume that you are familiar with coloured Petri nets and have at least some idea of constructing modules using the formalism. You can learn more about the modeling and the formalism in one or both of these books:

The User Interface of CPN Tools

First steps

Graphical User Interface

- **Index**
- **Marking menus**
- **Other tools**
- **Palette tools**



Uma outra extensão importante para as redes de Petri é o tempo. O tempo é uma grandeza positiva que corre inexoravelmente do passado para o futuro, isto é, desde um instante quando se começou a contar o tempo para um tempo futuro.

Assim, o tempo seria normalmente uma grandeza contínua (teoricamente). Entretanto o tempo de simulação que costumamos contar é discreto e marca tão somente a sucessão de estados. É possível portanto ter um *tempo discreto*, contado por um relógio independente dos eventos analisados.

Chandler Ramchandani, em sua dissertação de mestrado no MIT em Setembro de 1973 propôs a primeira temporização que se tem noticia das Redes de Petri.

ANALYSIS OF ASYNCHRONOUS CONCURRENT SYSTEMS BY TIMED PETRI NETS

by

Chander Ramchandani

B.Tech., Indian Institute of Technology, New Delhi
(1968)

S.M., Massachusetts Institute of Technology
(1970)

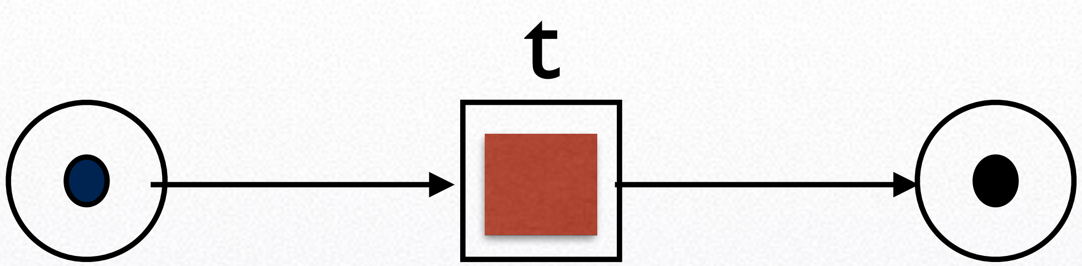
SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September, 1973

Signature of Author _______________________
Department of Electrical Engineering, July 3, 1973

Certified by _______________________
Thesis Supervisor

Accepted by _______________________
Chairman, Departmental Committee on Graduate Students

No caso de uma Timed Petri Net estilo Ramchandani temos que uma transição habilitada (a proposta era basicamente para transições) que não fosse instantânea teria as marcas das pré-condições recolhidas à transição e esta ficaria habilitada e contando o tempo de simulação até que pudesse ser disparada.



Mais recentemente estas redes são associadas a um tempo (real) especificado e são simplesmente chamadas de Deterministic Timed Petri Nets

Chapter 16

Timed Petri Nets

José Reinaldo Silva and Pedro M. G. del Foyo

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/50117

1. Introduction

In the early 60's a young researcher in Darmstadt looked for a good representation for communicating systems processes that were mathematically sound and had, at the same time, a visual intuitive flavor. This event marked the beginning of a schematic approach that become very important to the modeling of distributed systems in several and distinct areas of knowledge, from Engineering to biologic systems. Carl Adam Petri presented in 1962 his PHD which included the first definition of what is called today a Petri Net. Since its creation Petri Nets evolved from a sound representation to discrete dynamic systems into a general schemata, capable to represent knowledge about processes and (discrete and distributed) systems according to their internal relations and not to their work domain. Among other advantages, that feature opens the possibility to reuse some experiences acquired in the design of known and well tested systems while treating new challenges.

In the conventional approach, the key issue for modeling is the partial ordering among constituent events and the properties that arise from the arrangement of state and transitions once some basic interpretation rules are preserved. Such representation can respond from several systems of practical use where the foundation for analysis is based in reachability and other property analysis. However, there are some cases where such approach is not enough to represent processes completely, for instance, when the assumption that all transitions can fire instantaneously is no longer a good approximation. In such cases a time delay can be associated to firing transitions. This is absolutely equivalent (in a broader sense) to say that firing pre-conditions must hold for a time delay before the firing is completed. The first approach is called T-time Petri Net and the second P-time Petri Nets.

Thus, what we have in conclusion is that even in a hypothesis that we should consider only firing pre-conditions[1] [31][19] a time delay is associated with a transition location and consequently to its firing. Several applications in manufacturing, business, workflow and

[1] In many text books and review articles the enabling condition is presented using only firing pre-conditions as a requirement. This can be justified since the use of this week firing condition is sufficient if a complete net, that is, that includes its dual part, is used

INTECH
open science | open minds

Definition 6. [Timed Petri Net] A timed Petri net is a six-tuple

$$N = (P, T, A, w, M_0, f)$$

where

(P, T, A, w, M_0) is a marked Petri net

$f : T \to \mathbb{R}^+$ is a firing time function that assigns a positive real number to each transition on the net

Therefore, the firing rule has to be modified in order to consider time elapses in the transition firing. If an enabled transition $t_j \in enb(M)$ then it will fire after $f(t_j)$ times units since it became enabled. The system state is not only determined by the net marking but also by a timer attached to every enabled transition in the net.

Silva, J. R. and del Foyo, P.M.G.

Fim