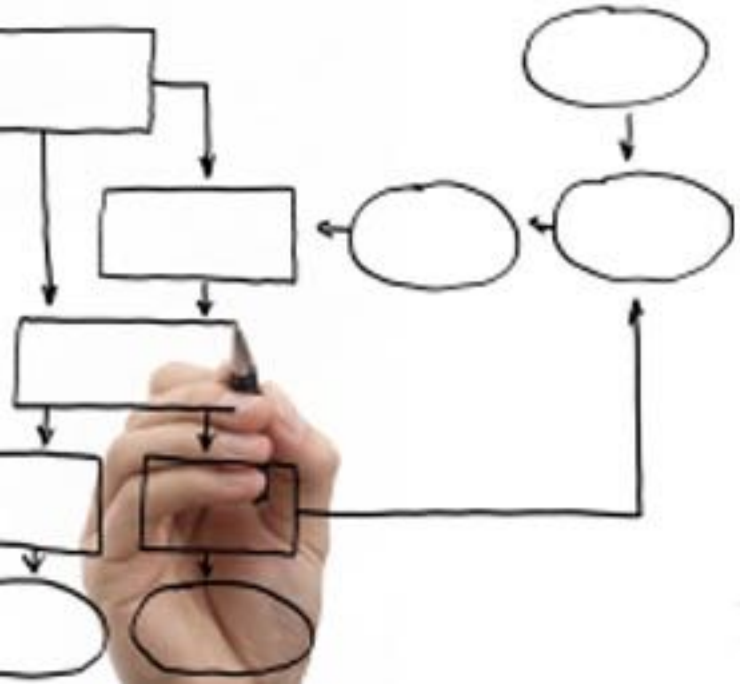
Business Processes Modelling

MPB (6 cfu, 295AA)

Roberto Bruni
http://www.di.unipi.it/~bruni



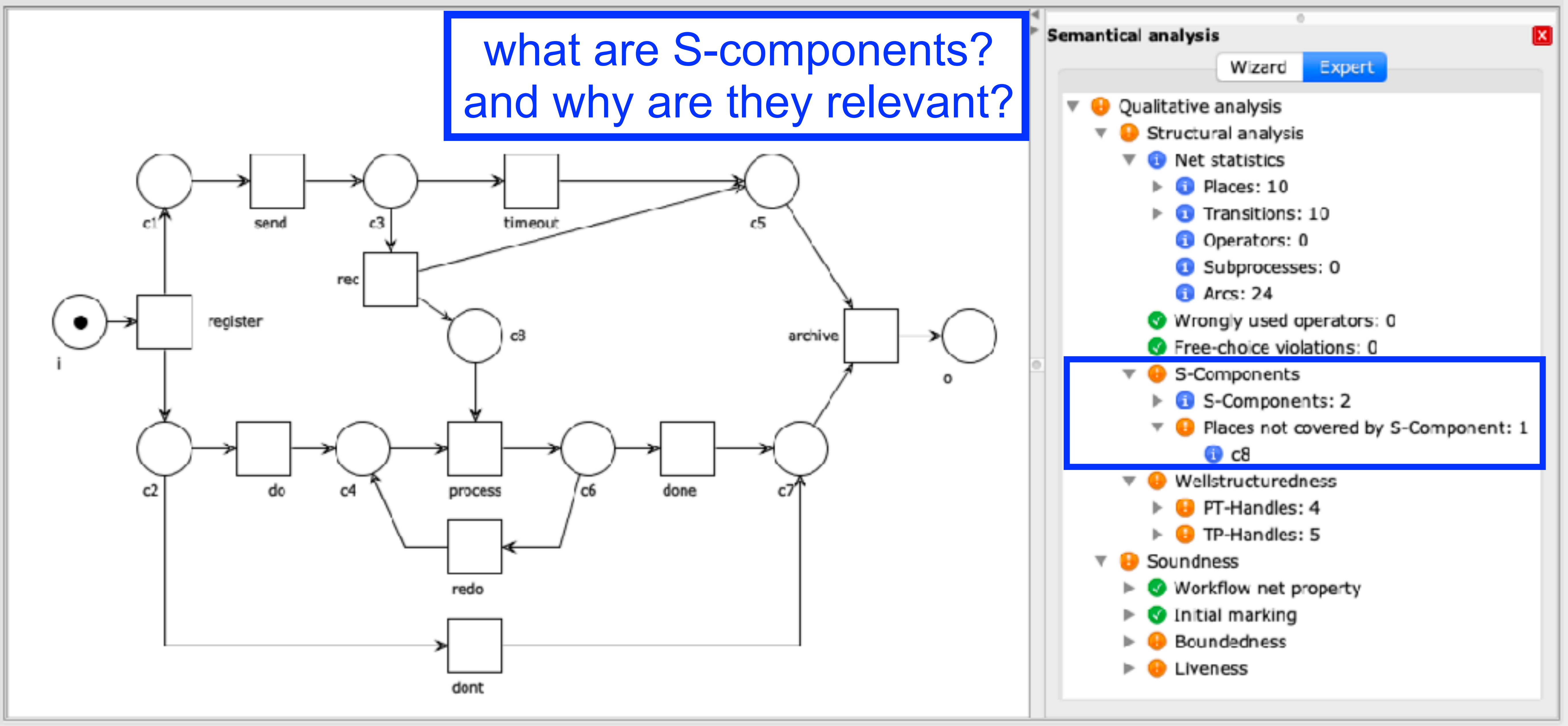
19 - Diagnosis for WF nets

Object

We study suitable diagnosis techniques for unsound Workflow nets

Diagnosing workflow processes using Woflan (article, optional reading)
http://wwwis.win.tue.nl/~wvdaalst/publications/p135.pdf

Woped



what are S-components? and why are they relevant?

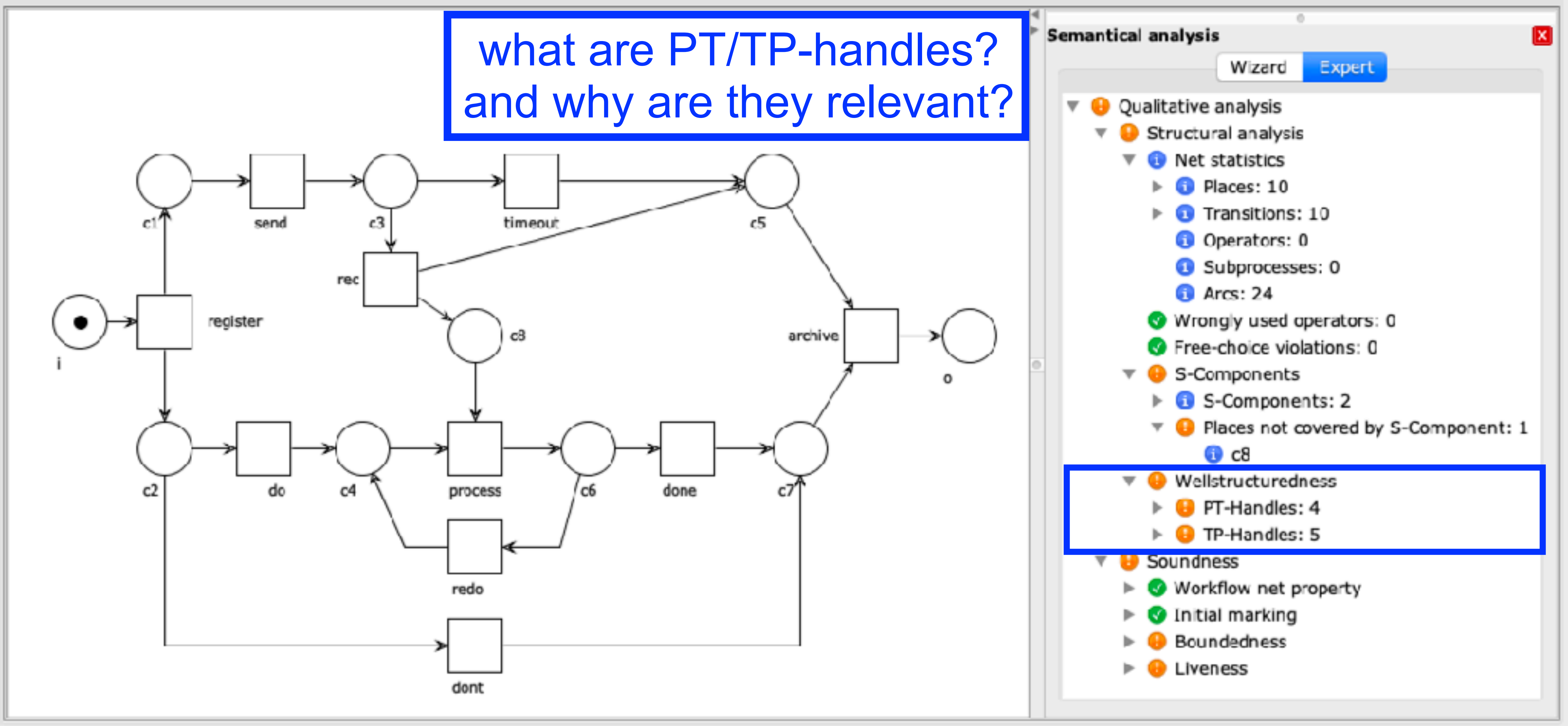
i
register
c1
c2
send
c3
timeout
c5
rec
c8
archive
o
do
c4
process
c6
done
c7
redo
dont

Semantical analysis

Wizard
Expert

- Qualitative analysis
  - Structural analysis
    - Net statistics
      - Places: 10
      - Transitions: 10
      - Operators: 0
      - Subprocesses: 0
      - Arcs: 24
    - Wrongly used operators: 0
    - Free-choice violations: 0
    - S-Components
      - S-Components: 2
      - Places not covered by S-Component: 1
        - c8
    - Wellstructuredness
      - PT-Handles: 4
      - TP-Handles: 5
  - Soundness
    - Workflow net property
    - Initial marking
    - Boundedness
    - Liveness

Woped


c1
send
c3
timeout
c5
rec
register
i
c8
archive
o
c2
do
c4
process
c6
done
c7
redo
dont
Semantical analysis
Wizard
Expert

what are PT/TP-handles?
and why are they relevant?

- Qualitative analysis
  - Structural analysis
    - Net statistics
      - Places: 10
      - Transitions: 10
      - Operators: 0
      - Subprocesses: 0
      - Arcs: 24
    - Wrongly used operators: 0
    - Free-choice violations: 0
    - S-Components
      - S-Components: 2
      - Places not covered by S-Component: 1
        - c8
    - Wellstructuredness
      - PT-Handles: 4
      - TP-Handles: 5
  - Soundness
    - Workflow net property
    - Initial marking
    - Boundedness
    - Liveness

S-Coverability

Rank Theorem (main result, proof omitted)

Theorem:

A free-choice system (P,T,F,M_0) is live and bounded

iff

1. it has at least one place and one transition
2. it is connected
3. M_0 marks every proper siphon
4. it has a positive S-invariant
5. it has a positive T-invariant
6. $\text{rank}(N) = |C_N| - 1$

(where C_N is the set of clusters)

A technique to find a positive S-invariant

A case is often composed by parallel threads of control
(each thread imposing some order over its tasks)

Decompose the net N in suitable S-nets
so that any place of N belongs to some S-net
(the same place can appear in more S-nets)

Each S-net induces a uniform S-invariant

A positive S-invariant is obtained
as the sum of the S-invariants of each subnet

S-component

take a set of nodes

Definition: Let $N = (P, T, F)$ and $\emptyset \subset X \subseteq P \cup T$
Let $N' = (P \cap X, T \cap X, F \cap (X \times X))$ be a subnet of N.
N' is an **S-component** if

forget the arcs to other nodes

1. it is a strongly connected S-net
2. for every place $p \in X \cap P$, we have $\bullet p \cup p \bullet \subseteq X$

if a place p is taken
then all transitions attached to p must be selected

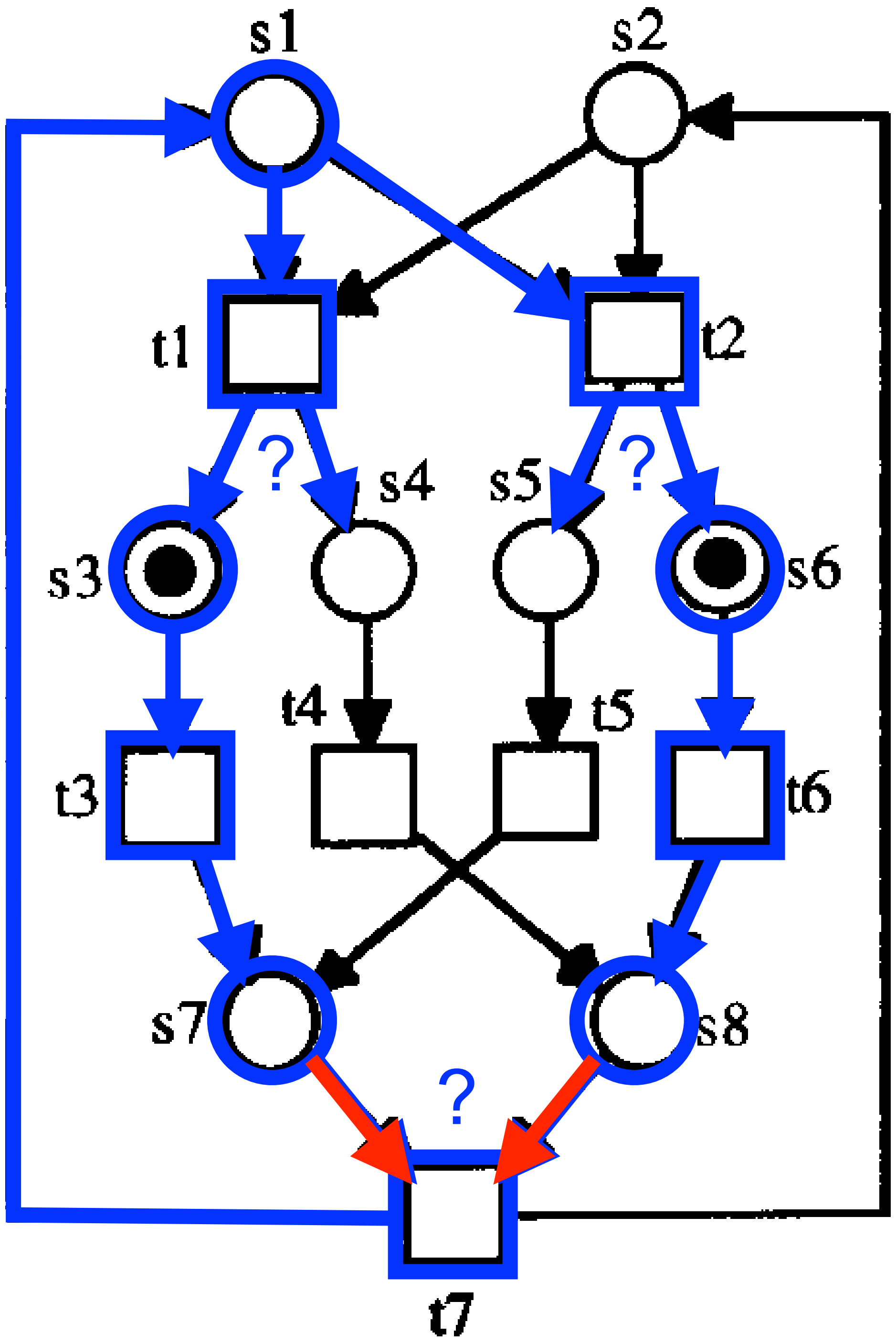
S-cover

Definition: an **S-cover** of a net N
is a set **C** of S-components of N such that
every place p of N belongs to one or more S-components in **C**

N is **S-coverable** if it has an S-cover

if a place p is taken
then all transitions attached to p must be selected

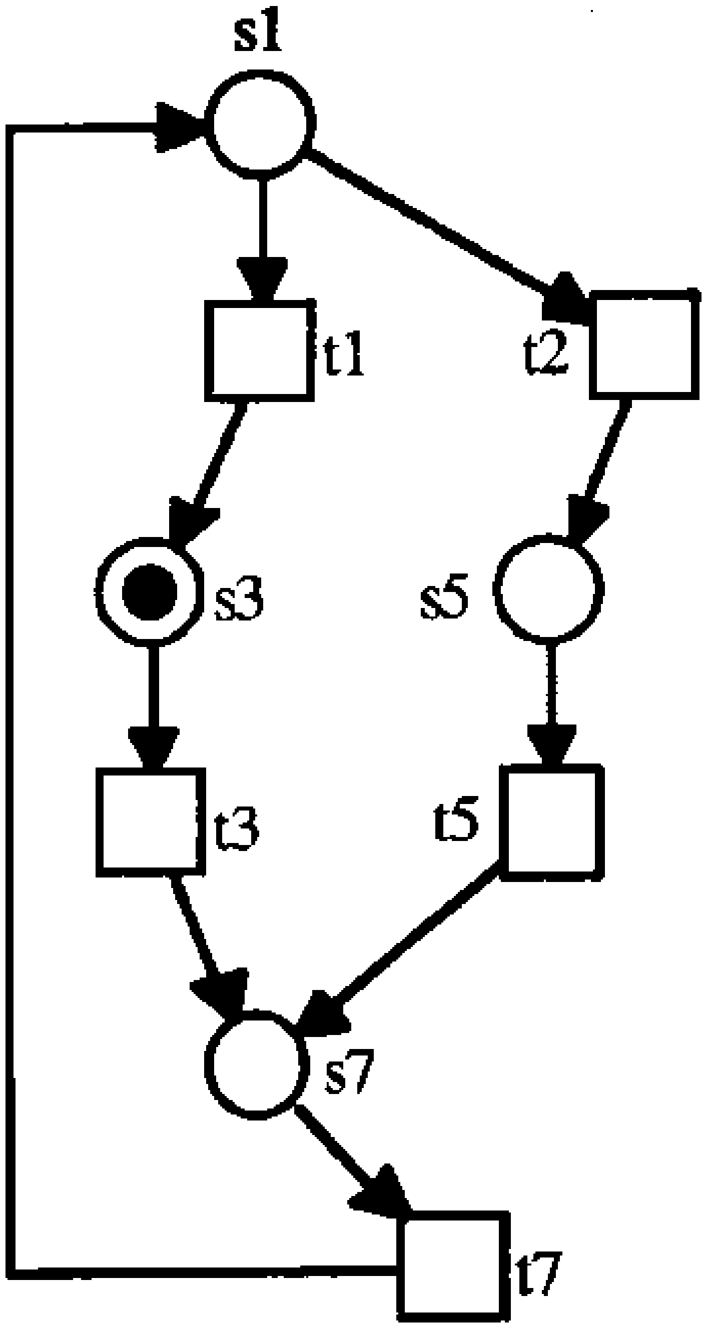
S-cover: example

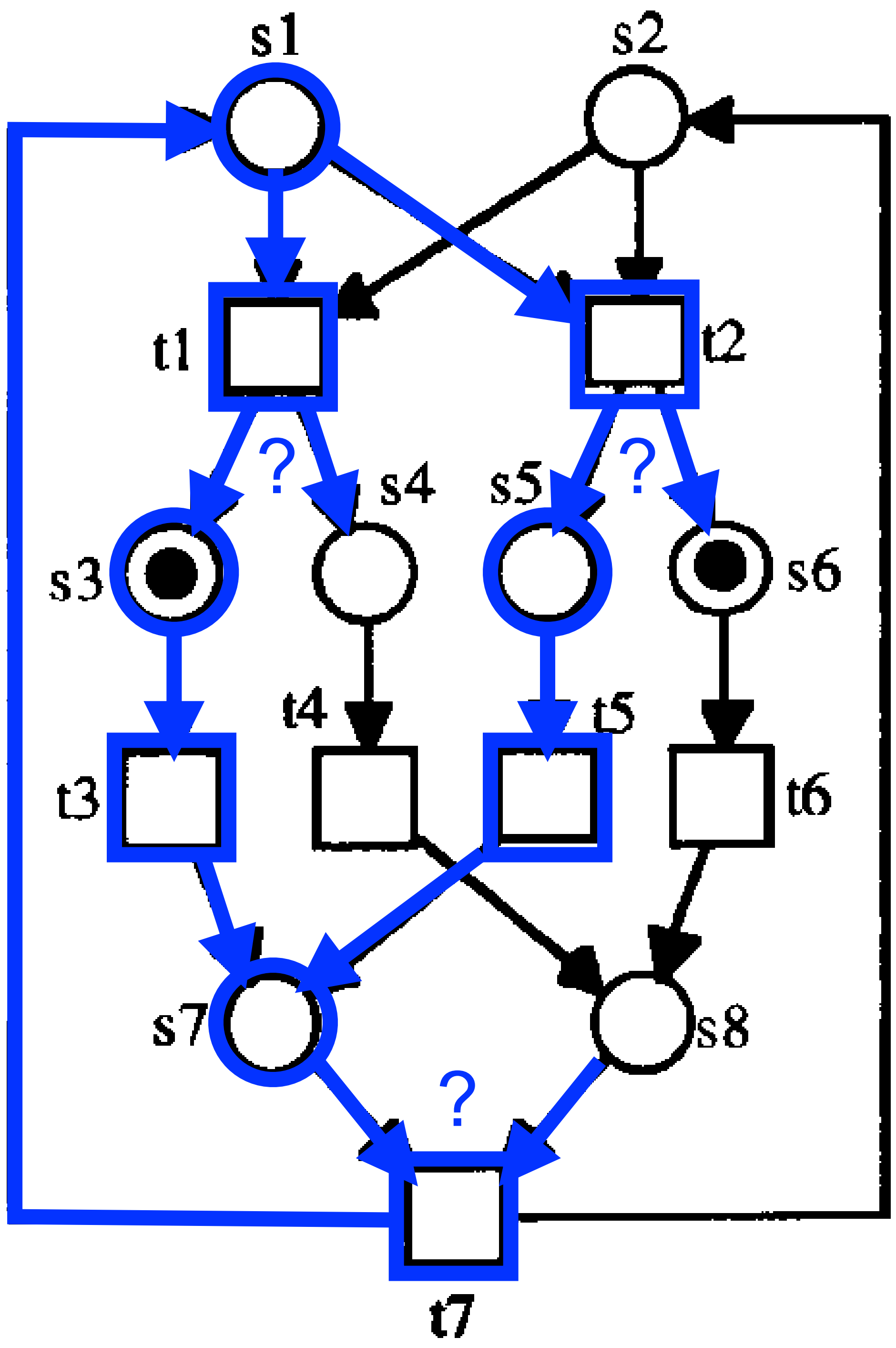

s1
s2
t1
t2
?
?
s4
s5
s3
s6
t4
t5
t3
t6
s7
s8
?
t7

not an S-net

if a place p is taken
then all transitions attached to p must be selected

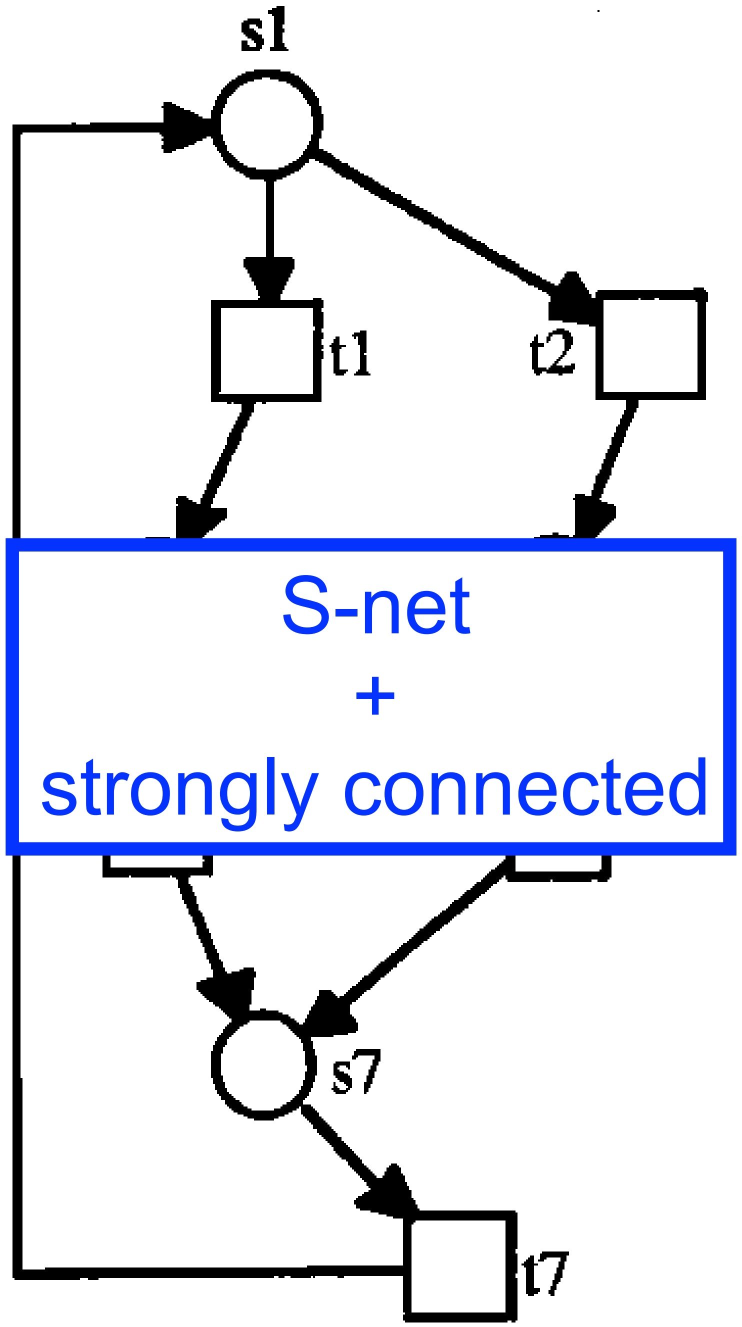
S-cover: example


s1
t1
t2
s3
s5
t3
t5
s7
t7


s1
s2
t1
t2
?
s4
s5
?
s3
s6
t4
t5
t3
t6
s7
?
s8
t7

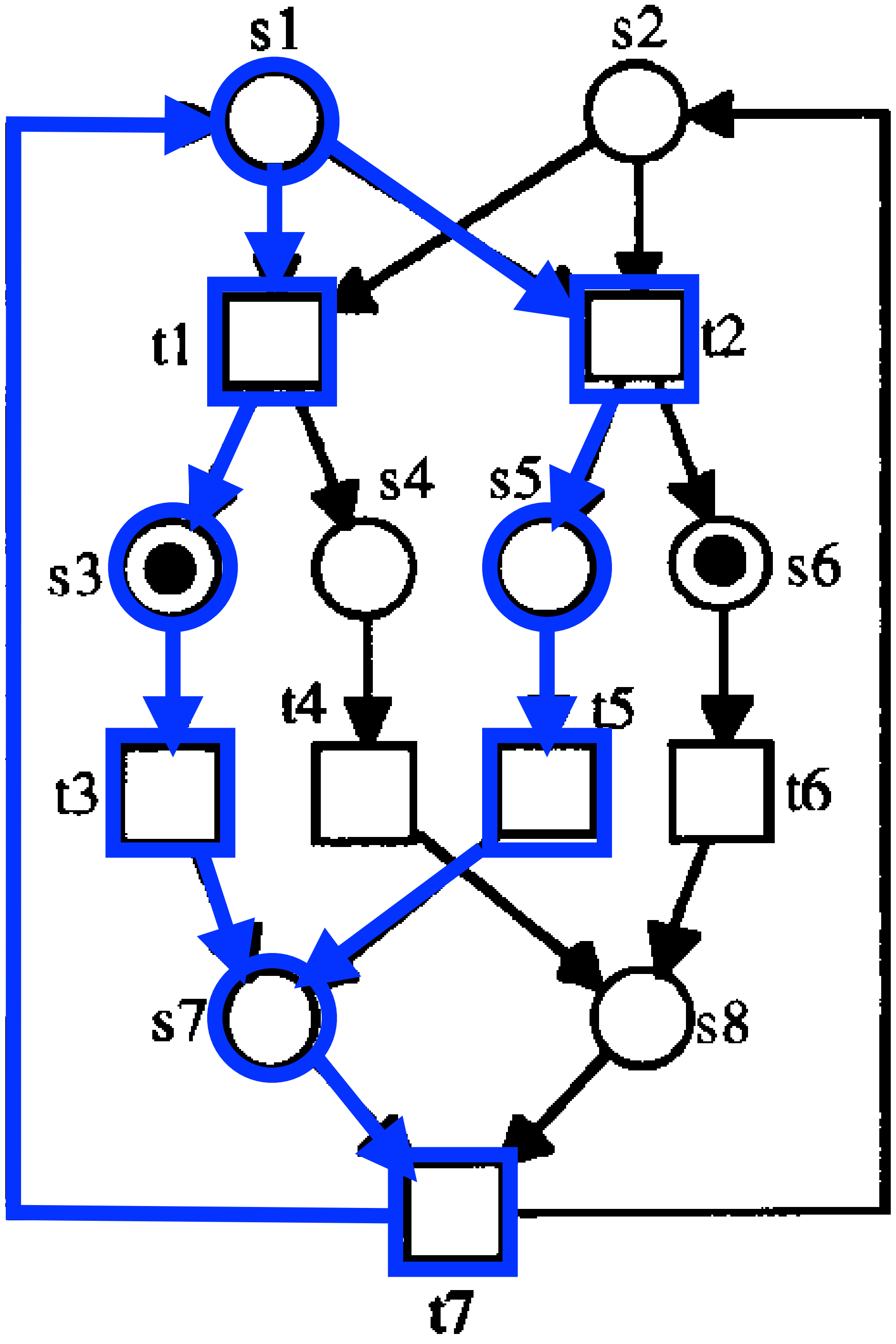
if a place p is taken
then all transitions attached to p must be selected

S-cover: example


s1
t1
t2
s7
t7

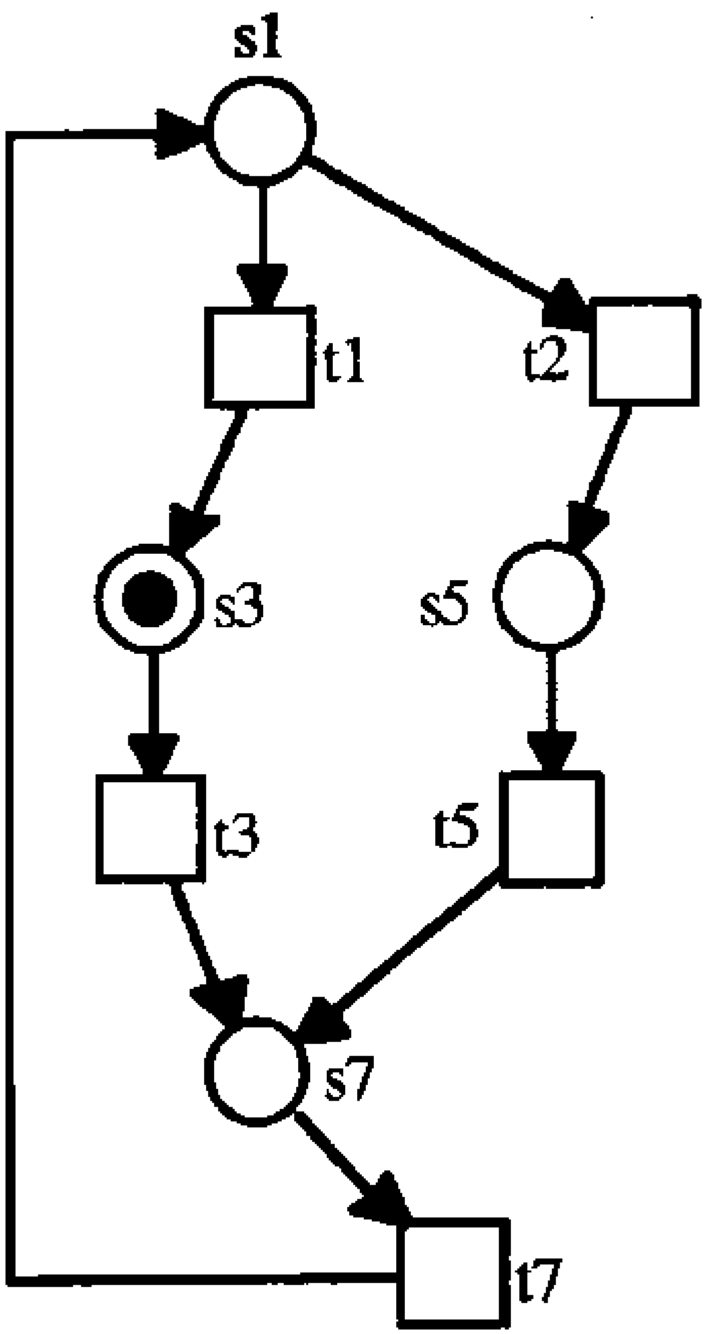
S-net
+
strongly connected

S-component

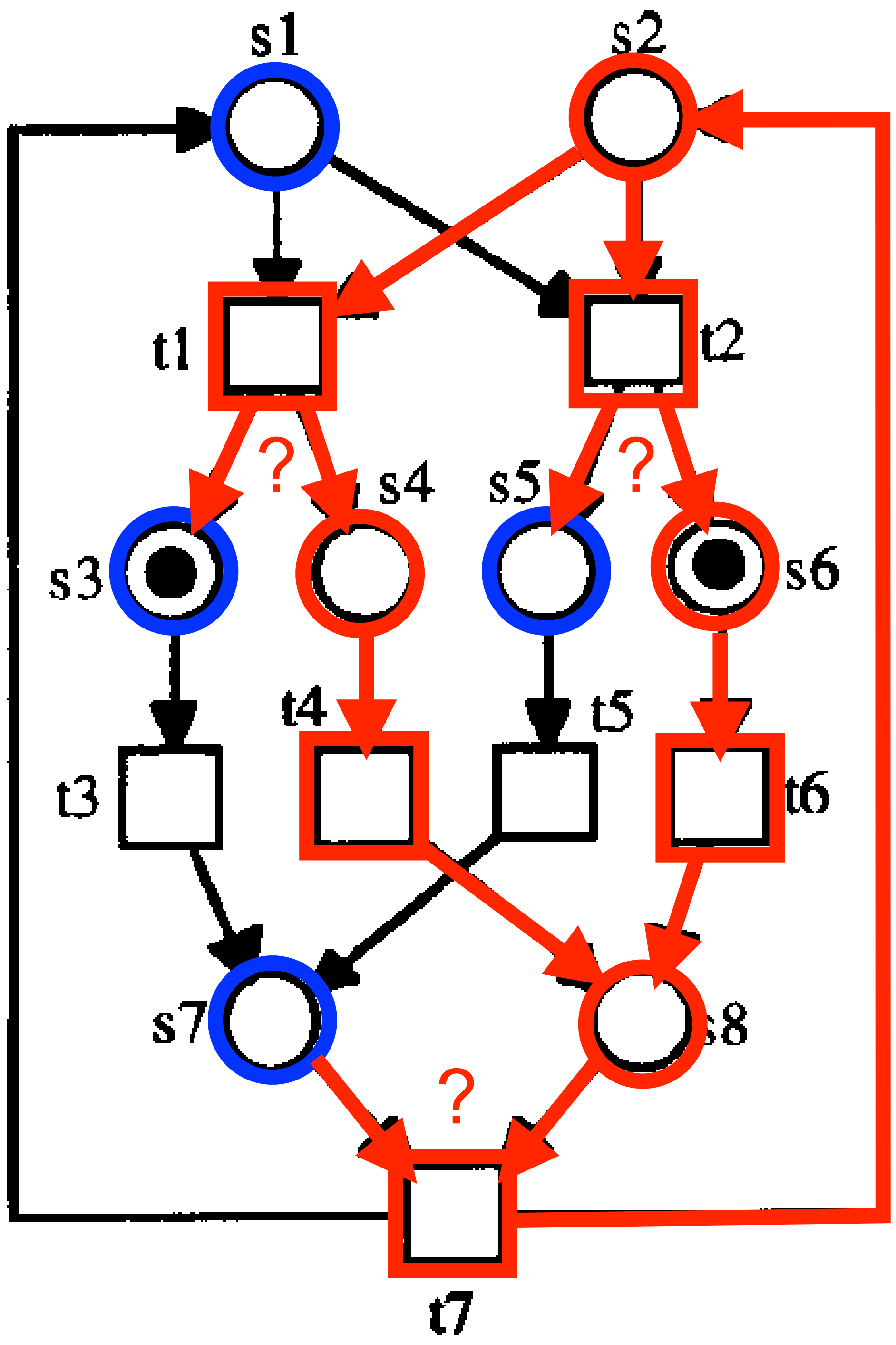

s1
s2
t1
t2
s3
s4
s5
s6
t4
t5
t3
t6
s7
s8
t7

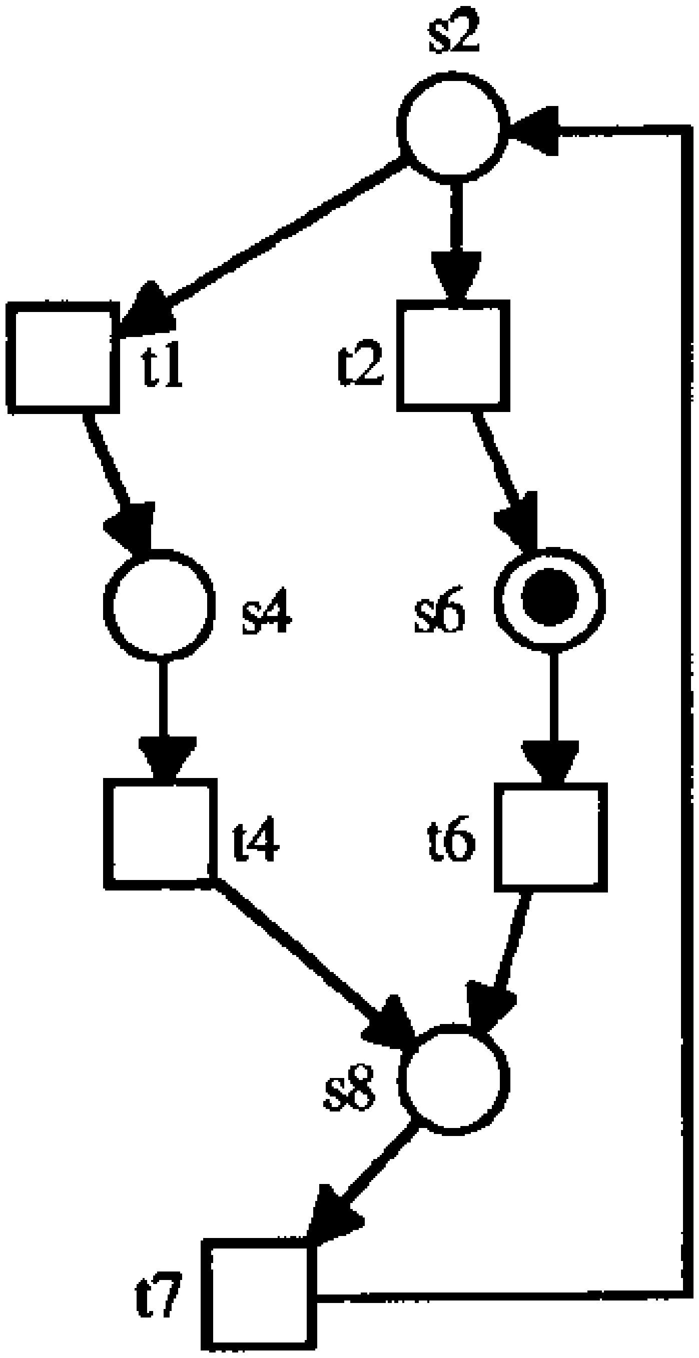
if a place p is taken
then all transitions attached to p must be selected

S-cover: example


s1
t1
t2
s3
s5
t3
t5
s7
t7

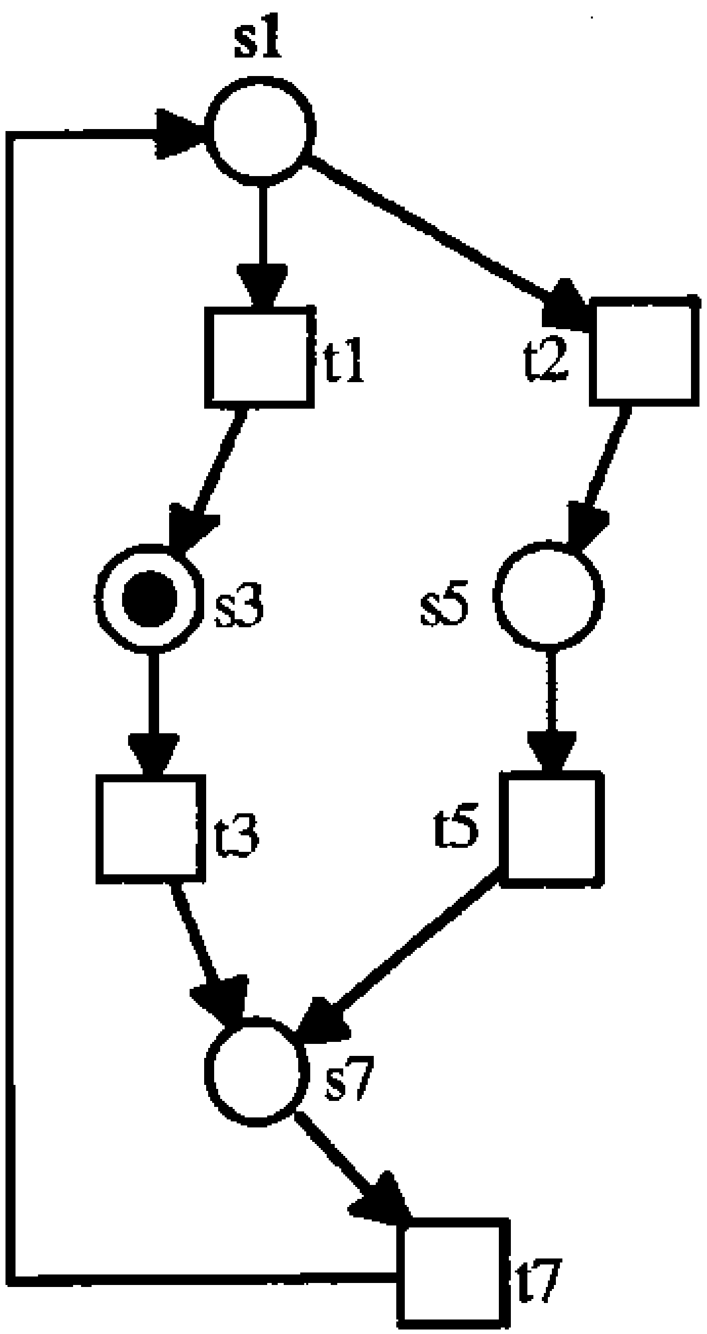
S-component


s1
s2
t1
t2
?
s4
s5
?
s3
s6
t4
t5
t3
t6
s7
s8
?
t7

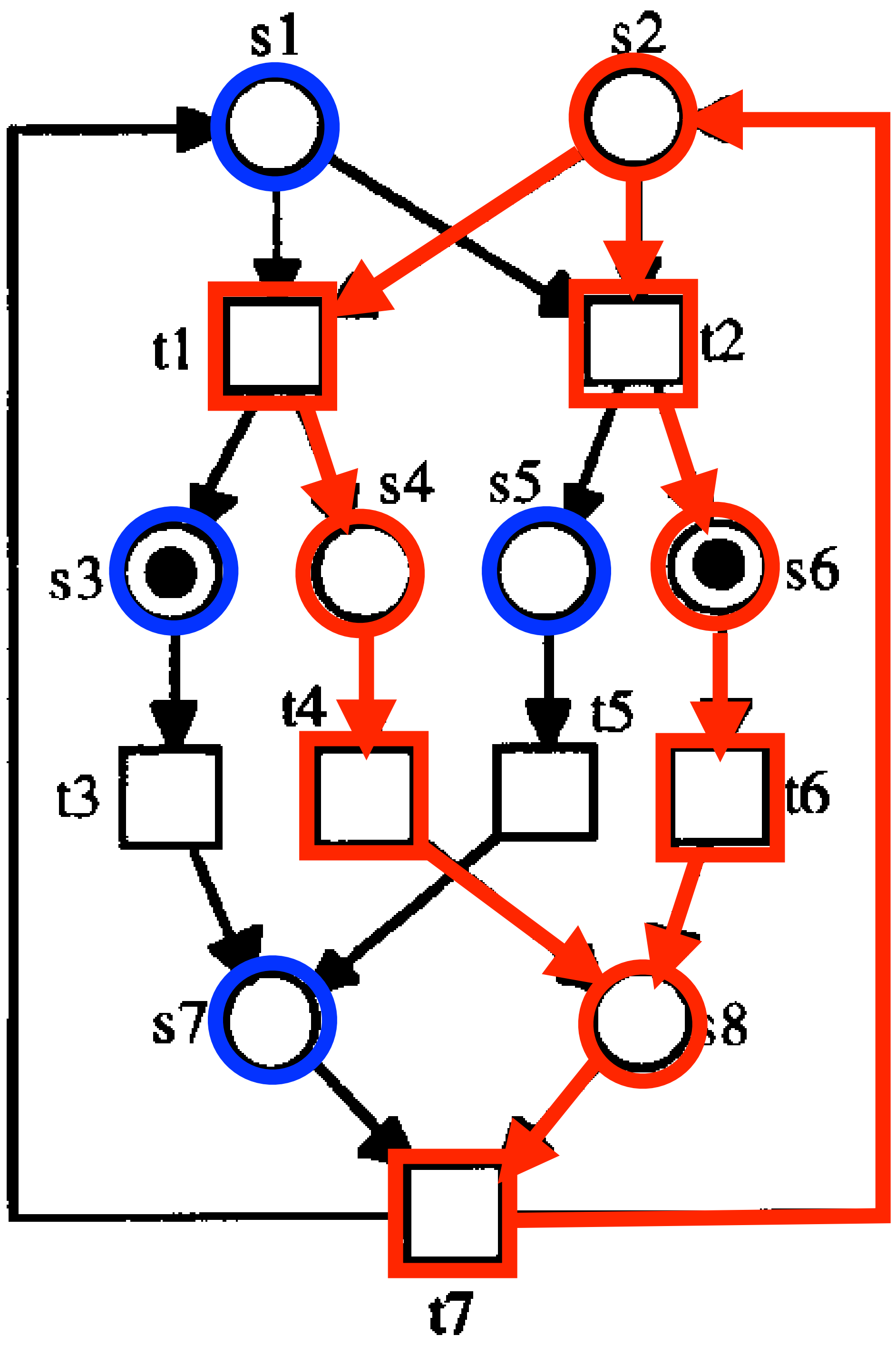

s2
t1
t2
s4
s6
t4
t6
s8
t7

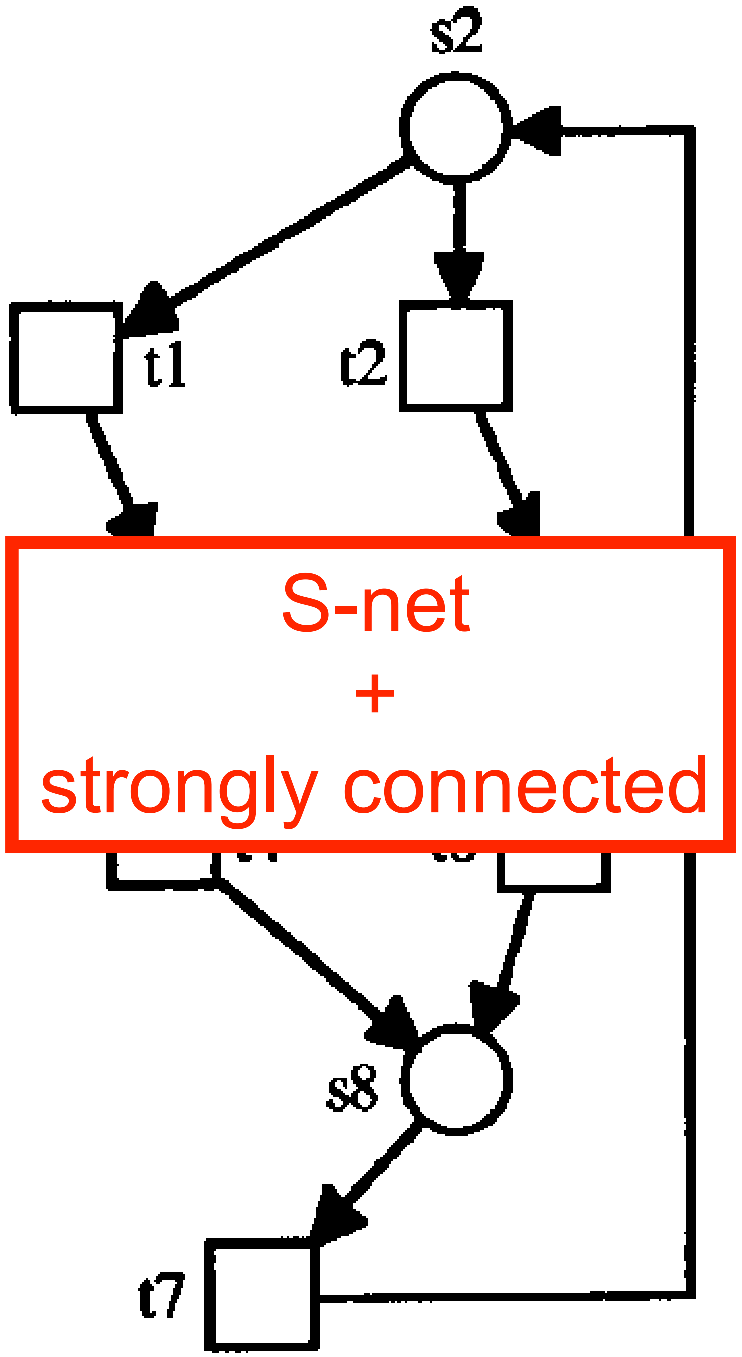
if a place p is taken
then all transitions attached to p must be selected

S-cover: example


s1
t1
t2
s3
s5
t3
t5
s7
t7

S-component


s1
s2
t1
t2
s4
s5
s3
s6
t4
t5
t3
t6
s7
s8
t7

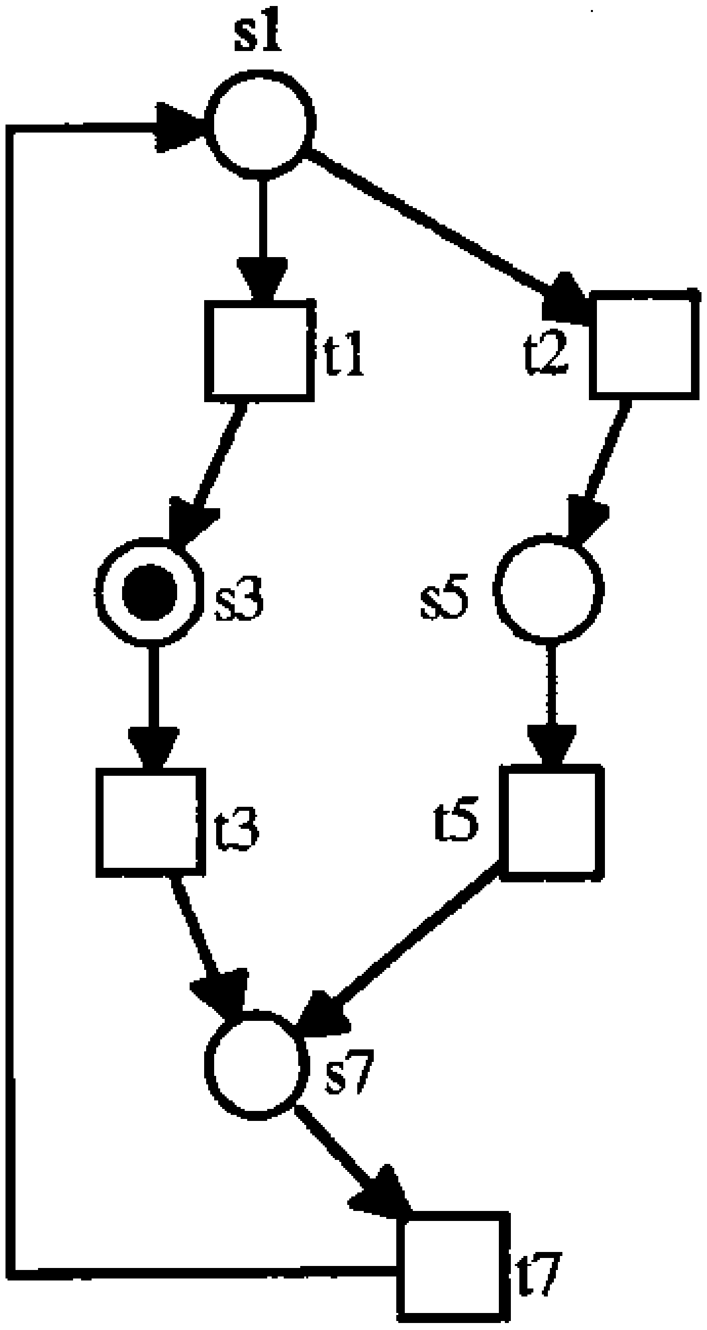

s2
t1
t2
s8
t7

S-net
+
strongly connected

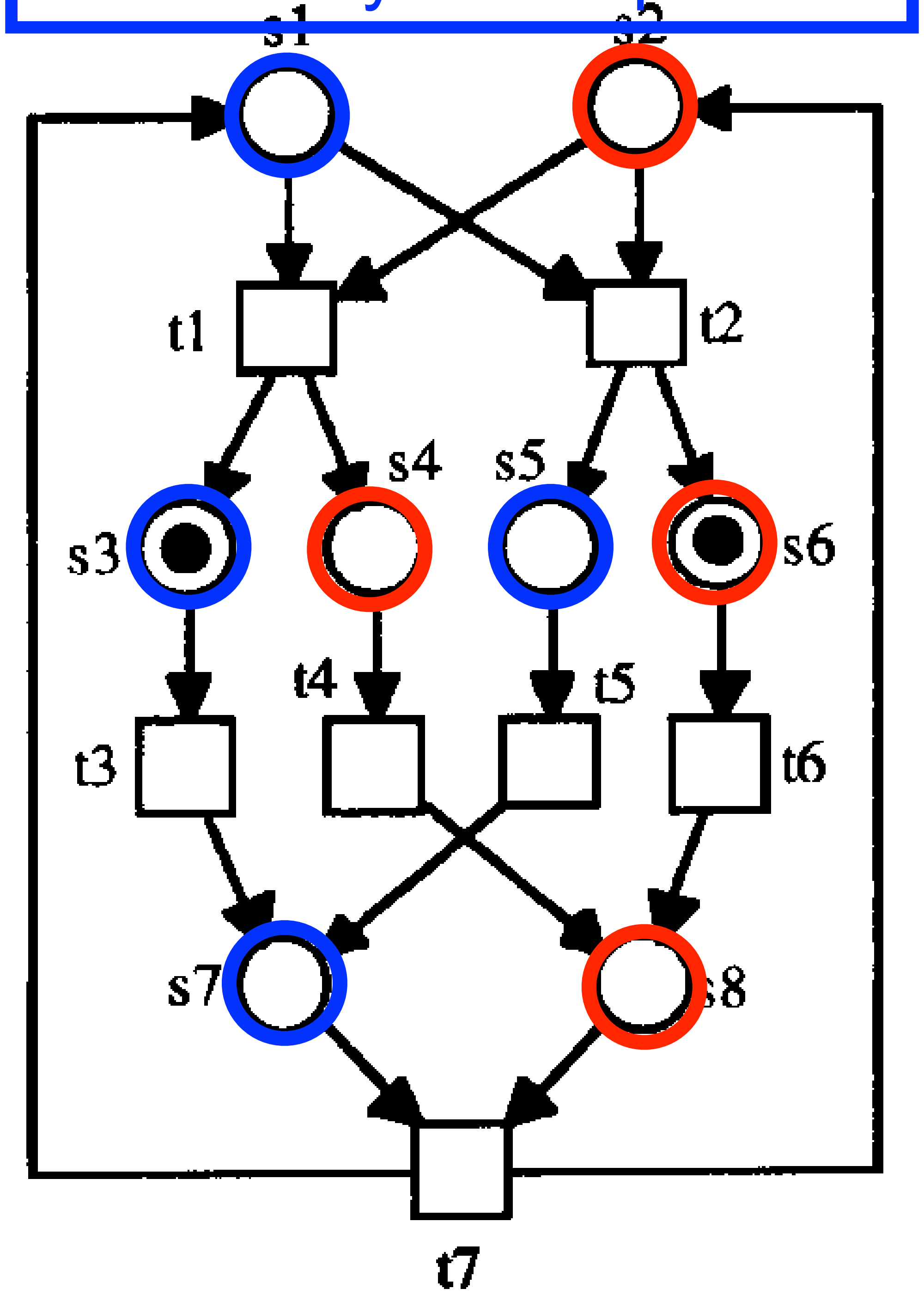
S-component

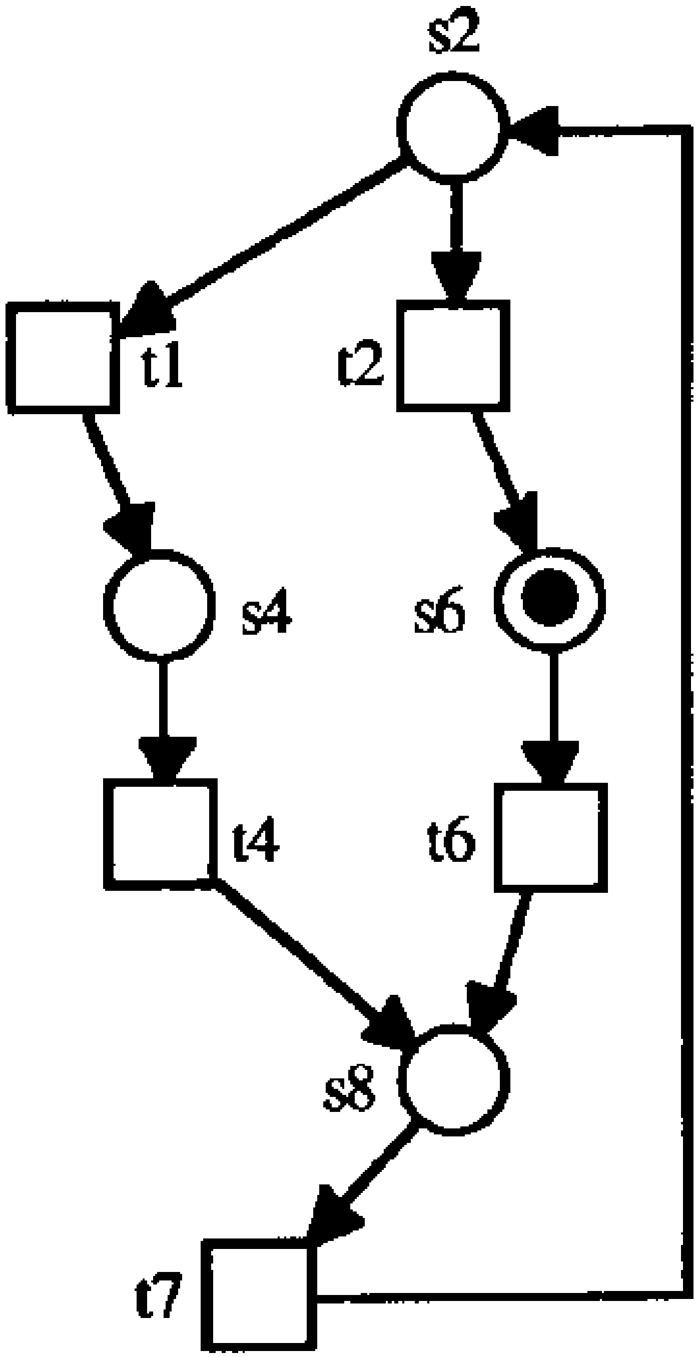
S-cover: example

covered by S-components


s1
t1
t2
s3
s5
t3
t5
s7
t7

S-component

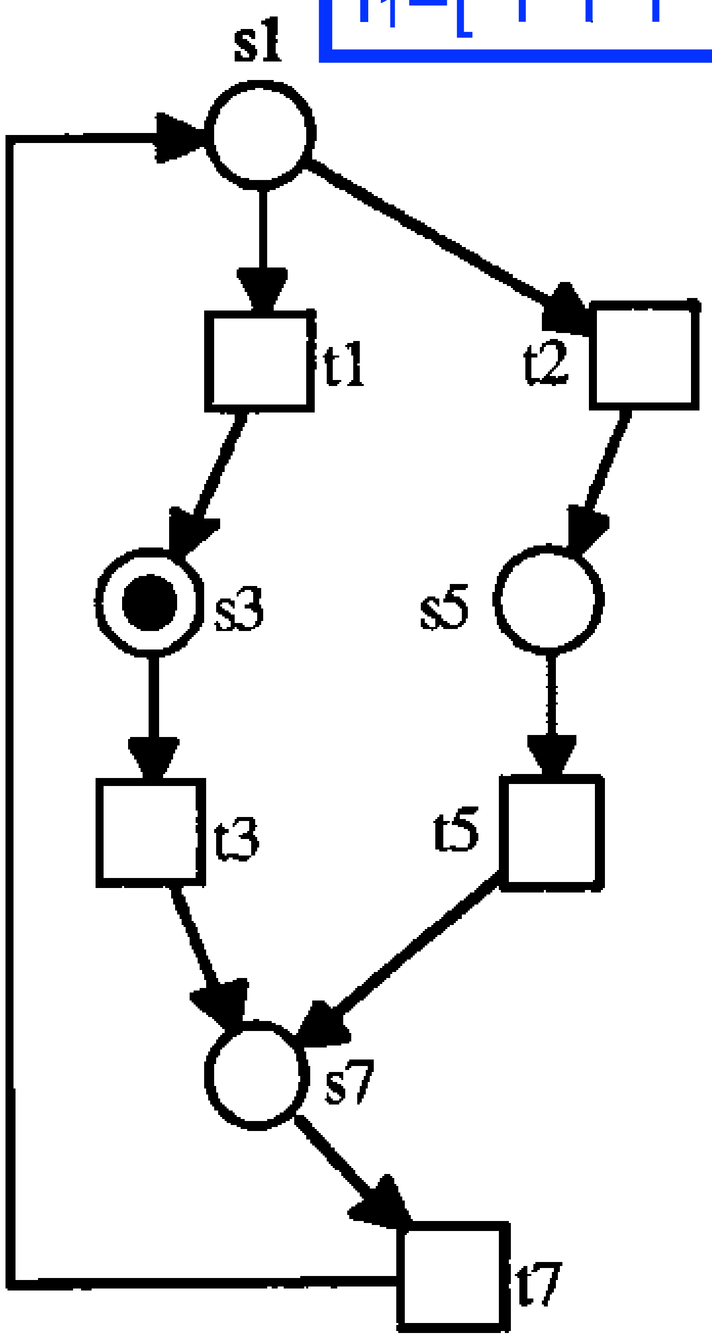

s1
s2
t1
t2
s4
s5
s3
s6
t4
t5
t3
t6
s7
s8
t7


s2
t1
t2
s4
s6
t4
t6
s8
t7

S-component

S-cover: example

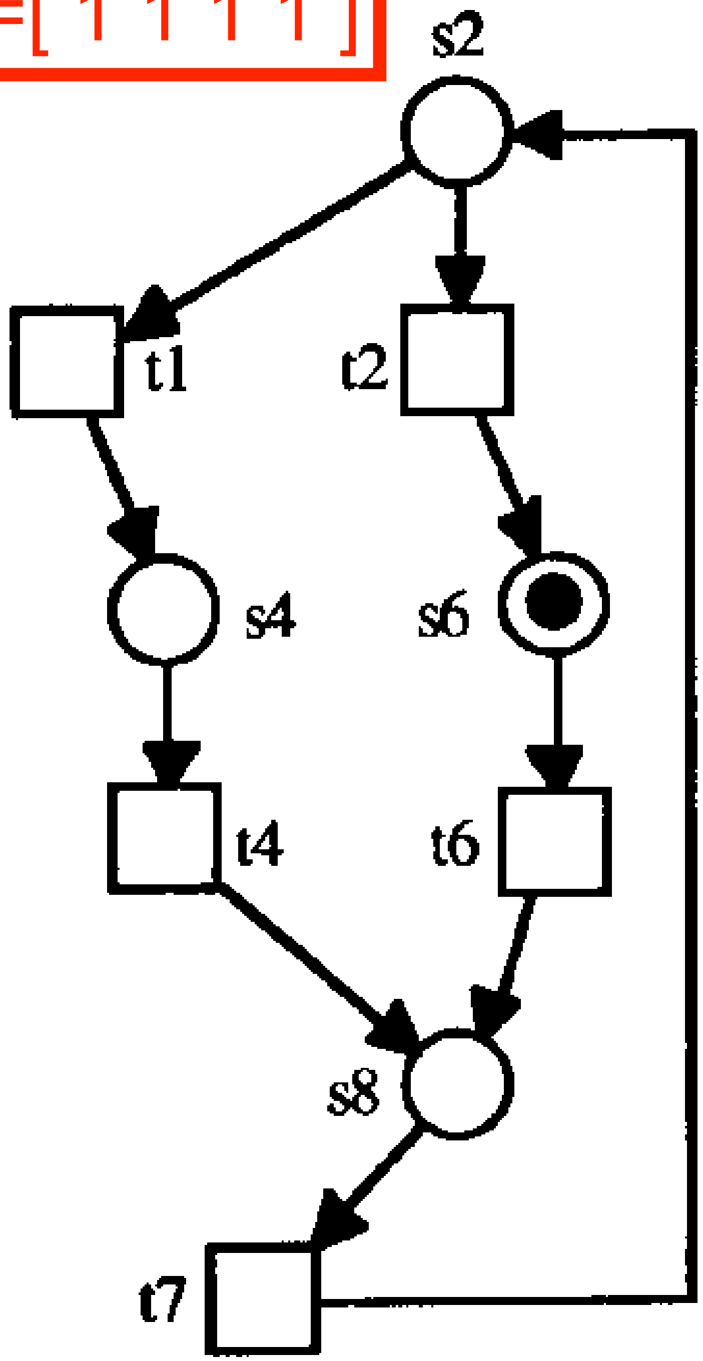
I_1=[1 1 1 1]


s1
t1
t2
s3
s5
t3
t5
s7
t7

S-component

S-cover

I_2=[1 1 1 1]

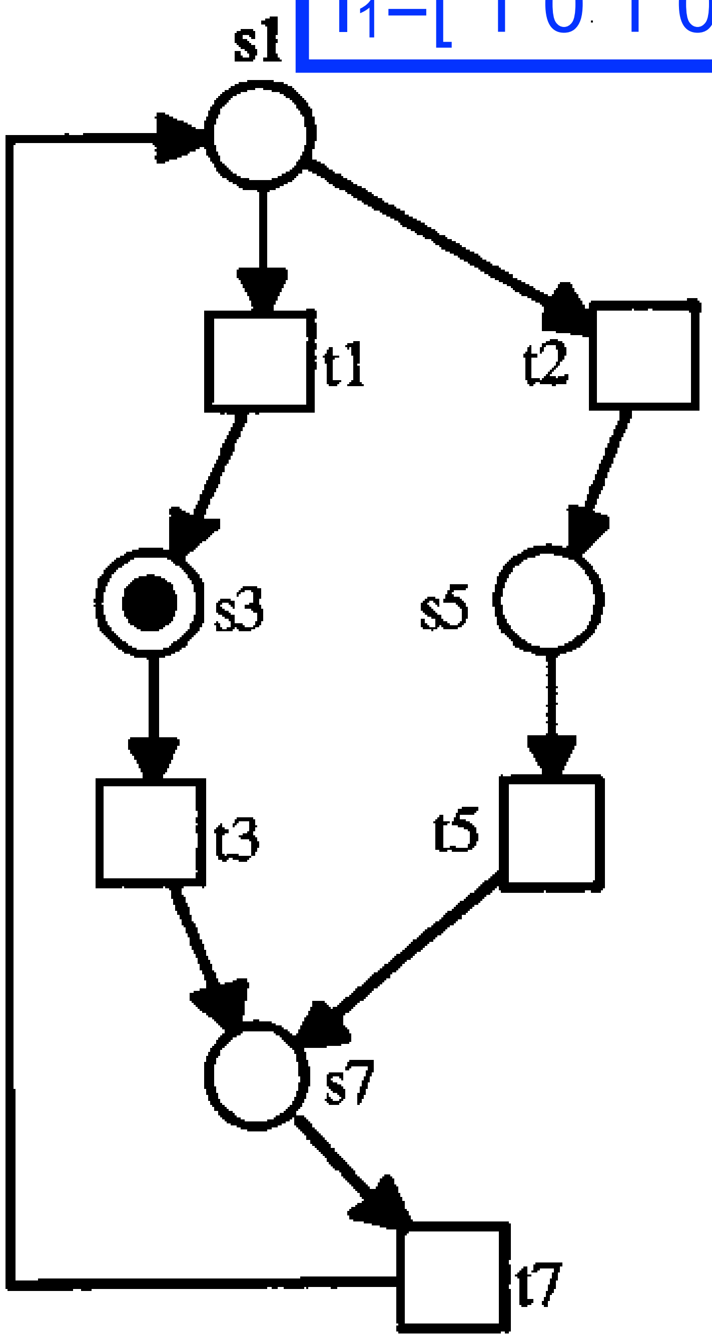

s2
t1
t2
s4
s6
t4
t6
s8
t7

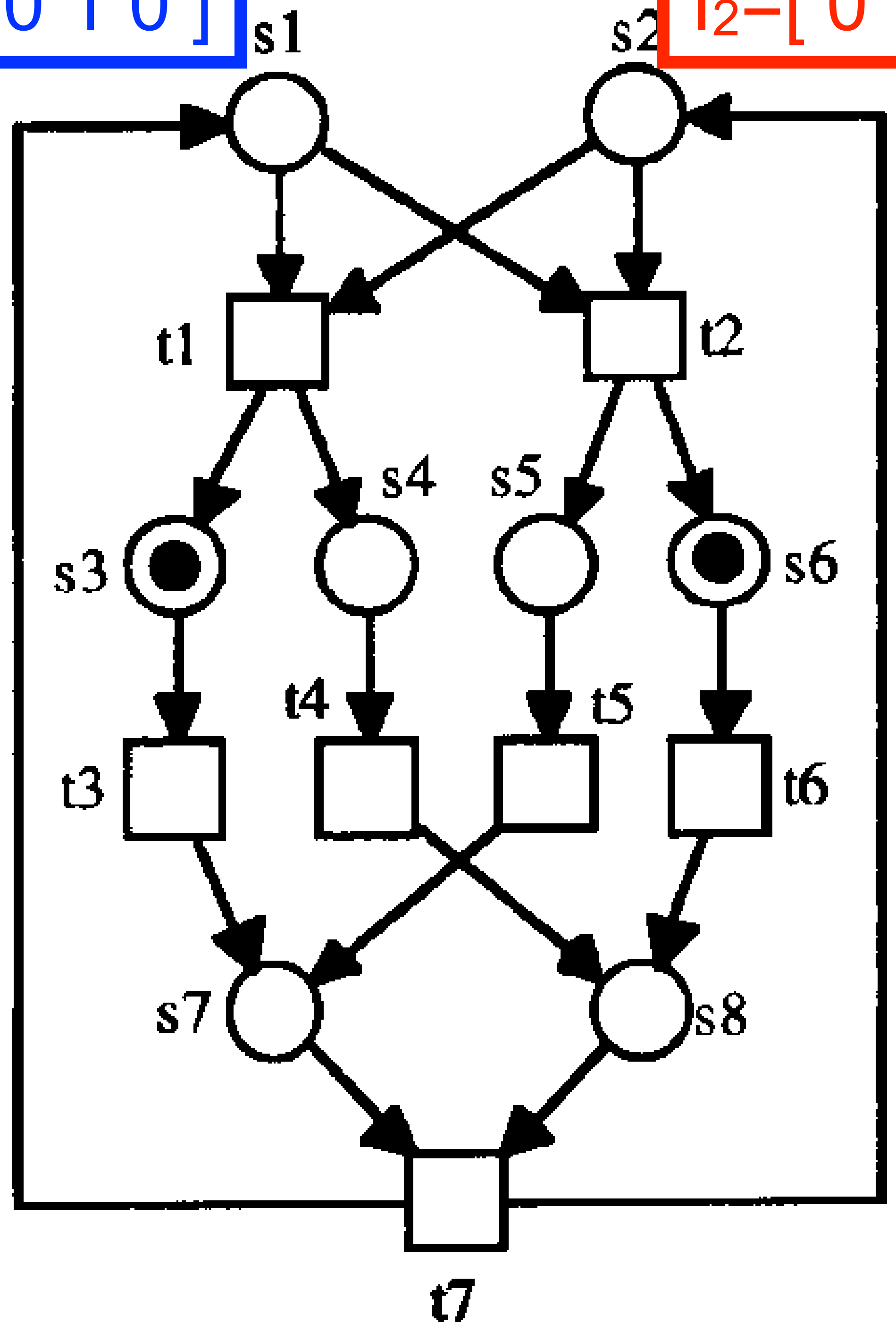
S-component

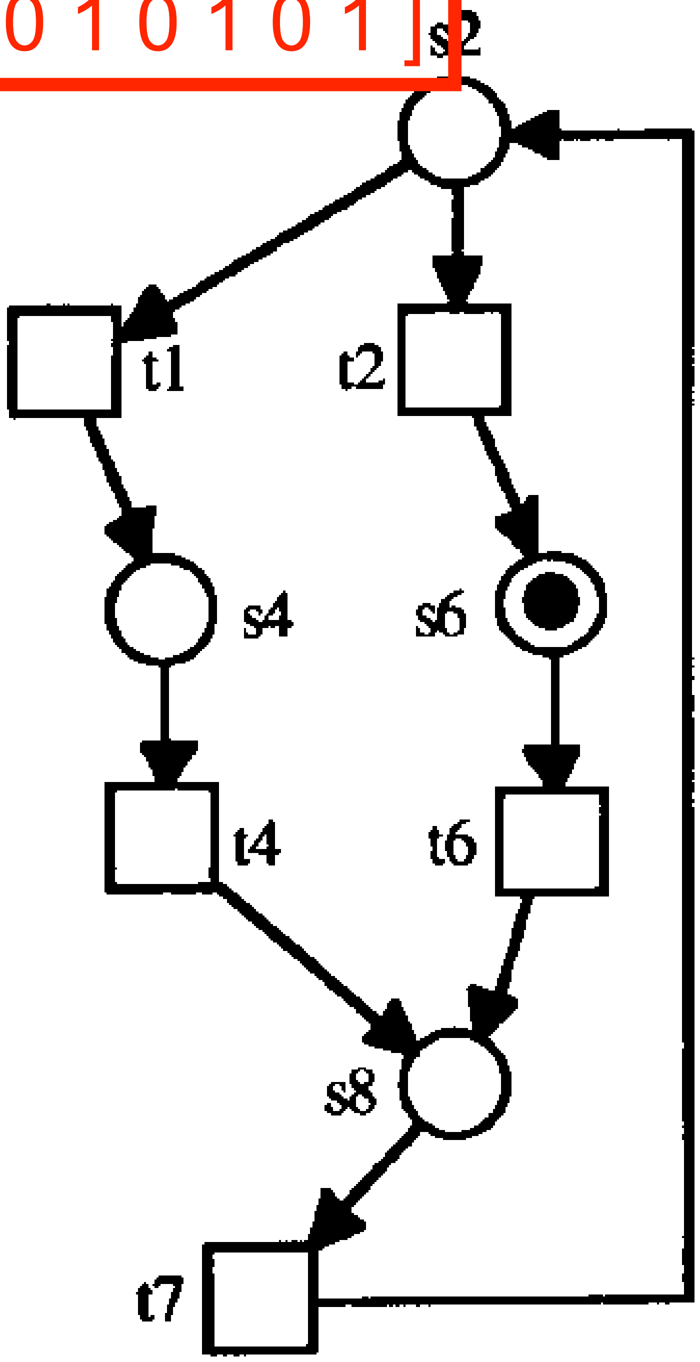
S-cover: example

$I_1=[\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ ]$

$I_2=[\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ ]$


s1
t1
t2
s3
s5
t3
t5
s7
t7

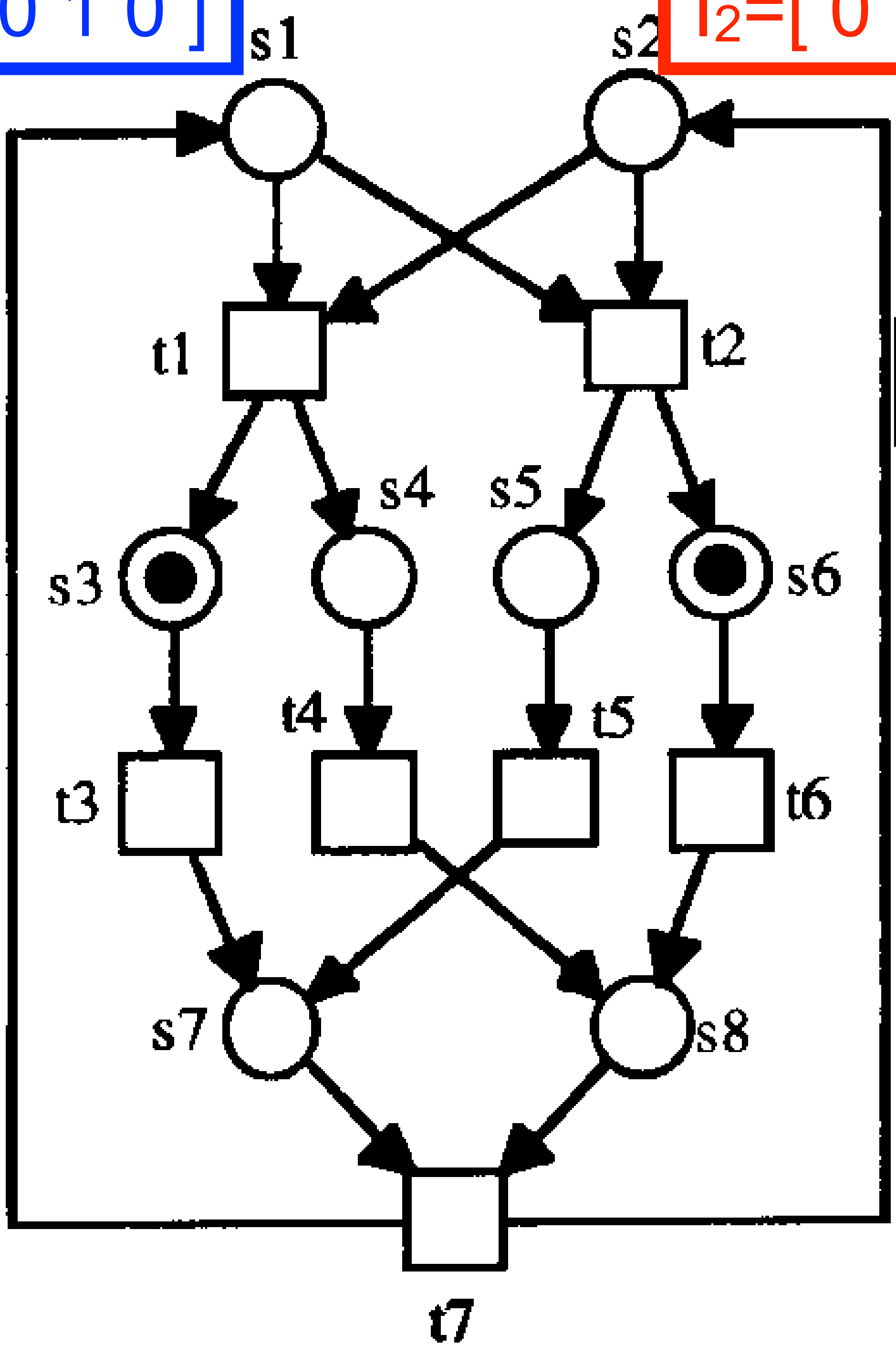

s1
s2
t1
t2
s4
s5
s3
s6
t4
t5
t3
t6
s7
s8
t7


s2
t1
t2
s4
s6
t4
t6
s8
t7

S-cover: example

$I_1=[\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ ]$

$I_2=[\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ ]$


s1
s2
t1
t2
s3
s4
s5
s6
t3
t4
t5
t6
s7
s8
t7

$I_1+I_2=[\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ ]$

positive S-invariant

S-coverability theorem

Theorem: If a free-choice system is live and bounded then it is S-coverable

(proof omitted)

Consequence:

free-choice + not S-coverable => not (live and bounded)

S-Coverability diagnosis

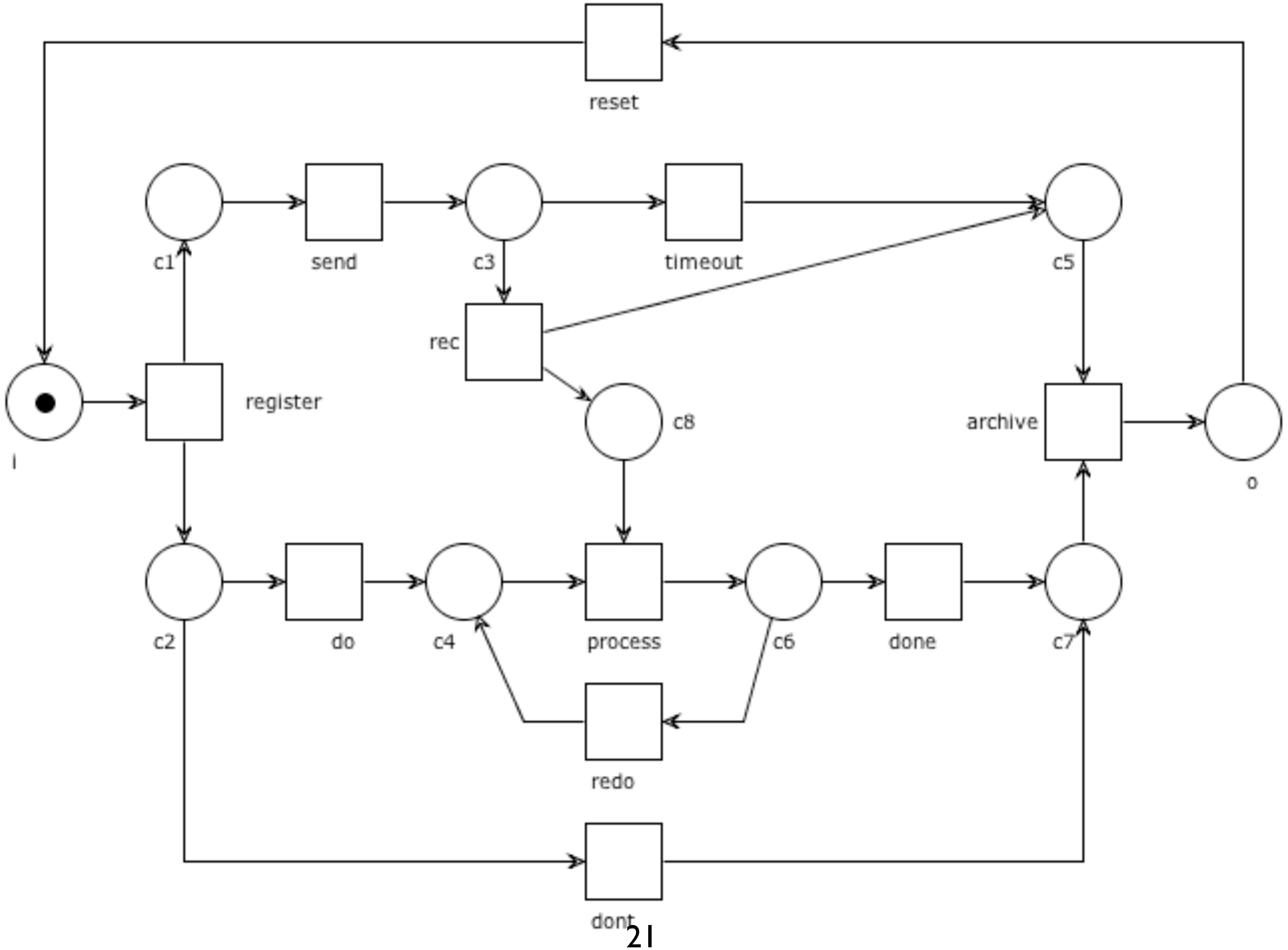
N is sound iff N* is live and bounded (Main Theorem)
N is free-choice iff N* is free-choice

If N* is free-choice, live and bounded
it must be S-coverable (S-coverability theorem)

Corollary: If N is sound and free-choice,
then N* must be S-coverable

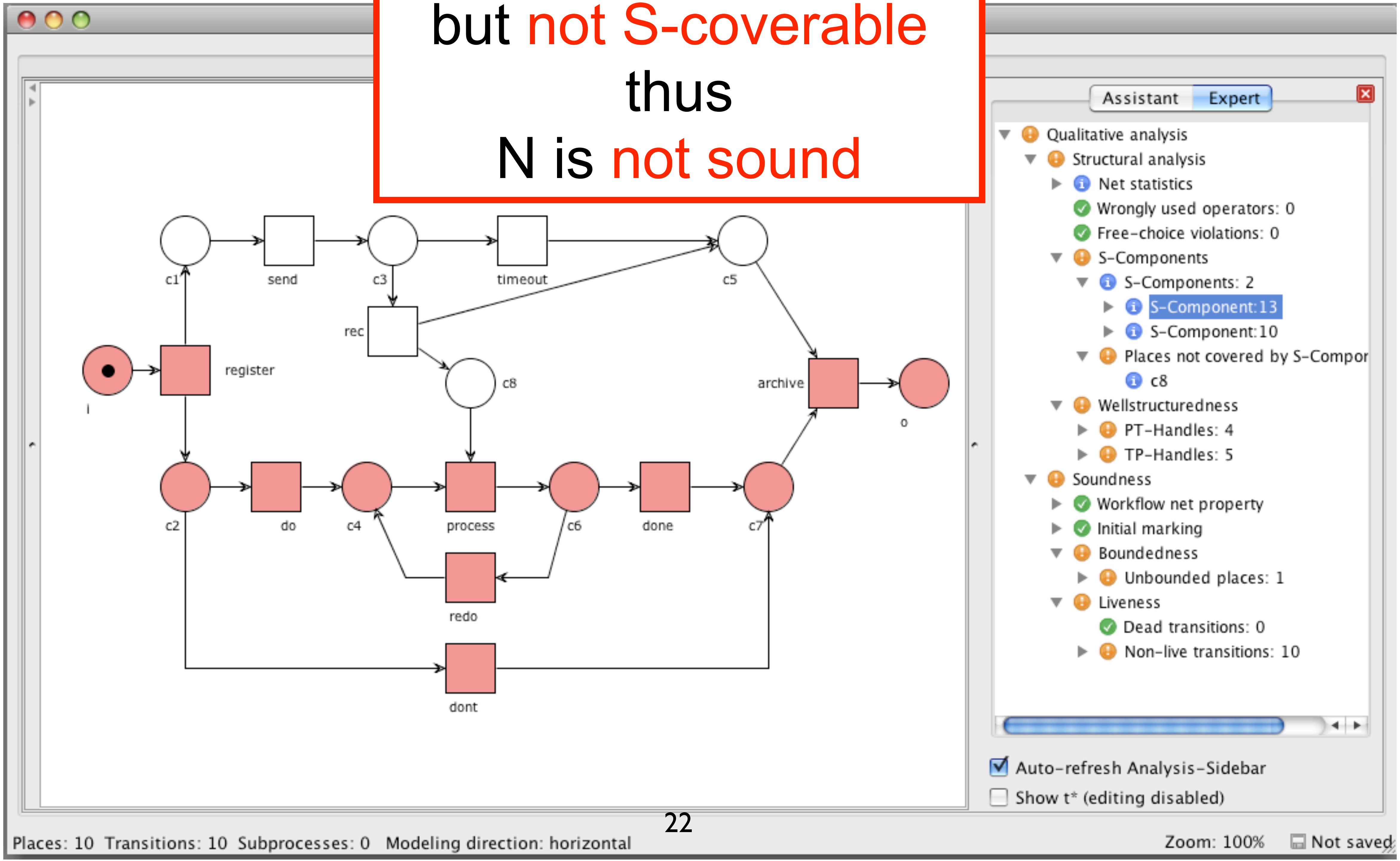
N free-choice + N* not S-coverable => N not sound

S-cover for N* ?


reset
c1
send
c3
timeout
c5
rec
i
register
c8
archive
o
c2
do
c4
process
c6
done
c7
redo
dont

WoPeD Diagnosis

N* is free-choice
but not S-coverable
thus
N is not sound


c1
send
c3
timeout
c5
rec
register
c8
archive
i
o
c2
do
c4
process
c6
done
c7
redo
dont
Assistant
Expert

- Qualitative analysis
  - Structural analysis
    - Net statistics
    - Wrongly used operators: 0
    - Free-choice violations: 0
    - S-Components
      - S-Components: 2
        - S-Component:13
        - S-Component:10
      - Places not covered by S-Compor
        - c8
    - Wellstructuredness
      - PT-Handles: 4
      - TP-Handles: 5
  - Soundness
    - Workflow net property
    - Initial marking
    - Boundedness
      - Unbounded places: 1
    - Liveness
      - Dead transitions: 0
      - Non-live transitions: 10

☑ Auto-refresh Analysis-Sidebar

☐ Show t* (editing disabled)

Places: 10 Transitions: 10 Subprocesses: 0 Modeling direction: horizontal

Zoom: 100% Not saved

Be careful

reset transition is implicit in WoPeD

WoPeD shows S-components for N*
(not for N)

T-Coverability

Rank Theorem (main result, proof omitted)

Theorem:

A free-choice system (P,T,F,M_0) is live and bounded

iff

1. it has at least one place and one transition
2. it is connected
3. M_0 marks every proper siphon
4. it has a positive S-invariant
5. it has a positive T-invariant
6. rank(N) = $|C_N|$ - 1

(where C_N is the set of clusters)

A technique to find a positive T-invariant

Decompose the net N in suitable T-nets
so that any transition of N belongs to some T-net
(the same transition can appear in more T-nets)

Each T-net induces a uniform T-invariant

A positive T-invariant is obtained
as the sum of the T-invariants of each subnet

T-component

take a set of nodes

Definition: Let $N = (P, T, F)$ and $\emptyset \subset X \subseteq P \cup T$
Let $N' = (P \cap X, T \cap X, F \cap (X \times X))$ be a subnet of N.
N' is a **T-component** if

forget the arcs to other nodes

1. it is a strongly connected T-net
2. for every transition $t \in X \cap T$, we have $\bullet t \cup t \bullet \subseteq X$

if a transition is selected
then all the attached places must be selected

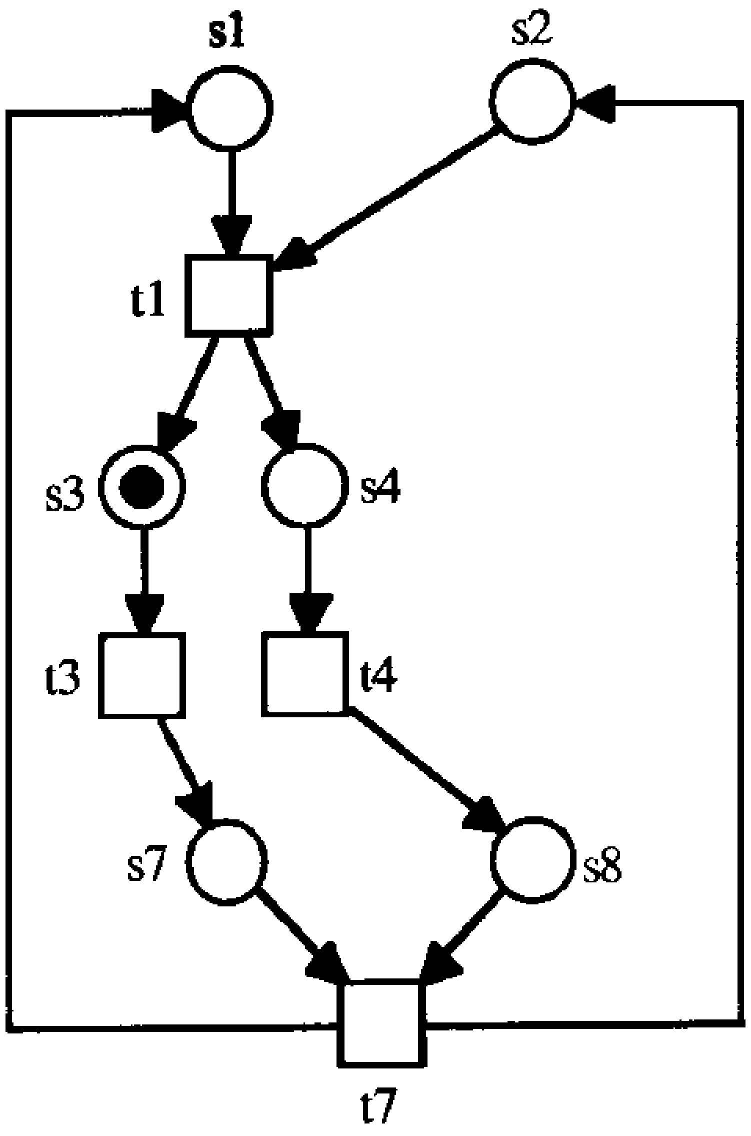
T-cover

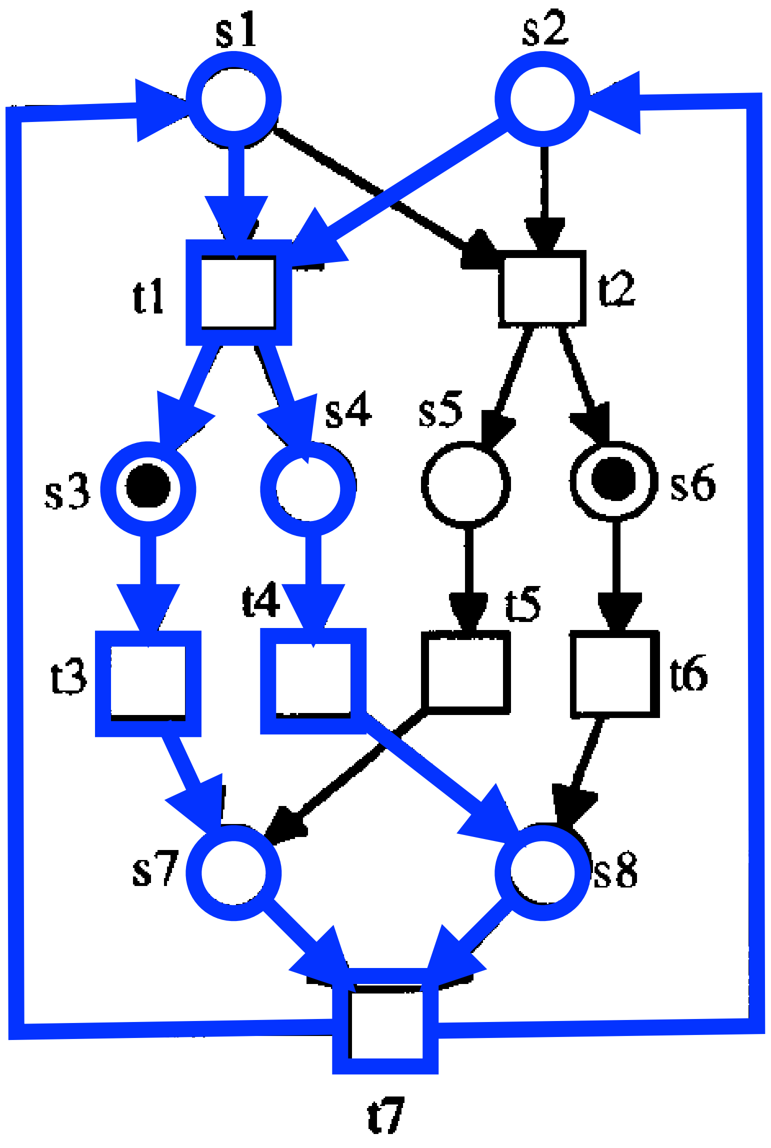
Definition: a **T-cover** of a net N
is a set **C** of T-components of N such that
every transition t of N is in one or more T-components in **C**

N is **T-coverable** if it has a T-cover

if a transition is selected
then all the attached places must be selected

T-cover: example

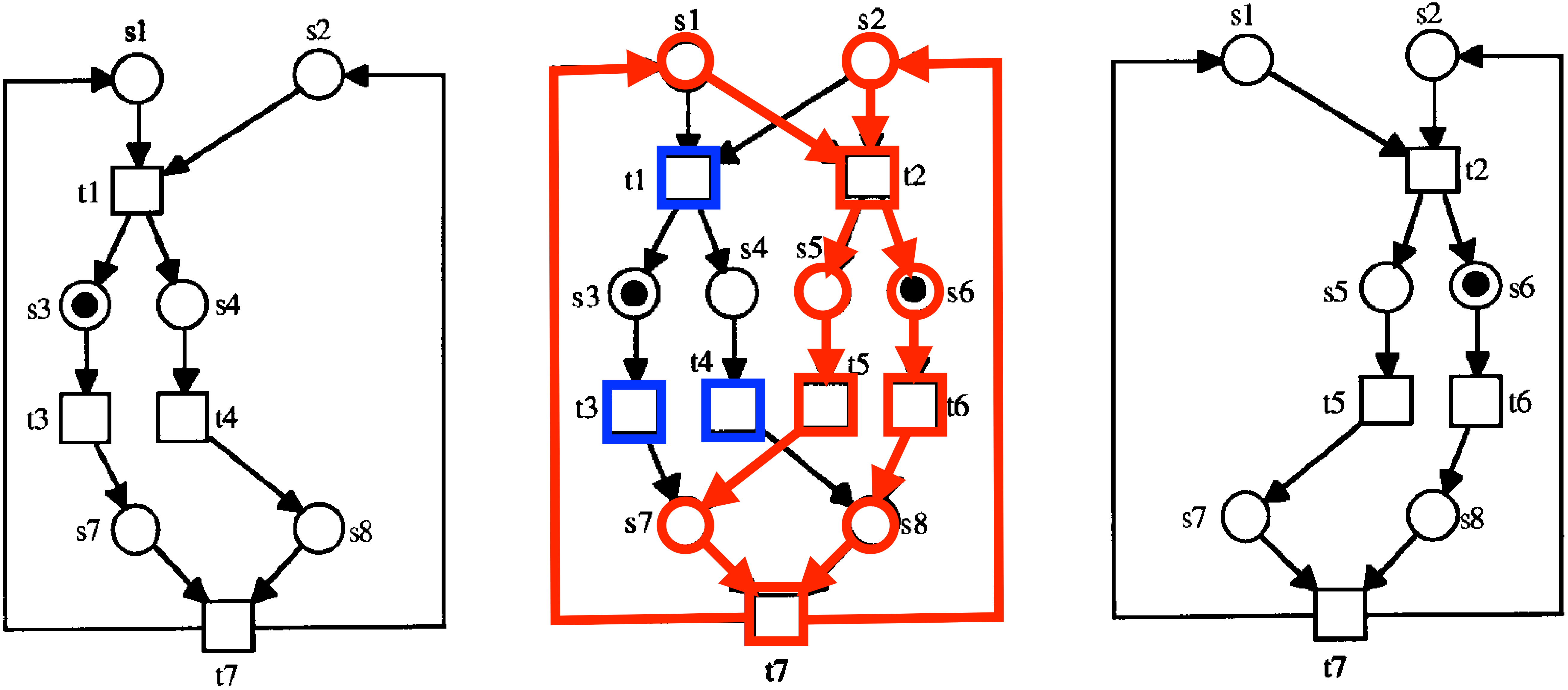

s1
s2
t1
s3
s4
t3
t4
s7
s8
t7


s1
s2
t1
t2
s4
s5
s3
s6
t4
t5
t3
t6
s7
s8
t7

T-component

if a transition is selected
then all the attached places must be selected

T-cover: example

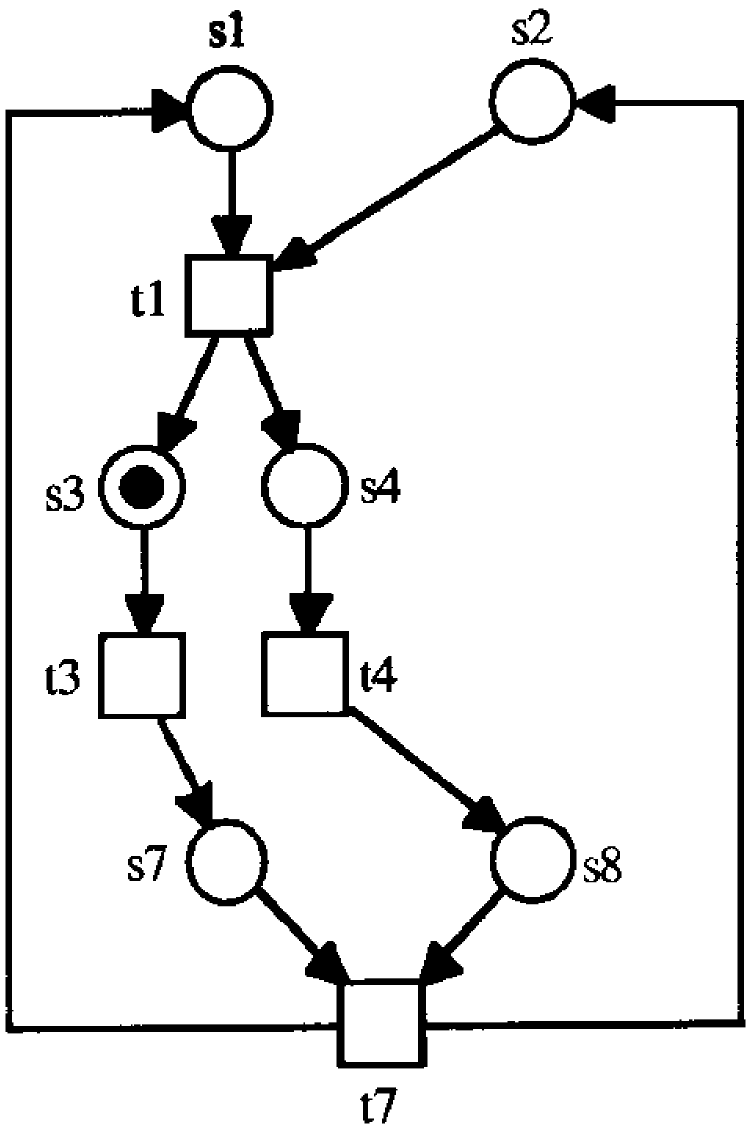

s1
s2
t1
s3
s4
t3
t4
s7
s8
t7
s1
s2
t1
t2
s3
s4
s5
s6
t3
t4
t5
t6
s7
s8
t7
s1
s2
t2
s5
s6
t5
t6
s7
s8
t7

T-component

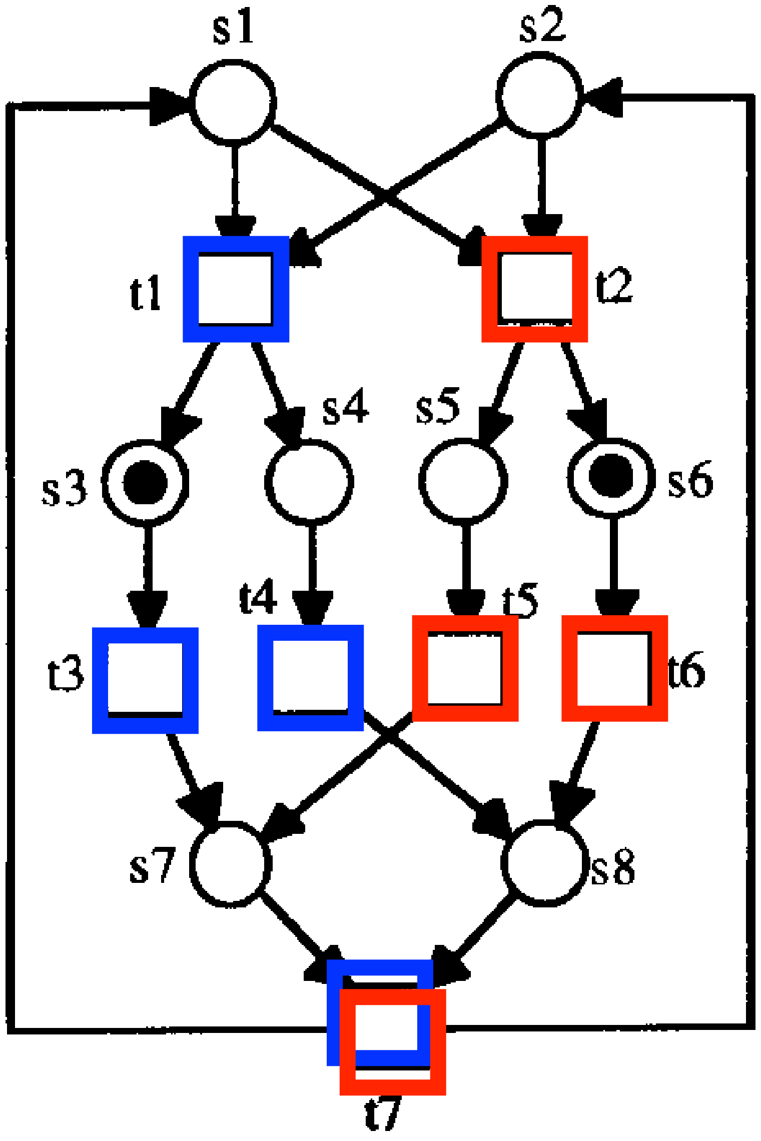
T-component

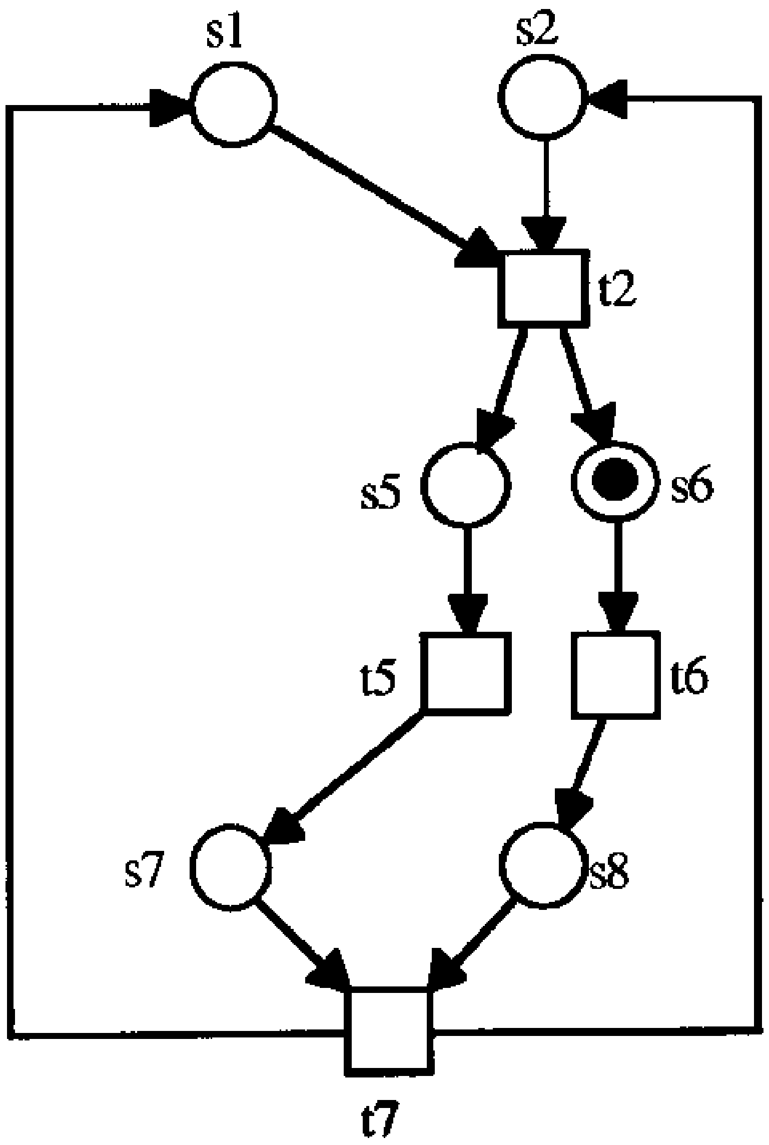
T-cover: example

covered by T-components


s1
s2
t1
s3
s4
t3
t4
s7
s8
t7

T-component

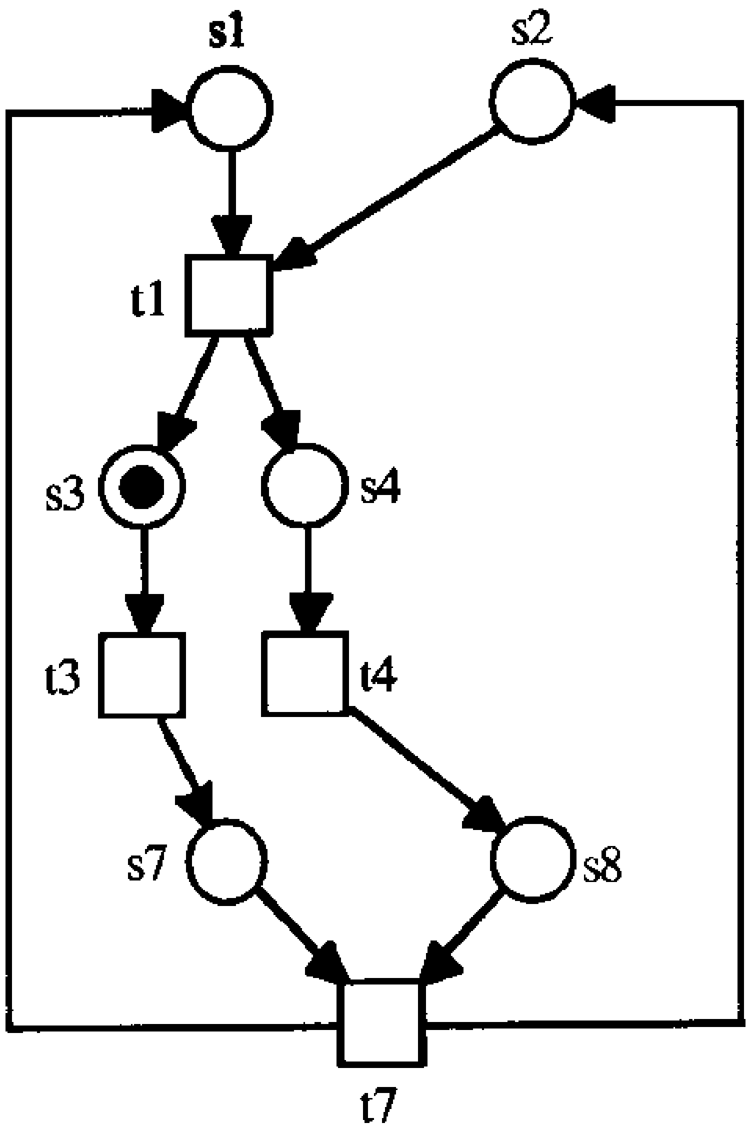

s1
s2
t1
t2
s3
s4
s5
s6
t3
t4
t5
t6
s7
s8
t7


s1
s2
t2
s5
s6
t5
t6
s7
s8
t7

T-component

T-cover: example

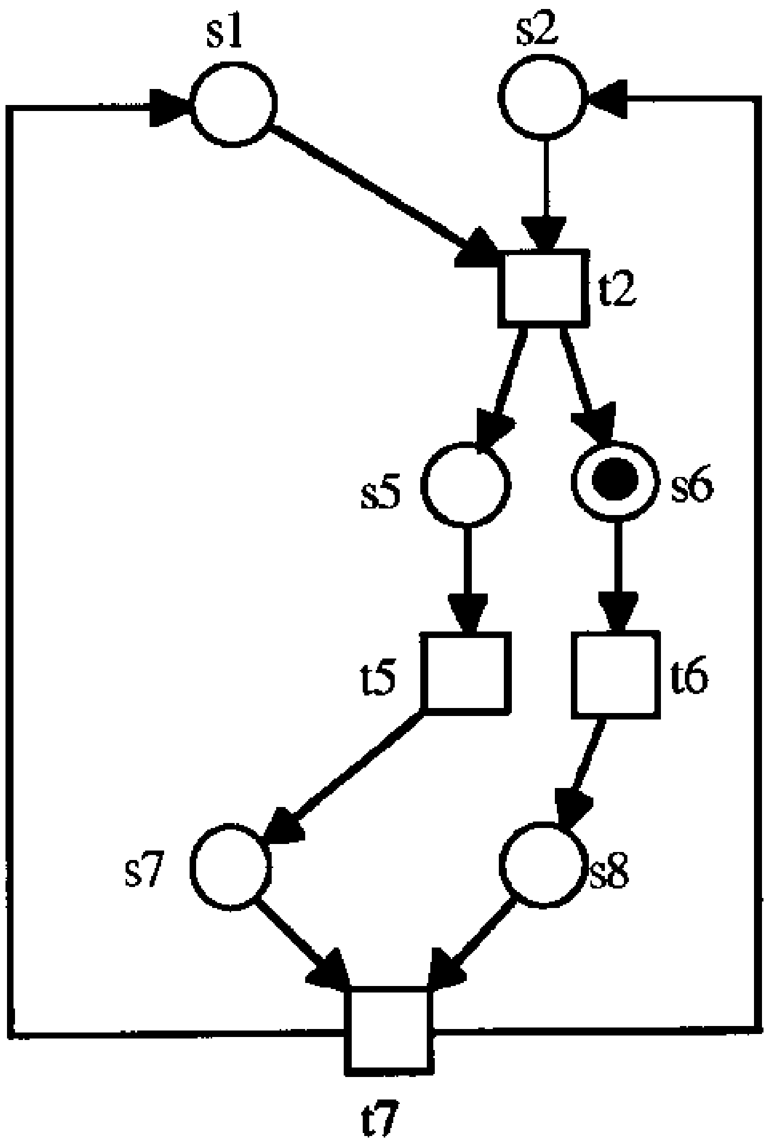
J_1=[1 1 1 1]


s1
s2
t1
s3
s4
t3
t4
s7
s8
t7

T-component

T-cover

J_2=[1 1 1 1]

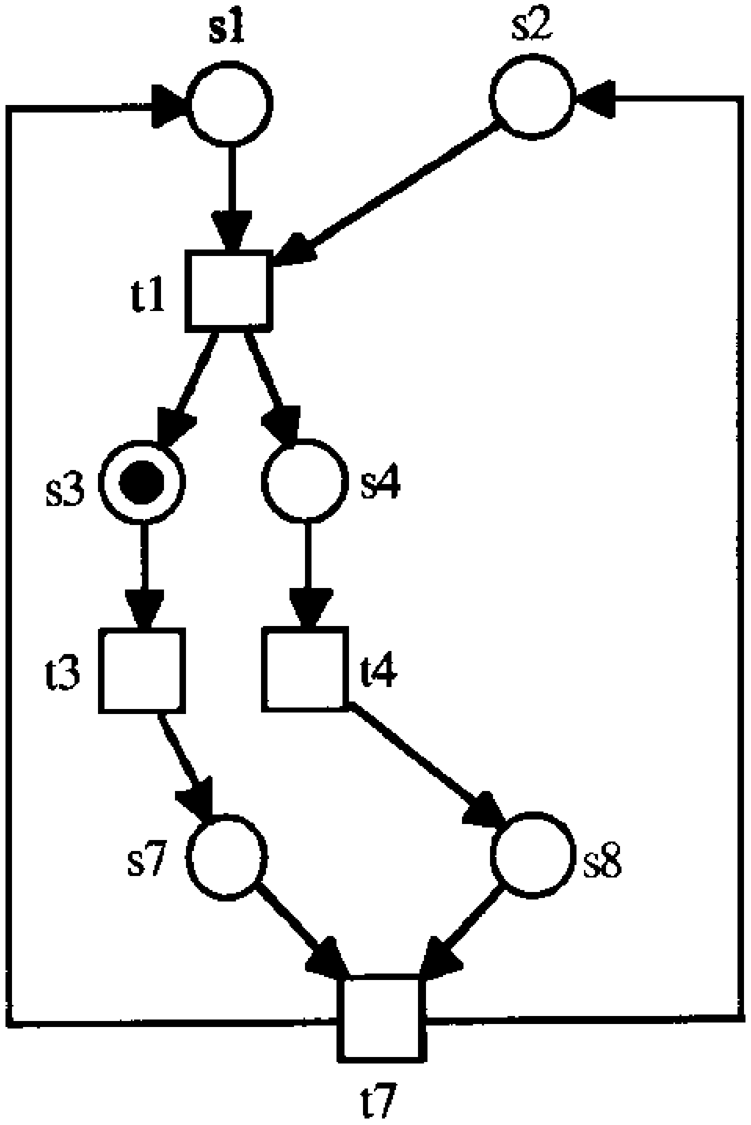

s1
s2
t2
s5
s6
t5
t6
s7
s8
t7

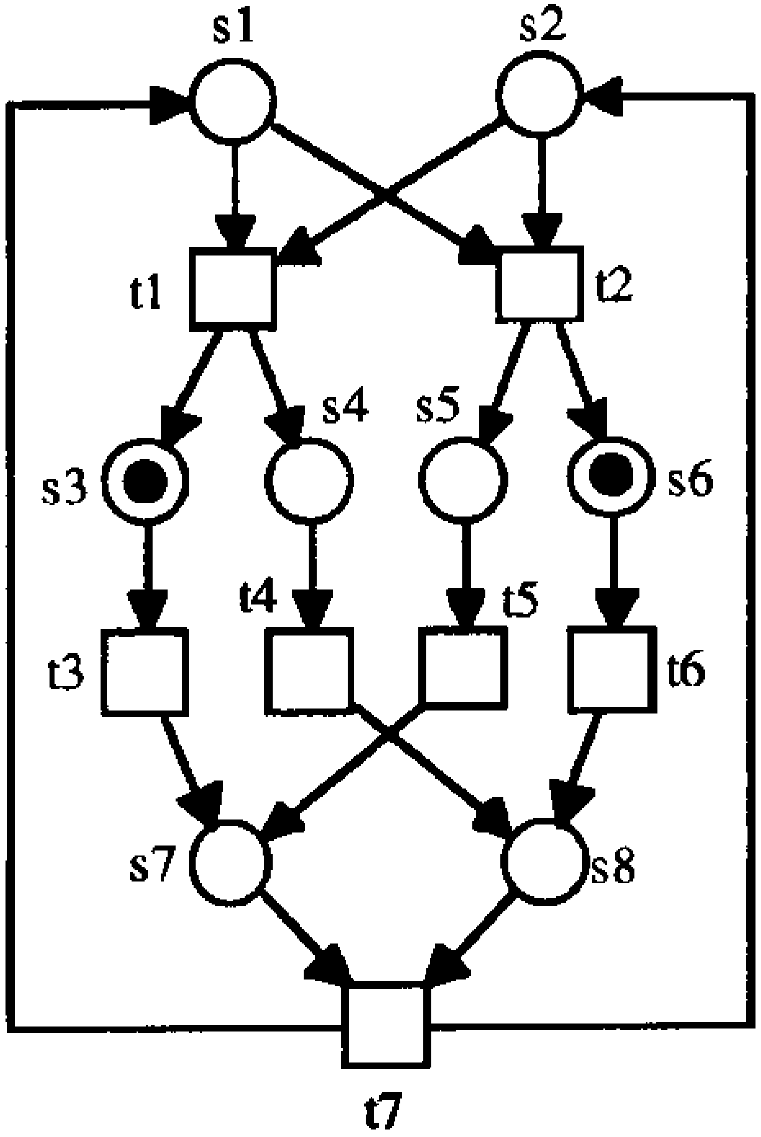
T-component

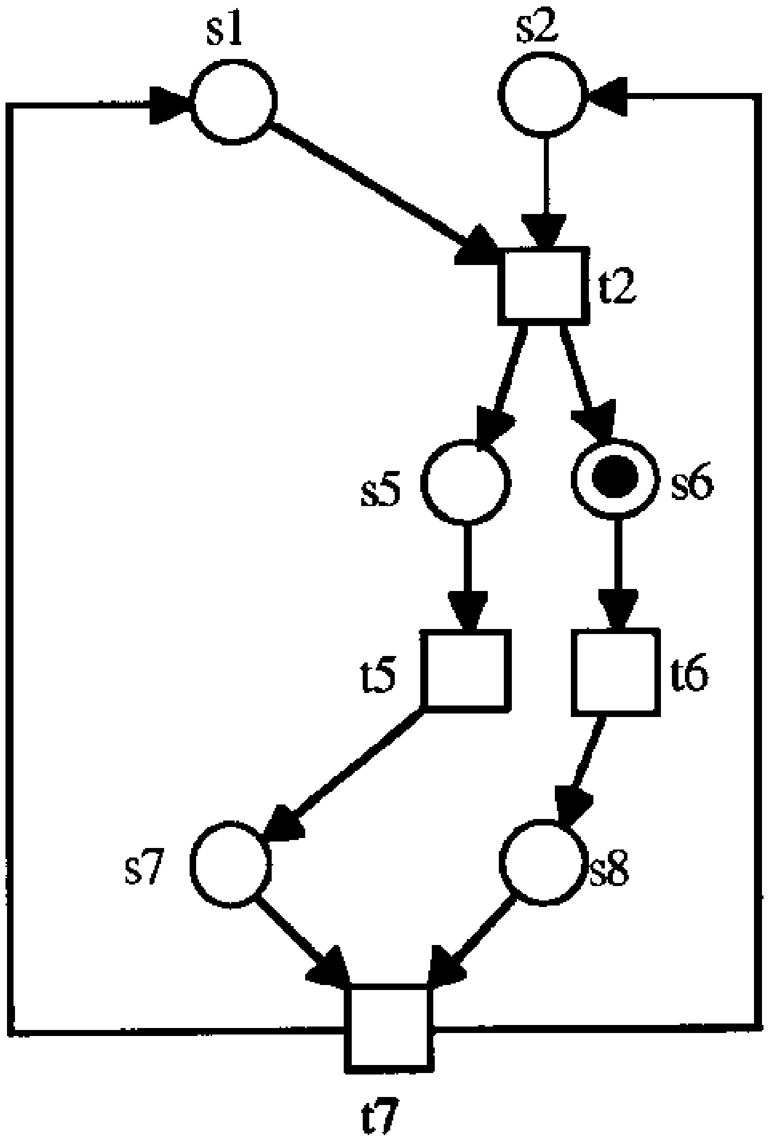
T-cover: example

$J_1=[\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ ]$

$J_2=[\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ ]$


s1
s2
t1
s3
s4
t3
t4
s7
s8
t7


s1
s2
t1
t2
s3
s4
s5
s6
t3
t4
t5
t6
s7
s8
t7


s1
s2
t2
s5
s6
t5
t6
s7
s8
t7

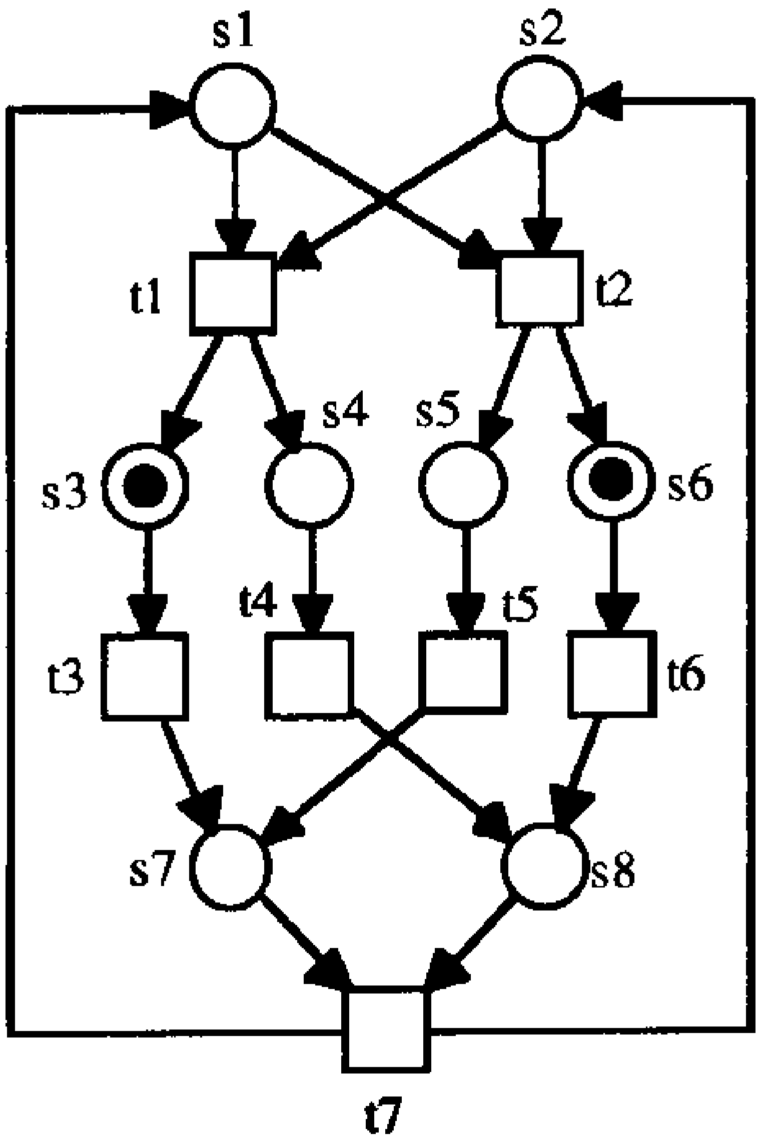
T-component

T-component

T-cover: example

J_1=[1 0 1 1 0 0 1]

J_2=[0 1 0 0 1 1 1]


s1
s2
t1
t2
s3
s4
s5
s6
t3
t4
t5
t6
s7
s8
t7

J_1+J_2=[1 1 1 1 1 1 2]

positive T-invariant

T-coverability theorem

Theorem: If a free-choice system is live and bounded then it is T-coverable
(proof omitted)

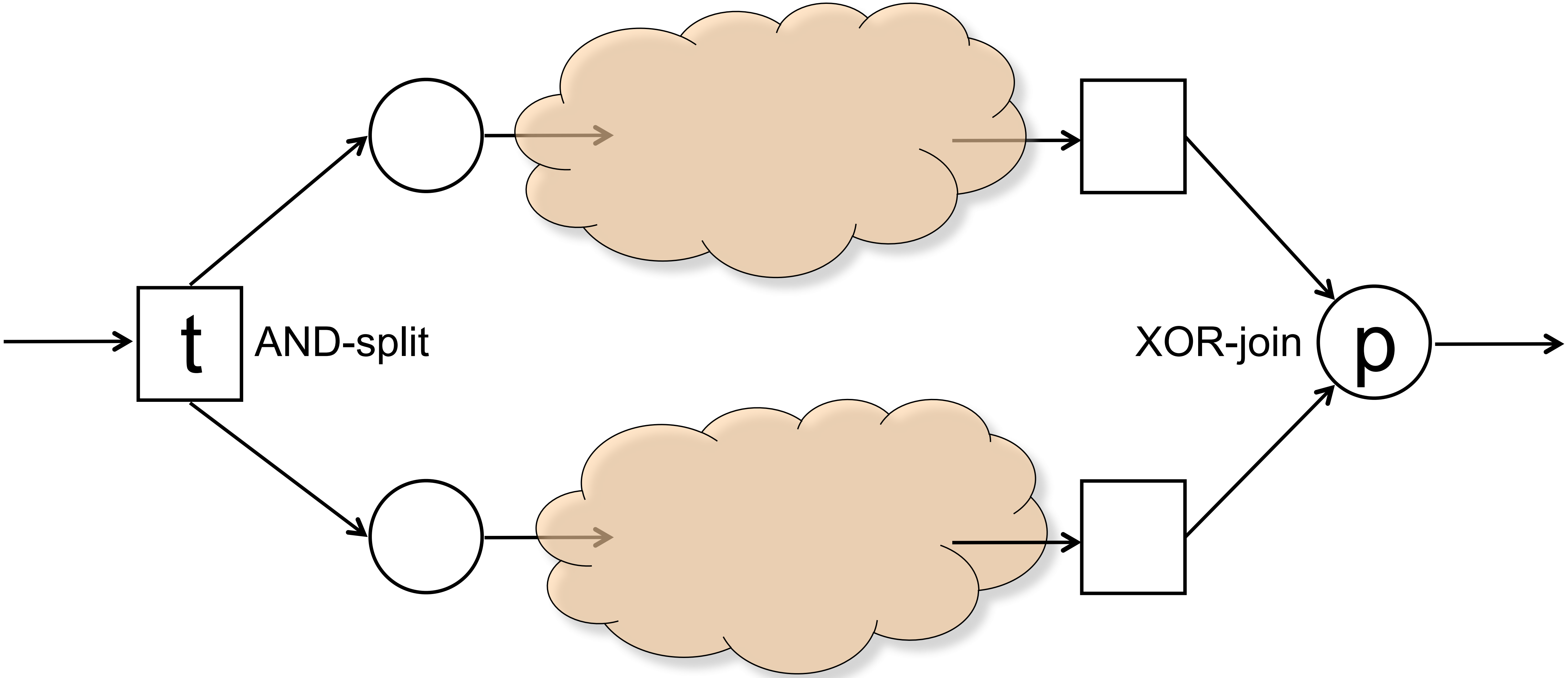
Consequence:
free-choice + not T-coverable => not (live and bounded)

Well-structuredness (PT/TP-handles)

TP-handles

Two parallel flows initiated by an AND-split should not be joined by a XOR-join

(multiple tokens can appear in the same place)

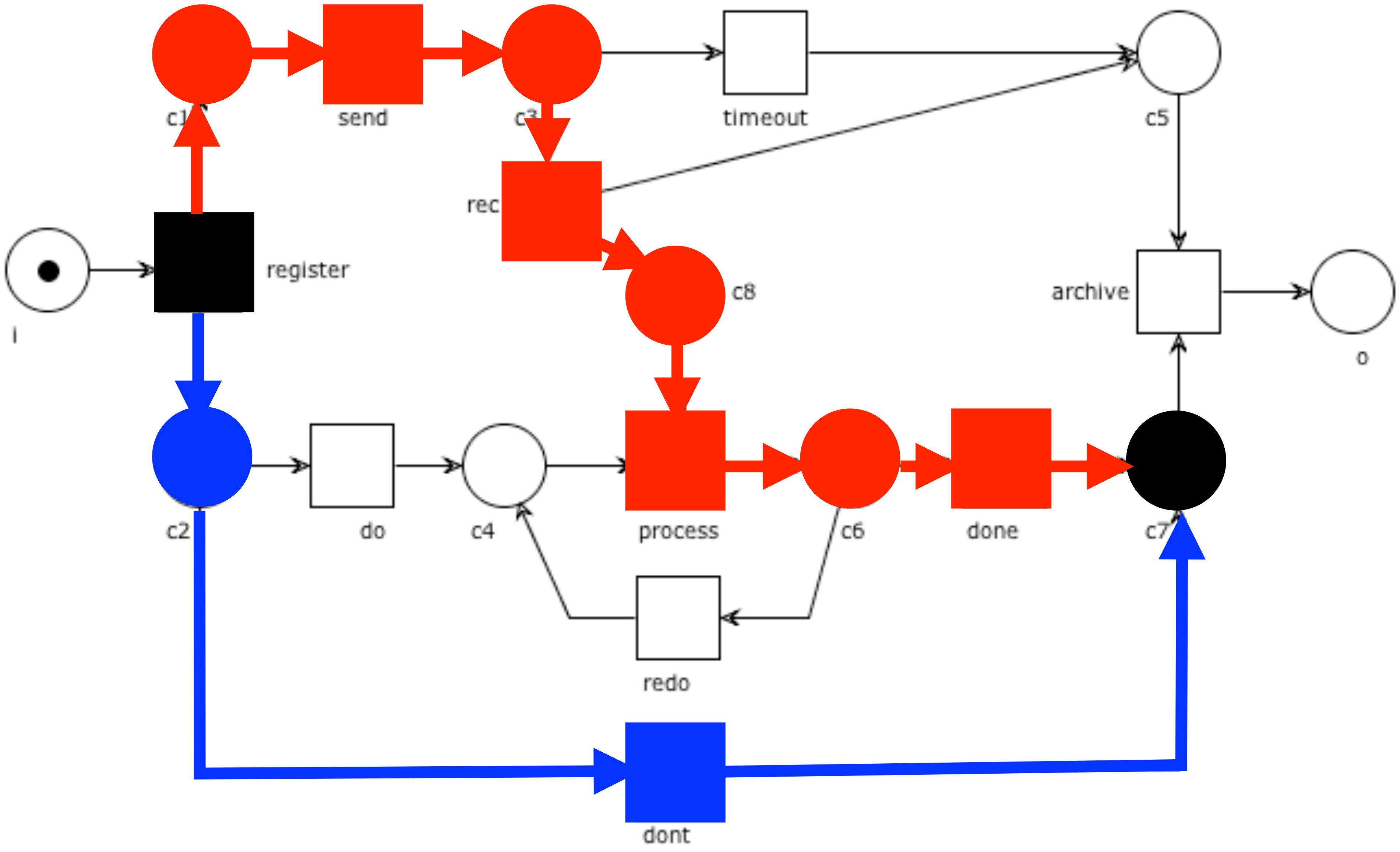

t
AND-split
XOR-join
p

TP-handles

Definition:

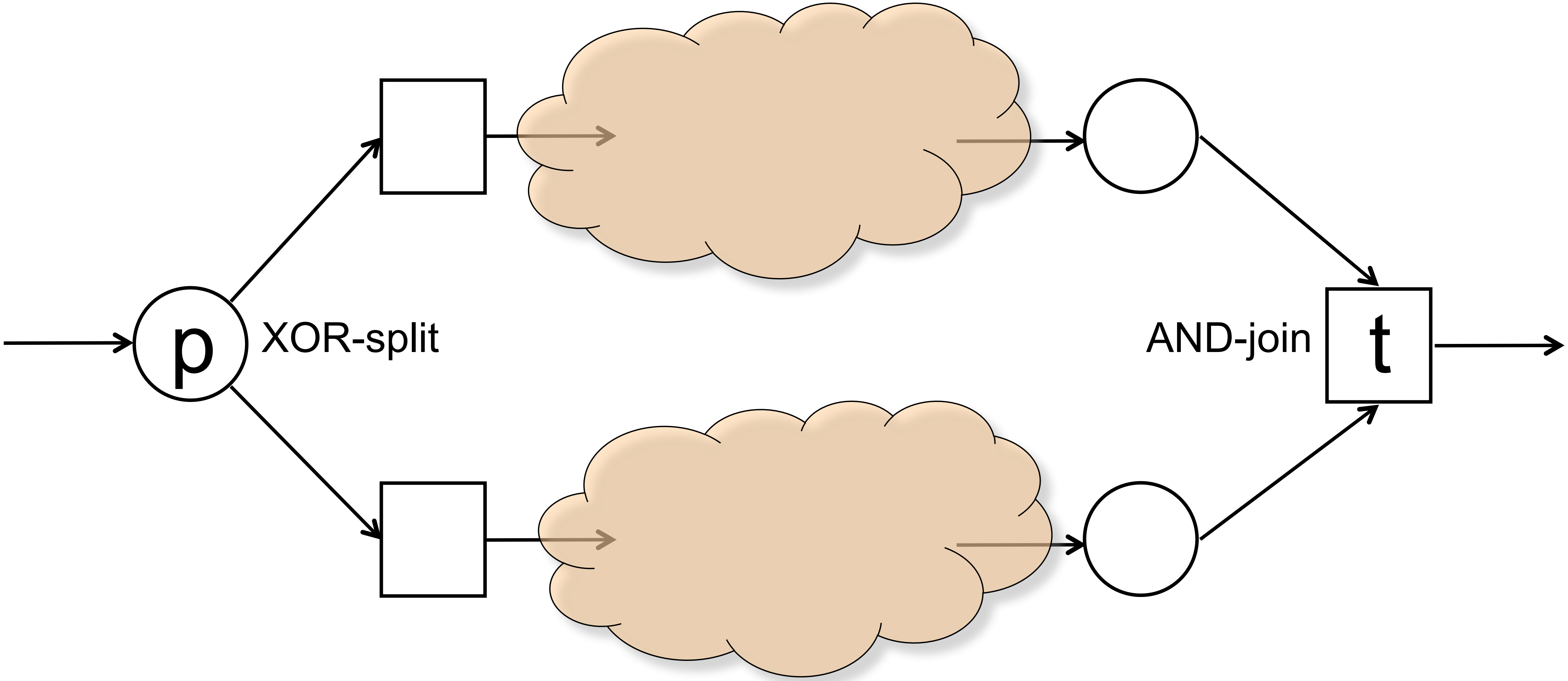
A transition t and a place p form a **TP-handle**
if there are
two distinct elementary paths c_1 and c_2 from t to p
such that the only nodes they have in common are t, p

Example: TP-handle


c1
send
c3
timeout
c5
rec
i
register
c8
archive
o
c2
do
c4
process
c6
done
c7
redo
dont

PT-handles

Two alternative flows created via a XOR-split should not be synchronized by an AND-join **(the net could deadlock)**

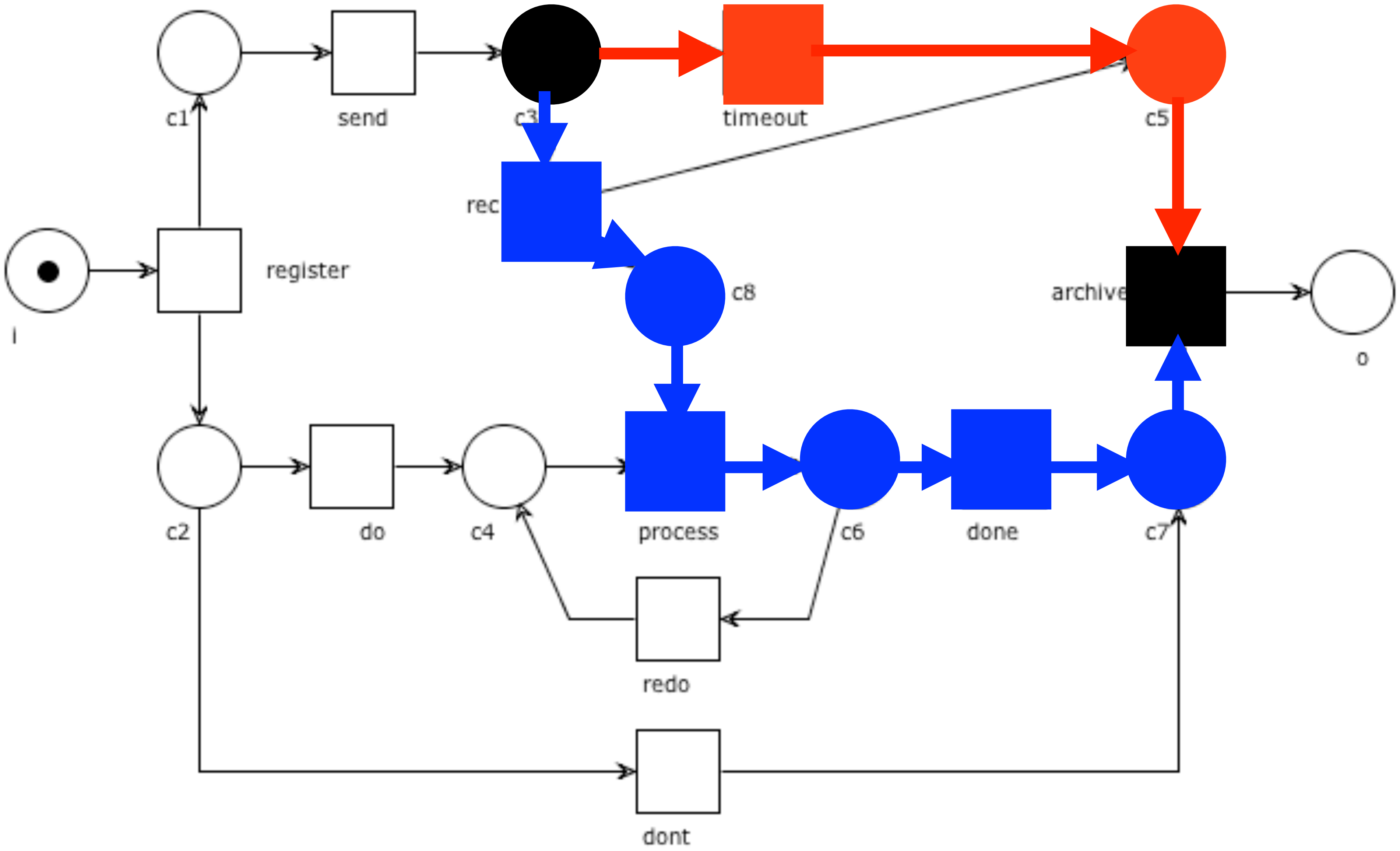

p
XOR-split
AND-join
t

PT-handles

Definition:

A place p and a transition t form a **PT-handle**
if there are
two distinct elementary paths c_1 and c_2 from p to t
such that the only nodes they have in common are p, t

Example: PT-handle


c1
send
c3
timeout
c5
rec
i
register
c8
archive
o
c2
do
c4
process
c6
done
c7
redo
dont

Well-Structured Nets

Definition: A net is **well-handled** if
it has neither TP-handles nor PT-handles

Definition: A workflow net N is **well-structured** if
N* is well-handled

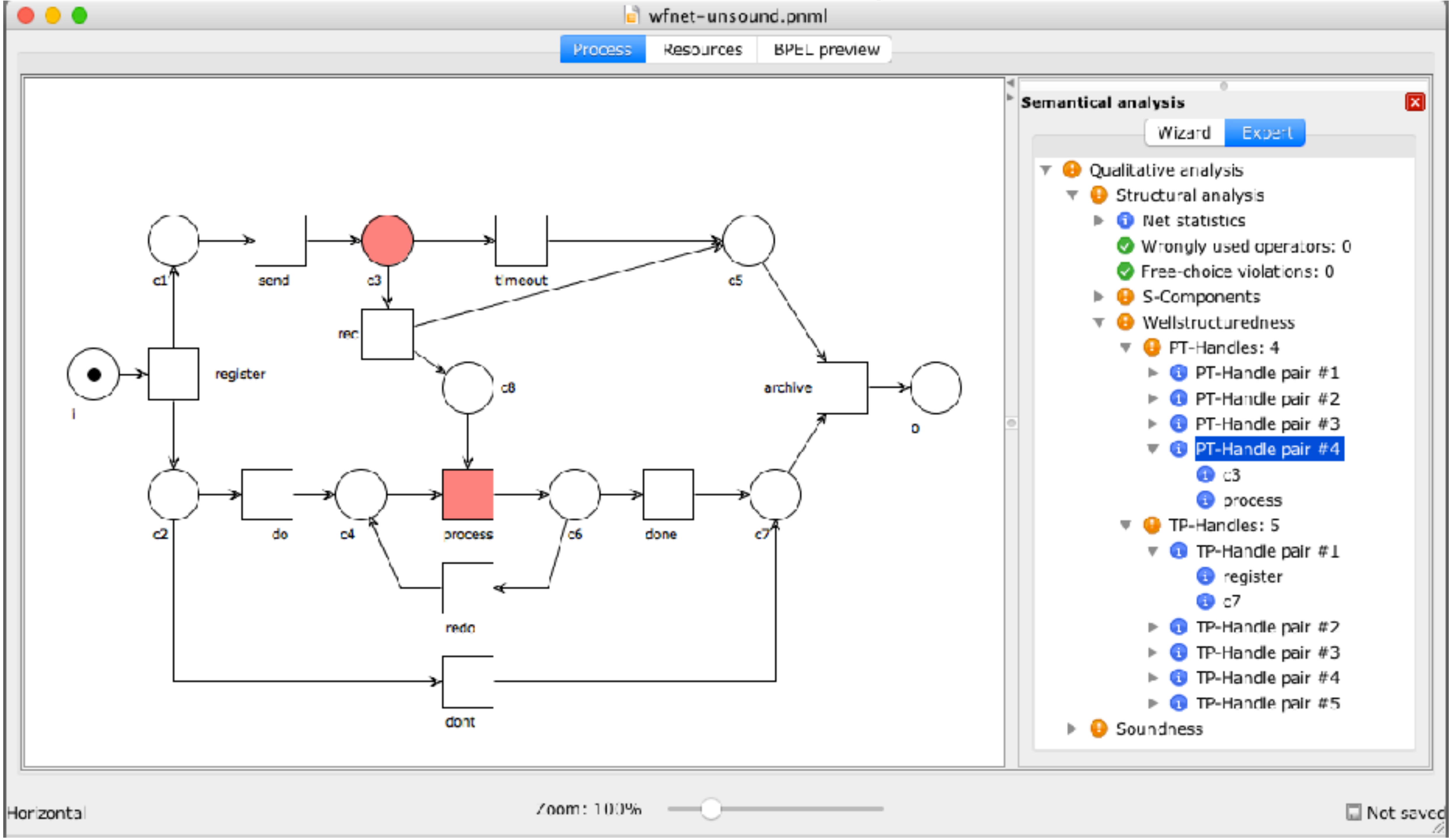
Be careful

N well-structured = N* well-handled

reset transition is implicit in WoPeD

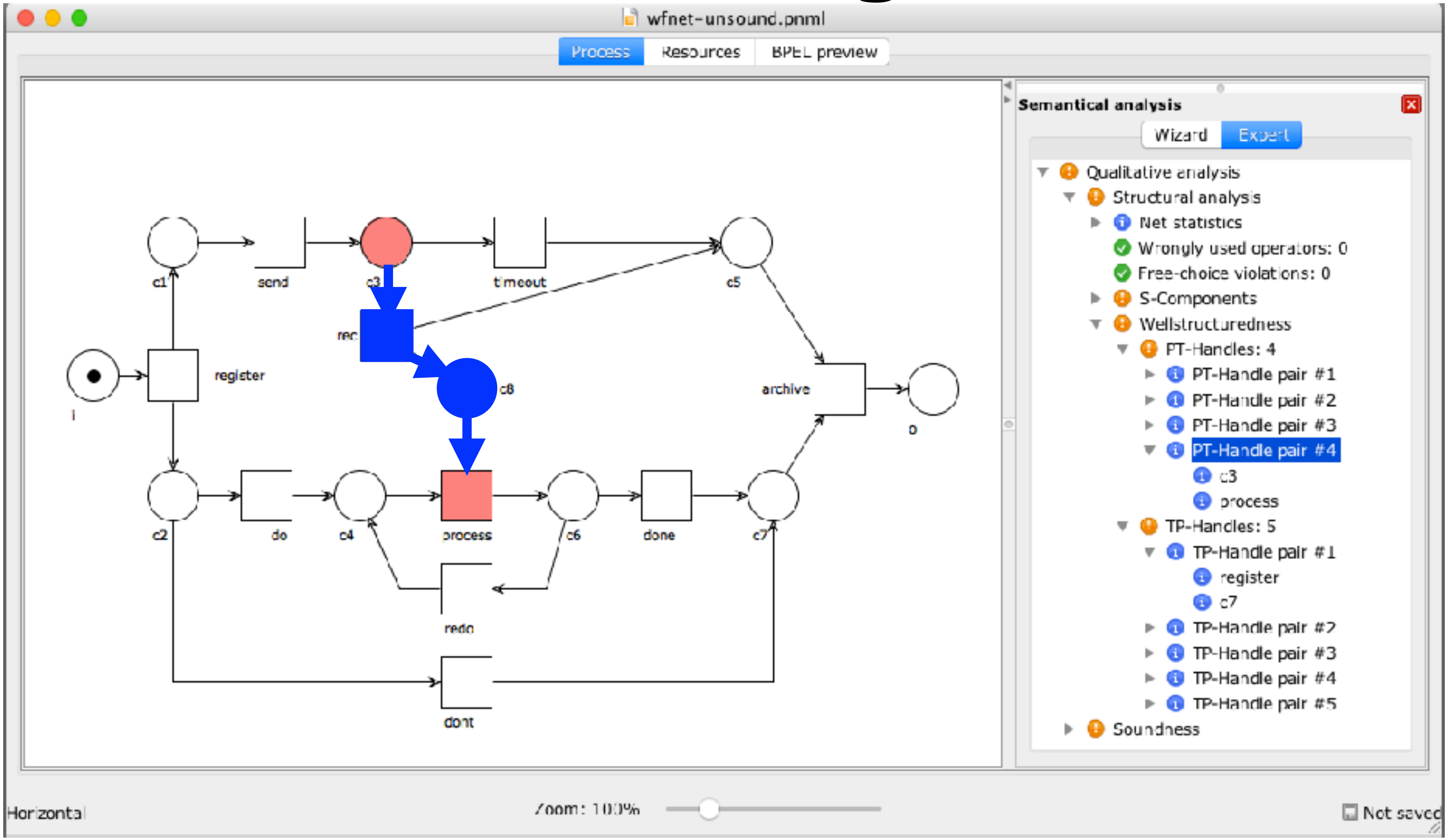
WoPeD marks PT/TP-handles over N*
(not over N)

WoPeD Diagnosis


wfnet-unsound.pnml
Process
Resources
BPEL preview
c1
send
c3
timeout
c5
rec
register
i
c8
archive
o
c2
do
c4
process
c6
done
c7
redo
dont
Semantical analysis
Wizard
Expert
Horizontal
Zoom: 100%
Not saved

- Qualitative analysis
  - Structural analysis
    - Net statistics
    - Wrongly used operators: 0
    - Free-choice violations: 0
    - S-Components
    - Wellstructuredness
      - PT-Handles: 4
        - PT-Handle pair #1
        - PT-Handle pair #2
        - PT-Handle pair #3
        - PT-Handle pair #4
          - c3
          - process
      - TP-Handles: 5
        - TP-Handle pair #1
          - register
          - c7
        - TP-Handle pair #2
        - TP-Handle pair #3
        - TP-Handle pair #4
        - TP-Handle pair #5
  - Soundness

WoPeD Diagnosis


wfnet-unsound.pnml
Process
Resources
BPEL preview
c1
send
c3
timeout
c5
rec
register
c8
archive
i
o
c2
do
c4
process
c6
done
c7
redo
dont
Wizard
Expert

Semantical analysis

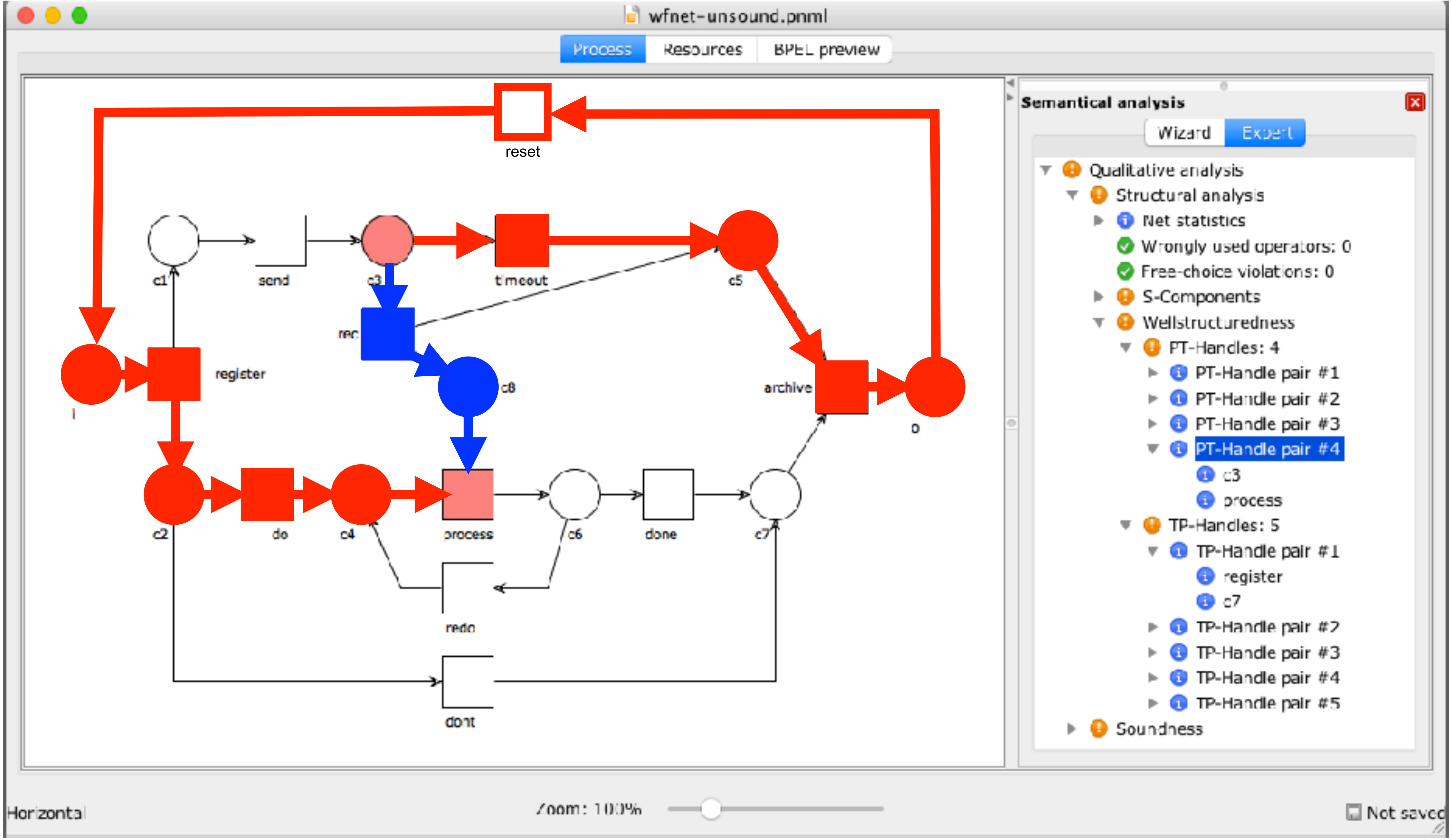
- Qualitative analysis
  - Structural analysis
    - Net statistics
    - Wrongly used operators: 0
    - Free-choice violations: 0
    - S-Components
    - Wellstructuredness
      - PT-Handles: 4
        - PT-Handle pair #1
        - PT-Handle pair #2
        - PT-Handle pair #3
        - PT-Handle pair #4
          - c3
          - process
      - TP-Handles: 5
        - TP-Handle pair #1
          - register
          - c7
        - TP-Handle pair #2
        - TP-Handle pair #3
        - TP-Handle pair #4
        - TP-Handle pair #5
  - Soundness

Horizontal

Zoom: 100%

☐ Not saved

WoPeD Diagnosis


wfnet-unsound.pnml
Process
Resources
BPEL preview
reset
c1
send
c3
timeout
c5
rec
c8
register
i
archive
o
c2
do
c4
process
c6
done
c7
redo
dont
Semantical analysis
Wizard
Expert
Horizontal
Zoom: 100%
Not saved

- Qualitative analysis
  - Structural analysis
    - Net statistics
    - Wrongly used operators: 0
    - Free-choice violations: 0
    - S-Components
    - Wellstructuredness
      - PT-Handles: 4
        - PT-Handle pair #1
        - PT-Handle pair #2
        - PT-Handle pair #3
        - PT-Handle pair #4
          - c3
          - process
      - TP-Handles: 5
        - TP-Handle pair #1
          - register
          - c7
        - TP-Handle pair #2
        - TP-Handle pair #3
        - TP-Handle pair #4
        - TP-Handle pair #5
  - Soundness

Well-structuredness, S-coverability and Soundness

Theorem: If N is sound and well-structured,
then N* is S-coverable
(proof omitted)

Consequence:
N well-structured + N* not S-coverable => N not sound

Error sequences

Woflan

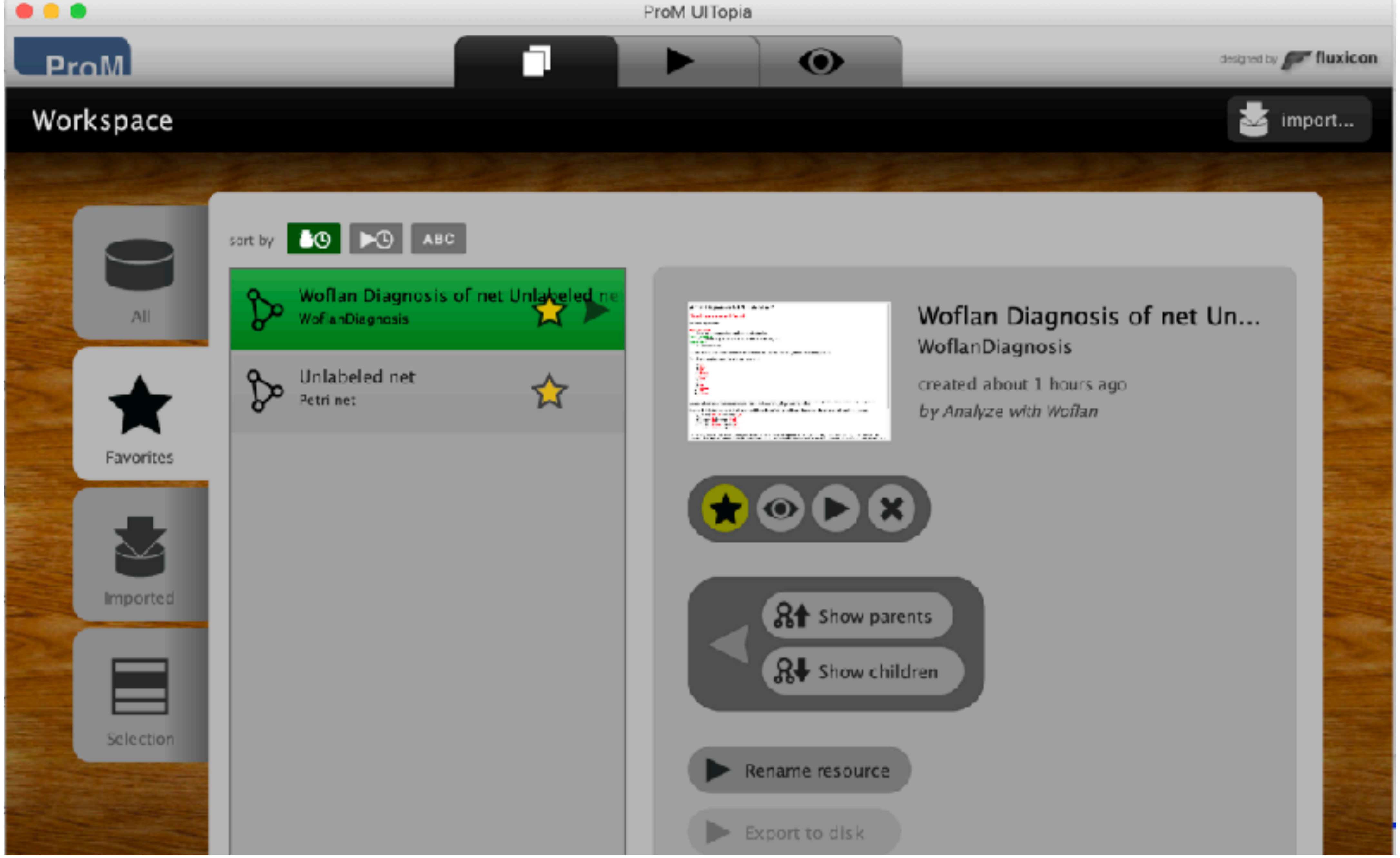
http://www.win.tue.nl/woflan/

WOrkFLow ANalyzer
(Microsoft Windows only)

Woflan tells us if N is a sound workflow net
(Is N a workflow net? Is N* bounded? Is N* live?)
if not, provides some diagnostic information

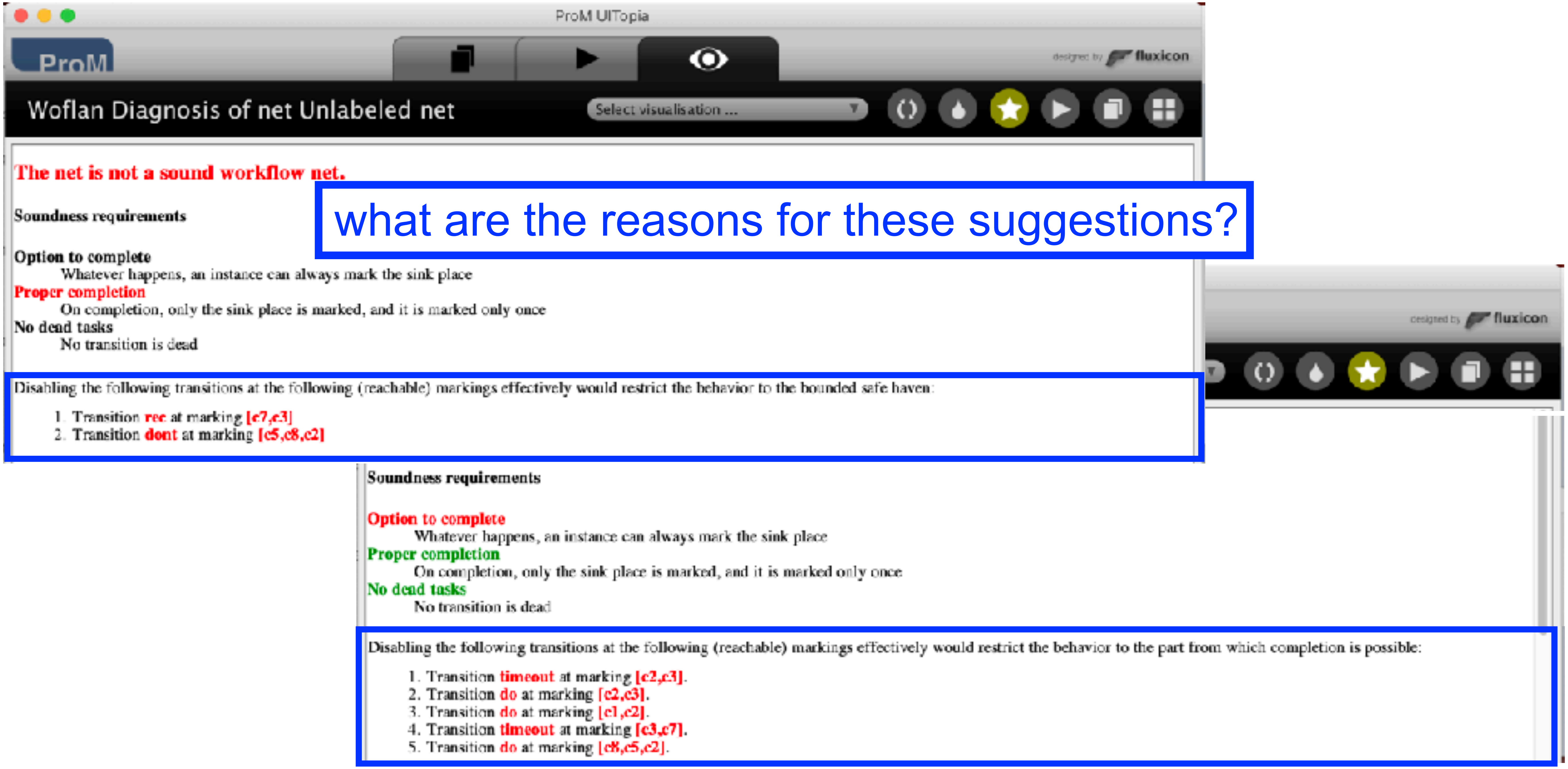
Woflan now a ProM plugin

http://promtools.org/



ProM UITopia
ProM
designed by fluxicon
Workspace
import...
sort by
ABC
All
Favorites
Imported
Selection
Woflan Diagnosis of net Unlabeled ne
WoflanDiagnosis
Unlabeled net
Petri net
Woflan Diagnosis of net Un...
WoflanDiagnosis
created about 1 hours ago
by Analyze with Woflan
Show parents
Show children
Rename resource
Export to disk

Woflan (in ProM)



what are the reasons for these suggestions?

The net is not a sound workflow net.

Soundness requirements

Option to complete
Whatever happens, an instance can always mark the sink place
Proper completion
On completion, only the sink place is marked, and it is marked only once
No dead tasks
No transition is dead

Disabling the following transitions at the following (reachable) markings effectively would restrict the behavior to the bounded safe haven:

1. Transition **rec** at marking **[c7,c3]**
2. Transition **dont** at marking **[c5,c8,c2]**

Soundness requirements

Option to complete
Whatever happens, an instance can always mark the sink place
Proper completion
On completion, only the sink place is marked, and it is marked only once
No dead tasks
No transition is dead

Disabling the following transitions at the following (reachable) markings effectively would restrict the behavior to the part from which completion is possible:

1. Transition **timeout** at marking **[c2,c3]**.
2. Transition **do** at marking **[c2,c3]**.
3. Transition **do** at marking **[c1,c2]**.
4. Transition **timeout** at marking **[c3,c7]**.
5. Transition **do** at marking **[c8,c5,c2]**.

Diagnostic information

The sets of:
unbounded places of N*
dead transitions of N*
non-live transitions of N*

may provide useful information for
the diagnosis of behavioural errors

Unfortunately, this information is not always sufficient
to determine the exact cause of the error

Behavioural error sequences help us to locate problems

Error sequences

Rationale:
We want to find firing sequences such that:

1. every continuation of such sequences will lead to an error

2. they are as short as possible
(none of their prefixes satisfies the above property)

Informally:
error sequences are scenarios that capture
the essence of errors made in the workflow design
(violate “option to complete” or “proper completion”)

Error sequences: Non-live sequences

Non-Live sequences: informally

A **non-live sequence** is a
firing sequence as short as possible
such that **completion of the case is no longer possible**

i.e. a witness for transition reset being non-live in N*

Non-Live sequences: fundamental property

Let N be such that:
N* is bounded
N (or equivalently N*) has no dead task

Then, N* is live
iff
N has no non-live sequences

Non-Live sequences: graphically

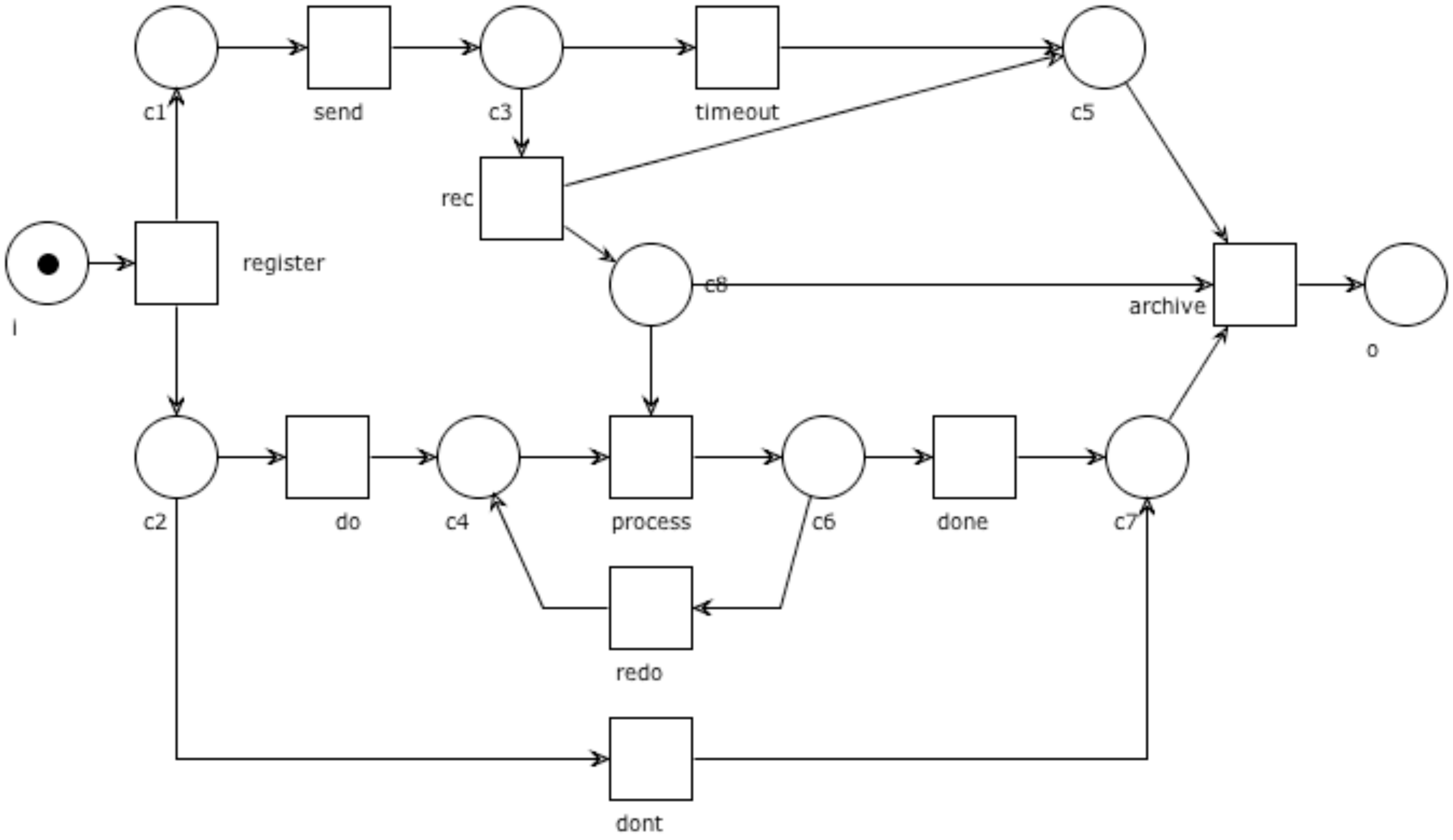
The analysis is possible in bounded systems only

Compute the RG of N*
Color in **red** all nodes from which there is **no path** to o

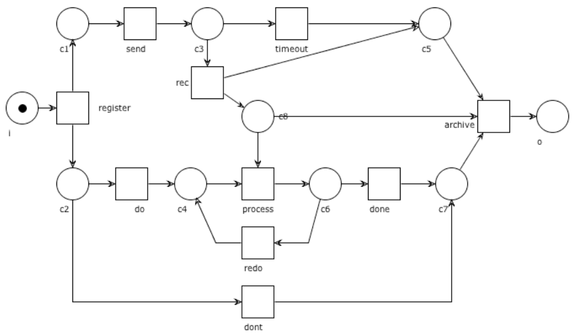
Color in **green** all nodes from which **all paths** lead to o

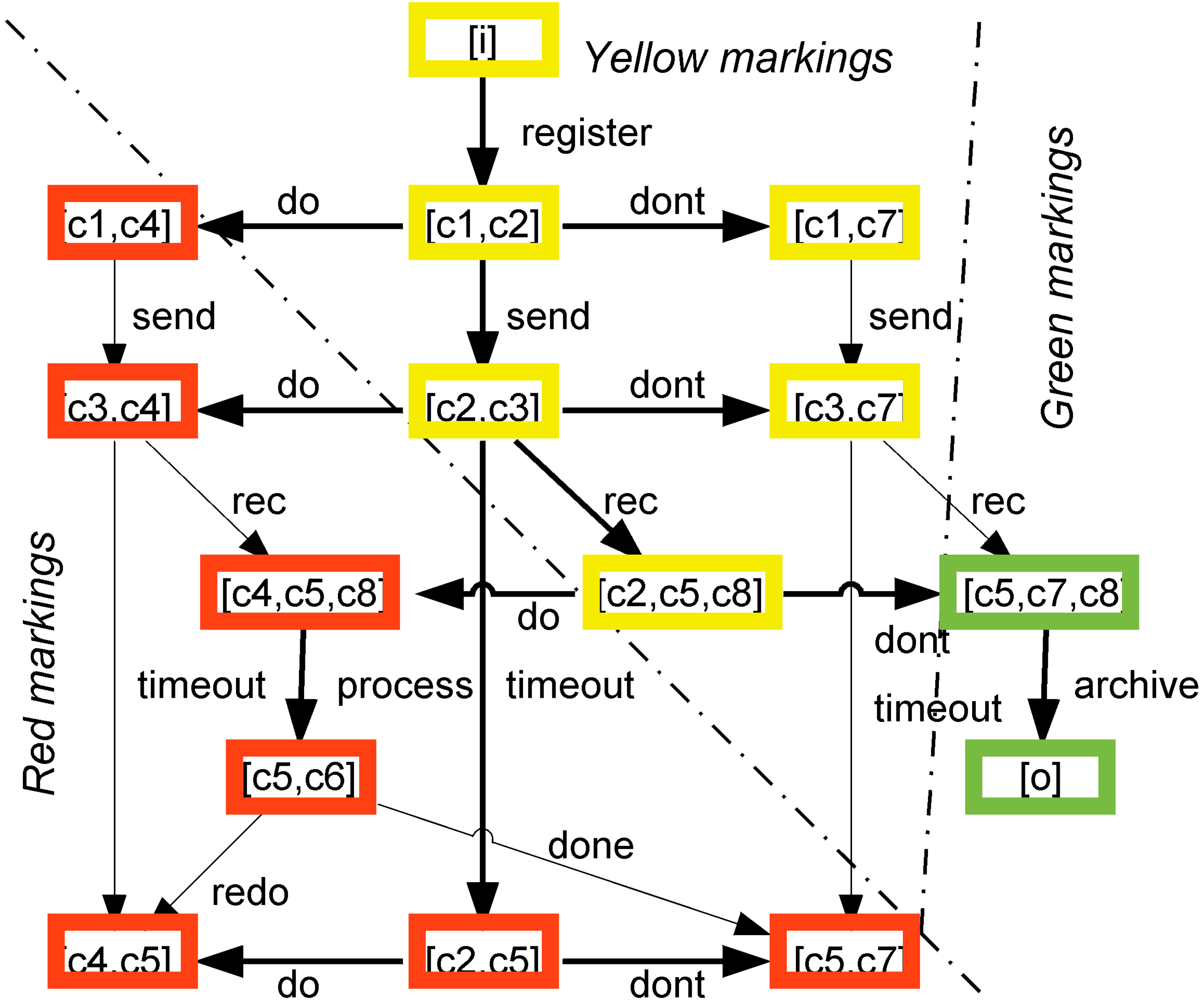
Color in yellow all remaining nodes
(some but not all paths lead to o)

Example: N


c1
send
c3
timeout
c5
rec
register
i
c8
archive
o
c2
do
c4
process
c6
done
c7
redo
dont

Example: RG (N)


c1
send
c3
timeout
c5
rec
register
c8
archive
i
o
c2
do
c4
process
c6
done
c7
redo
dont


[i]
Yellow markings
register
[c1,c4]
do
[c1,c2]
dont
[c1,c7]
send
send
send
Green markings
[c3,c4]
do
[c2,c3]
dont
[c3,c7]
rec
rec
rec
[c4,c5,c8]
do
[c2,c5,c8]
[c5,c7,c8]
dont
Red markings
timeout
process
timeout
timeout
archive
[c5,c6]
[o]
done
redo
[c4,c5]
do
[c2,c5]
dont
[c5,c7]

Non-live sequences:

register, do

register, send, do

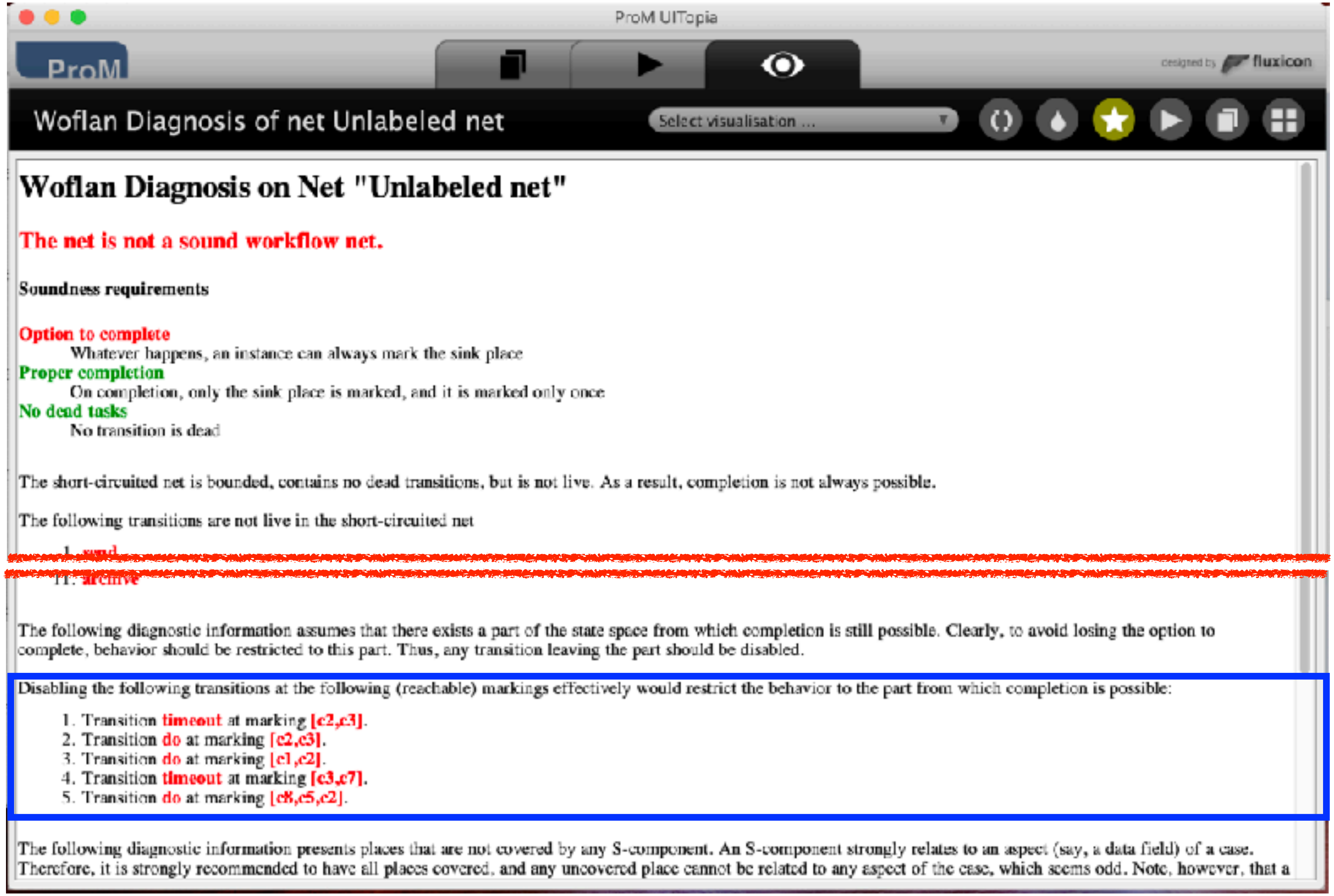
register, send, timeout

register, send, rec, do

register, send, dont, timeout

register, dont, send, timeout

Woflan (in ProM)



Woflan Diagnosis on Net "Unlabeled net"

The net is not a sound workflow net.

Soundness requirements

Option to complete
Whatever happens, an instance can always mark the sink place
Proper completion
On completion, only the sink place is marked, and it is marked only once
No dead tasks
No transition is dead

The short-circuited net is bounded, contains no dead transitions, but is not live. As a result, completion is not always possible.

The following transitions are not live in the short-circuited net

1. send
11. archive

The following diagnostic information assumes that there exists a part of the state space from which completion is still possible. Clearly, to avoid losing the option to complete, behavior should be restricted to this part. Thus, any transition leaving the part should be disabled.

Disabling the following transitions at the following (reachable) markings effectively would restrict the behavior to the part from which completion is possible:

1. Transition timeout at marking [c2,c3].
2. Transition do at marking [c2,c3].
3. Transition do at marking [c1,c2].
4. Transition timeout at marking [c3,c7].
5. Transition do at marking [c8,c5,c2].

The following diagnostic information presents places that are not covered by any S-component. An S-component strongly relates to an aspect (say, a data field) of a case. Therefore, it is strongly recommended to have all places covered, and any uncovered place cannot be related to any aspect of the case, which seems odd. Note, however, that a

Error sequences: Unbounded sequences

Unbounded sequences: informally

An **unbounded sequence** is a firing sequence of **minimal length** such that every continuation **invalidates proper completion**

i.e. a witness for unboundedness

Unbounded sequences: fundamental property

N* is bounded
iff
N has no unbounded sequences

Undesired markings:
infinite-weighted markings or markings greater than o

Unbounded sequences: graphically

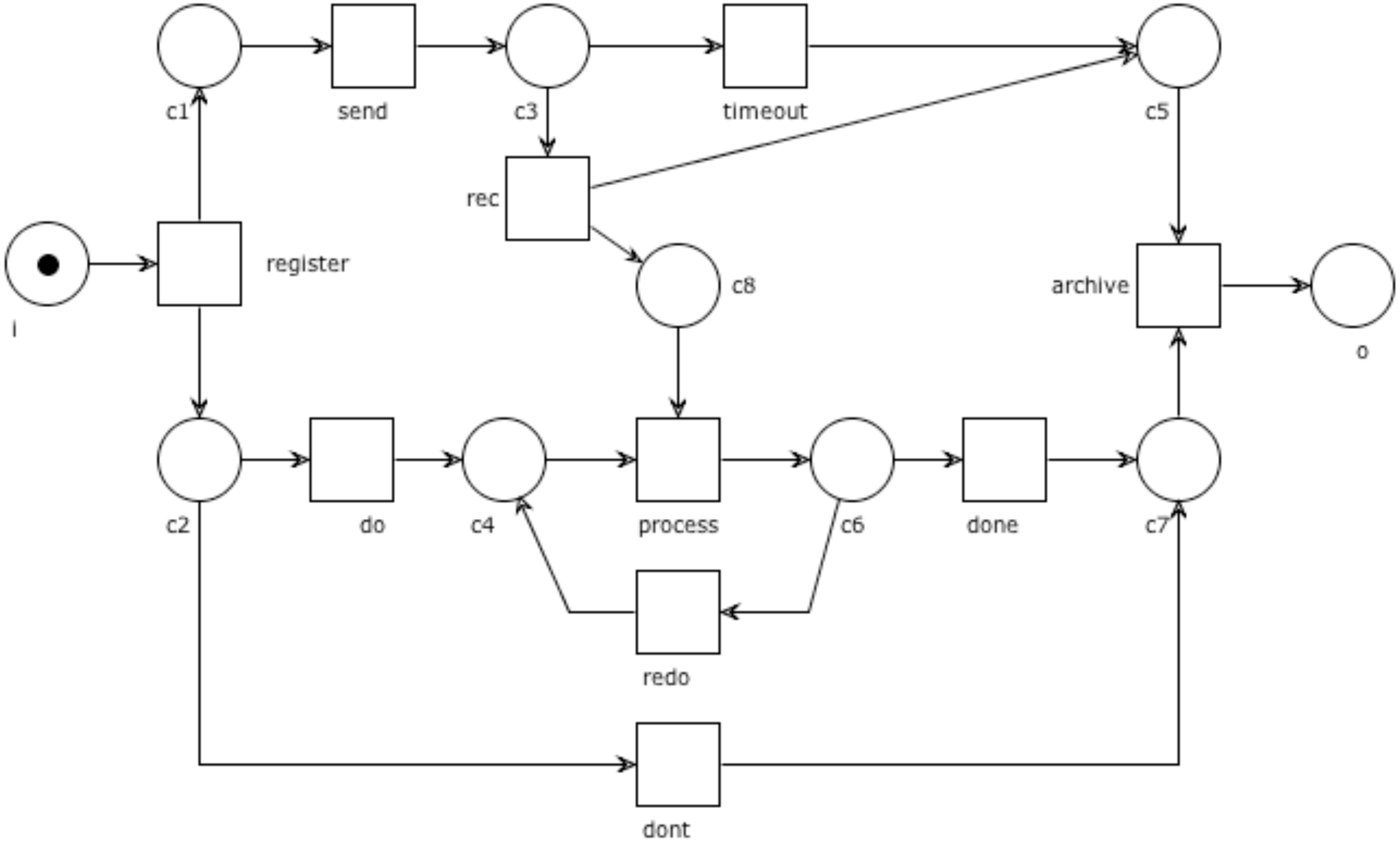
Compute the CG of N*

Color in green all nodes from which
undesired markings are not reachable

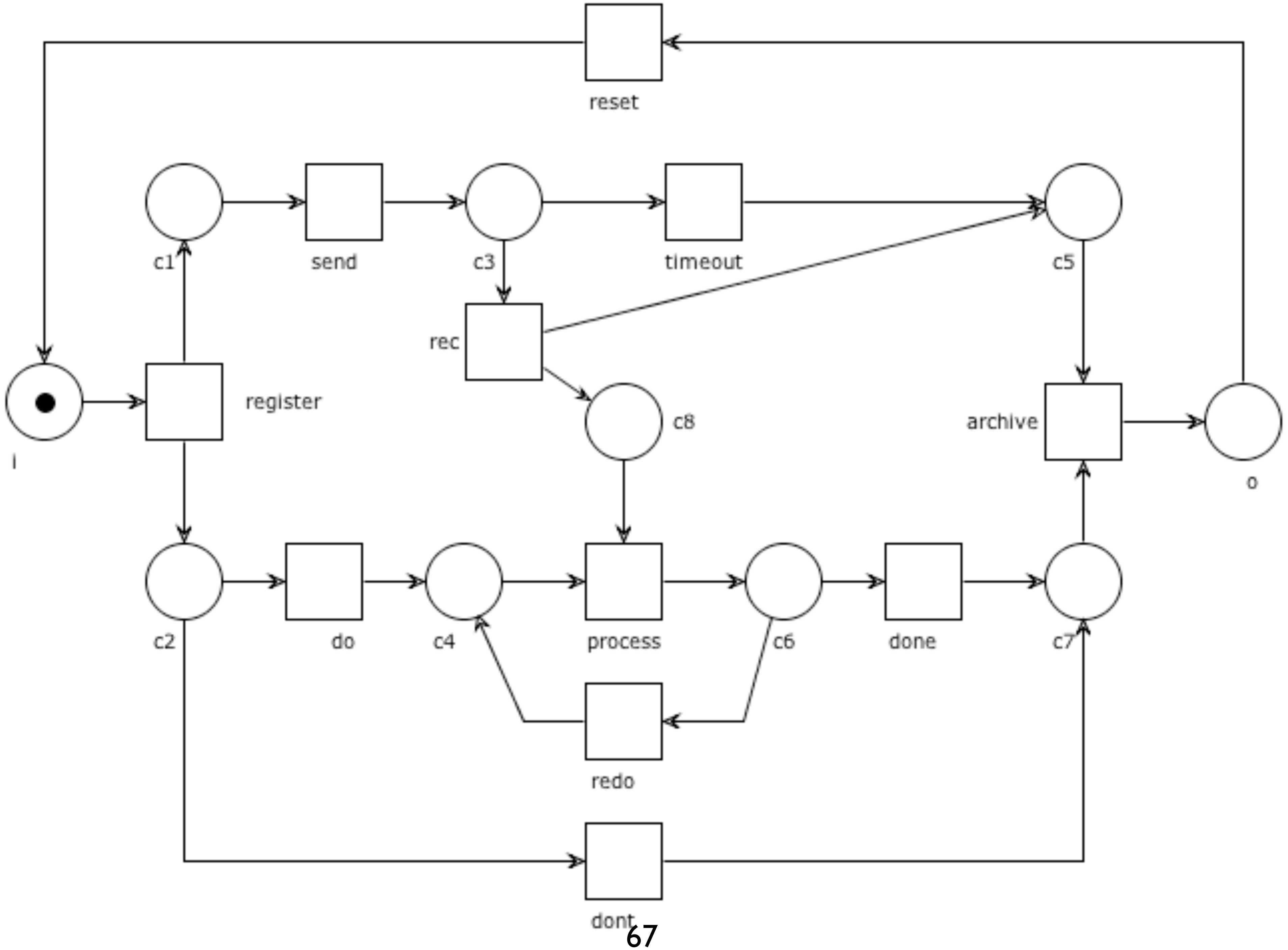
Color in red all nodes from which
no green marking is reachable
(undesired markings are unavoidable)

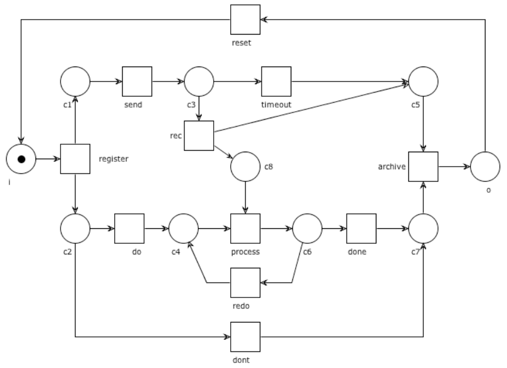
Color in yellow all remaining nodes
(undesired markings are reachable but avoidable)

Example: N

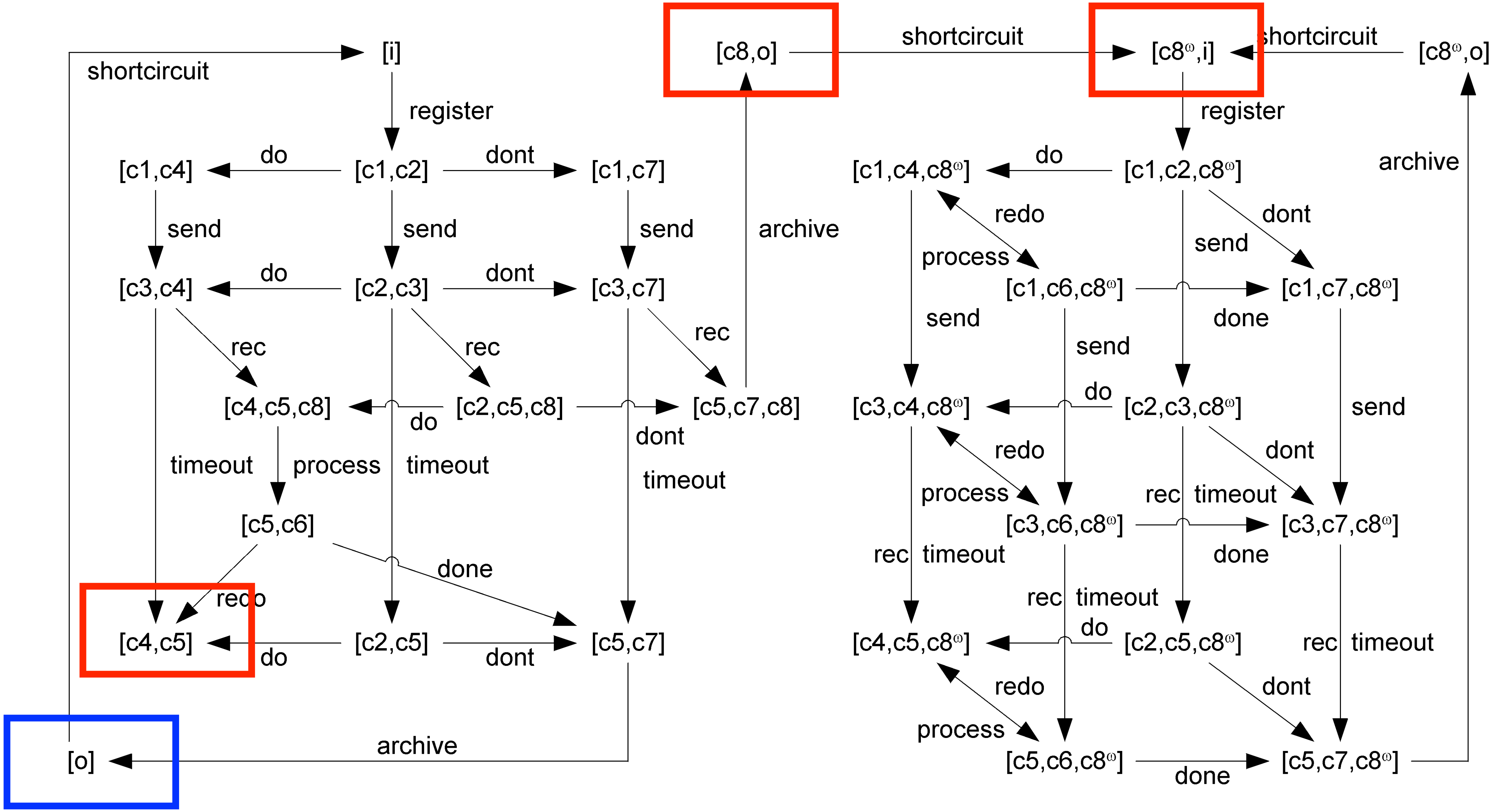

c1
send
c3
timeout
c5
rec
register
i
c8
archive
o
c2
do
c4
process
c6
done
c7
redo
dont

Example: N*


reset
c1
send
c3
timeout
c5
rec
i
register
c8
archive
o
c2
do
c4
process
c6
done
c7
redo
dont


reset
c1
send
c3
timeout
c5
rec
c8
register
archive
i
o
c2
do
c4
process
c6
done
c7
redo
dont

Example: CG (N*)


[i]
shortcircuit
register
[c1,c4]
do
[c1,c2]
dont
[c1,c7]
send
send
send
[c3,c4]
do
[c2,c3]
dont
[c3,c7]
rec
rec
rec
[c4,c5,c8]
do
[c2,c5,c8]
[c5,c7,c8]
dont
timeout
process
timeout
timeout
[c5,c6]
done
redo
[c4,c5]
do
[c2,c5]
dont
[c5,c7]
[o]
archive
[c8,o]
archive
shortcircuit
[c8^ω,i]
shortcircuit
[c8^ω,o]
register
archive
[c1,c4,c8^ω]
do
[c1,c2,c8^ω]
redo
dont
process
send
[c1,c6,c8^ω]
[c1,c7,c8^ω]
send
done
send
[c3,c4,c8^ω]
do
[c2,c3,c8^ω]
send
redo
dont
process
rec
timeout
[c3,c6,c8^ω]
[c3,c7,c8^ω]
rec
timeout
done
rec
timeout
do
[c4,c5,c8^ω]
[c2,c5,c8^ω]
rec
timeout
redo
dont
process
[c5,c6,c8^ω]
done
[c5,c7,c8^ω]

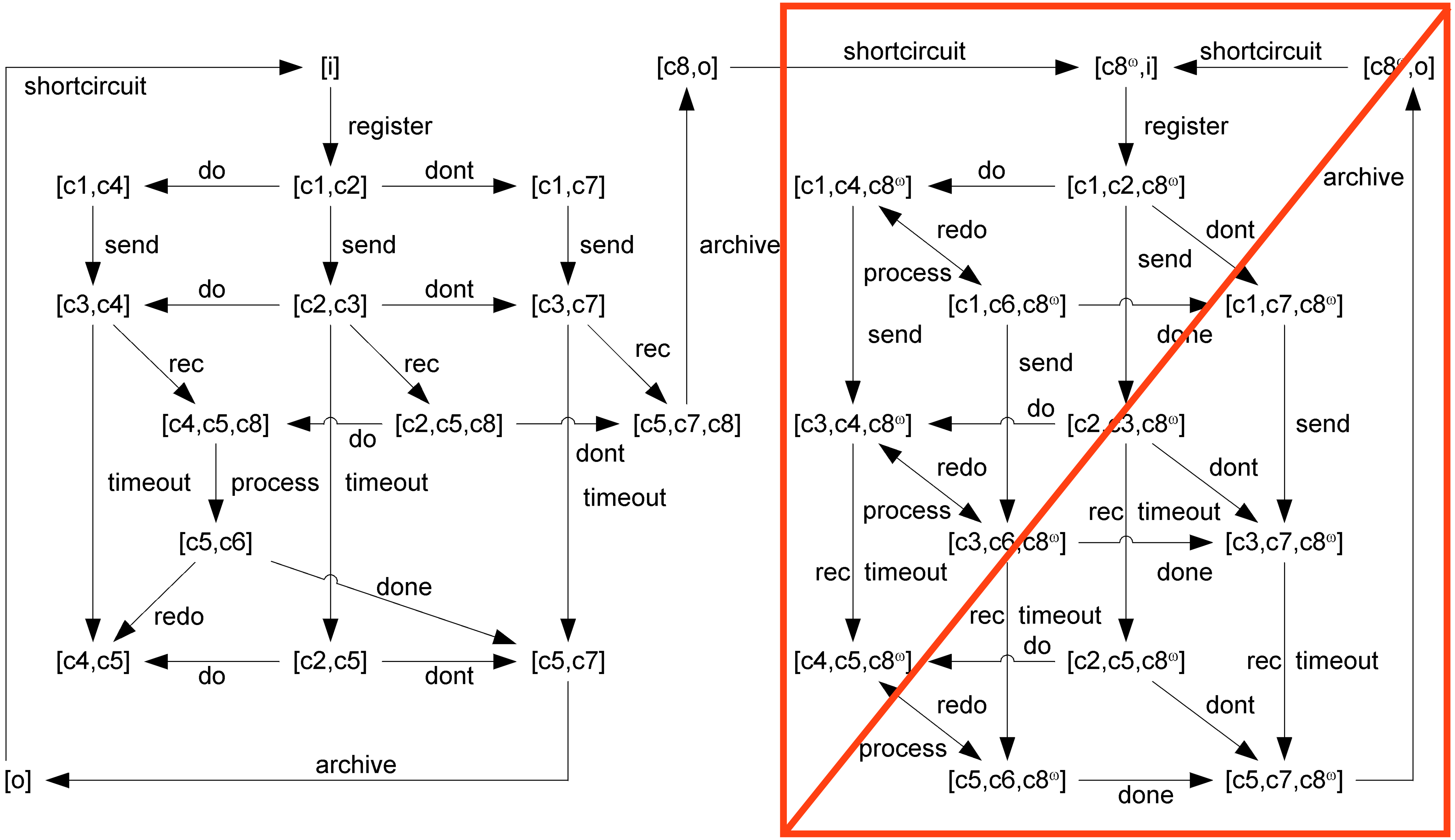
Restricted coverability graph (RCG)

CG can become very large

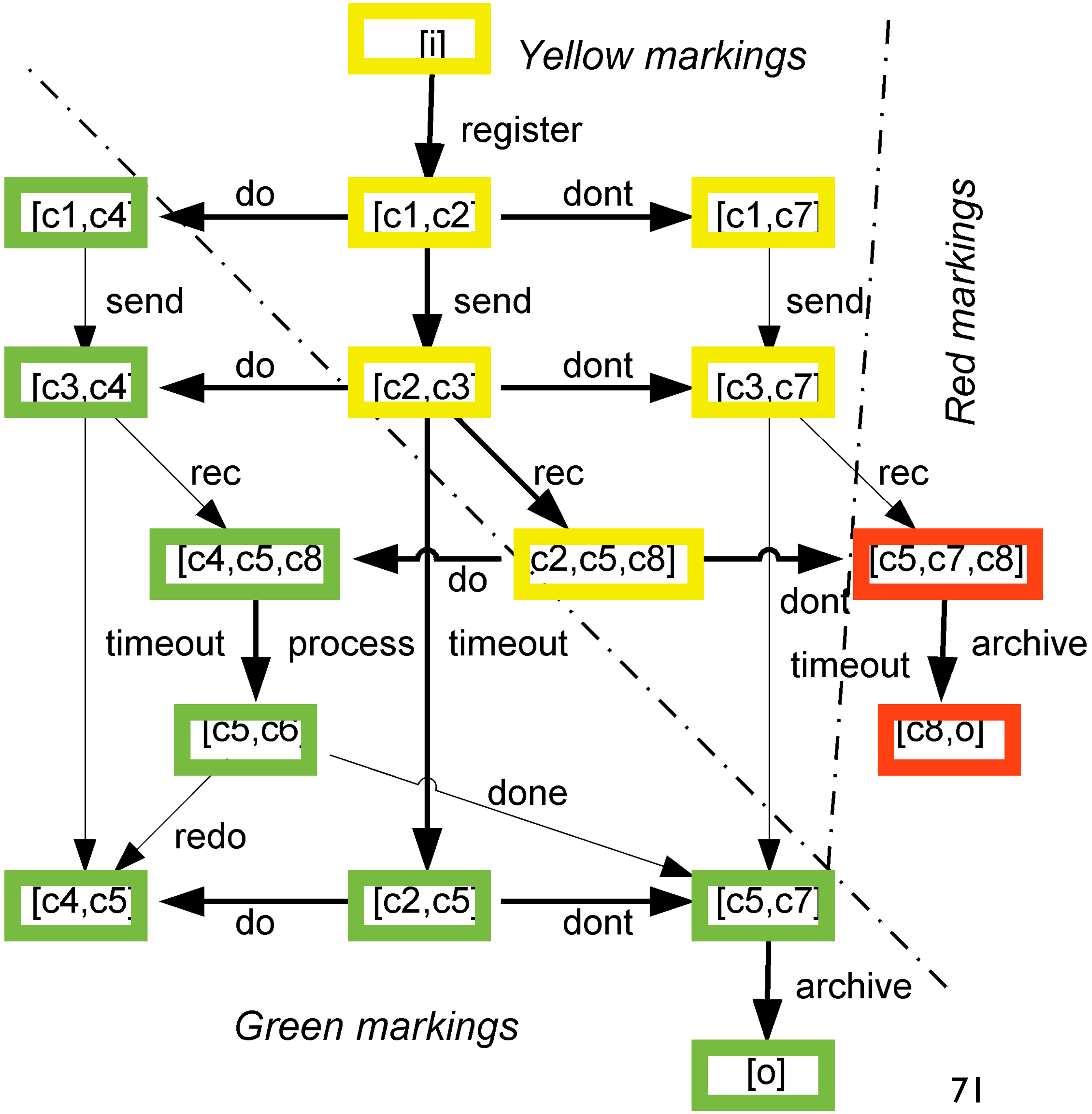
Basic observation:
infinite-weighted markings leads to infinite-weighted markings
and they will be all red

We can just avoid computing them!

Example: Restricted CG vs CG


shortcircuit
[i]
register
[c1,c4]
do
[c1,c2]
dont
[c1,c7]
send
send
send
[c3,c4]
do
[c2,c3]
dont
[c3,c7]
rec
rec
rec
[c4,c5,c8]
do
[c2,c5,c8]
[c5,c7,c8]
dont
timeout
process
timeout
timeout
[c5,c6]
done
redo
[c4,c5]
do
[c2,c5]
dont
[c5,c7]
[o]
archive
[c8,o]
archive
shortcircuit
[c8ω,i]
shortcircuit
[c8ω,o]
register
[c1,c4,c8ω]
do
[c1,c2,c8ω]
archive
redo
dont
send
process
[c1,c6,c8ω]
[c1,c7,c8ω]
send
done
send
[c3,c4,c8ω]
do
[c2,c3,c8ω]
send
redo
dont
process
rec
timeout
[c3,c6,c8ω]
[c3,c7,c8ω]
rec
timeout
done
rec
timeout
[c4,c5,c8ω]
do
[c2,c5,c8ω]
rec
timeout
redo
dont
process
[c5,c6,c8ω]
done
[c5,c7,c8ω]

Example: RCG (N*)


[i]
Yellow markings
register
[c1,c4]
do
[c1,c2]
dont
[c1,c7]
send
send
send
Red markings
[c3,c4]
do
[c2,c3]
dont
[c3,c7]
rec
rec
rec
[c4,c5,c8]
do
[c2,c5,c8]
[c5,c7,c8]
dont
timeout
process
timeout
timeout
archive
[c5,c6]
[c8,o]
done
redo
[c4,c5]
do
[c2,c5]
dont
[c5,c7]
archive
Green markings
[o]

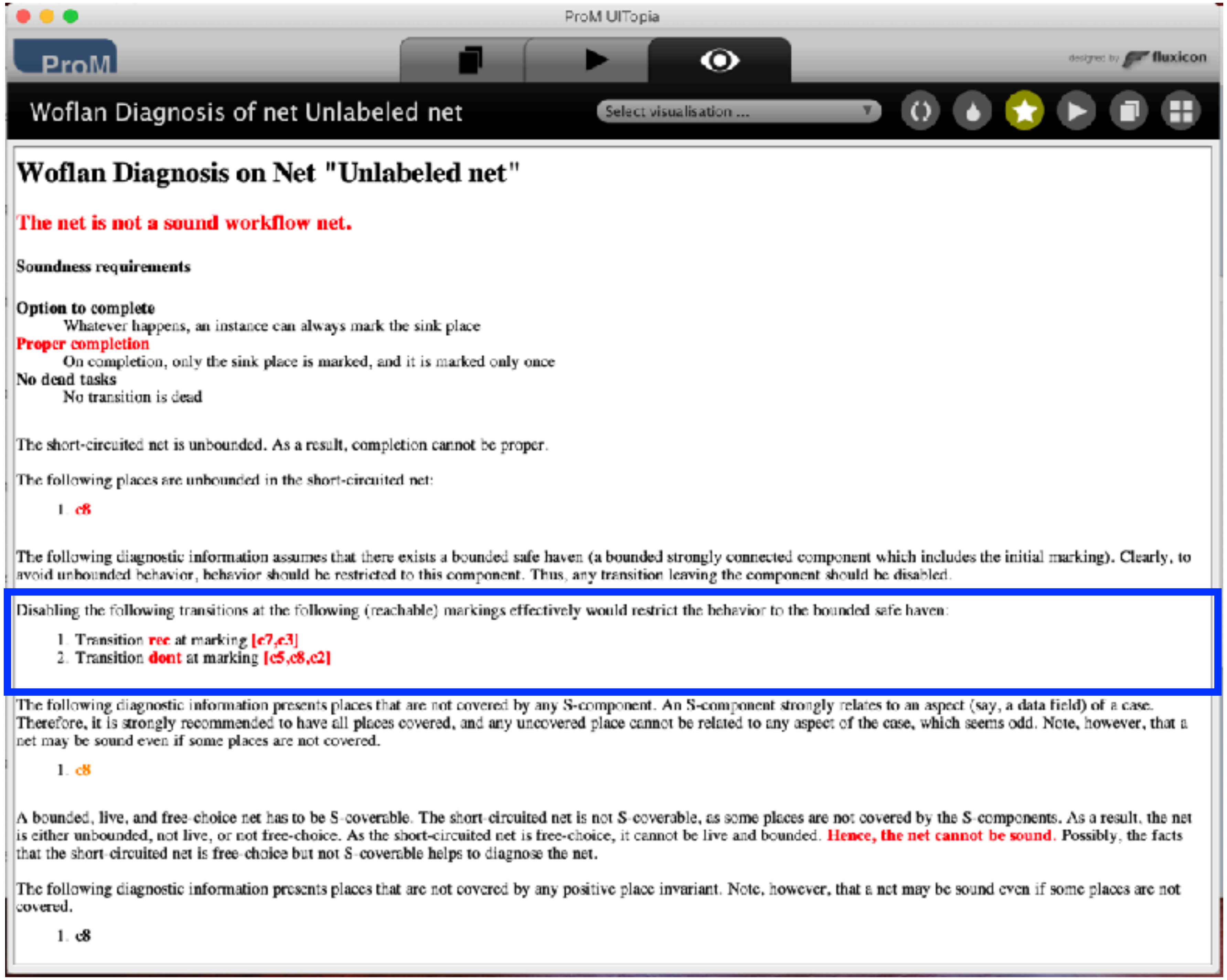
Unbounded sequences:

register, dont, send, rec

register, send, dont, rec

register, send, rec, dont

Woflan (in ProM)



Woflan Diagnosis on Net "Unlabeled net"

The net is not a sound workflow net.

Soundness requirements

Option to complete
Whatever happens, an instance can always mark the sink place
Proper completion
On completion, only the sink place is marked, and it is marked only once
No dead tasks
No transition is dead

The short-circuited net is unbounded. As a result, completion cannot be proper.

The following places are unbounded in the short-circuited net:

1. c8

The following diagnostic information assumes that there exists a bounded safe haven (a bounded strongly connected component which includes the initial marking). Clearly, to avoid unbounded behavior, behavior should be restricted to this component. Thus, any transition leaving the component should be disabled.

Disabling the following transitions at the following (reachable) markings effectively would restrict the behavior to the bounded safe haven:

1. Transition **rec** at marking **[c7,c3]**
2. Transition **dont** at marking **[c5,c8,c2]**

The following diagnostic information presents places that are not covered by any S-component. An S-component strongly relates to an aspect (say, a data field) of a case. Therefore, it is strongly recommended to have all places covered, and any uncovered place cannot be related to any aspect of the case, which seems odd. Note, however, that a net may be sound even if some places are not covered.

1. c8

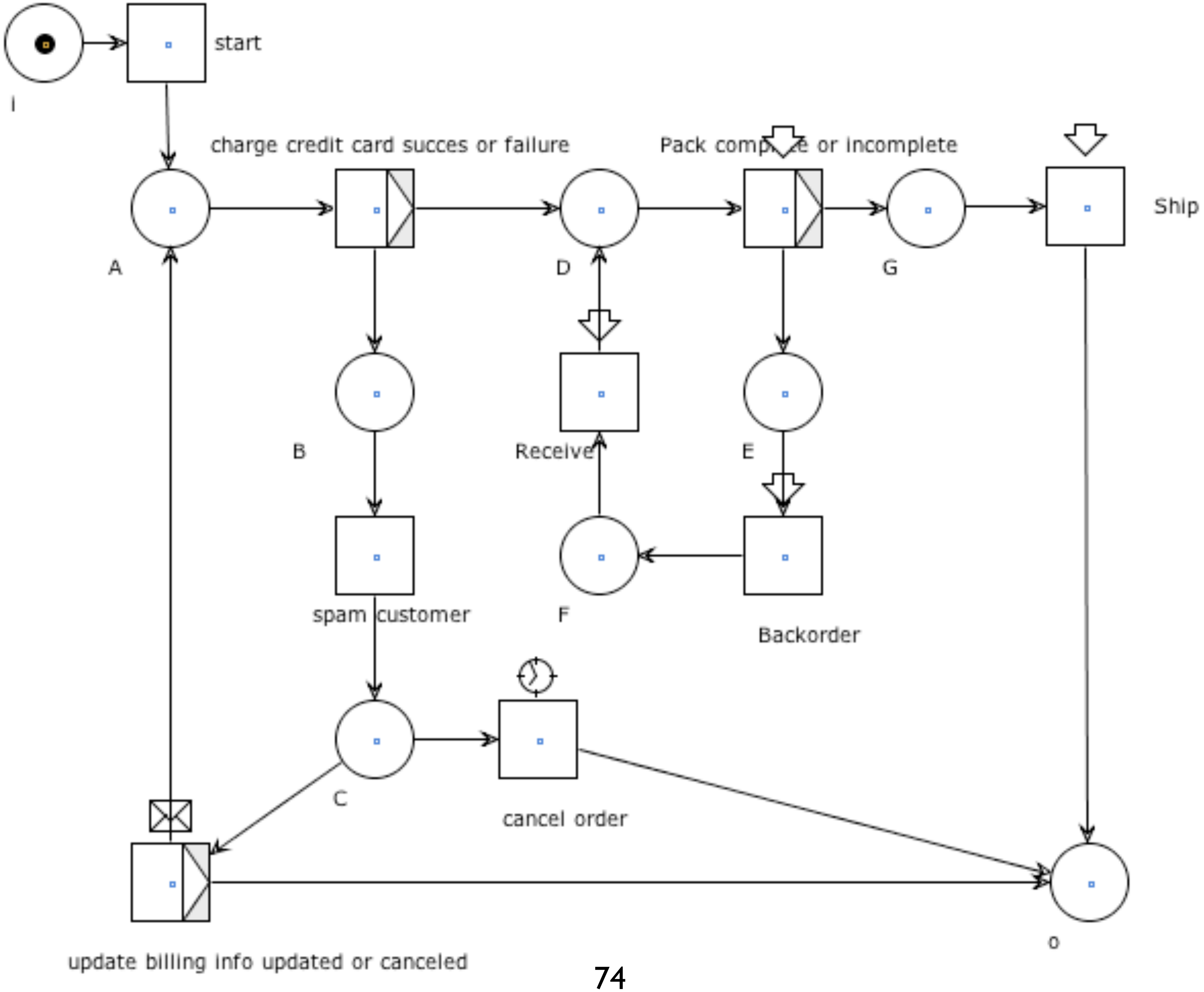
A bounded, live, and free-choice net has to be S-coverable. The short-circuited net is not S-coverable, as some places are not covered by the S-components. As a result, the net is either unbounded, not live, or not free-choice. As the short-circuited net is free-choice, it cannot be live and bounded. **Hence, the net cannot be sound.** Possibly, the facts that the short-circuited net is free-choice but not S-coverable helps to diagnose the net.

The following diagnostic information presents places that are not covered by any positive place invariant. Note, however, that a net may be sound even if some places are not covered.

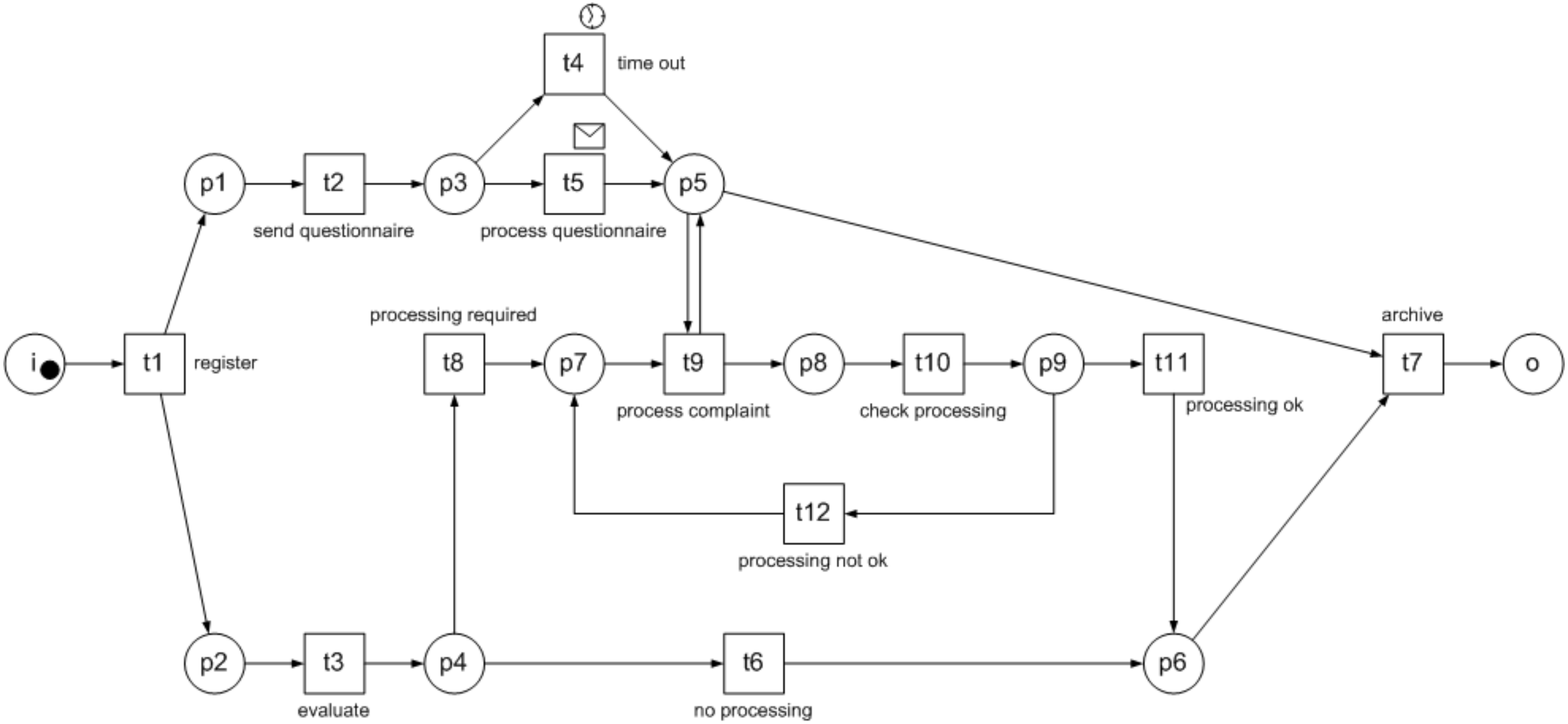
1. c8

Practice with WoPeD (and Woflan)

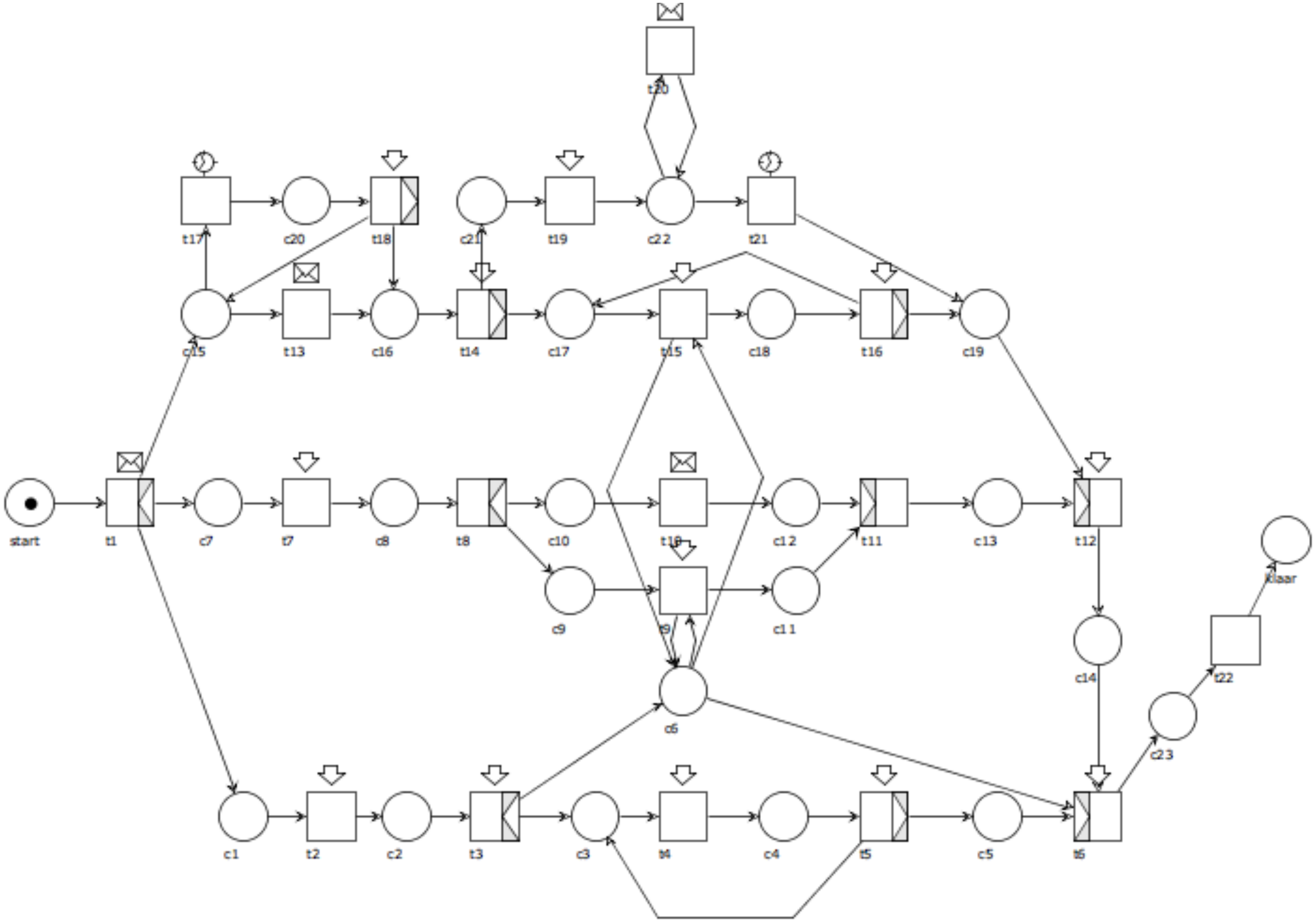
Analyse this net


start
i
charge credit card succes or failure
Pack complete or incomplete
Ship
A
D
G
B
Receive
E
spam customer
F
Backorder
C
cancel order
update billing info updated or canceled
o

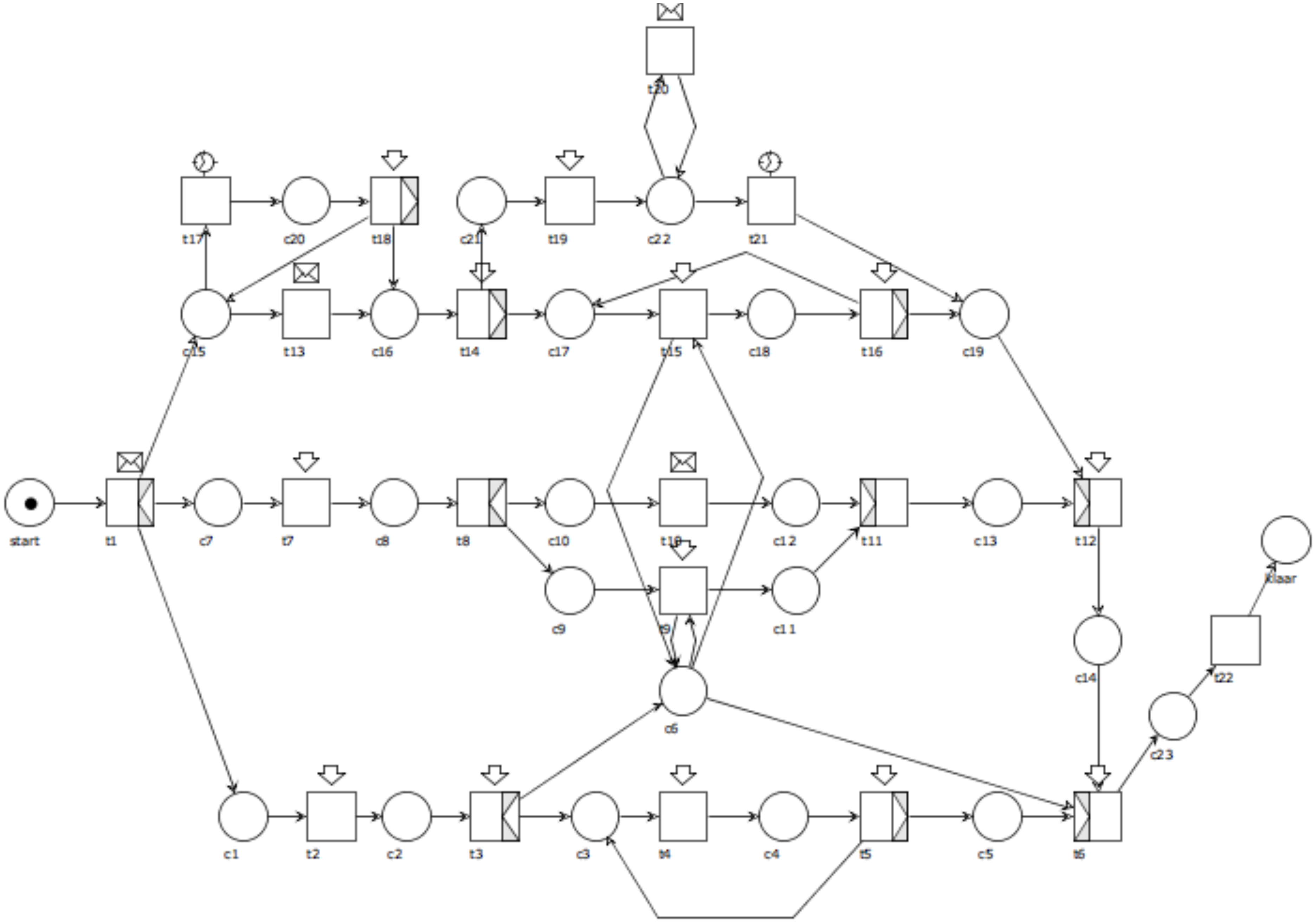
Analyse this net


t4
time out
p1
t2
p3
t5
p5
send questionnaire
process questionnaire
processing required
archive
i
t1
register
t8
p7
t9
p8
t10
p9
t11
t7
o
process complaint
check processing
processing ok
t12
processing not ok
p2
t3
p4
t6
p6
evaluate
no processing

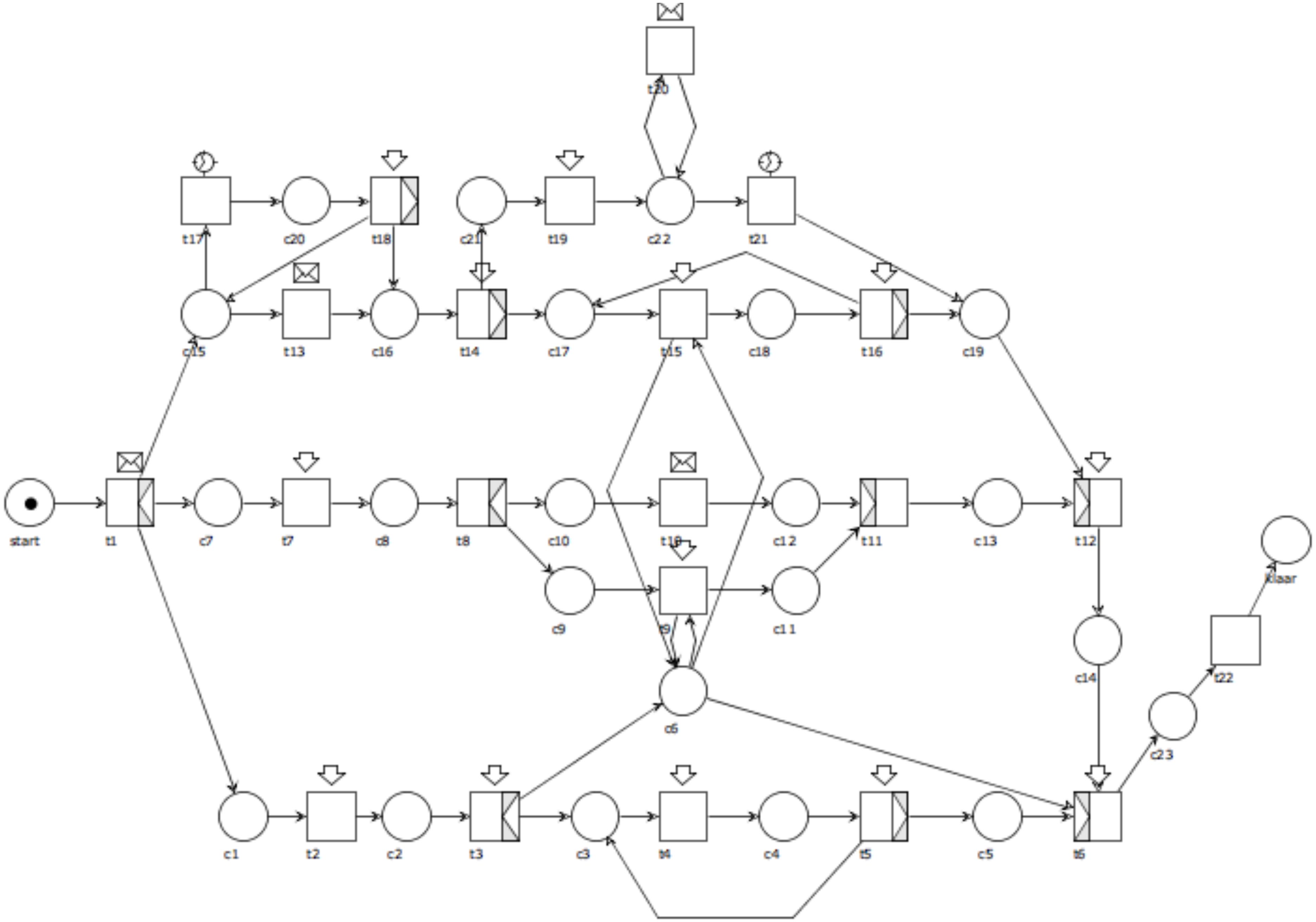
Is this net free-choice?


t20
t17
c20
t18
c21
t19
c22
t21
c15
t13
c16
t14
c17
t15
c18
t16
c19
start
t1
c7
t7
c8
t8
c10
t10
c12
t11
c13
t12
klaar
c9
t9
c11
c14
t22
c6
c23
c1
t2
c2
t3
c3
t4
c4
t5
c5
t6

Is this net S-coverable?


t20
t17
c20
t18
c21
t19
c22
t21
c15
t13
c16
t14
c17
t15
c18
t16
c19
start
t1
c7
t7
c8
t8
c10
t10
c12
t11
c13
t12
klaar
c9
t9
c11
c14
t22
c6
c23
c1
t2
c2
t3
c3
t4
c4
t5
c5
t6

Is this net sound?


t20
t17
c20
t18
c21
t19
c22
t21
c15
t13
c16
t14
c17
t15
c18
t16
c19
start
t1
c7
t7
c8
t8
c10
t10
c12
t11
c13
t12
klaar
c9
t9
c11
c14
t22
c6
c23
c1
t2
c2
t3
c3
t4
c4
t5
c5
t6

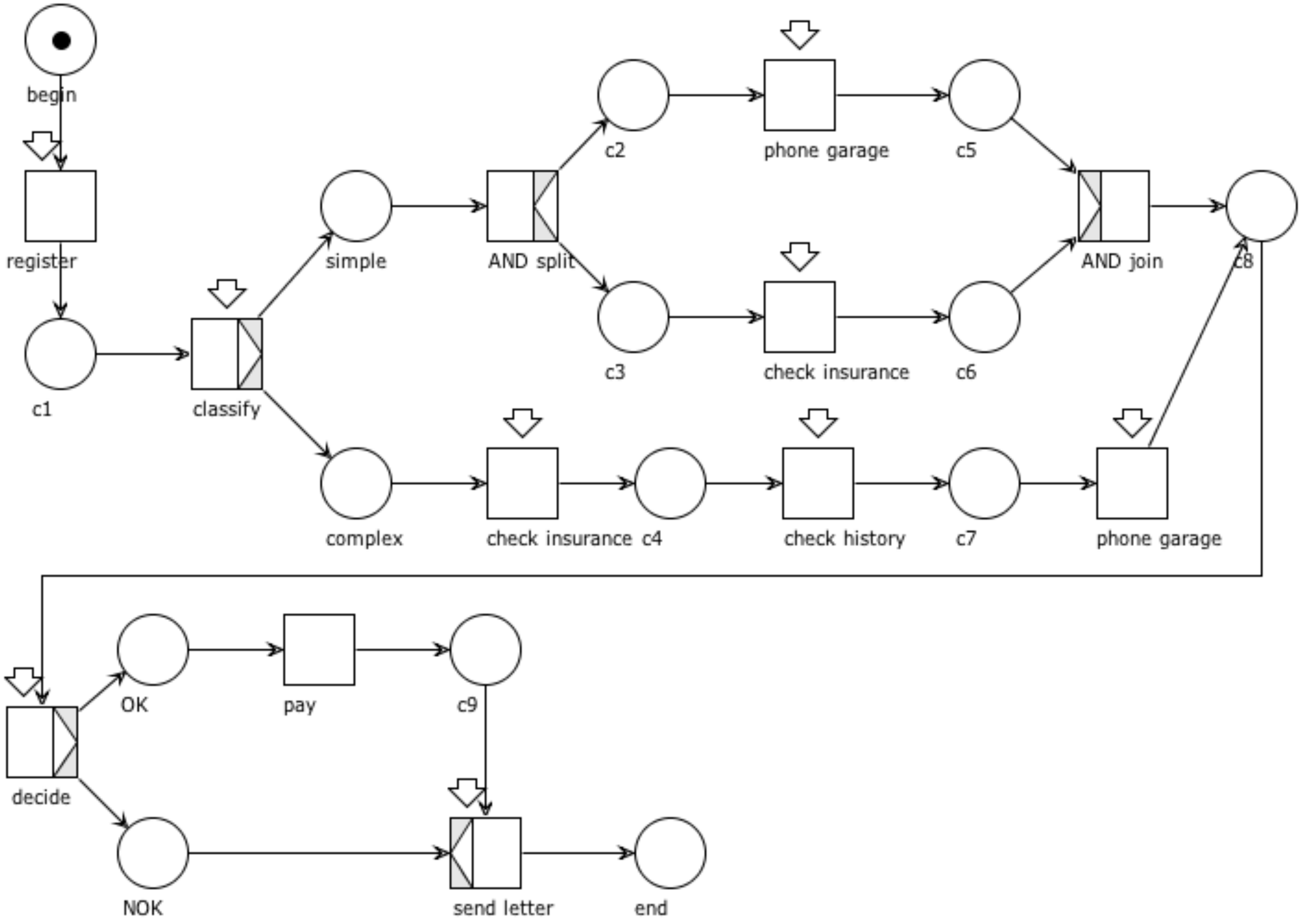
Design Example: Car Damage

- An insurance company uses the following procedure for the processing of the claims
- Every claim, reported by a customer, is registered
- After the registration, the claim is classified
- There are two categories: simple and complex claims.
  - For simple claims two tasks need to be executed:
    check insurance and phone garage.
    These tasks are *independent* of each other.
  - The complex claims require three tasks:
    check insurance, check damage history and phone garage.
    These tasks need to be *executed sequentially* in the order specified.
- After executing the two/three tasks a decision is taken with two possible outcomes: OK (positive) or NOK (negative).
- If the decision is positive, then insurance company will pay.
- In any event, the insurance company sends a letter to the customer.

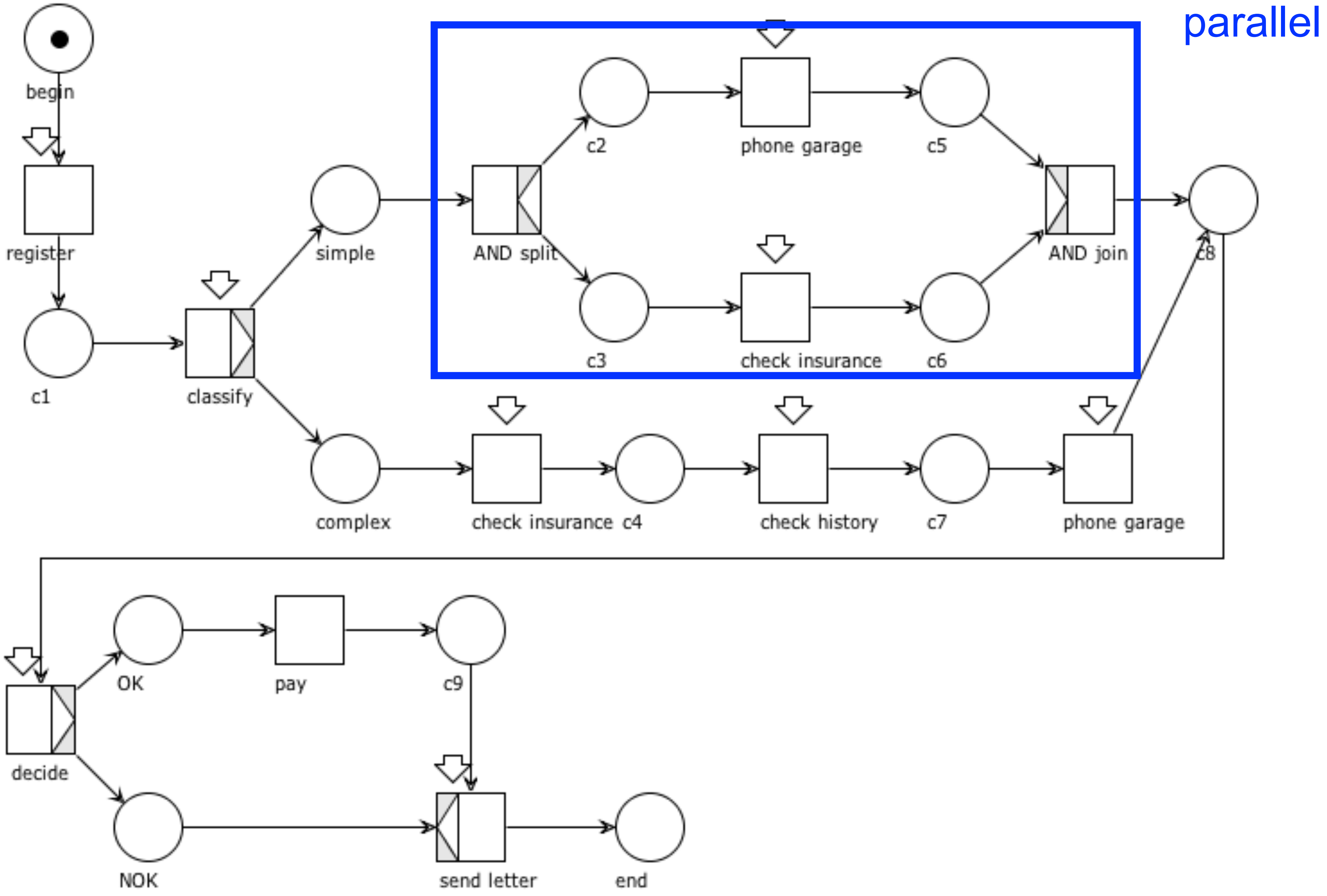
Design Example: Car Damage

- An insurance company uses the following procedure for the processing of the claims
- Every claim, reported by a customer, is **registered**
- After the registration, the claim is **classified**
- There are two categories: **simple** and **complex** claims.
    - For simple claims two tasks need to be executed: **check insurance** and **phone garage**. These tasks are *independent* of each other.
    - The complex claims require three tasks: **check insurance**, **check damage history** and **phone garage**. These tasks need to be *executed sequentially* in the order specified.
- After executing the two/three tasks a **decision** is taken with two possible outcomes: **OK** (positive) or **NOK** (negative).
- If the decision is positive, then insurance company will **pay**.
- In any event, the insurance company **sends a letter** to the customer.

Design Example: Car Damage

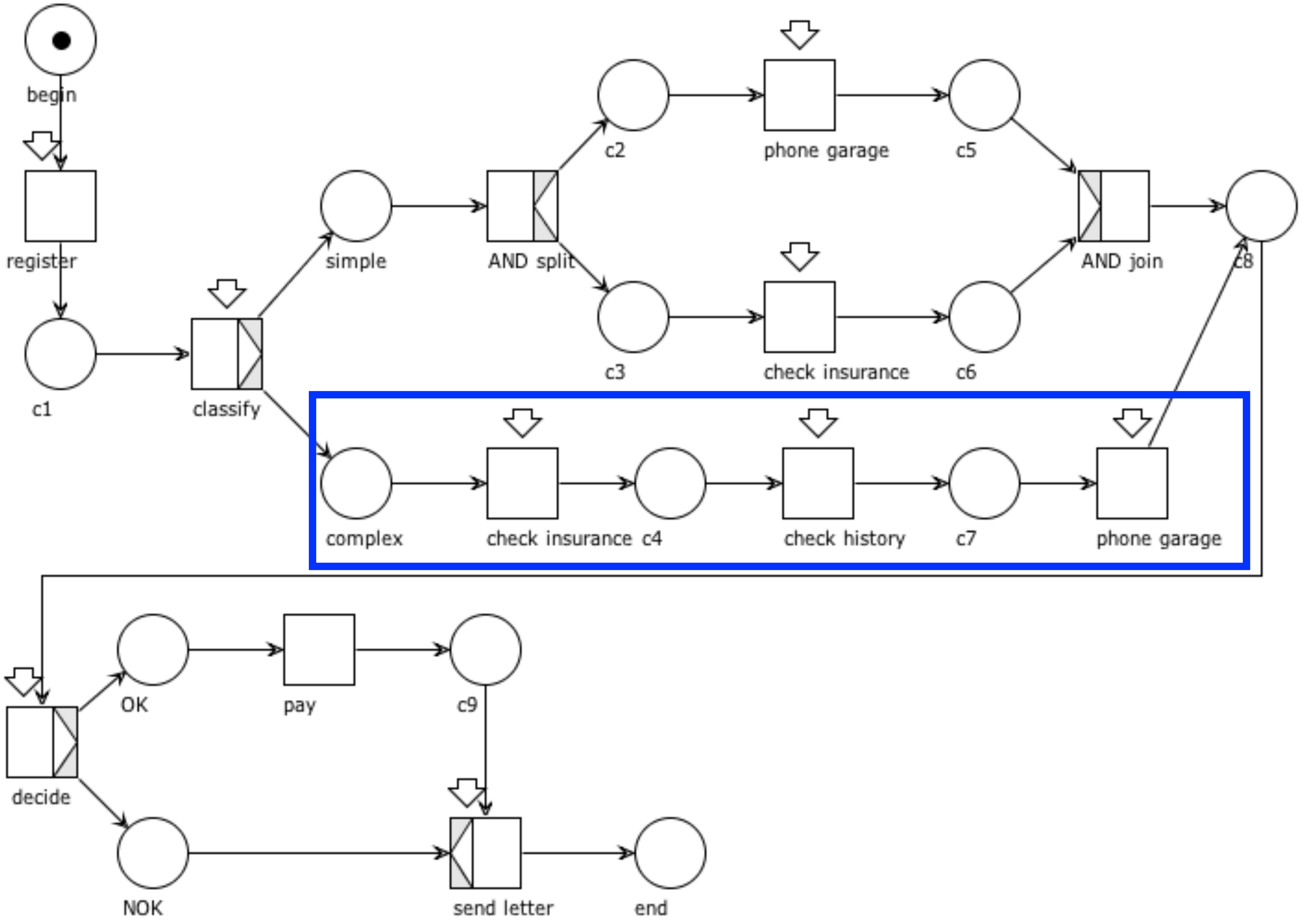

begin
register
c1
classify
simple
complex
AND split
c2
c3
phone garage
check insurance
c5
c6
AND join
c8
check insurance
c4
check history
c7
phone garage
decide
OK
pay
c9
NOK
send letter
end

Design Example: Car Damage


begin
register
c1
classify
simple
complex
AND split
c2
c3
phone garage
check insurance
c5
c6
AND join
c8
check insurance
c4
check history
c7
phone garage
decide
OK
pay
c9
NOK
send letter
end

parallel

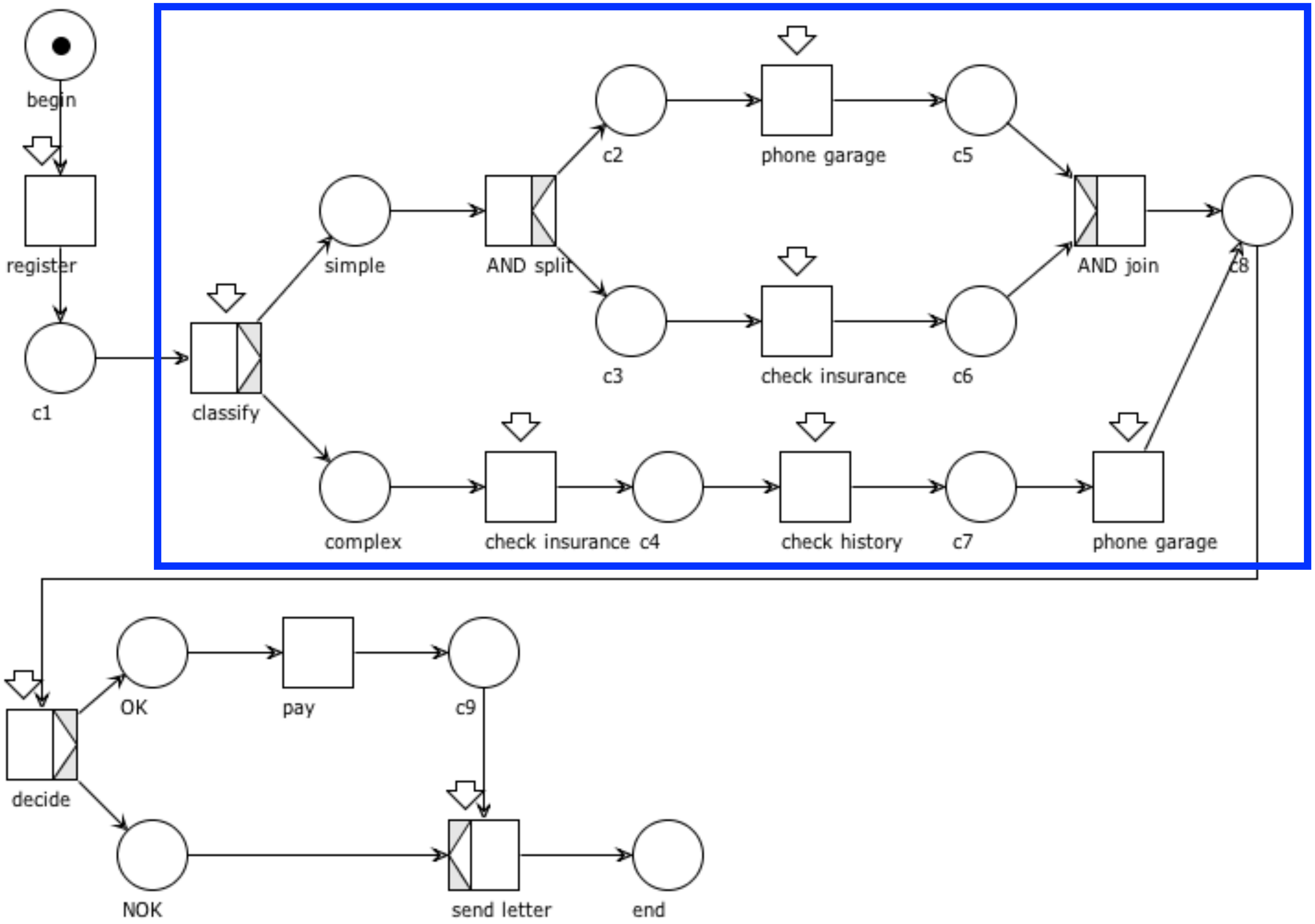
Design Example: Car Damage



begin
register
c1
classify
simple
AND split
c2
phone garage
c5
c3
check insurance
c6
AND join
c8
complex
check insurance
c4
check history
c7
phone garage
decide
OK
pay
c9
NOK
send letter
end

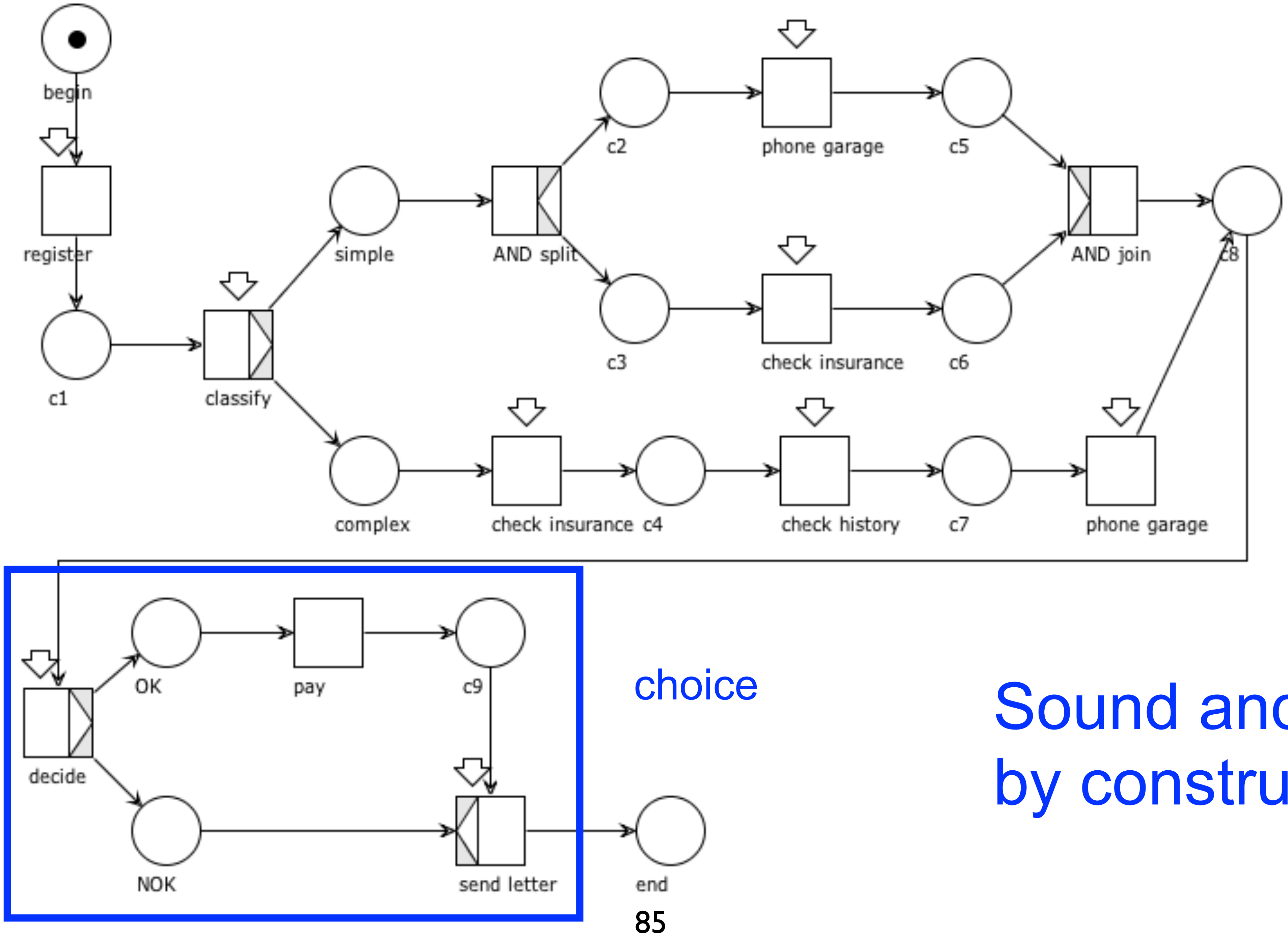
sequence

Design Example: Car Damage


begin
register
c1
classify
simple
AND split
c2
phone garage
c5
AND join
c3
check insurance
c6
c8
complex
check insurance
c4
check history
c7
phone garage
decide
OK
pay
c9
NOK
send letter
end

choice

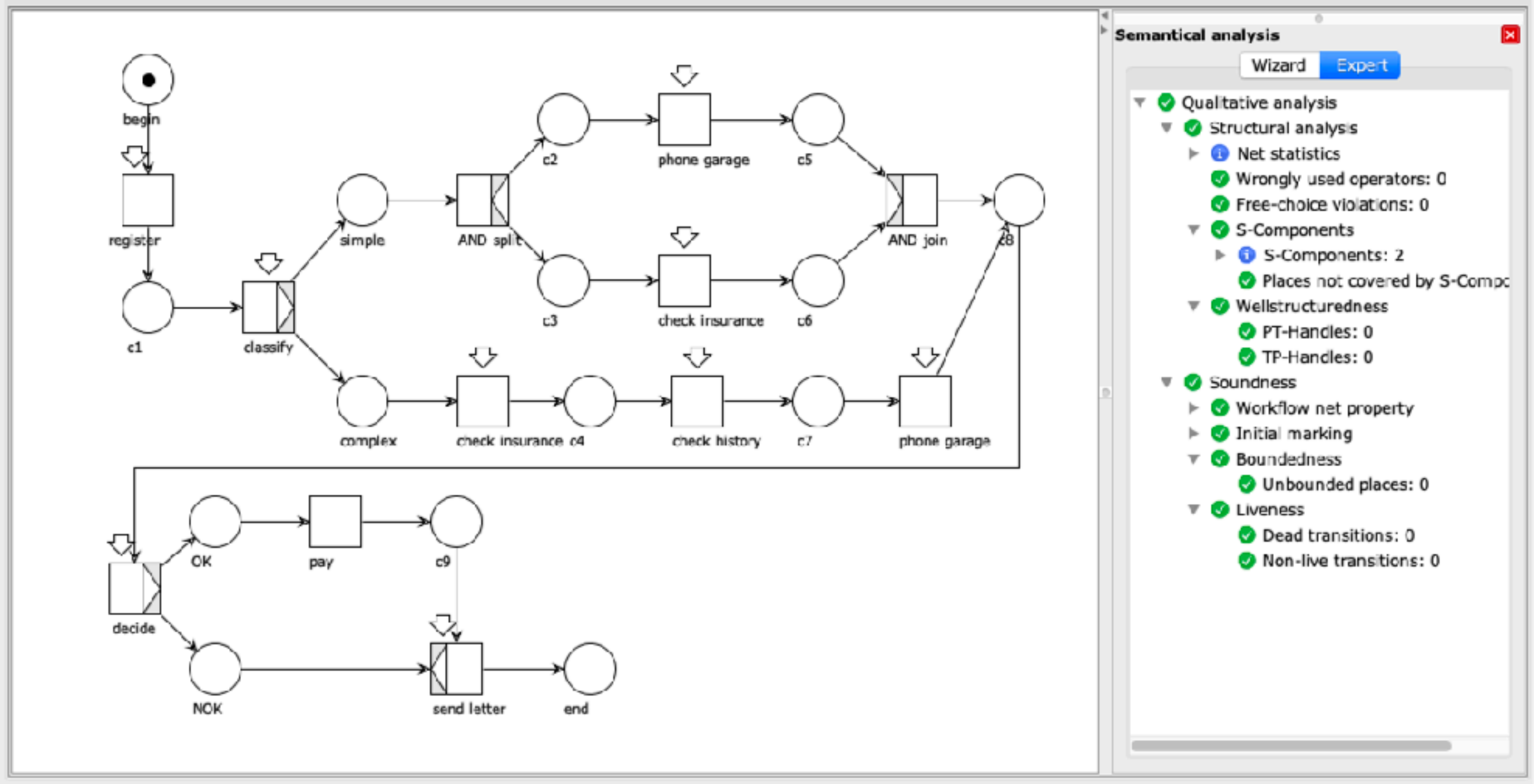
Design Example: Car Damage


begin
register
c1
classify
simple
complex
AND split
c2
c3
phone garage
check insurance
c5
c6
AND join
c8
check insurance
c4
check history
c7
phone garage
decide
OK
NOK
pay
c9
send letter
end

choice

Sound and safe by construction!

Design Example: Car Damage


begin
register
c1
classify
simple
complex
AND split
c2
c3
phone garage
check insurance
c5
c6
AND join
c8
check insurance
c4
check history
c7
phone garage
decide
OK
NOK
pay
c9
send latter
end
Semantical analysis
Wizard
Expert

- Qualitative analysis
  - Structural analysis
    - Net statistics
    - Wrongly used operators: 0
    - Free-choice violations: 0
    - S-Components
      - S-Components: 2
      - Places not covered by S-Comp
    - Wellstructuredness
      - PT-Handles: 0
      - TP-Handles: 0
  - Soundness
    - Workflow net property
    - Initial marking
    - Boundedness
      - Unbounded places: 0
    - Liveness
      - Dead transitions: 0
      - Non-live transitions: 0

Design and analysis of WF-nets

The workflow of a computer repair service (CRS) can be described as follows.
A customer brings in a defective computer and the CRS checks the defect and hands out a repair cost calculation back.
If the customer decides that the costs are acceptable, the process continues, otherwise she takes her computer home unrepaired.
The ongoing repair consists of two activities, which are executed sequentially but in an arbitrary order.
One activity is to check and repair the hardware,
whereas the other activity is to check and configure the software.
After both activities are completed, the proper system functionality is tested.
If an error is detected the repair procedure is repeated,
otherwise the repair is finished and the computer is returned.

Model the described workflow as a sound workflow net.