

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>

Some Pragmatic Considerations

We know that, for free-choice nets, liveness and boundedness can be decided efficiently (in polynomial time)

but we want to check soundness for a wider range of nets

Moreover, when a process is not sound, some diagnostic can be generated that indicates why it is flawed

Woflan

(now a ProM plugin)

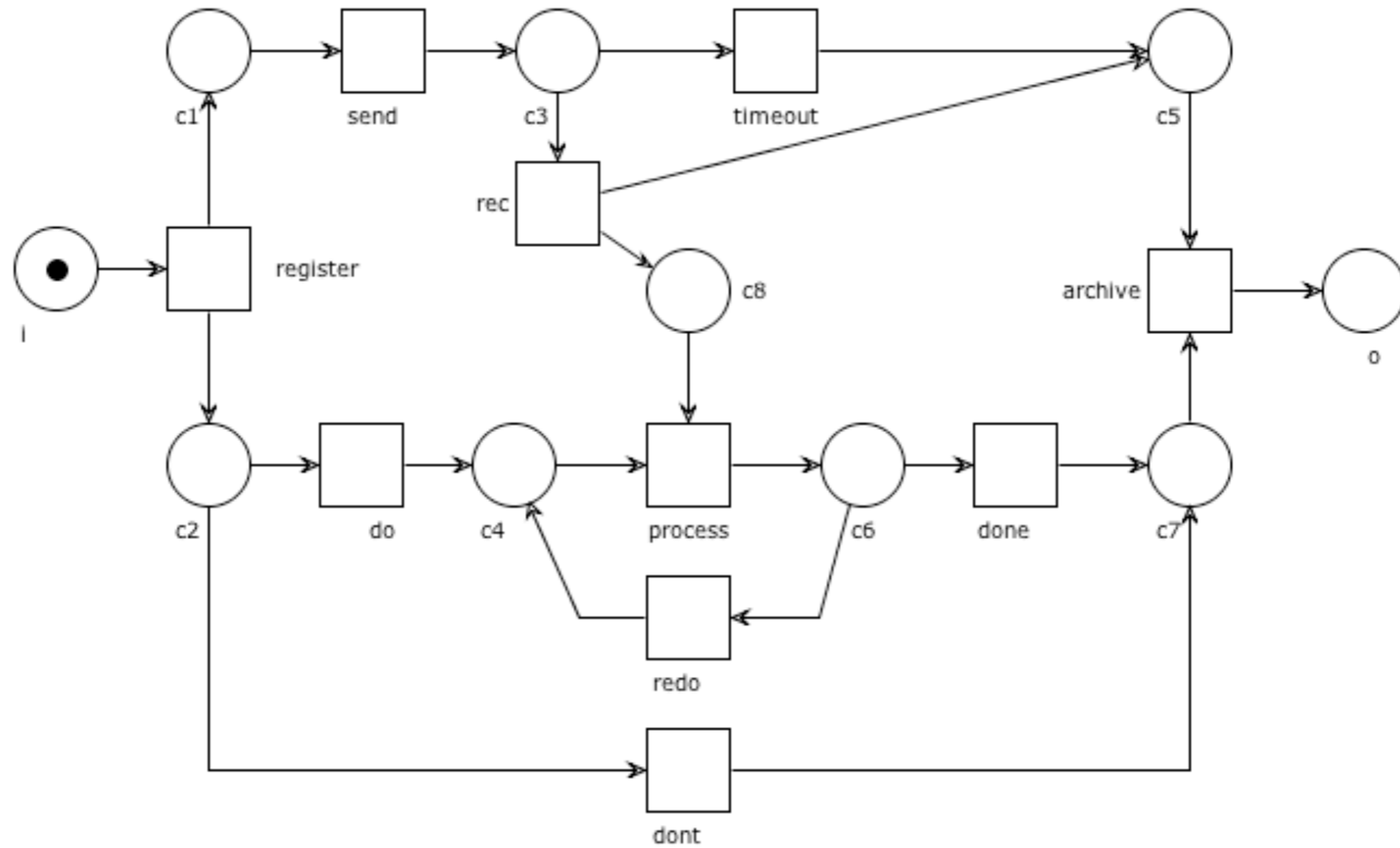


WOrkFLow ANalyzer (Windows only)

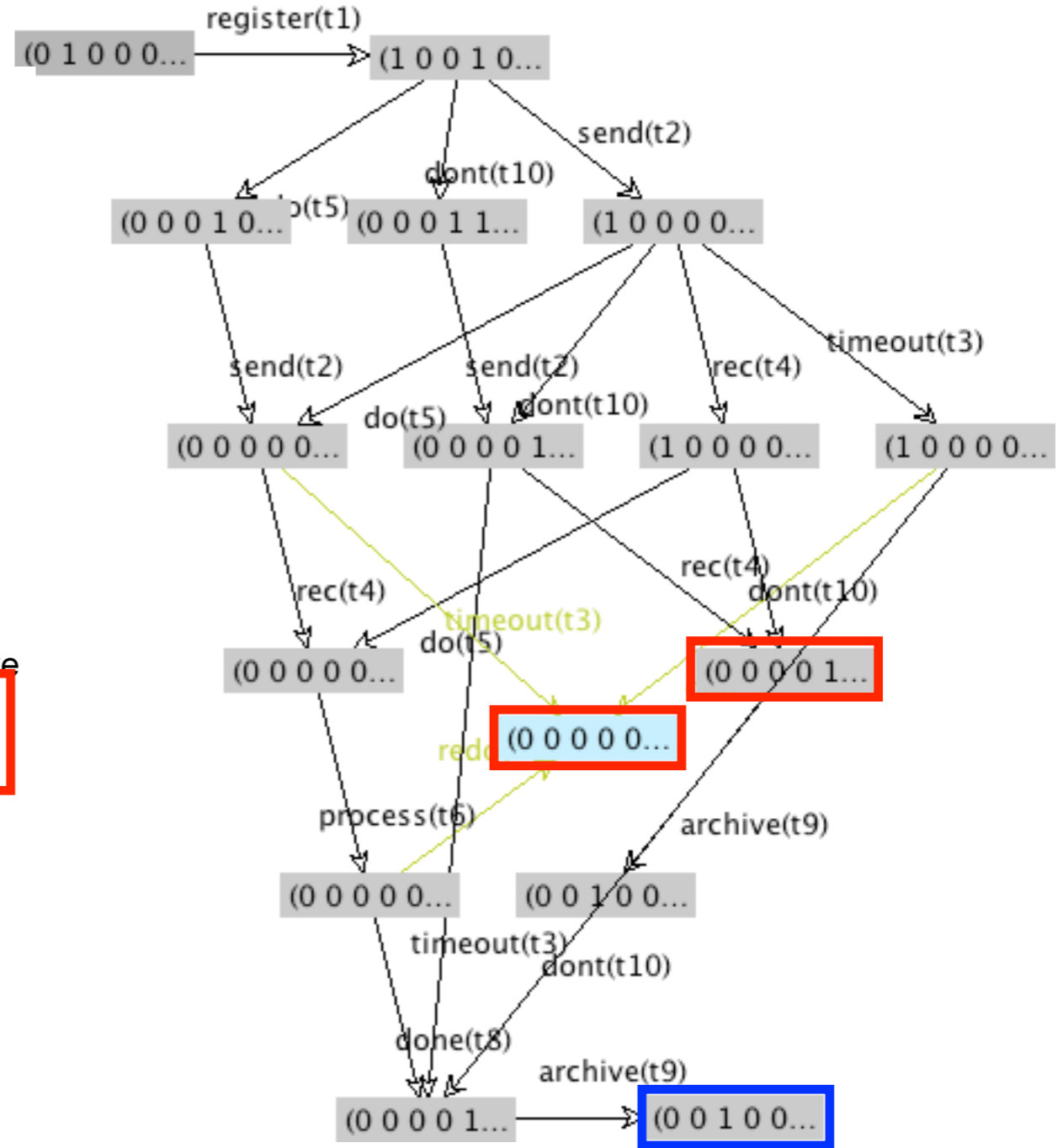
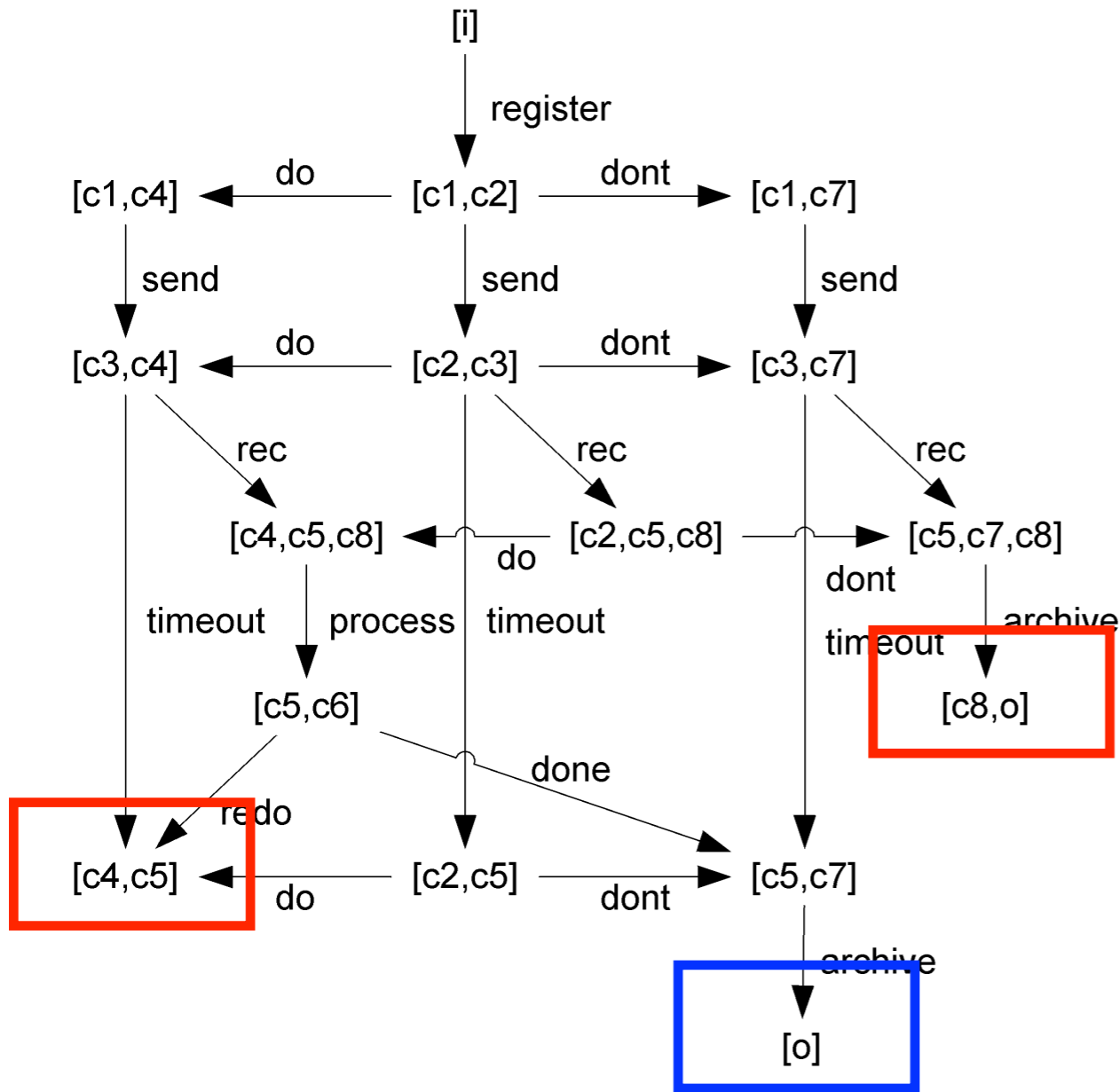
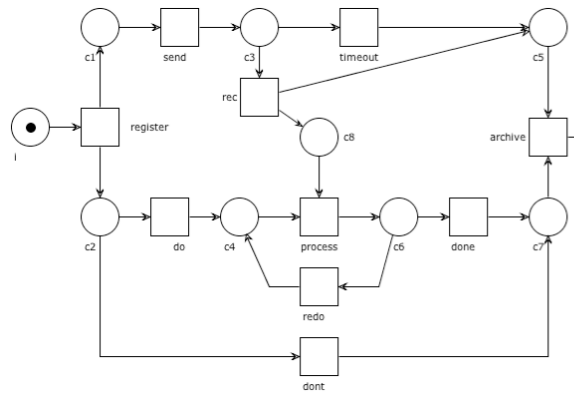
<http://www.win.tue.nl/woflan/>

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

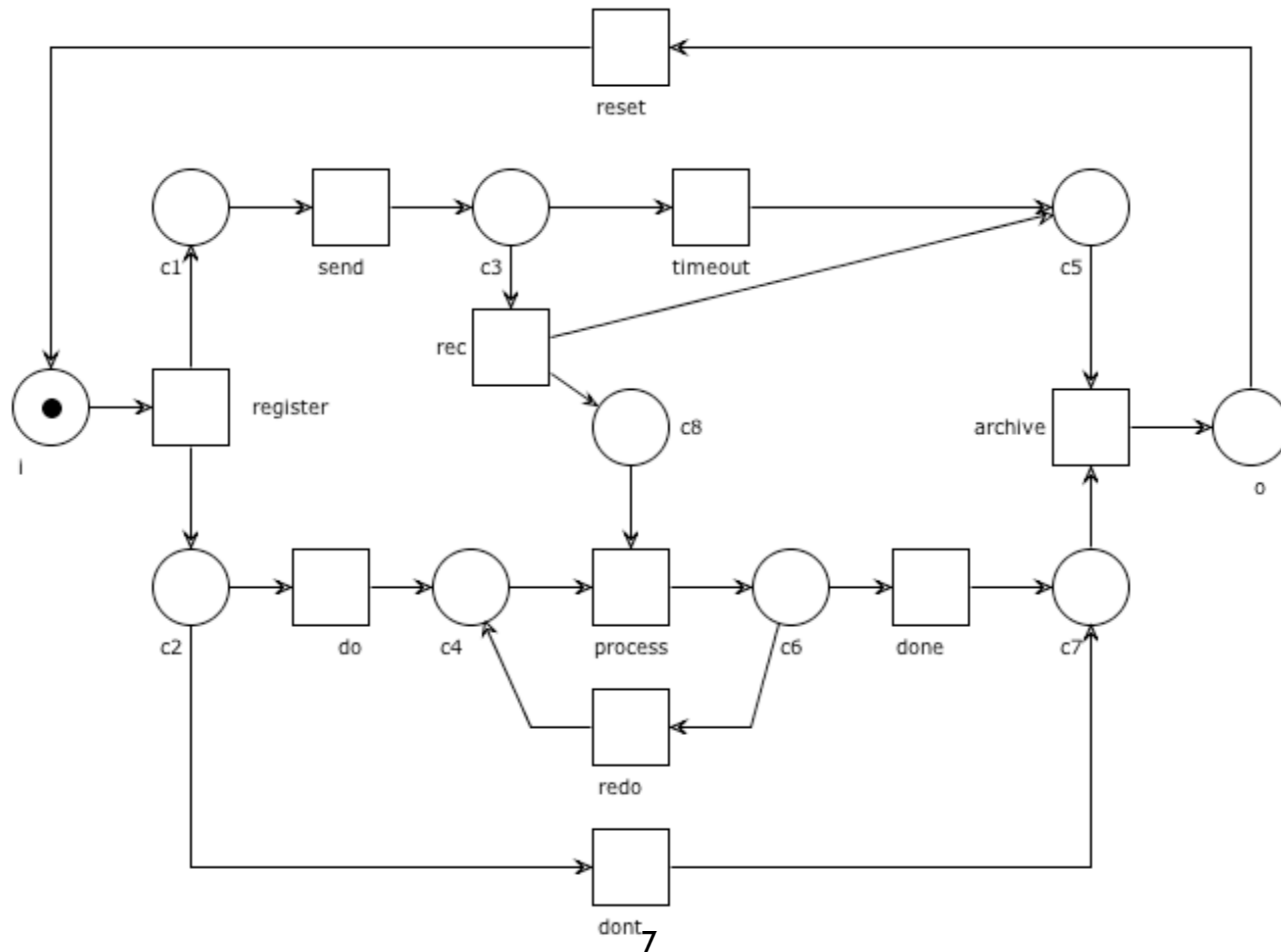
Running example

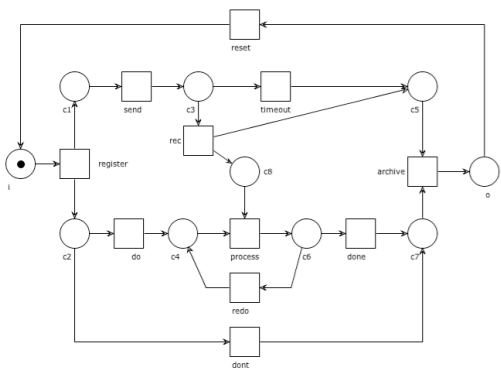


Running example

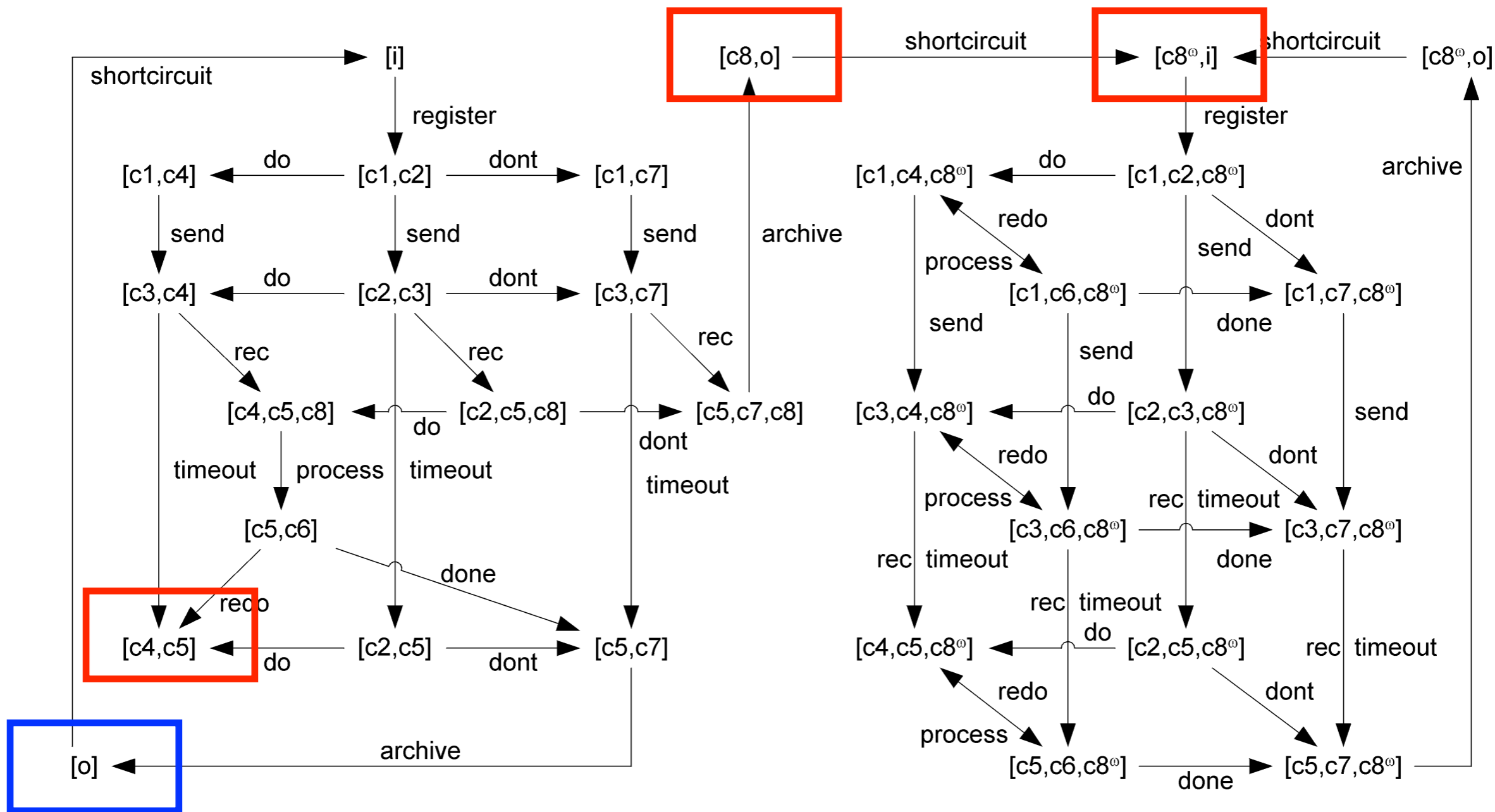


Running example: short-circuited





Running example: short-circuited



S-coverability diagnosis

Quick reminder

In a S -component,
the total number of tokens in its places is constant

Any S -component
induces a uniform invariant (weights 0 and 1)

A net is **S -coverable** iff
any $p \in P$ belongs to some S -component

S -coverability implies boundedness
(because it induces a positive S -invariant)

S-Invariant analysis

If every place of N^* is covered by a semi-positive S-invariant
then N^* is bounded

**Places not covered by semi-positive S-invariants
are potential sources of problems**

S-Coverability vs Soundness

S-coverability is one of the basic requirements any workflow process definition should satisfy

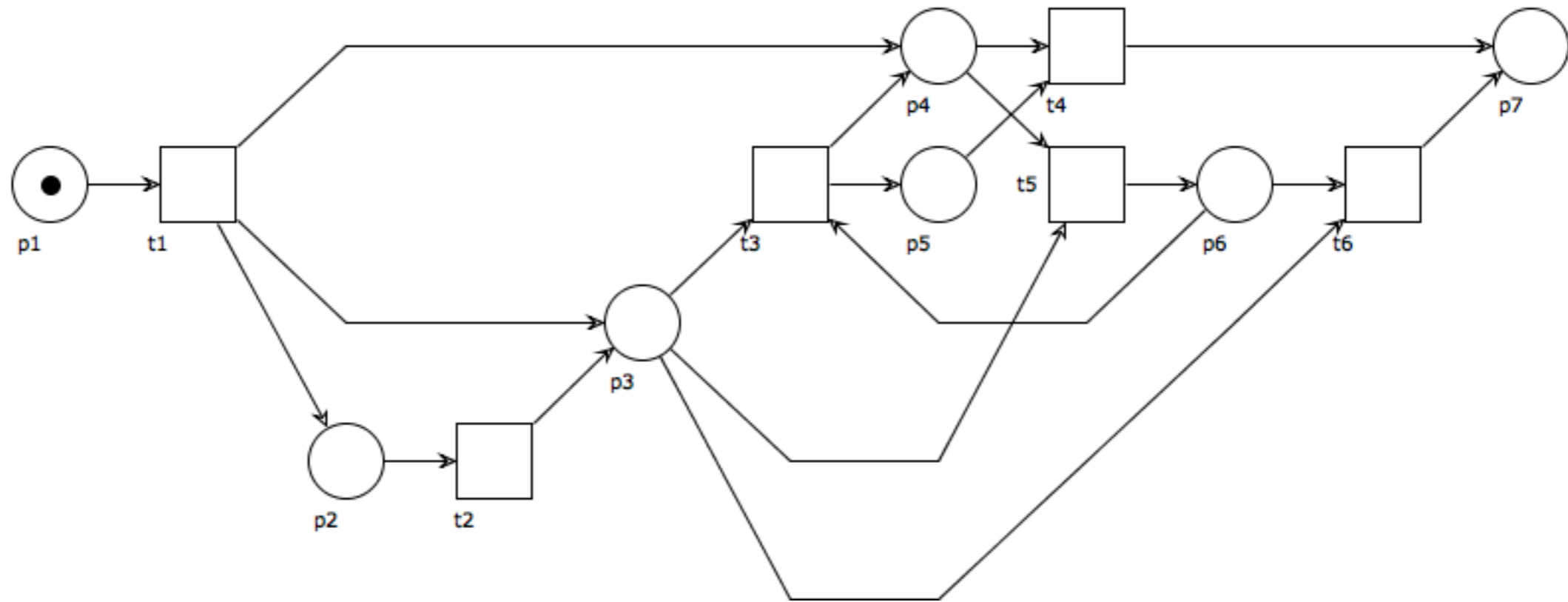
Still:

S-coverability is not a sufficient requirement for soundness

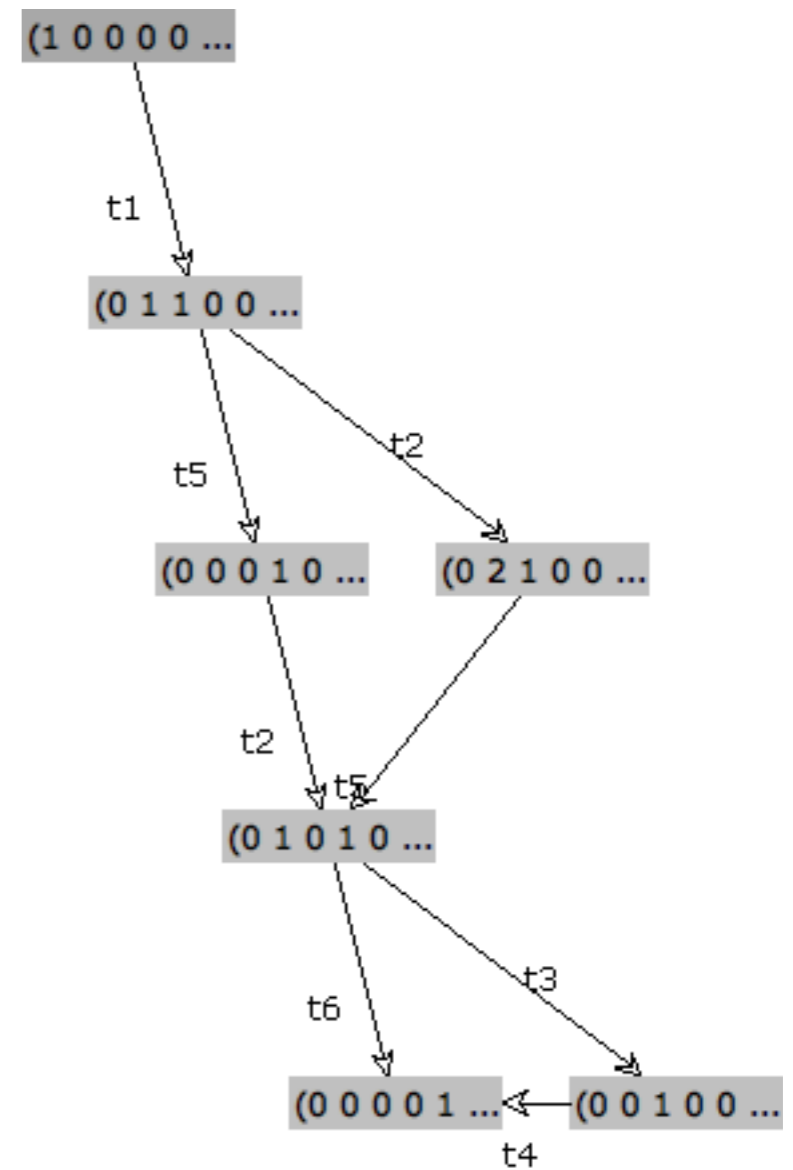
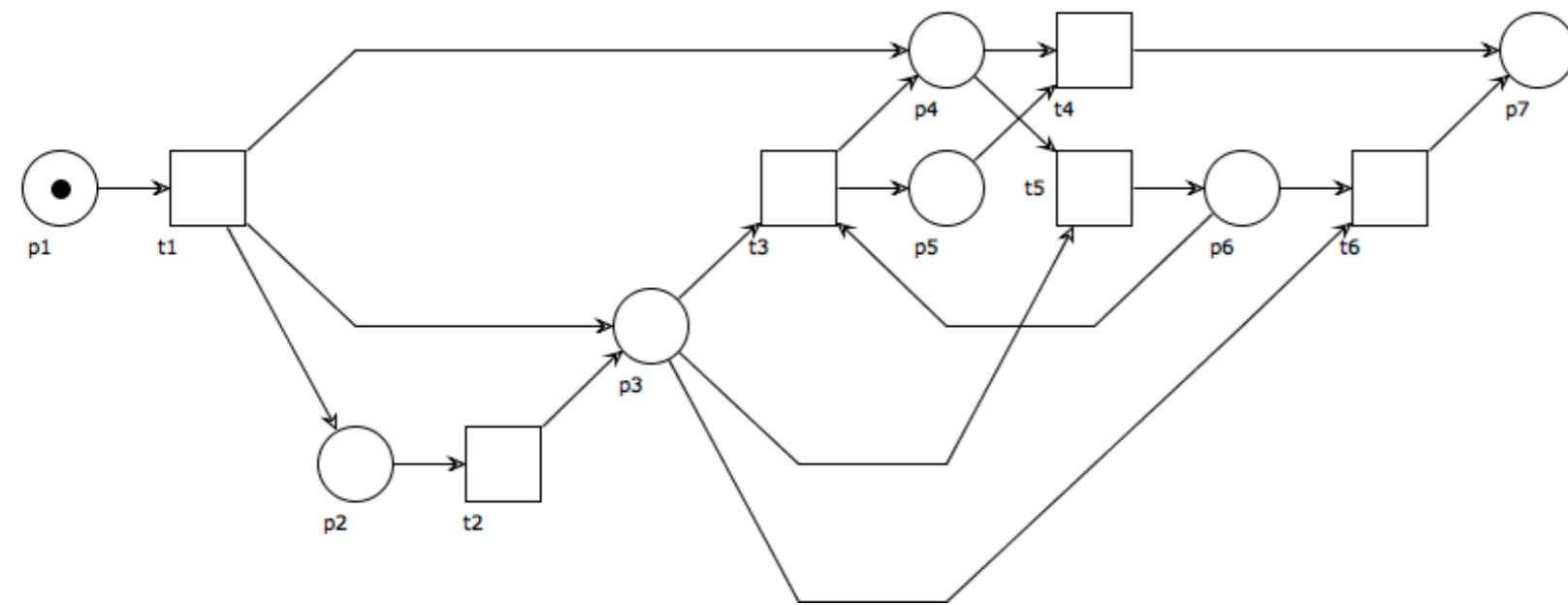
N^* can be S-coverable even if N is not sound

N can be sound even if N^* is not S-coverable

Example: N sound but
 N^* not S -coverable

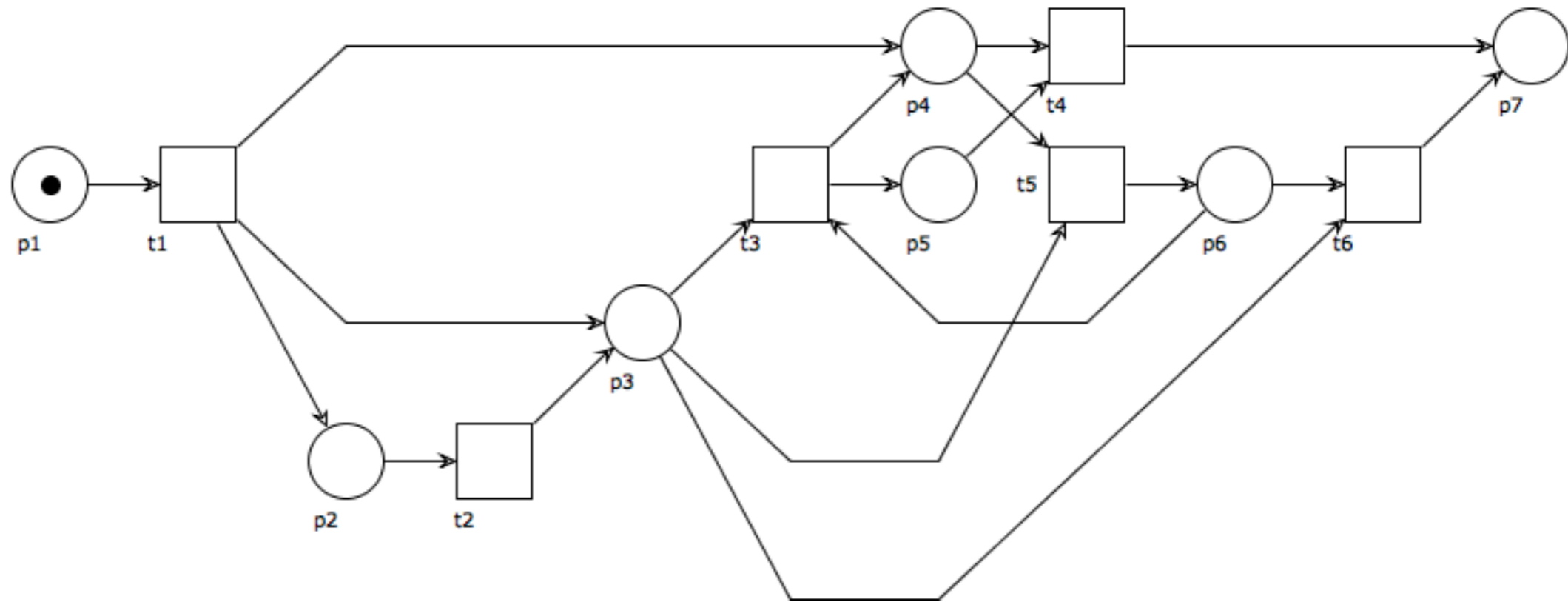


Example: N sound but N^* not S -coverable



Exercises

Find all (maximal) S-components using WoPeD



Exercises

Draw a workflow net N that is S -coverable but such that N^* is not live and bounded (i.e. N is not sound)

S-Coverability diagnosis

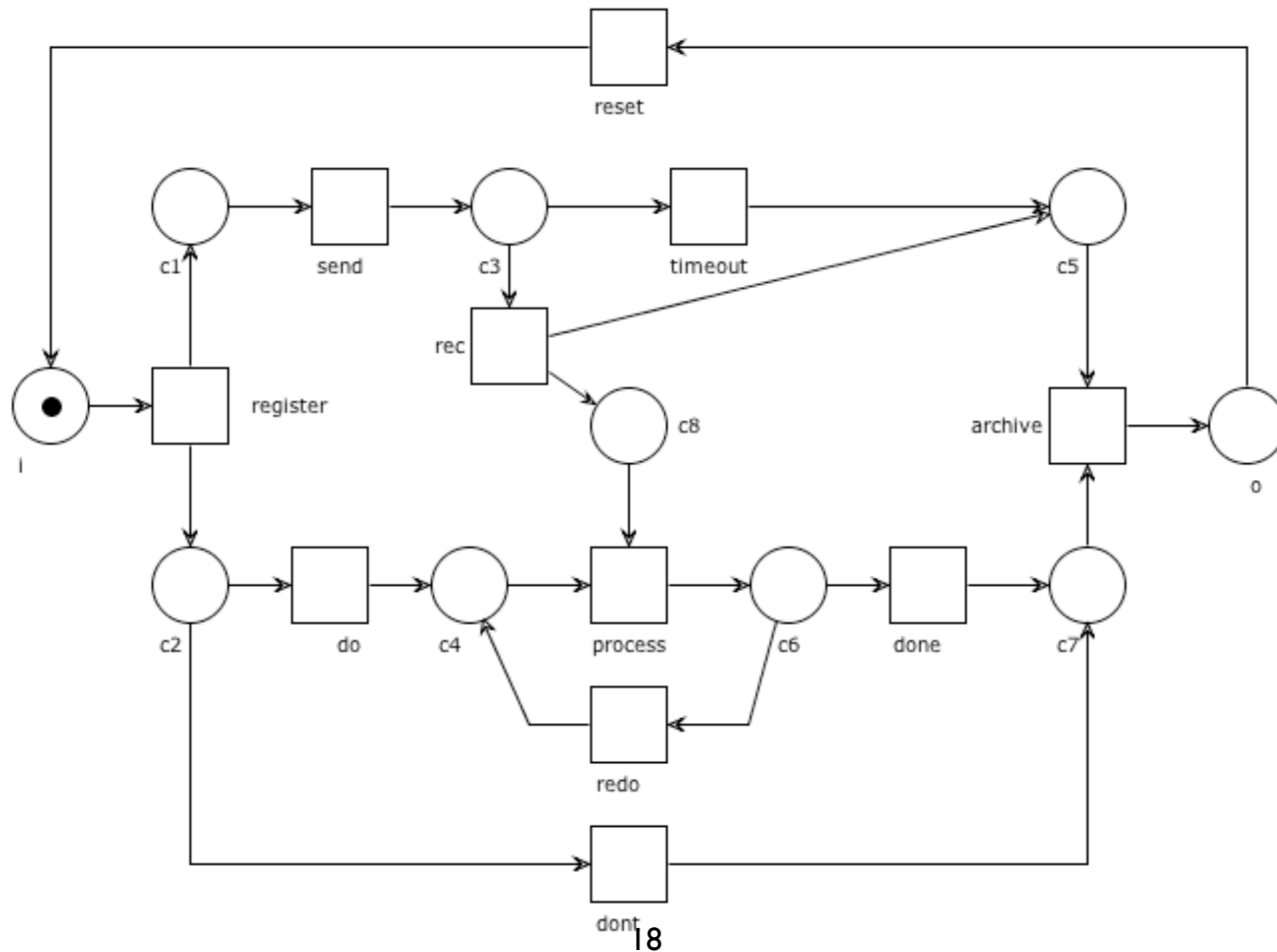
If N^* is free-choice, live and bounded
it must be S-coverable (S-coverability theorem)

(note that any S-component of N^* includes i, o, reset,
by strong-connectedness)

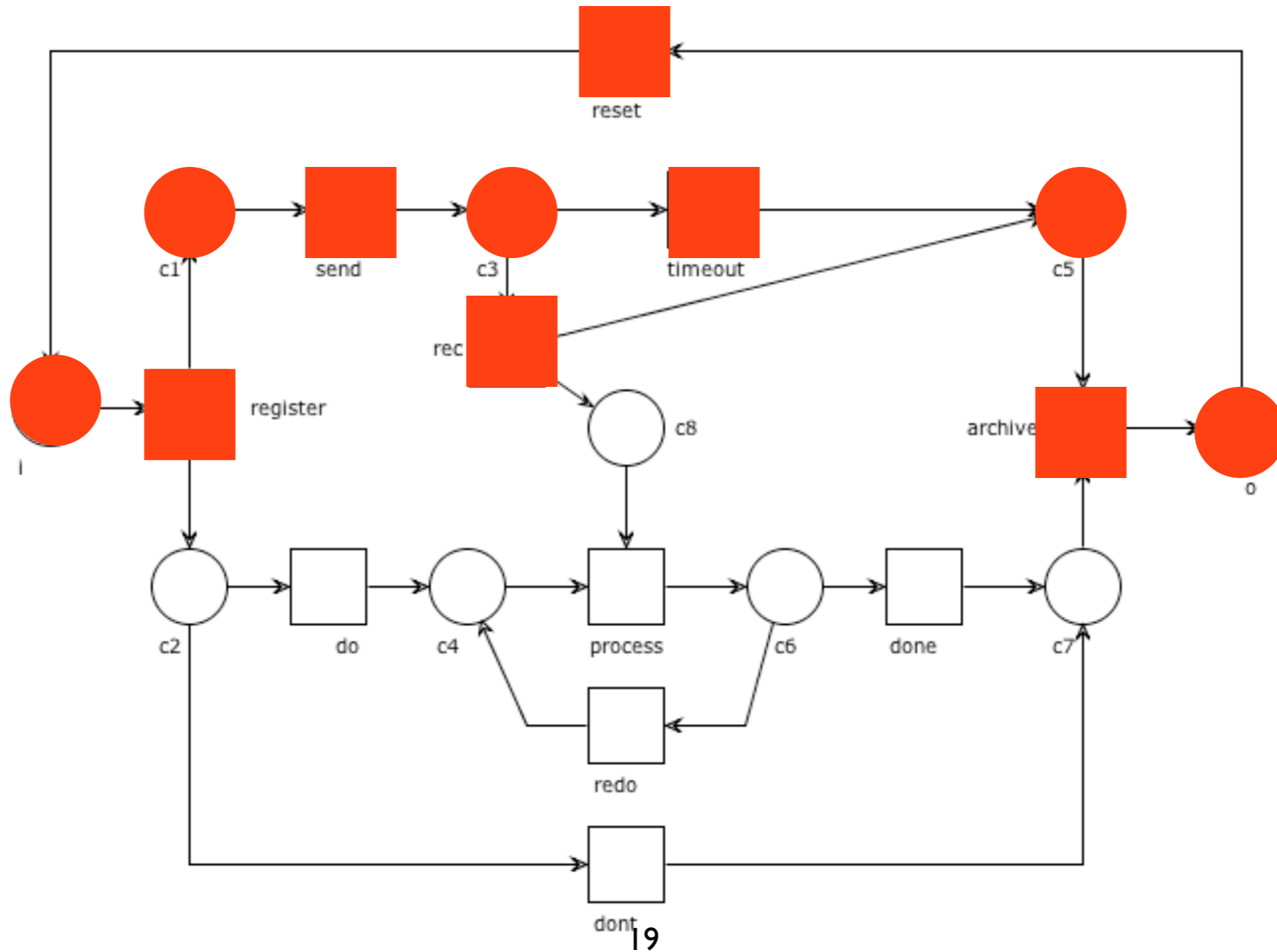
Corollary: If N is sound and free-choice,
then N^* must be S-coverable

N free-choice + N^* not S-coverable $\Rightarrow N$ not sound

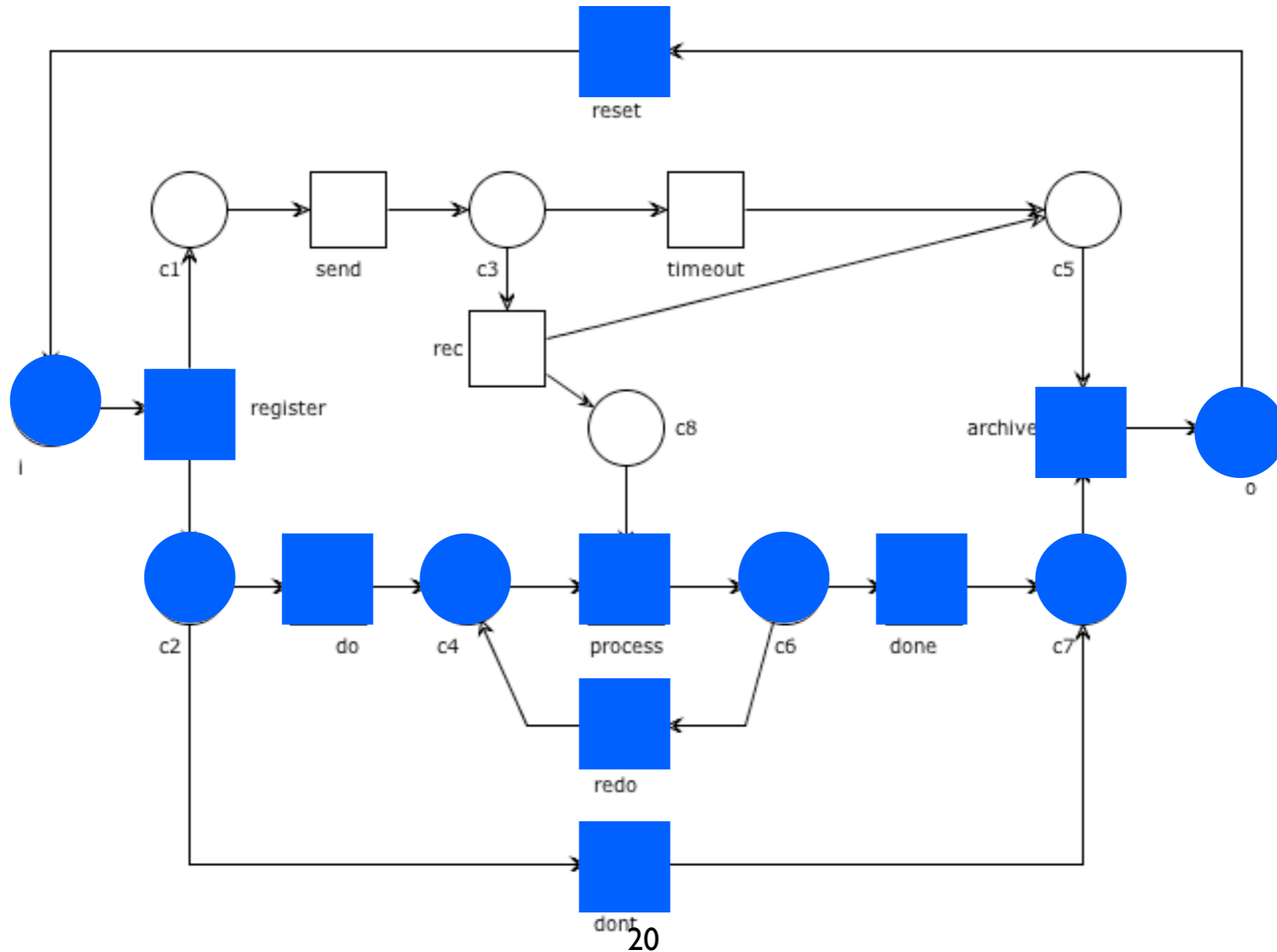
Running example: S-cover for N^* ?



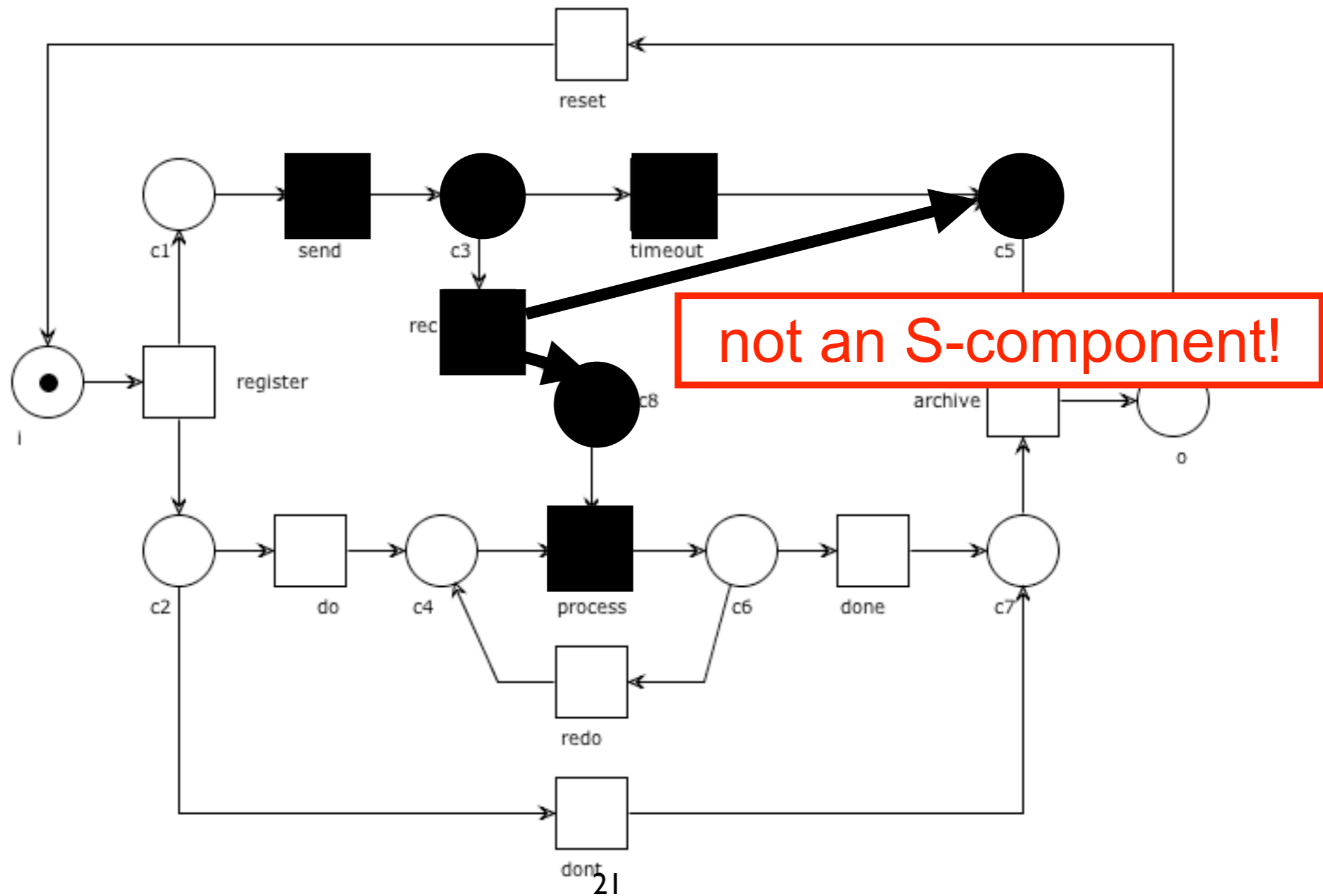
Running example: S-cover for N^* ?



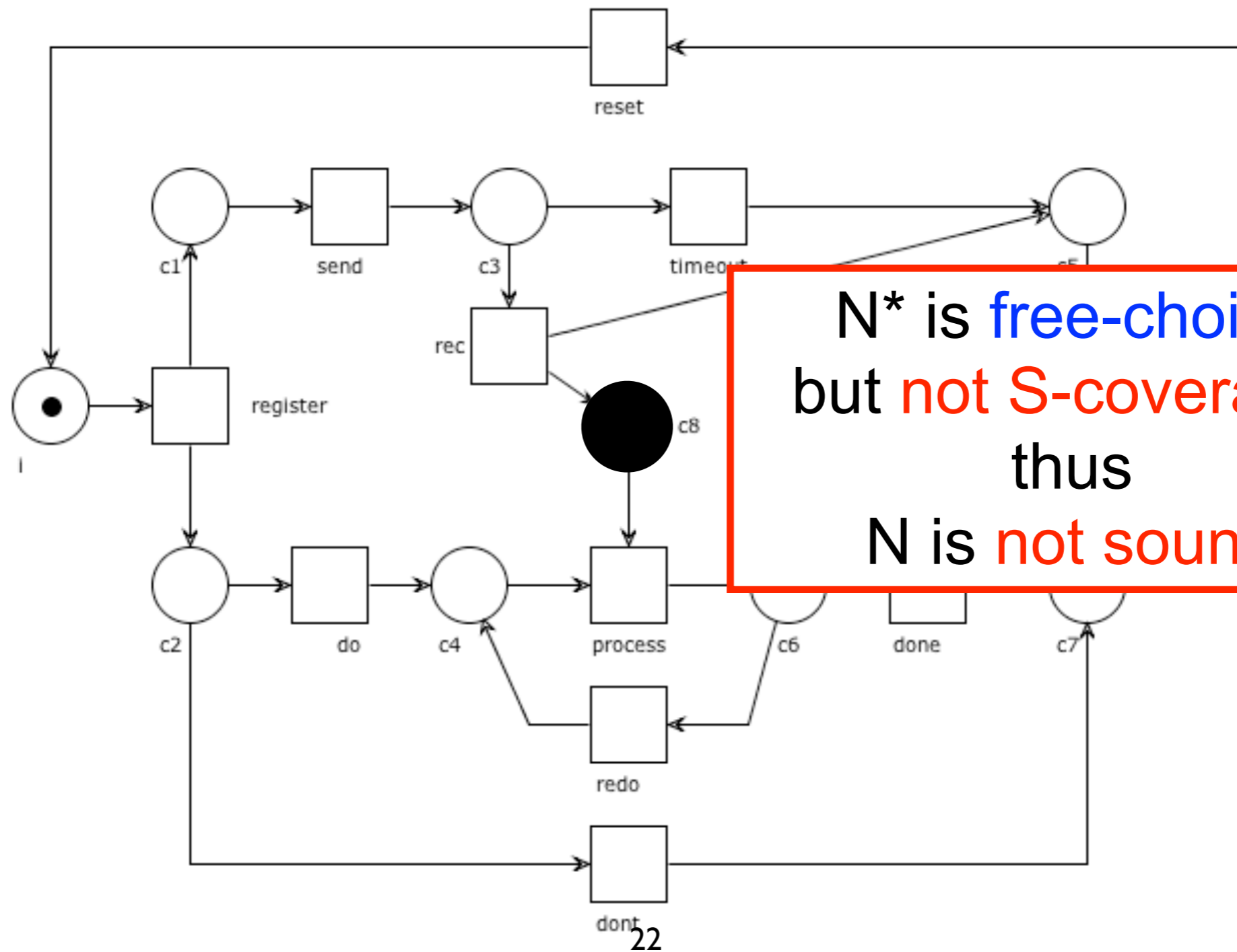
Running example: S-cover for N^*



Running example: S-cover for N^*



Running example: S-cover for N^* ? No



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

Running Example: WoPeD Diagnosis

The screenshot displays a software interface for analyzing a Petri net. The main window shows a workflow net with places c_1 through c_8 and transitions labeled 'register', 'send', 'rec', 'timeout', 'do', 'process', 'redo', 'done', 'dont', and 'archive'. The initial marking is at place c_1 and the final marking is at place c_8 . The sidebar on the right, titled 'Assistant Expert', provides a detailed diagnosis:

- 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-Component:
 - c_8
 - 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

At the bottom of the window, the following statistics are shown: Places: 10, Transitions: 10, Subprocesses: 0, Modeling direction: horizontal. The zoom level is 100% and the file is not saved.

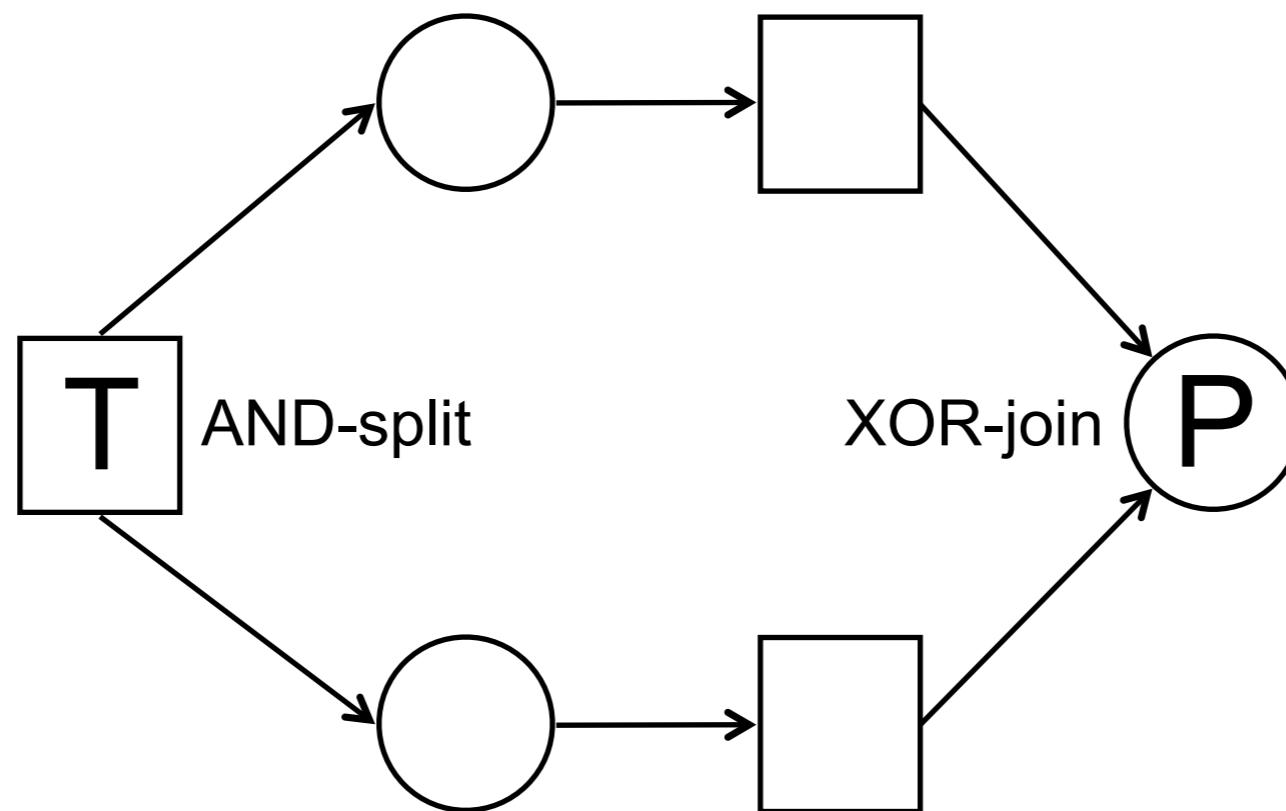
Split / Join Balancing

A good workflow design is characterized by a balance between AND/XOR-split and AND/XOR-joins

Any mismatch is a potential source of errors

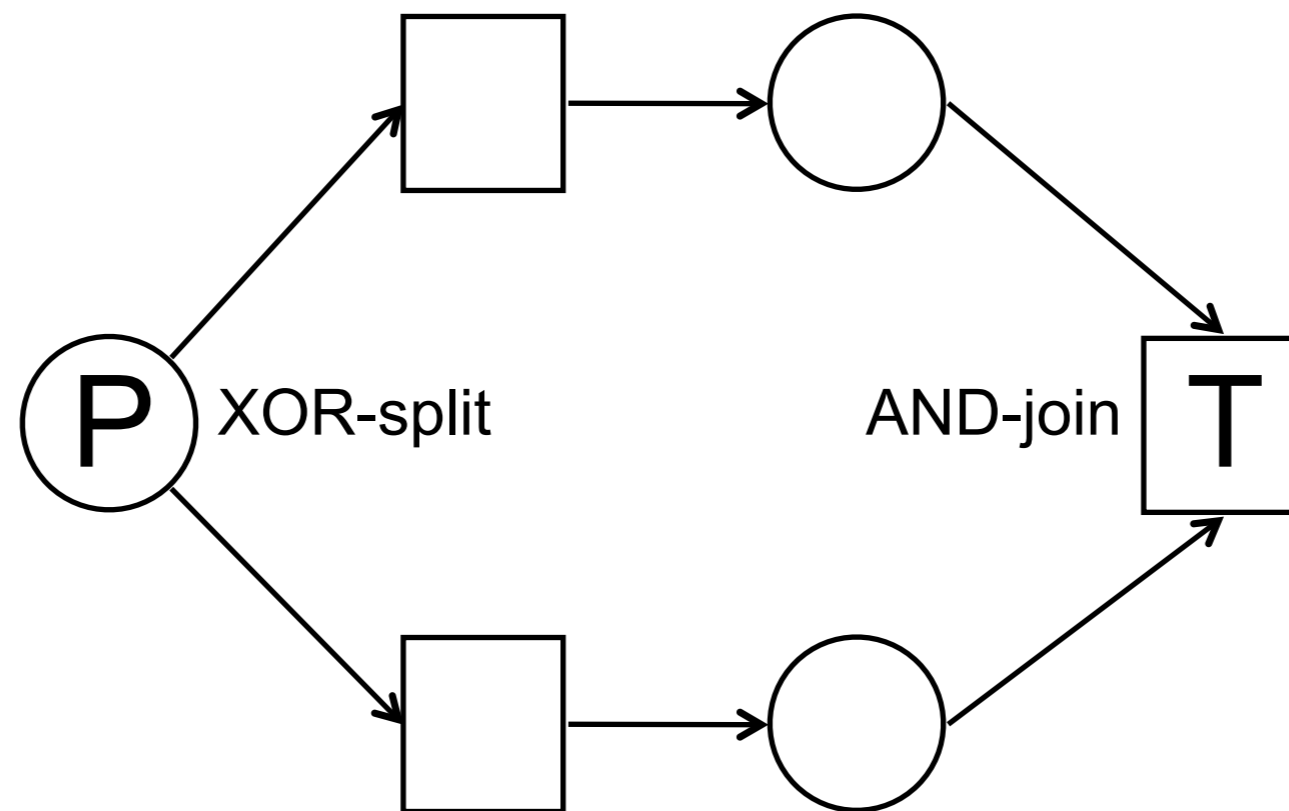
TP-handles

Two parallel flows initiated by an AND-split
should not be joined by a XOR-join
(multiple tokens appear in the same place)



PT-handles

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



TP- and PT-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

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

Well-Structured Nets

Definition: A net is **well-handled** iff
for any pair of nodes x and y of different kinds
(one place and one transition)
any two elementary paths c_1 and c_2 from x to y
coincide or have some other nodes in common apart x, y

well-handled = no PT-handles and no TP-handles

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

S-coverability diagnosis

Theorem:

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

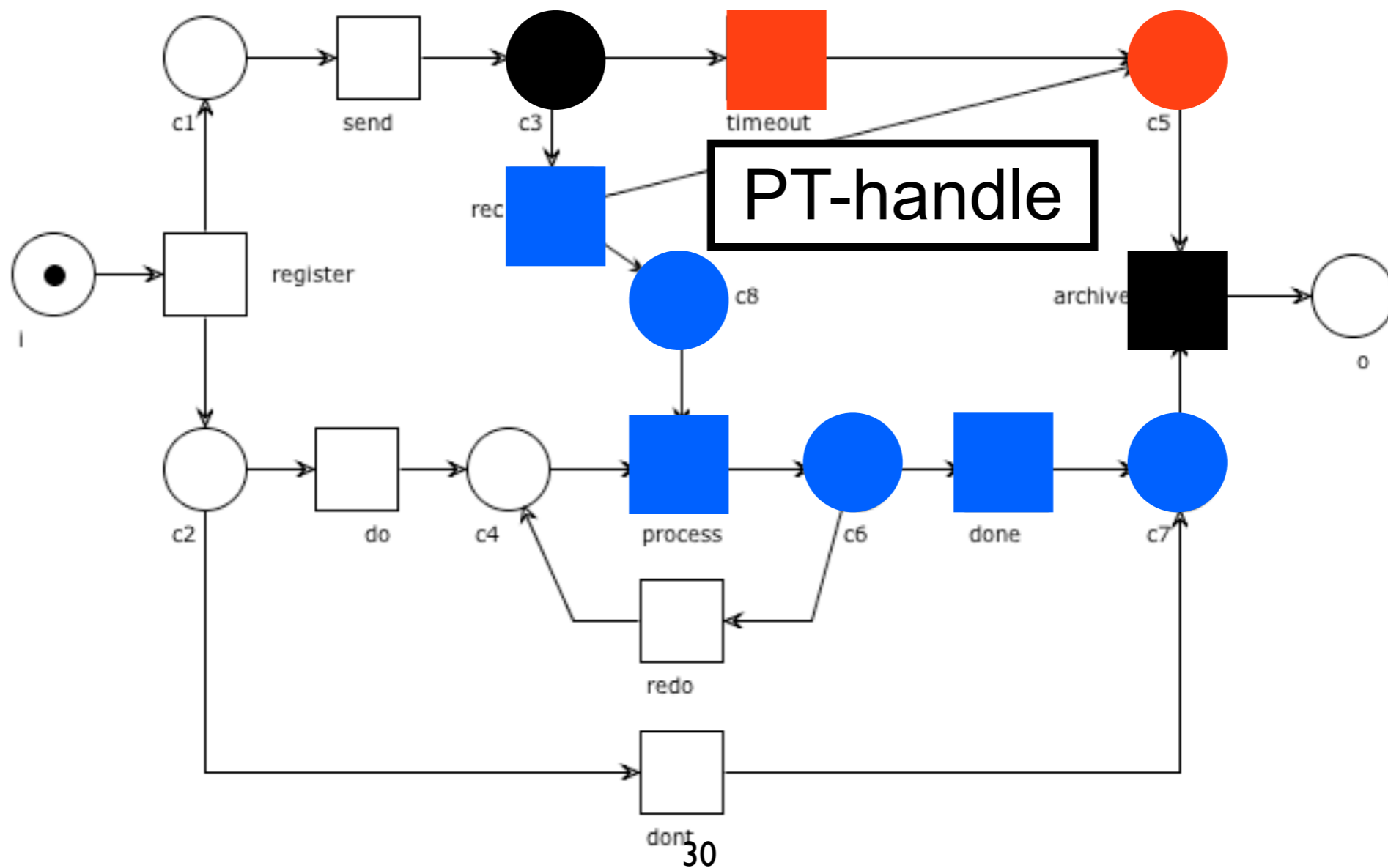
N well-structured + N^* not S-coverable \Rightarrow N not sound

Note that

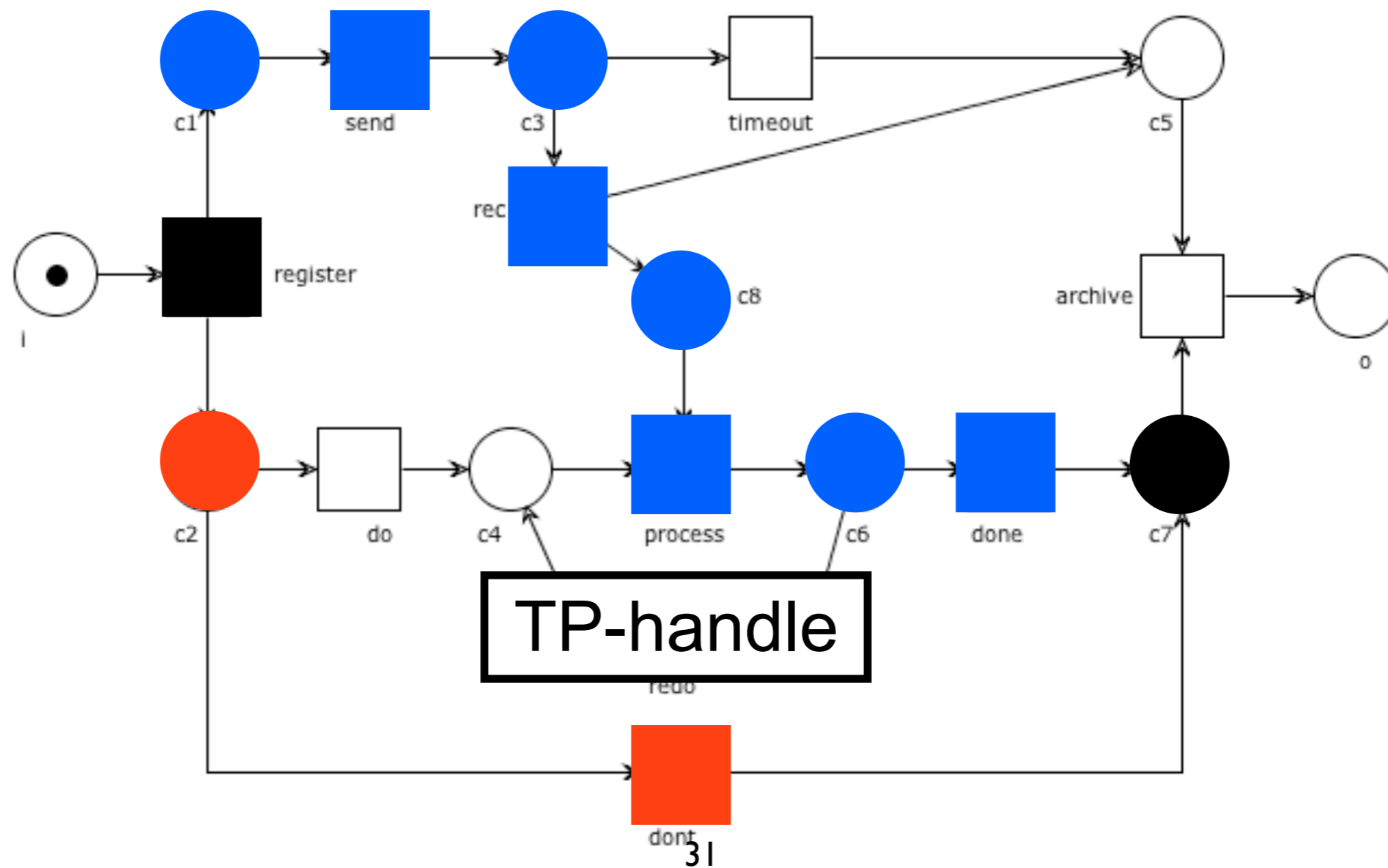
If N is not well-structured, N can still be sound

especially if reset is involved in the handle
(it is a symptom, not a disease)

Running example: Well-structured? No



Running example: Well-structured? No



Be careful

N well-structured = N^* well-handled

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

Running Example: WoPeD Diagnosis

The screenshot displays a software interface for analyzing a Petri net. The main window, titled "wfnet-unsound.pnml", shows a Petri net with places c_1 through c_8 and transitions labeled "register", "send", "rec", "timeout", "do", "process", "redo", "dout", "archive", and "o". Places c_3 and "process" are highlighted in red. The interface includes tabs for "Process", "Resources", and "BPEL preview".

On the right, a "Semantical analysis" panel is open, showing a tree structure of analysis results:

- 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

At the bottom of the window, the status bar shows "Horizontal", "Zoom: 100%", and "Not saved".

Liveness and boundedness

VS

Soundness requirements

Improper completion

Suppose **N completes improperly:**
from i we can reach $i+L$

We can do the same on N^*
then we fire reset and reach $i+L$

we can repeat the same run and reach $i+2L$
and then $i+3L$ and then $i+4L$ and then ... $i+kL$

then **N^* has some unbounded places**
(all p such that $L(p) > 0$)

Unsoundness from unboundedness

Improper completion of N implies unboundeness of N^*

Symptom: N^* has some unbounded places

Disease: maybe proper completion does not hold for N

Consequences of boundedness

If N^* is bounded, then:
if $o+L$ is reachable from i in N , then $L=0$

If N^* is bounded, then
either N satisfies
both option to complete and proper completion
or N does not satisfy option to complete

Completion option failure

Suppose N does not satisfy the “option to complete”:
then from i we can reach M
from which we cannot mark o

We can do the same on N^*
then reset is dead from M
i.e. reset is non-live in N^*

N^* has non-live transitions (including reset)

Unsoundness from non-liveness

Option to complete fail for N implies non-liveness of N^*

Symptom: reset transition is non-live in N^*

Disease: maybe option to complete does not hold for N

Unsoundness from Non-Liveness

If N^* is bounded and has dead transitions, then

if reset is dead

N and N^* have the same finite reachability graph

hence N has the same dead tasks as N^*

(except reset)

if reset is not dead

the reachability graphs of N and N^* differ only for $o \xrightarrow{\text{reset}} i$

(because N^* is bounded)

hence N has the same dead tasks as N^*

Unsoundness from Non-Liveness

Symptom: N^* has non-live transitions

Disease: N could have dead tasks

Symptom: N^* is bounded and has dead transitions

Disease: N has the same dead tasks as N^*

Error sequences

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
(pointing to different types of errors)

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

Behavioural error sequences can overcome this problem

Error sequences

Rationale:

We want to find firing sequences such that:

1. every continuation of such sequences will lead to an error
2. they have minimal length
(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”)

Non-Live sequences: informally

A **non-live sequence** is a firing sequence of **minimal length** 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)

Non-Live sequences: remarks

No **red** node implies no **yellow** node

No **green** node implies no **yellow** node

Non-Live sequences: formally

Definition:

An occurrence sequence

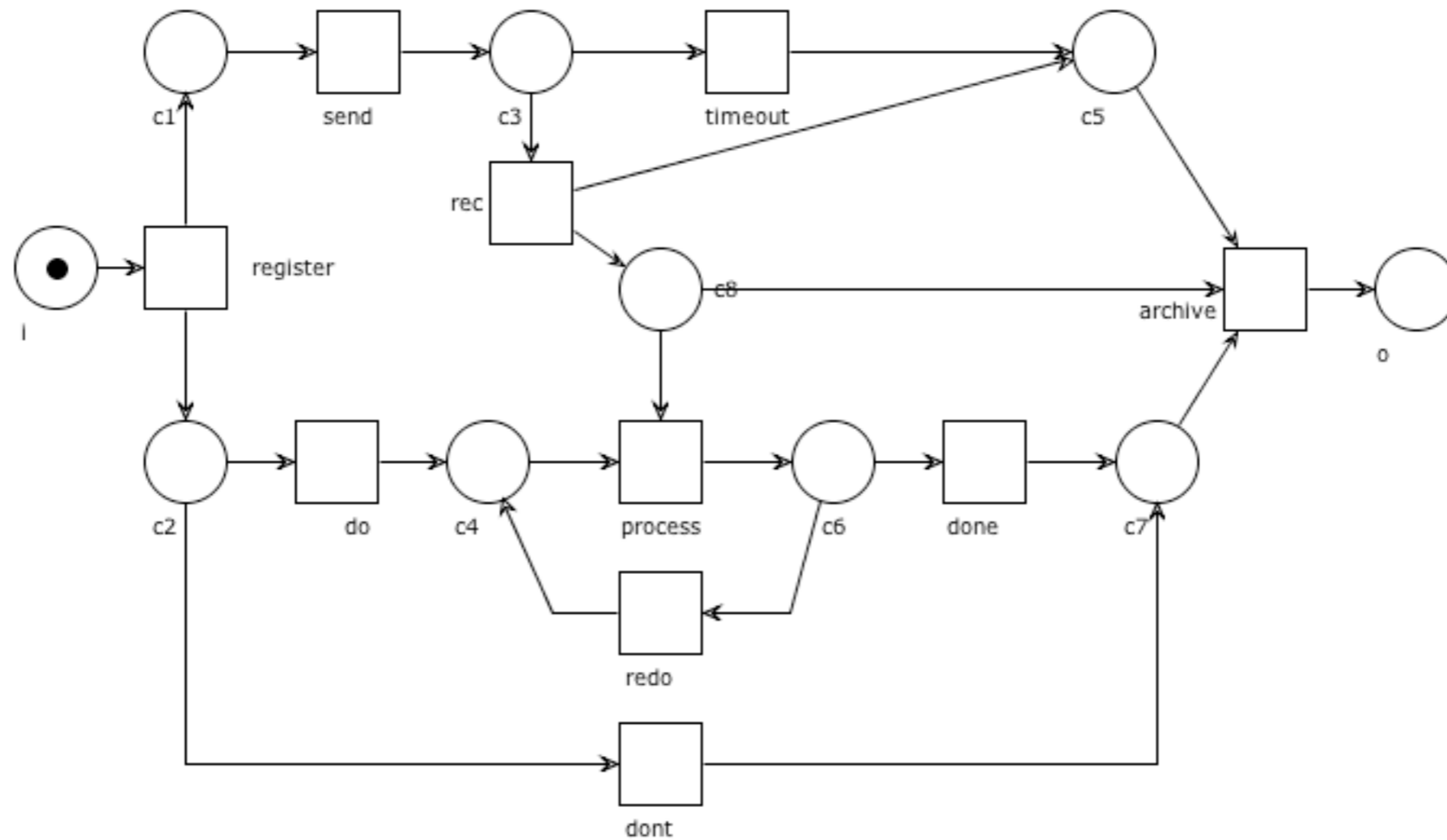
$i \xrightarrow{t_1} M_1 \dots M_{k-1} \xrightarrow{t_k} M_k$ is **non-live** if

- all markings are distinct
- M_{k-1} is yellow
- M_k is red

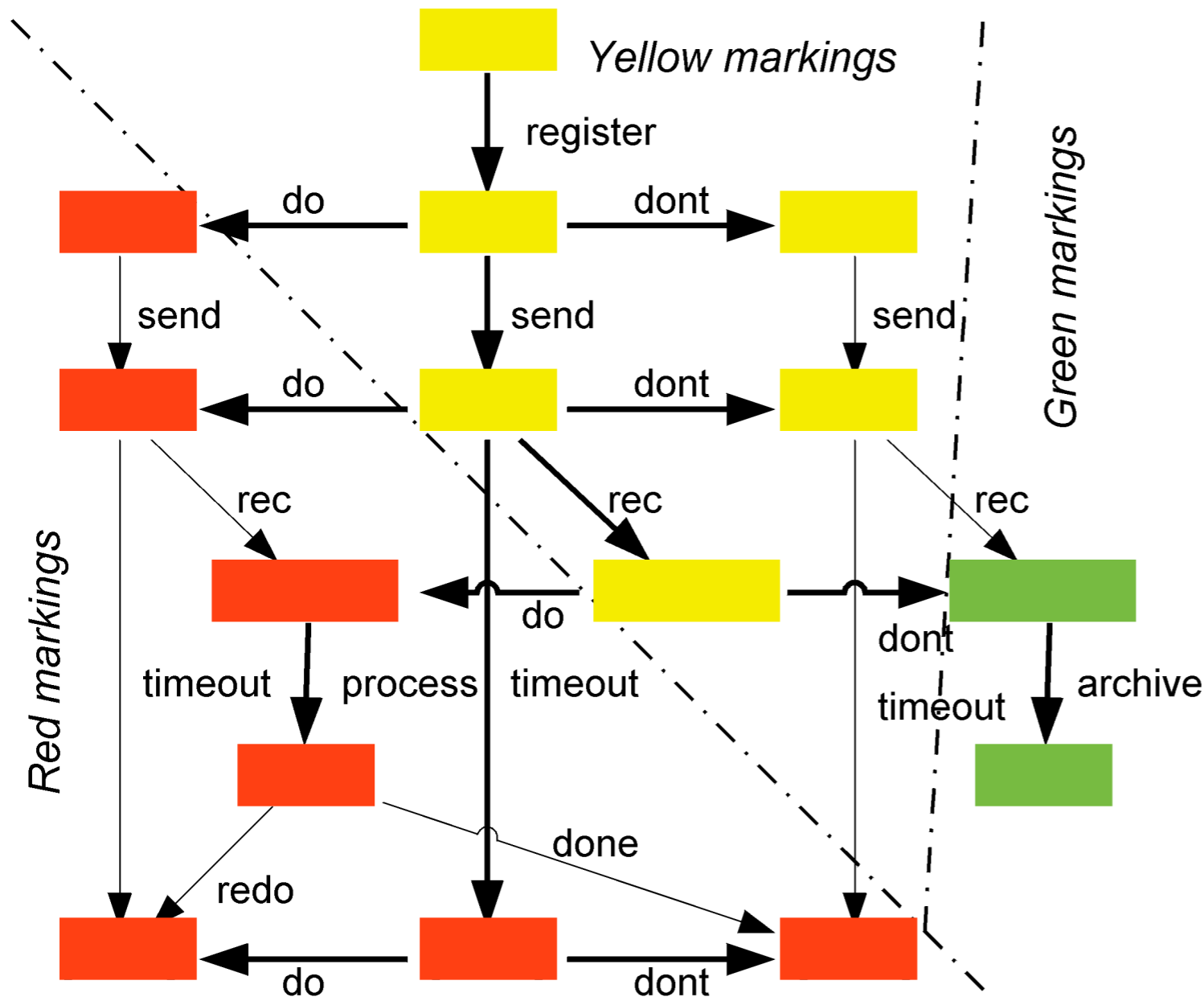
Firing t_k removes
the option to complete!

Then, the firing sequence $t_1 \dots t_k$ is also called **non-live**

Running example: slight variant



Running example variant: colored RG



Non-live sequences:

register, do

register, send, do

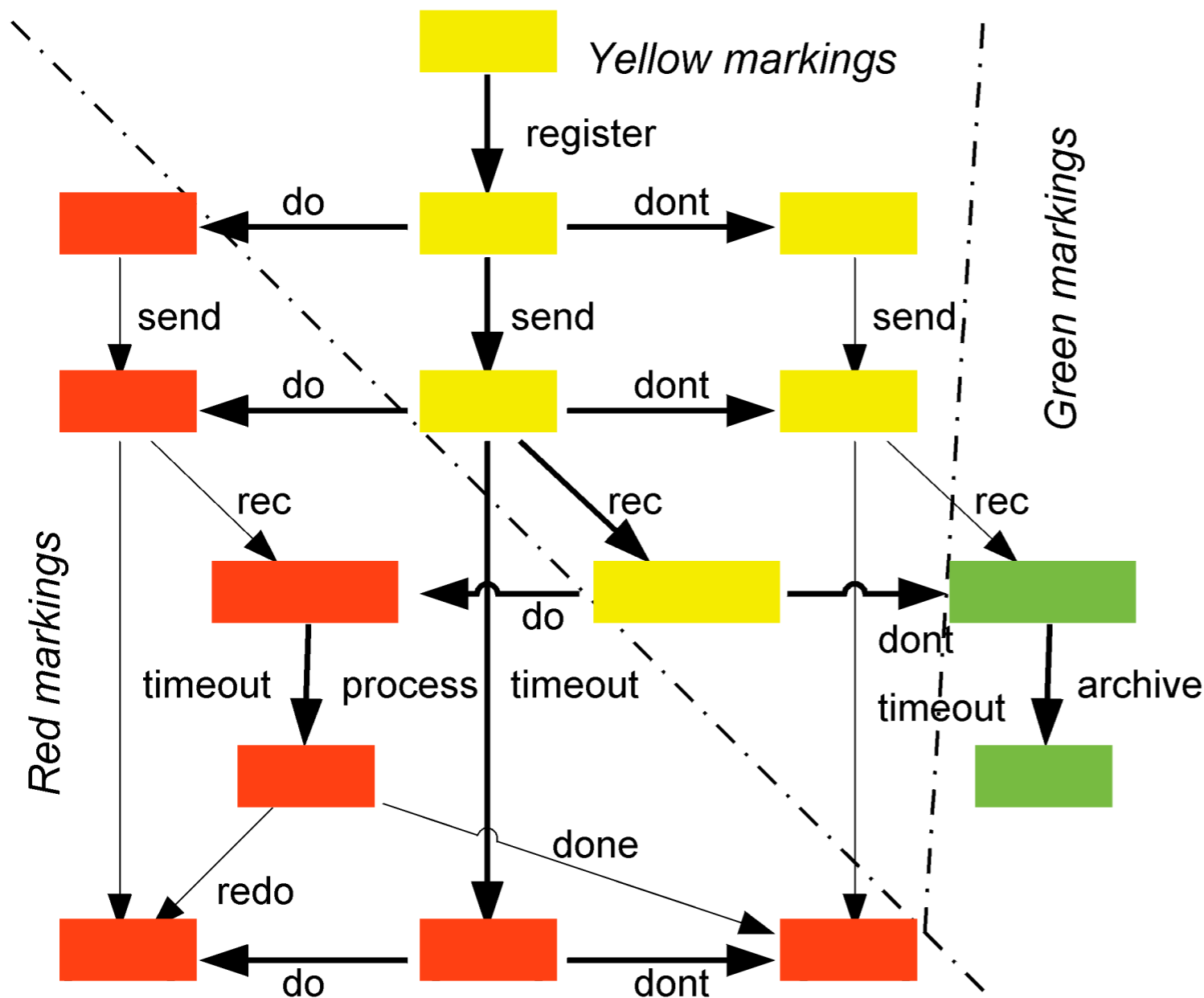
register, send, timeout

register, send, rec, do

register, send, dont, timeout

and also?

Running example variant: colored RG



Non-live sequences:

register, do

register, send, do

register, send, timeout

register, send, rec, do

register, send, dont, timeout

register, dont, send, timeout

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 0

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)

Unbounded sequences: remarks

No **red** node implies no **yellow** node

No **green** node implies no **yellow** node

Restricted coverability graph (RCG)

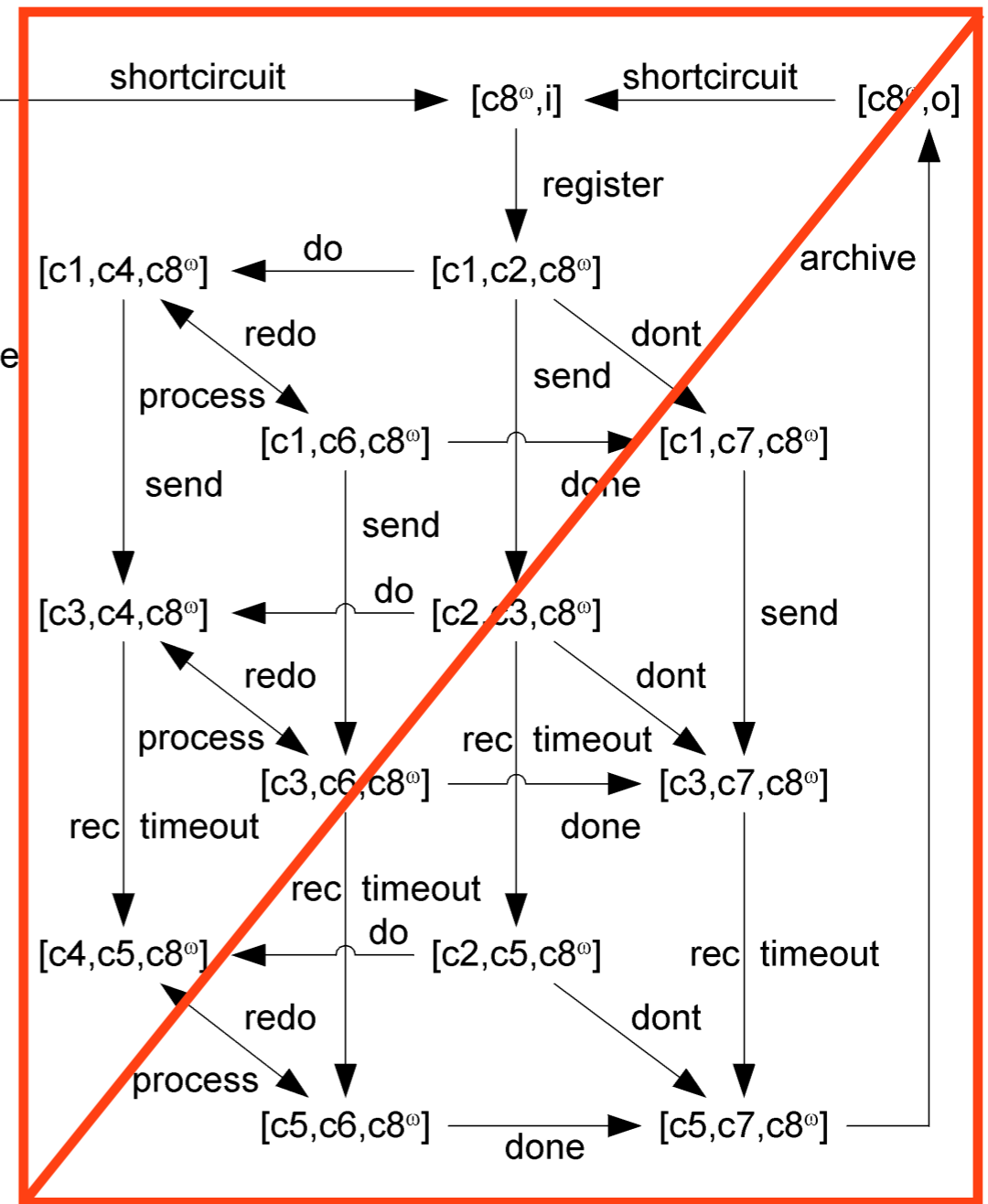
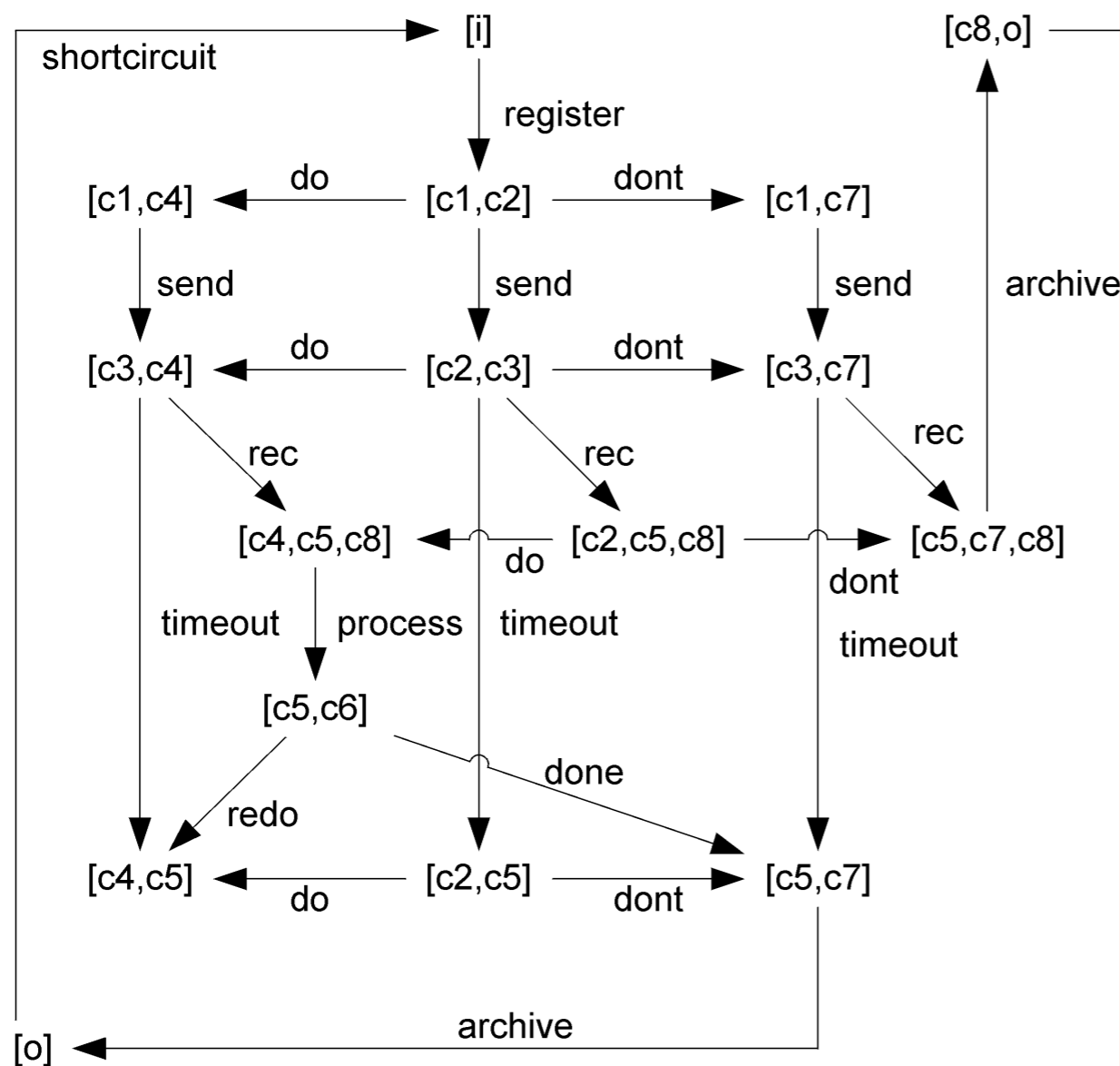
CG can become very large (intractable!)

Basic observation:

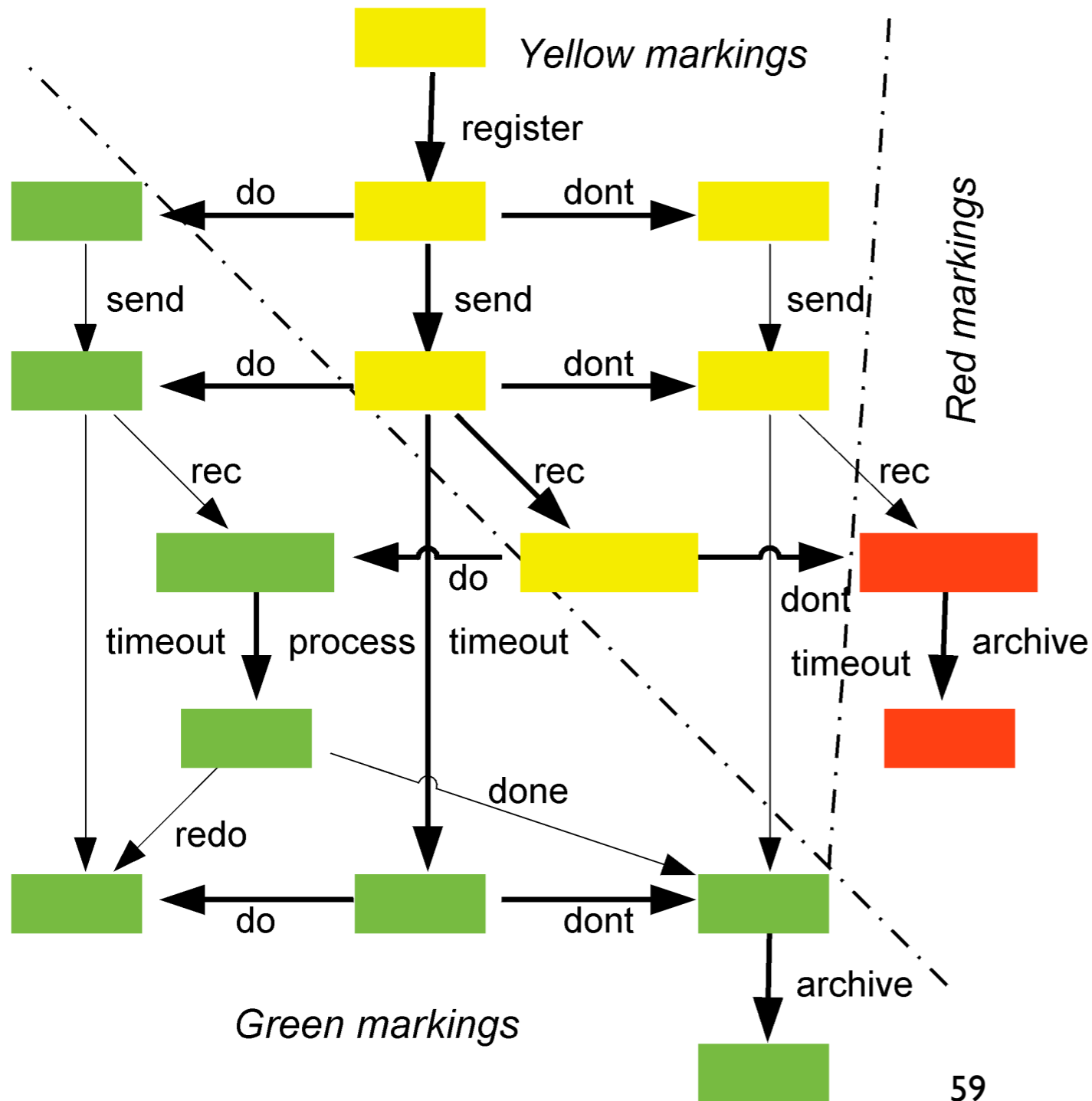
infinite-weighted markings leads to infinite-weighted markings
and they will be all red

We can just avoid computing them!

Running example: RCG vs CG

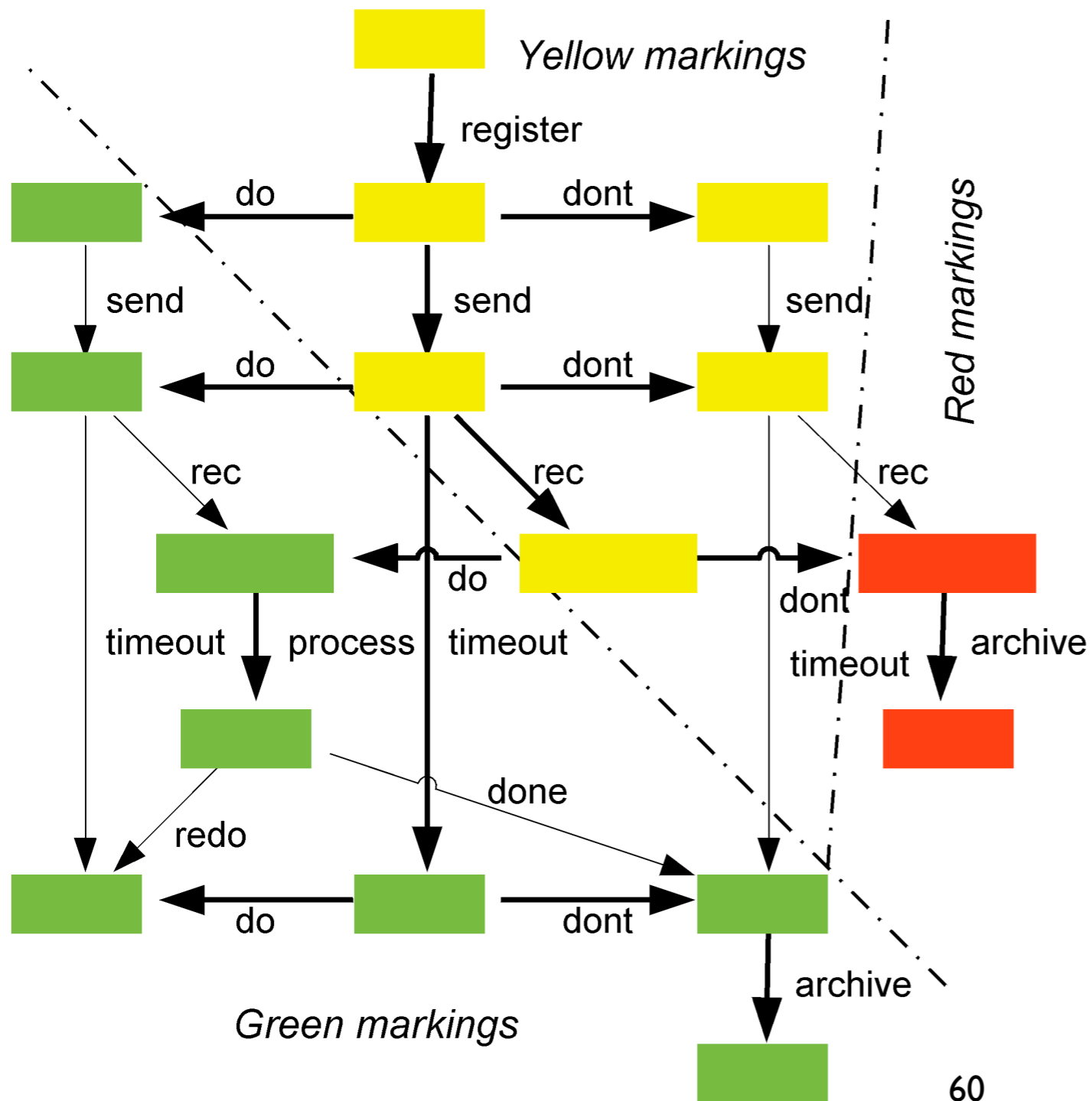


Running example: colored RCG



Unbounded sequences:
 register, dont, send, rec
 register, send, dont, rec
 and also?

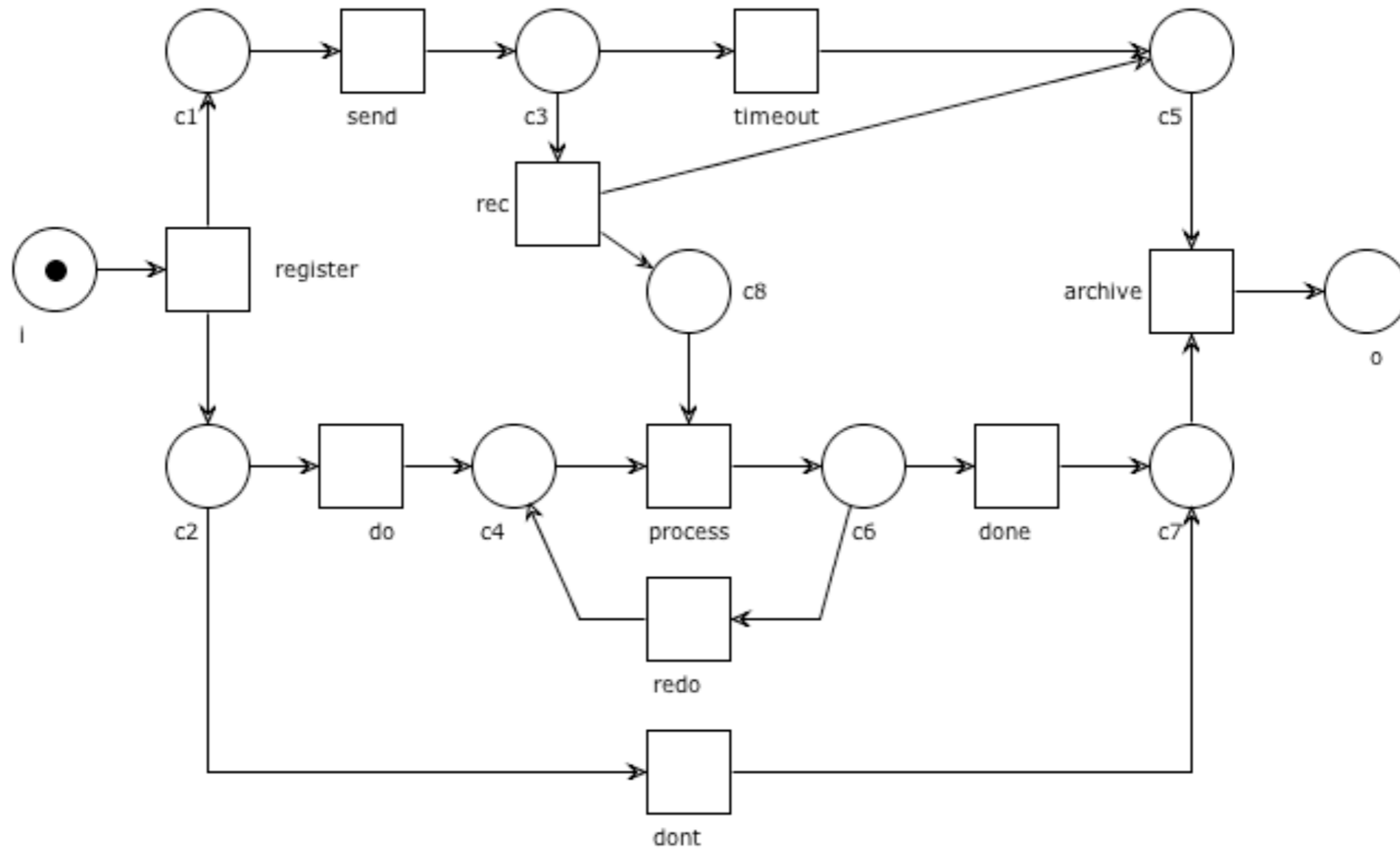
Running example: colored RCG



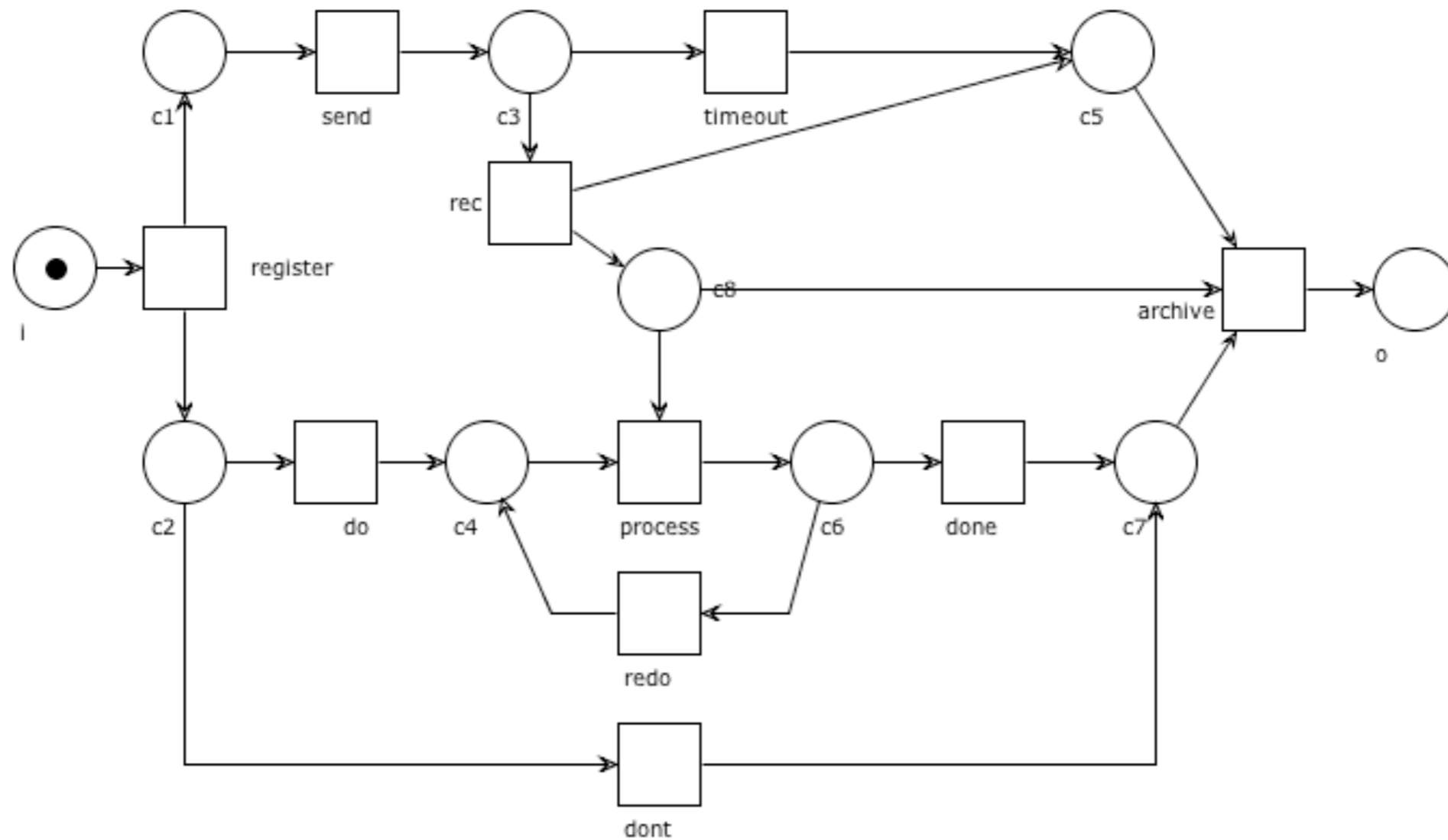
Unbounded sequences:
 register, dont, send, rec
 register, send, dont, rec
register, send, rec, dont

Practice with WoPeD (and Woflan)

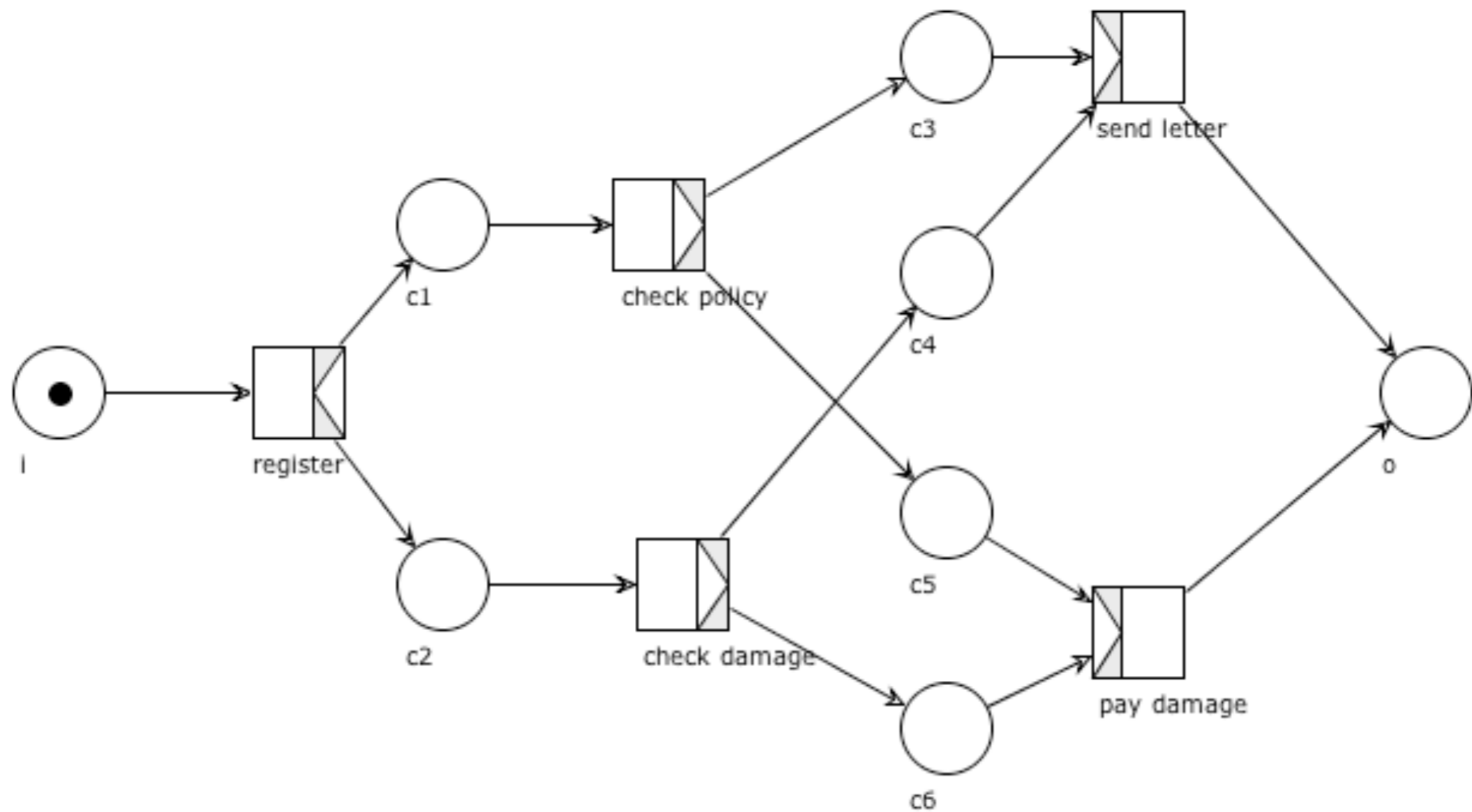
Analyse the running example



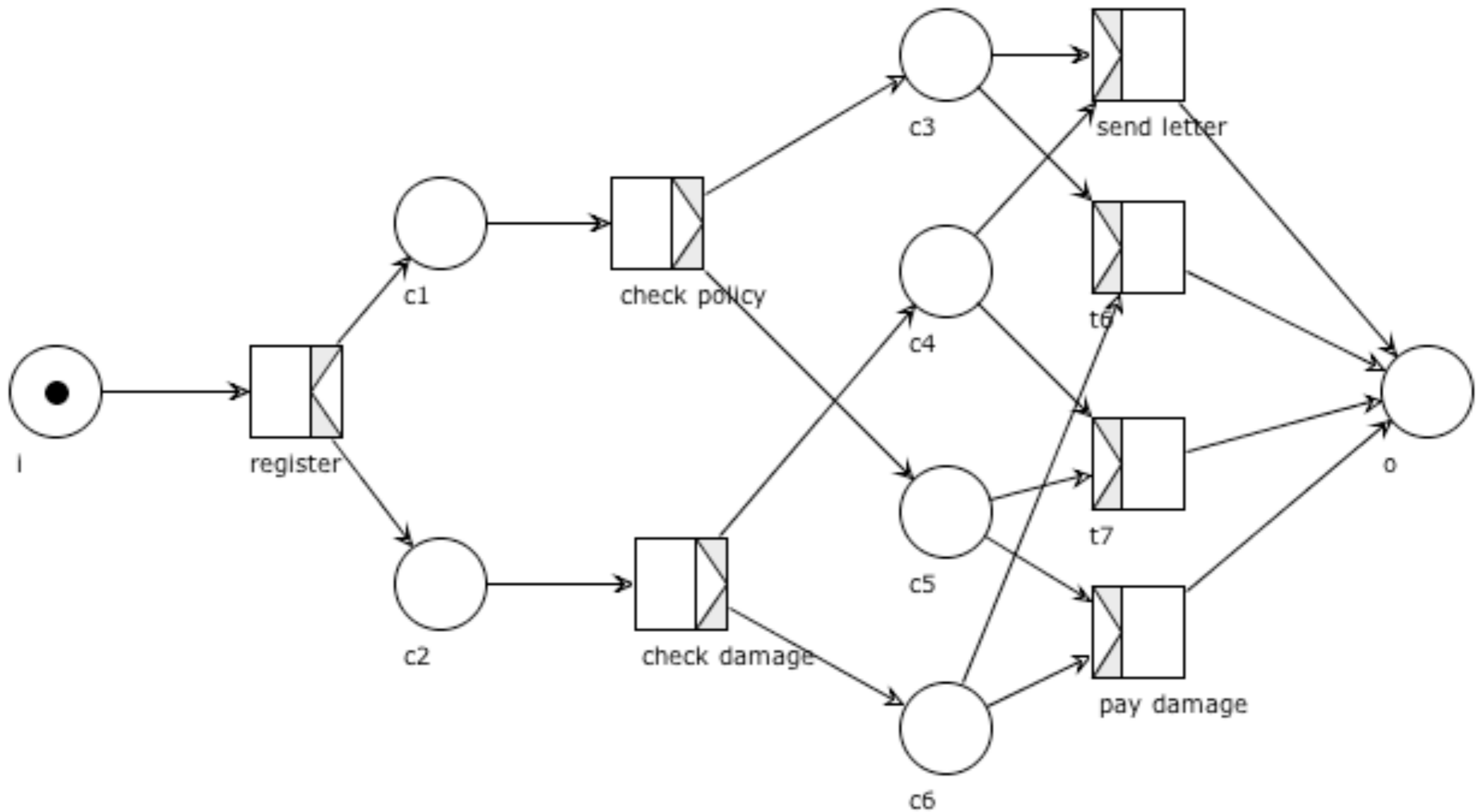
Analyse the running example variant



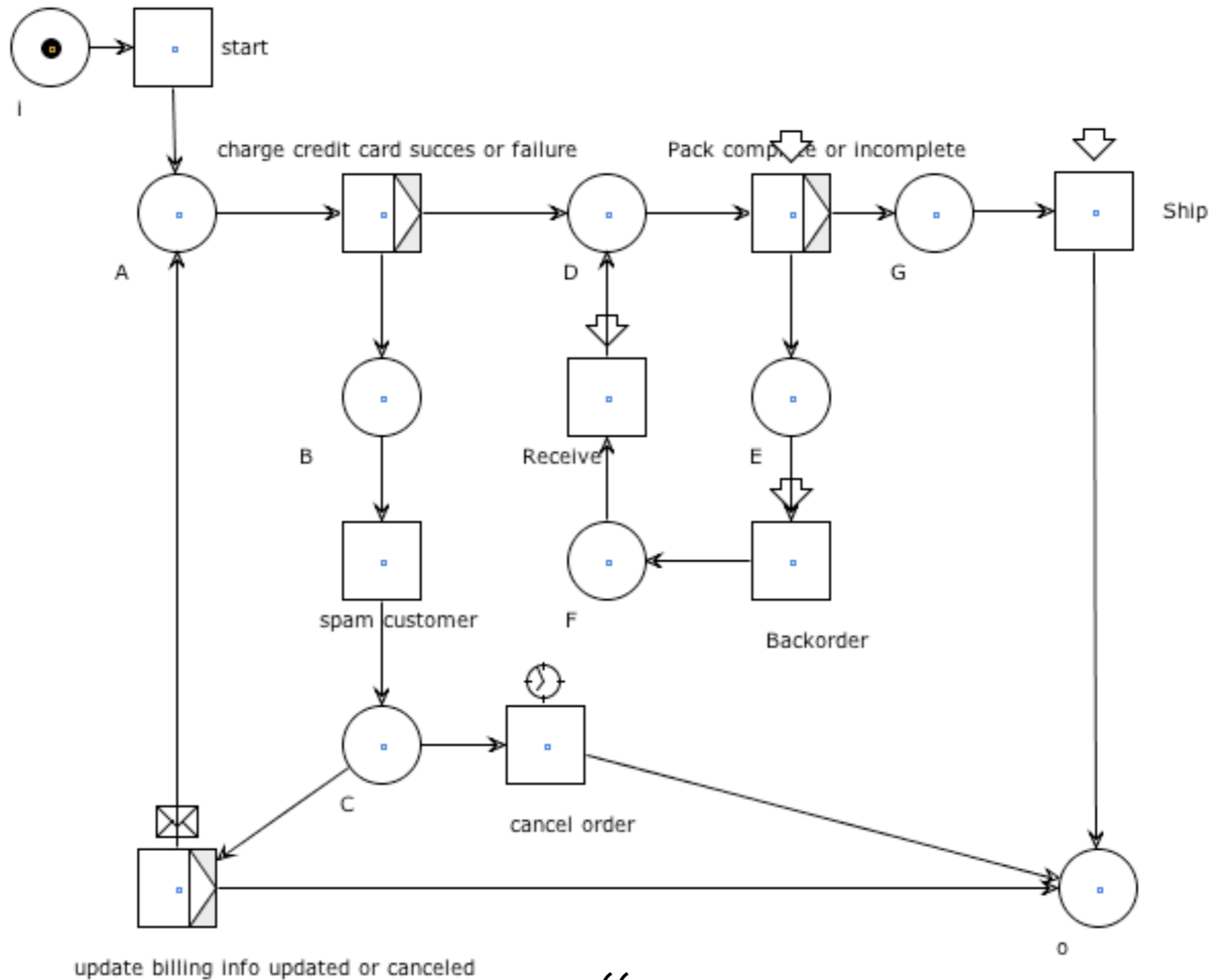
Analyse this net



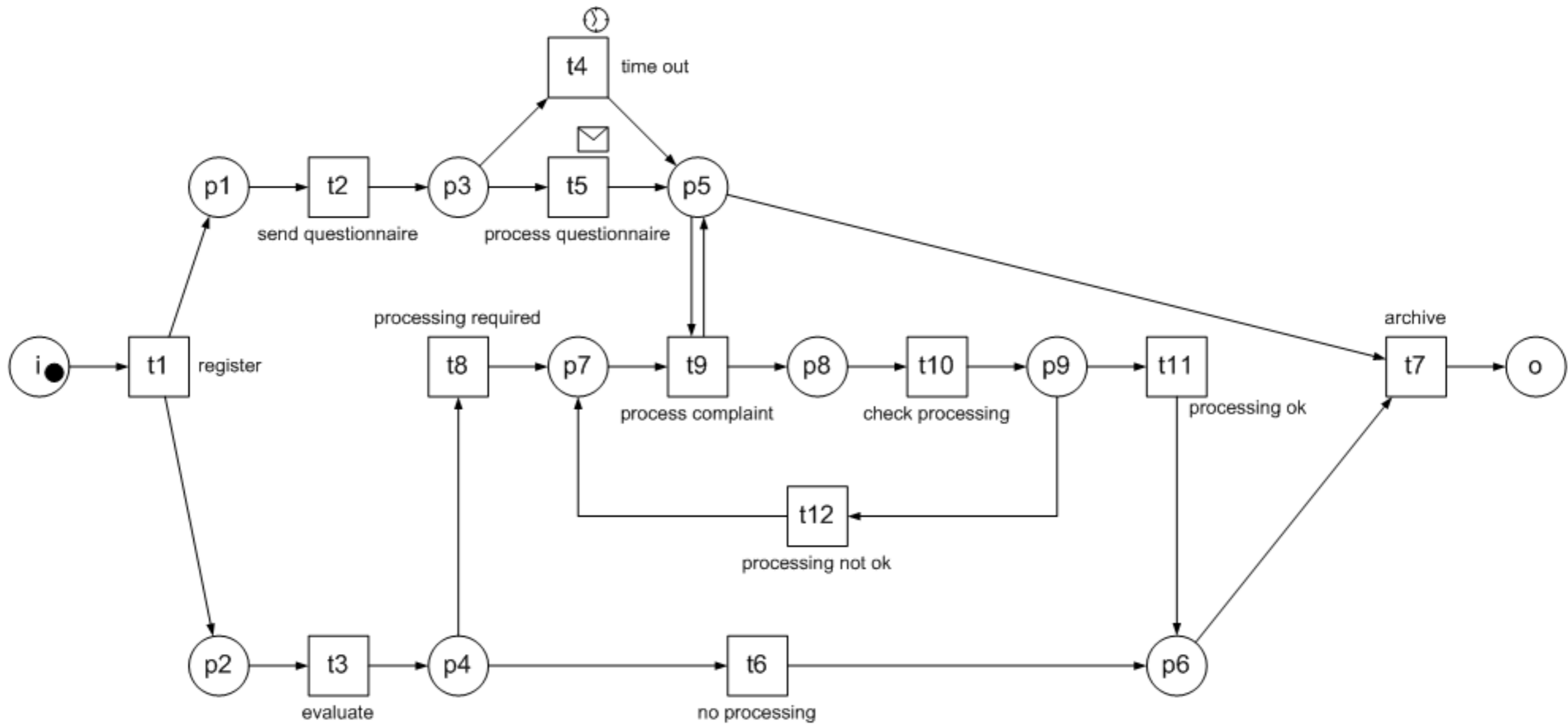
Analyse this net



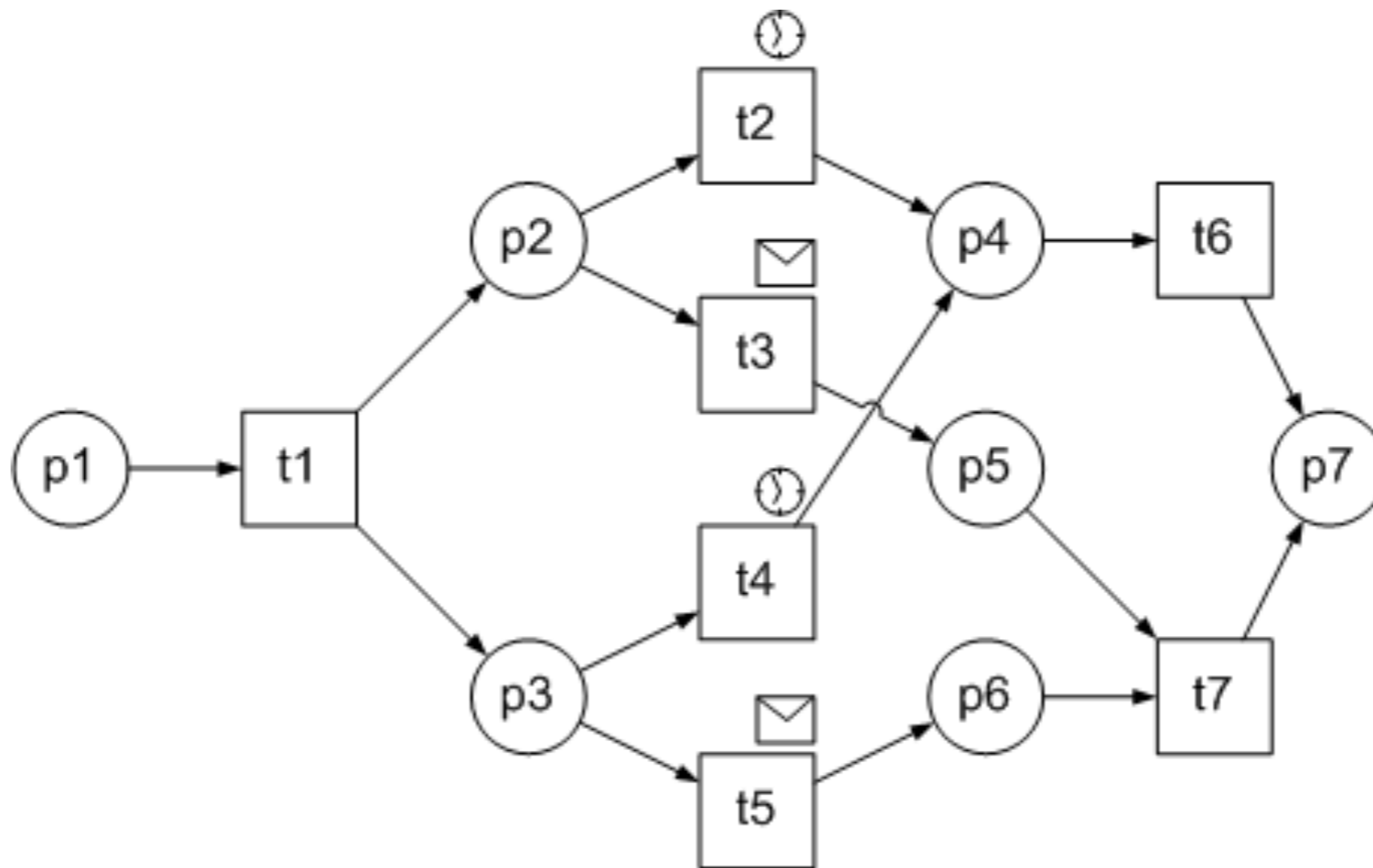
Analyse this net



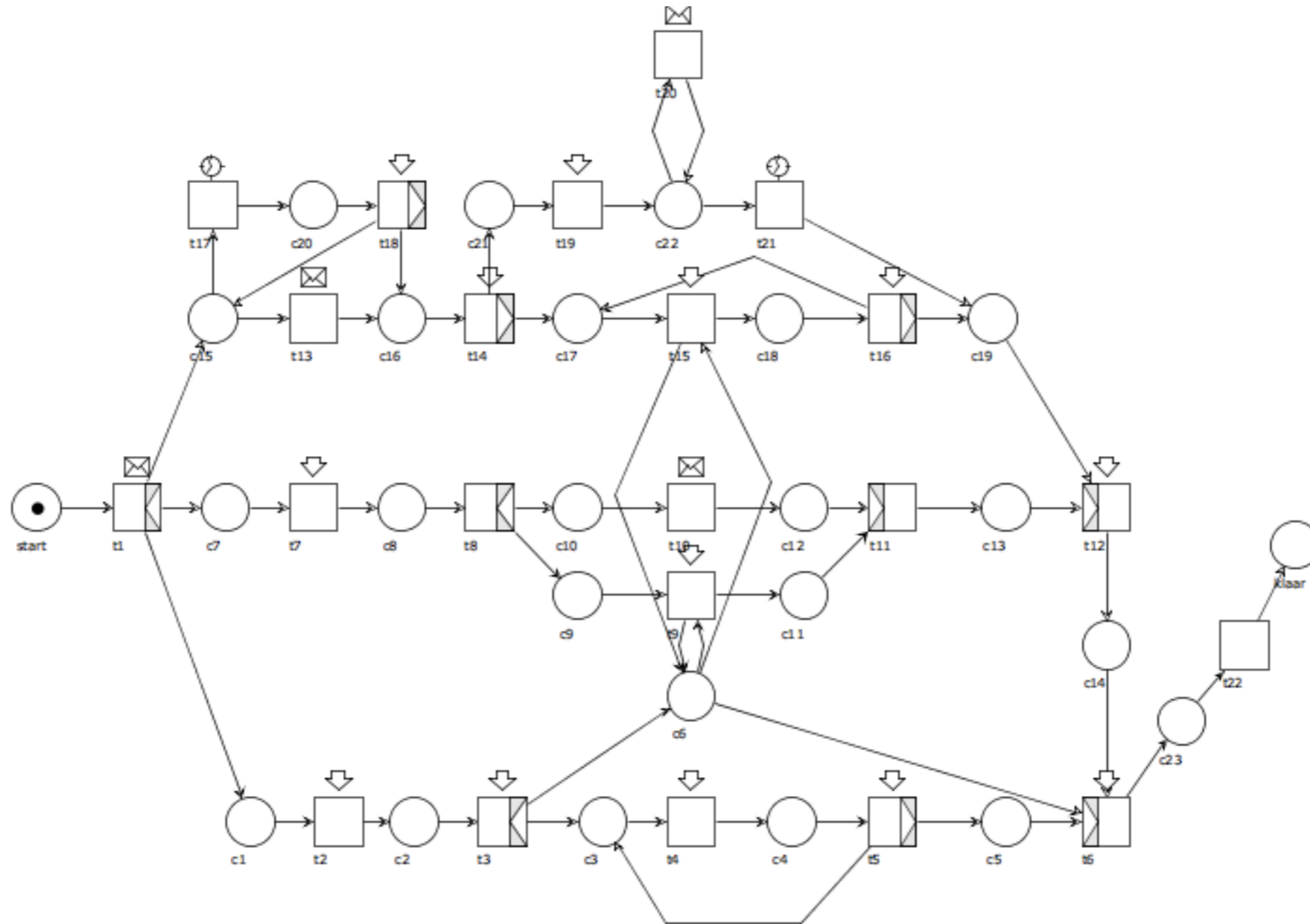
Analyse this net



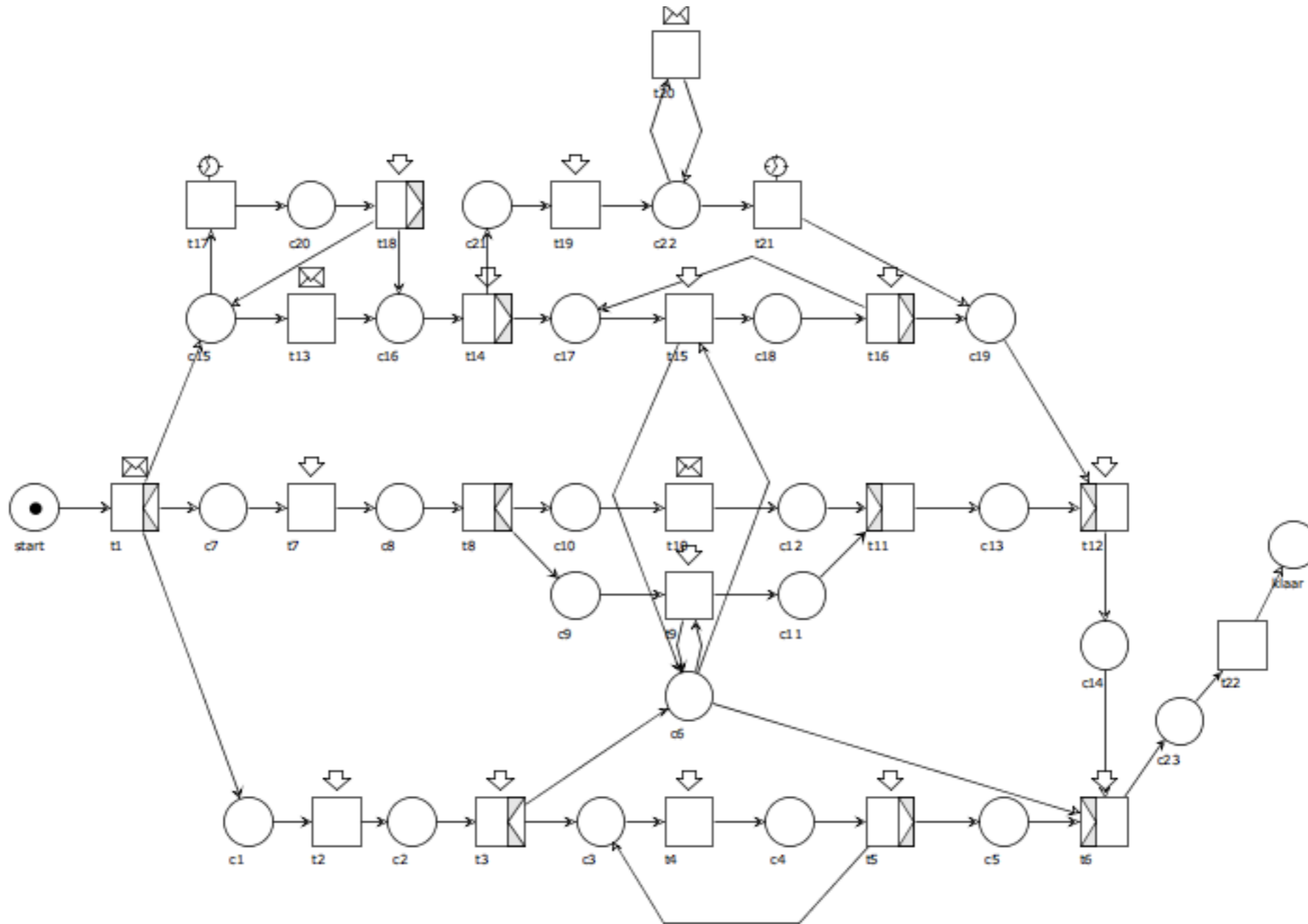
Analyse this net



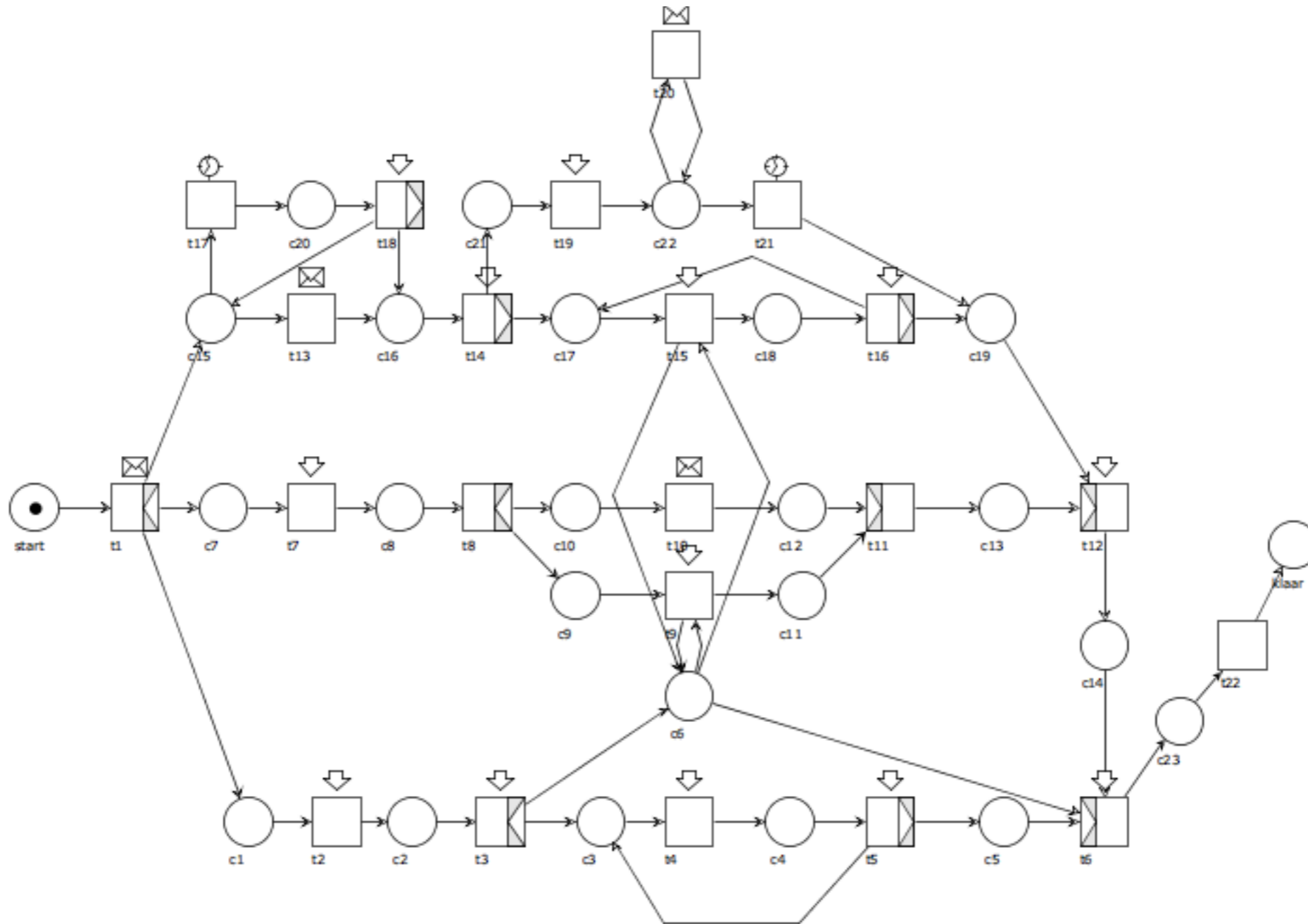
Is this net free-choice?



Is this net S-coverable?



Is this net sound?



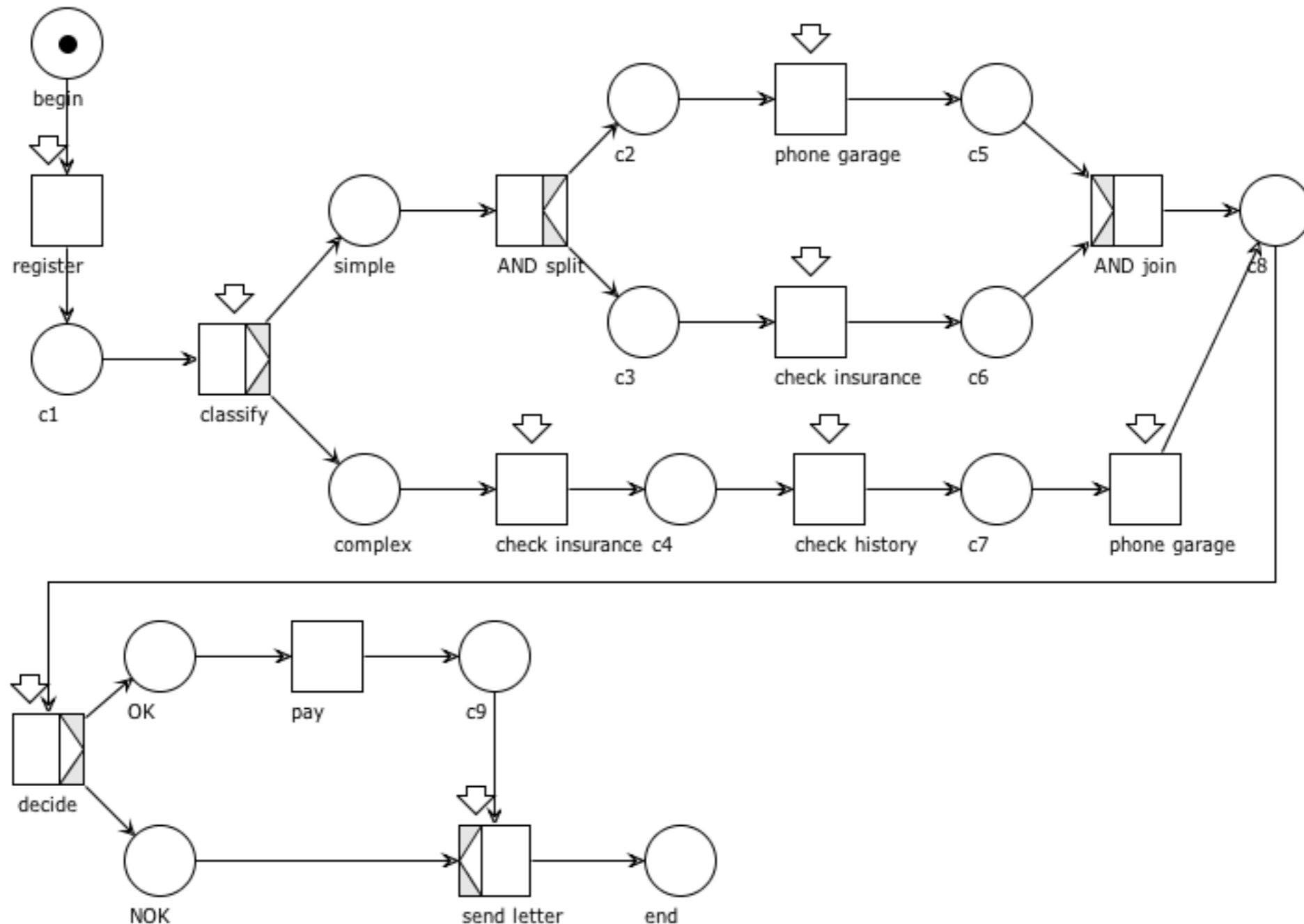
Design Example: Car Damage

- 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 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.