Business Processes Modelling

MPB (6 cfu, 295AA)

Roberto Bruni

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

19 - Diagnosis for WF nets

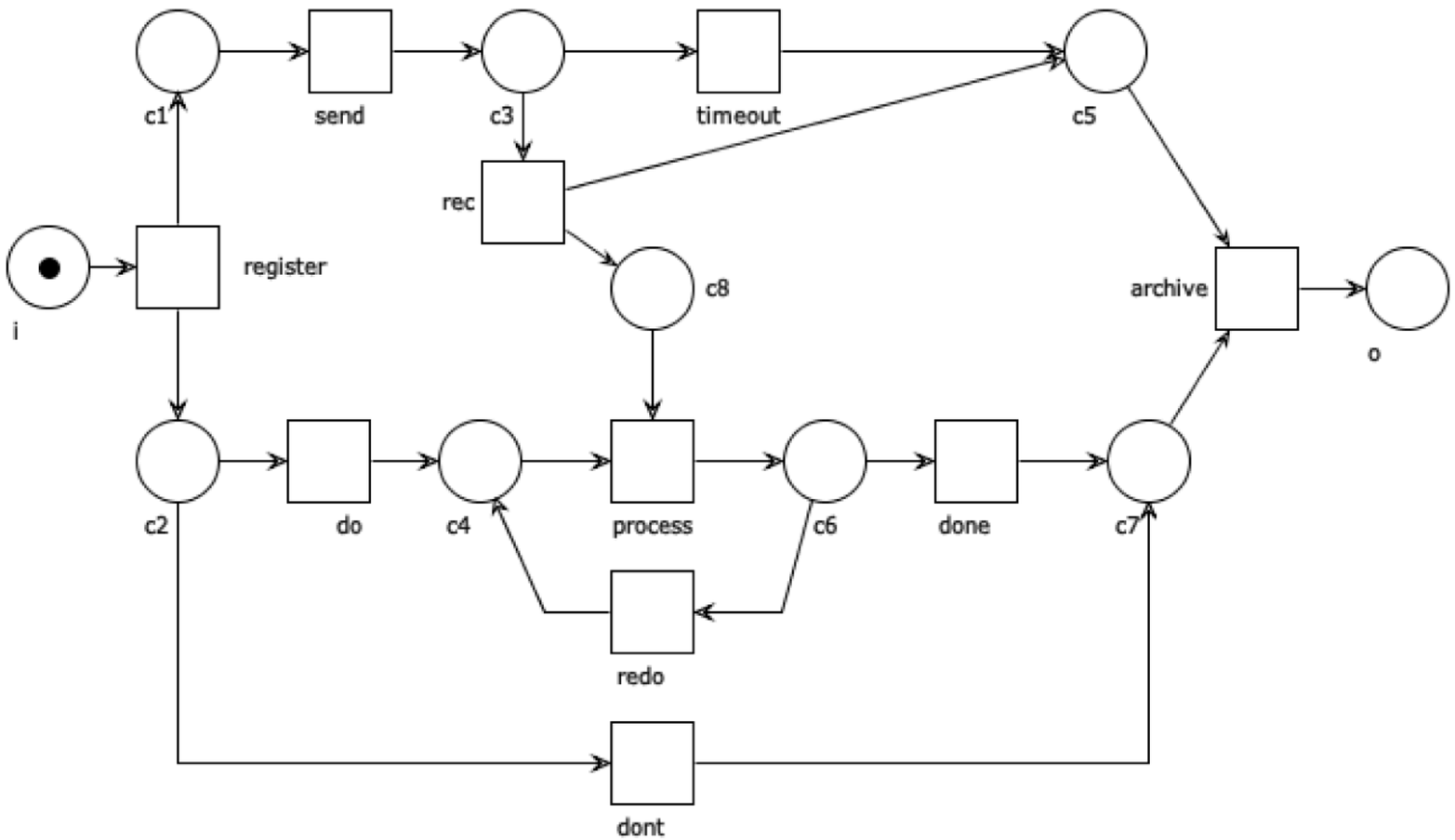
Object

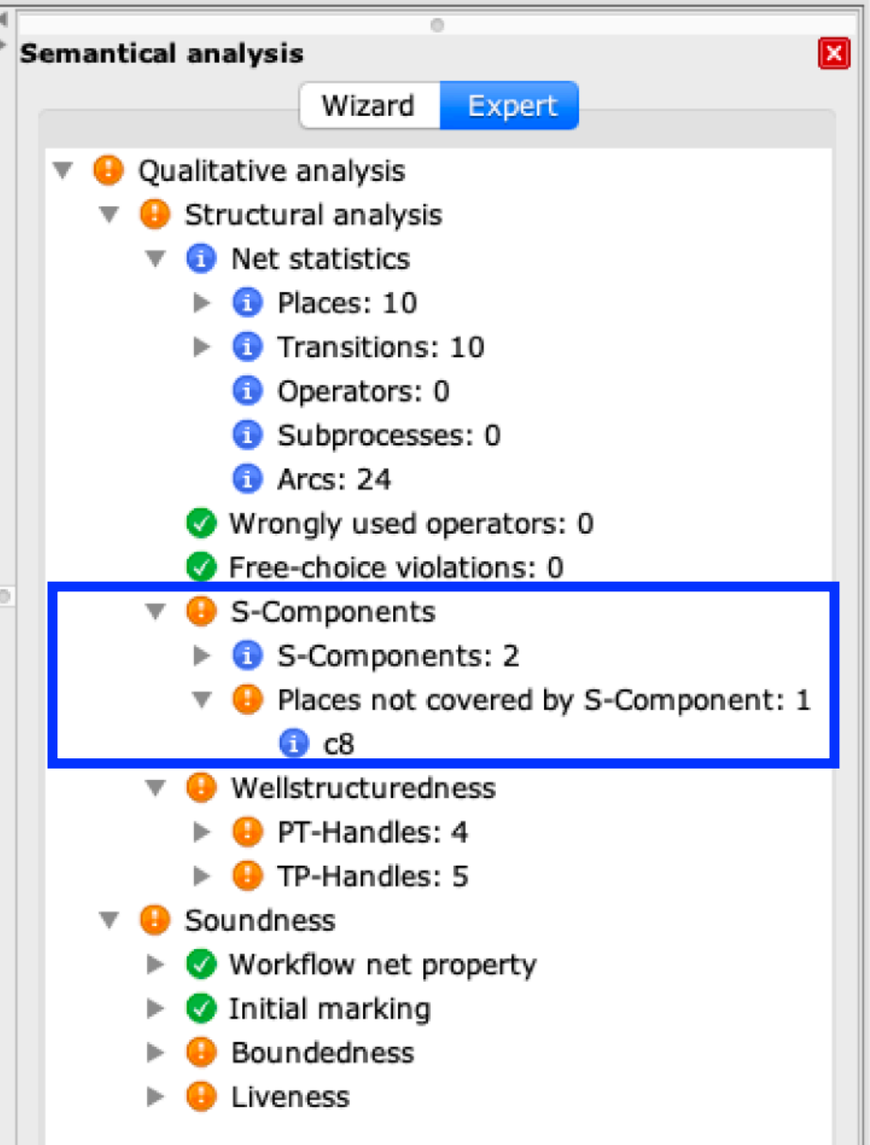
We study suitable diagnosis techniques for unsound Workflow nets

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

Woped

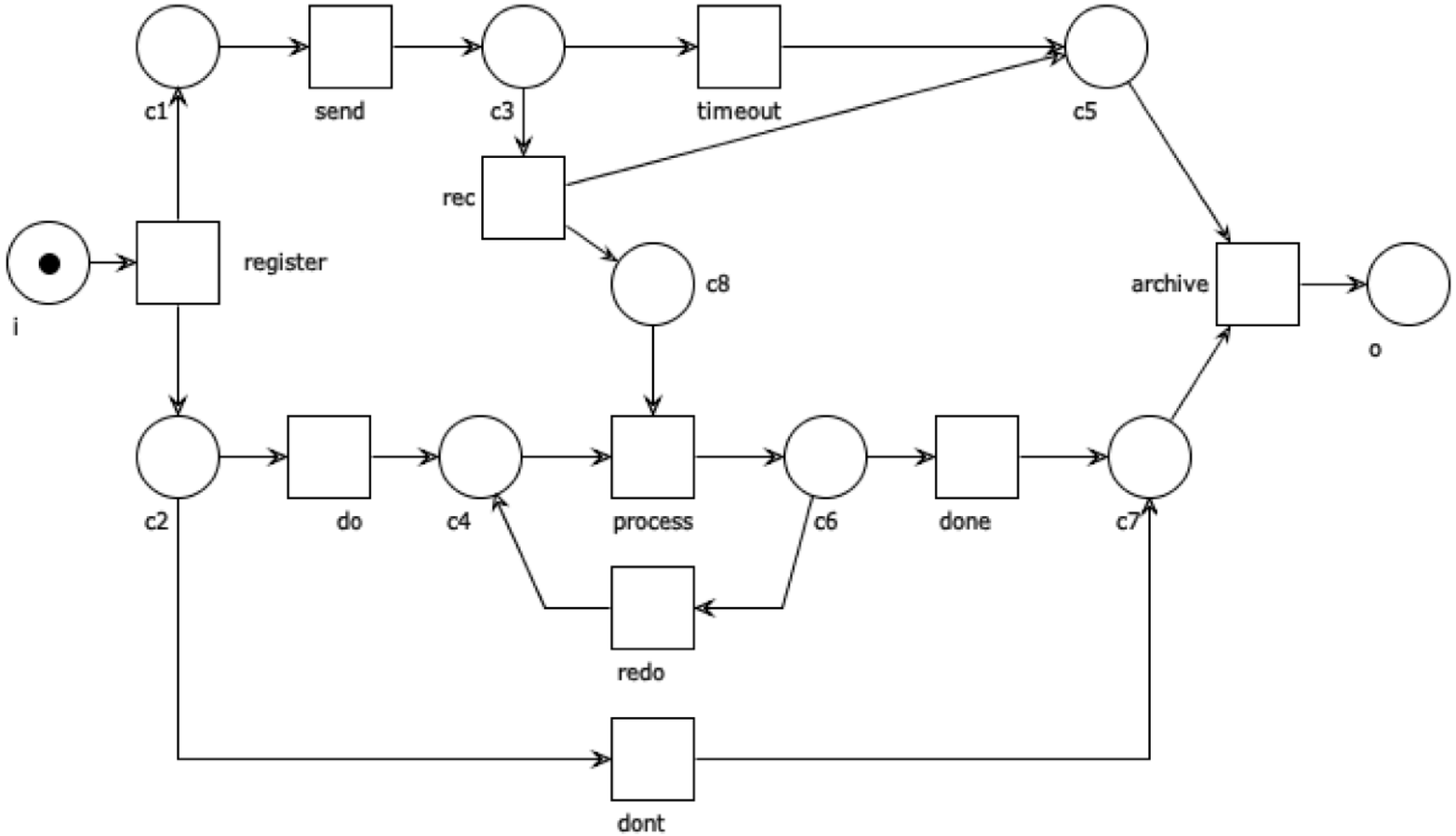
what are S-components?
and why are they relevant?

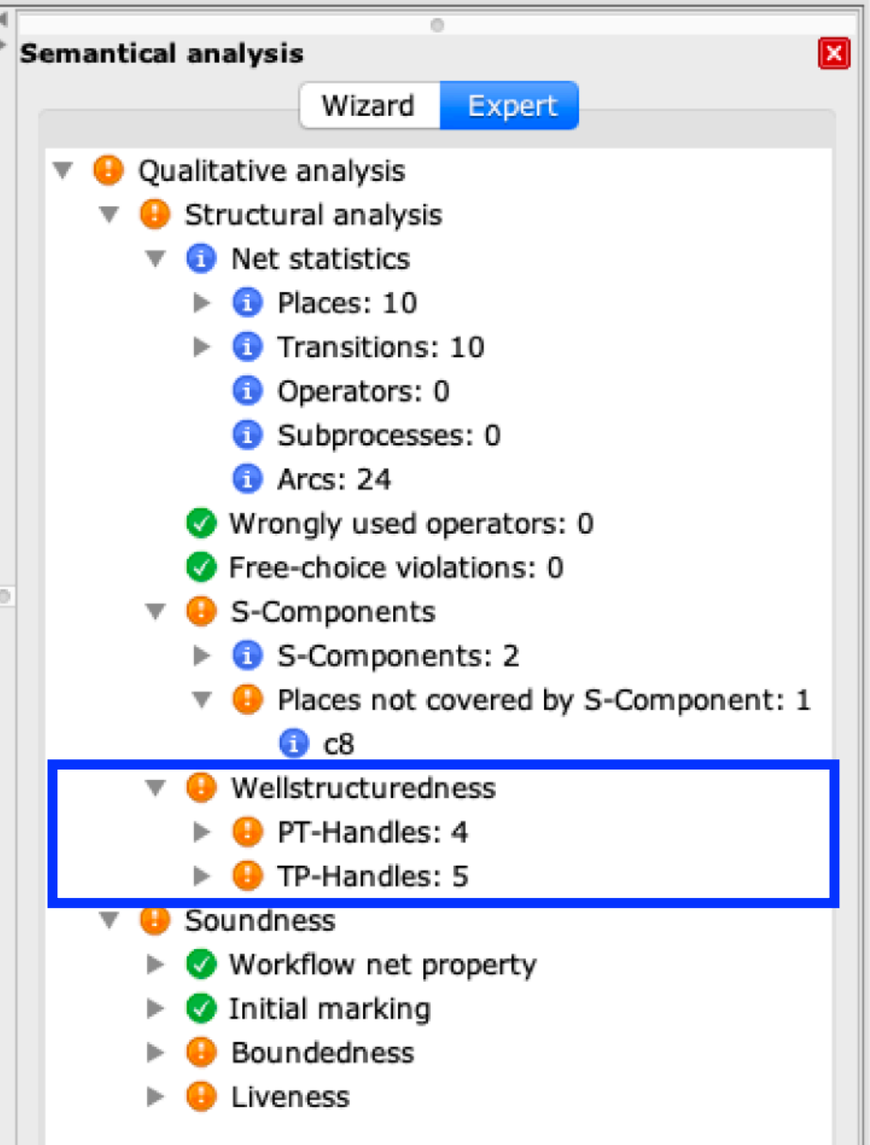




Woped

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





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. $rank(N) = |C_N| - 1$

(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

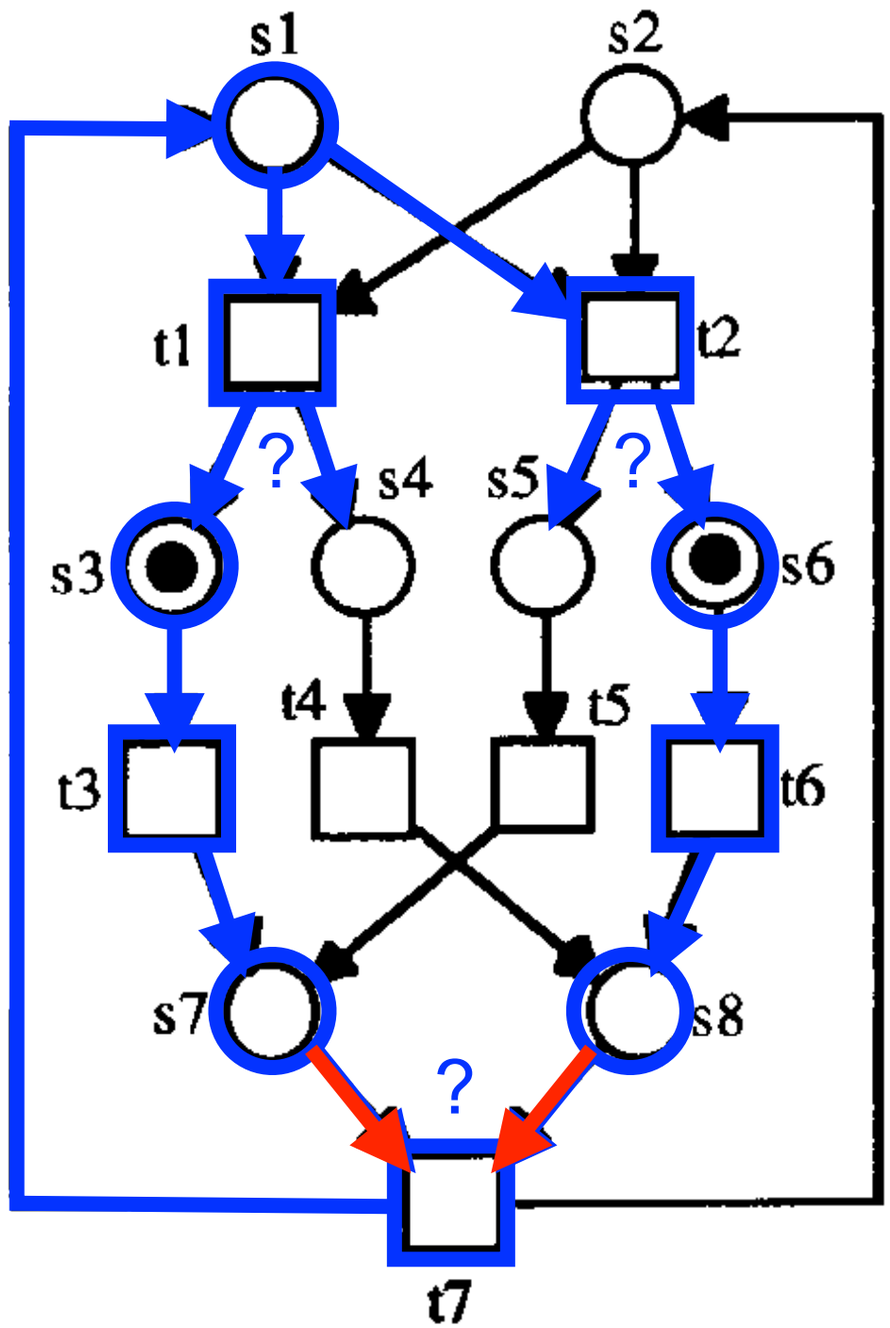
S-cover

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

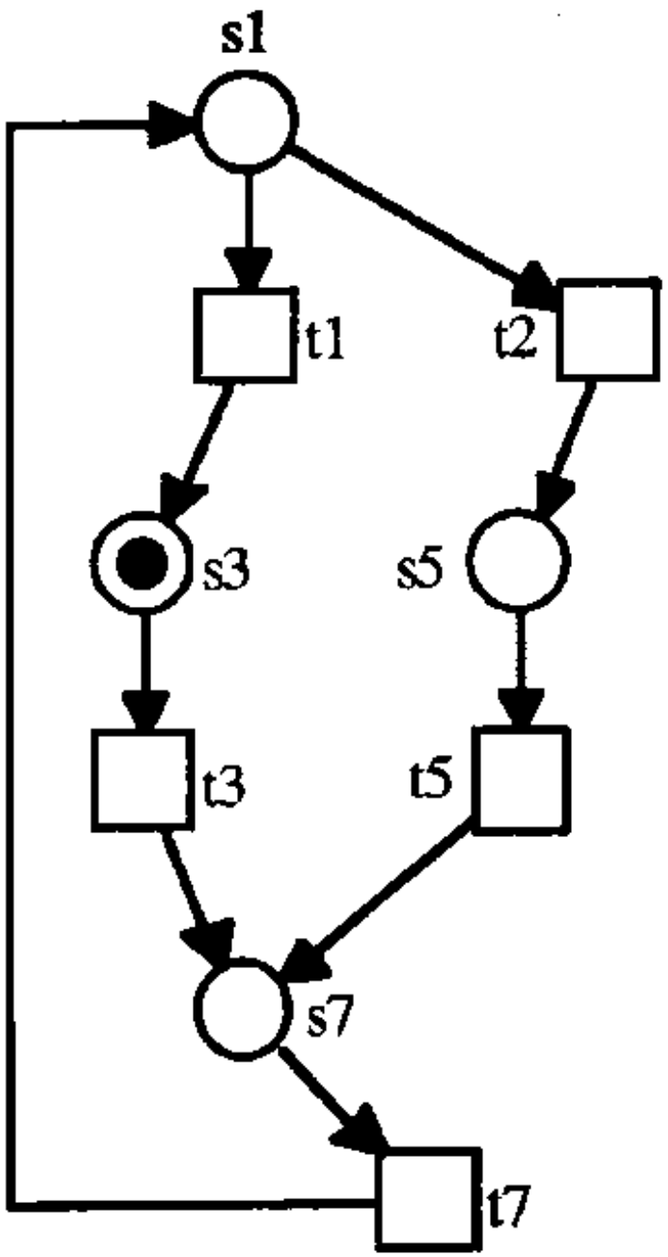
S-cover: example

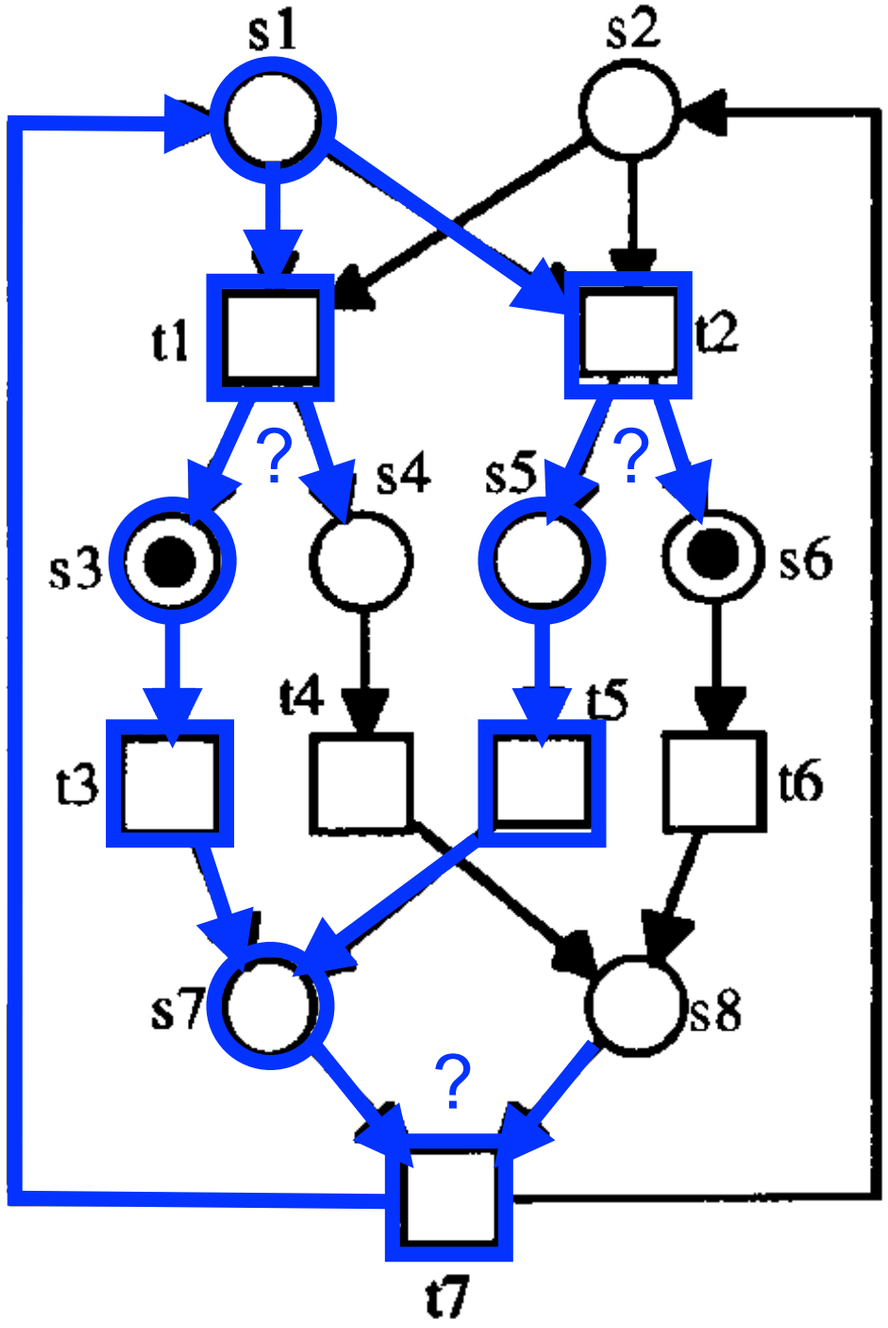


not an S-net

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

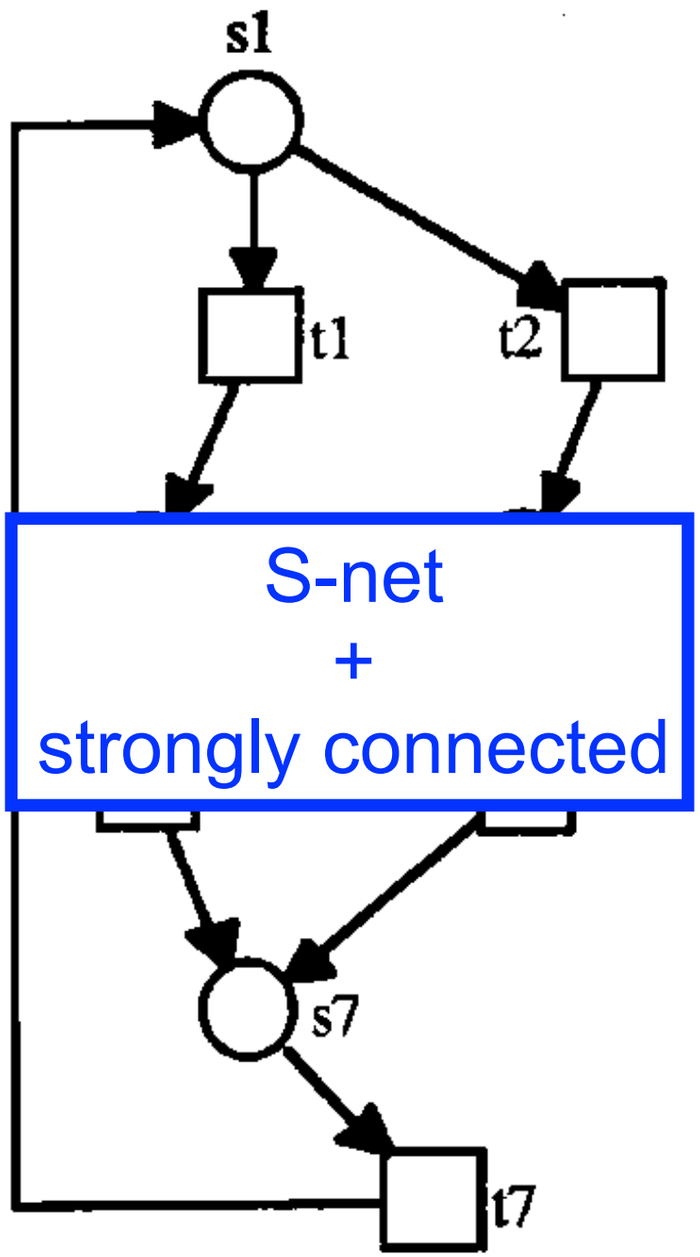
S-cover: example





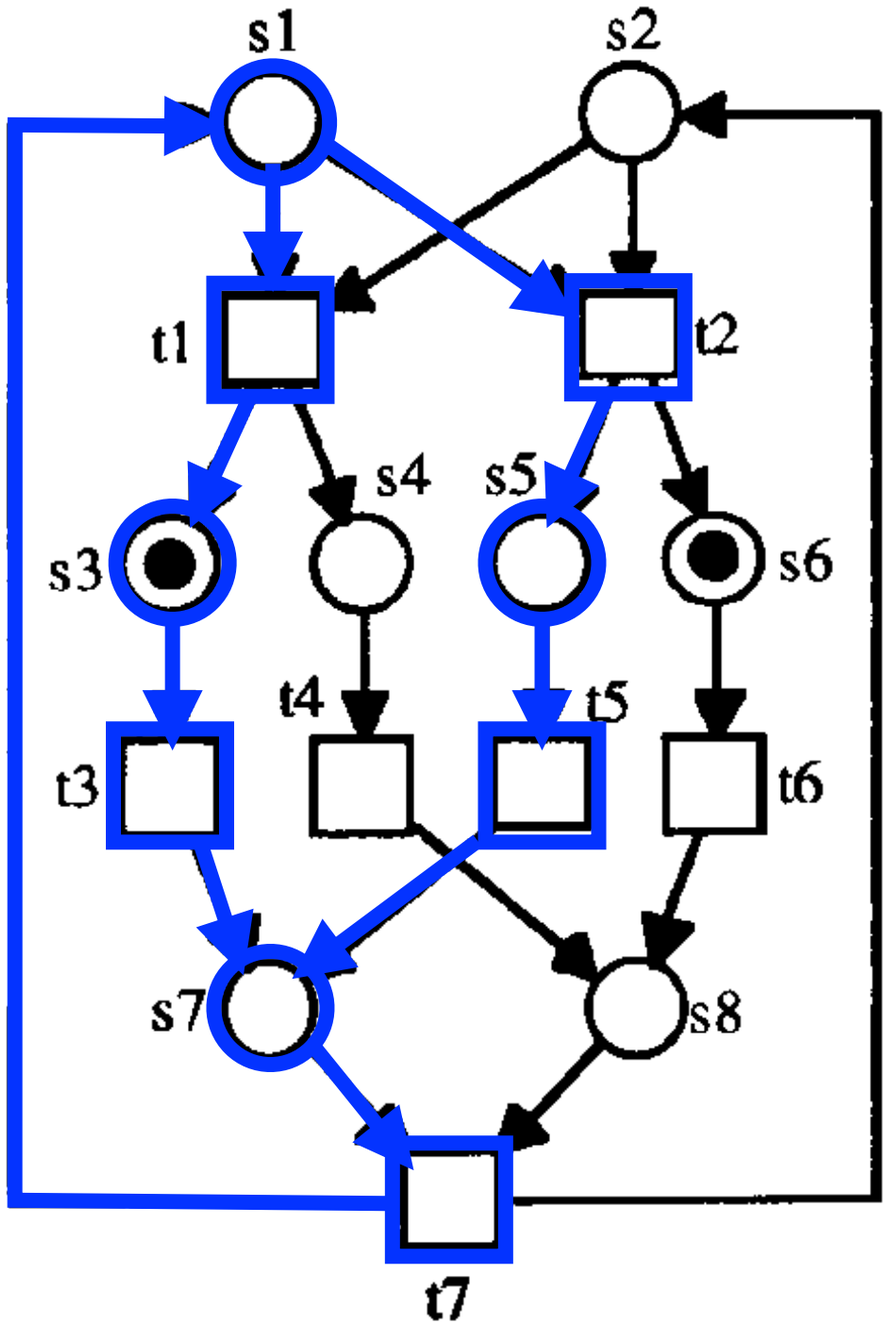
if a place p is taken
then all transitions attached to p must be selected

S-cover: example



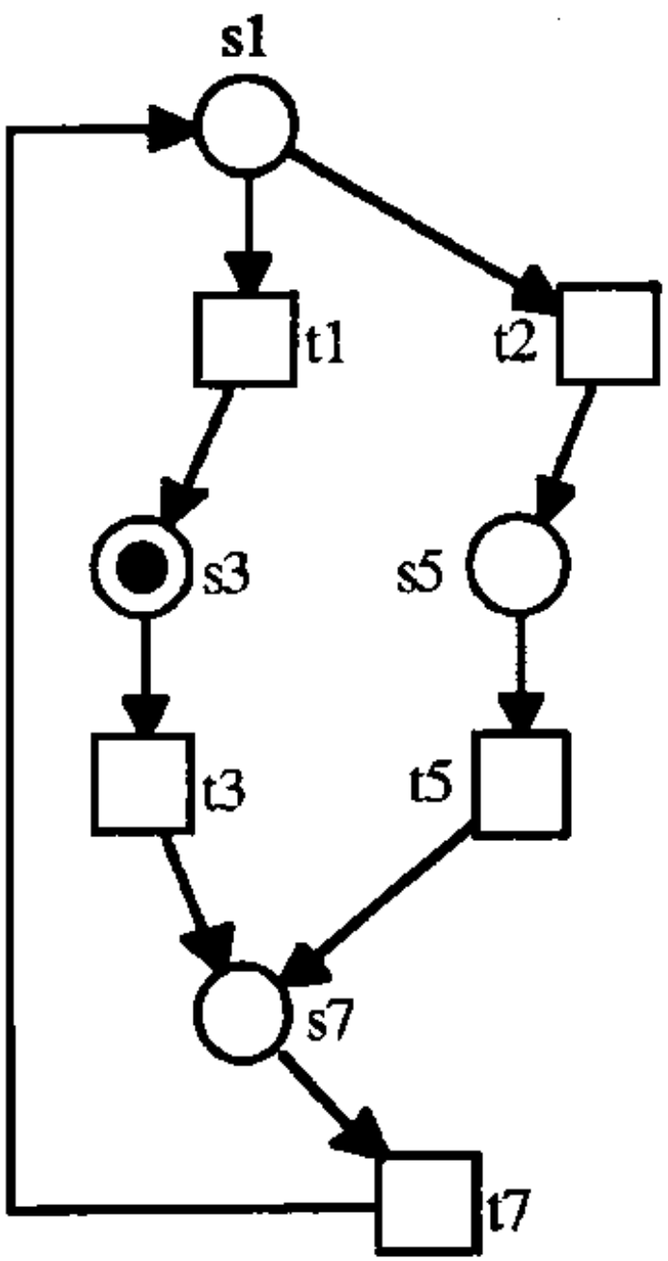
S-net
+
strongly connected

S-component

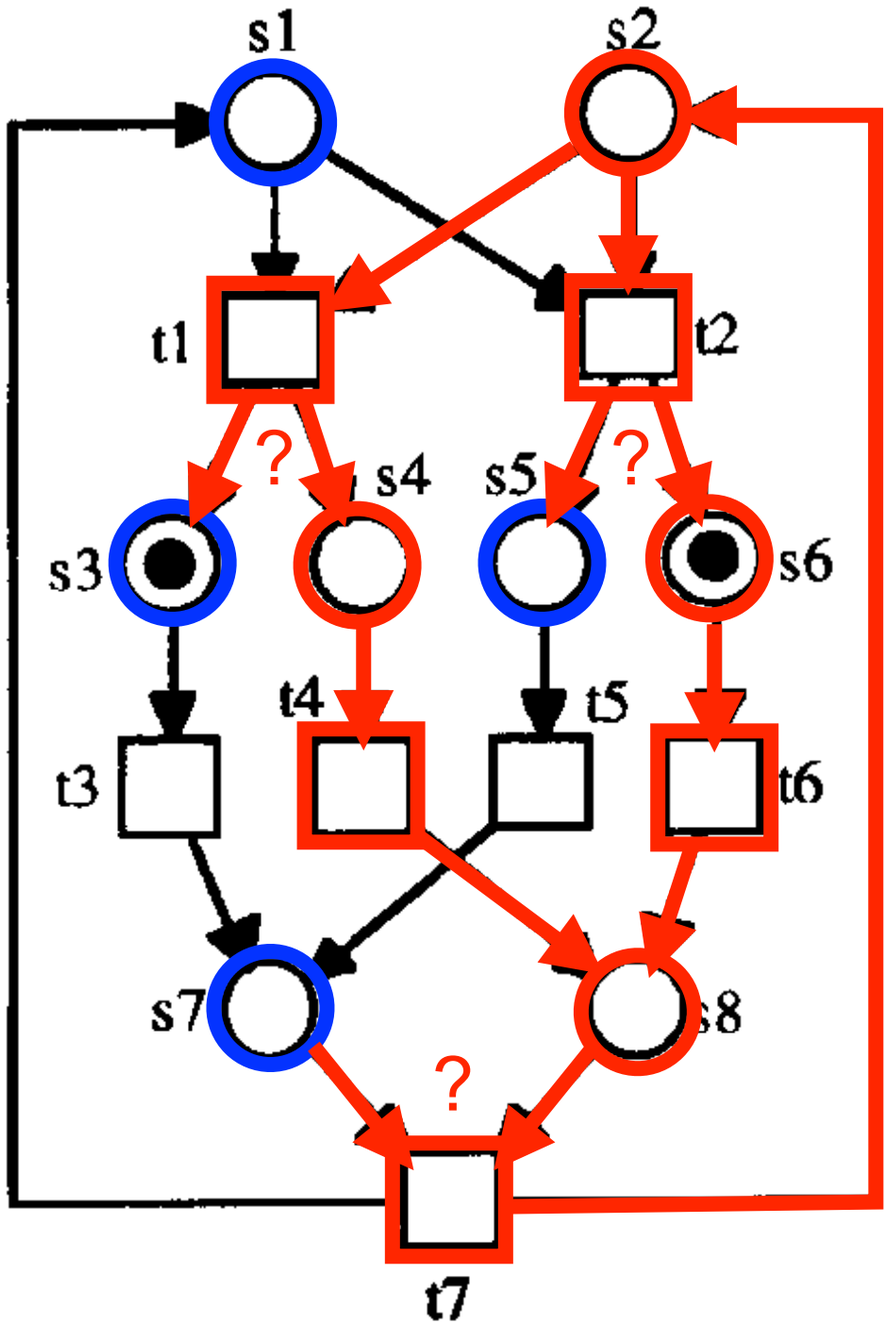


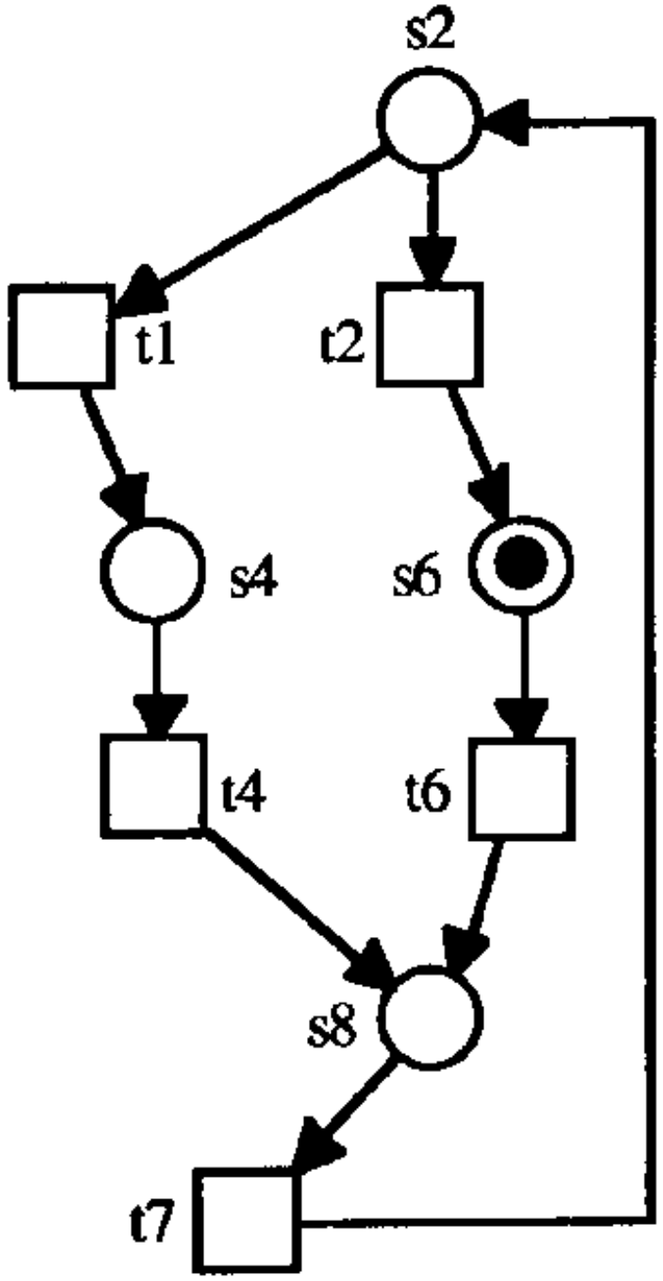
if a place p is taken
then all transitions attached to p must be selected

S-cover: example



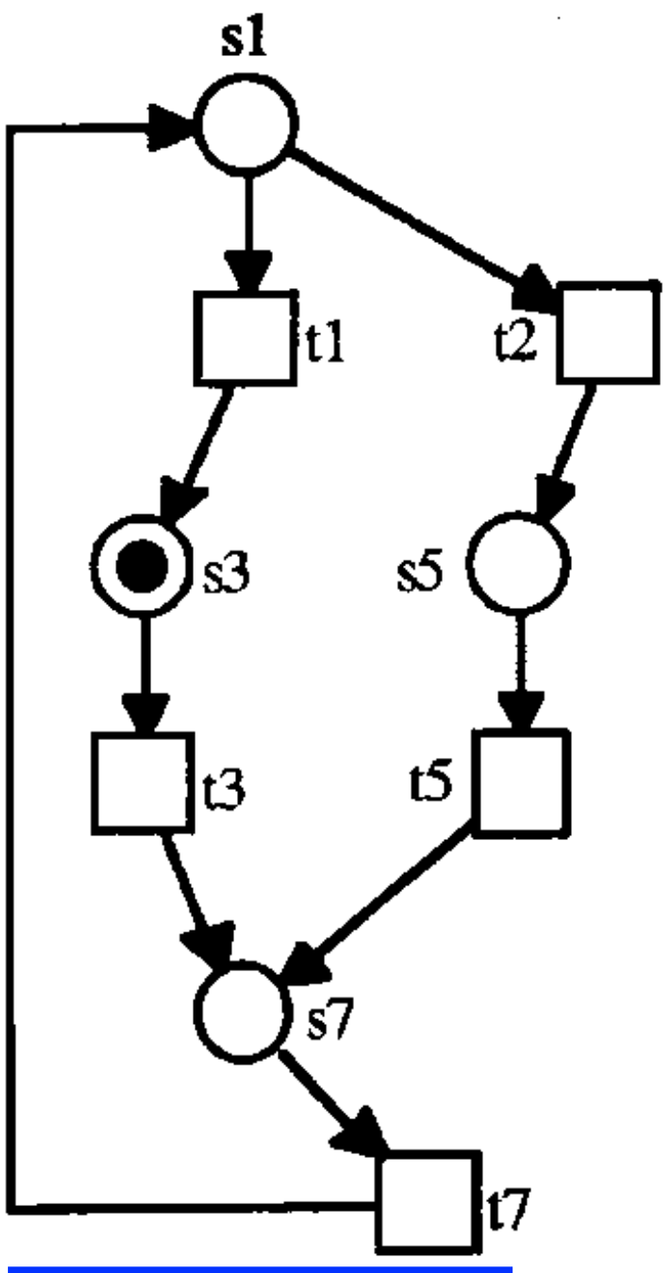
S-component



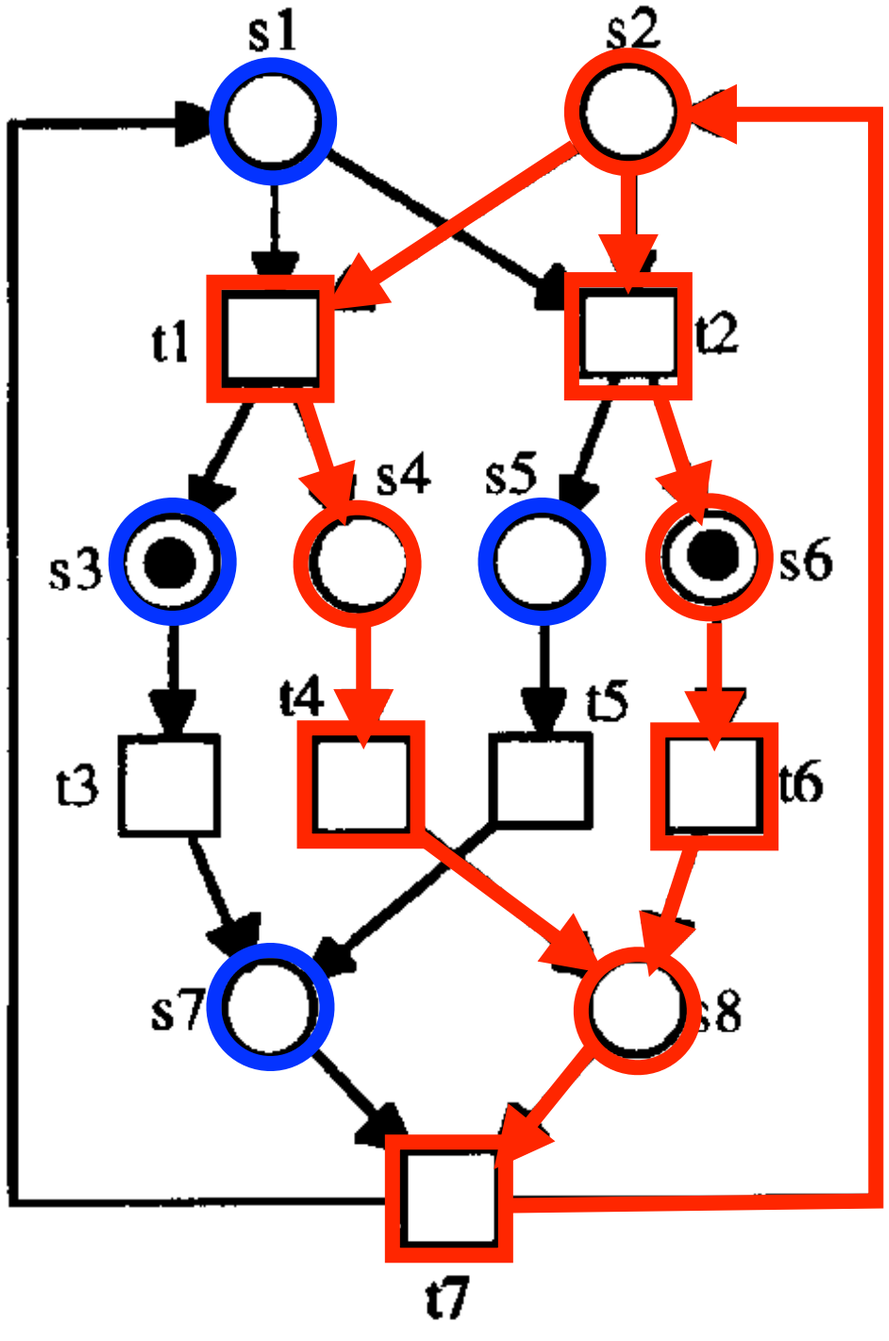


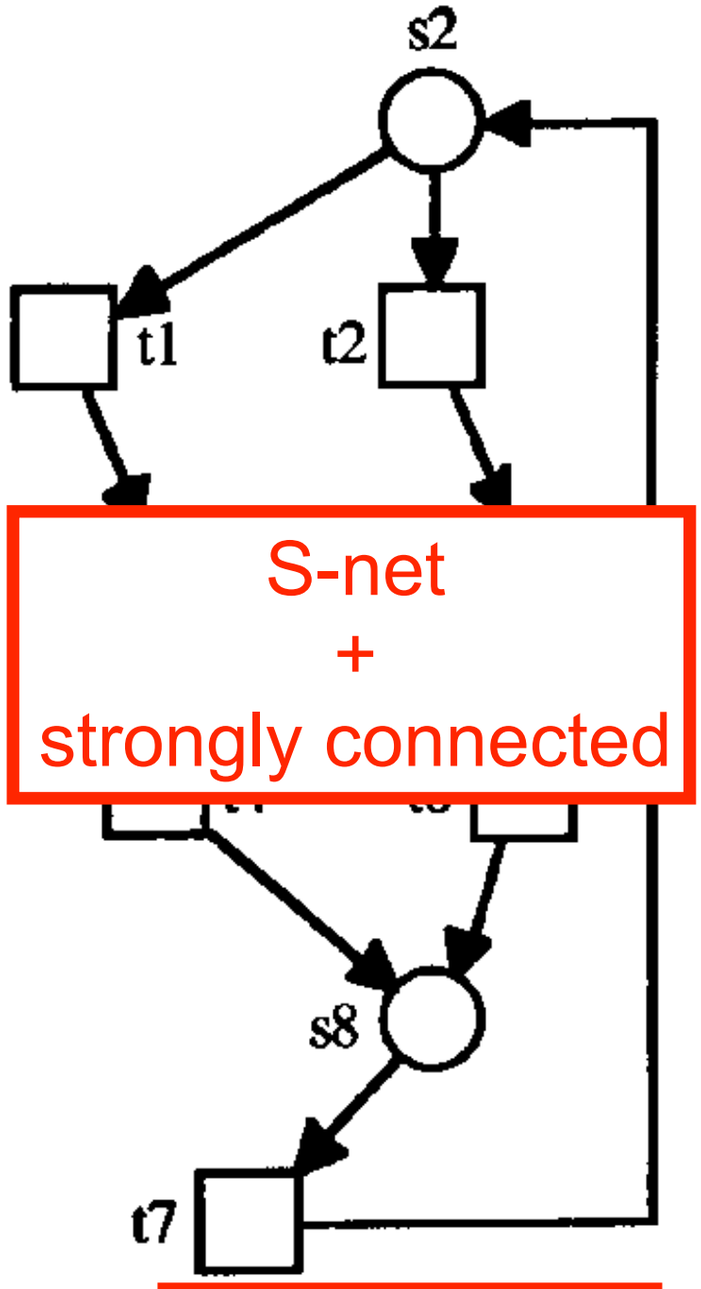
if a place p is taken
then all transitions attached to p must be selected

S-cover: example



S-component

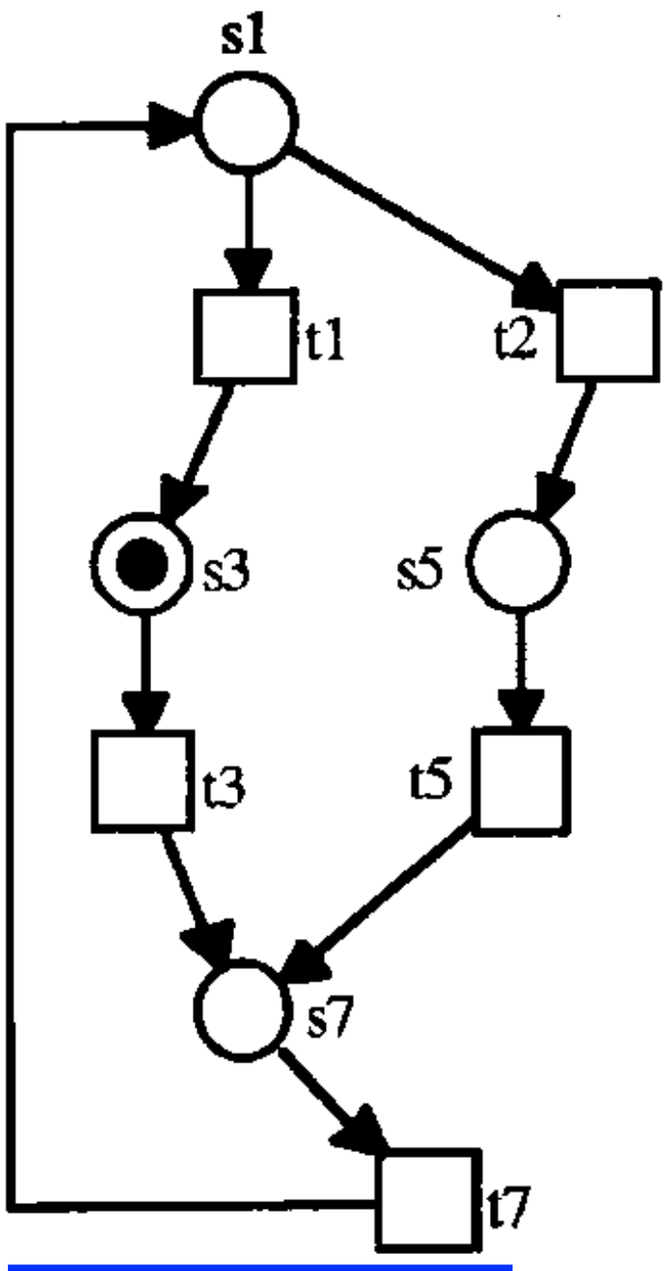




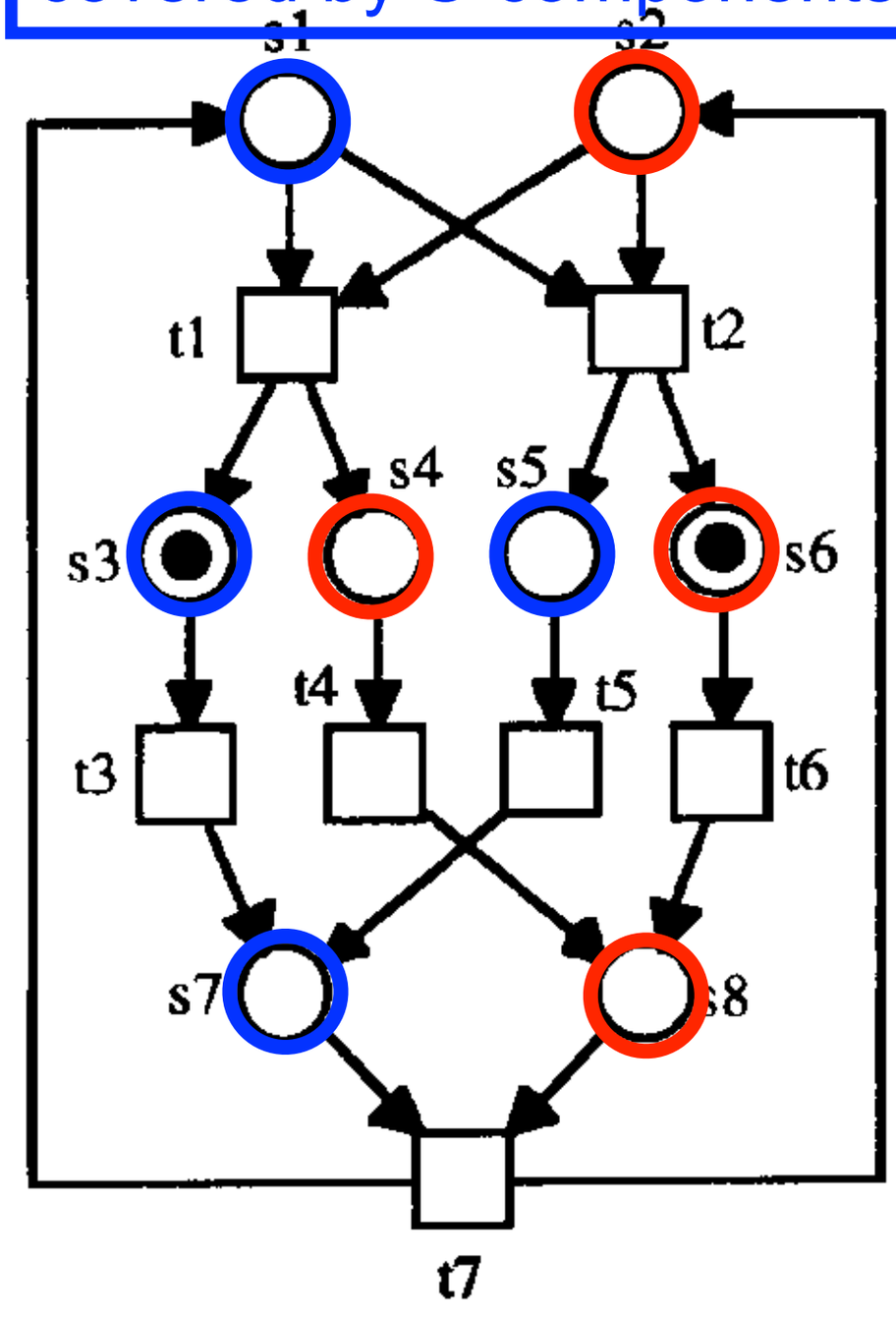
S-component

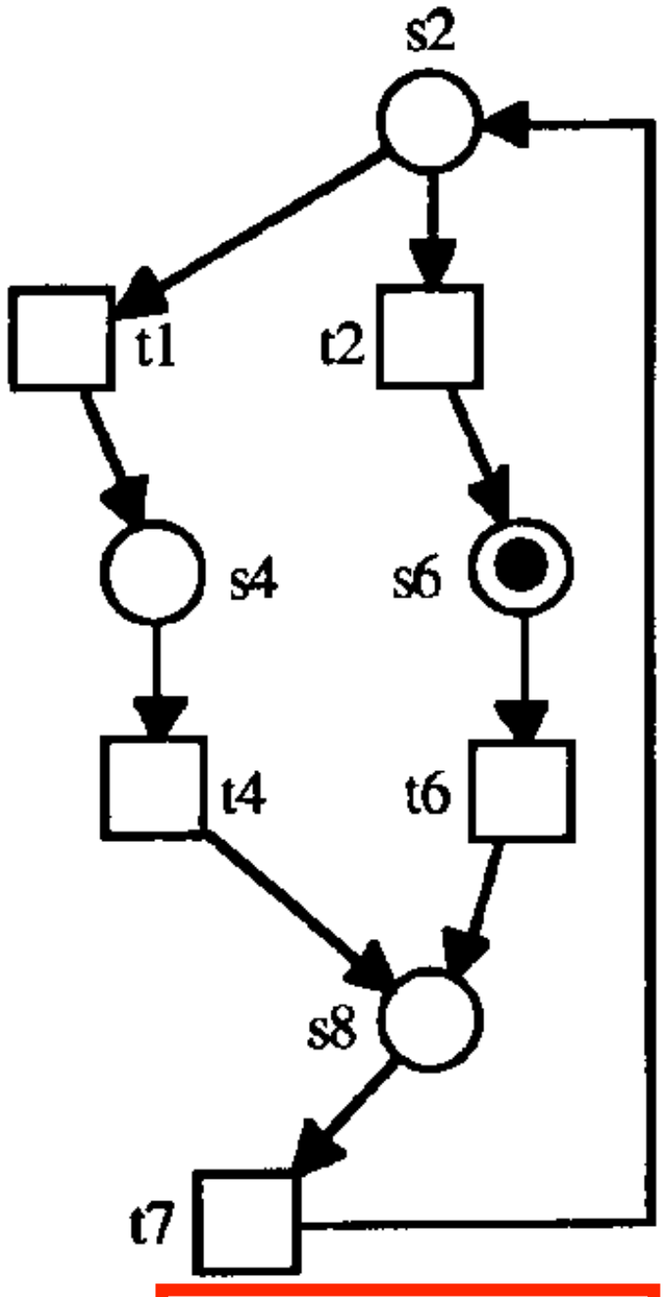
S-cover: example

covered by S-components



S-component

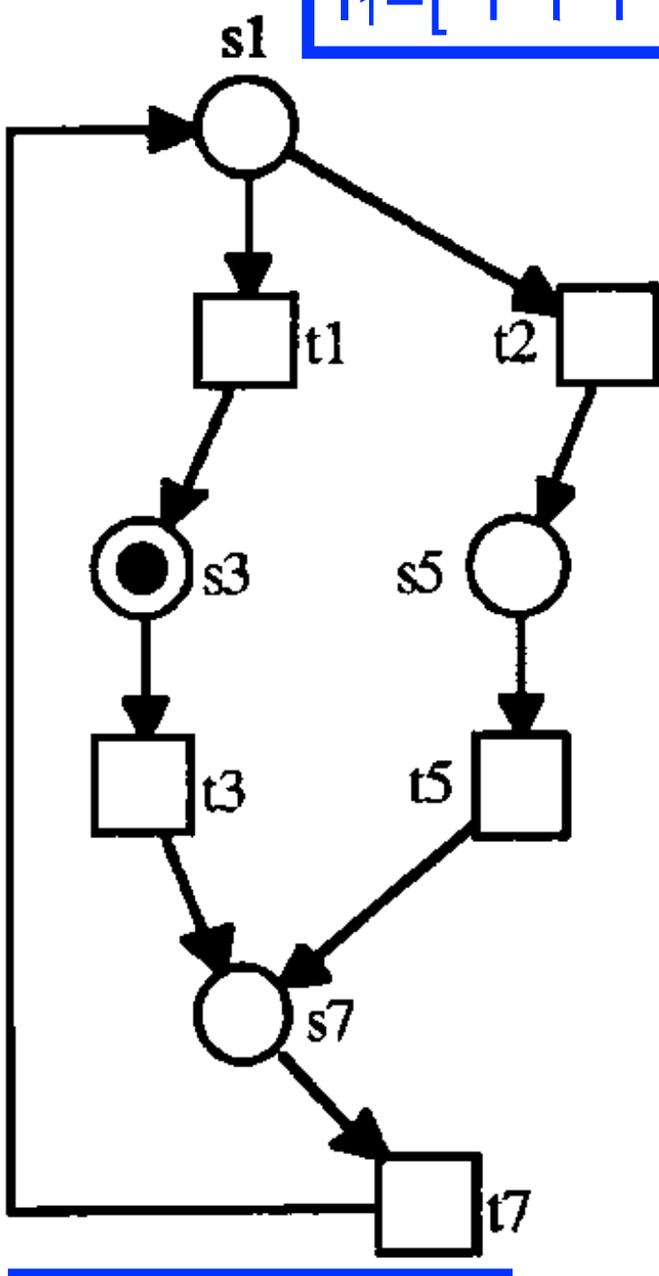




S-component

S-cover: example

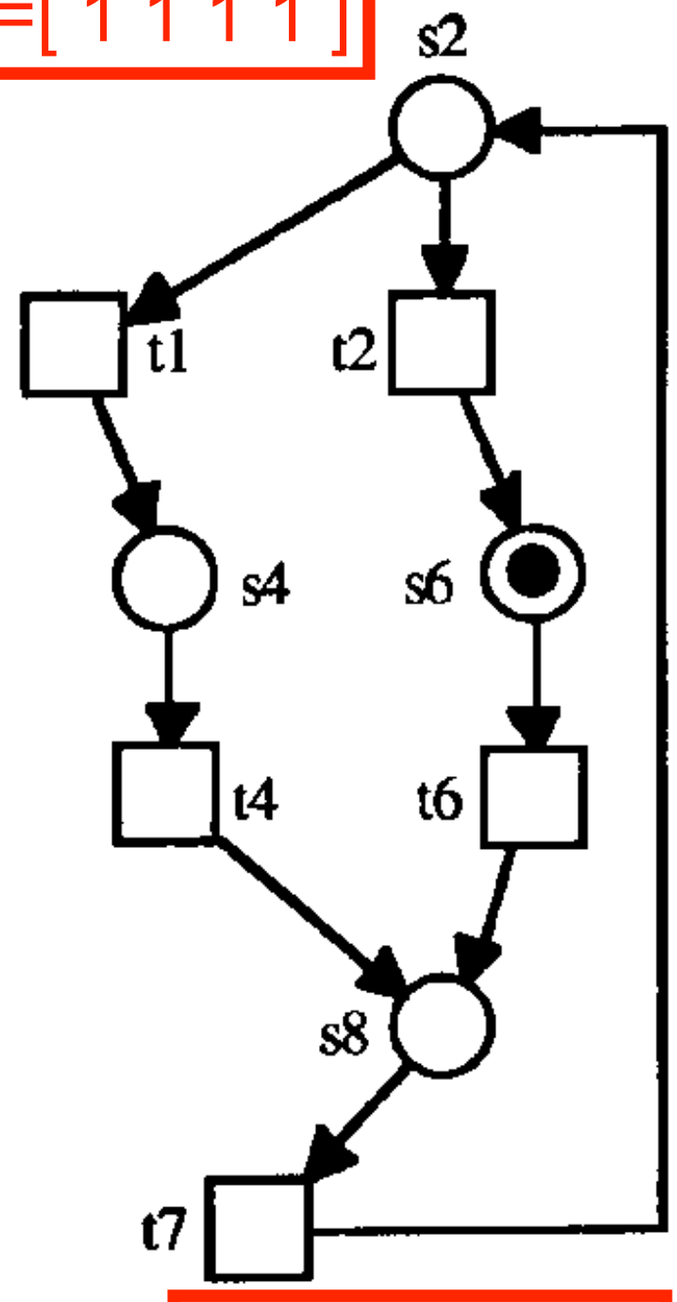
I_1=[1 1 1 1]



S-component

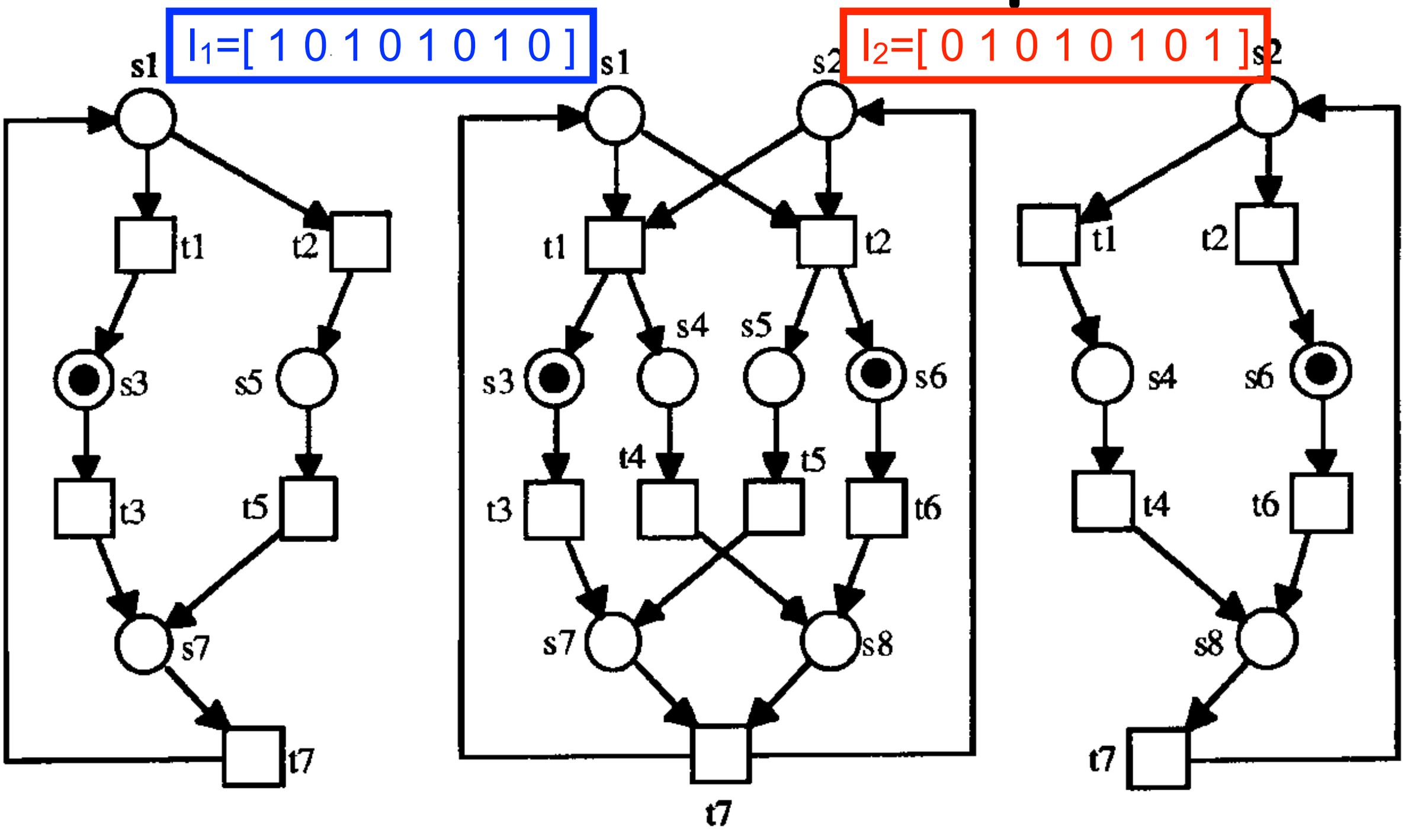
S-cover

I_2=[1 1 1 1]



S-component

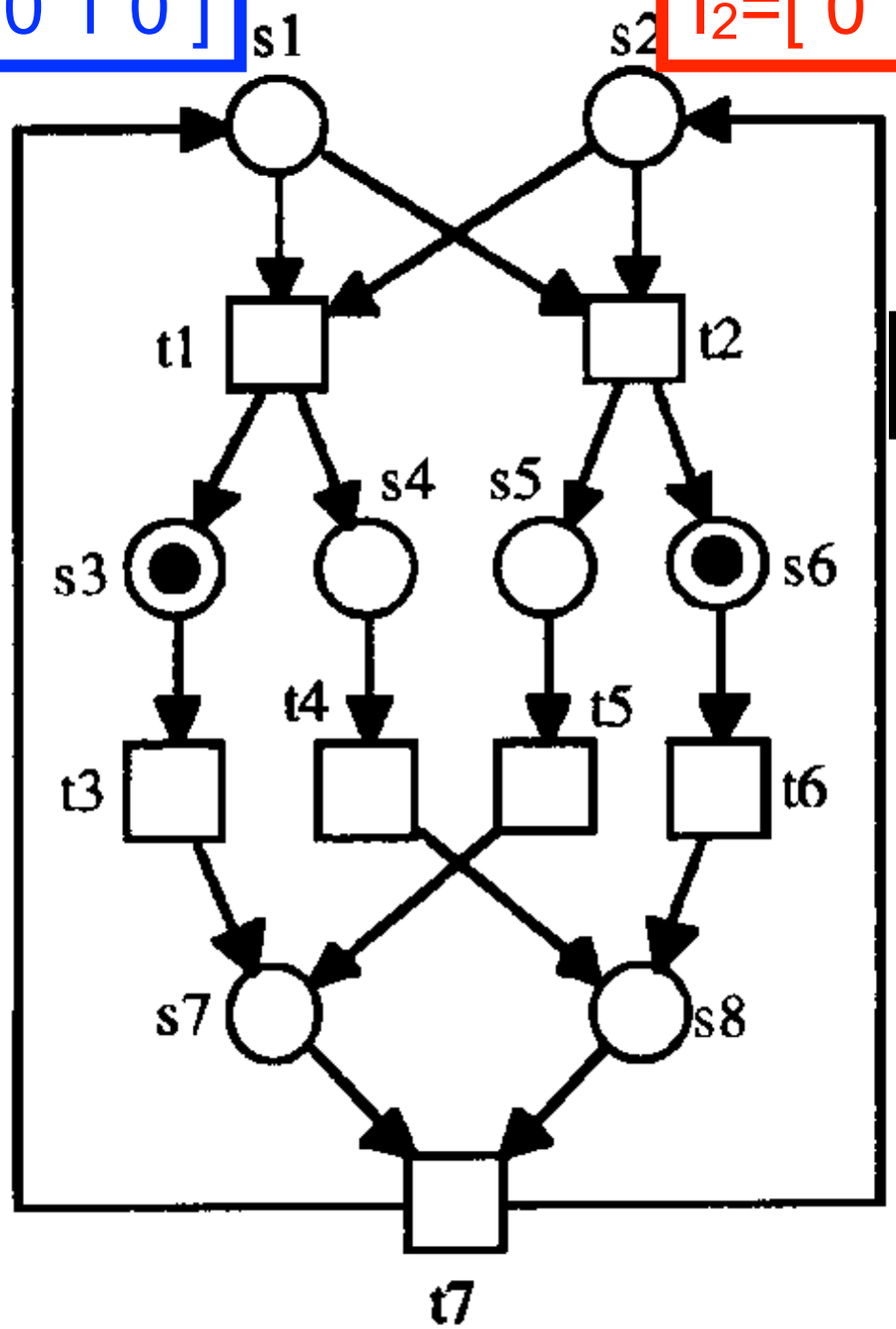
S-cover: example



S-cover: example

I_1=[1 0 1 0 1 0 1 0]

I_2=[0 1 0 1 0 1 0 1]



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

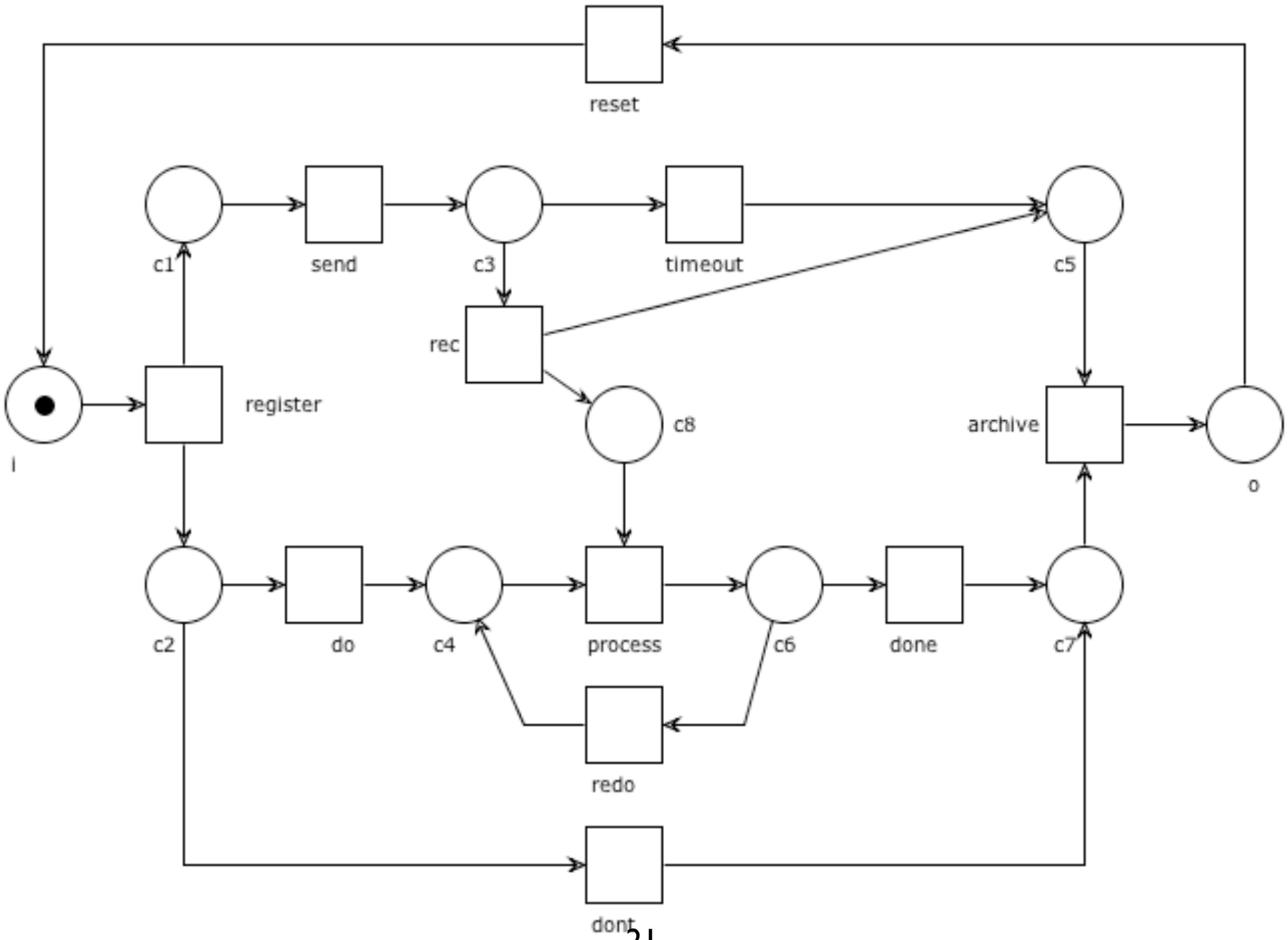
N is sound iff N* is live and bounded (Main Theorem)
N is free-choice iff N* is free-choice

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

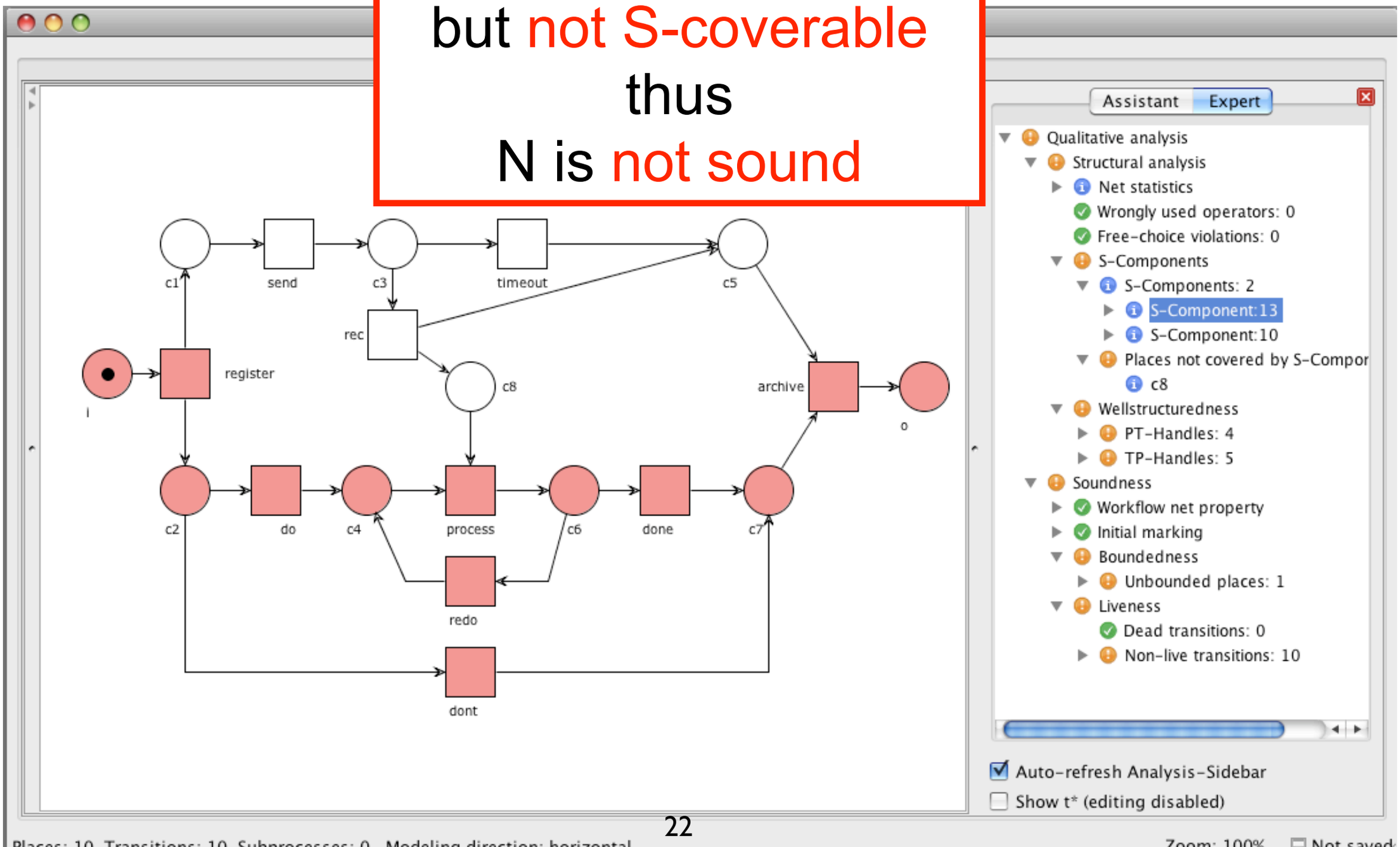
N free-choice + N* not S-coverable => N not sound

S-cover for N* ?


reset
c1
send
c3
timeout
c5
rec
register
c8
archive
i
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



Be careful

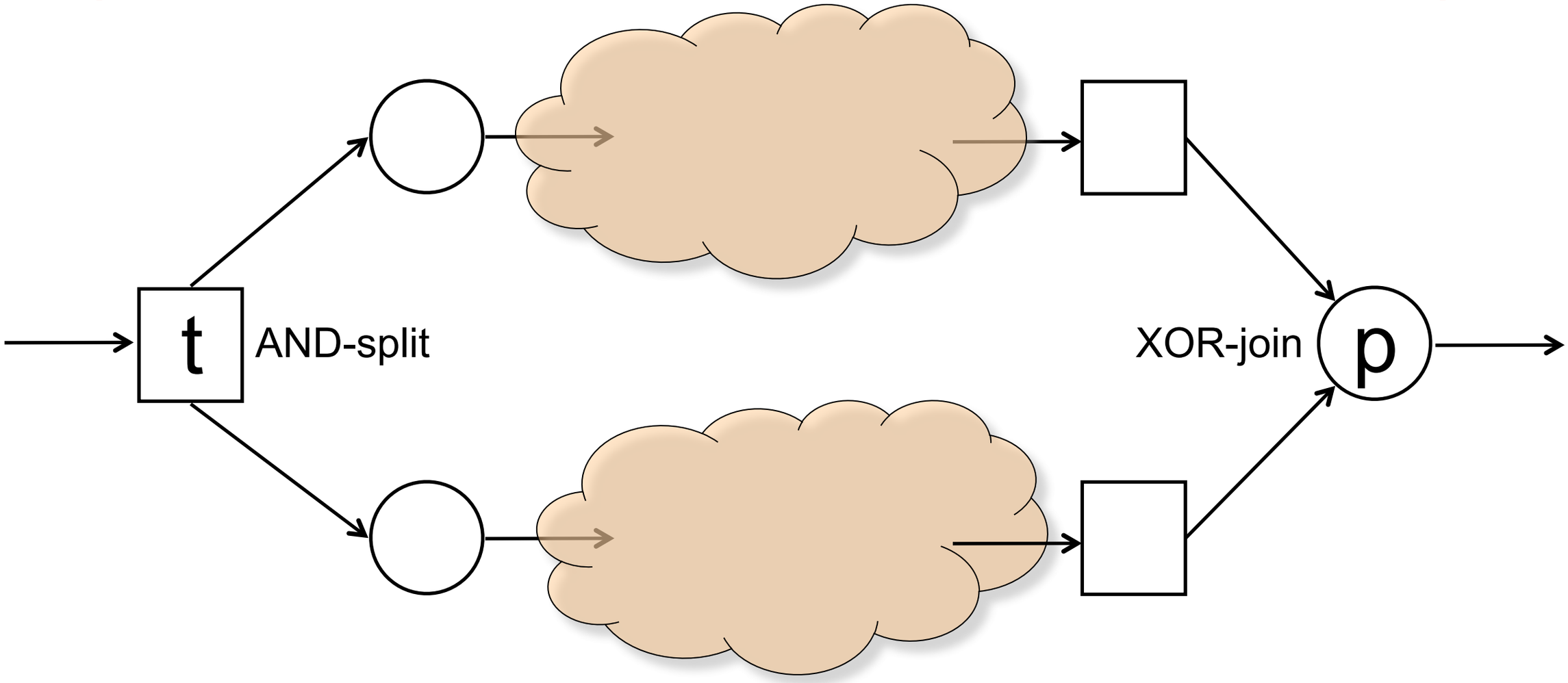
reset transition is implicit in WoPeD

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

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)

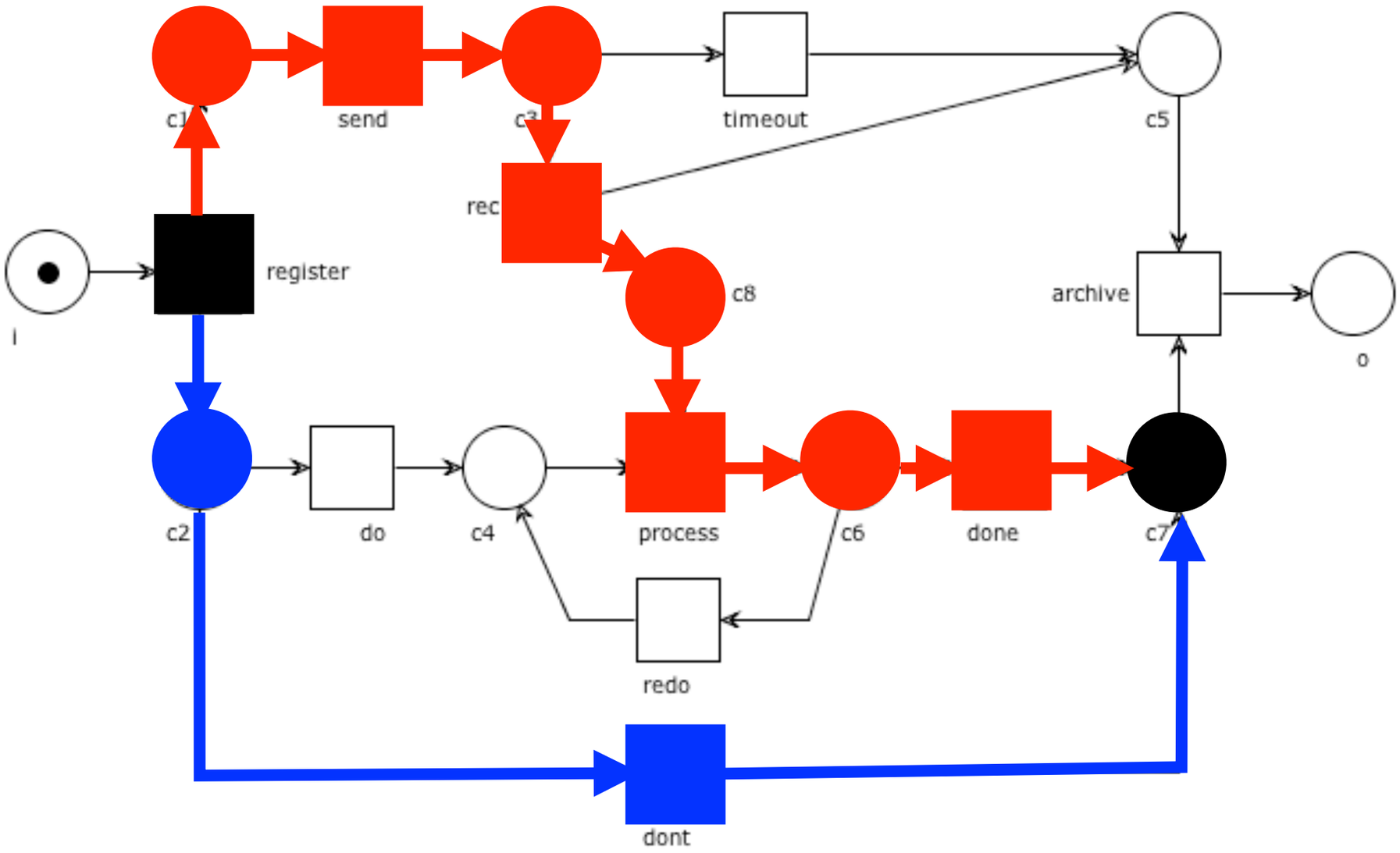


TP-handles

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

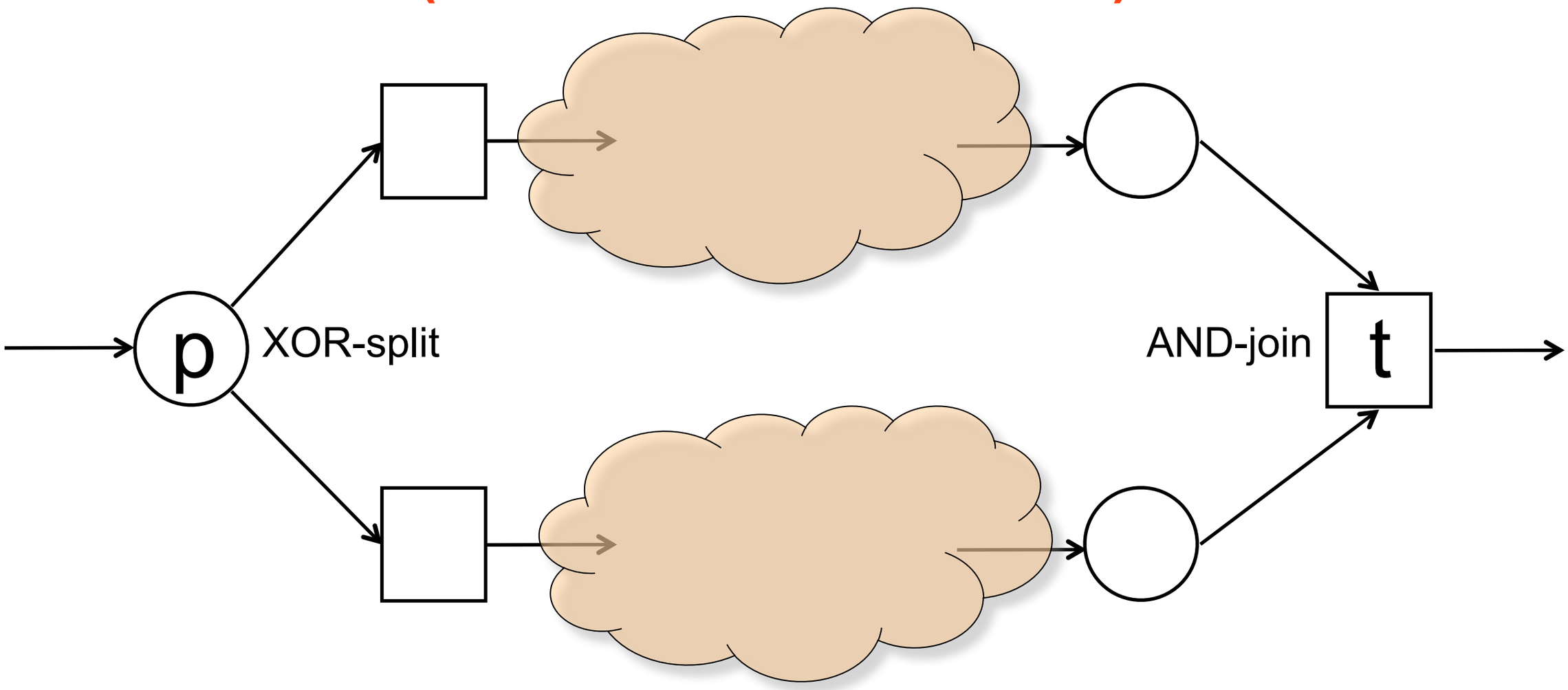
Example: TP-handle



PT-handles

Two alternative flows created via a XOR-split should not be synchronized by an AND-join

(the net could deadlock)

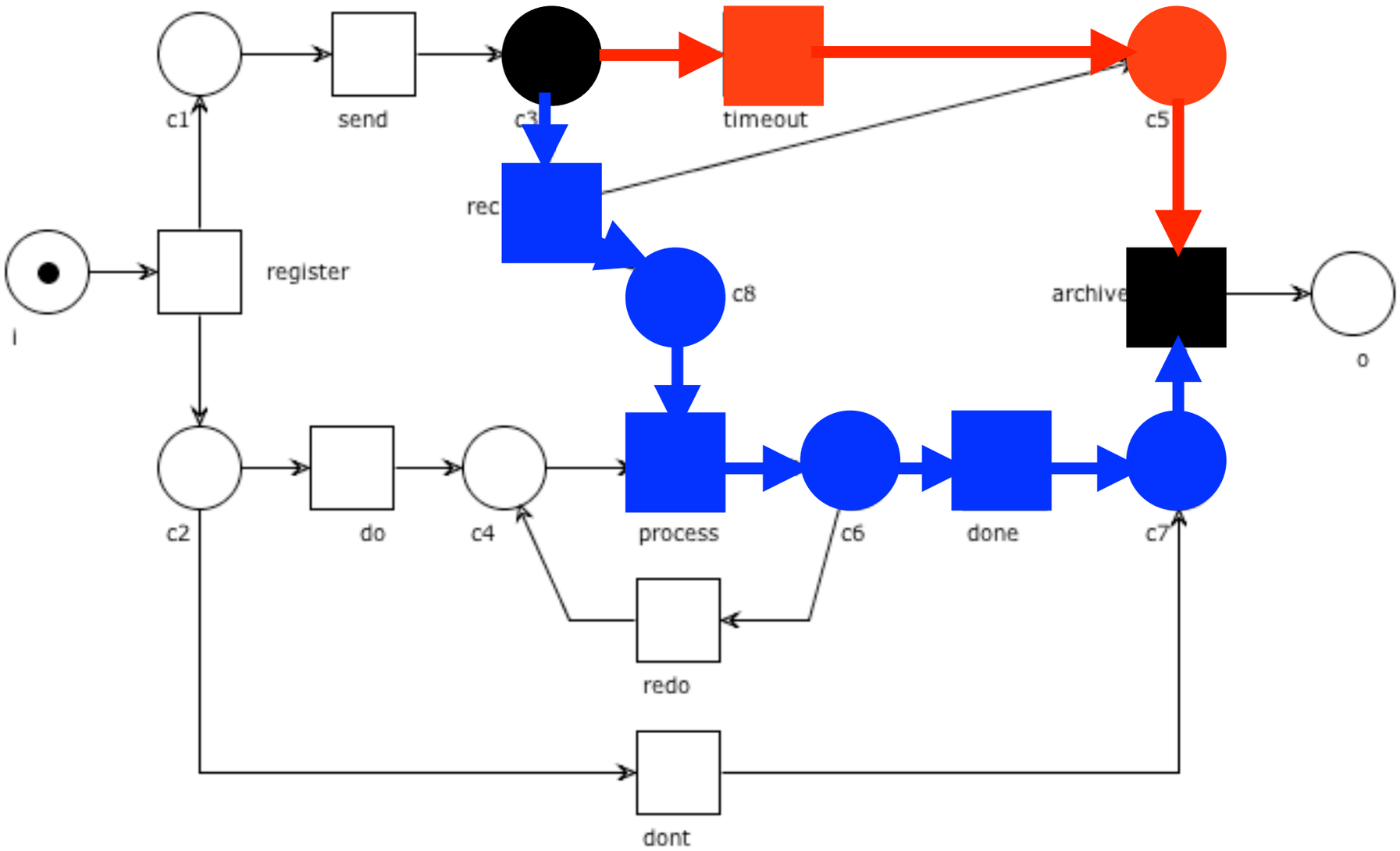


PT-handles

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



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

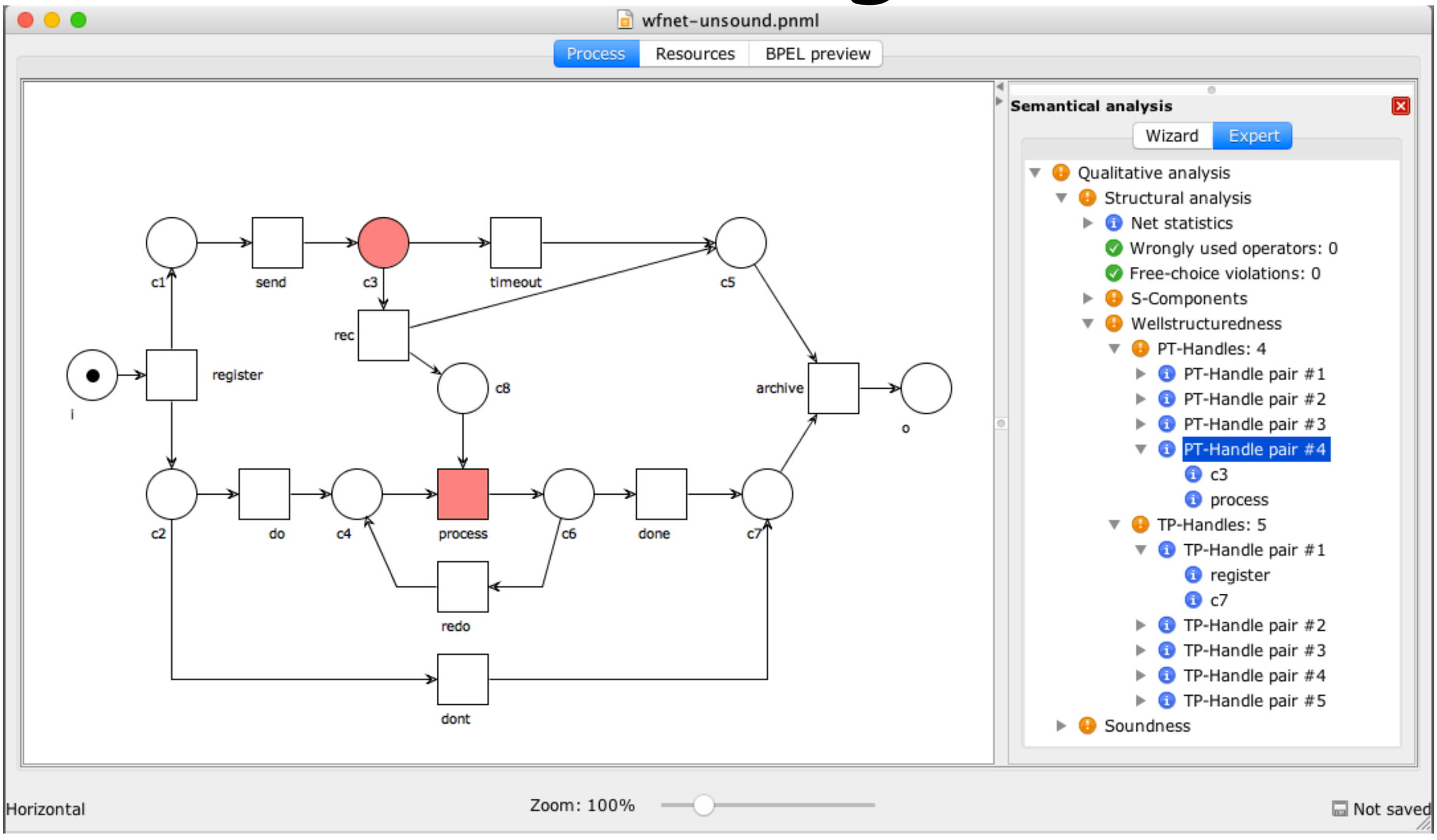
Be careful

N well-structured = N* well-handled

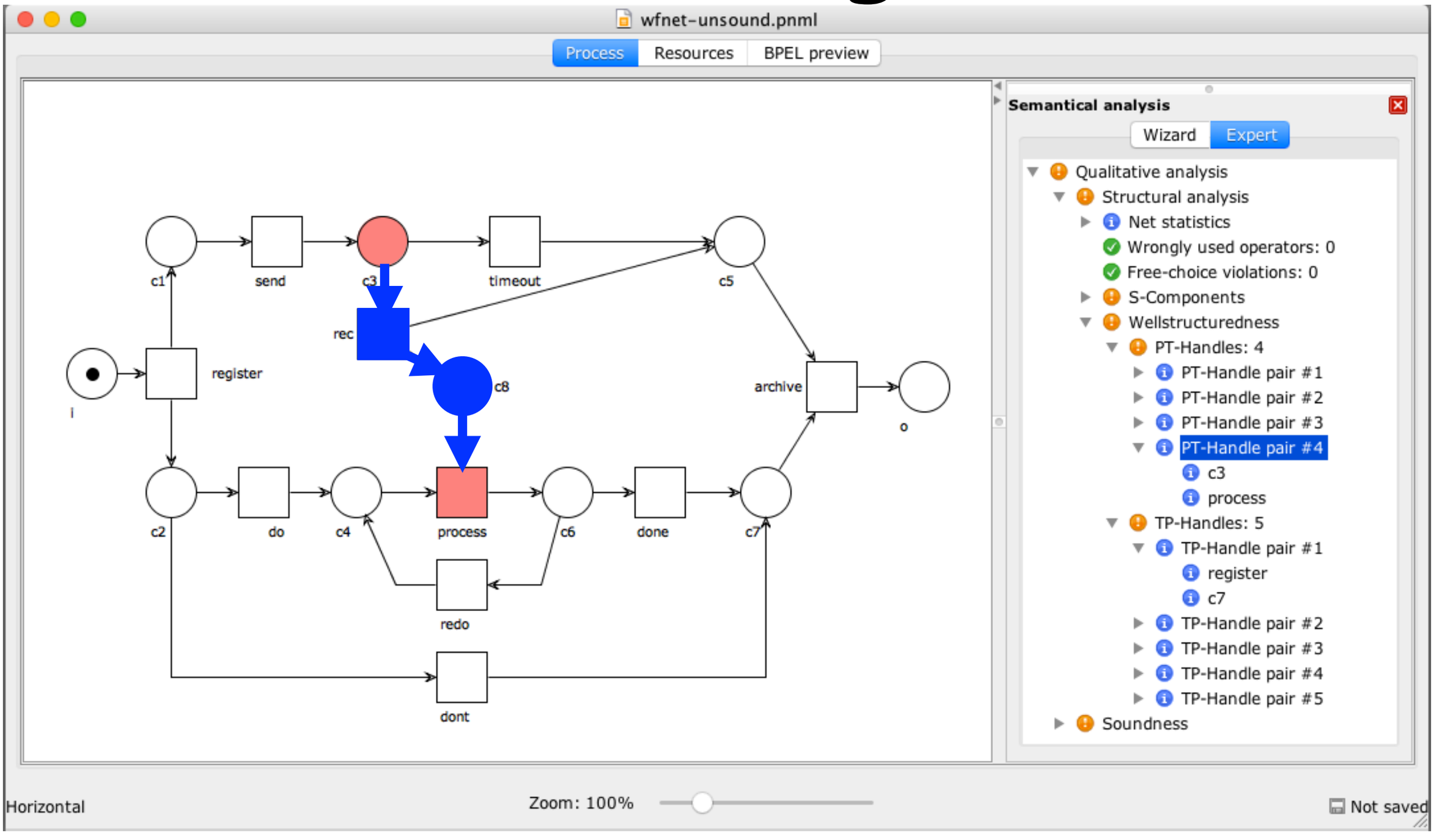
reset transition is implicit in WoPeD

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

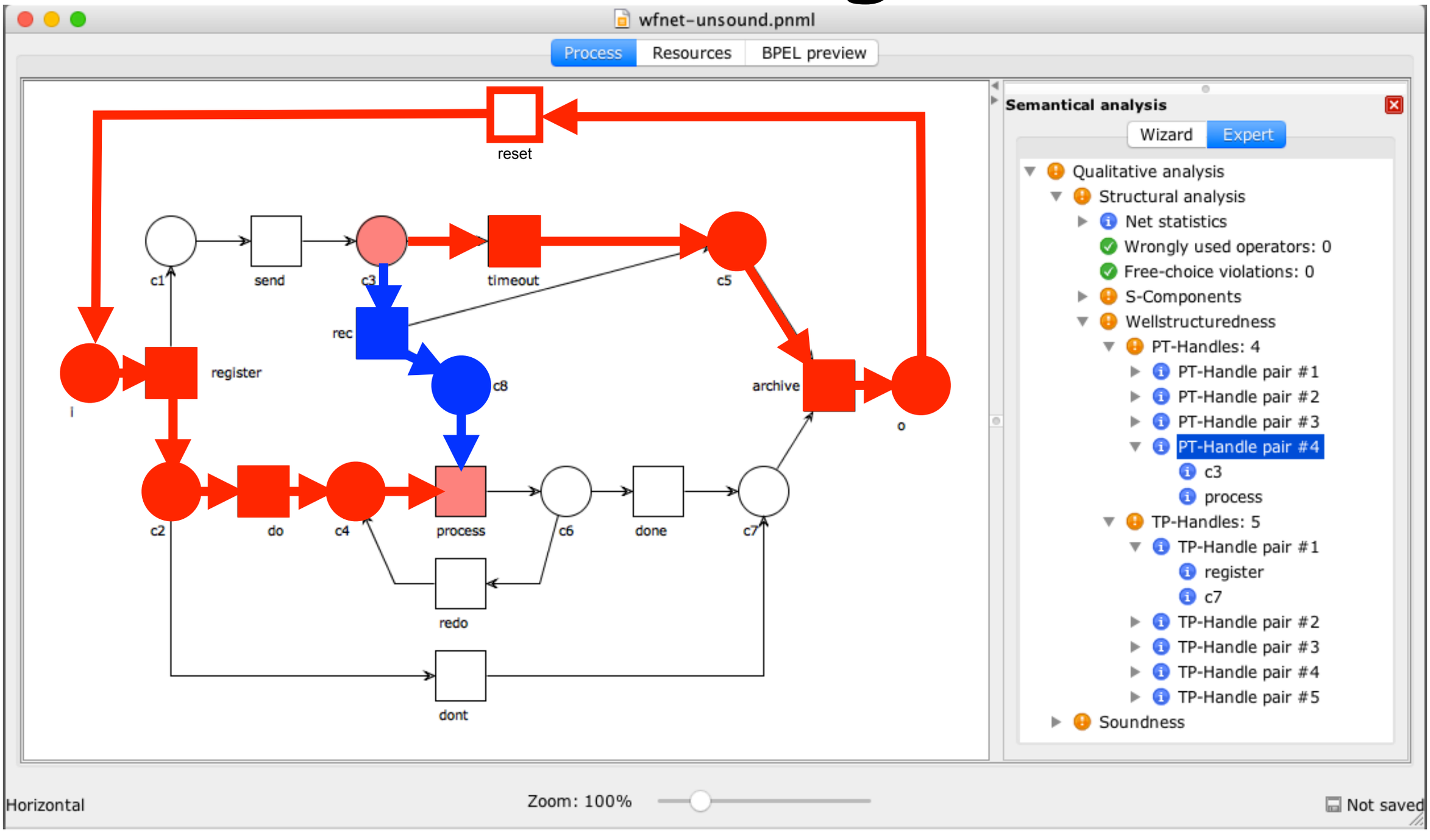
WoPeD Diagnosis



WoPeD Diagnosis



WoPeD Diagnosis



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

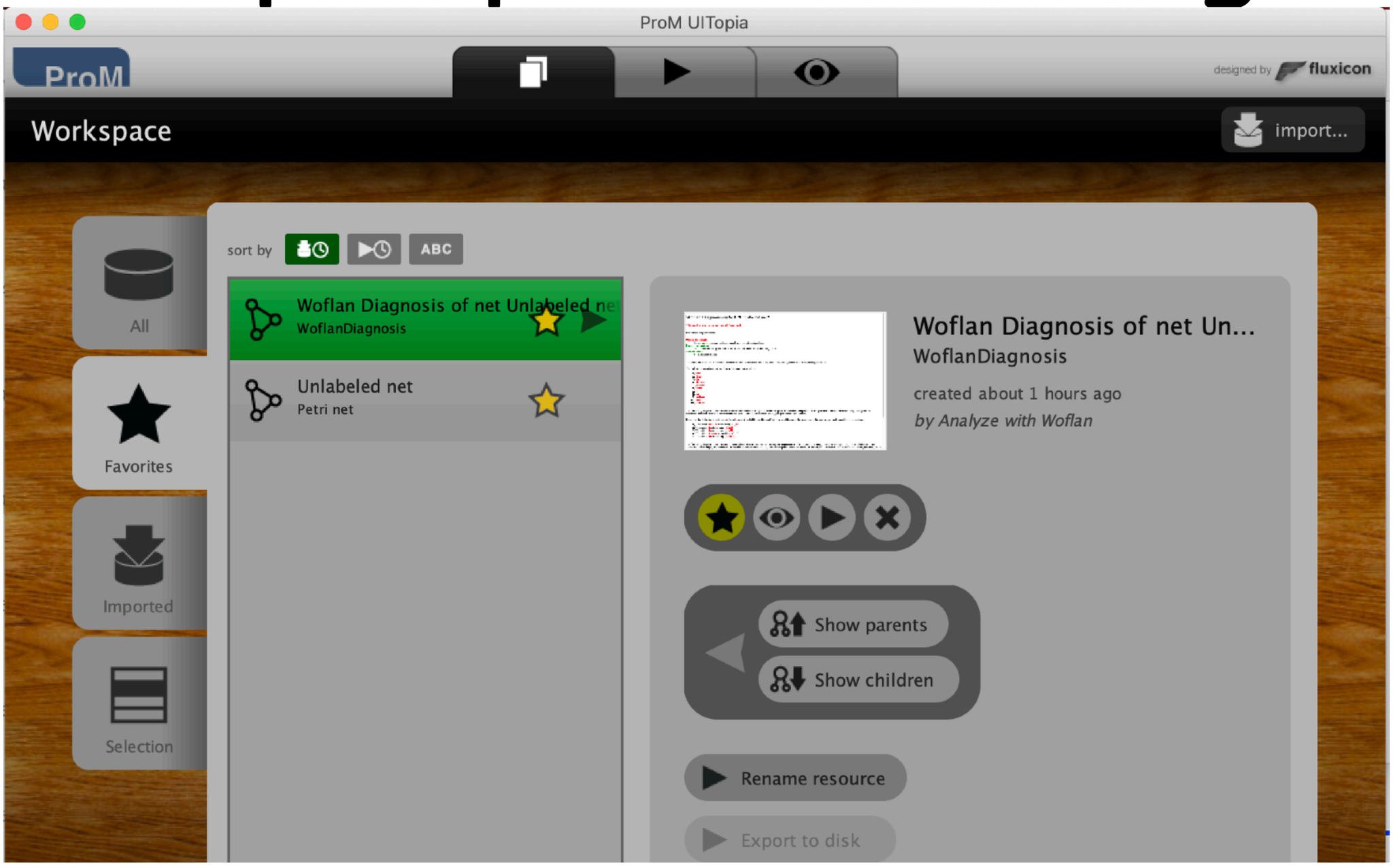
http://www.win.tue.nl/woflan/

WOrkFLow ANalyzer
(Microsoft Windows only)

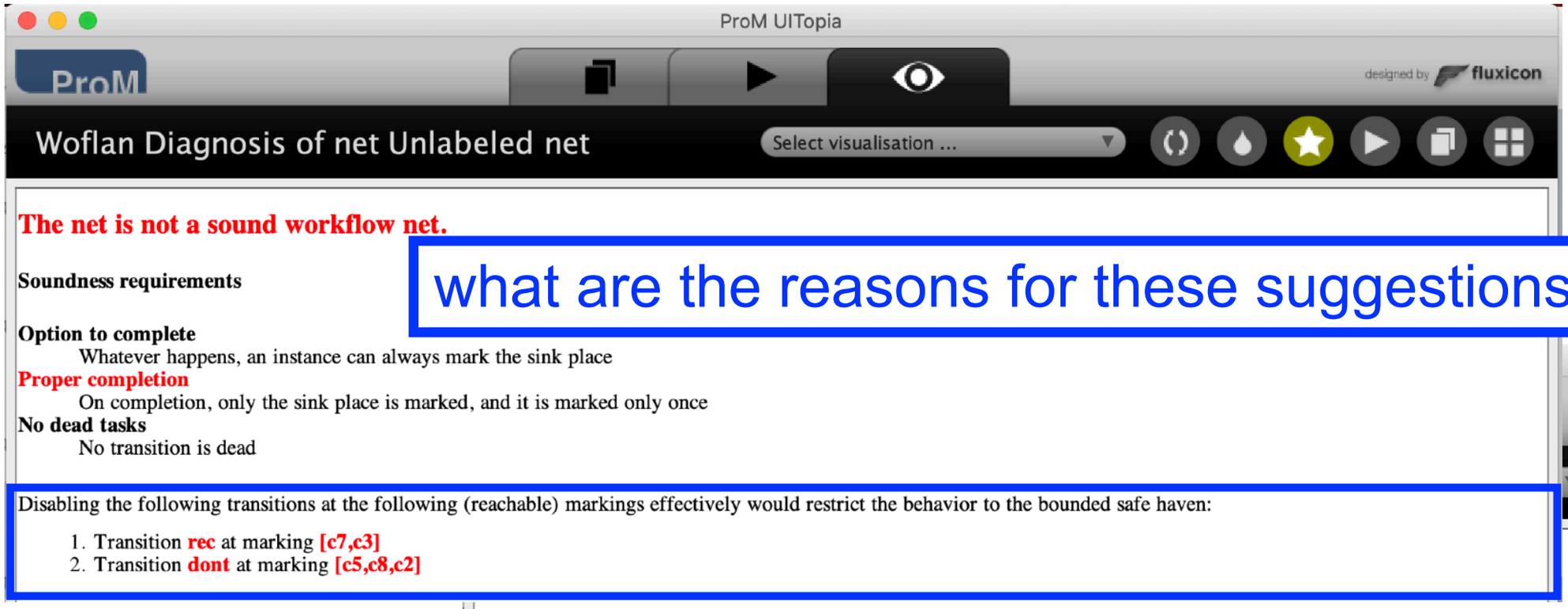
Woflan tells us if N is a sound workflow net
(Is N a workflow net? Is N* bounded? Is N* live?)
if not, provides some diagnostic information

Woflan now a ProM plugin

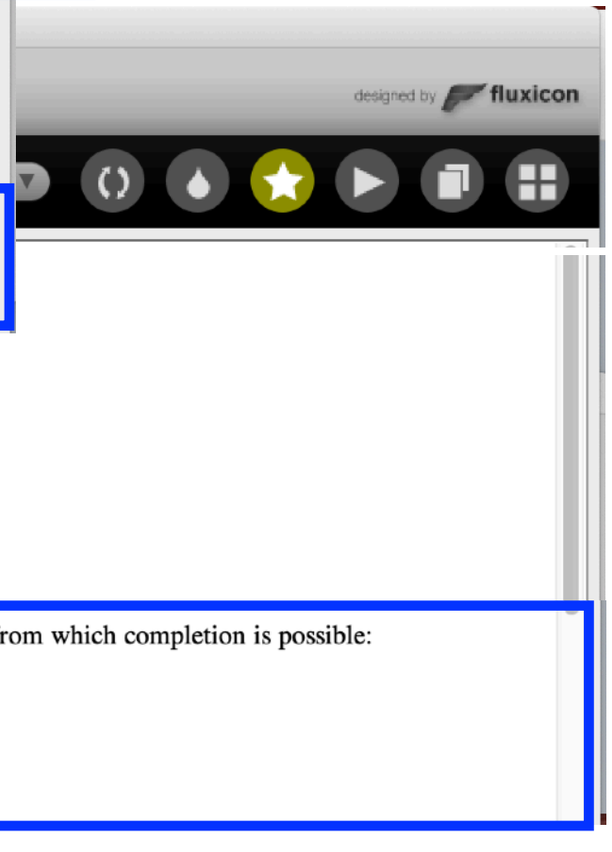
http://promtools.org/



Woflan (in ProM)



what are the reasons for these suggestions?



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

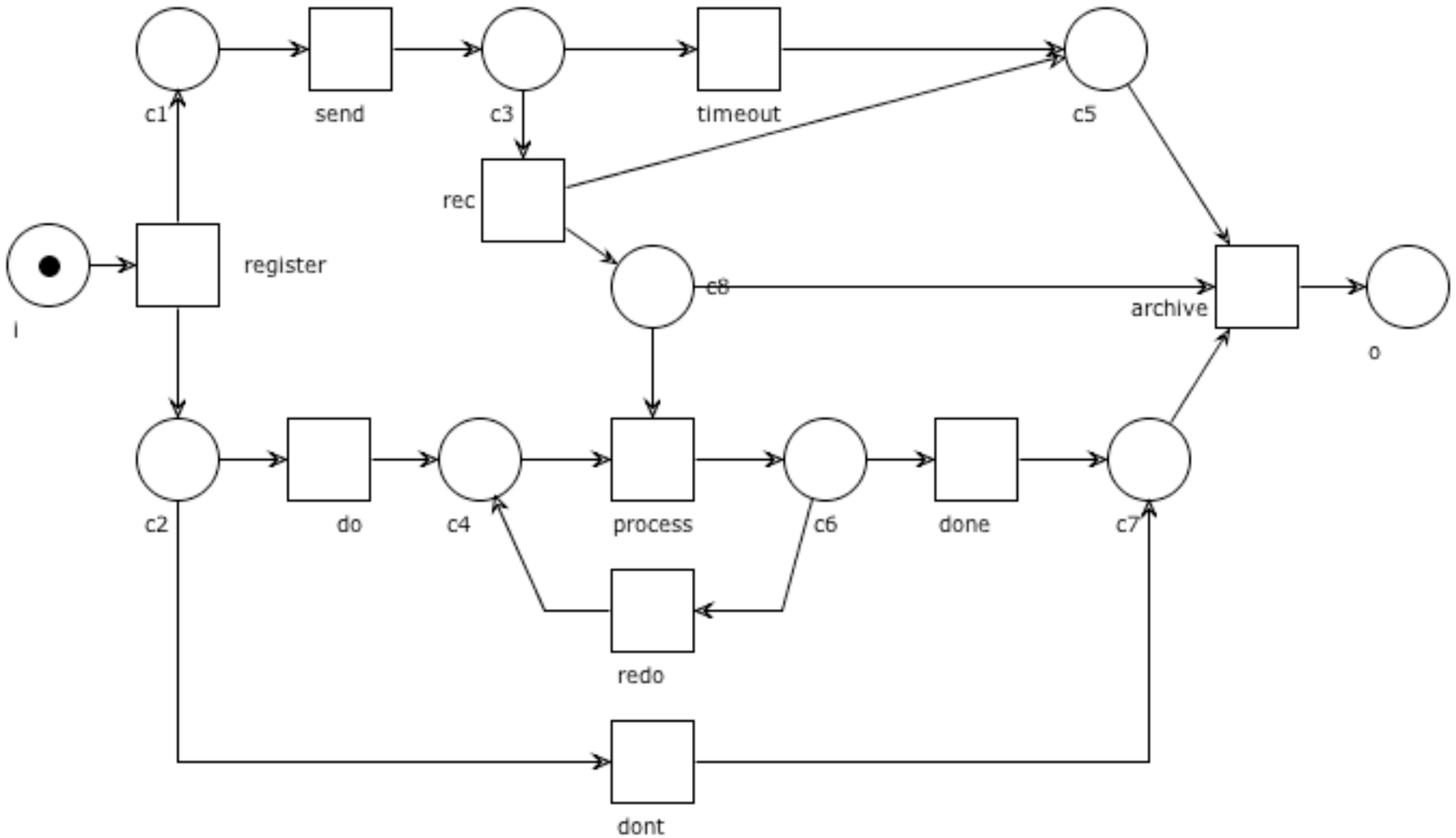
Compute the RG of N*

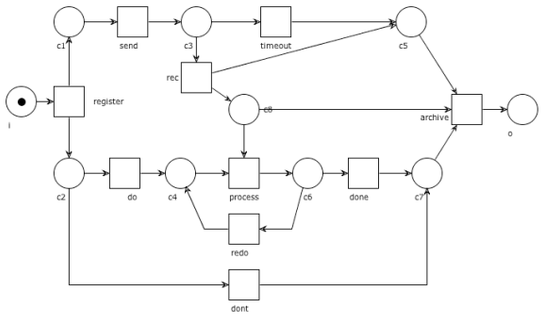
Color in **red** all nodes from which there is **no path** to o

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

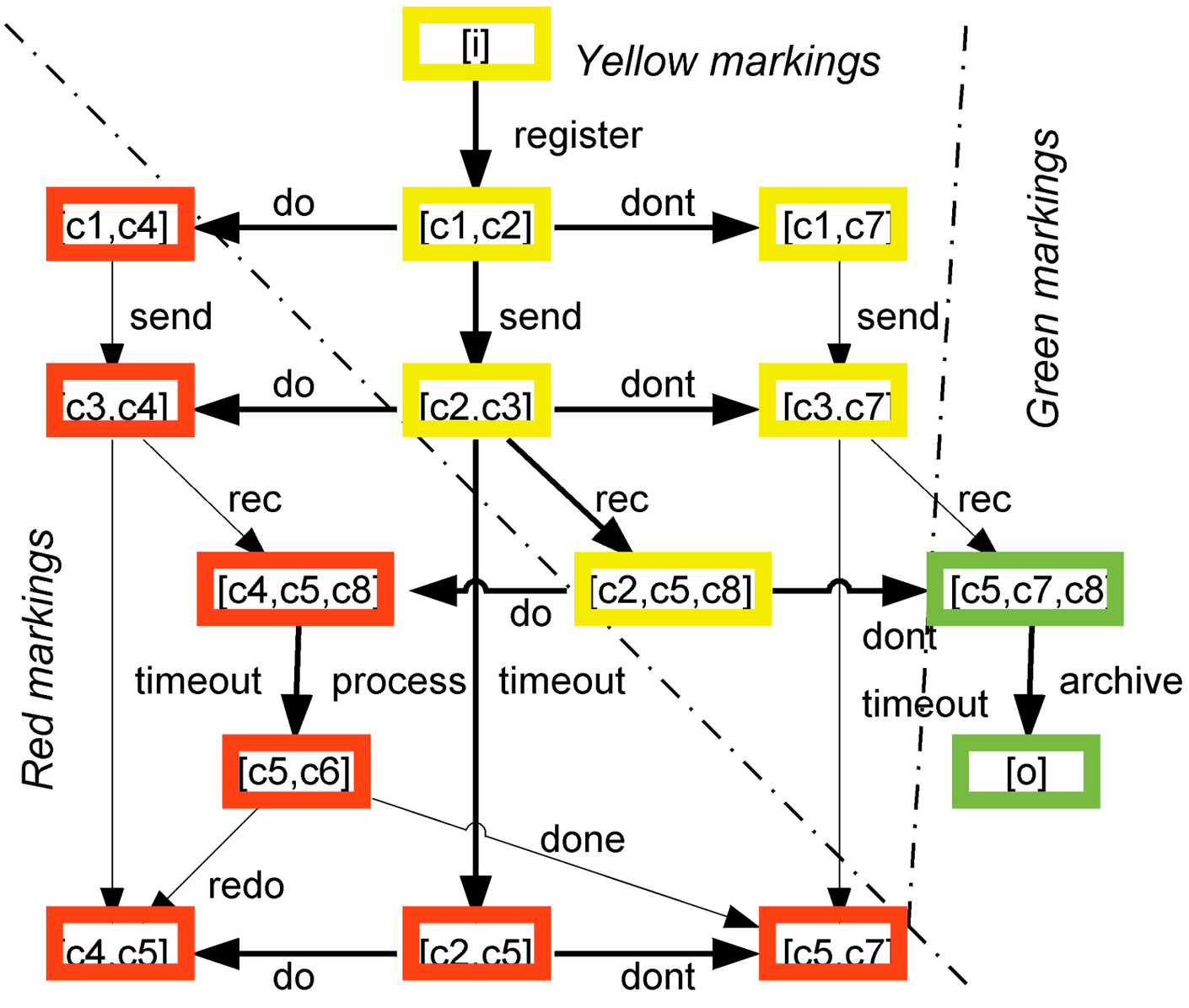
Color in yellow all remaining nodes
(some but not all paths lead to o)

Example: N


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



Example: RG (N)



Non-live sequences:

register, do

register, send, do

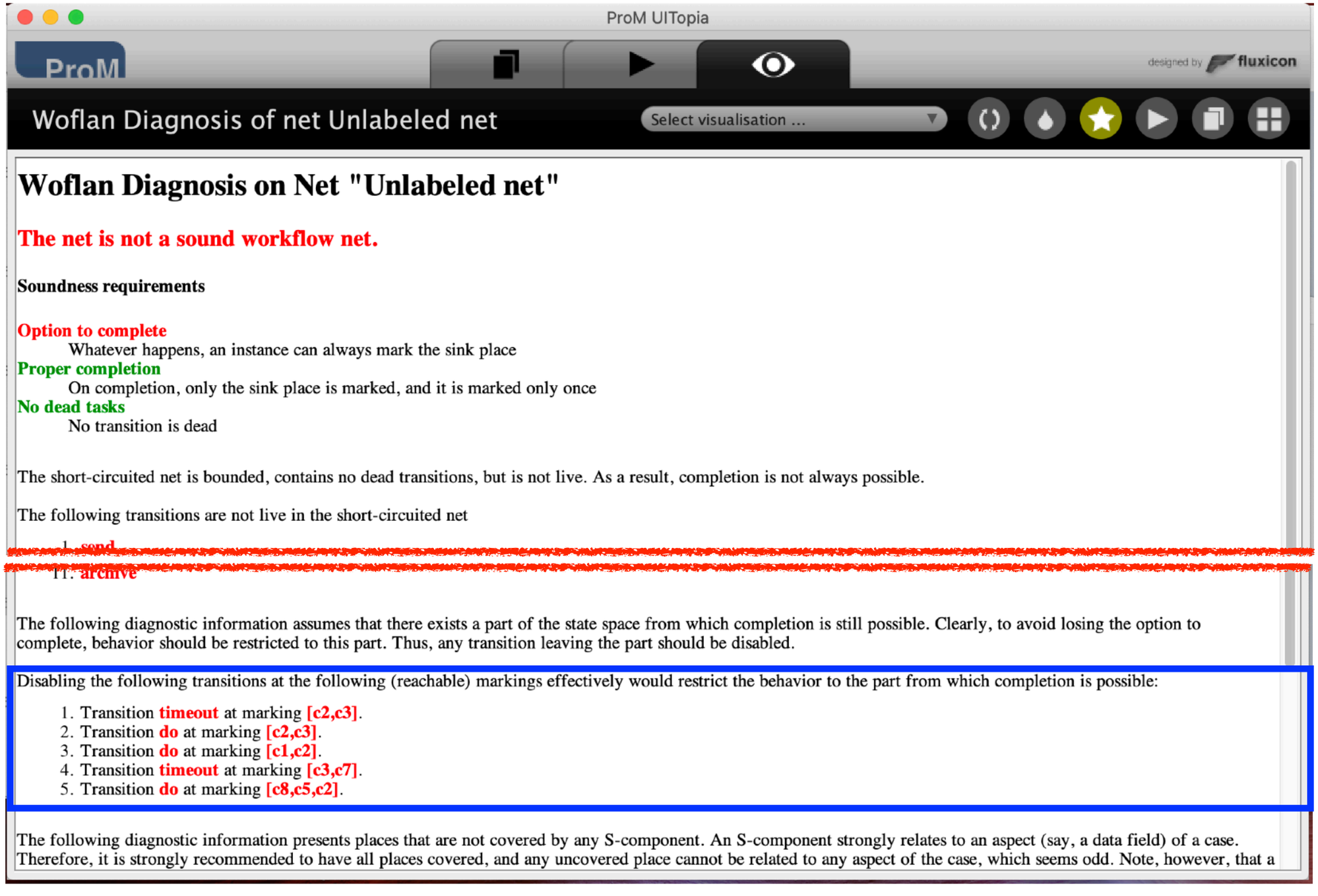
register, send, timeout

register, send, rec, do

register, send, dont, timeout

register, dont, send, timeout

Woflan (in ProM)



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

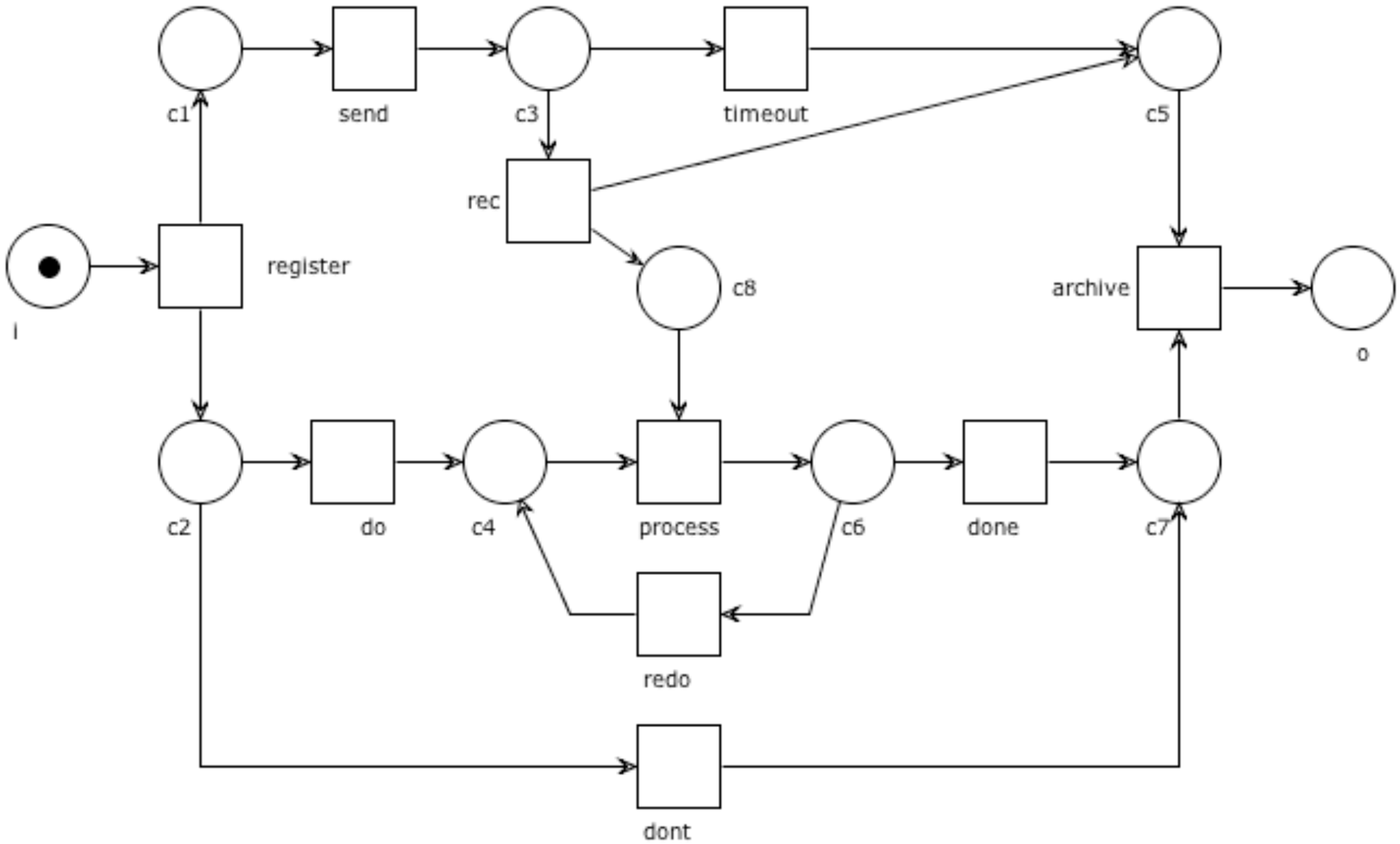
Compute the CG of N*

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

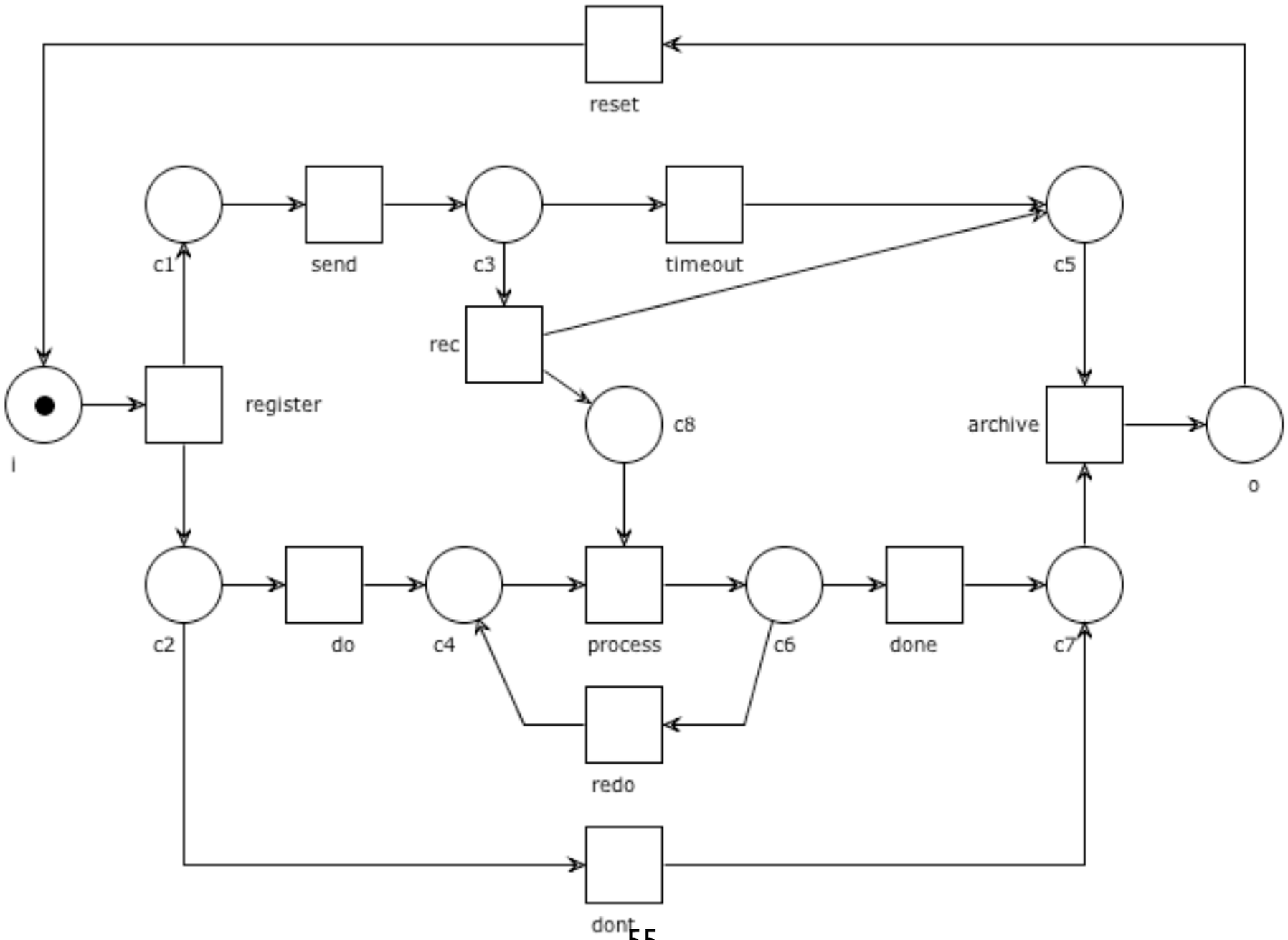
Color in **red** all nodes from which
no green marking is reachable
(undesired markings are unavoidable)

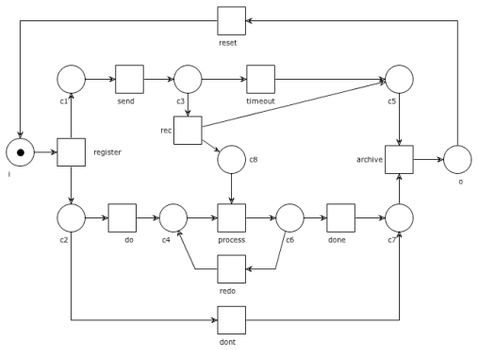
Color in yellow all remaining nodes
(undesired markings are reachable but avoidable)

Example: N

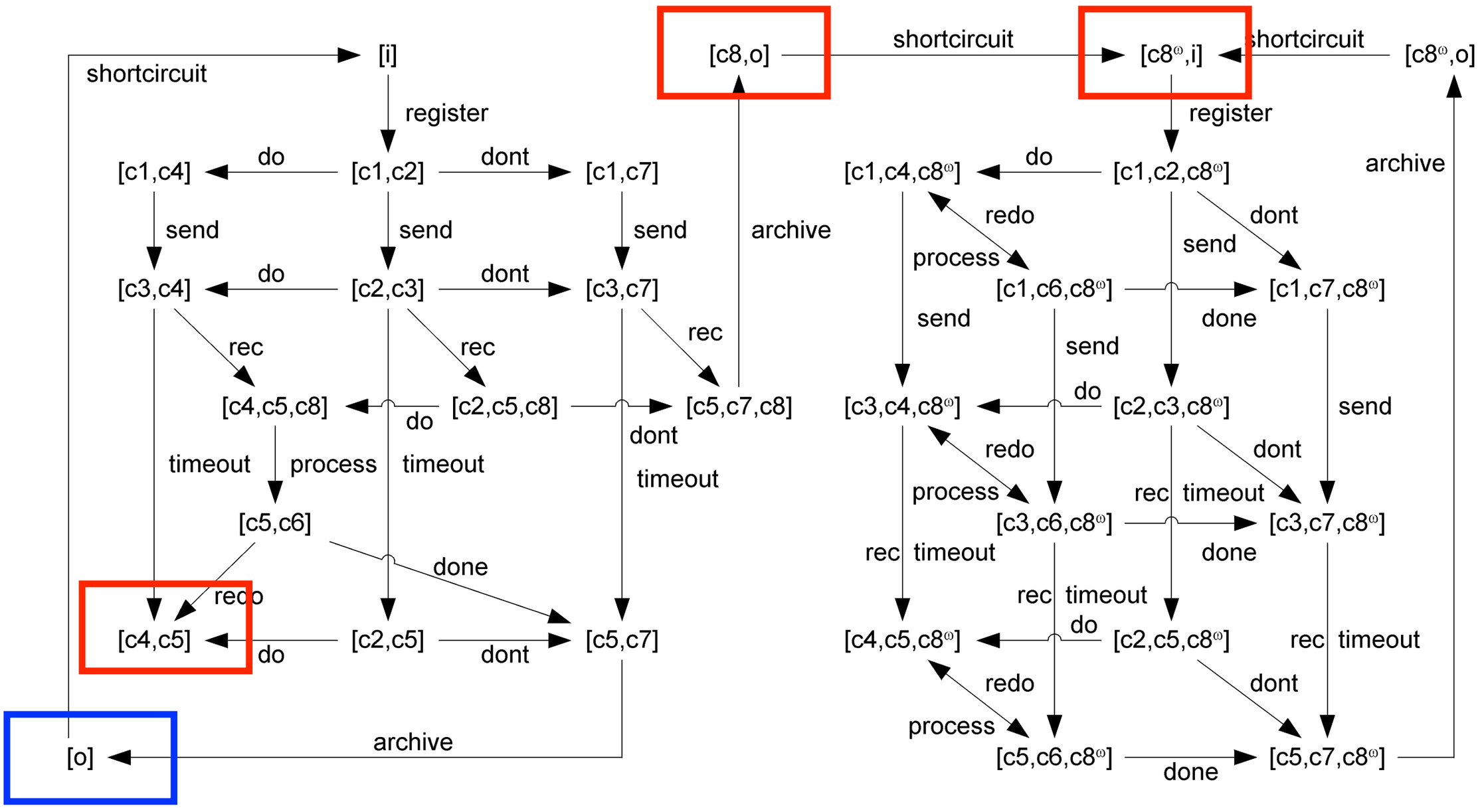

c1
send
c3
timeout
c5
rec
i
register
c8
archive
o
c2
do
c4
process
c6
done
c7
redo
dont

Example: N*


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


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

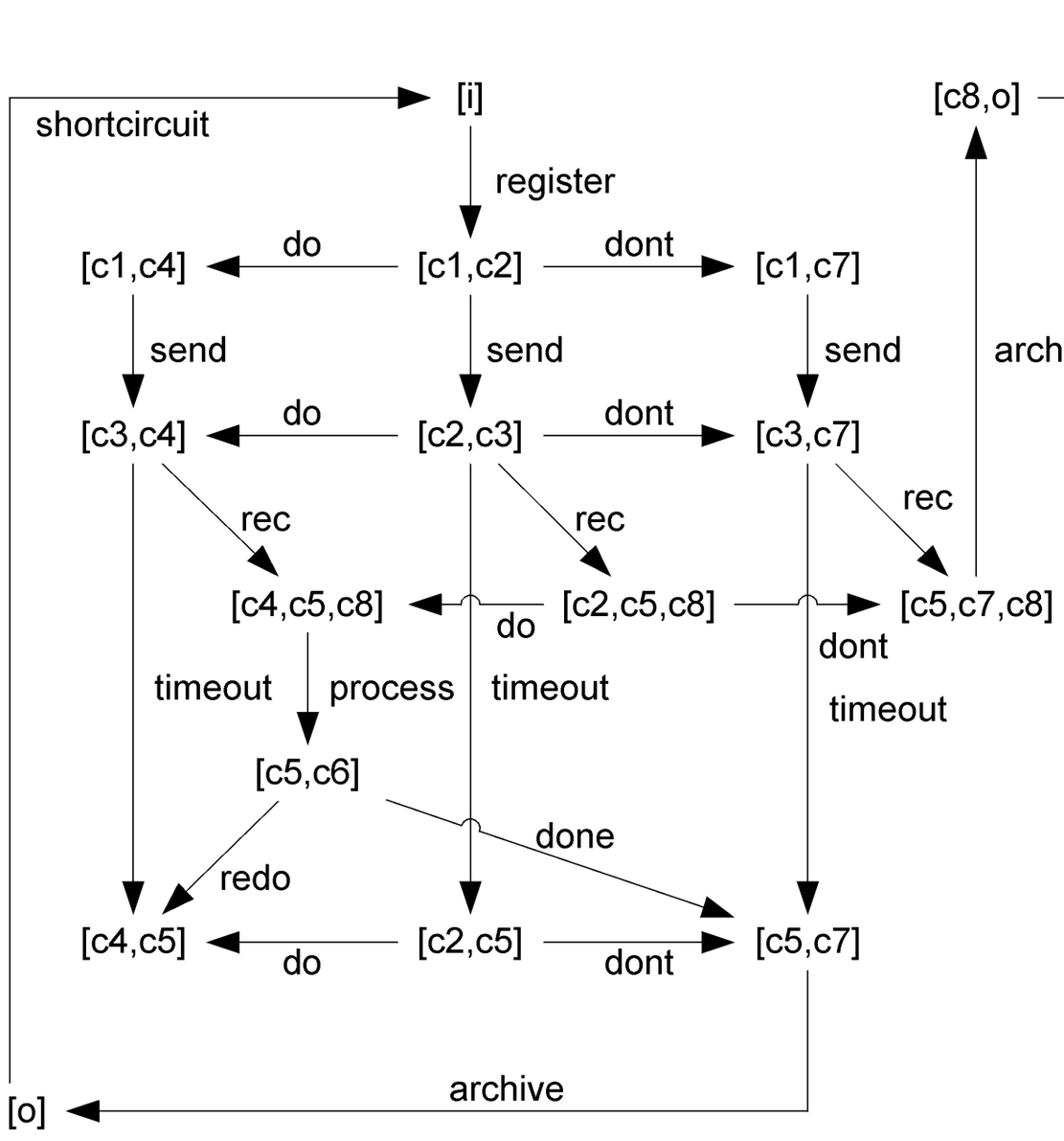
Restricted coverability graph (RCG)

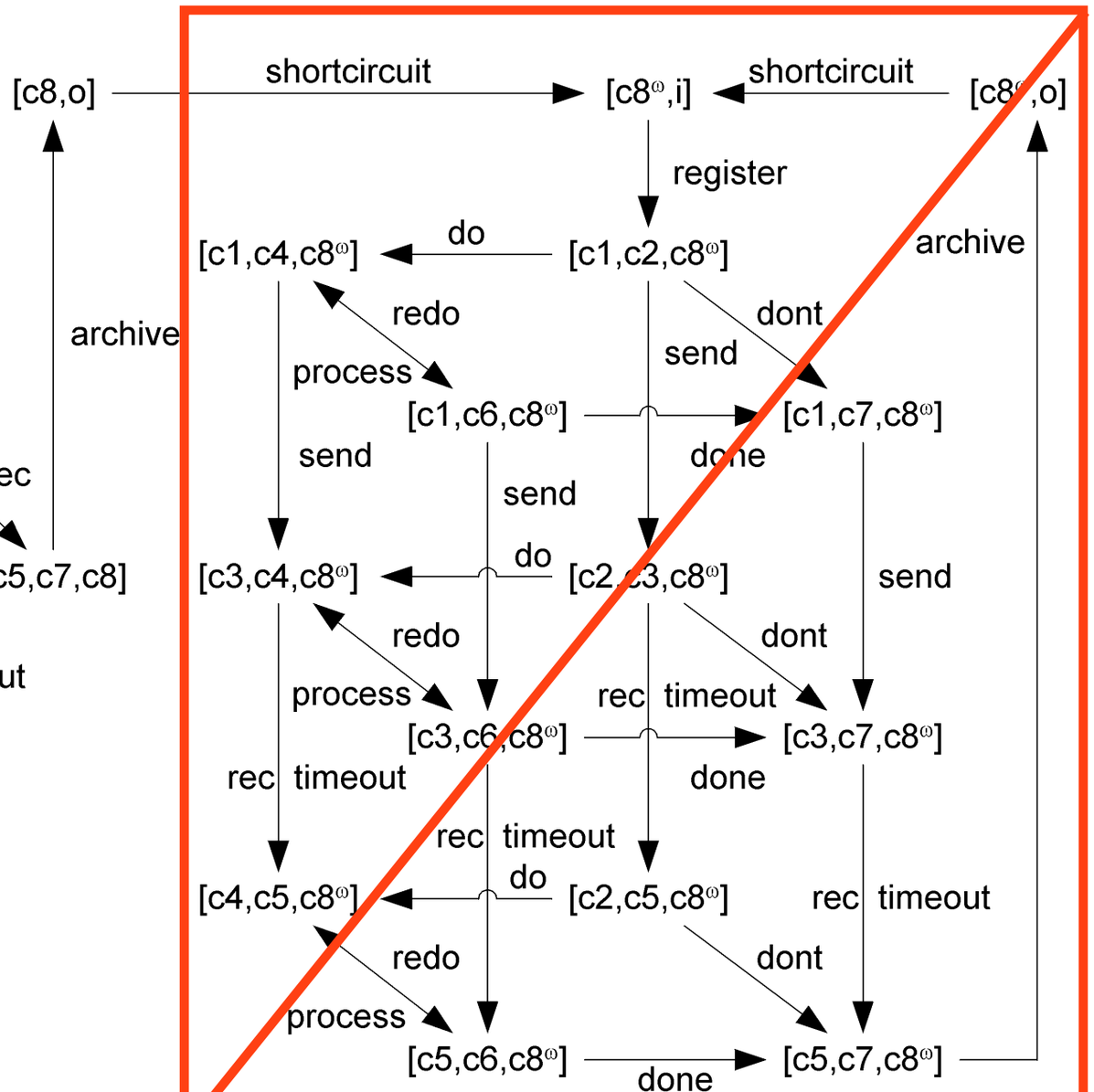
CG can become very large

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

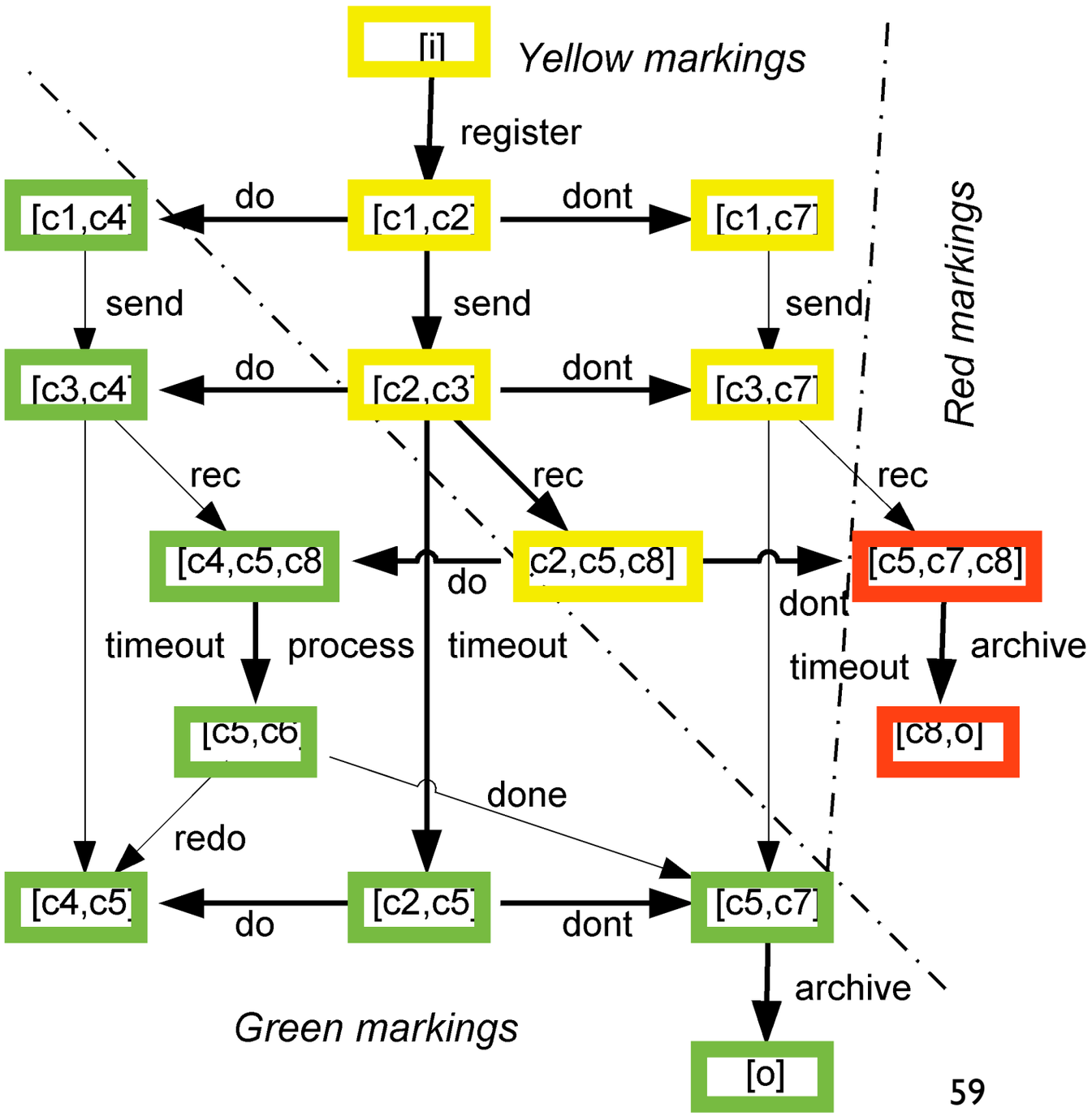
We can just avoid computing them!

Example:
Restricted CG vs CG





Example: RCG (N*)



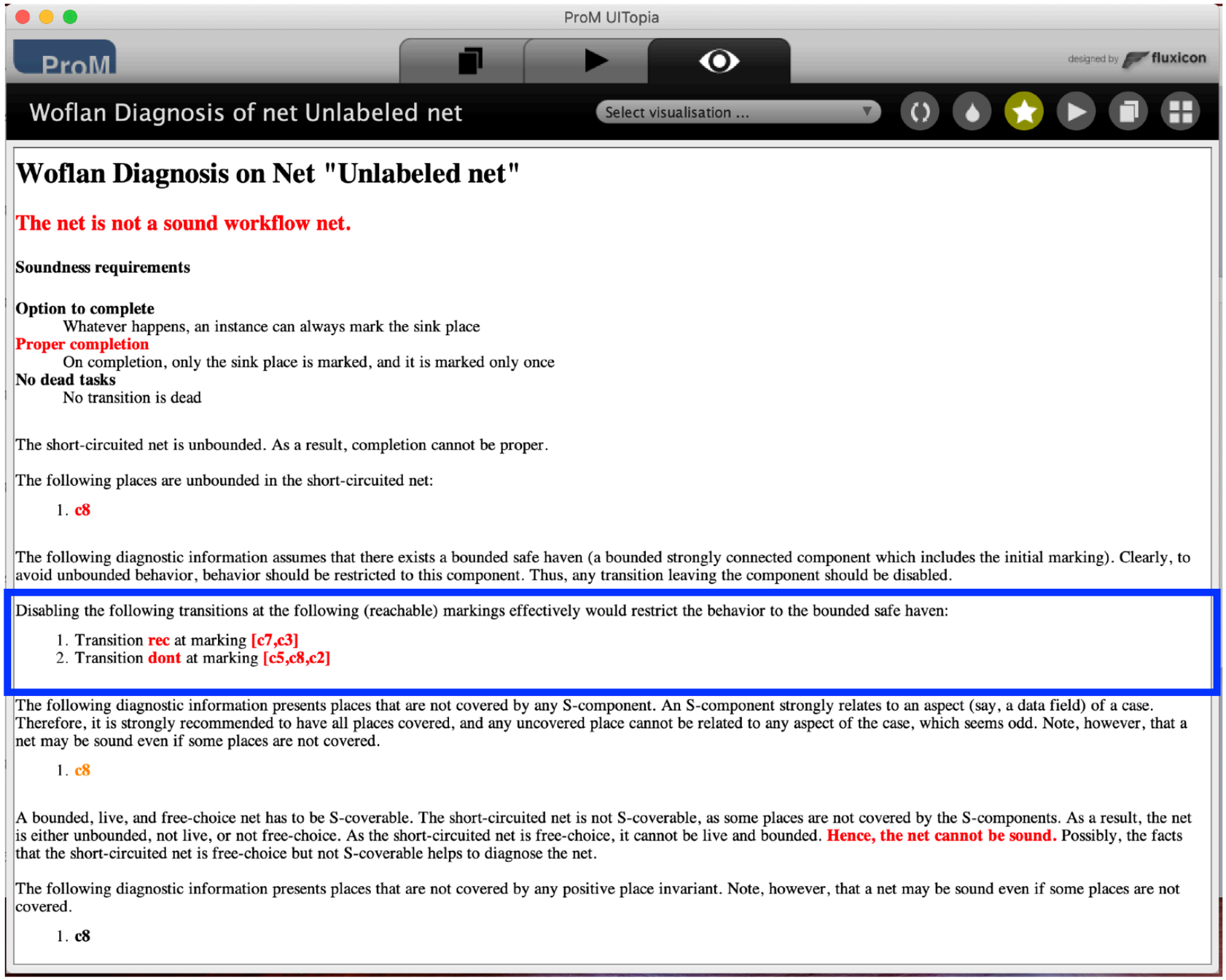
Unbounded sequences:

register, dont, send, rec

register, send, dont, rec

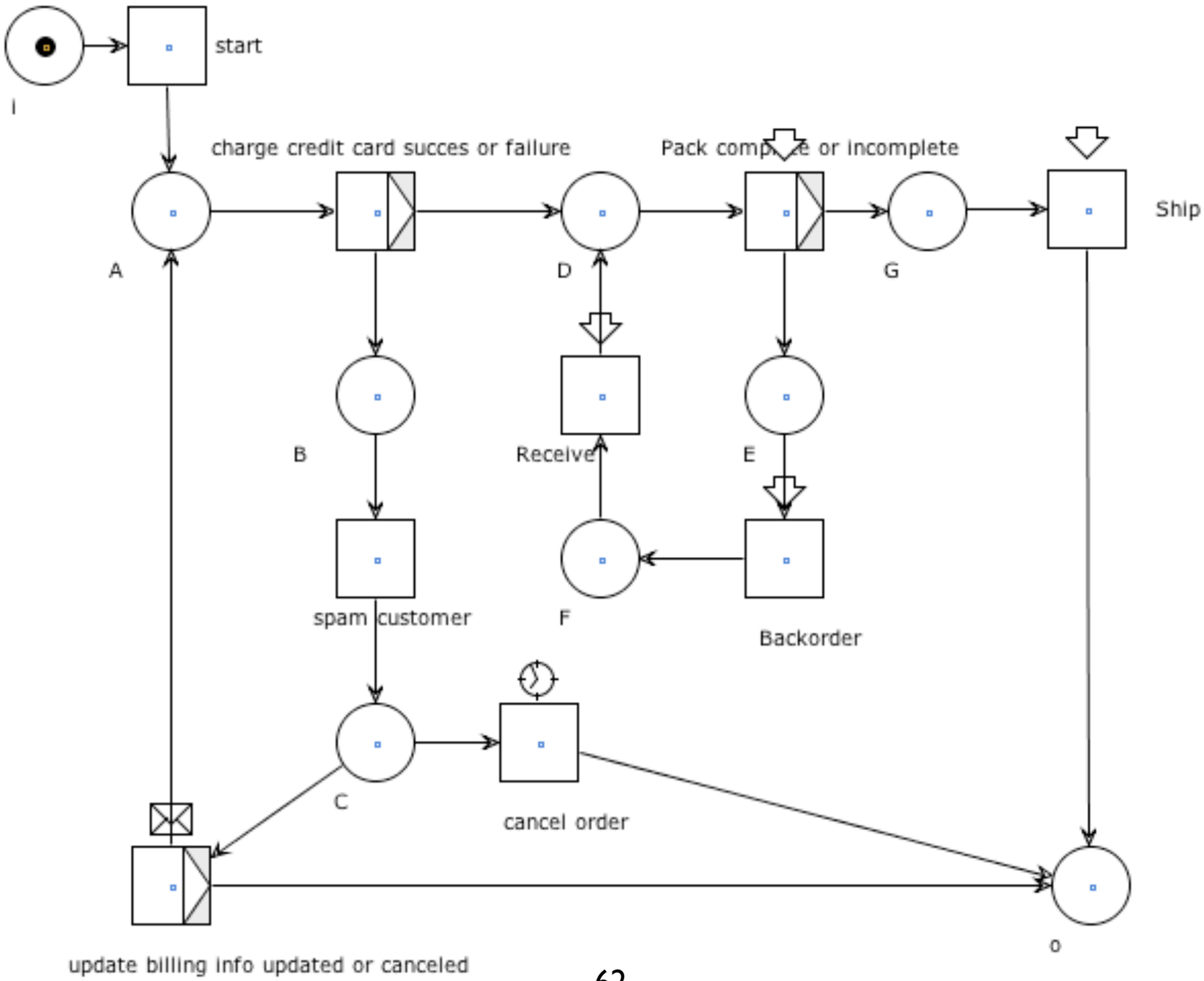
register, send, rec, dont

Woflan (in ProM)

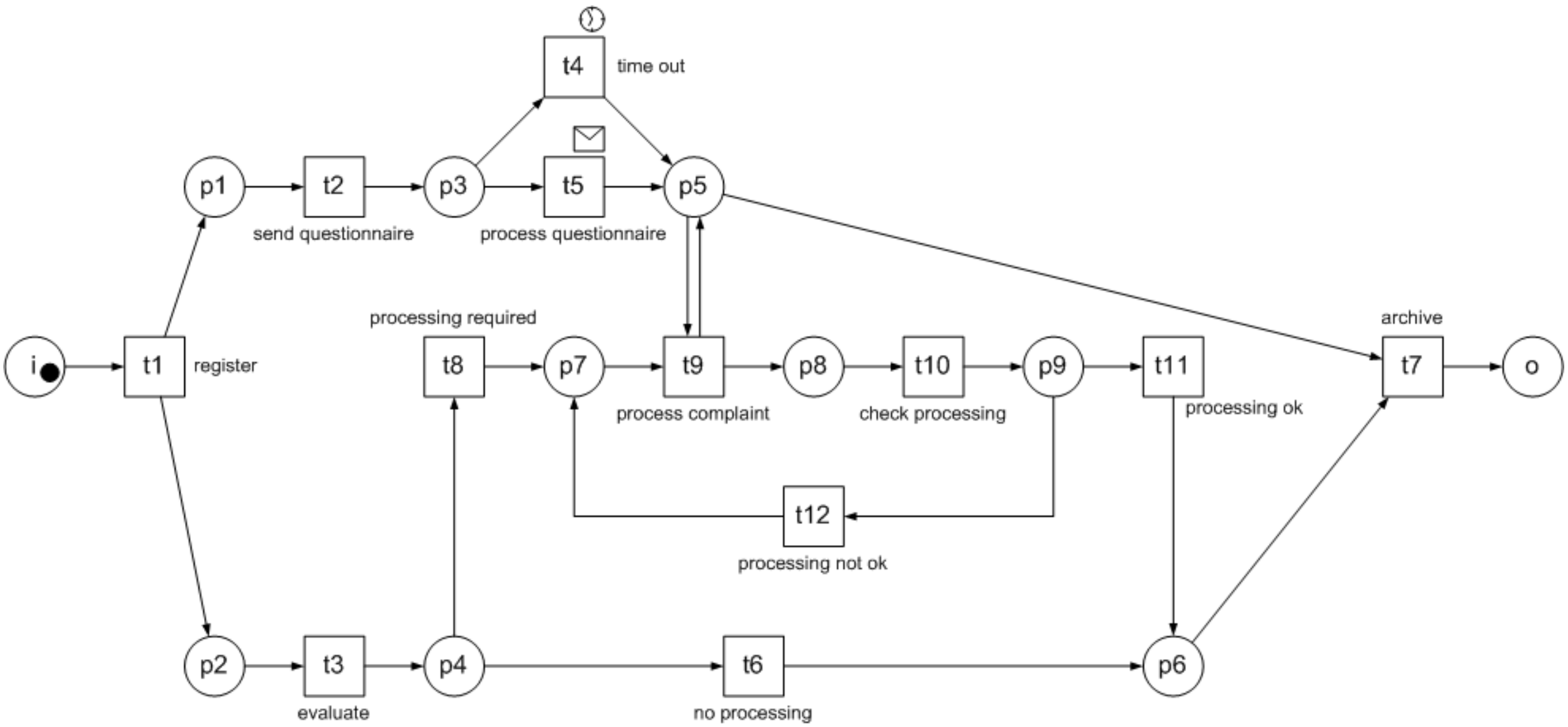


Practice with WoPeD (and Woflan)

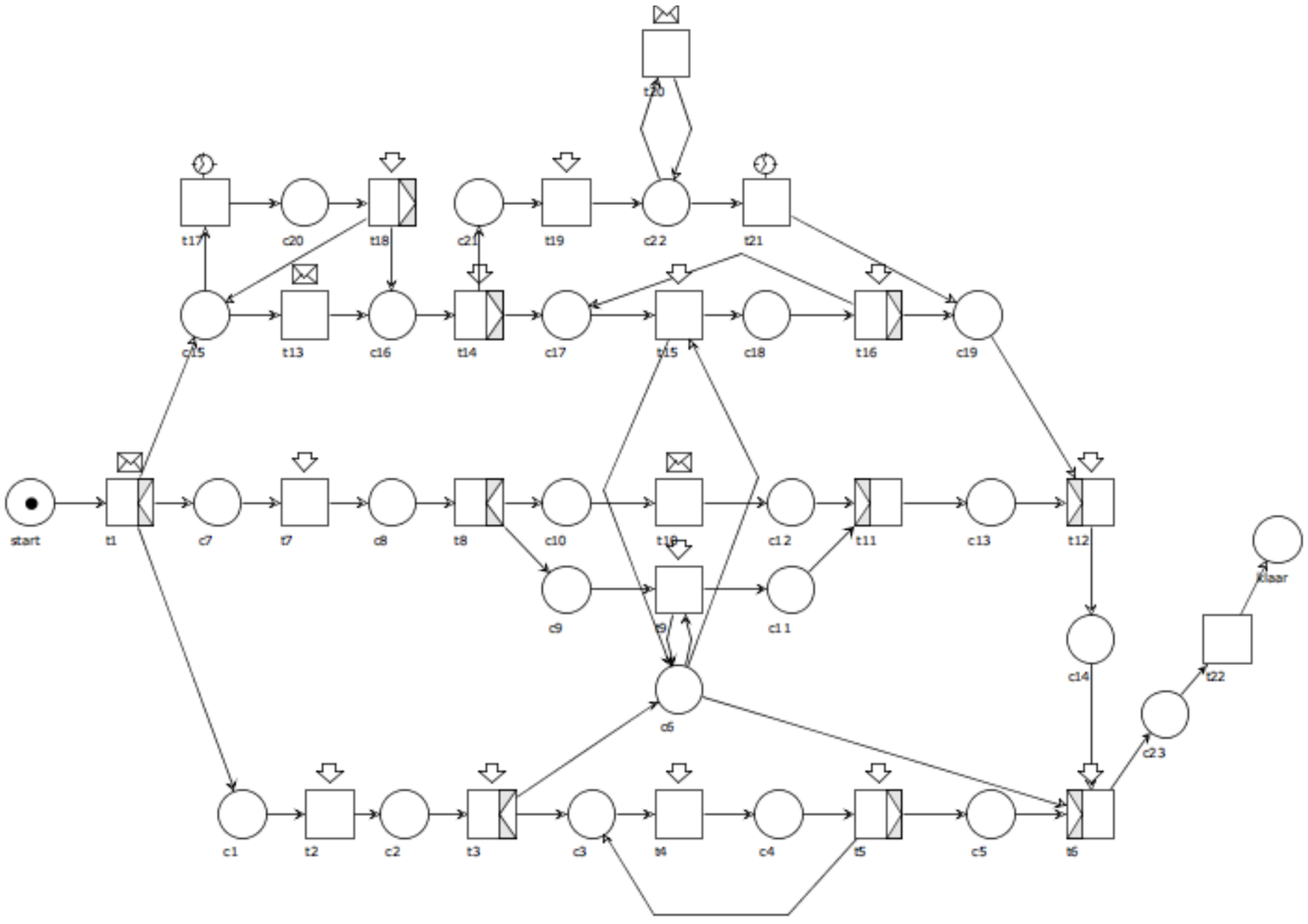
Analyse this net



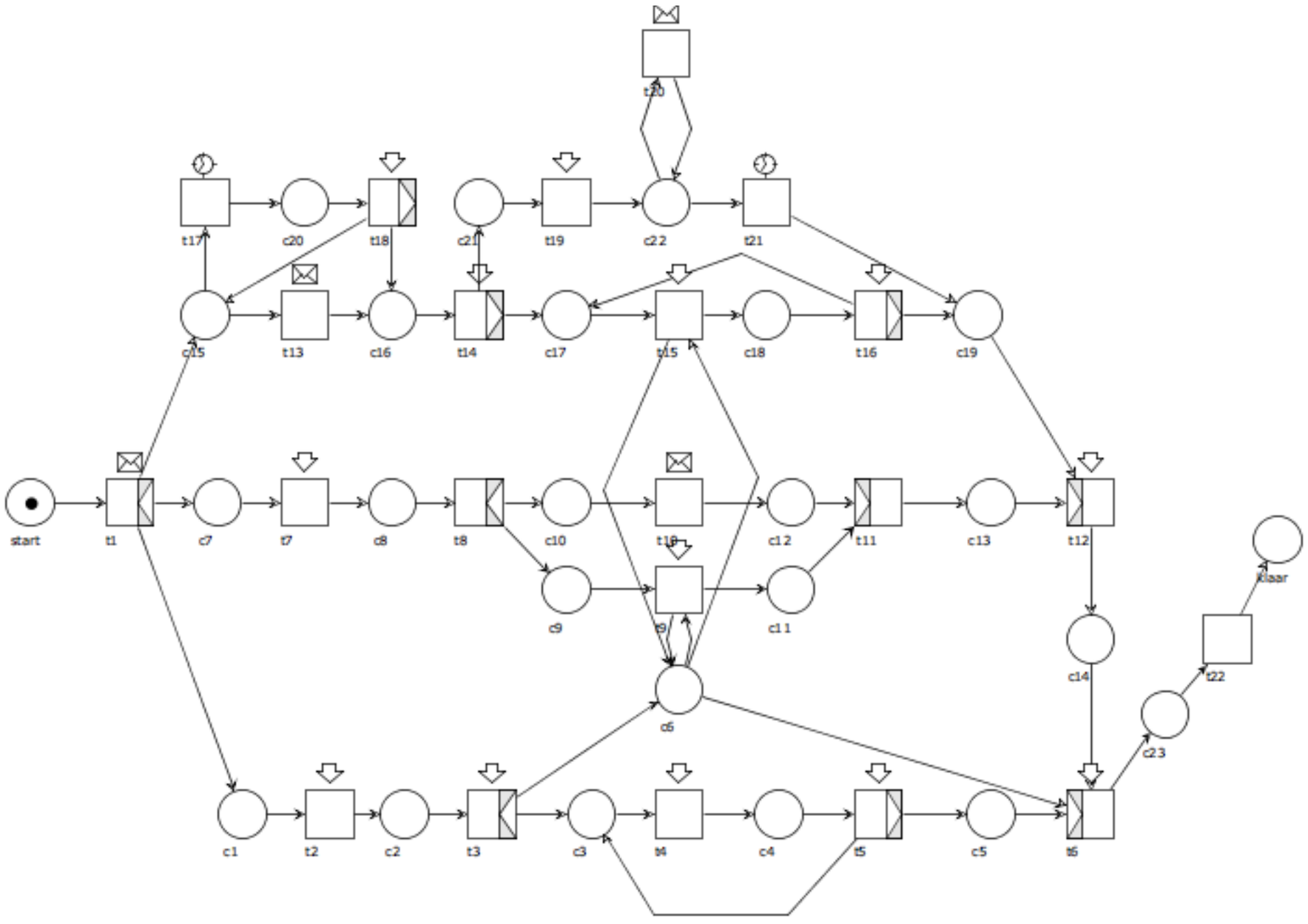
Analyse this net



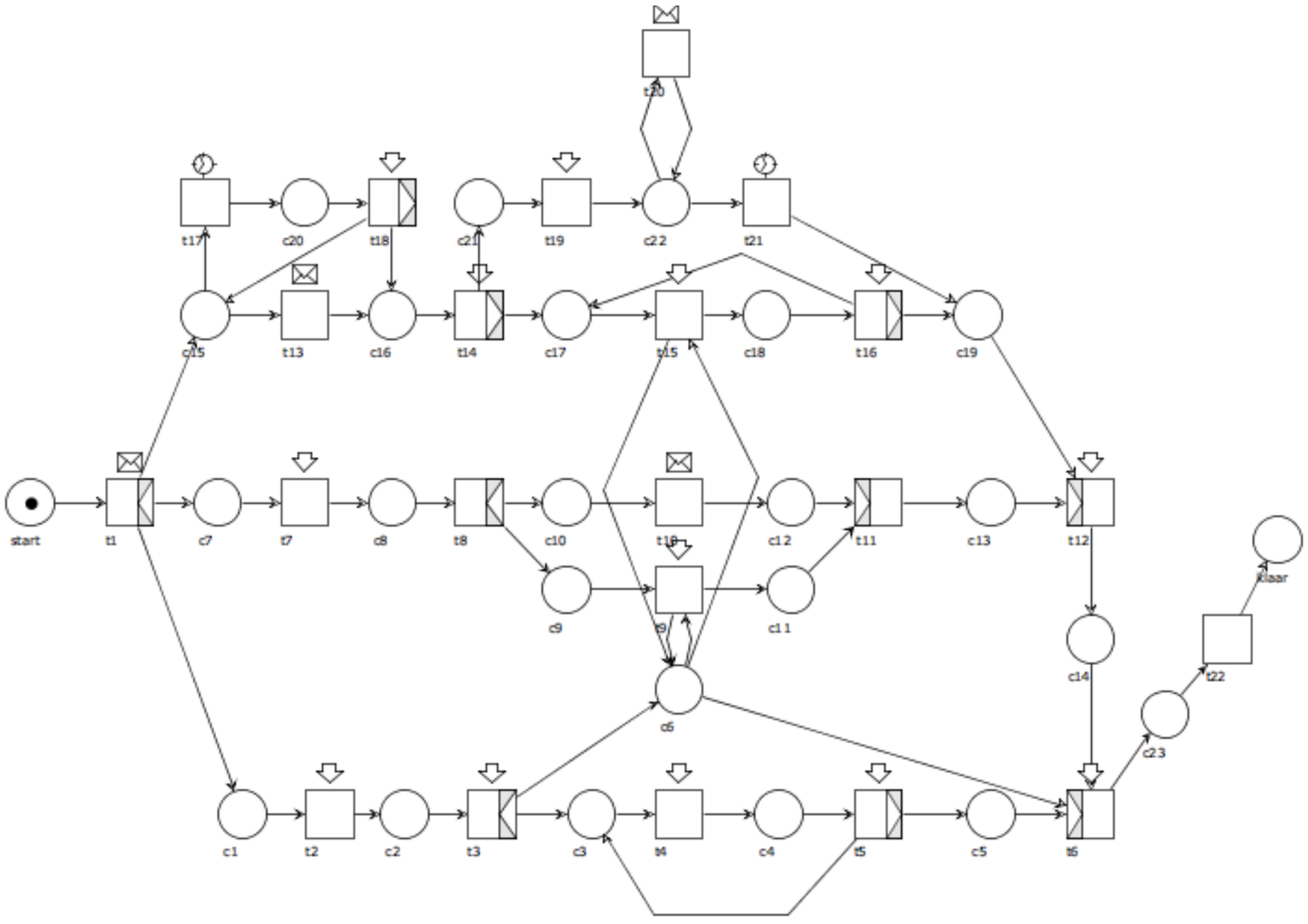
Is this net free-choice?



Is this net S-coverable?



Is this net sound?



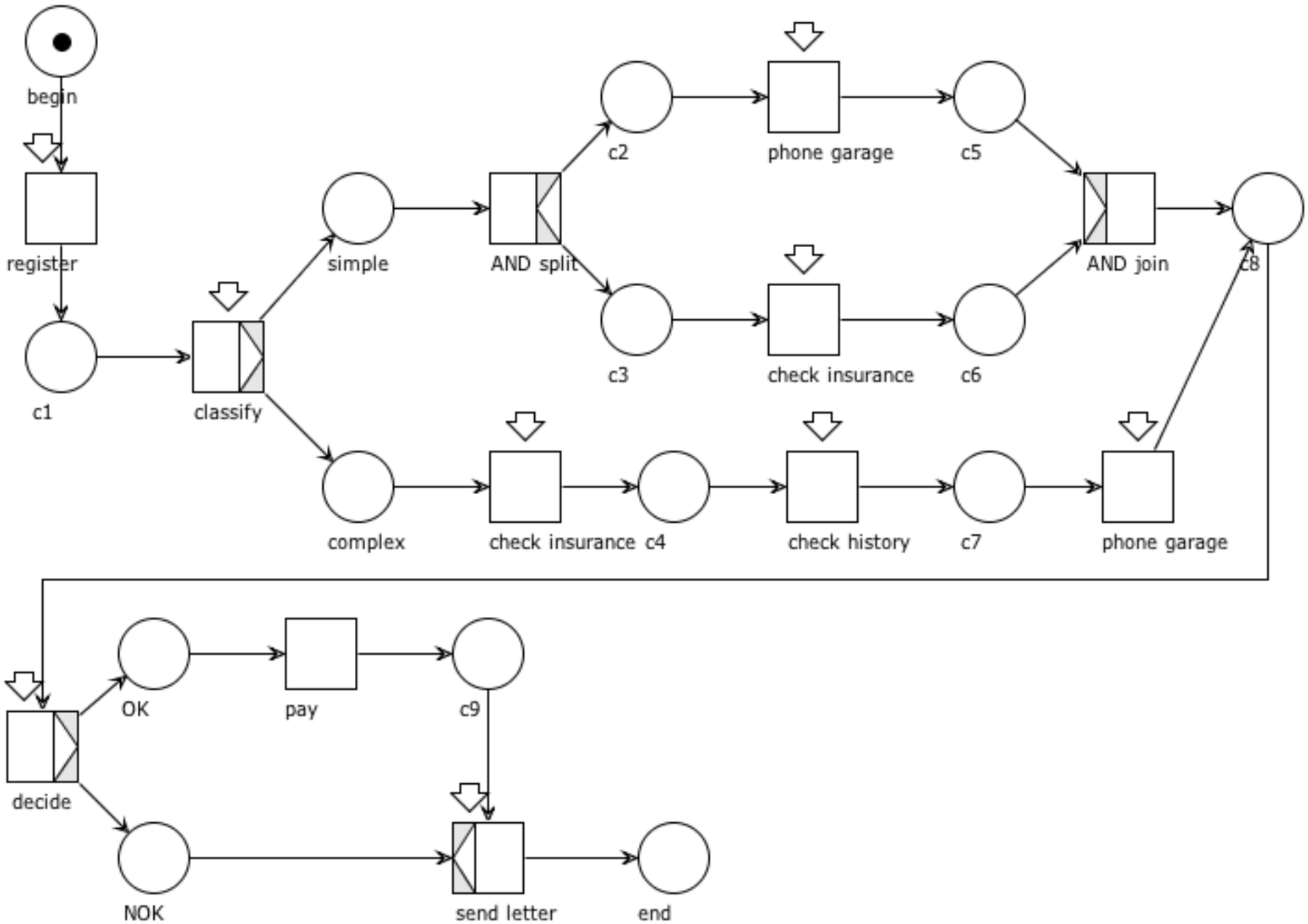
Design Example: Car Damage

- 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.

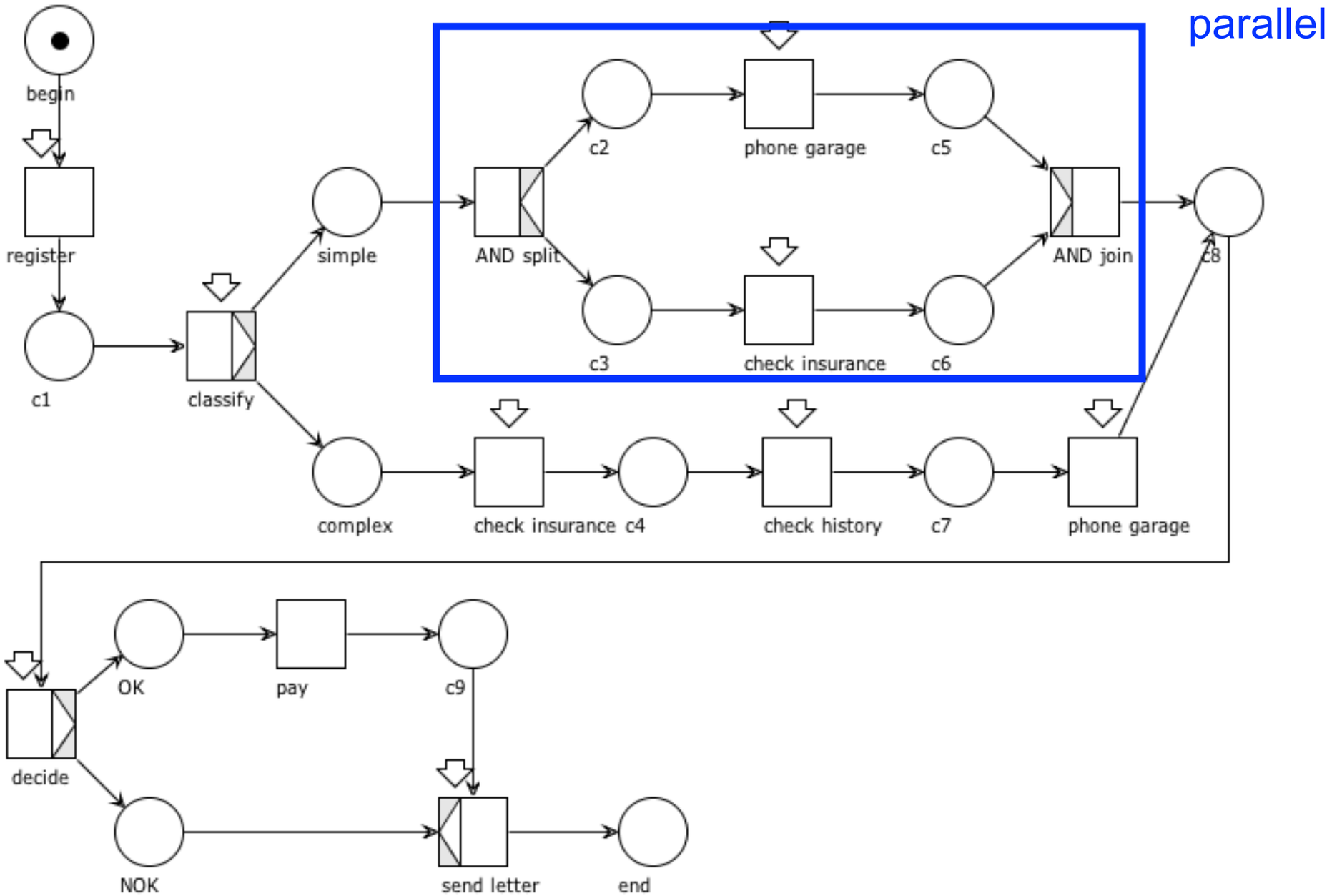
Design Example: Car Damage

- 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.

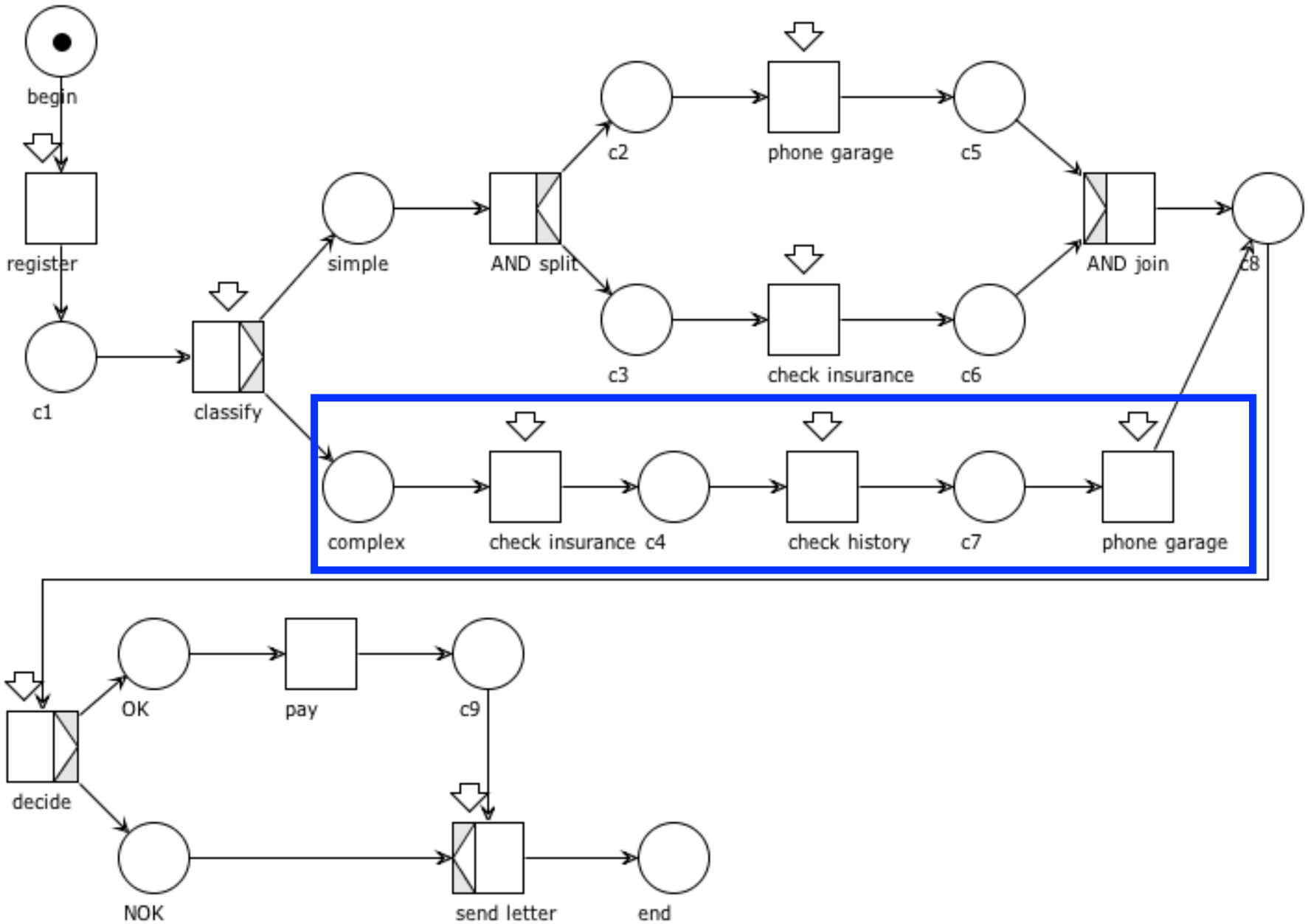
Design Example: Car Damage



Design Example: Car Damage

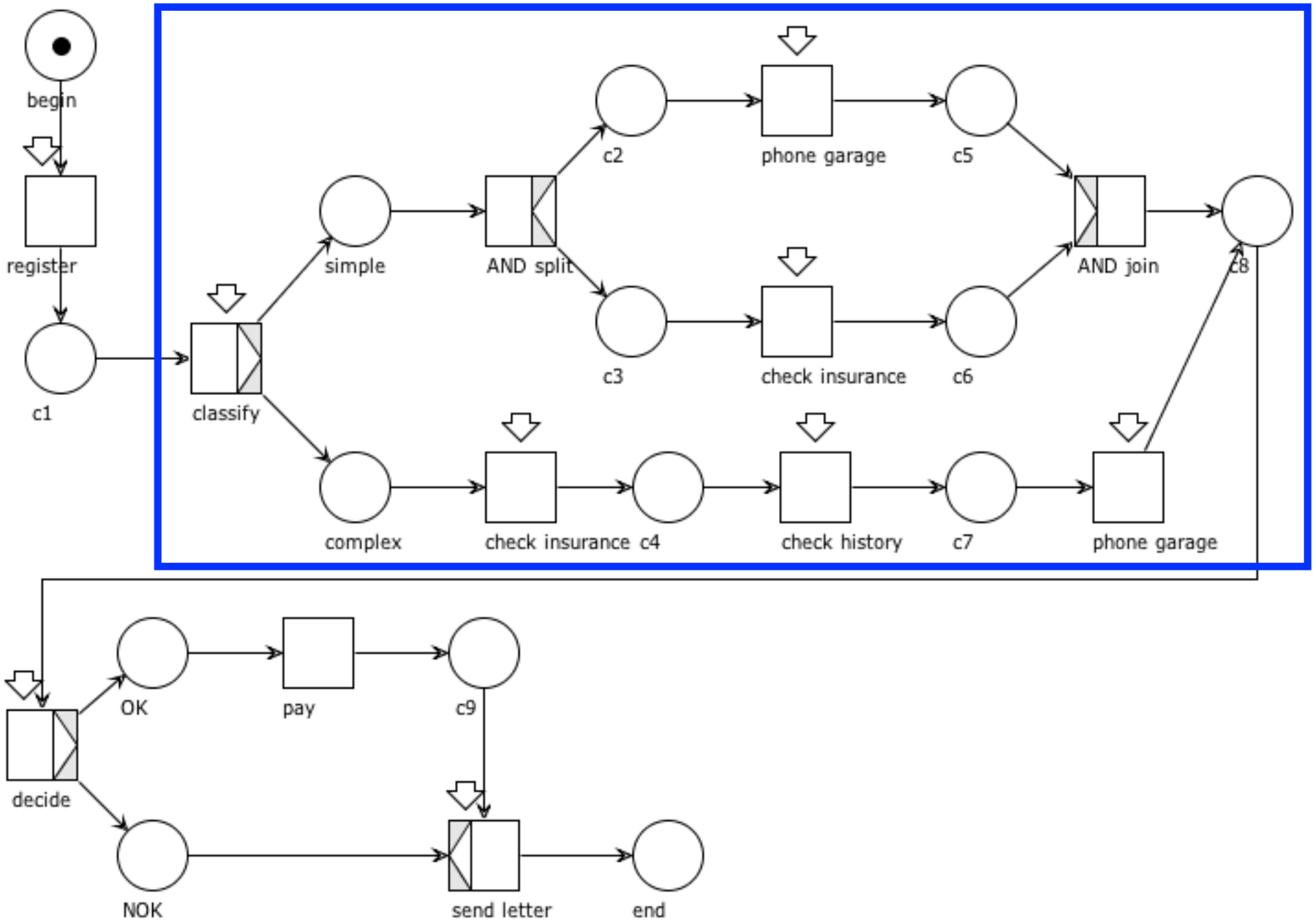


Design Example: Car Damage



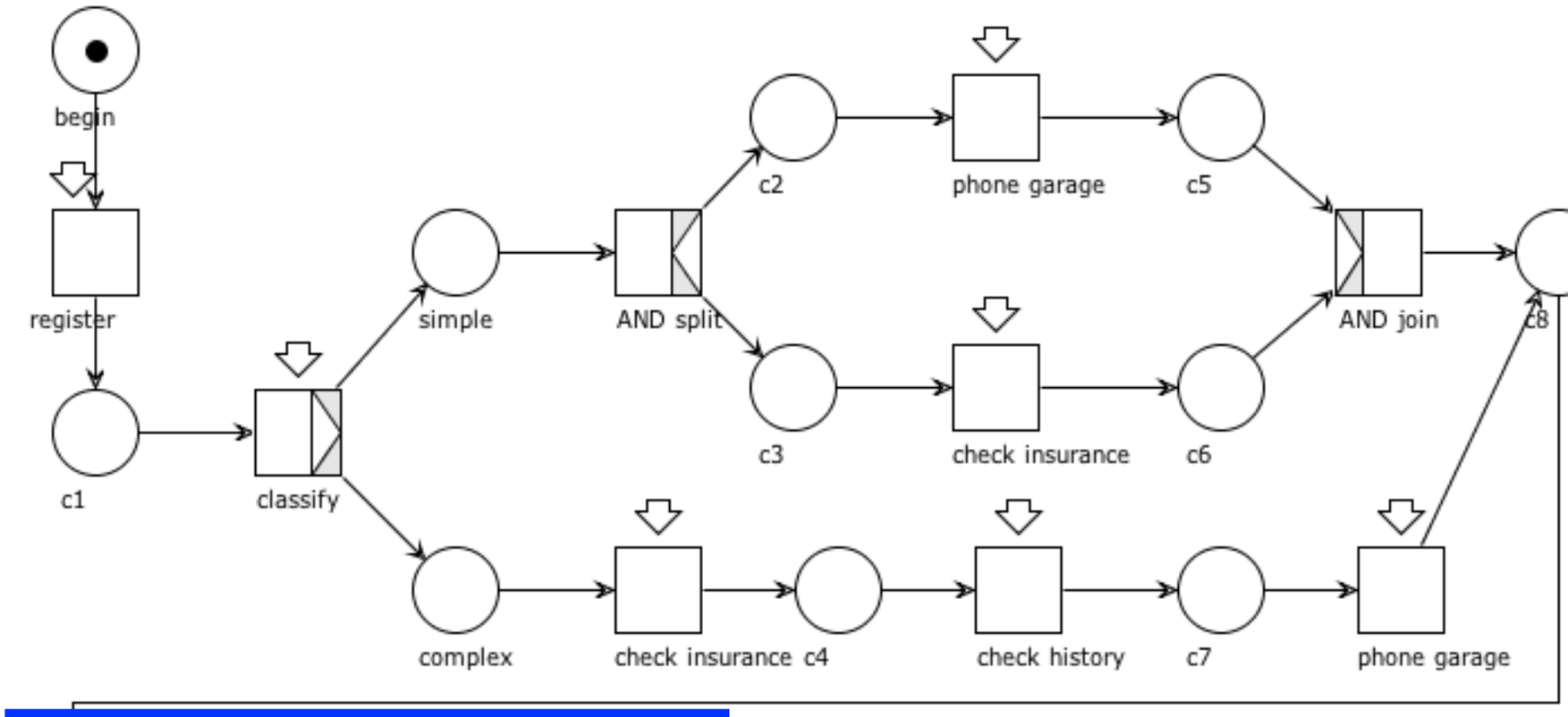
sequence

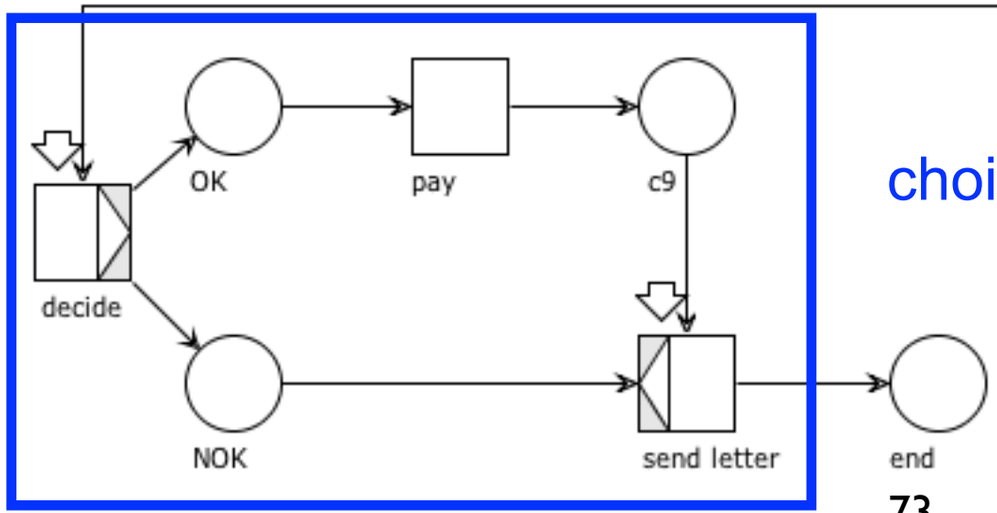
Design Example: Car Damage



choice

Design Example: Car Damage

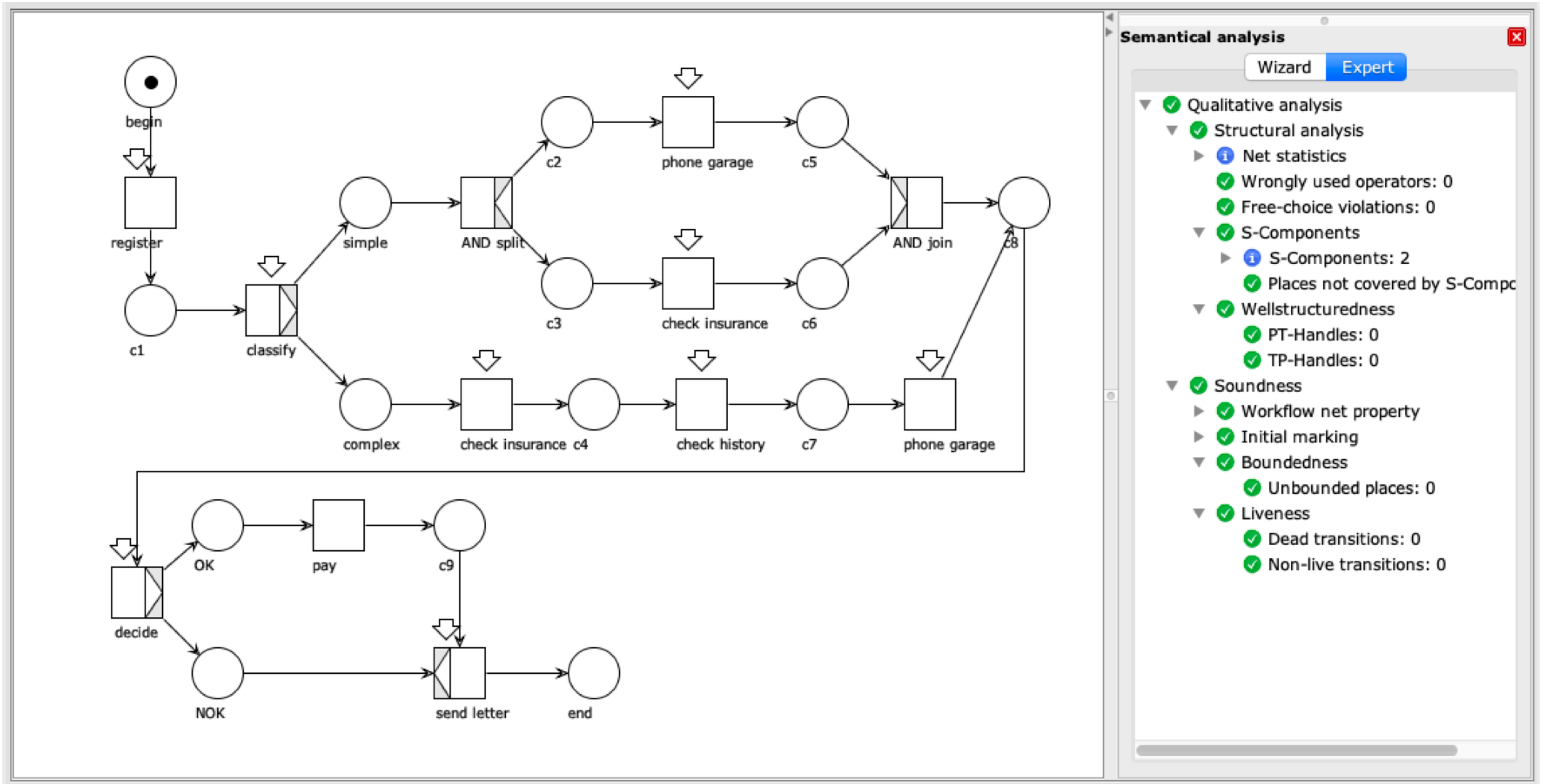




choice

Sound and safe by construction!

Design Example: Car Damage



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.