PMR 5237

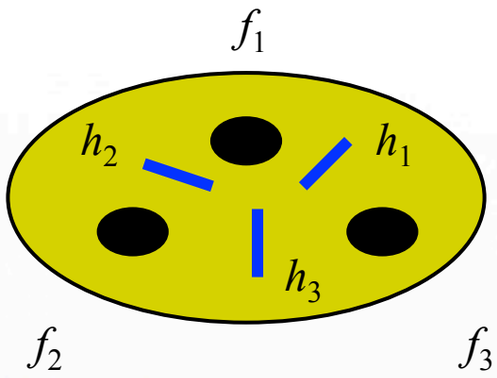
Modelagem e Design de Sistemas Discretos em Redes de Petri

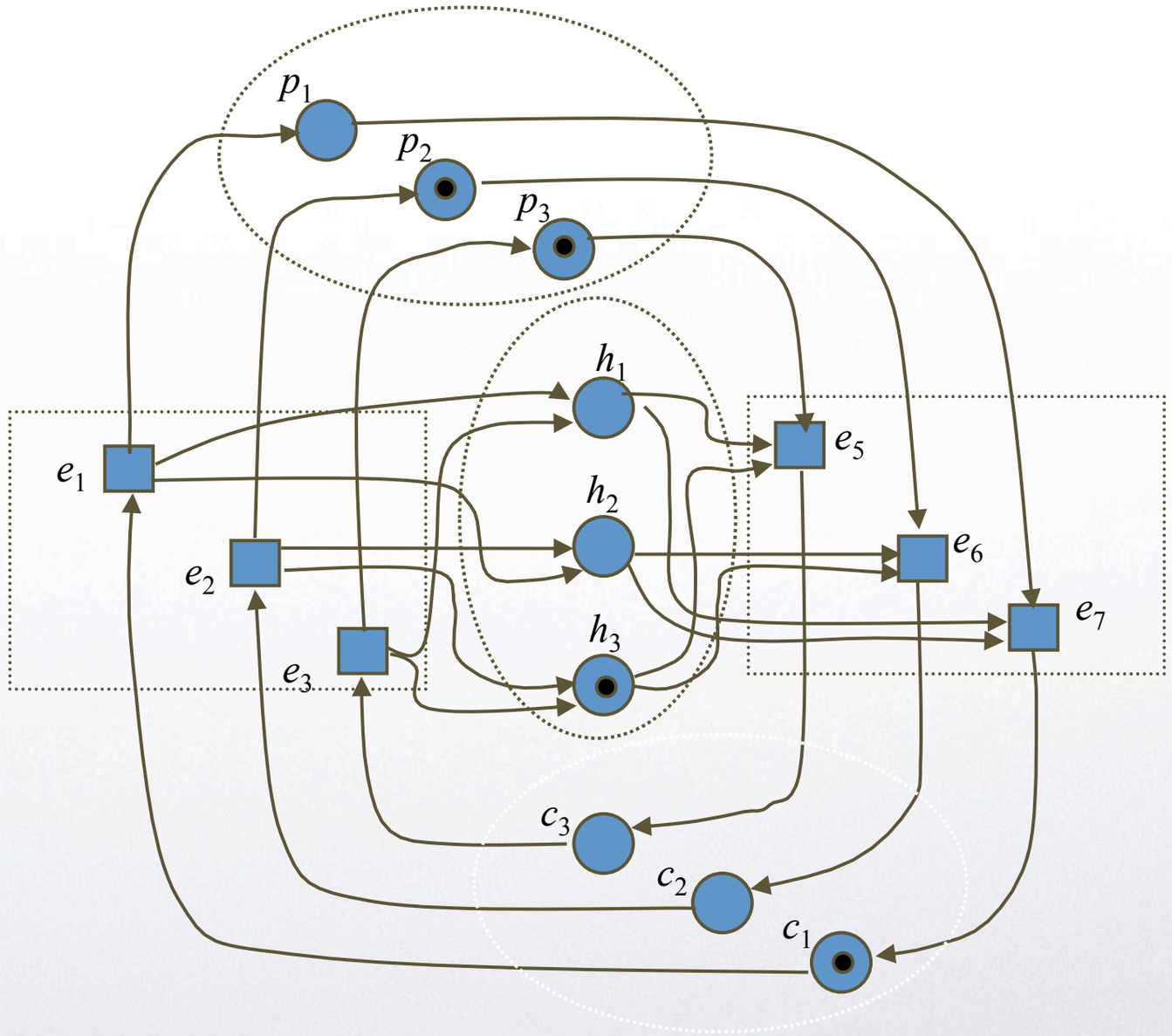
Aula 8: Redes Coloridas

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

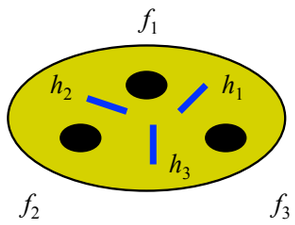
Dobramento em RdP

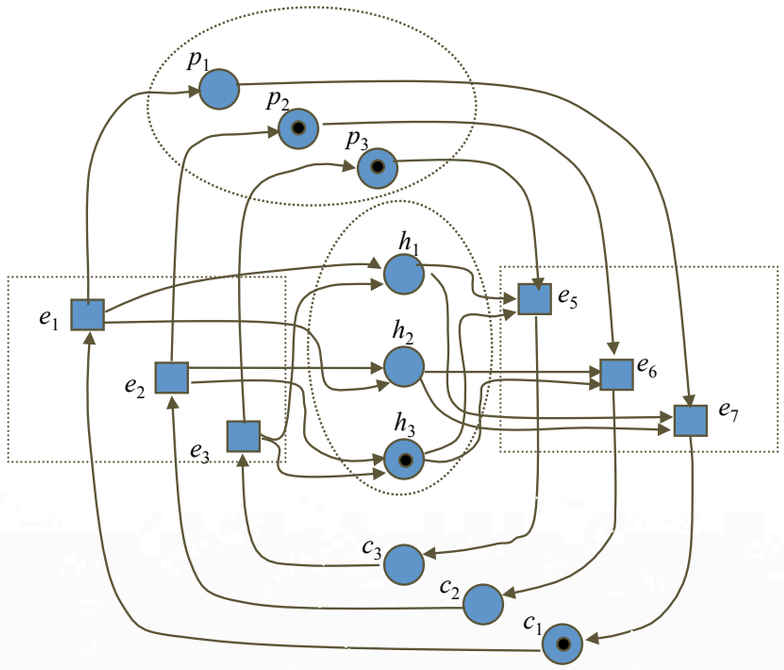
O exemplo dos filósofos

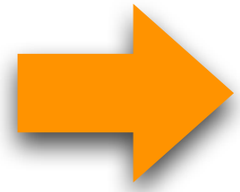


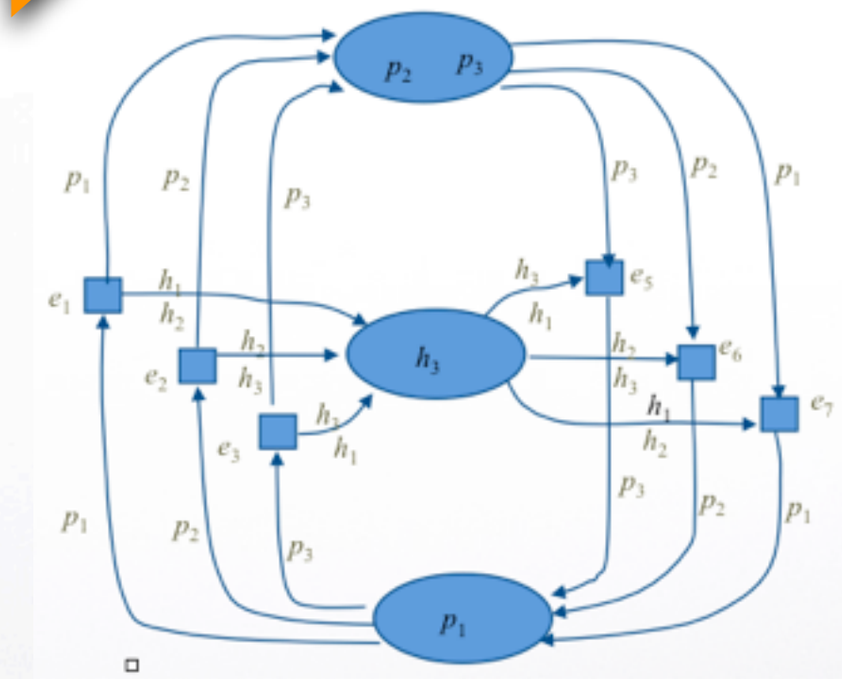


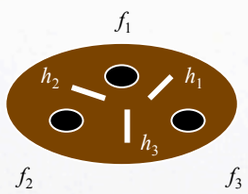
O exemplo dos filósofos





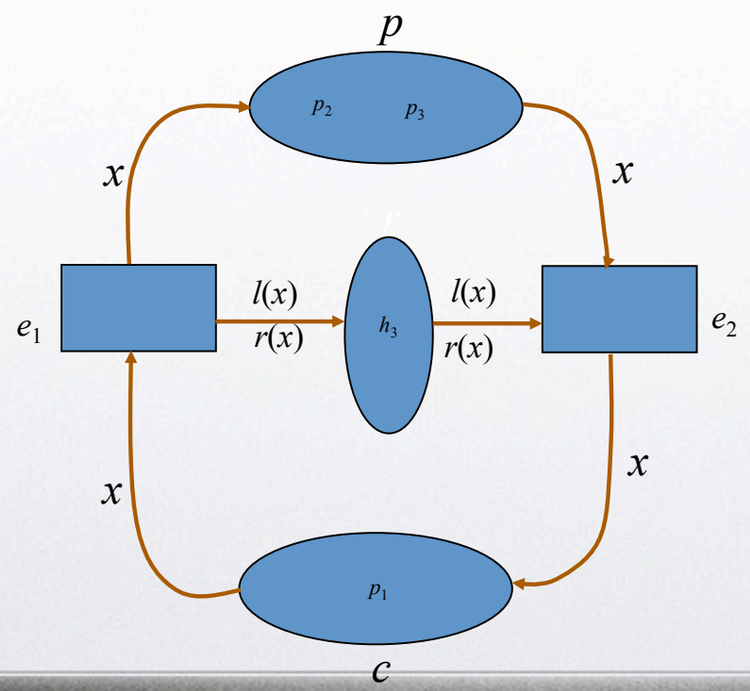






P={p_1, p_2, p_3}
H={h_1, h_2, h_3}
U=P ∪ H

l : P → H
p_i → h_i
r : P → H
p_1 → h_2
p_2 → h_3
p_3 → h_1



Redes Coloridas

1. Explorar as simetrias com o objetivo de reduzir o tamanho da rede.

Falha: o grafo é reduzido mas a informação para para as inscrições a complexidade da representação não se altera significativamente

2. Ter uma representação abstrata para a rede clássica.

Falha: verdade em princípio, mas ter que reduzir a representação para rede clássica cada vez que deseja fazer análise de propriedades pode não ser exatamente uma vantagem.

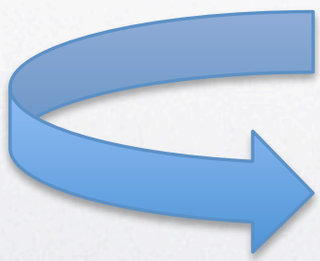
3. Aumentar o poder de expressão da representação baseada em redes.

Falha: simplesmente não é verdade.

Histórico das redes CPN

As redes coloridas surgiram nos anos 80 conjugando a representação gráfica das redes de Petri com o Standard ML, que representa tipos e cores.

No final dos anos 80 e princípio dos anos 90 surgiu o ambiente Dsign CPN proposto pelo mesmo grupo de Ahus, Dinamarca (Kurt Jensen).



A idéia é simplesmente ter um formalismo mais abstrato

Definição informal das CPNs

Kurt Jensen

Coloured Petri Nets (CP-nets or CPNs) is a graphical language for constructing models of concurrent systems and analyzing their properties. CP-nets is a discrete-event modeling language combining Petri Nets and the functional programming language CPN ML which is based on Standard ML.

Aplicações

Aplicações Típicas

- Protocolo de Comunicação
- Redes de Dados
- Algorítmos Distribuídos
- Sistemas Embarcados

Novas Aplicações

- Sistemas de workflow
- Sitemas de manufatura
- Sistemas multi-agente
- Processos de negócio
- Análise de Requisitos

Uso prático das redes CPN

Como no caso das redes clássicas o uso prático das redes CPN está associado a **Simulação** do modelo, e portanto ao estudo de cenários específicos e ao processo de evolução das marcas.

Temos portanto o mesmo problema de desenvolver métodos alternativos de análise baseados nas propriedades da rede e fugir do problema da atingibilidade.

Novos métodos → Verificação e Model checking

Características das redes CPN

- As marcas são divididas em conjuntos e separadas por tipo
- A área de declaração do sistema contém a identidade de cada variável assim como em declarações em ML
- Os arcos possuem inscrições e filtros que selecionam o tipo de marca que pode fluir por este arco.
- O comportamento dinâmico é dado pelo conjunto : grafo, inscrições e declaração.

Contexto formal

Kurt Jensen, An Introduction to the Practical Use of Coloured Petri Nets, Lect. Notes in Comp. Science 1492, 1998.

Processos especiais

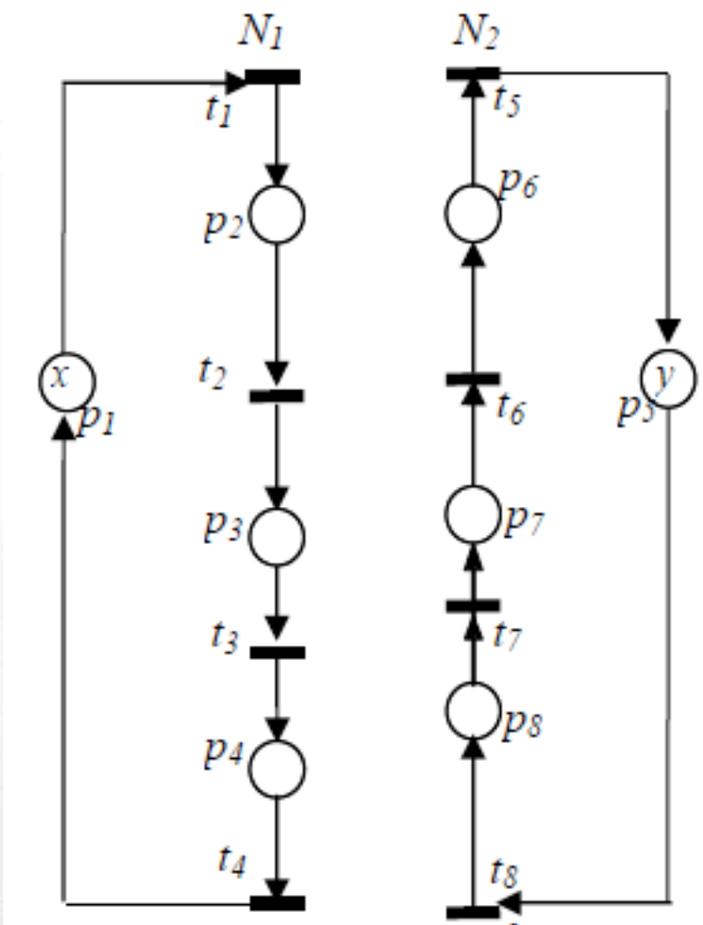


Fig. 1(b) Two S^2P.

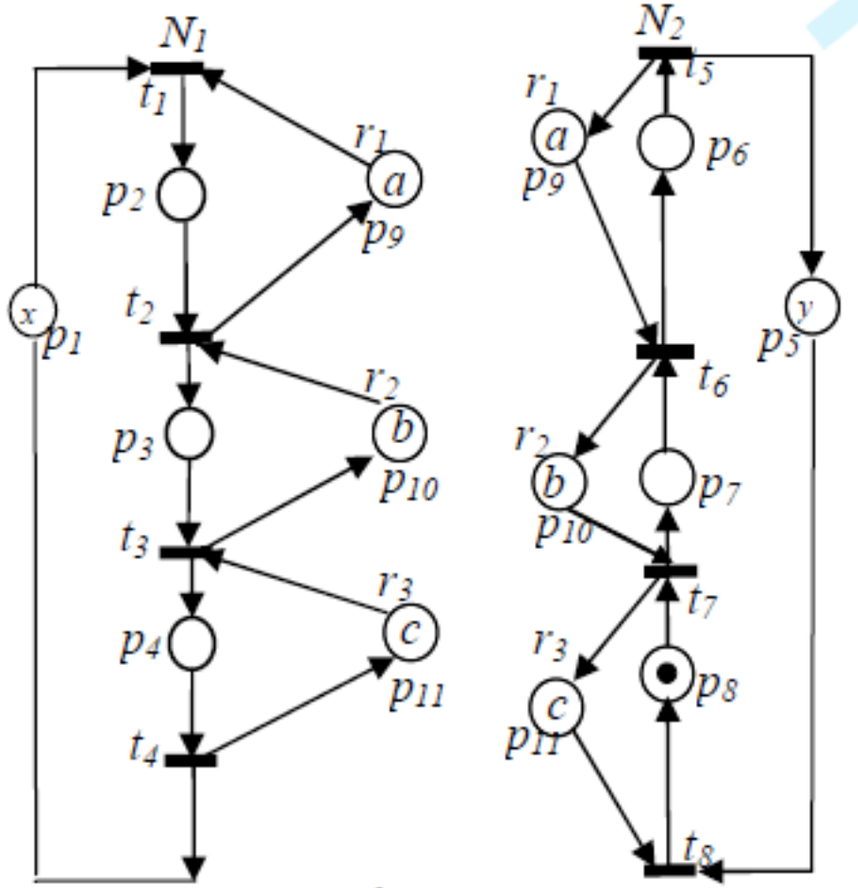


Fig. 1(c) S^2PR of N_1 and N_2

O problema do acoplamento com recursos

O sistema S2PR pode ainda ser acoplado de modo que os processos sequenciais interferem um no outro podendo causar atrasos e até deadlocks devido à falta de recursos no momento devido.

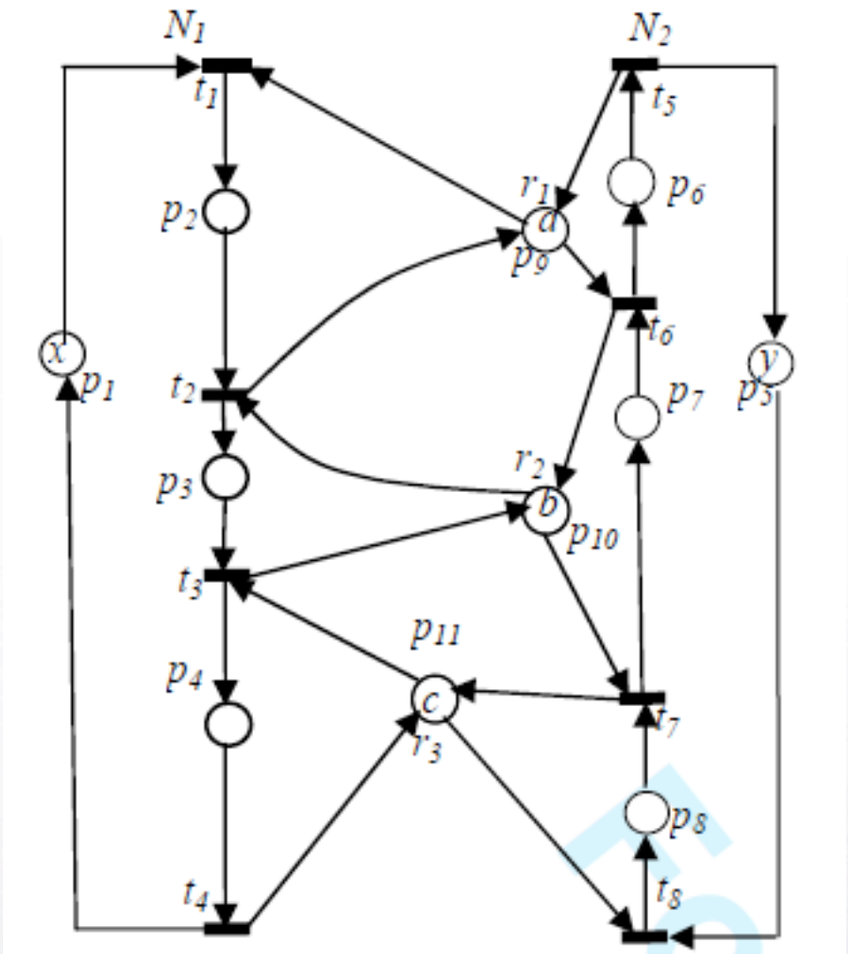


Fig. 1(a) S^{3}PR &3rd-order system ; $a=b=c=1$.

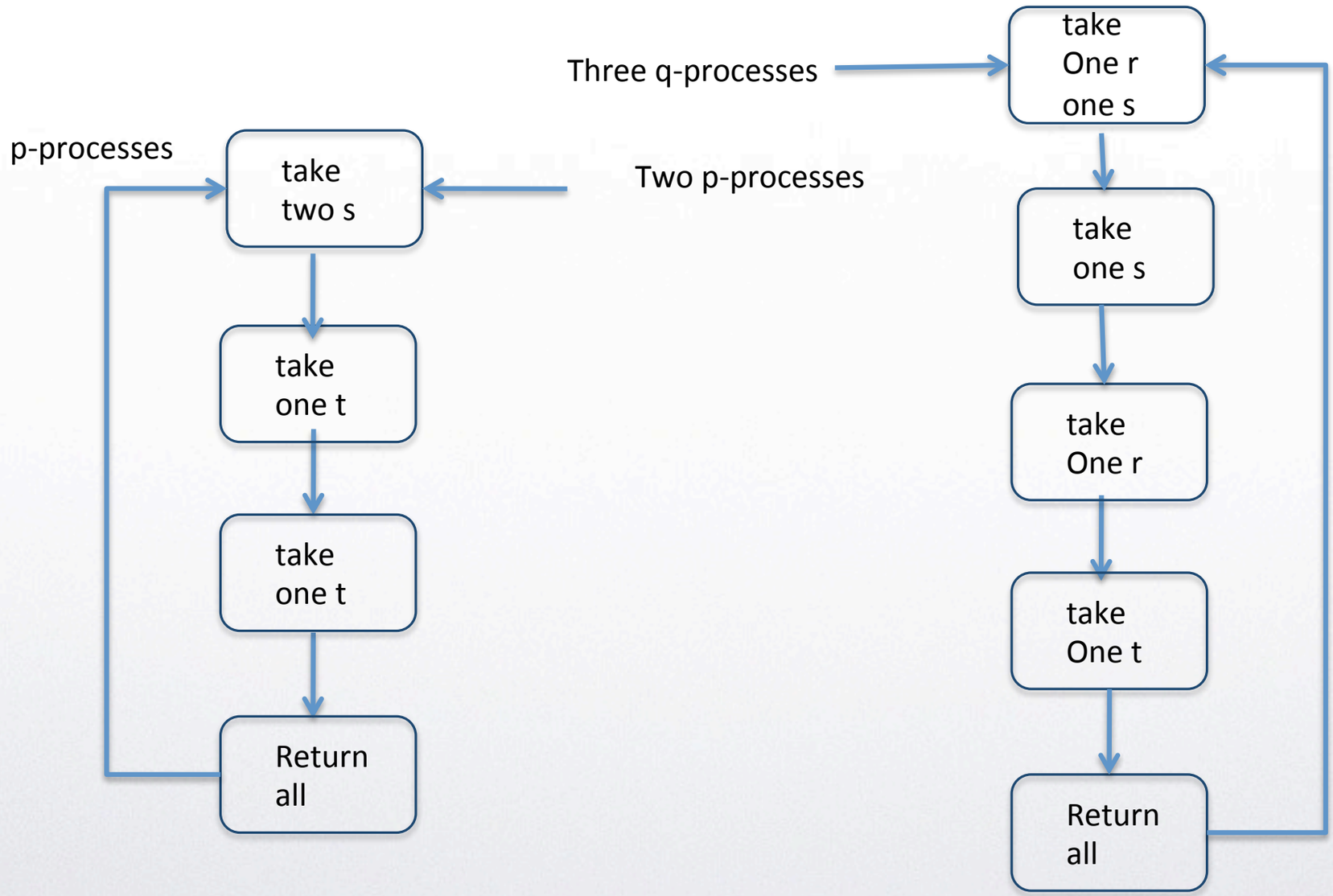
Li, Z.W. and Zhou, M.C., "Deadlock resolution in automated manufacturing systems: A novel Petri net approach," Springer, London, 2009

Introdução informal: O problema da alocação de recursos

Na indústria automotiva moderna é comum se ter vários processos ou linhas de montagem e nestes um ou mais tipos de automóvel sendo montados em pipeline. Isso traz um problema, que é ter o tipo certo de insumo ou recurso no tempo correto, para a matriz correta.

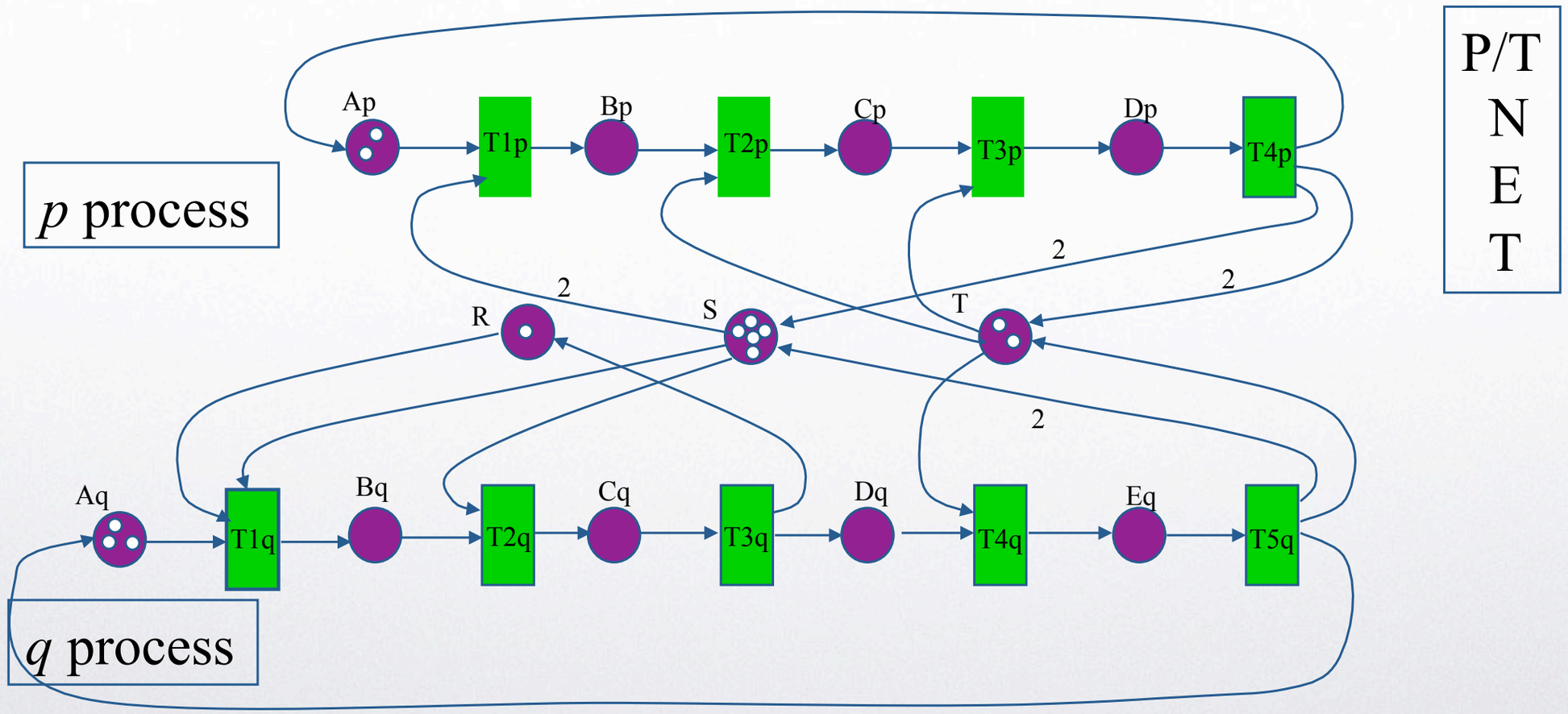
Um problema similar e igualmente importante é ter processos homogêneos mas compartilhando o mesmo centro de recursos.

Estruturação do problema



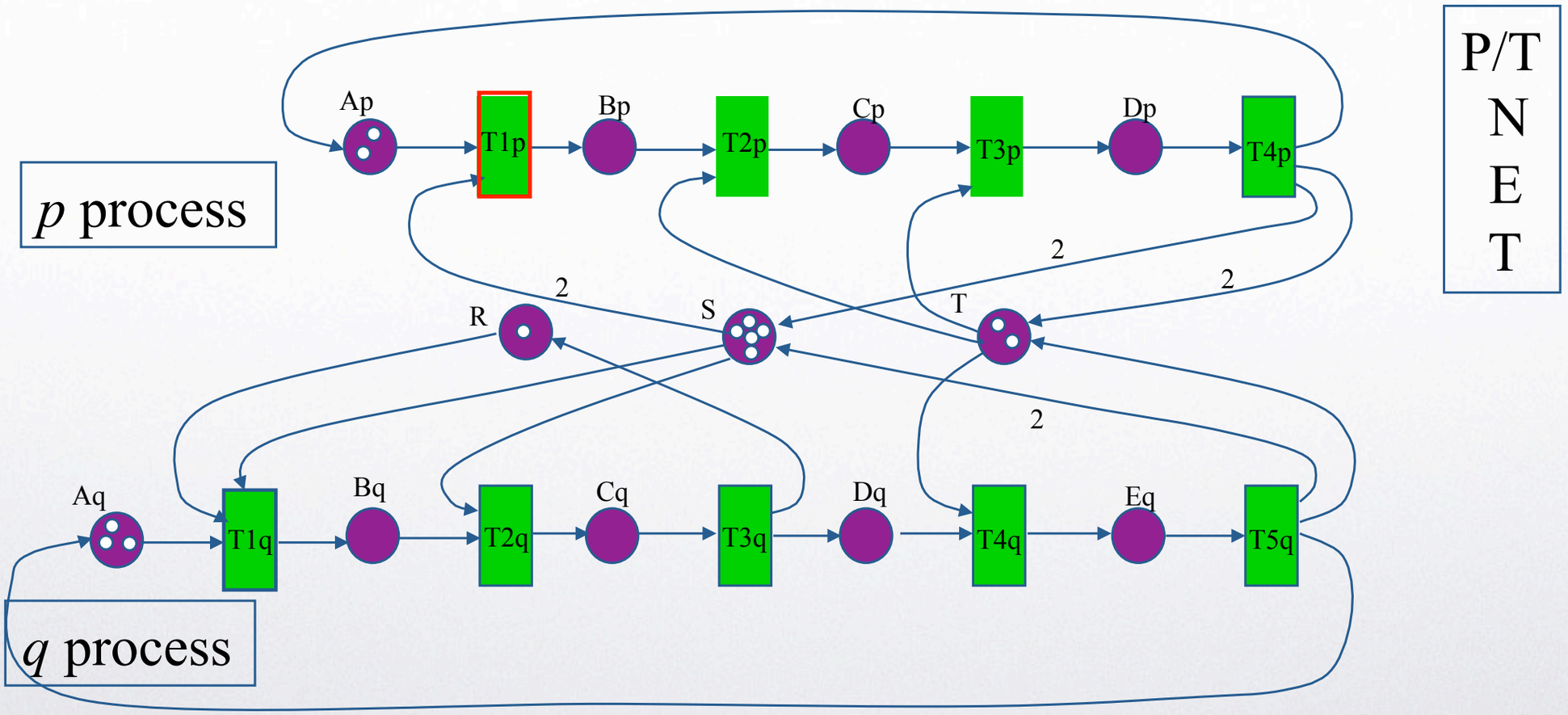
Modelagem clássica

Seja o seguinte sistema de alocação de recursos, caracterizado por dois processos concorrentes,



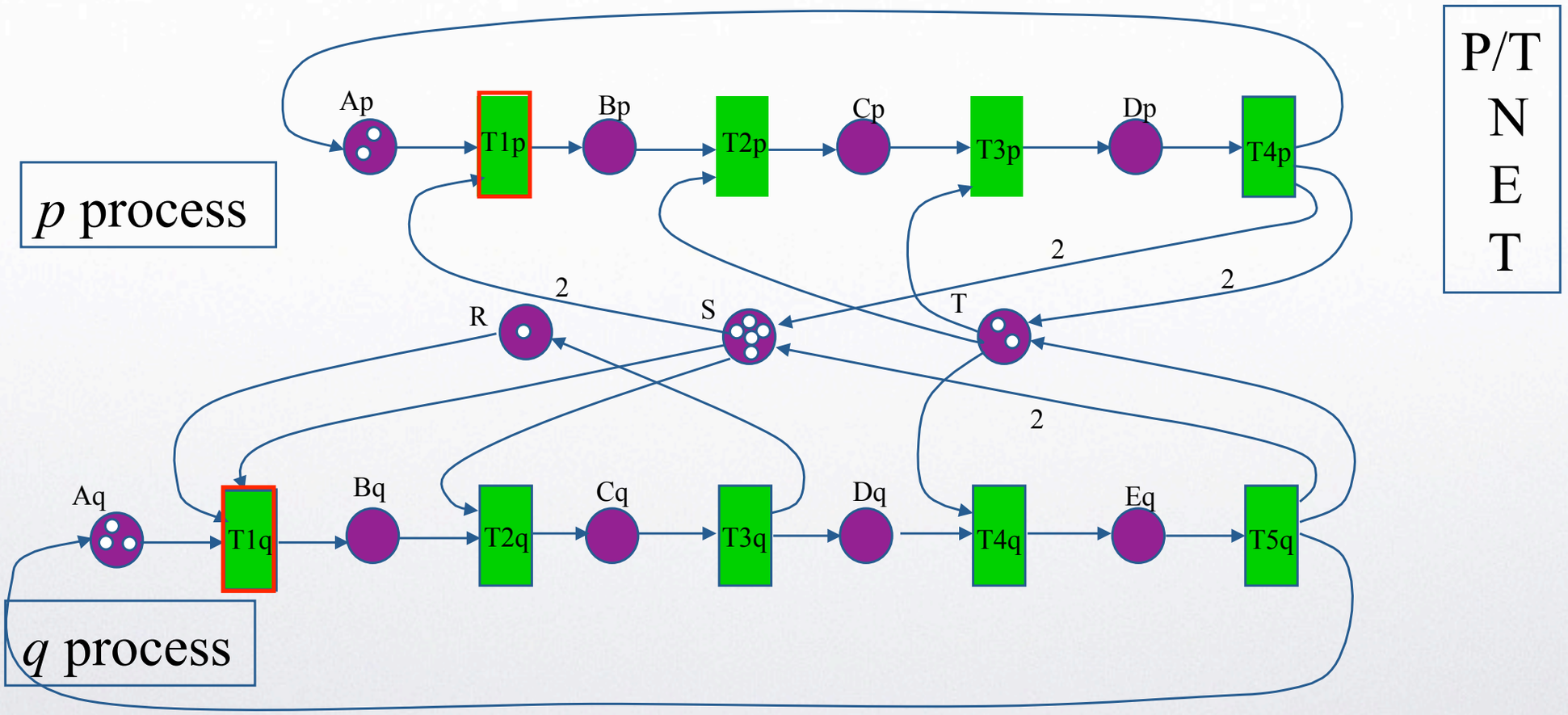
Modelagem clássica

Seja o seguinte sistema de alocação de recursos, caracterizado por dois processos concorrentes,

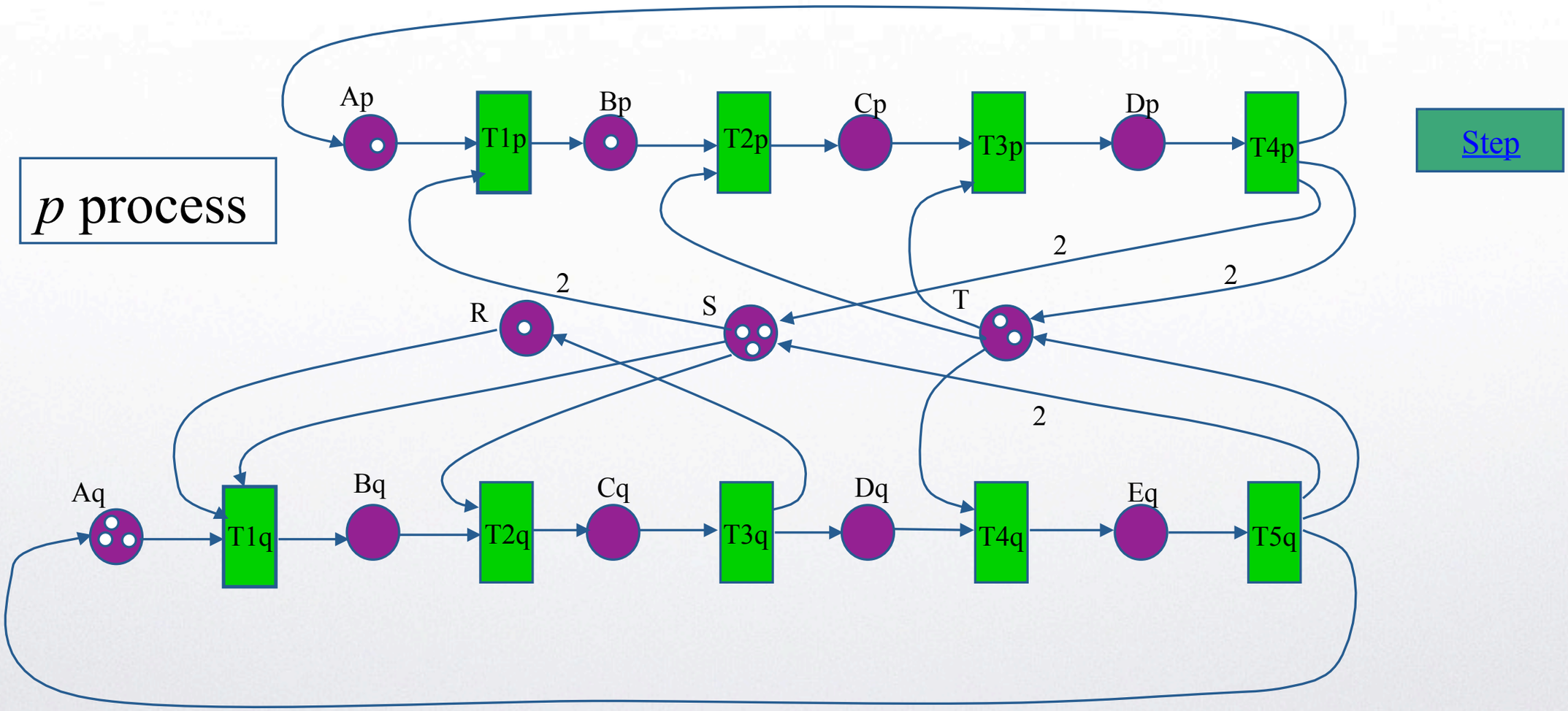


Modelagem clássica

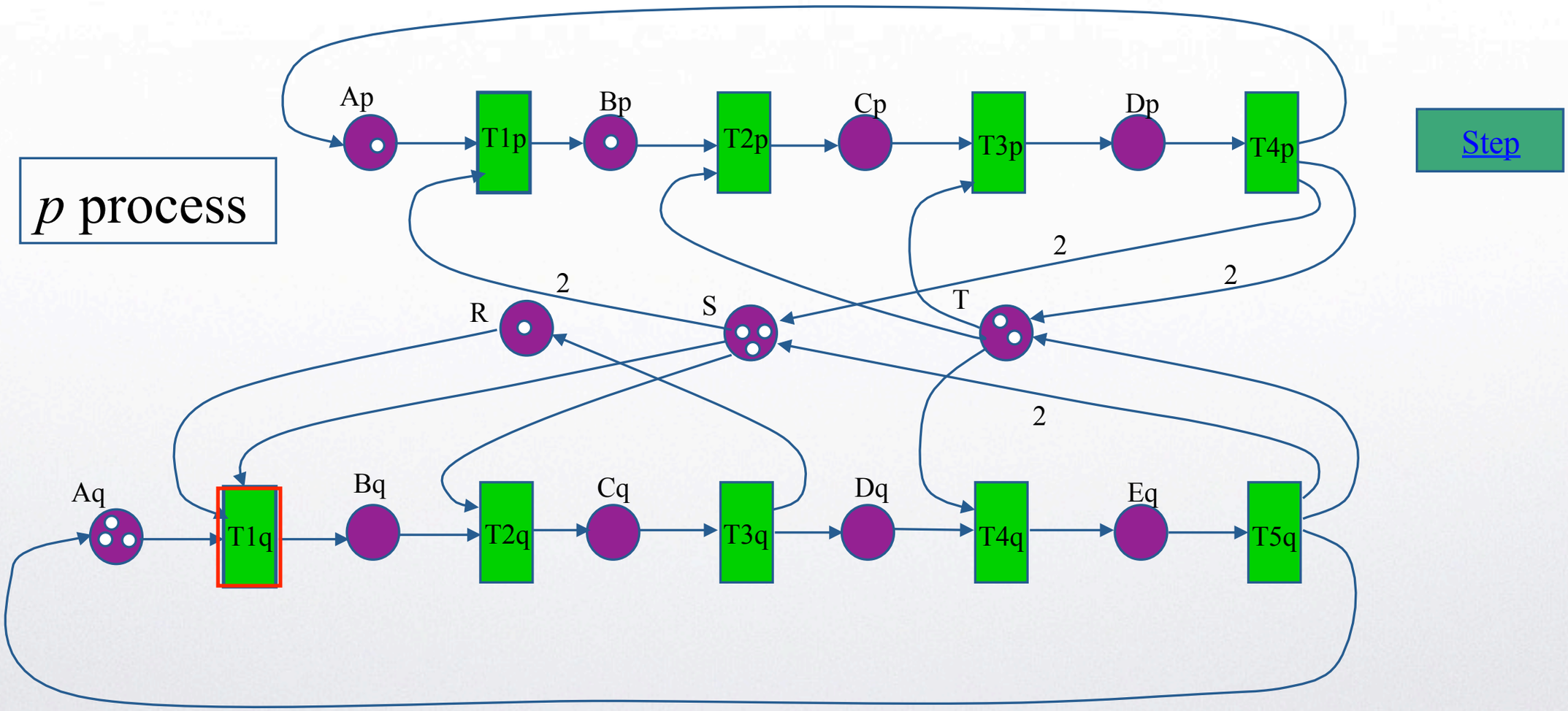
Seja o seguinte sistema de alocação de recursos, caracterizado por dois processos concorrentes,



Passos e multisets

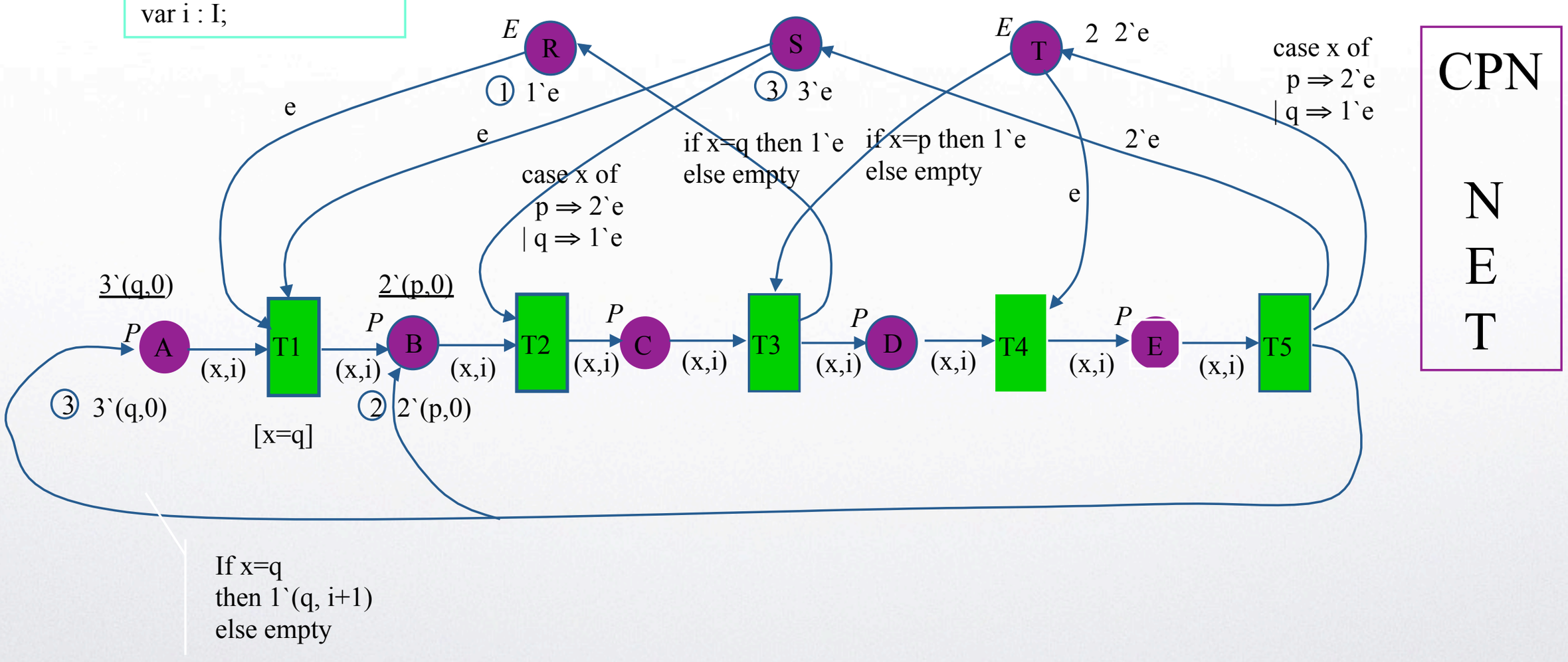


Passos e multisets



Para sintetizar uma rede uma rede colorida devemos explorar a simetria entre os processos, isto é, produzir um único processo que representa tanto o processo p como o processo q. Neste caso as marcas passam a ser distinguíveis para representar os dois processos.

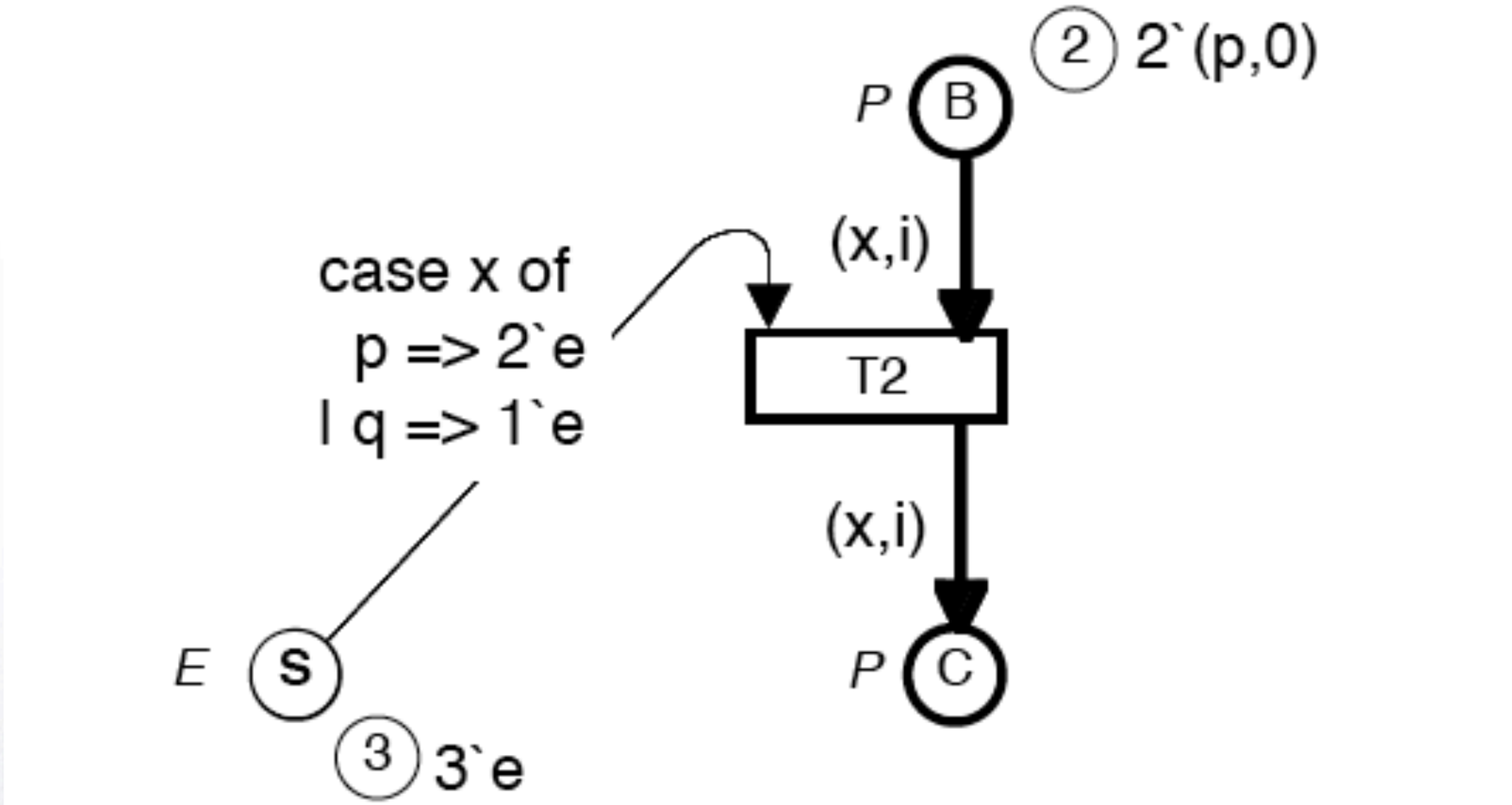
```
color U=with p | q;
color I = int;
color P = product U*I;
color E= with e;
var x : U;
var i : I;
```



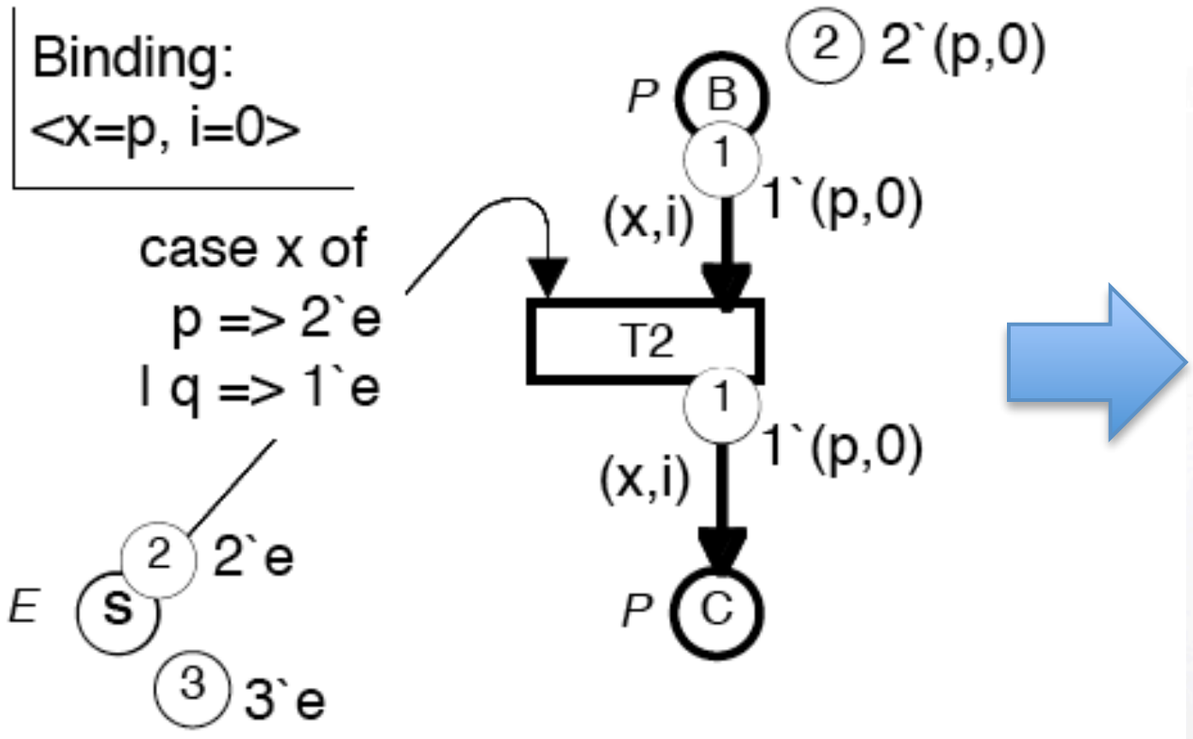
CPN NET

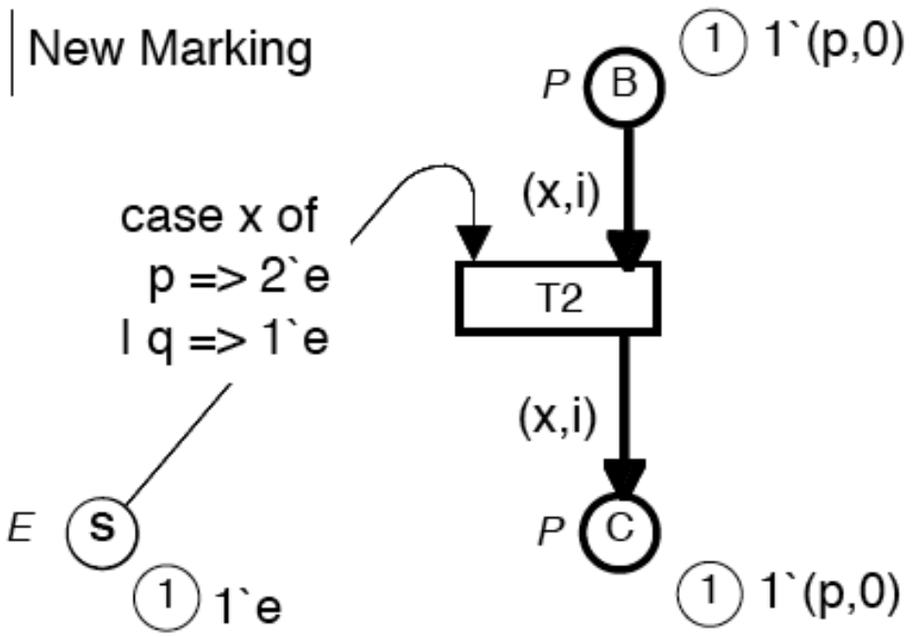
If x=q
then 1`(q, i+1)
else empty

Biding

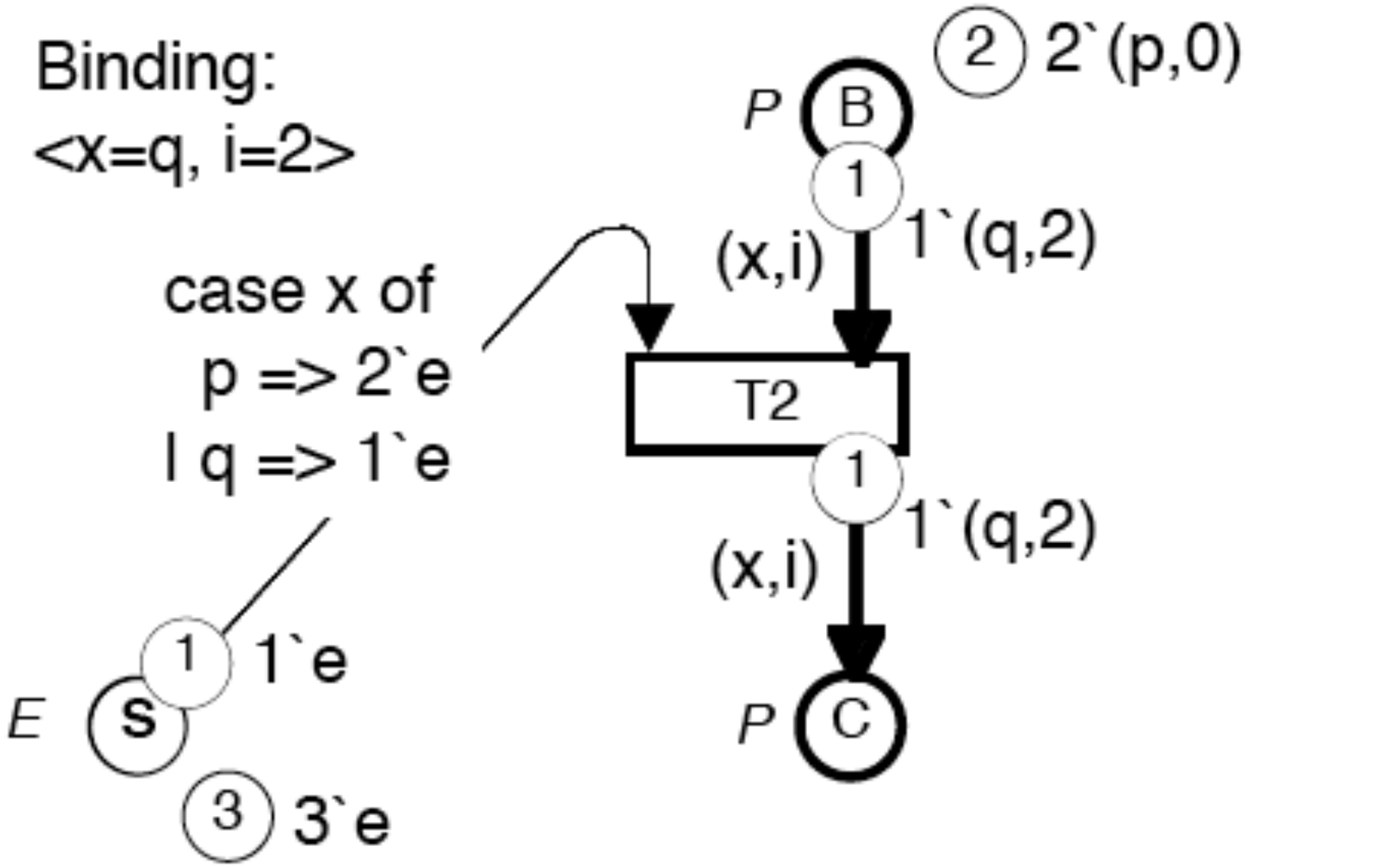


Disparo na rede

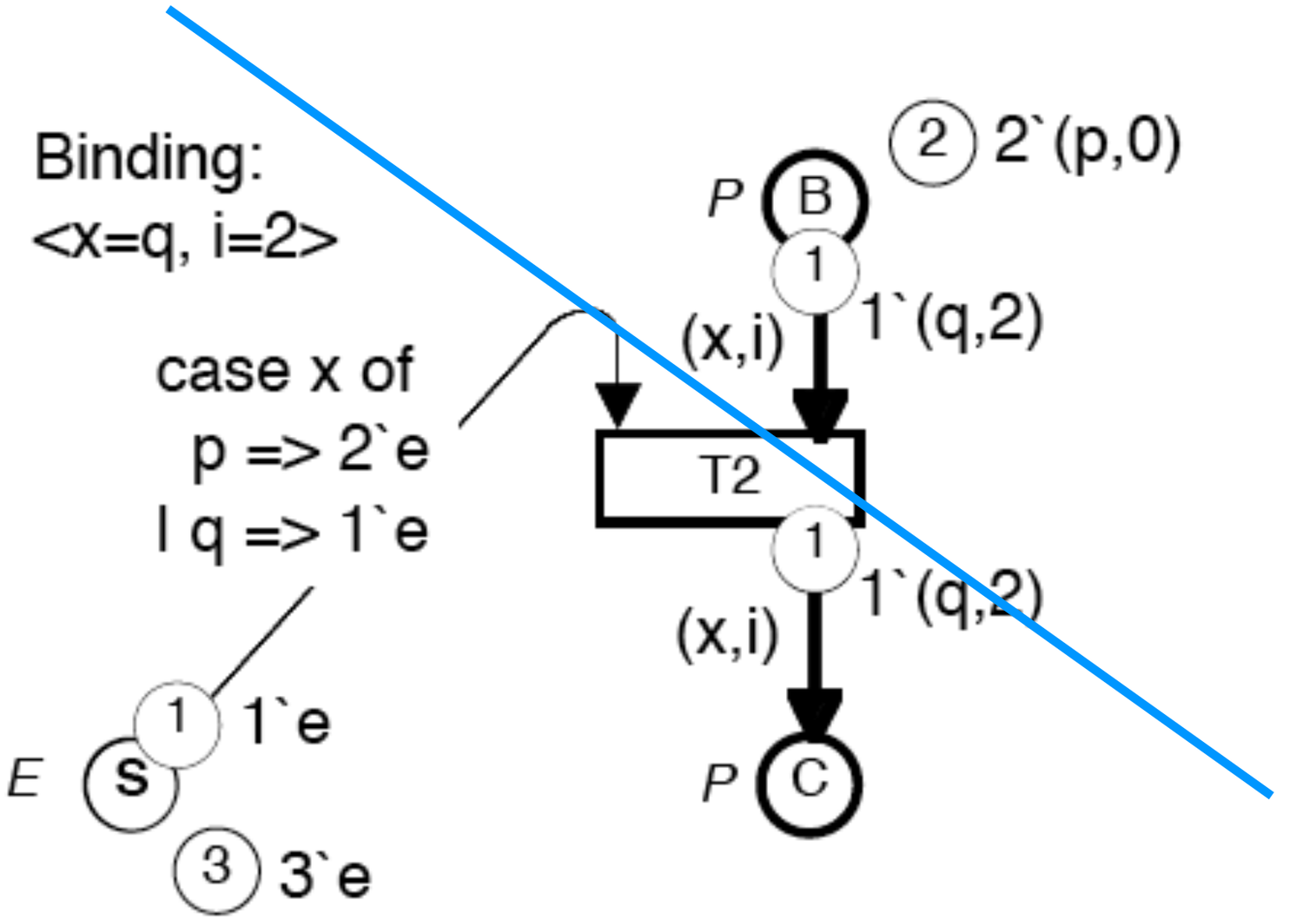




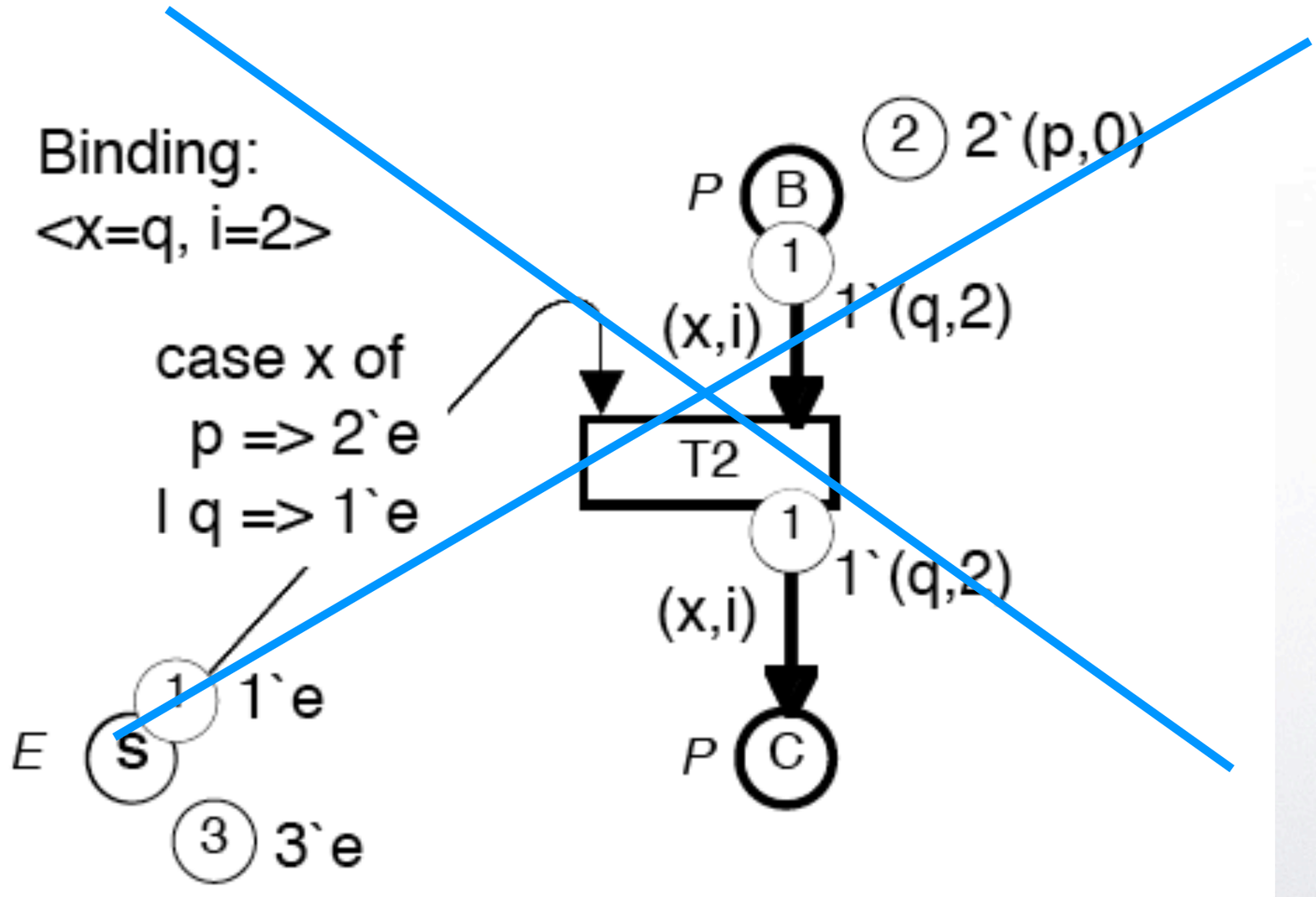
Strong type


Binding:
<x=q, i=2>
2
2`(p,0)
P
B
1
(x,i)
1`(q,2)
case x of
p => 2`e
| q => 1`e
T2
1
1`(q,2)
(x,i)
1
1`e
E
S
P
C
3
3`e

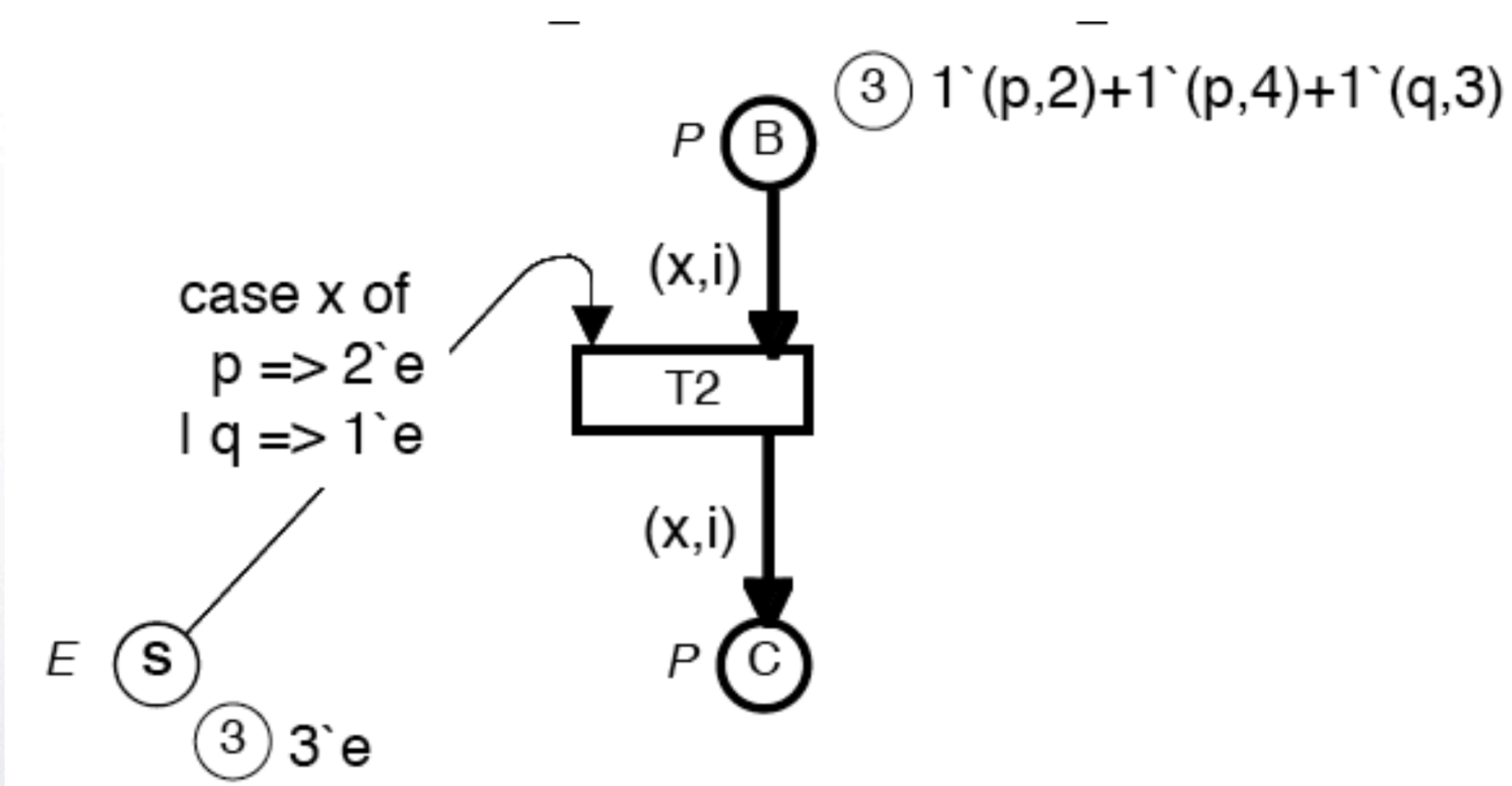
Strong type


Binding:
<x=q, i=2>
case x of
p => 2`e
| q => 1`e
E
S
1
1`e
3
3`e
P
B
2
2`(p,0)
1
1`(q,2)
(x,i)
T2
1
1`(q,2)
(x,i)
P
C

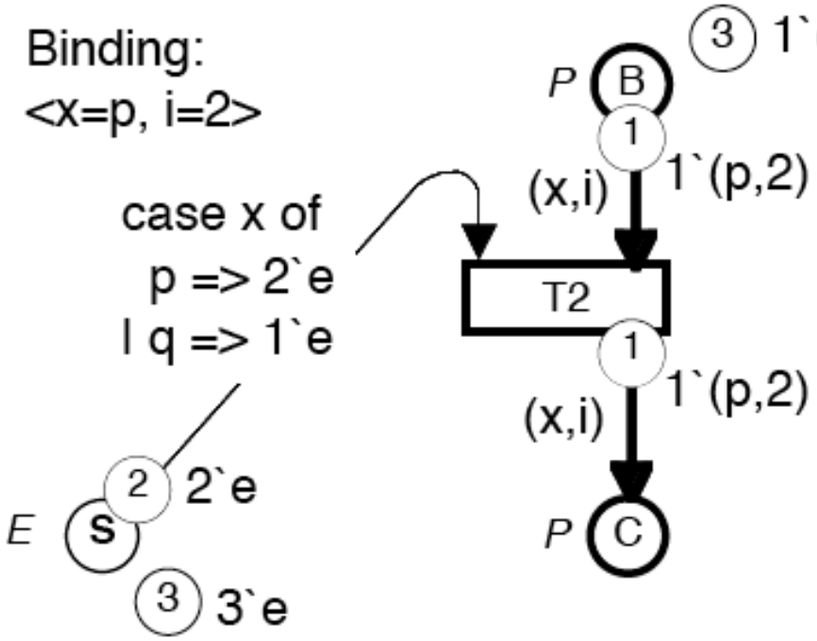
Strong type

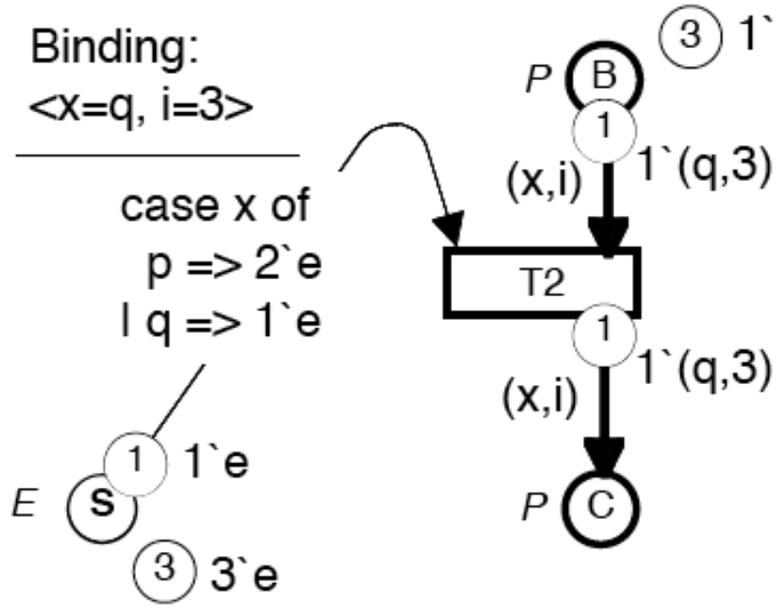

Binding:
<x=q, i=2>
case x of
p => 2`e
| q => 1`e
P
B
2
2`(p,0)
1
1`(q,2)
(x,i)
T2
1
1`(q,2)
(x,i)
P
C
E
S
1
1`e
3
3`e

Outro exemplo

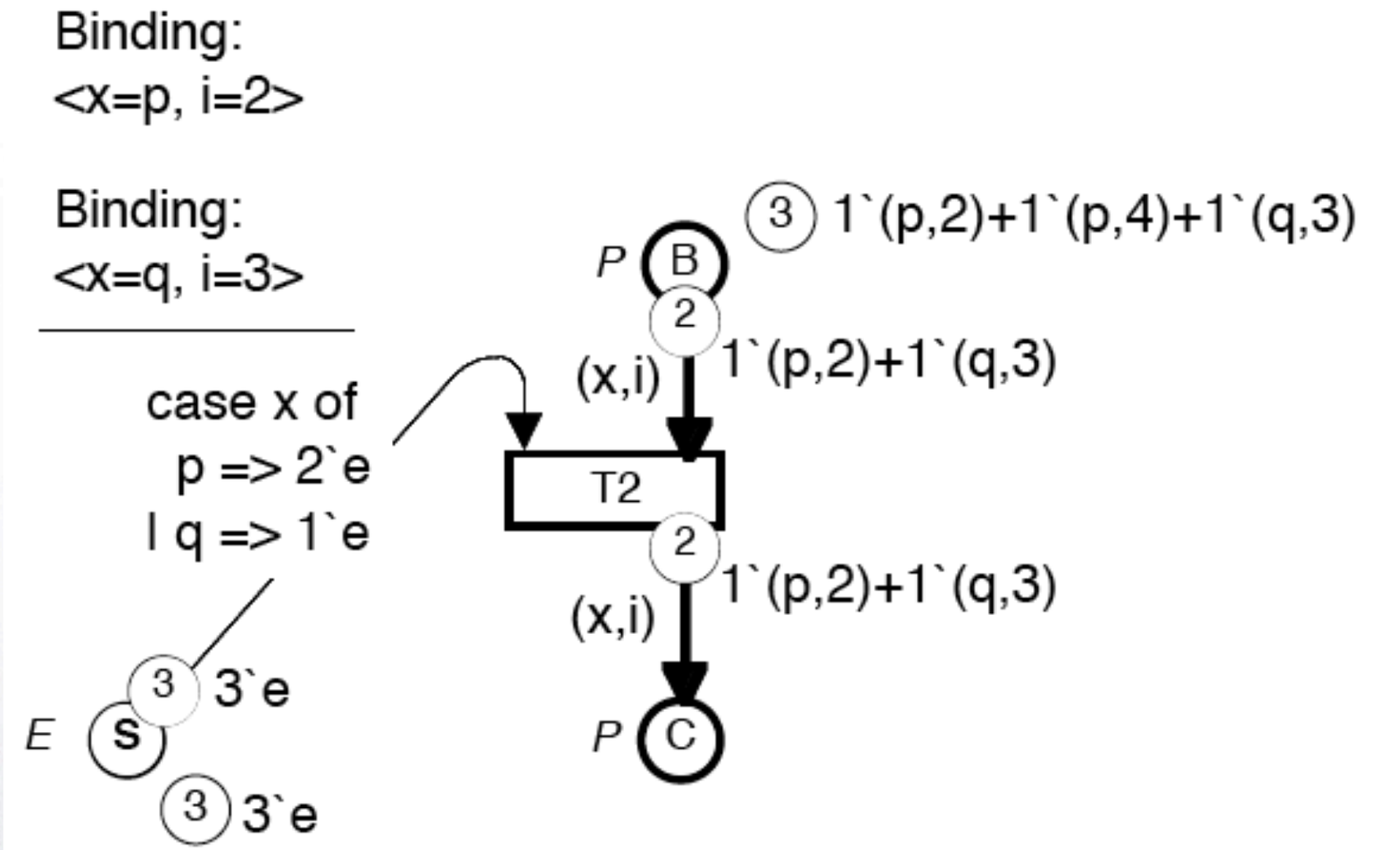


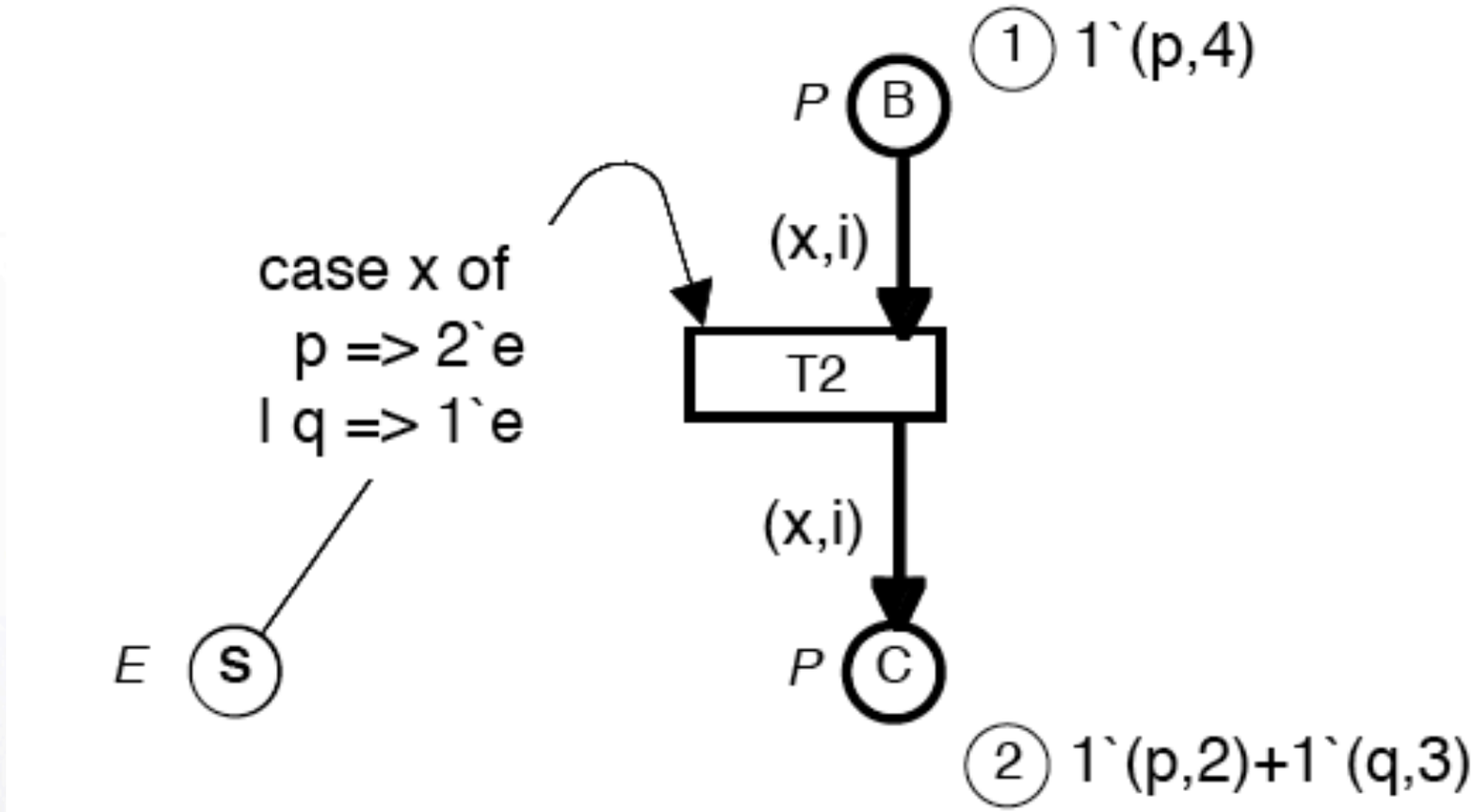
Uma vez respeitado o tipo qualquer possibilidade pode ocorrer.





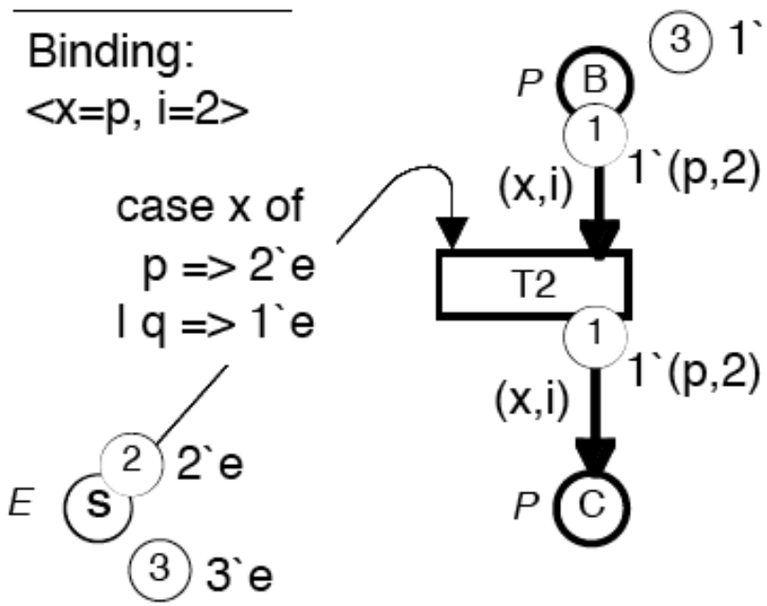
Concorrência

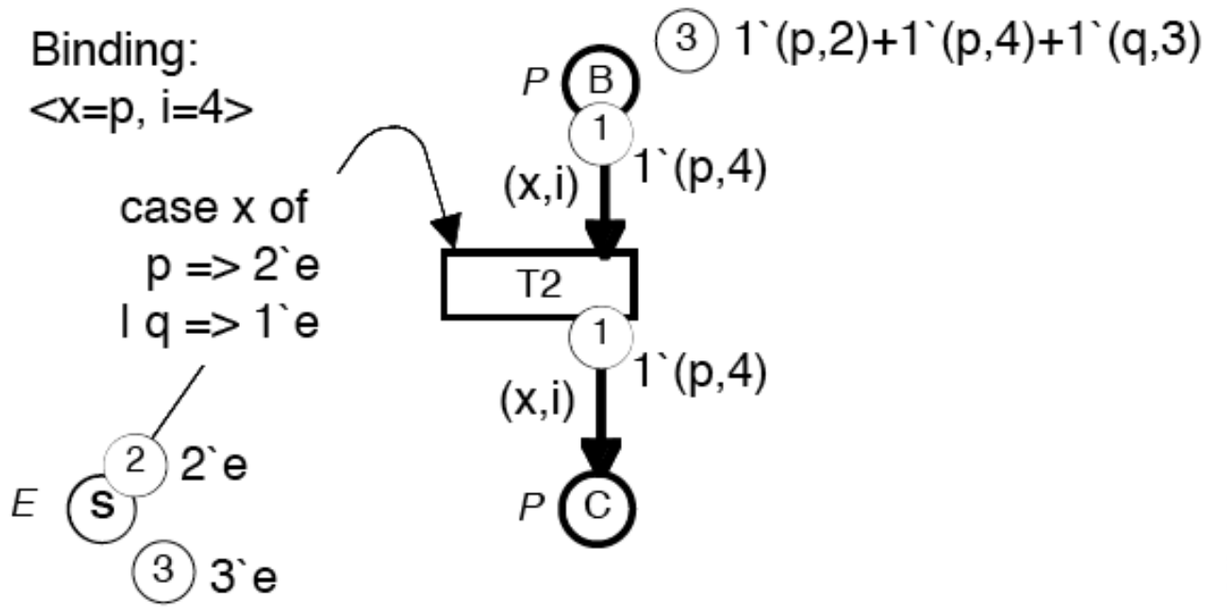



1 1`(p,4)
P B
(x,i)
case x of
p => 2`e
| q => 1`e
T2
(x,i)
E S
P C
2 1`(p,2)+1`(q,3)

Gerenciando conflito

A situação de conflito é similar, só que agora levando em conta os tipos





CPN Elements

Declarations:

- *Types*, *functions*, *operations* and *variables*.

Each *place* has the following inscriptions:

- *Name* (for identification).
- *Colour set* (specifying the type of tokens which may reside on the place).
- *Initial marking* (multi-set of token colours).

Each *transition* has the following inscriptions:

- *Name* (for identification).
- *Guard* (boolean expression containing some of the variables).

Each *arc* has the following inscriptions:

- *Arc expression* (containing some of the variables). When the arc expression is evaluated it yields a multi-set of token colours.

Enabling Condition

A *binding* assigns a *colour* (i.e., a value) to each *variable* of a transition.

A *binding element* is a pair (t,b) where t is a *transition* while b is a *binding* for the variables of t. Example: (T2,<x=p, i=2>).

Biding

A binding element is *enabled* if and only if:

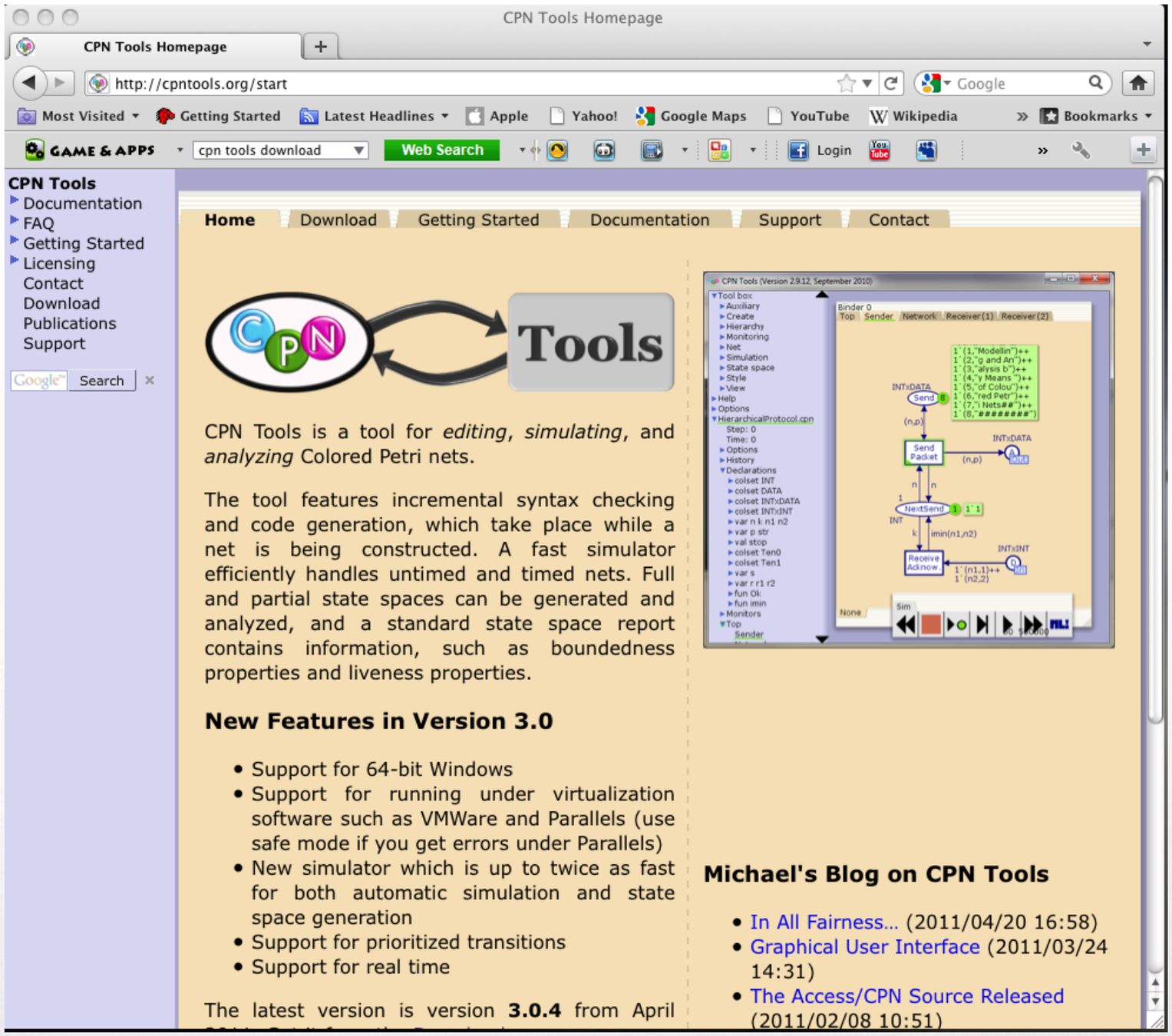
- There are *enough tokens* (of the correct colours on each input-place).
- The *guard* evaluates to true.

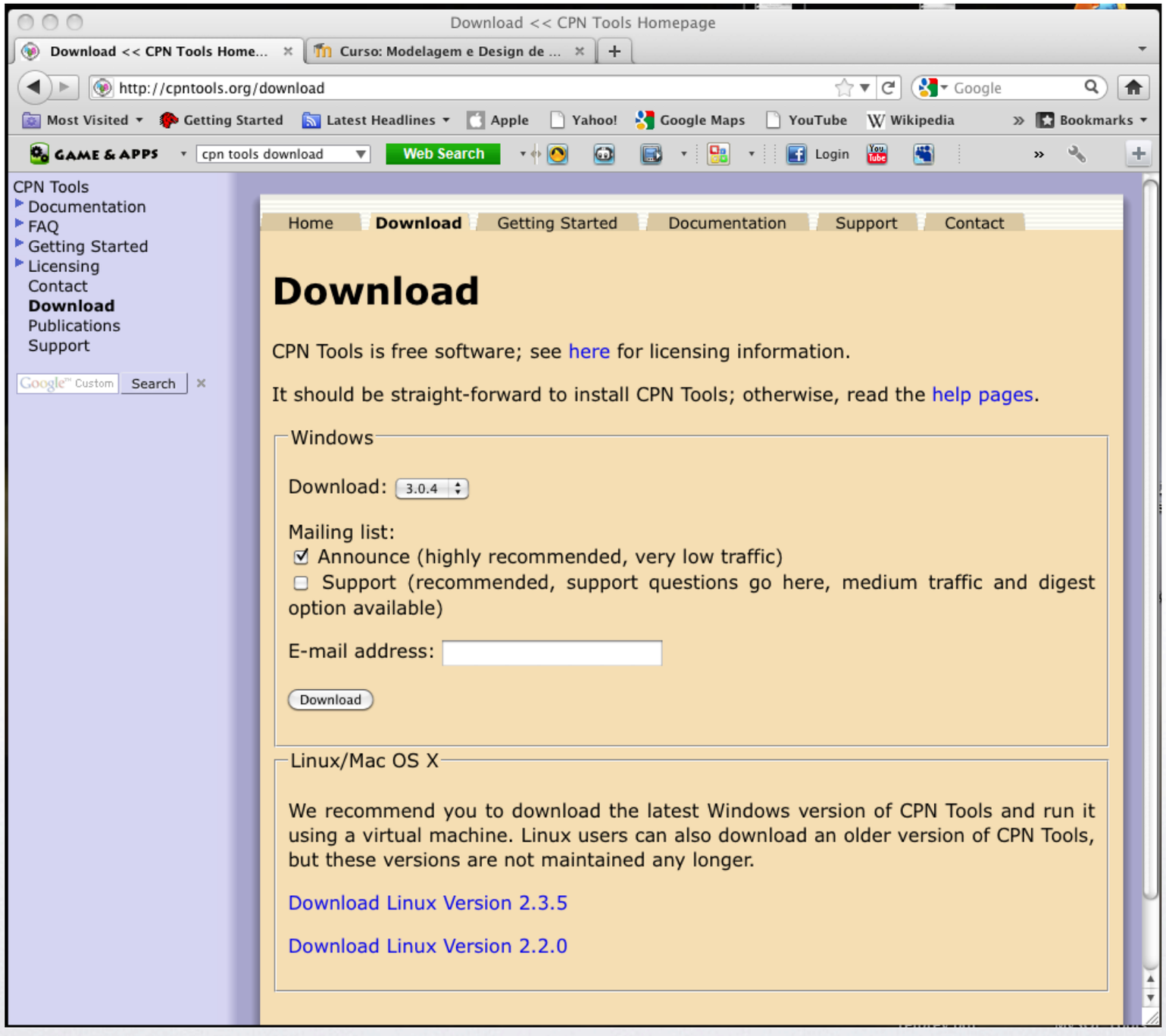
When a binding element is enabled it may *occur:*

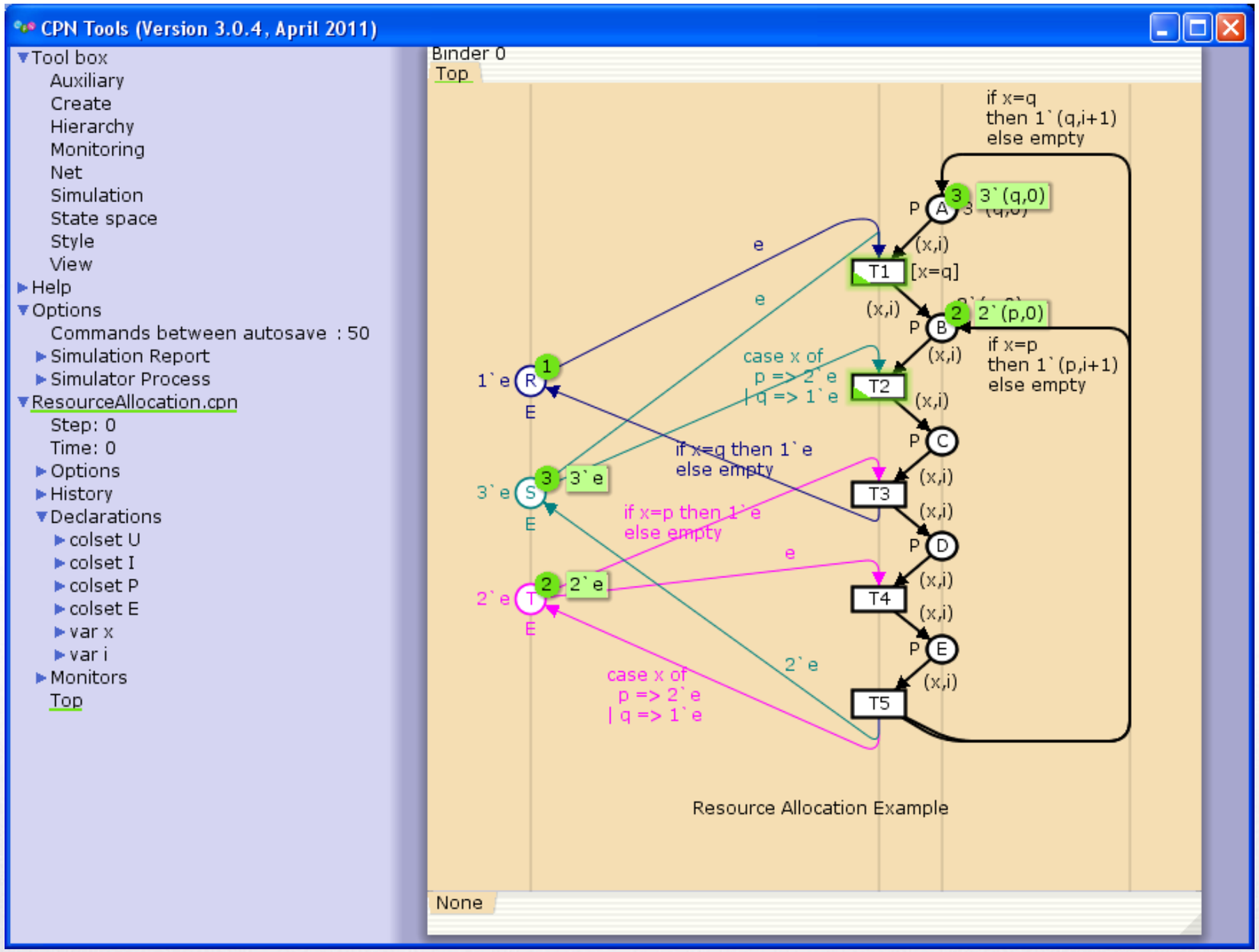
- A multi-set of tokens is *removed* from each input-place.
- A multi-set of tokens is *added* to each output-place.

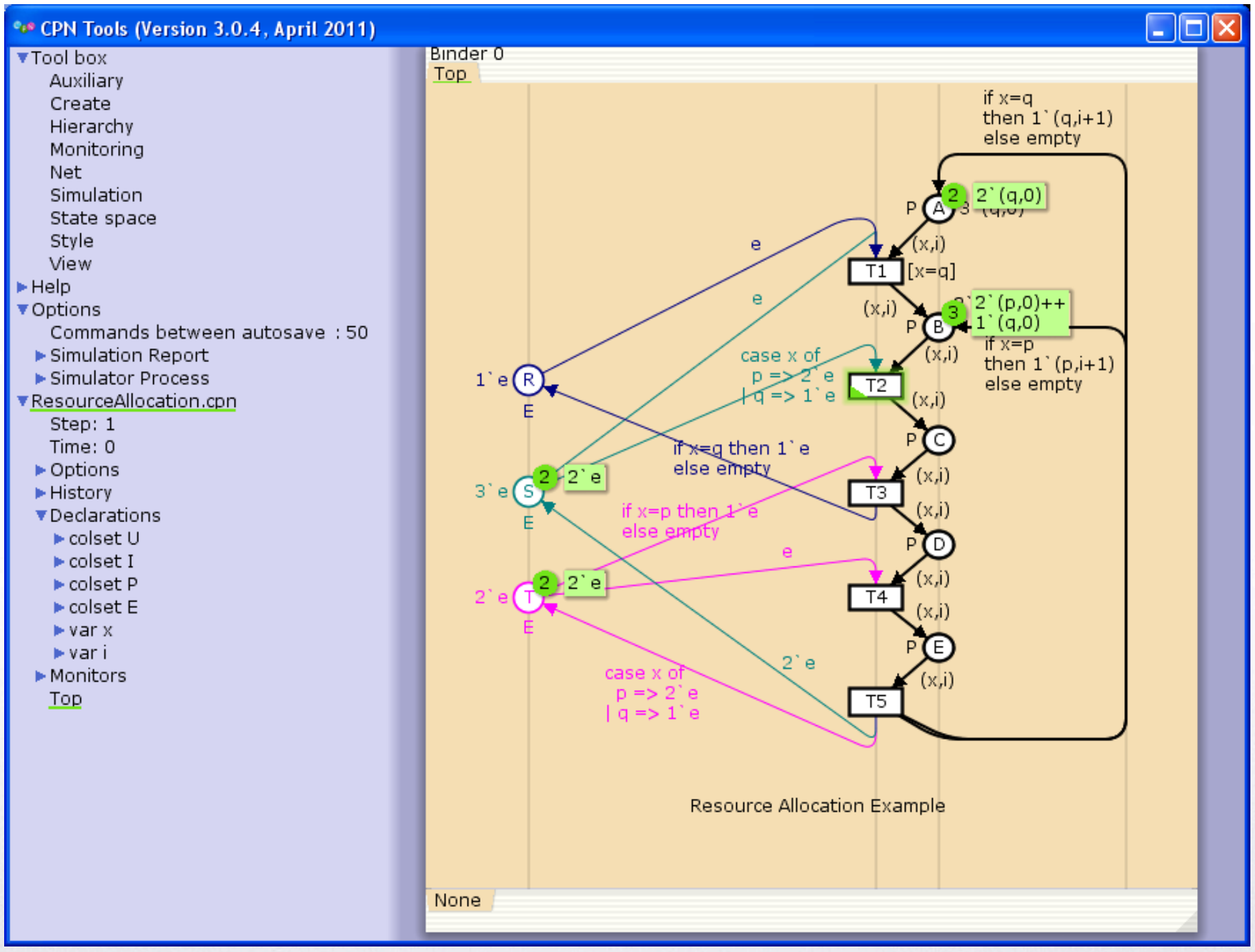
A binding element may occur *concurrently* to other binding elements – iff there are so many tokens that each binding element can get its "own share".

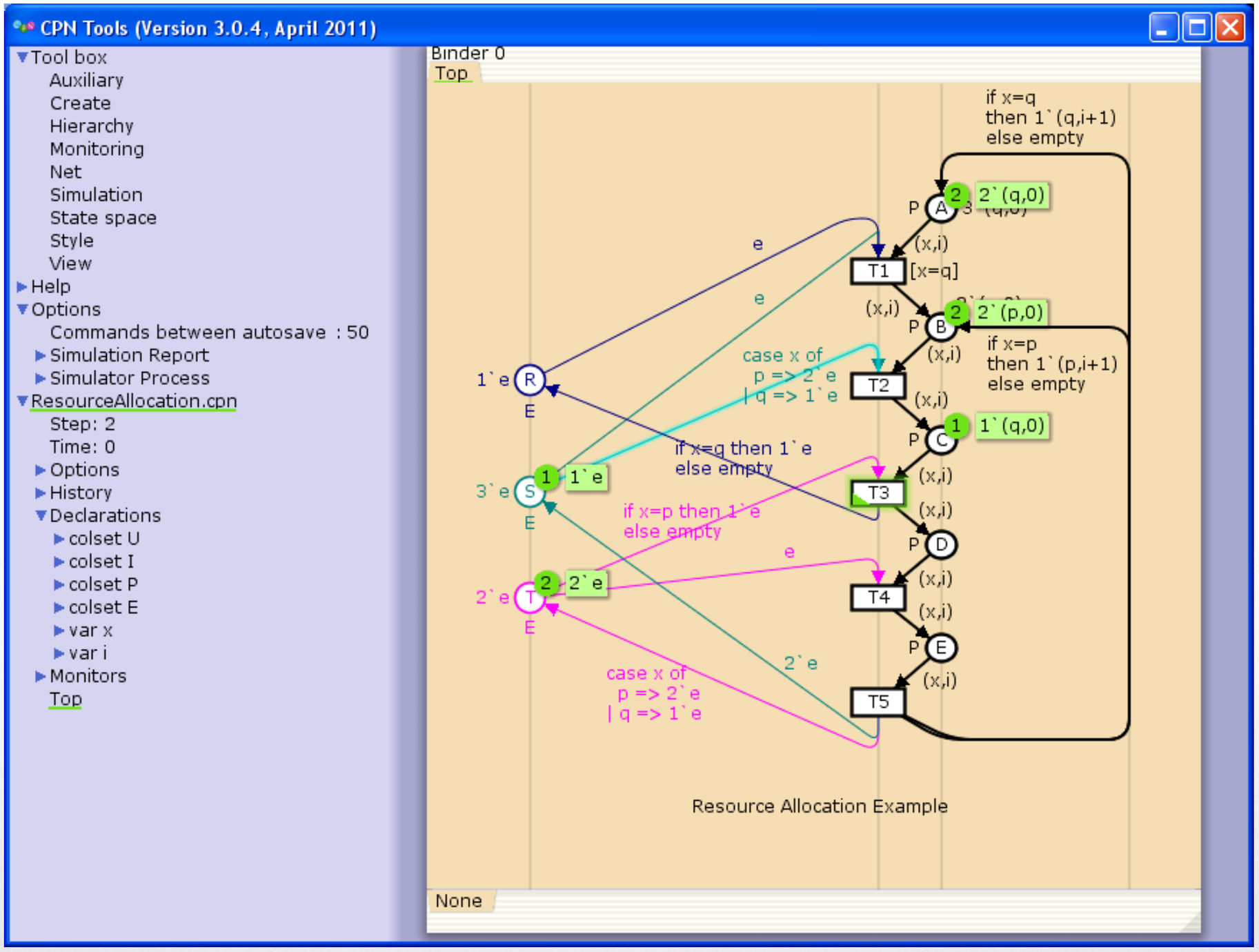
CPN Tools (cpntools.org)

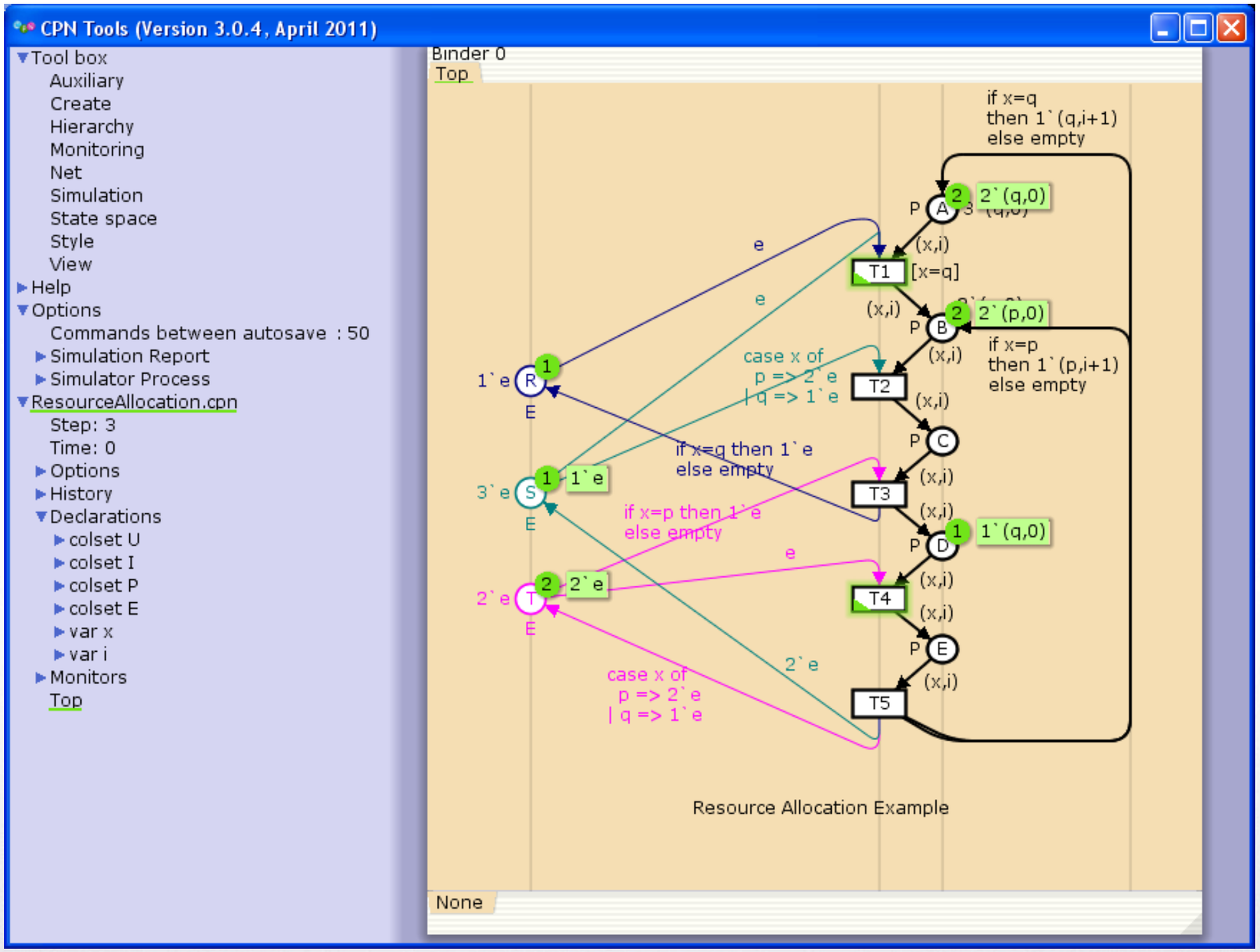


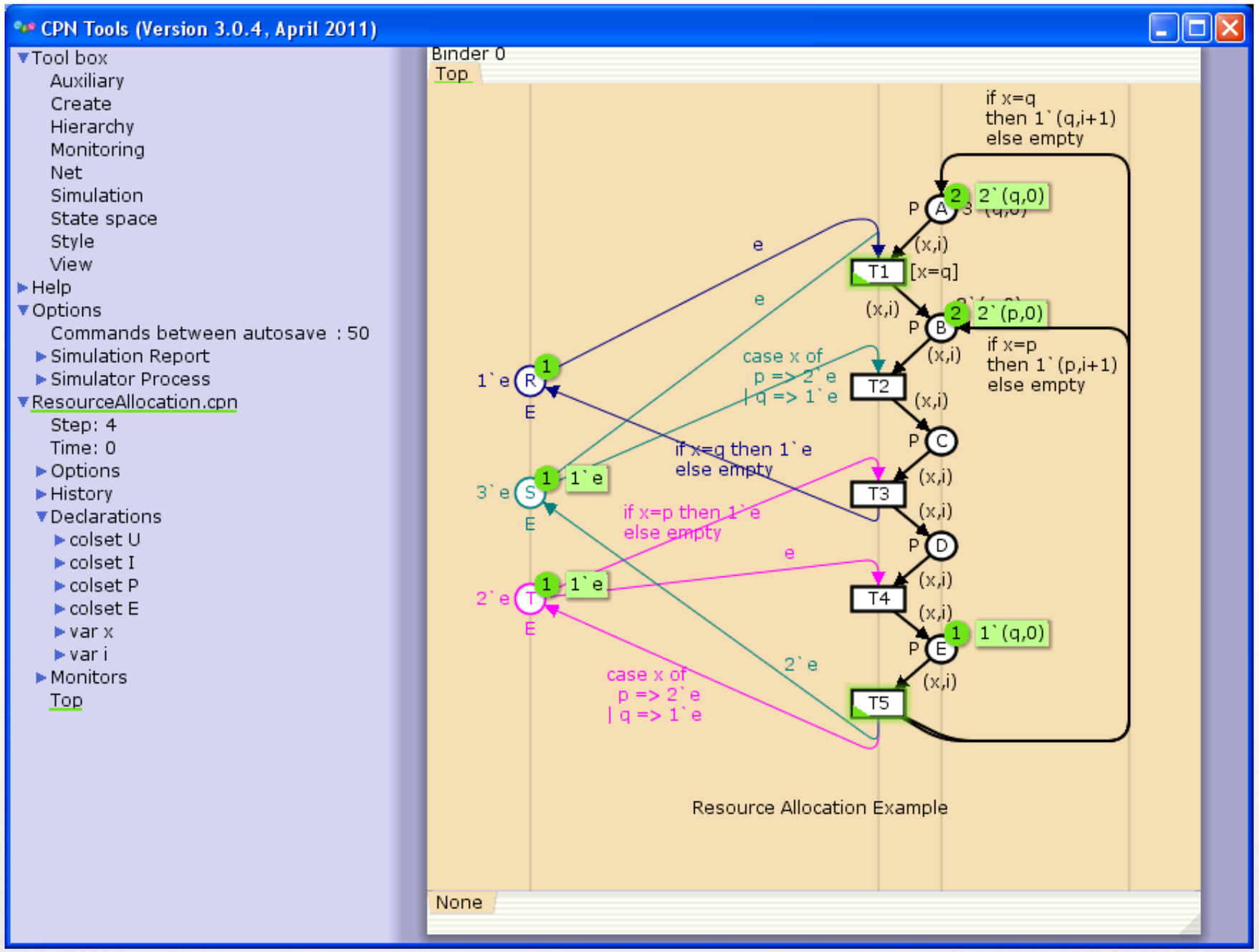

Download << CPN Tools Homepage
Download << CPN Tools Home...
Curso: Modelagem e Design de ...
http://cpntools.org/download
Most Visited
Getting Started
Latest Headlines
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
Bookmarks
GAME & APPS
cpn tools download
Web Search
Login
CPN Tools
Documentation
FAQ
Getting Started
Licensing
Contact
Download
Publications
Support
Search
Home
Download
Getting Started
Documentation
Support
Contact
Download
CPN Tools is free software; see here for licensing information.
It should be straight-forward to install CPN Tools; otherwise, read the help pages.
Windows
Download: 3.0.4
Mailing list:
Announce (highly recommended, very low traffic)
Support (recommended, support questions go here, medium traffic and digest option available)
E-mail address:
Download
Linux/Mac OS X
We recommend you to download the latest Windows version of CPN Tools and run it using a virtual machine. Linux users can also download an older version of CPN Tools, but these versions are not maintained any longer.
Download Linux Version 2.3.5
Download Linux Version 2.2.0

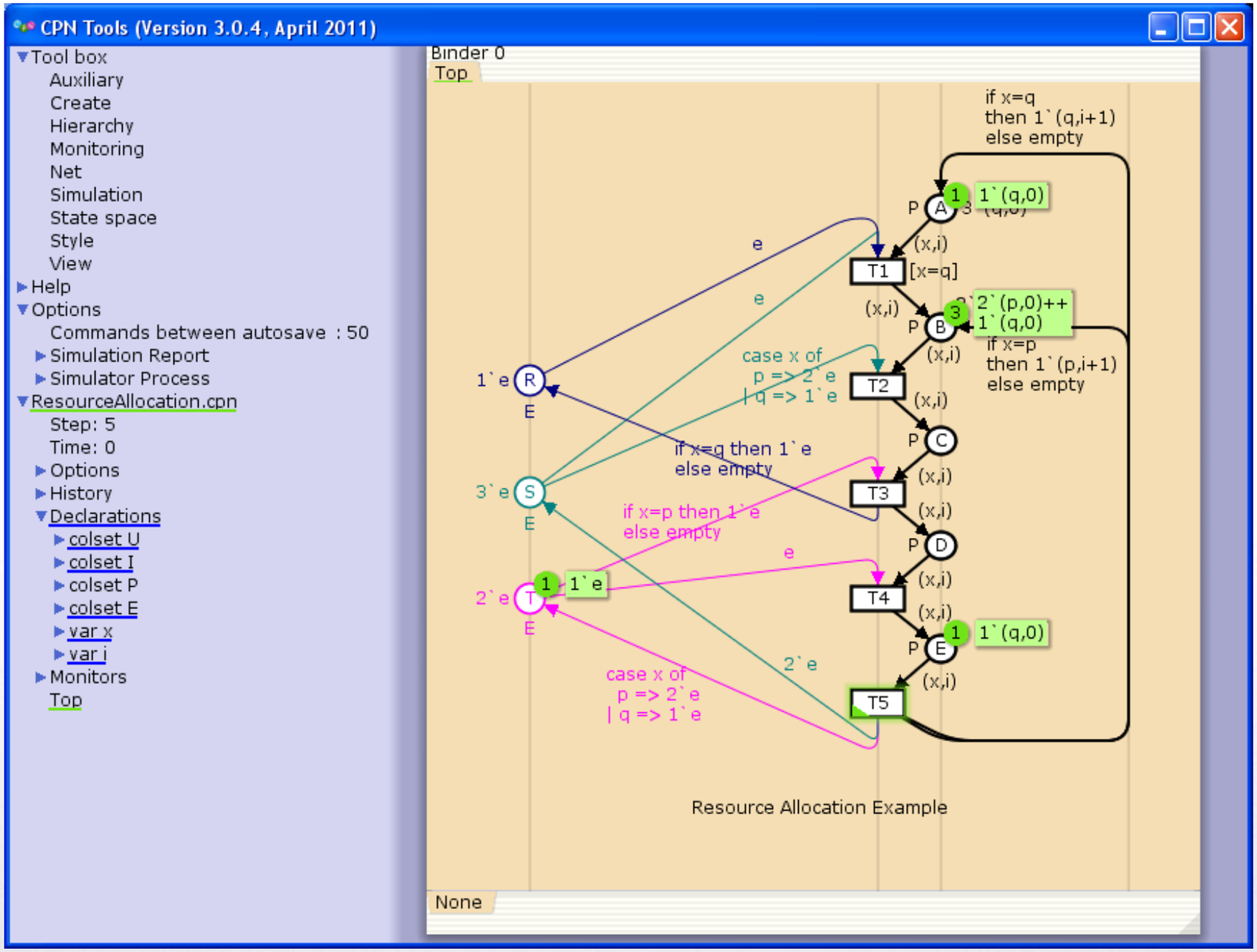

CPN Tools (Version 3.0.4, April 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
Commands between autosave : 50
Simulation Report
Simulator Process
ResourceAllocation.cpn
Step: 0
Time: 0
Options
History
Declarations
colset U
colset I
colset P
colset E
var x
var i
Monitors
Top
Binder 0
Top
if x=q
then 1`(q,i+1)
else empty
3`(q,0)
P
A
(x,i)
T1
[x=q]
2`(p,0)
B
if x=p
then 1`(p,i+1)
else empty
case x of
p => 2`e
| q => 1`e
T2
C
T3
D
T4
E
T5
1`e
R
E
3`e
S
if x=q then 1`e
else empty
if x=p then 1`e
else empty
2`e
T
2`e
case x of
p => 2`e
| q => 1`e
Resource Allocation Example
None

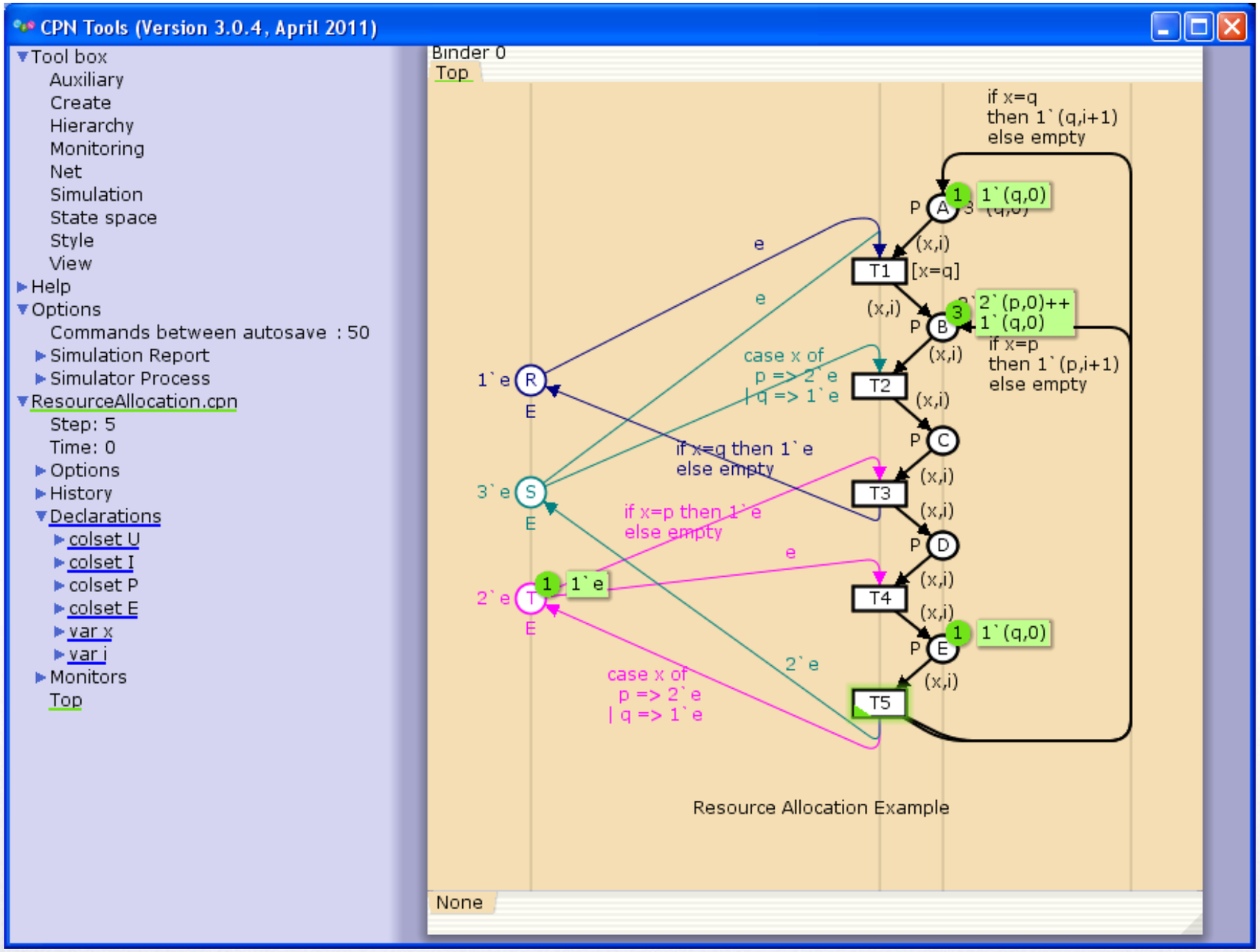

CPN Tools (Version 3.0.4, April 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
Commands between autosave : 50
Simulation Report
Simulator Process
ResourceAllocation.cpn
Step: 1
Time: 0
Options
History
Declarations
colset U
colset I
colset P
colset E
var x
var i
Monitors
Top
Binder 0
Top
if x=q
then 1`(q,i+1)
else empty
2`(q,0)
P
A
(x,i)
T1
[x=q]
e
e
2`(p,0)++
1`(q,0)
if x=p
then 1`(p,i+1)
else empty
B
case x of
p => 2`e
| q => 1`e
T2
C
T3
D
T4
E
T5
1`e
R
E
3`e
S
2`e
if x=q then 1`e
else empty
if x=p then 1`e
else empty
2`e
T
case x of
p => 2`e
| q => 1`e
Resource Allocation Example
None

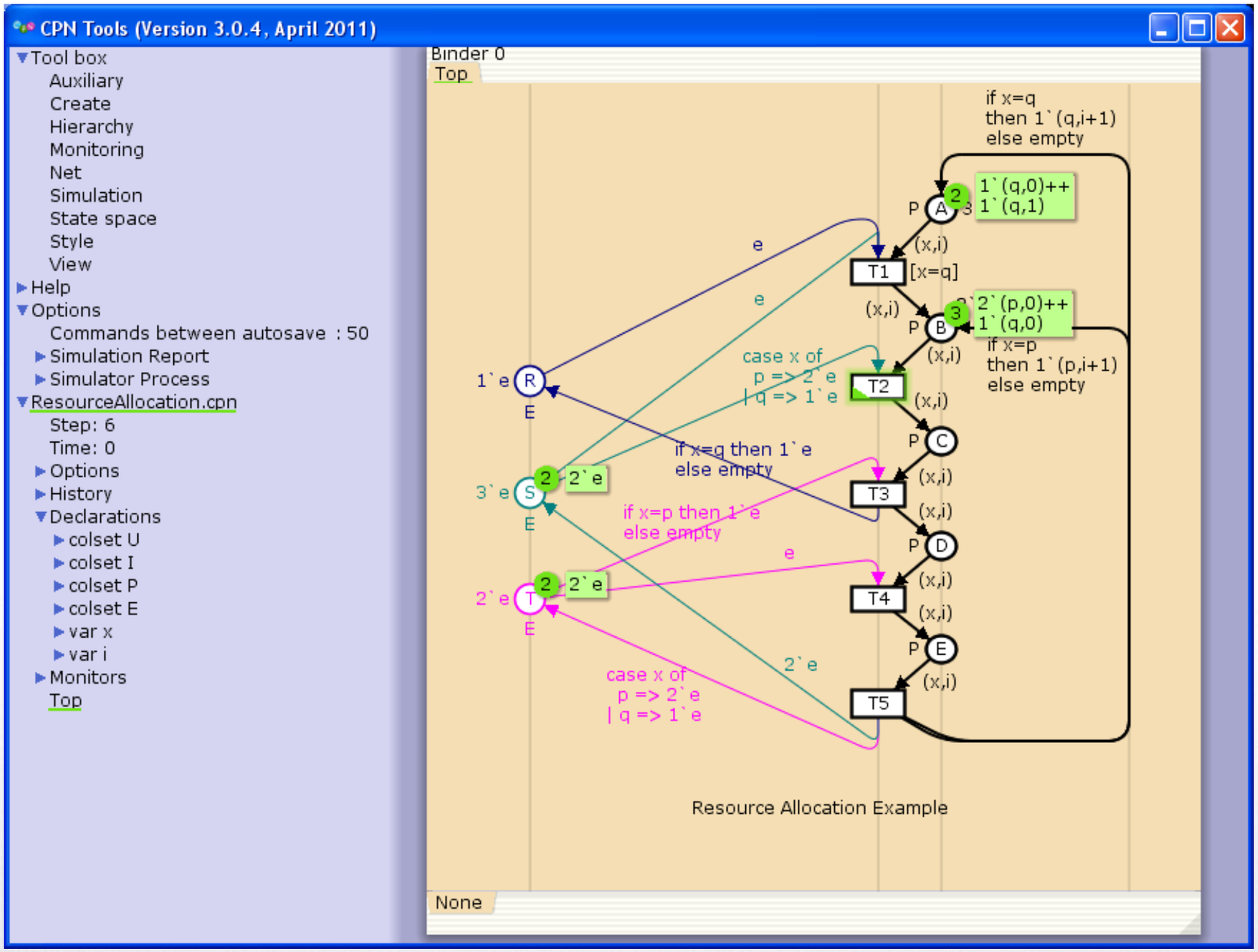

CPN Tools (Version 3.0.4, April 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
Commands between autosave : 50
Simulation Report
Simulator Process
ResourceAllocation.cpn
Step: 2
Time: 0
Options
History
Declarations
colset U
colset I
colset P
colset E
var x
var i
Monitors
Top
Binder 0
Top
if x=q
then 1`(q,i+1)
else empty
2`(q,0)
P
A
(x,i)
T1
[x=q]
2`(p,0)
B
if x=p
then 1`(p,i+1)
else empty
T2
1`(q,0)
C
T3
D
T4
E
T5
1`e
R
E
3`e
S
2`e
T
case x of
p => 2`e
| q => 1`e
if x=q then 1`e
else empty
if x=p then 1`e
else empty
e
2`e
Resource Allocation Example
None


CPN Tools (Version 3.0.4, April 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
Commands between autosave : 50
Simulation Report
Simulator Process
ResourceAllocation.cpn
Step: 3
Time: 0
Options
History
Declarations
colset U
colset I
colset P
colset E
var x
var i
Monitors
Top
Binder 0
Top
if x=q
then 1`(q,i+1)
else empty
2`(q,0)
P
A
(x,i)
T1
[x=q]
e
e
(x,i)
2`(p,0)
P
B
(x,i)
case x of
p => 2`e
| q => 1`e
T2
if x=p
then 1`(p,i+1)
else empty
(x,i)
P
C
1`e
R
E
if x=q then 1`e
else empty
(x,i)
T3
3`e
S
E
1`e
if x=p then 1`e
else empty
(x,i)
1`(q,0)
P
D
e
(x,i)
T4
2`e
T
E
2`e
(x,i)
P
E
case x of
p => 2`e
| q => 1`e
2`e
(x,i)
T5
Resource Allocation Example
None


CPN Tools (Version 3.0.4, April 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
Commands between autosave : 50
Simulation Report
Simulator Process
ResourceAllocation.cpn
Step: 4
Time: 0
Options
History
Declarations
colset U
colset I
colset P
colset E
var x
var i
Monitors
Top
Binder 0
Top
if x=q
then 1`(q,i+1)
else empty
2`(q,0)
P
A
(x,i)
T1
[x=q]
2`(p,0)
B
if x=p
then 1`(p,i+1)
else empty
case x of
p => 2`e
| q => 1`e
T2
C
T3
D
T4
E
1`(q,0)
T5
1`e
R
E
S
T
if x=q then 1`e
else empty
if x=p then 1`e
else empty
2`e
Resource Allocation Example
None

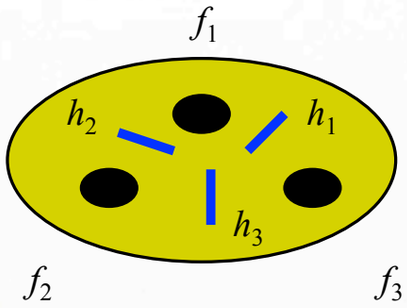

CPN Tools (Version 3.0.4, April 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
Commands between autosave : 50
Simulation Report
Simulator Process
ResourceAllocation.cpn
Step: 5
Time: 0
Options
History
Declarations
colset U
colset I
colset P
colset E
var x
var i
Monitors
Top
Binder 0
Top
if x=q
then 1`(q,i+1)
else empty
1`(q,0)
P
A
(x,i)
T1
[x=q]
e
(x,i)
P
B
2`(p,0)++
1`(q,0)
if x=p
then 1`(p,i+1)
else empty
case x of
p => 2`e
| q => 1`e
T2
(x,i)
P
C
(x,i)
T3
(x,i)
P
D
(x,i)
T4
(x,i)
P
E
1
1`(q,0)
(x,i)
T5
1`e
R
E
3`e
S
E
2`e
T
E
if x=q then 1`e
else empty
if x=p then 1`e
else empty
e
2`e
case x of
p => 2`e
| q => 1`e
Resource Allocation Example
None

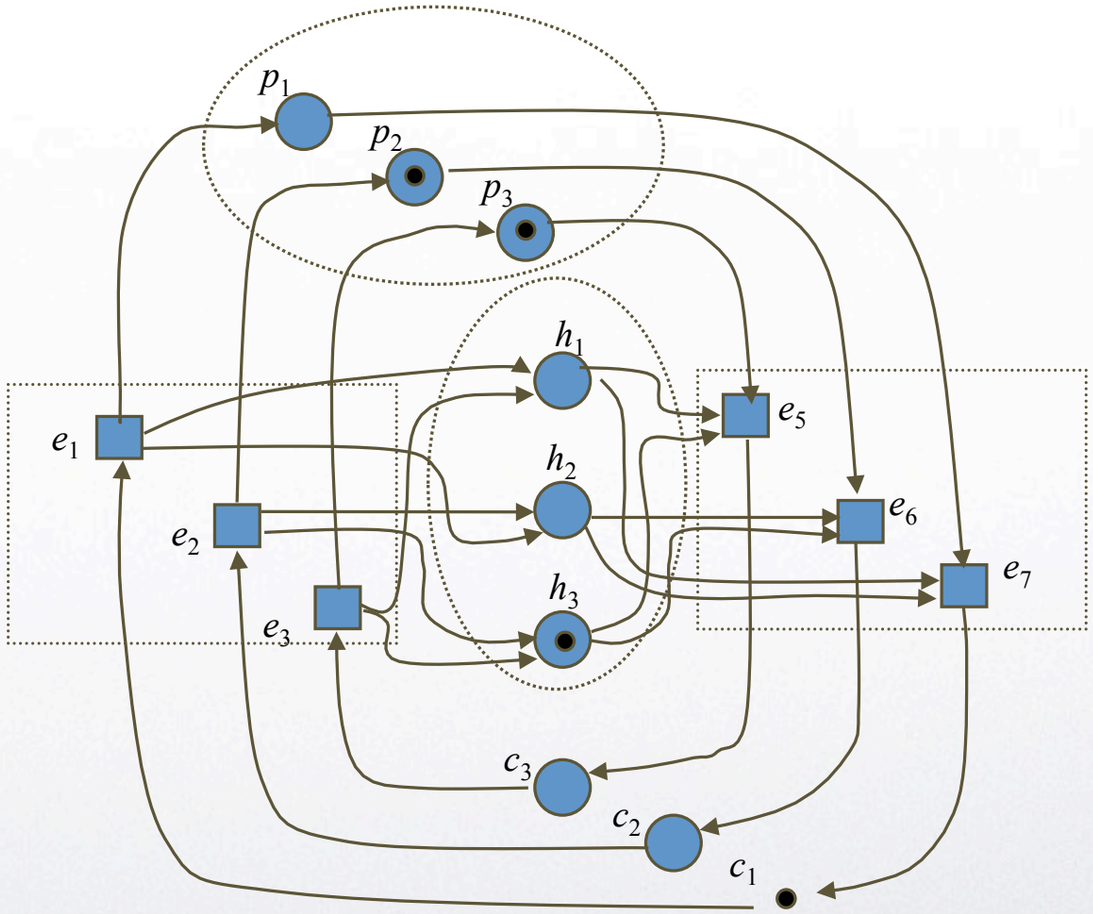

CPN Tools (Version 3.0.4, April 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
Commands between autosave : 50
Simulation Report
Simulator Process
ResourceAllocation.cpn
Step: 5
Time: 0
Options
History
Declarations
colset U
colset I
colset P
colset E
var x
var i
Monitors
Top
Binder 0
Top
if x=q
then 1`(q,i+1)
else empty
P
A
1
1`(q,0)
(x,i)
T1
[x=q]
e
e
(x,i)
P
B
3
2`(p,0)++
1`(q,0)
if x=p
then 1`(p,i+1)
else empty
(x,i)
T2
case x of
p => 2`e
| q => 1`e
1`e
R
E
(x,i)
P
C
if x=q then 1`e
else empty
(x,i)
T3
3`e
S
E
if x=p then 1`e
else empty
(x,i)
P
D
e
(x,i)
T4
(x,i)
P
E
1
1`(q,0)
2`e
T
E
1
1`e
2`e
(x,i)
case x of
p => 2`e
| q => 1`e
T5
Resource Allocation Example
None

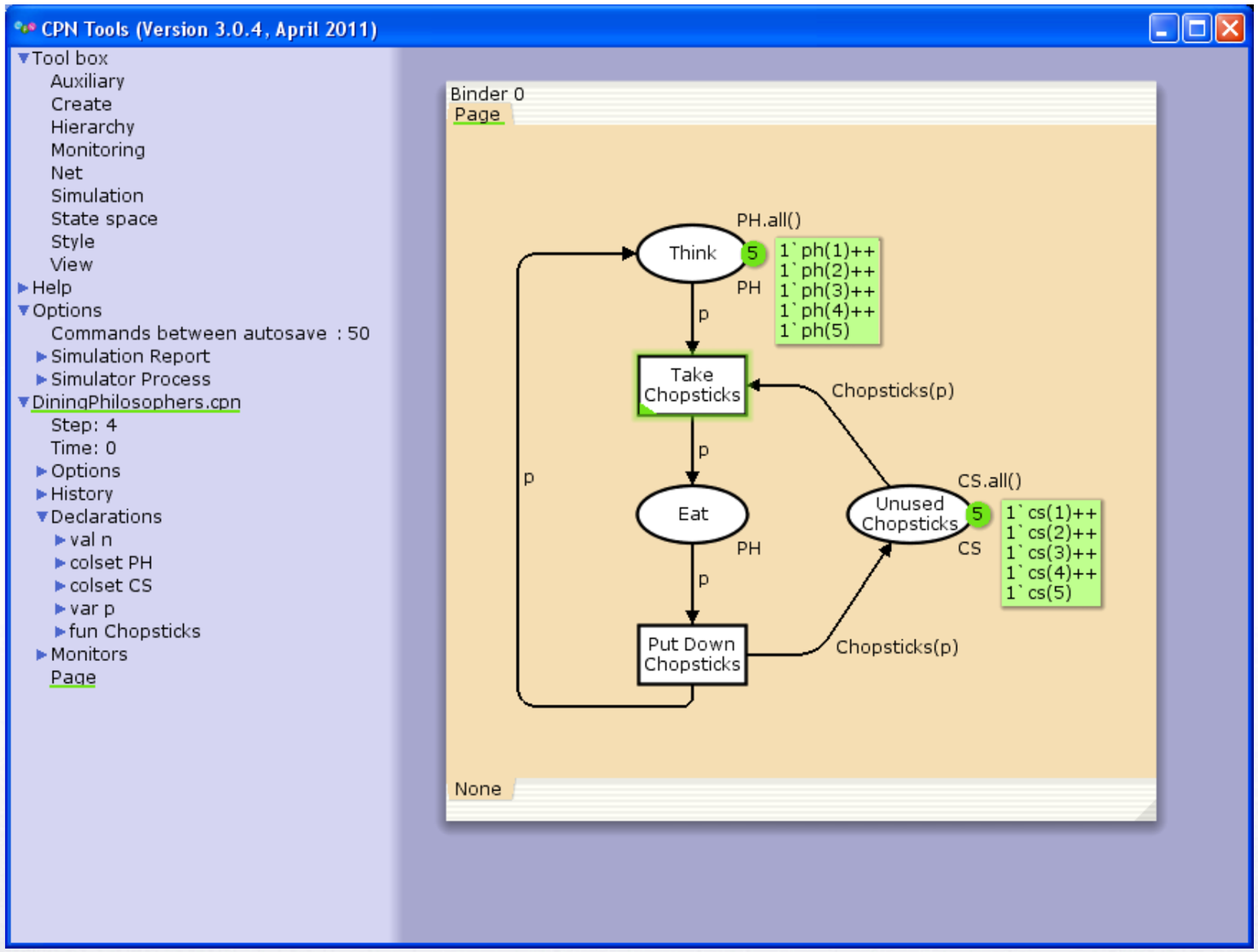

CPN Tools (Version 3.0.4, April 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
Commands between autosave : 50
Simulation Report
Simulator Process
ResourceAllocation.cpn
Step: 6
Time: 0
Options
History
Declarations
colset U
colset I
colset P
colset E
var x
var i
Monitors
Top
Binder 0
Top
if x=q
then 1`(q,i+1)
else empty
1`(q,0)++
1`(q,1)
P
A
(x,i)
T1
[x=q]
e
e
(x,i)
2`(p,0)++
1`(q,0)
P
B
3
if x=p
then 1`(p,i+1)
else empty
(x,i)
case x of
p => 2`e
| q => 1`e
T2
(x,i)
1`e
R
E
P
C
if x=q then 1`e
else empty
(x,i)
T3
3`e
S
2
2`e
E
if x=p then 1`e
else empty
(x,i)
P
D
e
2
2`e
2`e
T
E
T4
(x,i)
P
E
2`e
(x,i)
case x of
p => 2`e
| q => 1`e
T5
Resource Allocation Example
None

Modelagem clássica para o problema dos filósofos

O exemplo dos filósofos






CPN Tools (Version 3.0.4, April 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Options
Commands between autosave : 50
Simulation Report
Simulator Process
DiningPhilosophers.cpn
Step: 4
Time: 0
Options
History
Declarations
val n
colset PH
colset CS
var p
fun Chopsticks
Monitors
Page
Binder 0
Page
PH.all()
Think
5
PH
1`ph(1)++
1`ph(2)++
1`ph(3)++
1`ph(4)++
1`ph(5)
p
Take Chopsticks
Chopsticks(p)
p
p
CS.all()
Eat
PH
Unused Chopsticks
5
CS
1`cs(1)++
1`cs(2)++
1`cs(3)++
1`cs(4)++
1`cs(5)
p
Put Down Chopsticks
Chopsticks(p)
None

Modelagem com CPNs

- Diferente abordagem, mais abstração
- Classificar elementos do projeto segundo tipos bem definidos
- A observância dos bindings (strong type) é fundamental
- Explorando as simetrias (pode não aparecer desta forma)
- Maior facilidade para reutilização

Alguns problemas...

- Diferente abordagem, mais abstração

  Depende do background do designer, mas a questão base é escolher entre fazer a modelagem já direto em rede colorida ou começar pela rede clássica

- Classificar elementos do projeto segundo tipos bem definidos

  Consequencia do item anterior e novamente é uma questão de diciplina.

- A observância dos bindings (strong type) é fundamental

  Comum nos processos estruturados, choca-se com a excessiva carga funcional que os engenheiros costumar dar aos projetos.

- Explorando as simetrias (pode não aparecer desta forma)

  De fato deve ser feito também no caso clássico, o tipo de simetria a ser explorada é que é diferente.

- Maior facilidade para reutilização

  Será amplicado no tanto no caso clássico com no das redes de alto nível com a introdução da hierarquia, mas continua mais forte no caso das redes de alto nível, embora a escolha da componente reutilizável seja mais complexa.

Sobre o formalismo

Portanto, a única maneira de minorar os problema é voltar ao formalismo para explorar melhor a análise de propriedades, especialmente os invariantes.

O formalismo também abre a possibilidade de se ter outras interfaces mais amigáveis e adaptadas aos ambientes de trabalho, tendo o CPN Tools (por exemplo) por trás. Nesse caso a aderência ao formalismo das CPNs é importante e fundamental para a credibilidade da ferramenta.

CPN : definição formal

Definition: A Coloured Petri Net is a tuple CPN = (Σ, P, T, A, N, C, G, E, I) satisfying the following requirements:

(i) Σ is a finite set of non-empty types, called **colour sets**.

(ii) P is a finite set of **places**.

(iii) T is a finite set of **transitions**.

(iv) A is a finite set of **arcs** such that:

- $P \cap T = P \cap A = T \cap A = \emptyset$.

(v) N is a **node** function. It is defined from A into $P \times T \cup T \times P$.

(vi) C is a **colour** function. It is defined from P into Σ.

(vii) G is a **guard** function. It is defined from T into expressions such that:

- $\forall t \in T$: [Type(G(t)) = Bool $\wedge$ Type(Var(G(t))) $\subseteq \Sigma$].

(viii) E is an **arc expression** function. It is defined from A into expressions such that:

- $\forall a \in A$: [Type(E(a)) = $C(p(a))_{MS} \wedge$ Type(Var(E(a))) $\subseteq \Sigma$]

where p(a) is the place of N(a).

(ix) I is an **initialization** function. It is defined from P into closed expressions such that:

- $\forall p \in P$: [Type(I(p)) = $C(p)_{MS}$].

Fim