Business Processes Modelling

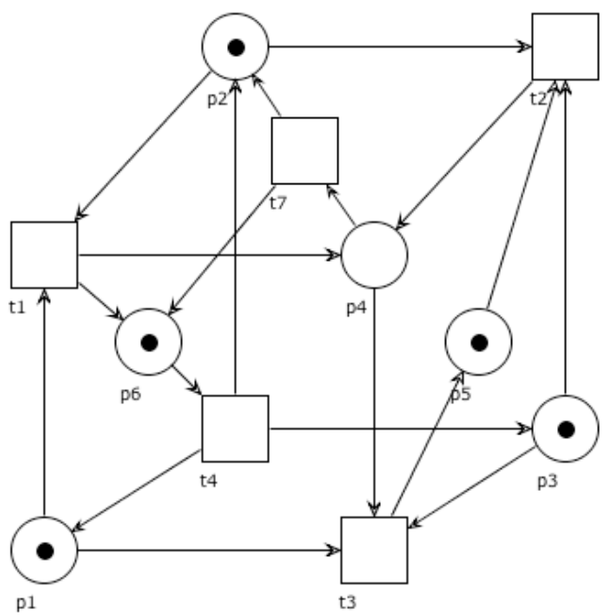
MPB (6 cfu, 295AA)

Roberto Bruni

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

09 - Occurrence graphs

Object



Formalization of basic concepts of Petri nets

Free Choice Nets (book, optional reading)
https://www7.in.tum.de/~esparza/bookfc.html

Petri nets: occurrence graph

Occurrence graph (aka Reachability graph)

The reachability graph is a graph that represents all possible occurrence sequences of a net

Nodes of the graphs = reachable markings
Arcs of the graphs = firings

Formally, $OG(N) = ([M_0\rangle, A)$ where $A \subseteq [M_0\rangle \times T \times [M_0\rangle$ s.t.

$$(M, t, M') \in A \quad \text{iff} \quad M \xrightarrow{t} M'$$

How to compute OG(N)

Adding all exiting arcs each time:

markings to explore

1. $Nodes = \{\},\ Arcs = \{\},\ Todo = \{M_0\}$

2. $M = next(Todo)$ select one marking to explore

3. $Nodes = Nodes \cup \{M\},\ Todo = Todo \setminus \{M\}$ update nodes

collect all firings from M

4. $Firings = \{(M, t, M') \mid \exists t \in T, \exists M' \in \mu(P),\ M \xrightarrow{t} M'\}$

find new markings to explore

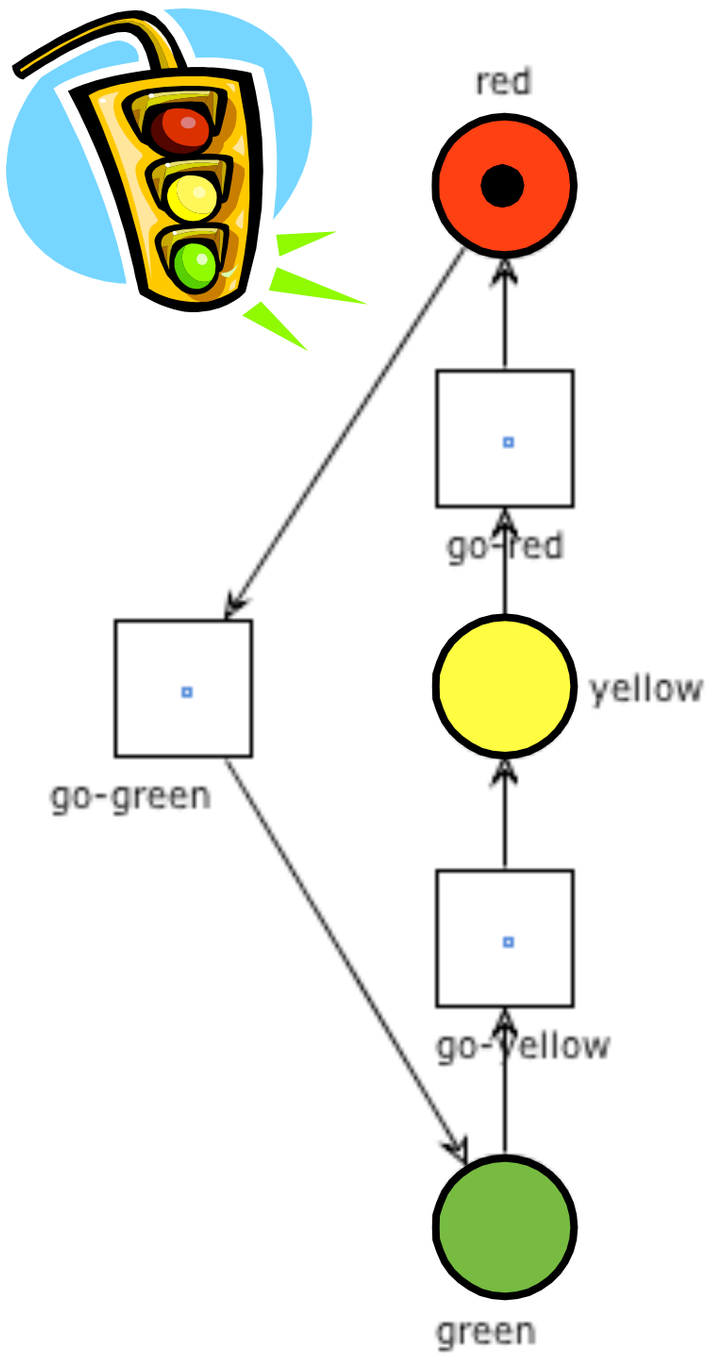
5. $New = \{M' \mid (M, t, M') \in Firings\} \setminus (Nodes \cup Todo)$

6. $Todo = Todo \cup New,\ Arcs = Arcs \cup Firings$ update nodes and arcs

7. $isEmpty(Todo)\ ?\ \mathbf{stop}\ :\ \mathbf{goto}\ 2$ repeat if there are still markings to be explored

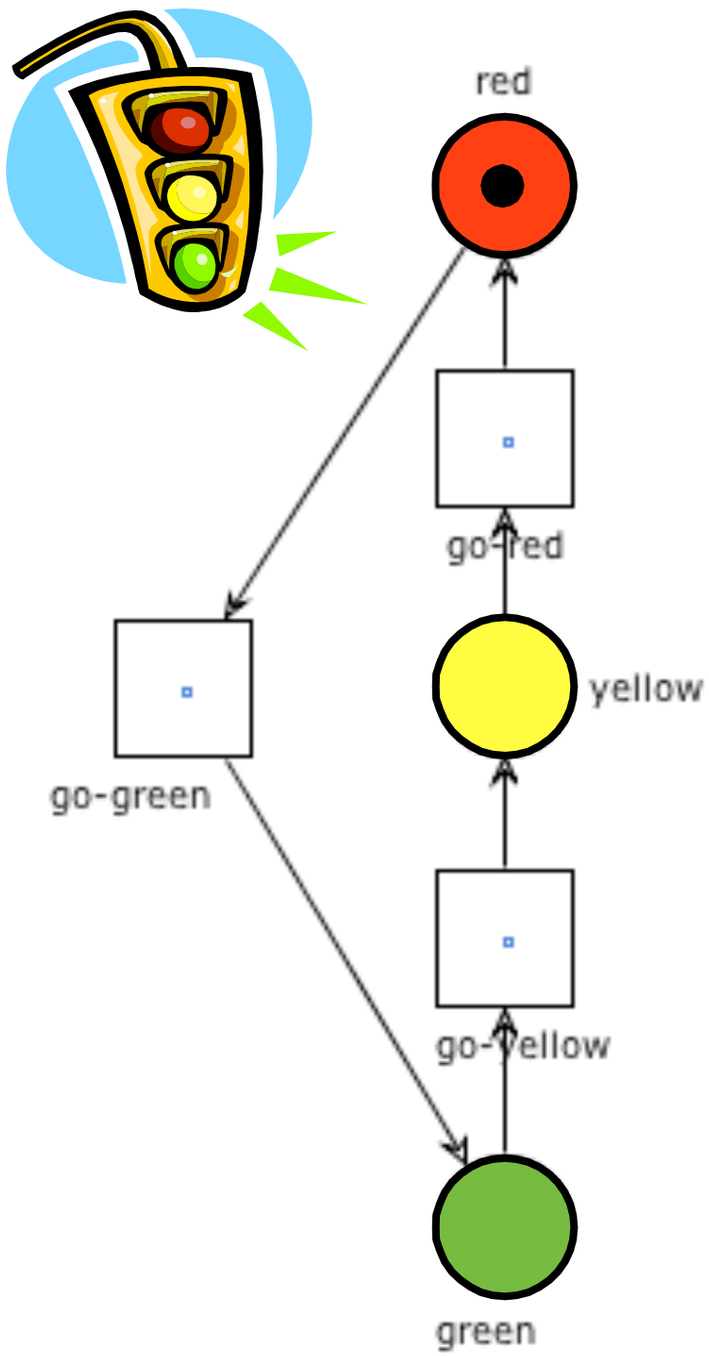
Example: traffic light

Todo = { red }



Example: traffic light

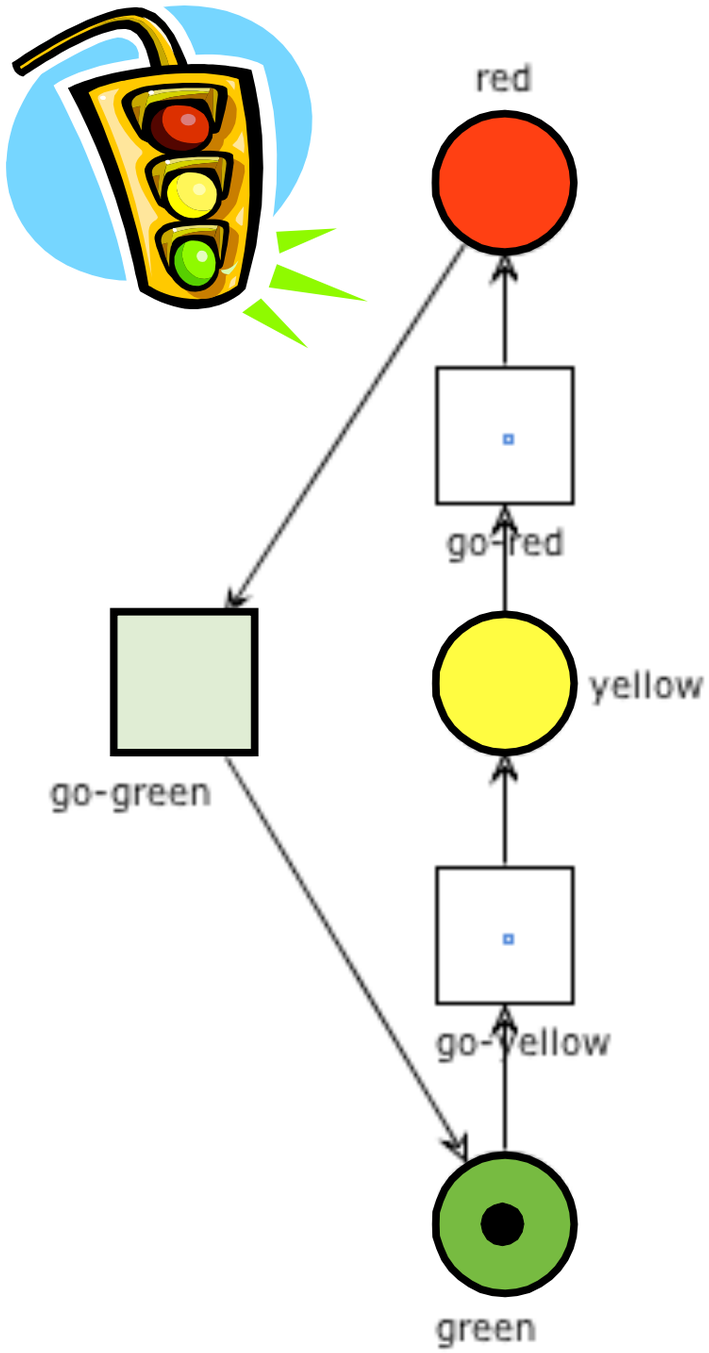
Todo = { }

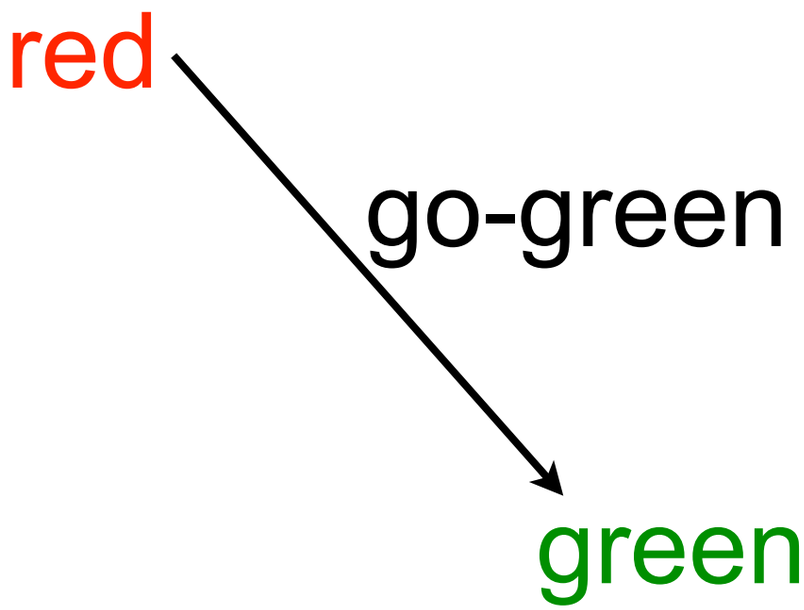


red

Example: traffic light

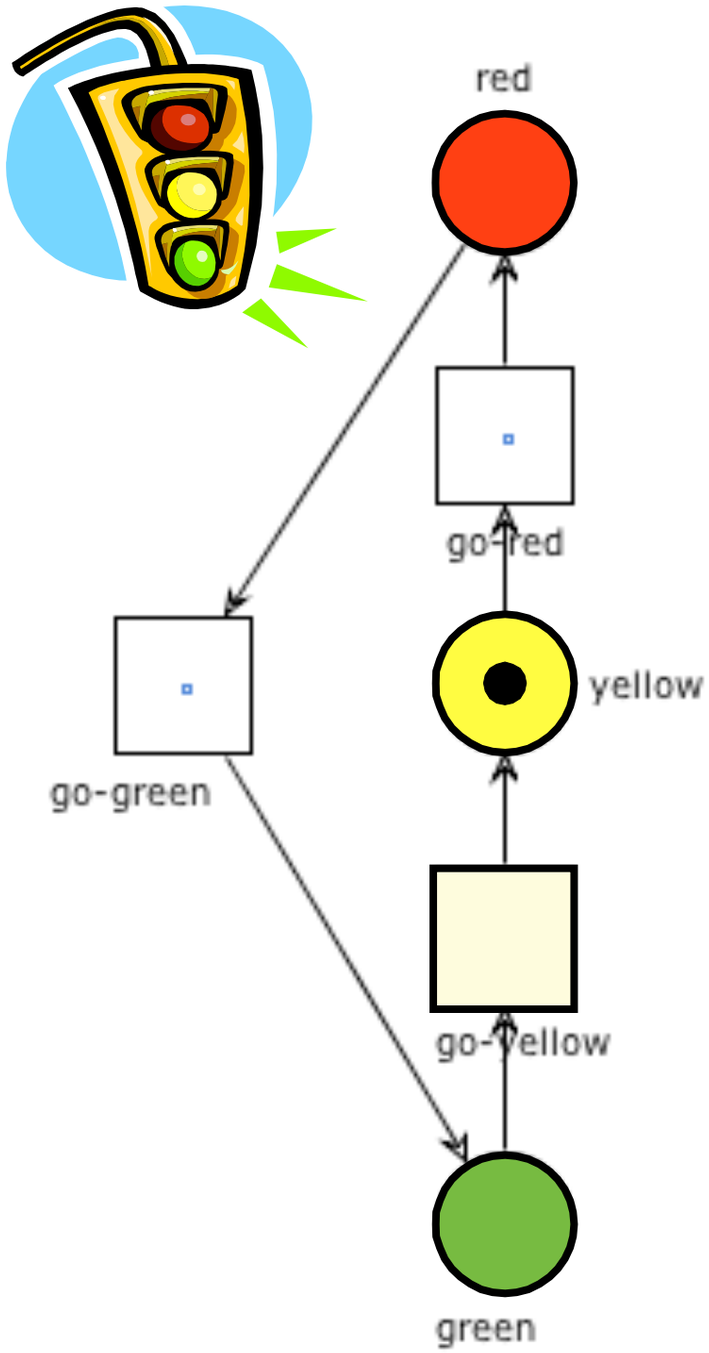
Todo = { green }

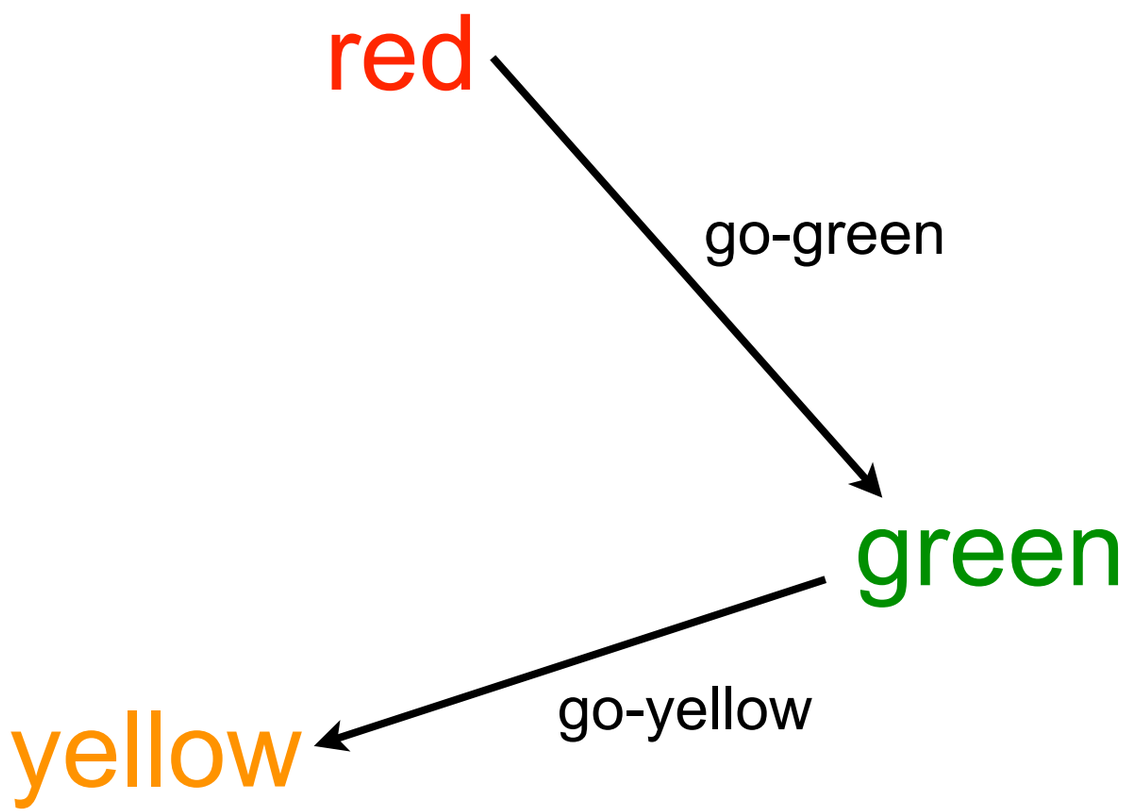




Example: traffic light

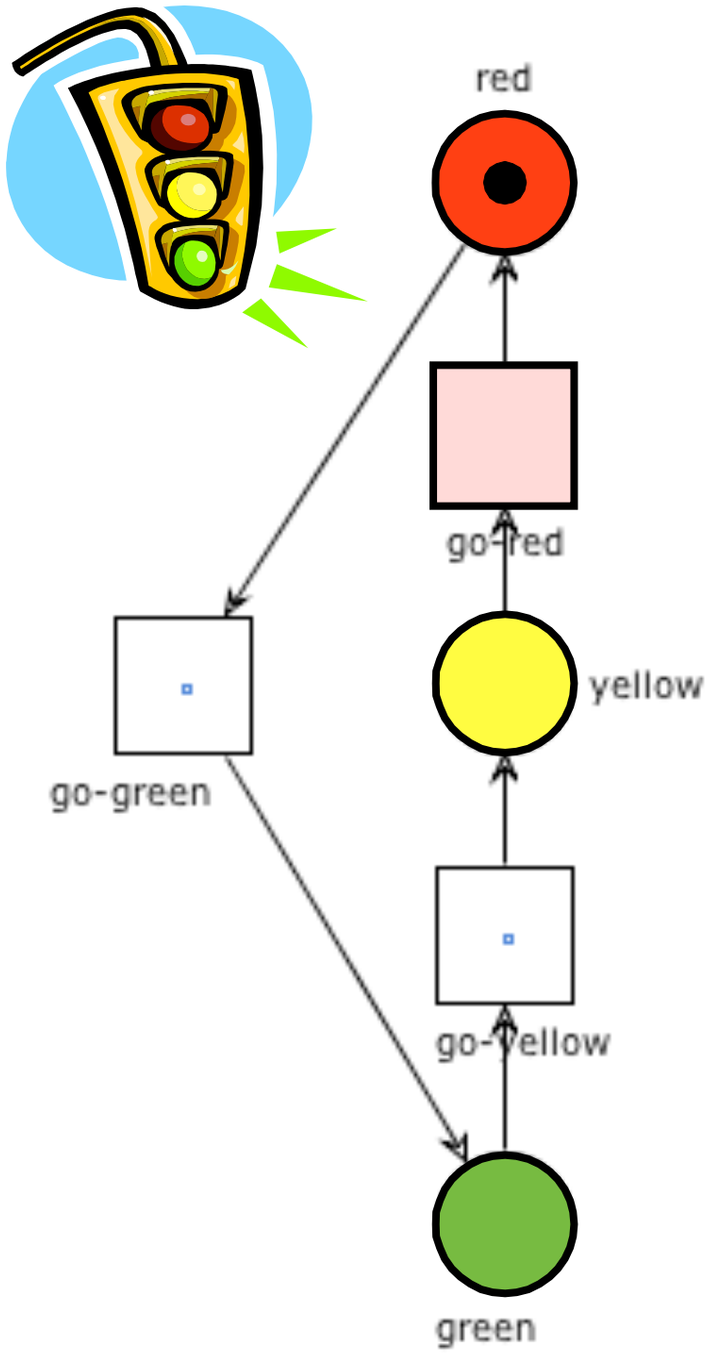
Todo = { yellow }

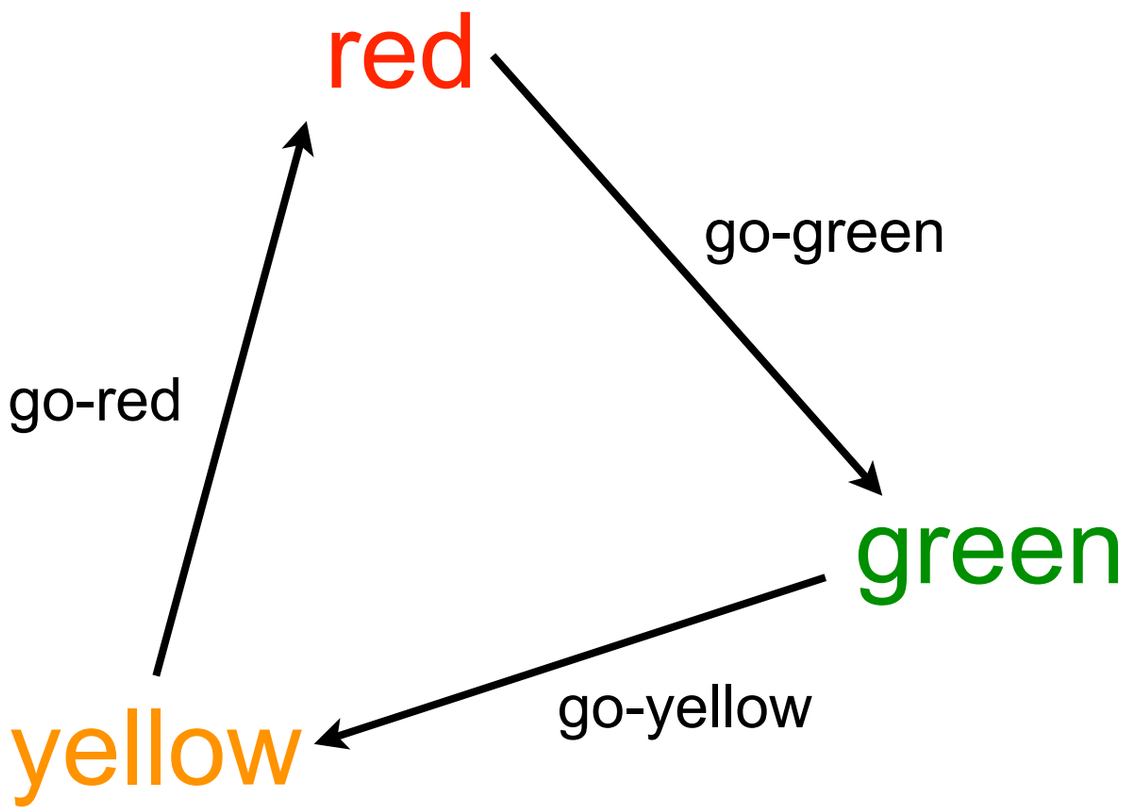




Example: traffic light

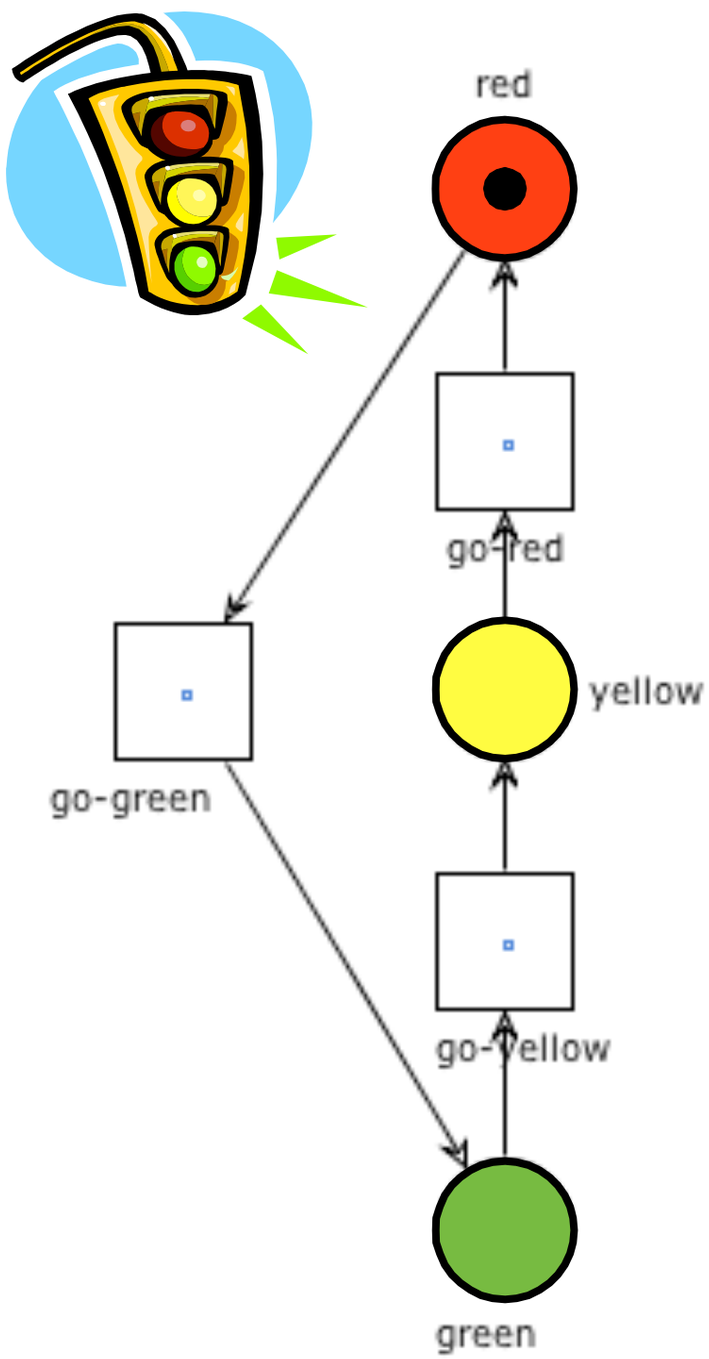
Todo = { }

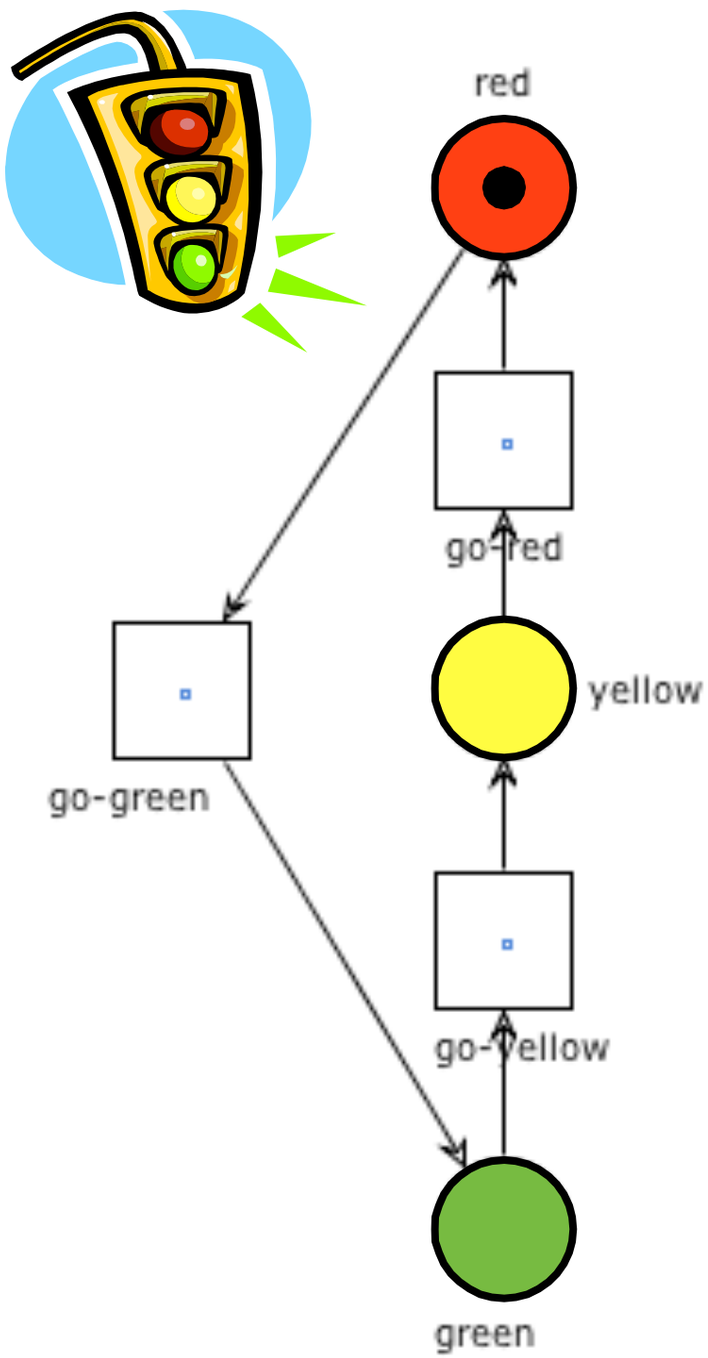




Example: two traffic lights

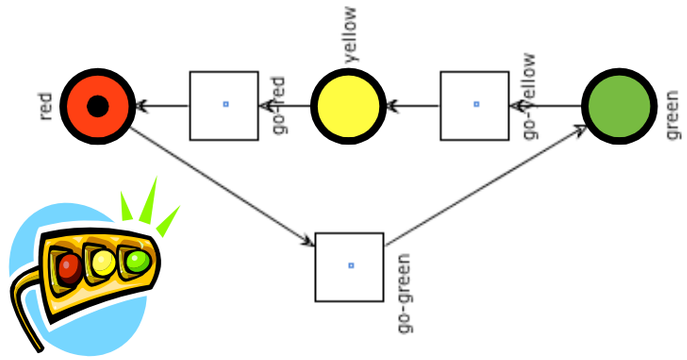
Todo = { red + red' }

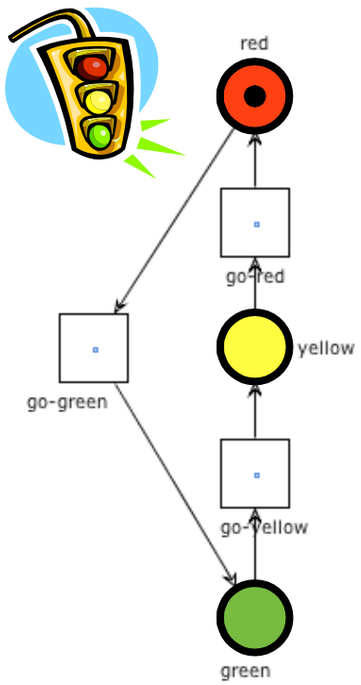




Example: two traffic lights

(we omit arc labels
for readability issues)



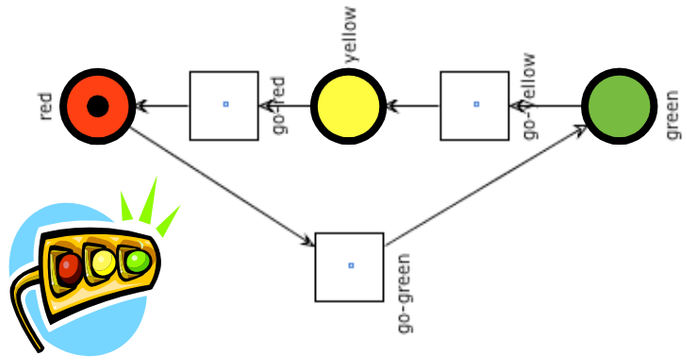


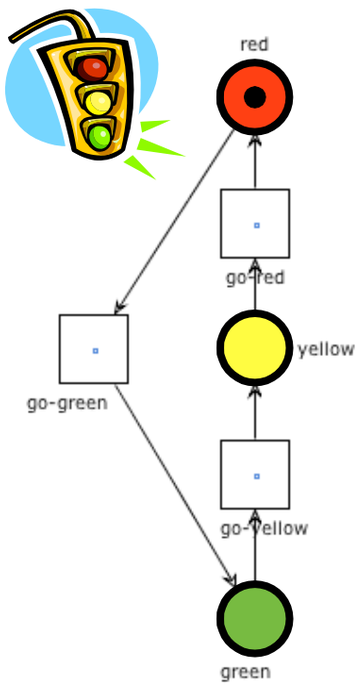
red + red’

Todo = { }

Example: two traffic lights

(we omit arc labels
for readability issues)





red + red’

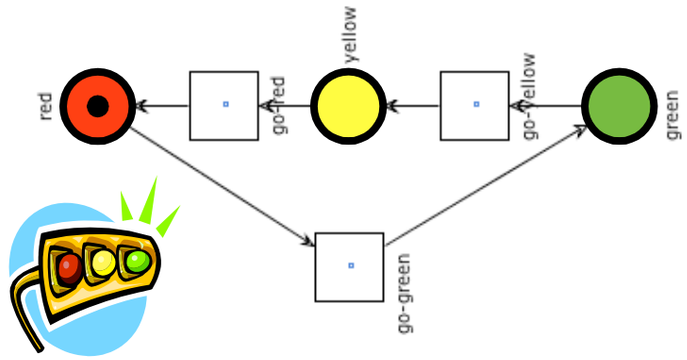
red + green’

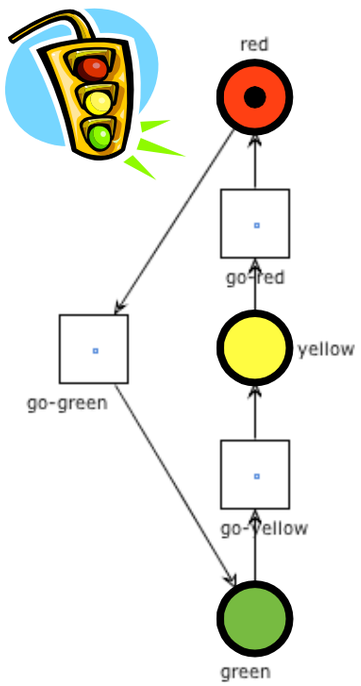
green + red’

Todo = { green+red’ , red+green' }

Example: two traffic lights

(we omit arc labels
for readability issues)





red + red’

red + green’

yellow + red’

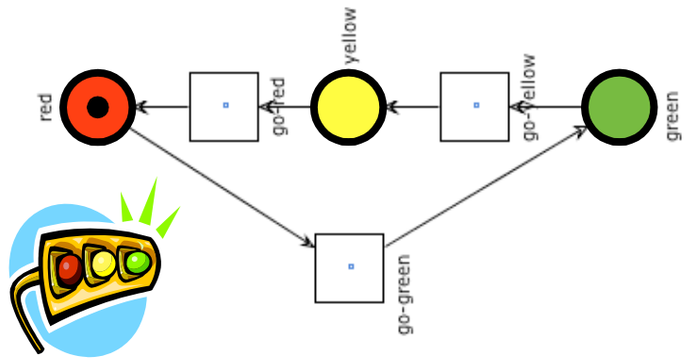
green + red’

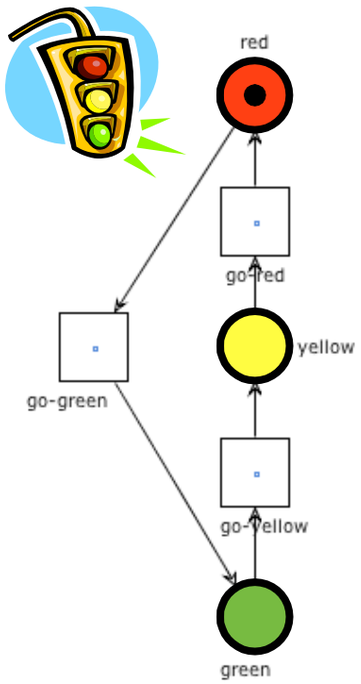
green + green’

Todo = { yellow+red’ , green+green' , red+green' }

Example: two traffic lights

(we omit arc labels
for readability issues)





red + red’

red + green’

yellow + red’

yellow + green’

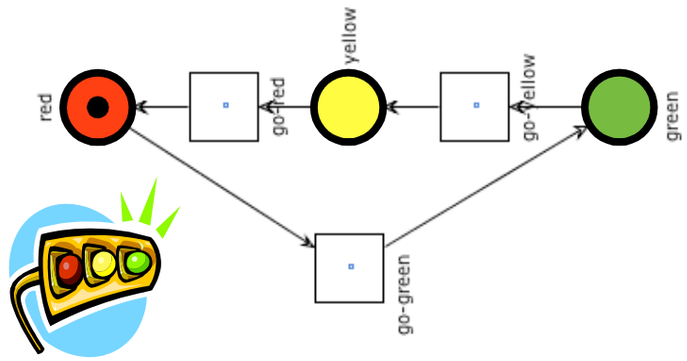
green + red’

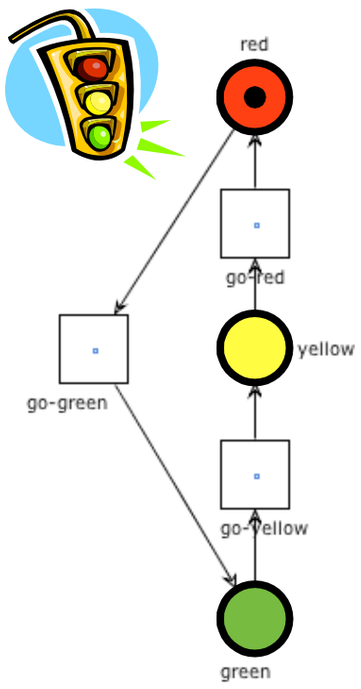
green + green’

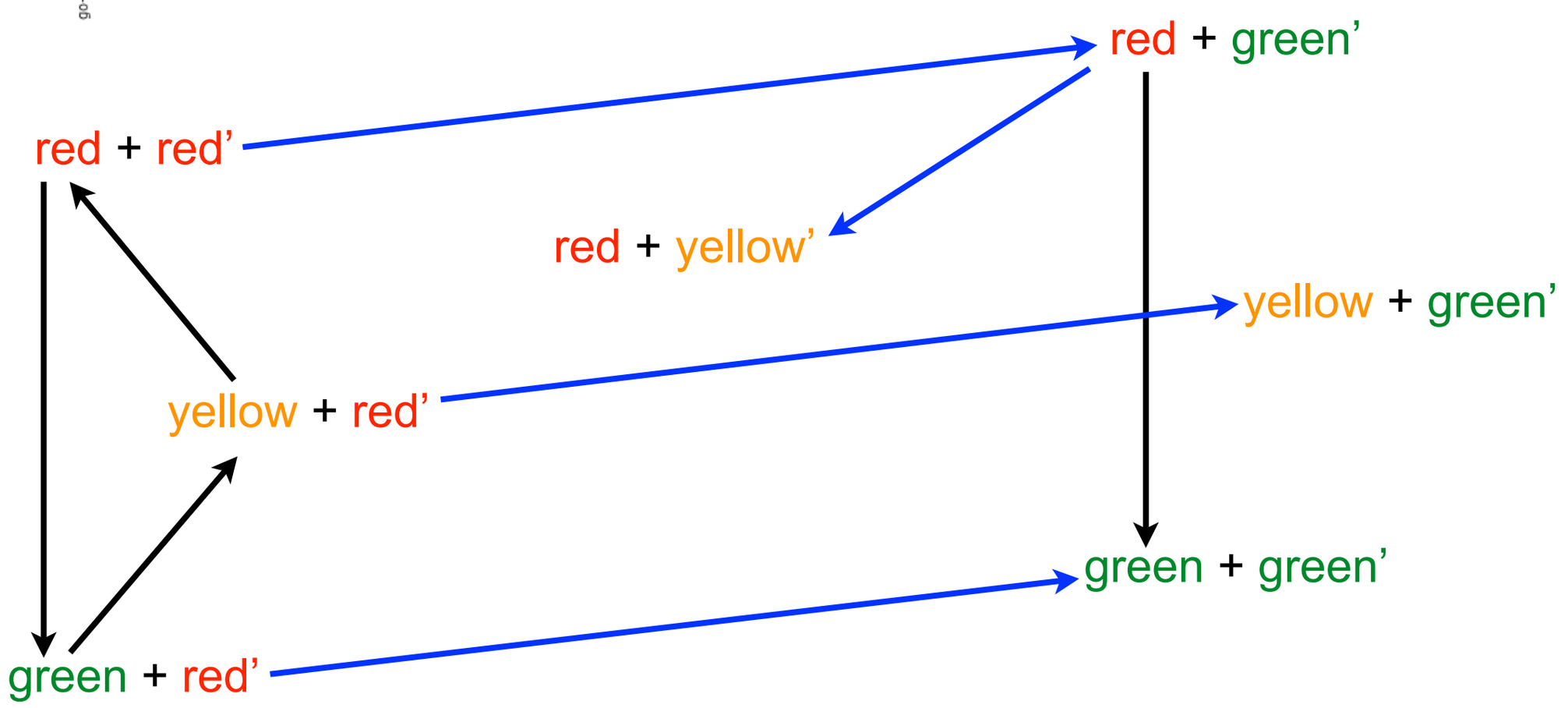
Todo = { yellow+green' , green+green' , red+green' }

Example: two traffic lights

(we omit arc labels
for readability issues)



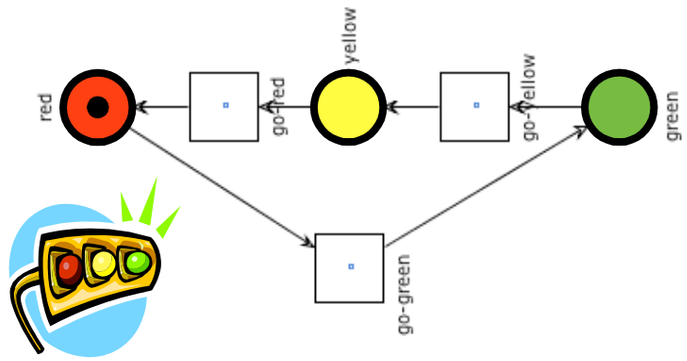


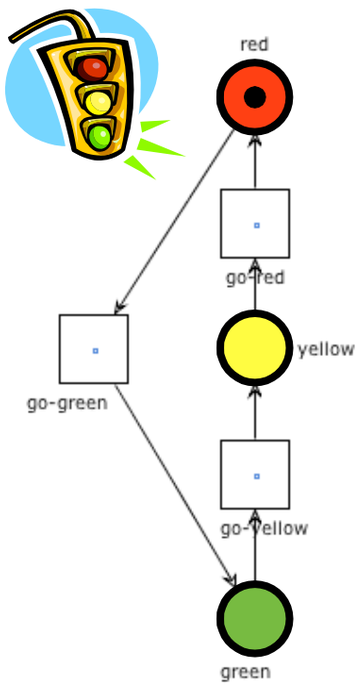


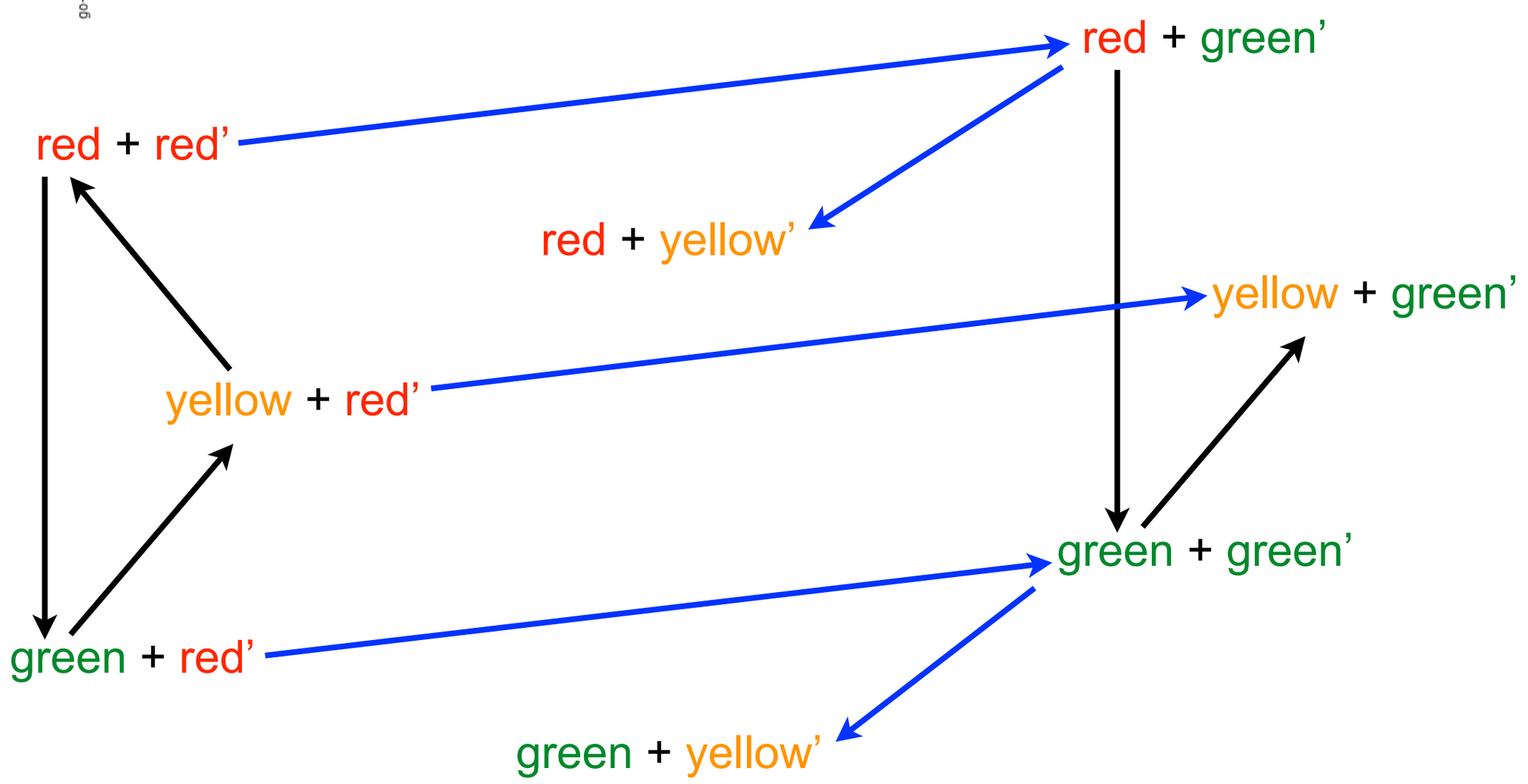
Todo = { yellow+green' , green+green' , red+yellow' }

Example: two traffic lights

(we omit arc labels
for readability issues)



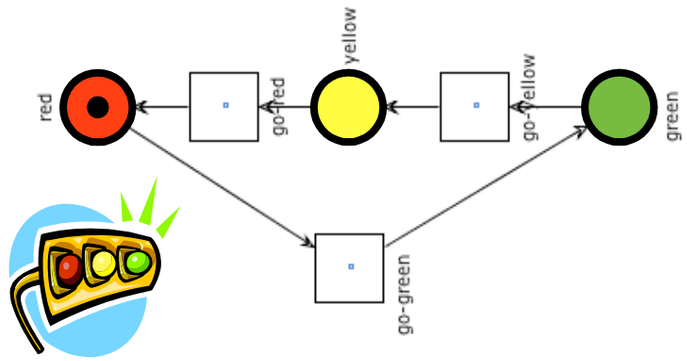


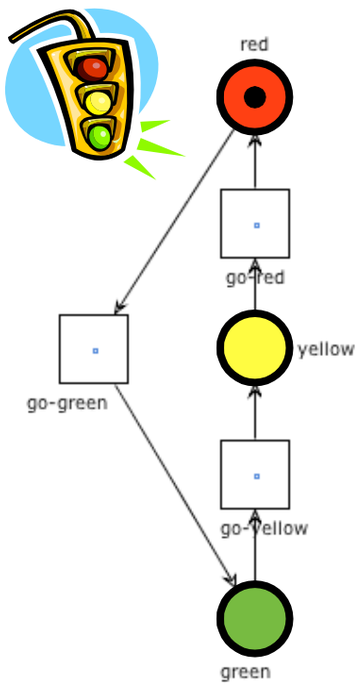


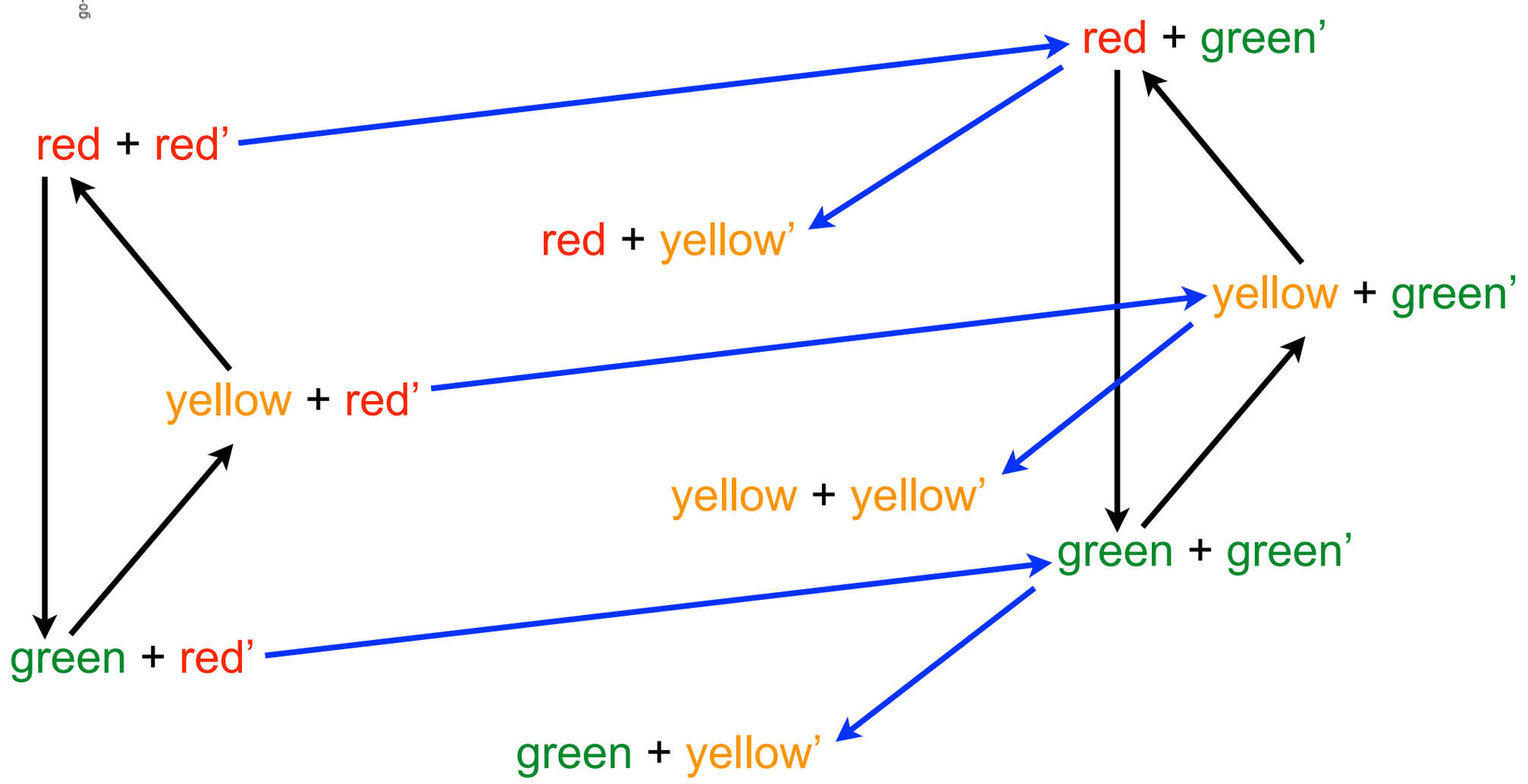
Todo = { yellow+green' , green+yellow' , red+yellow' }

Example: two traffic lights

(we omit arc labels
for readability issues)



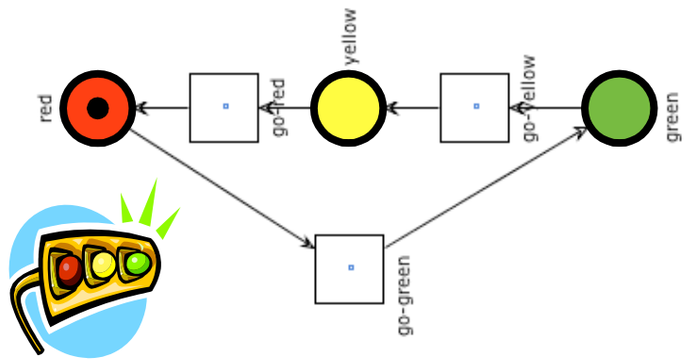


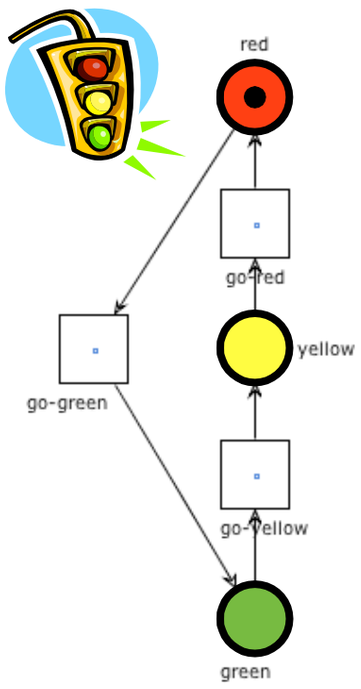


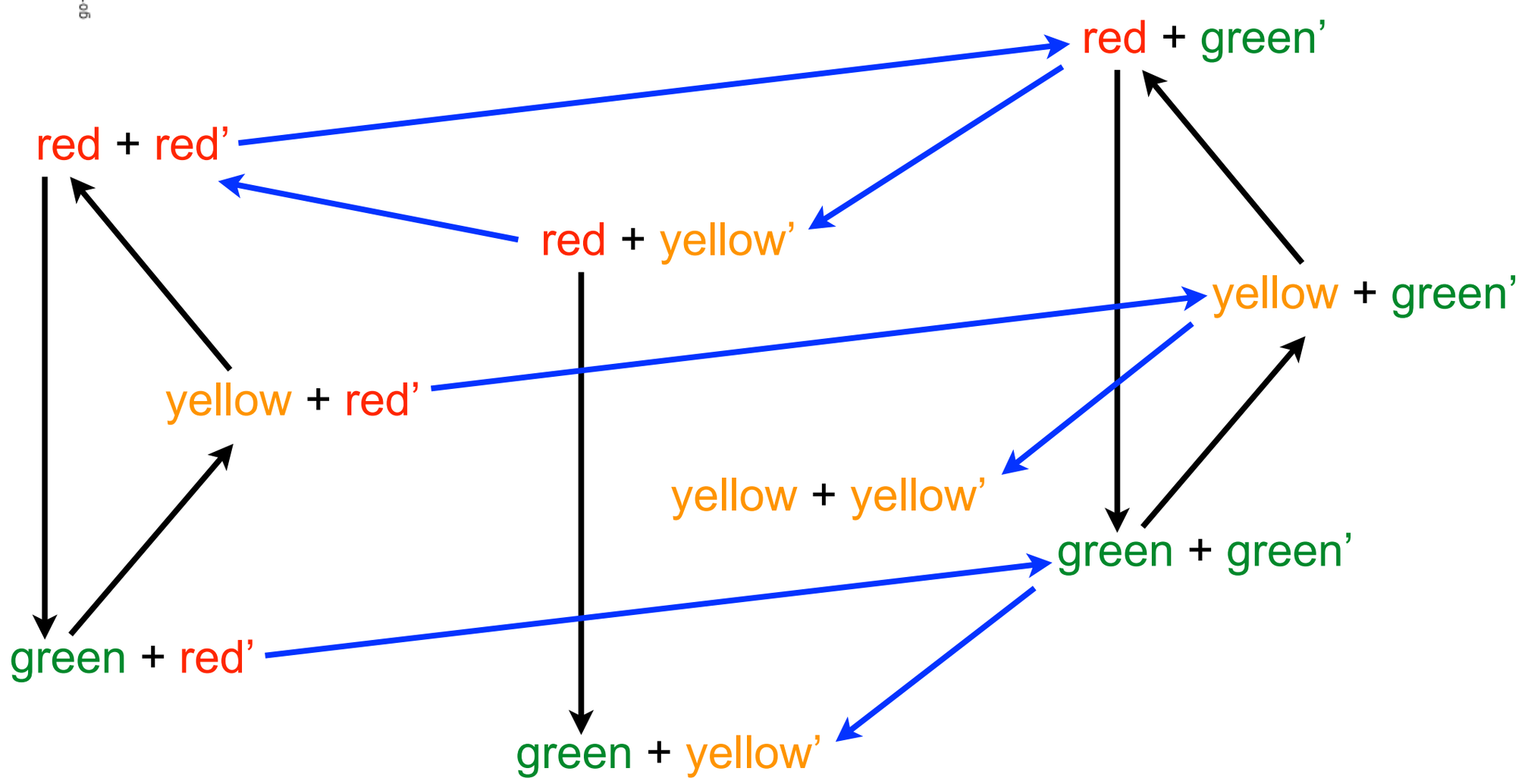
Todo = { yellow+yellow' , green+yellow' , red+yellow' }

Example: two traffic lights

(we omit arc labels
for readability issues)



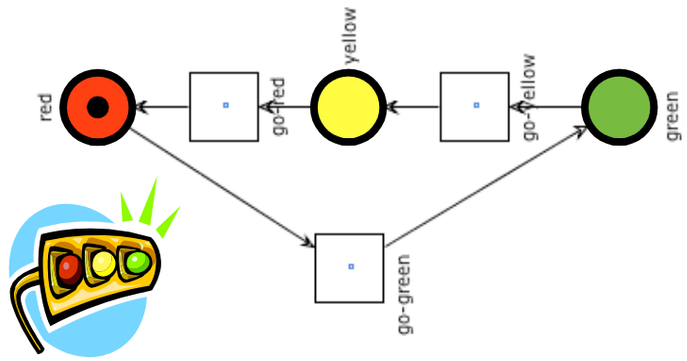


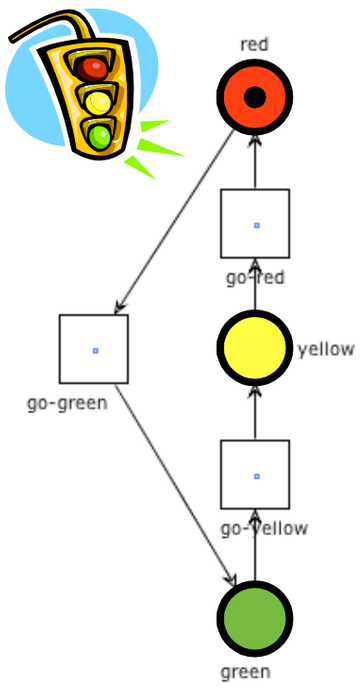


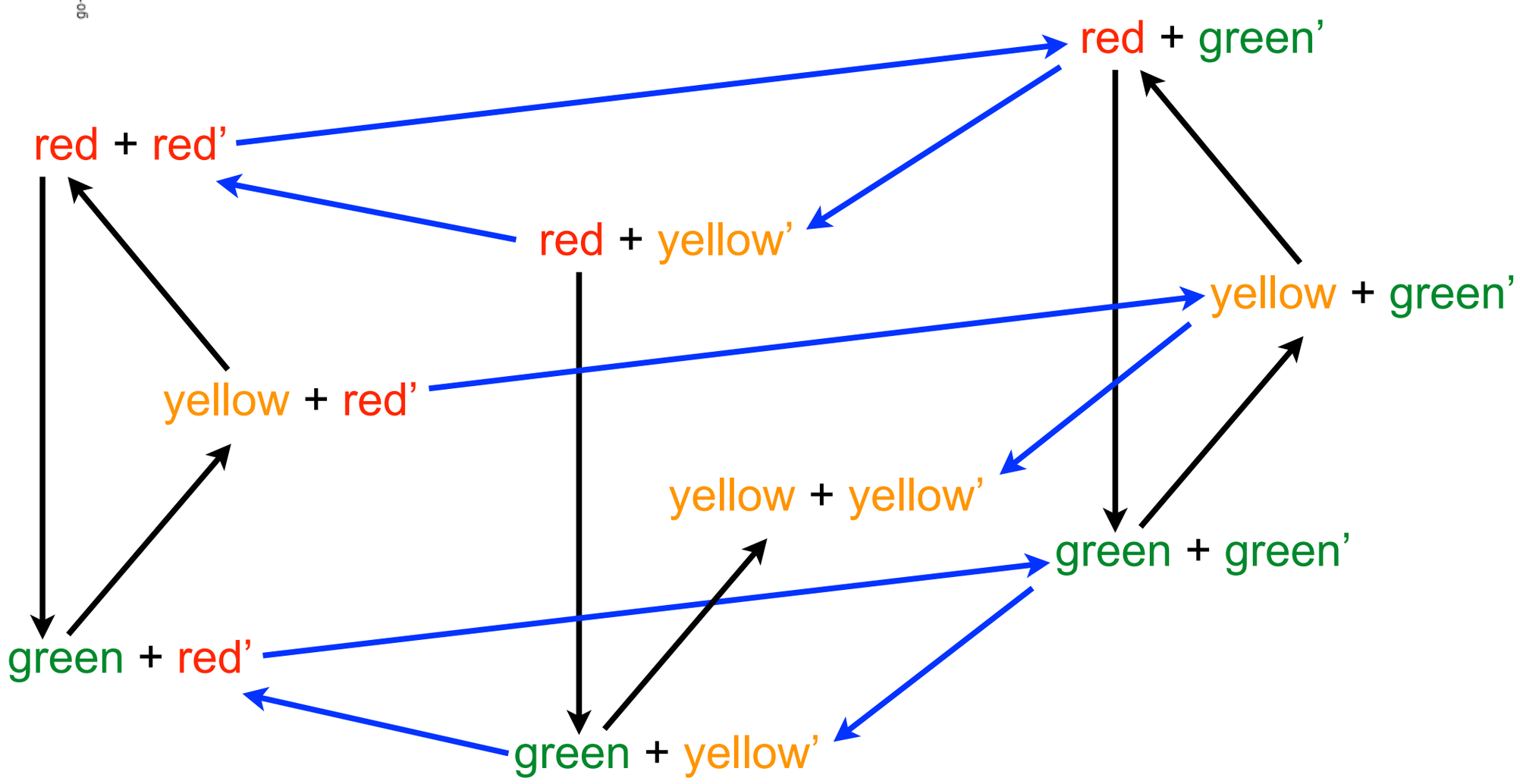
Todo = { yellow+yellow' , green+yellow' }

Example: two traffic lights

(we omit arc labels
for readability issues)



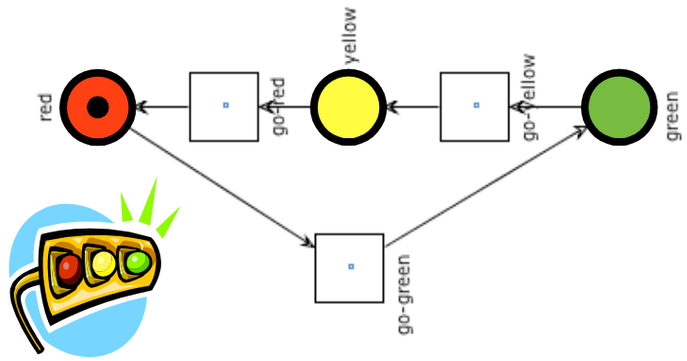


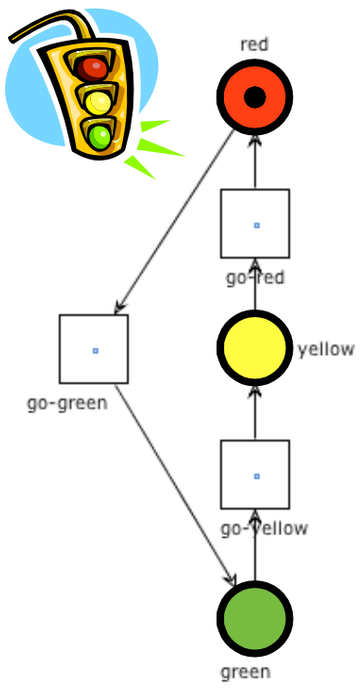


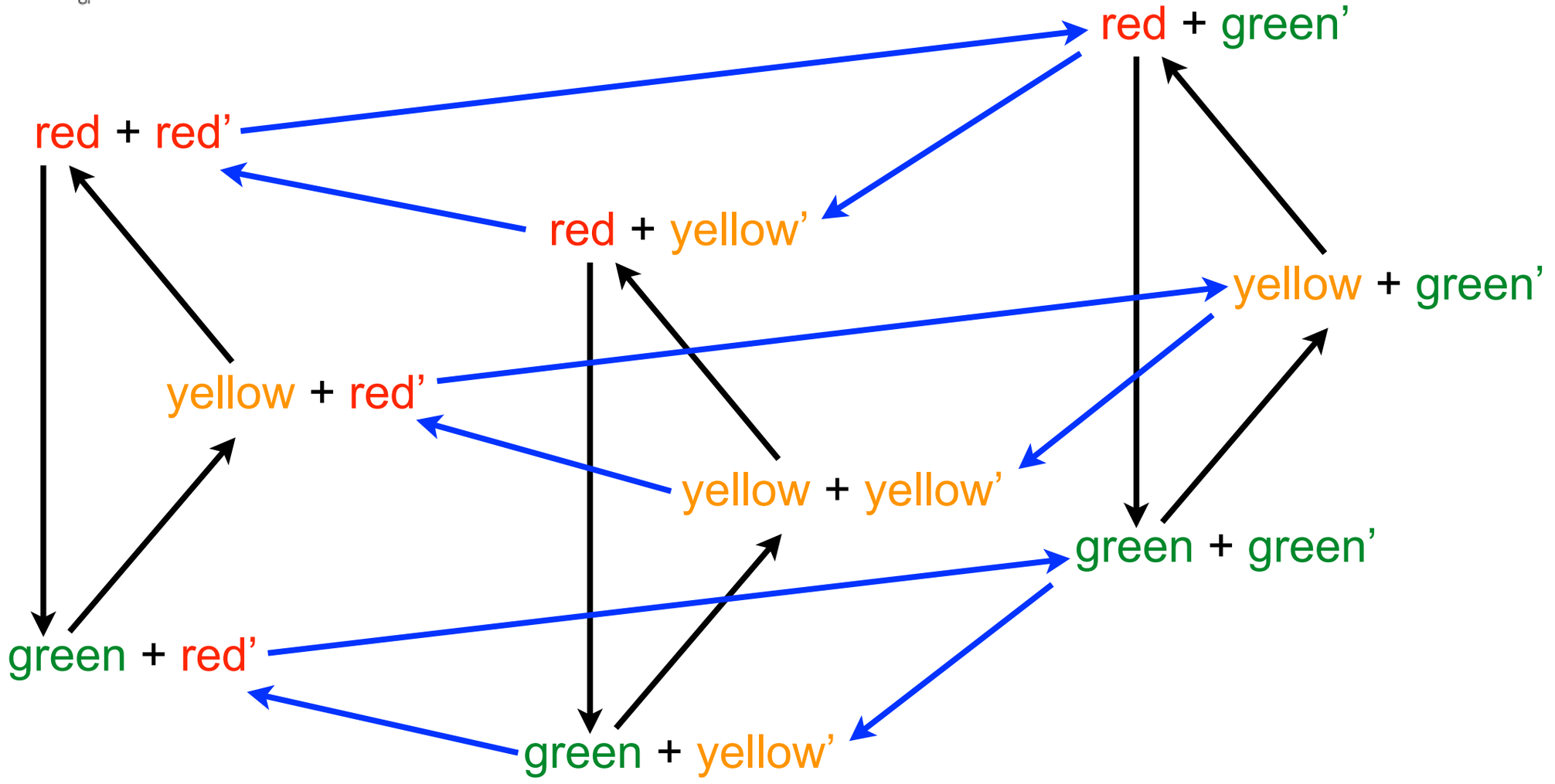
Todo = { yellow+yellow' }

Example: two traffic lights

(we omit arc labels
for readability issues)





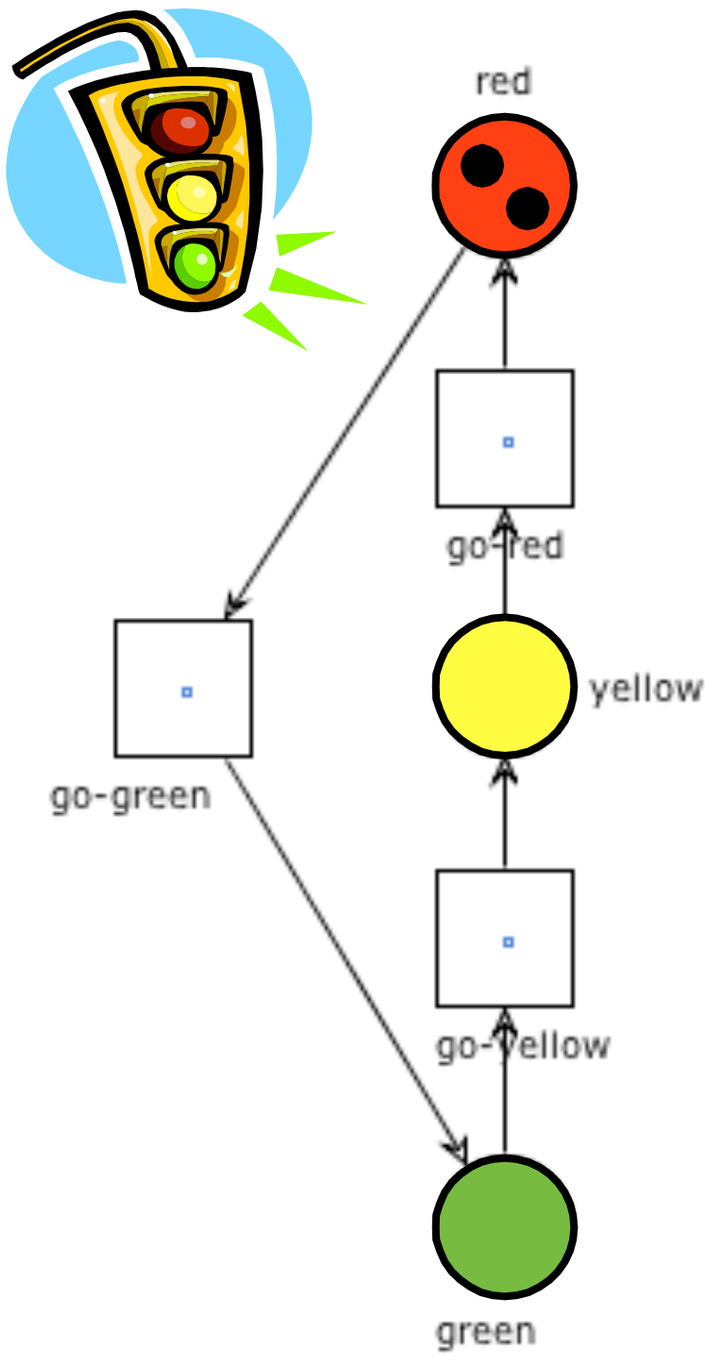


Todo = { }

Example: two traffic lights

(we omit arc labels
for readability issues)

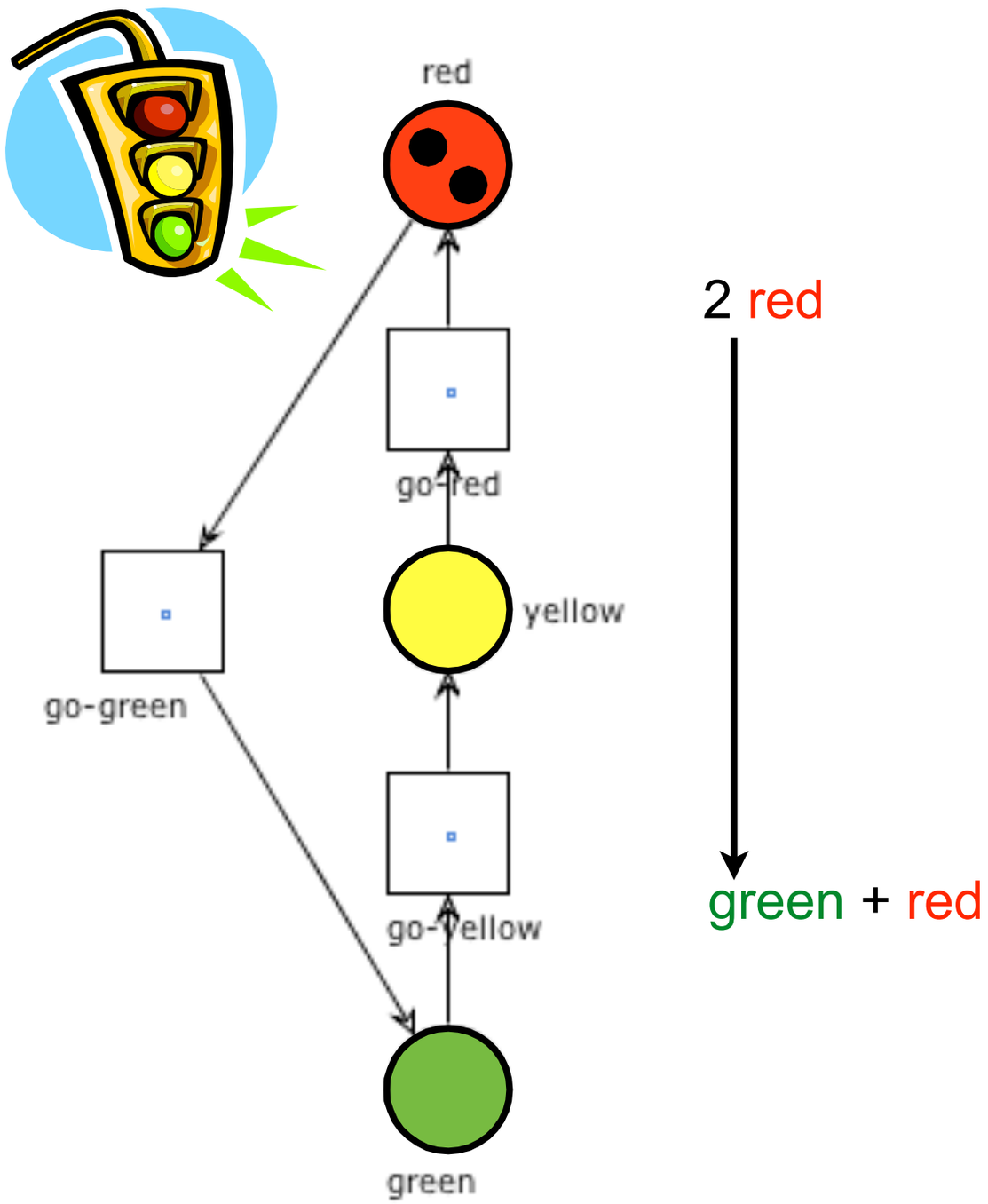
Todo = { 2red }



Example: two traffic lights

(we omit arc labels
for readability issues)

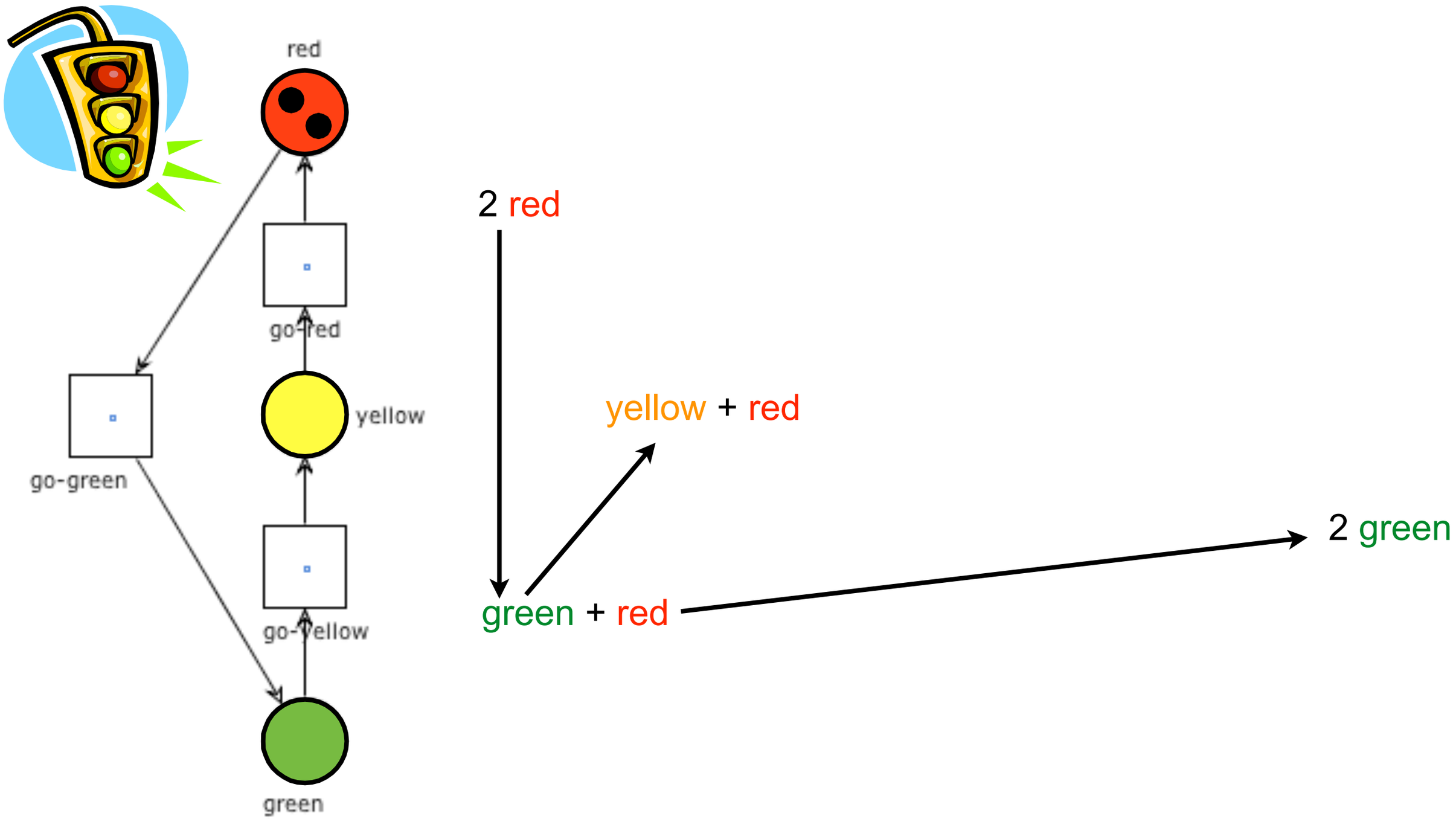
Todo = { green+red }



Example: two traffic lights

(we omit arc labels
for readability issues)

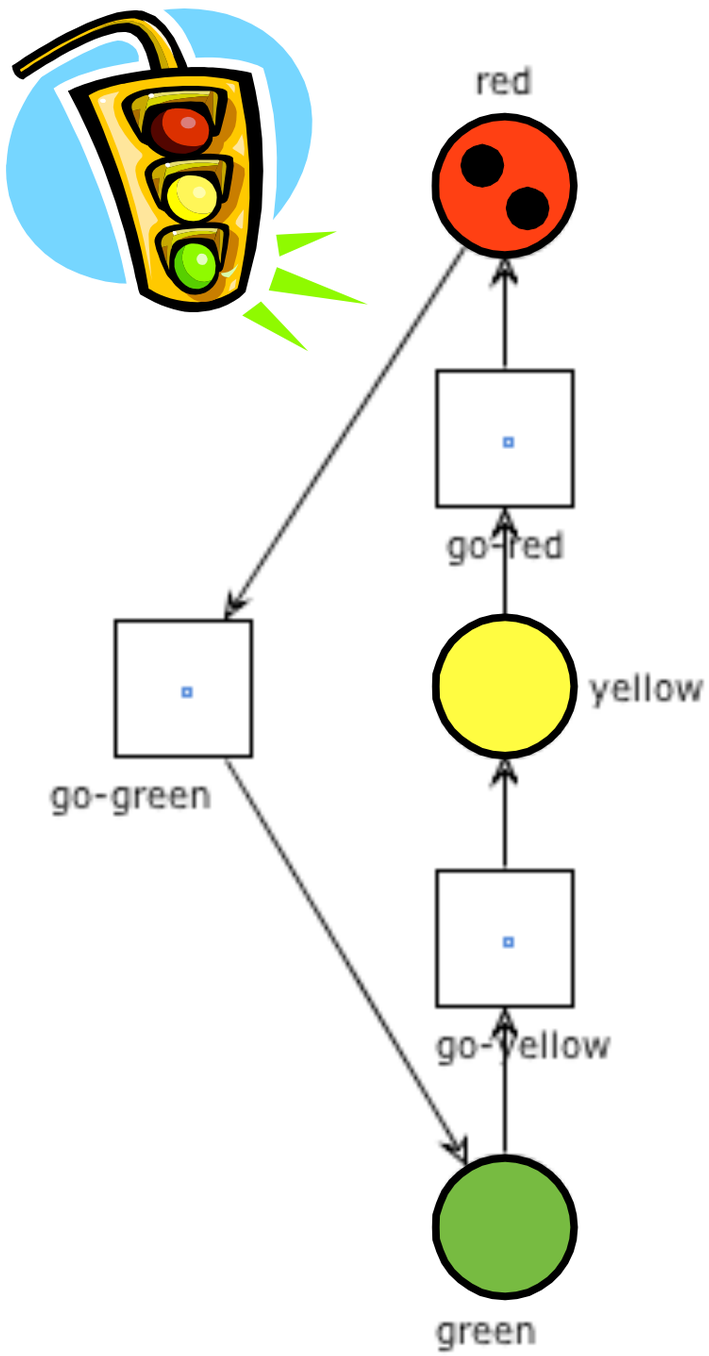
Todo = { 2green , yellow+red }

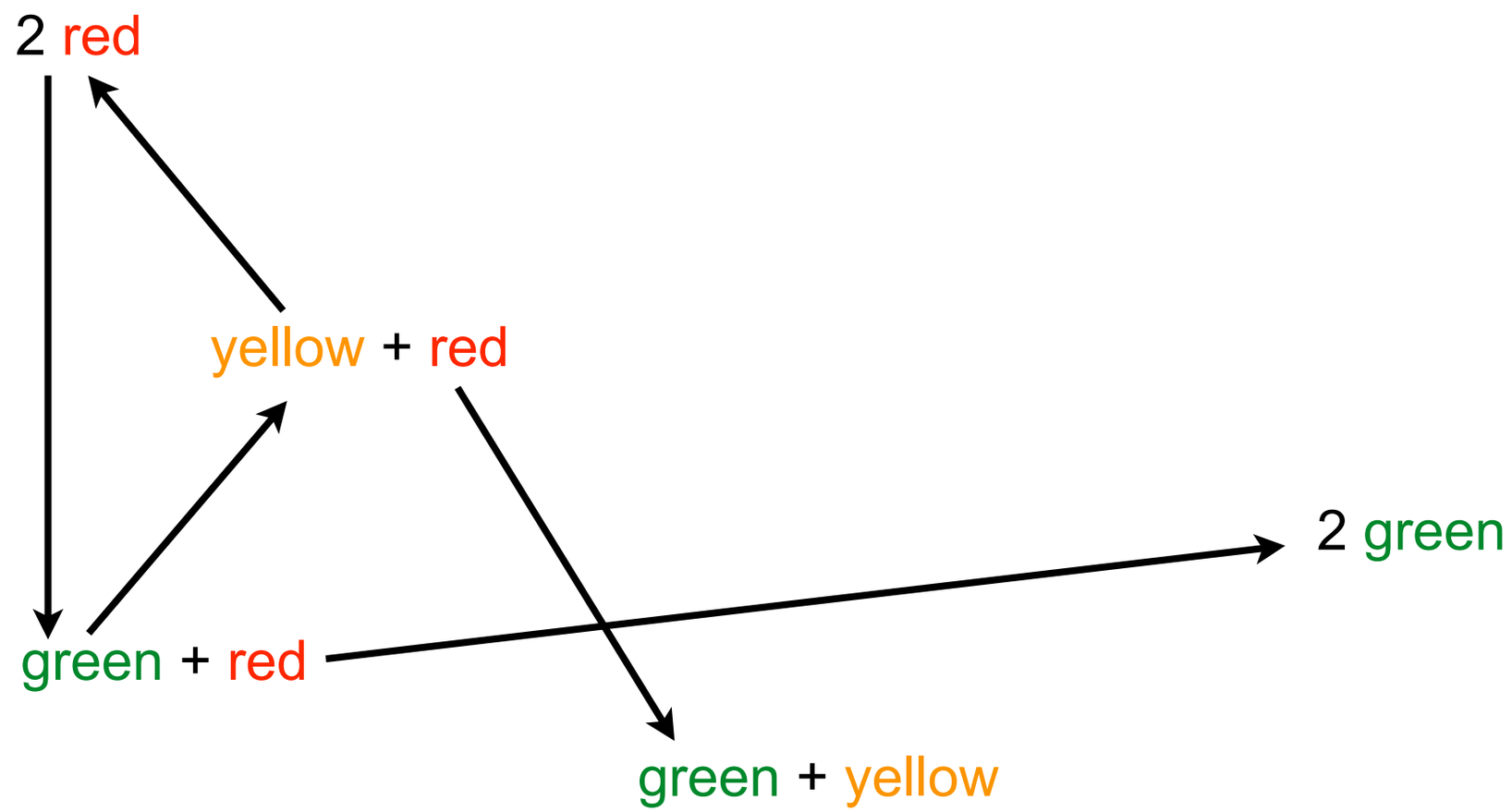


Example: two traffic lights

(we omit arc labels
for readability issues)

Todo = { 2green , green+yellow }

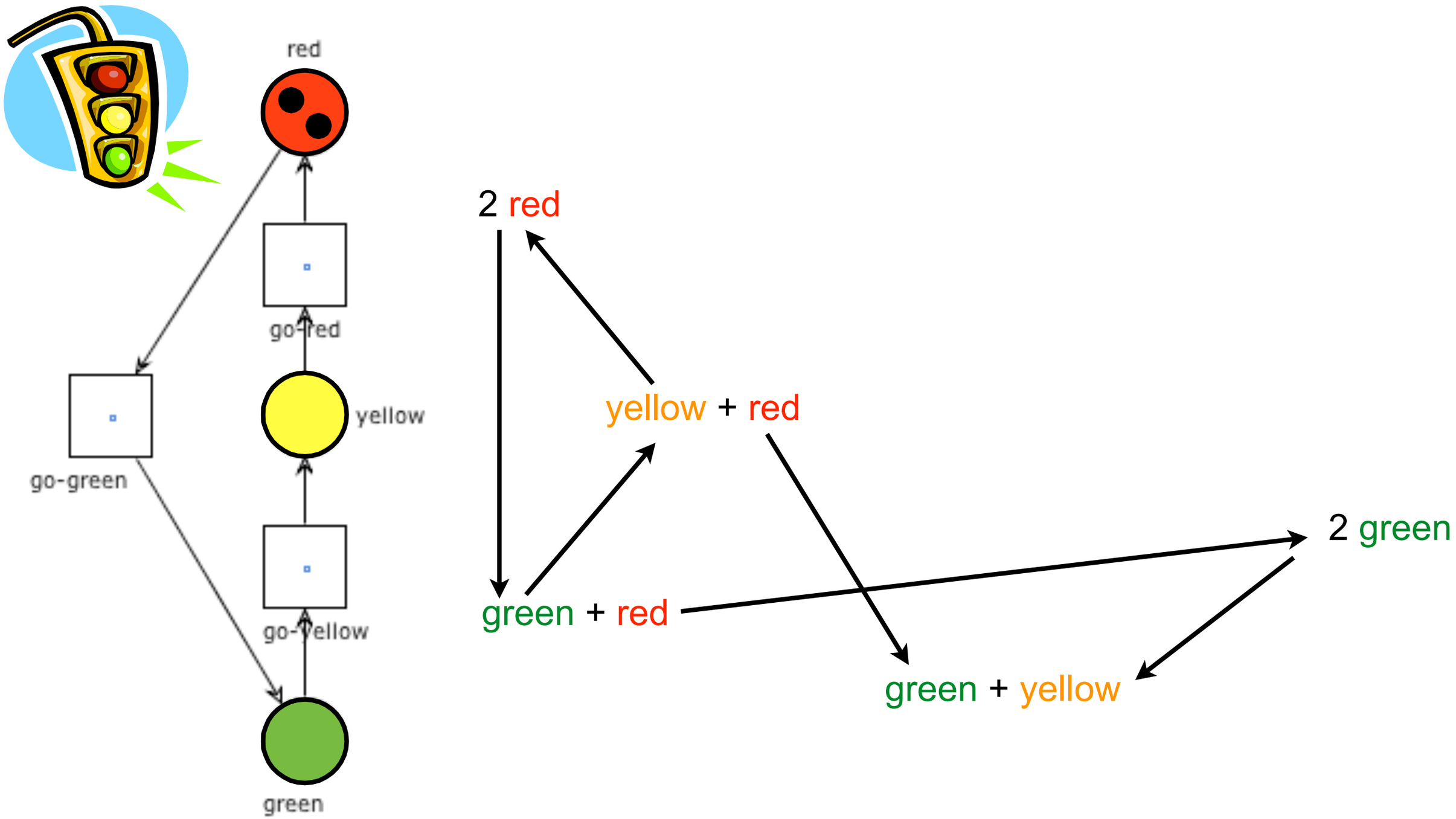




Example: two traffic lights

(we omit arc labels
for readability issues)

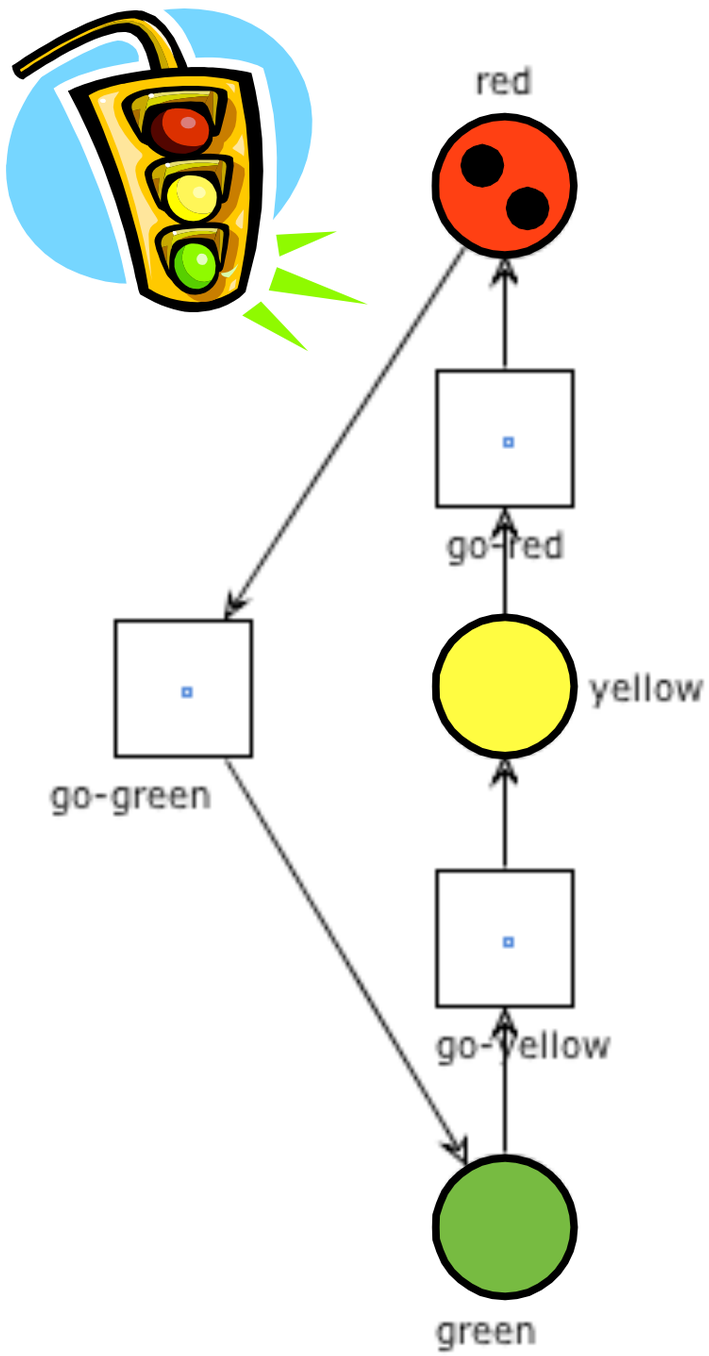
Todo = { green+yellow }

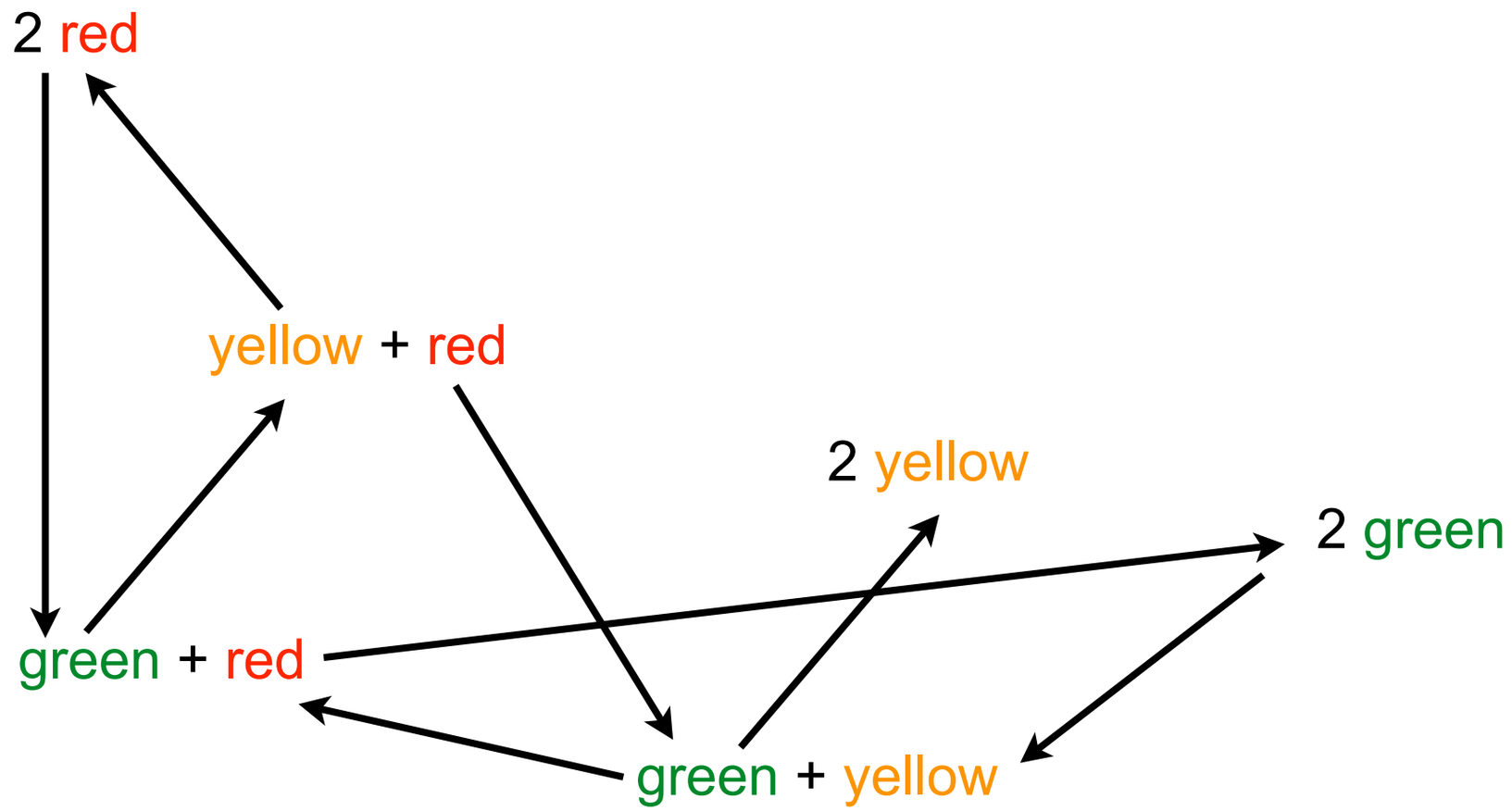


Example: two traffic lights

(we omit arc labels
for readability issues)

Todo = { 2yellow }

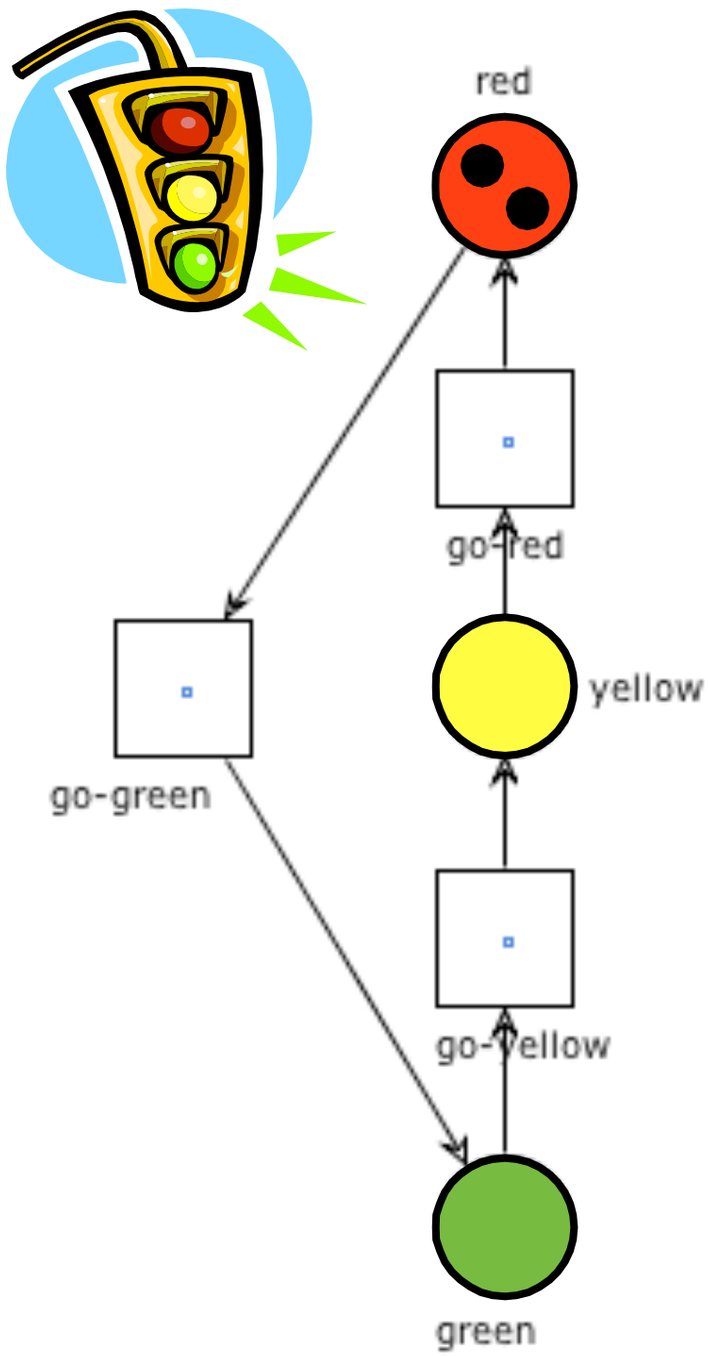


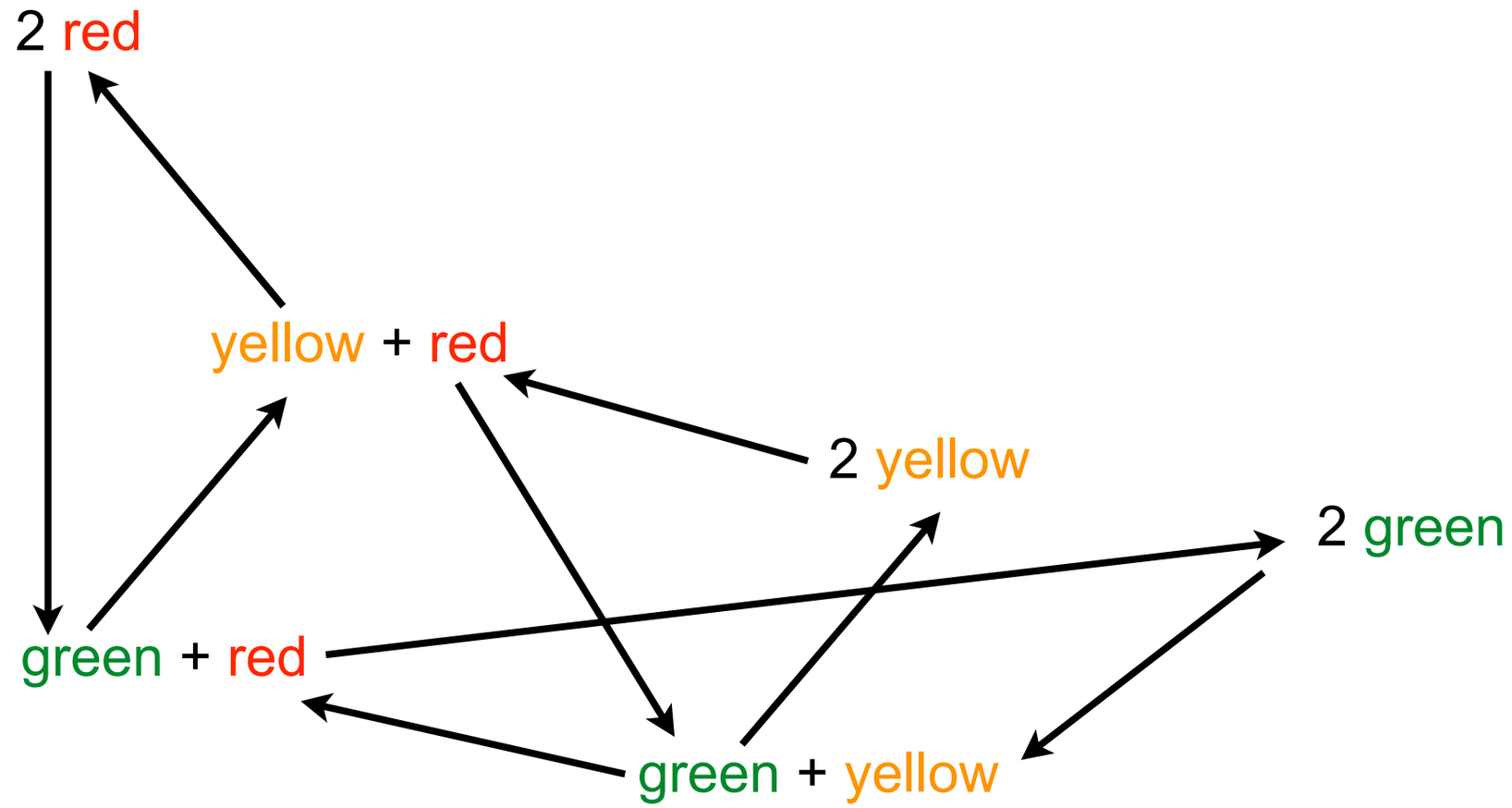


Example: two traffic lights

(we omit arc labels
for readability issues)

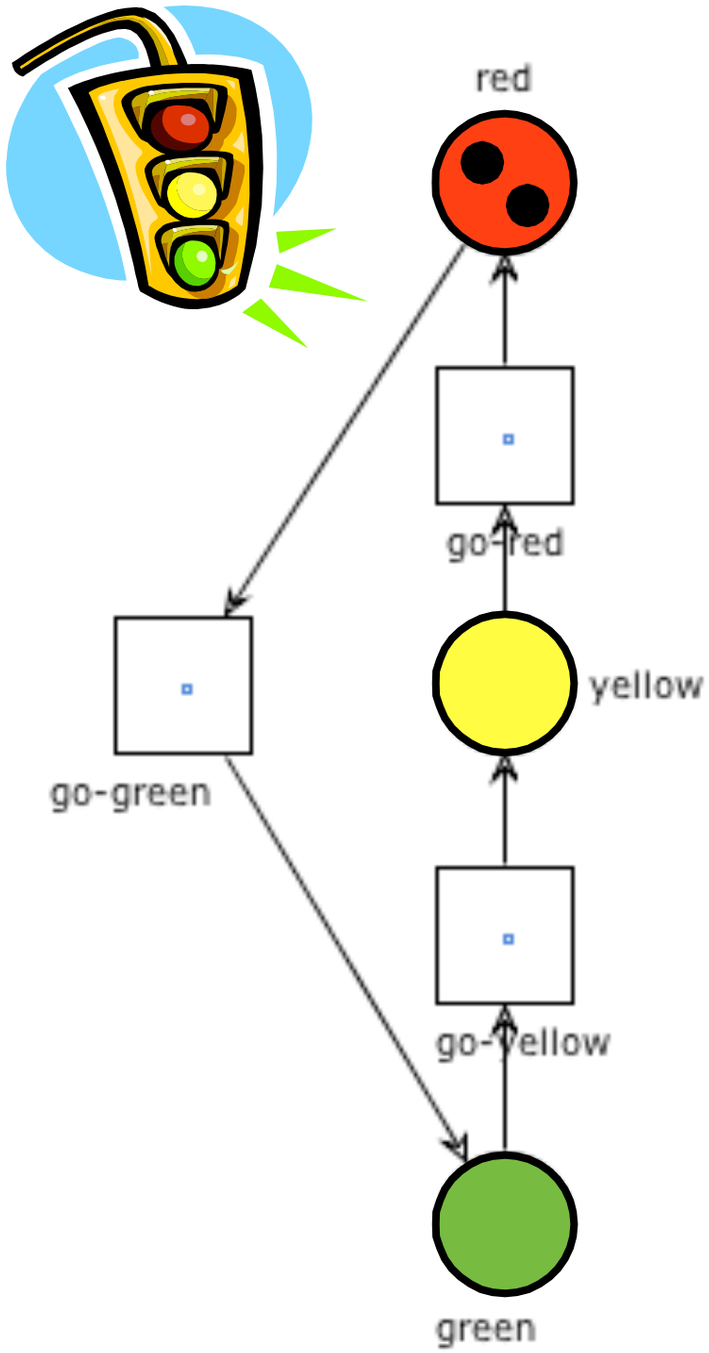
Todo = { }

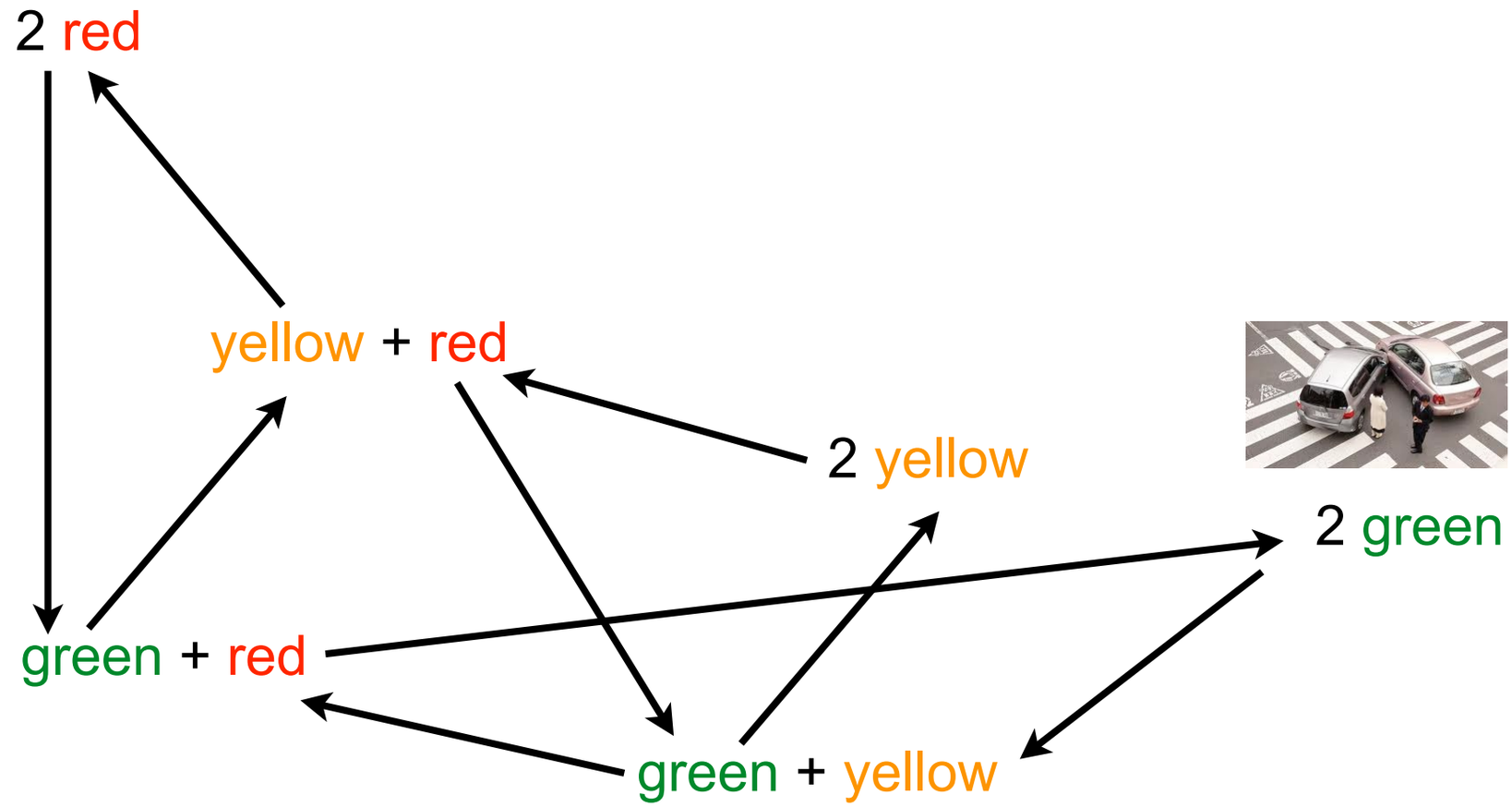




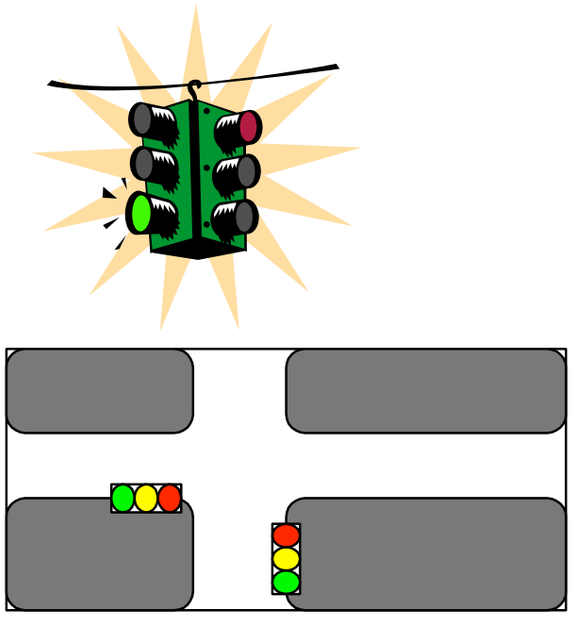
Example: two traffic lights

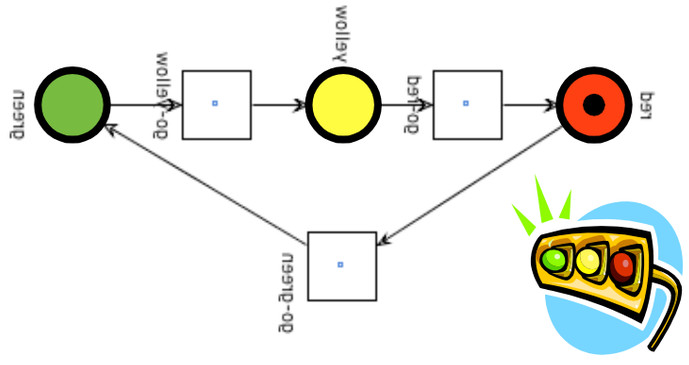
(we omit arc labels
for readability issues)





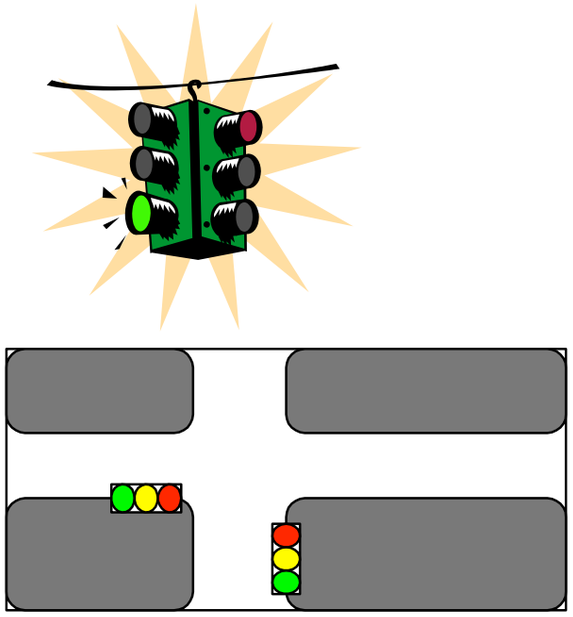
Question time



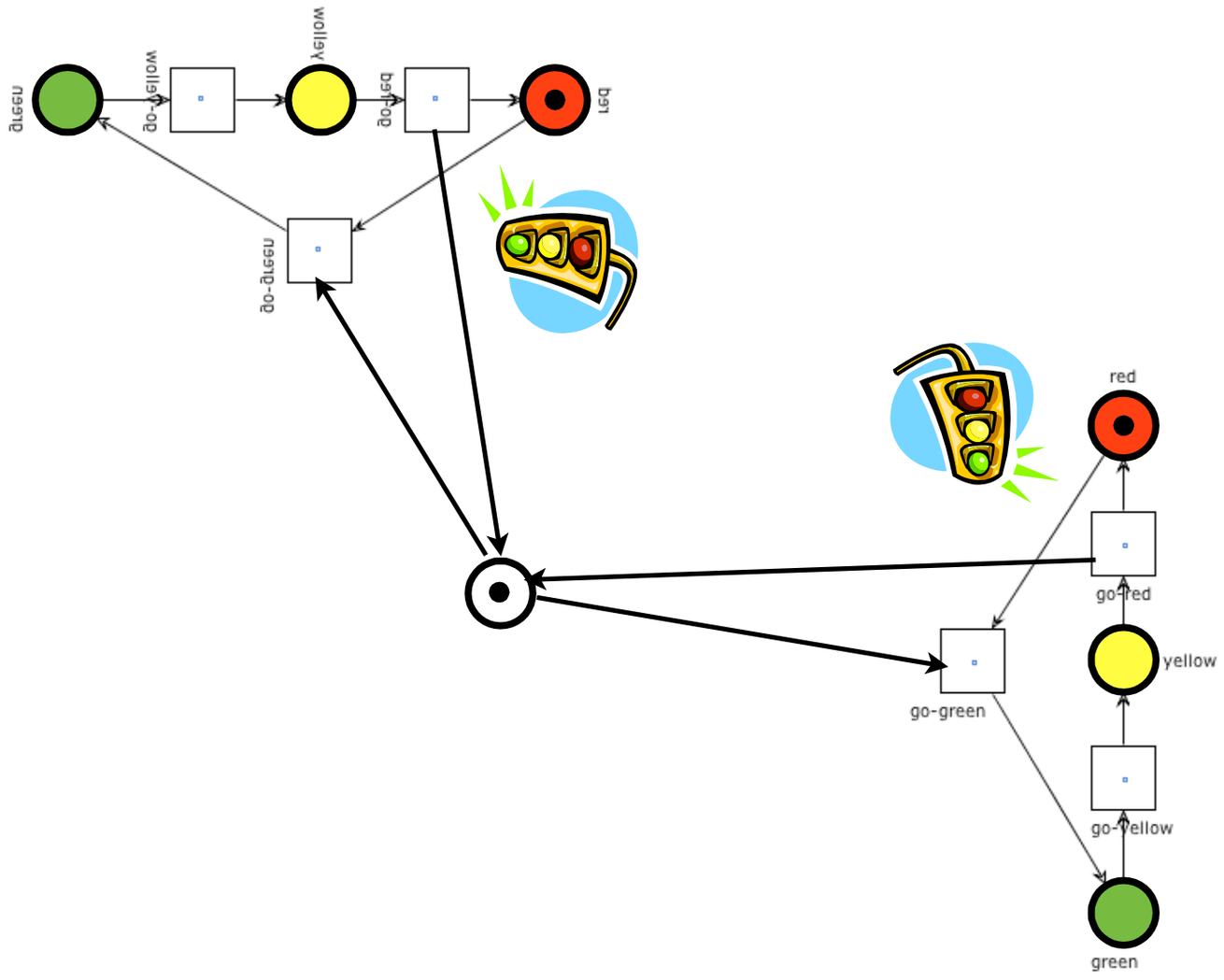


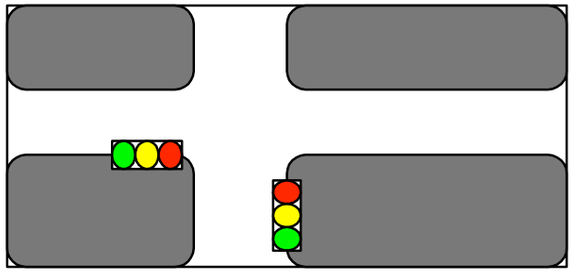
Complete the net in such a way that the two lights can never be green at the same time

Question time



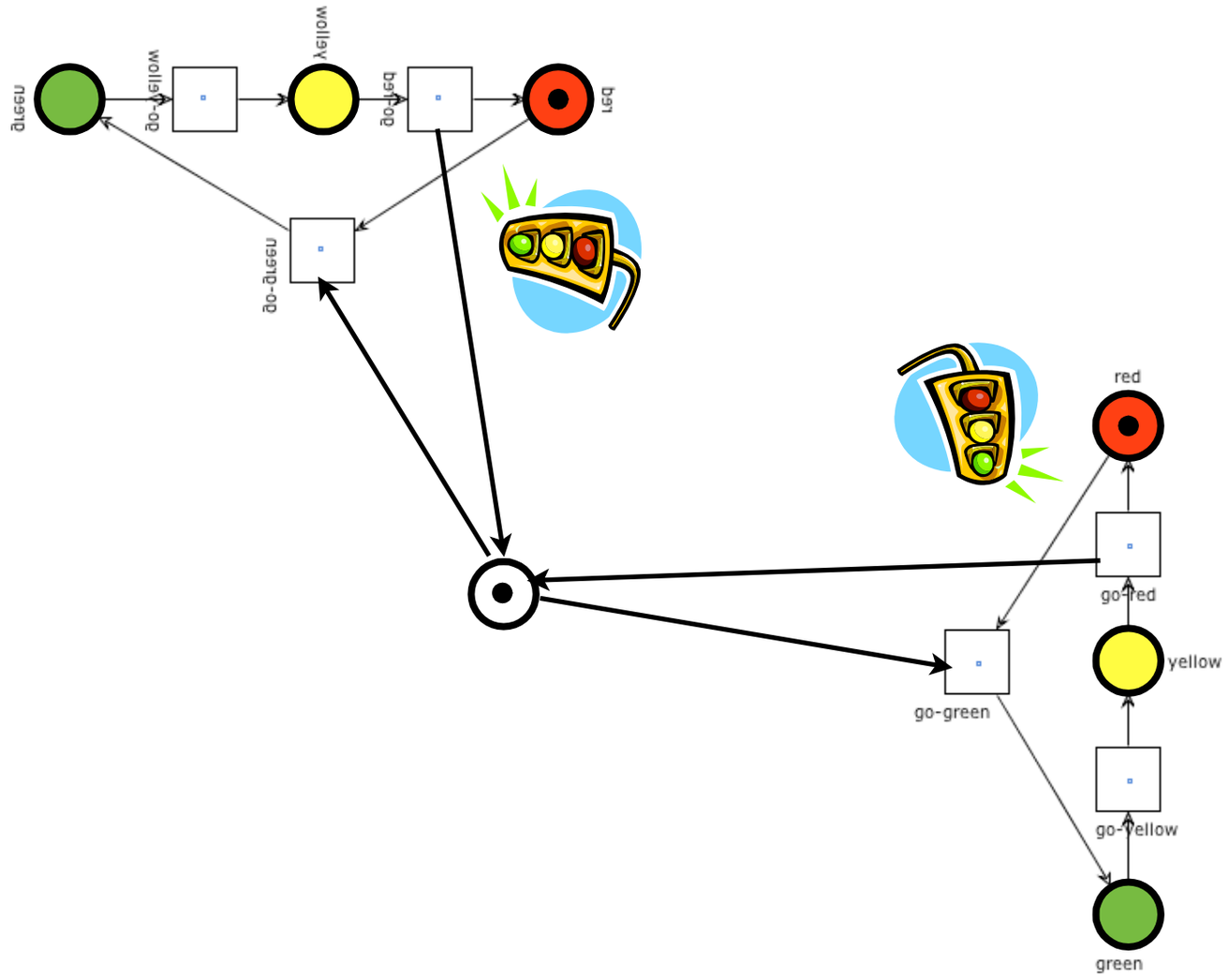
Complete the net in
such a way that
the two lights
can never be green
at the same time

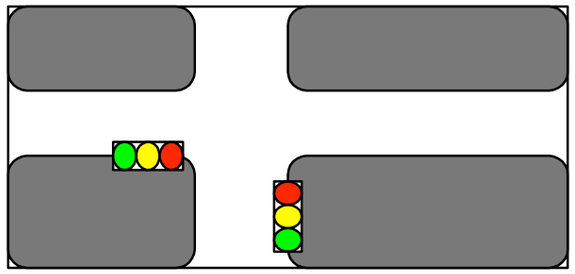




Exercises

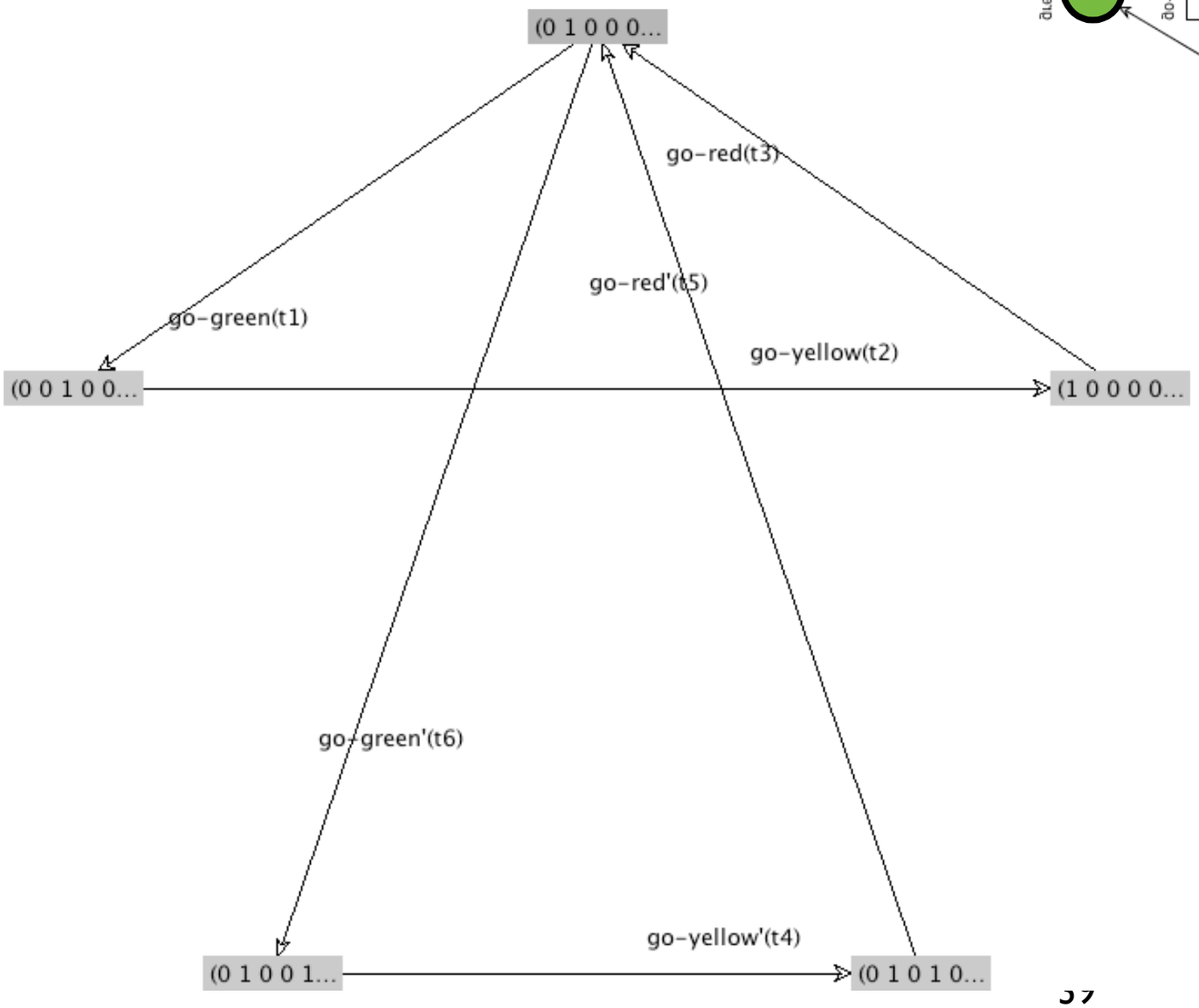
Draw the reachability graph of the last net

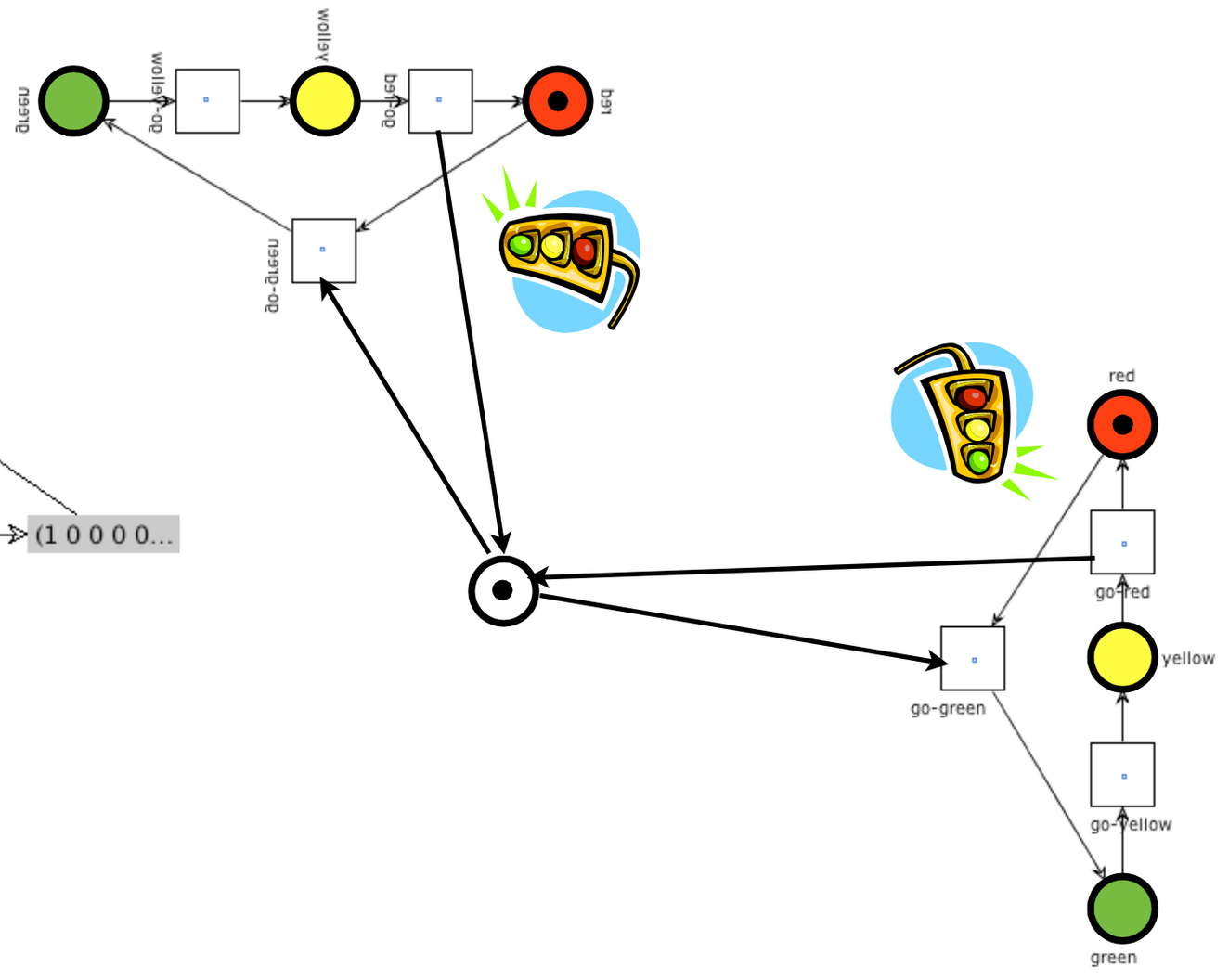


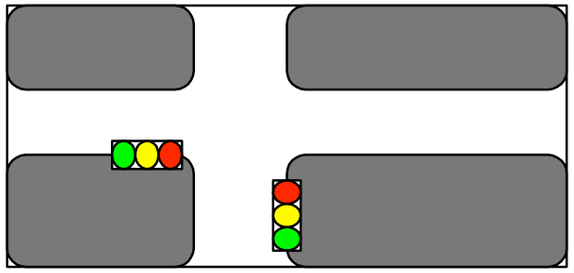


Exercises

Draw the reachability graph of the last net

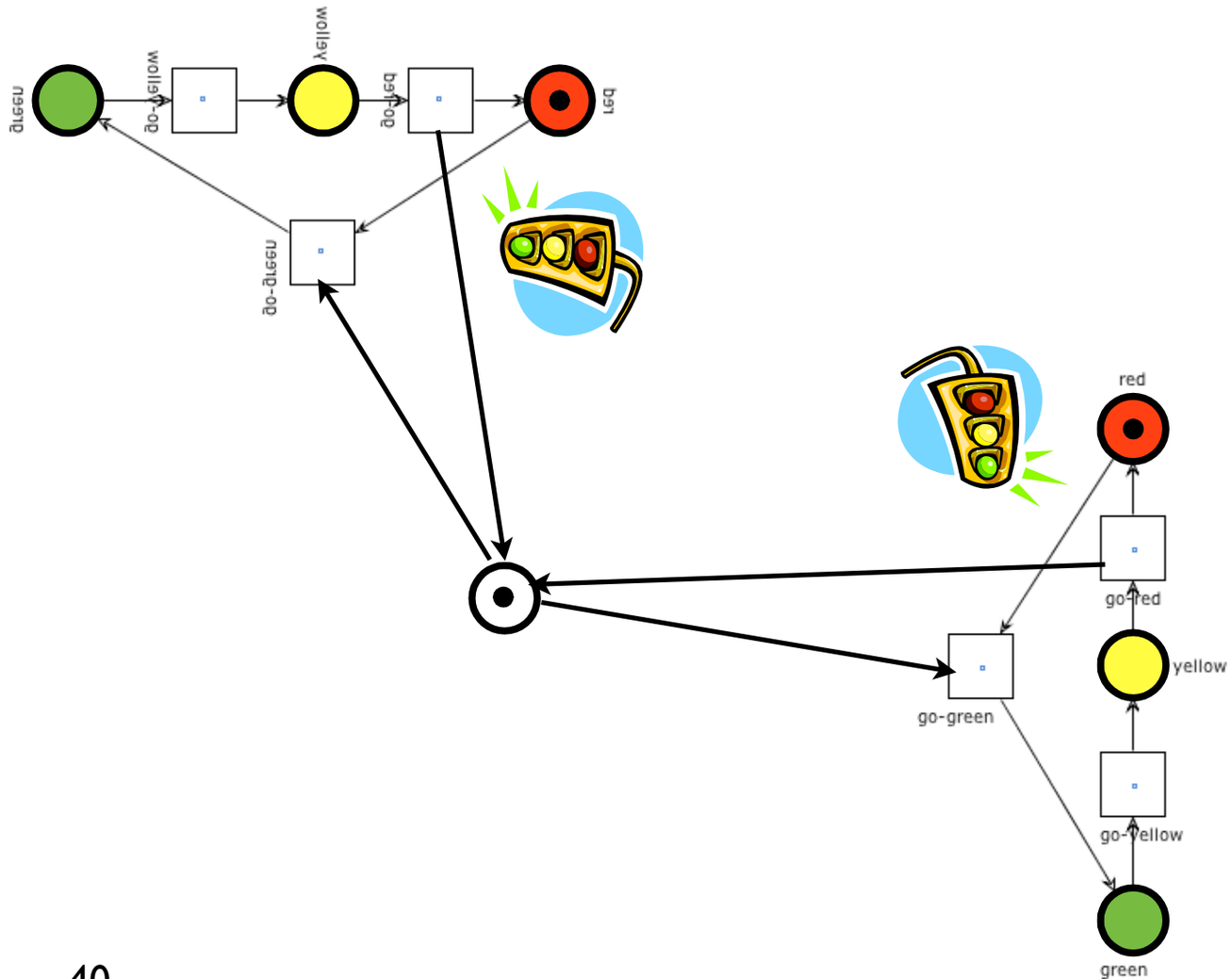


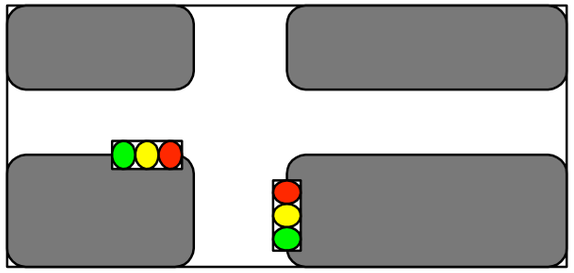




Exercises

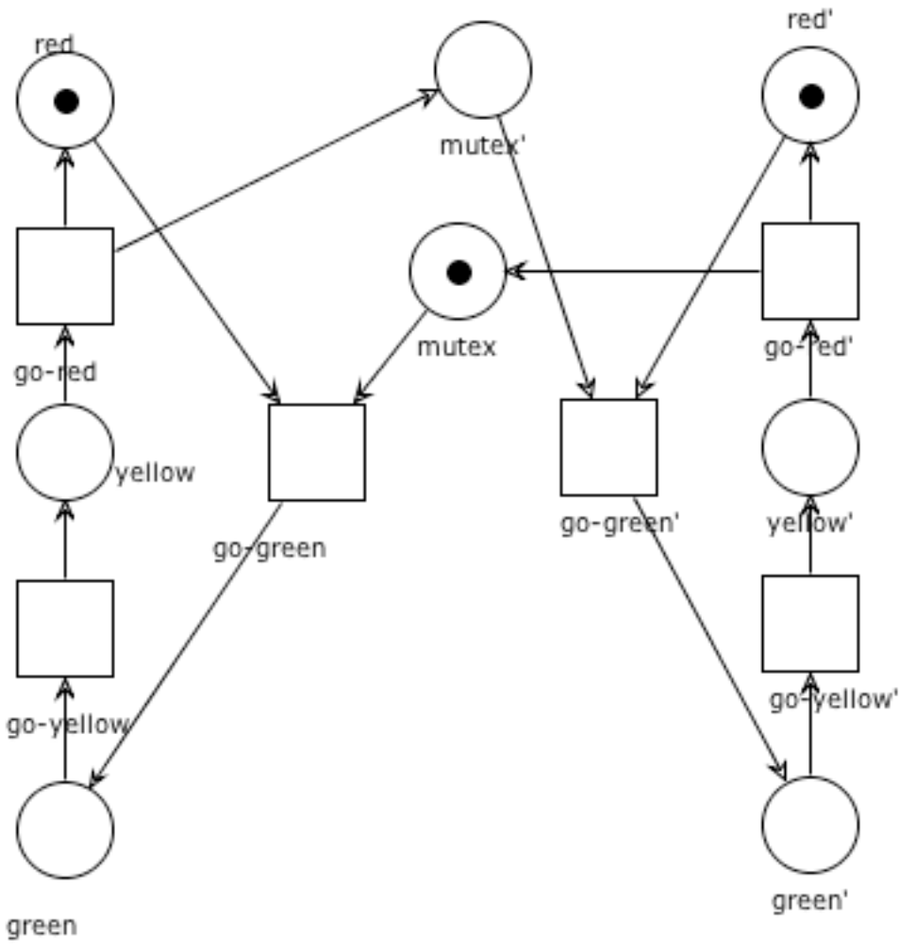
Modify the net so to guarantee that green alternate on the two traffic lights and then draw the reachability graph

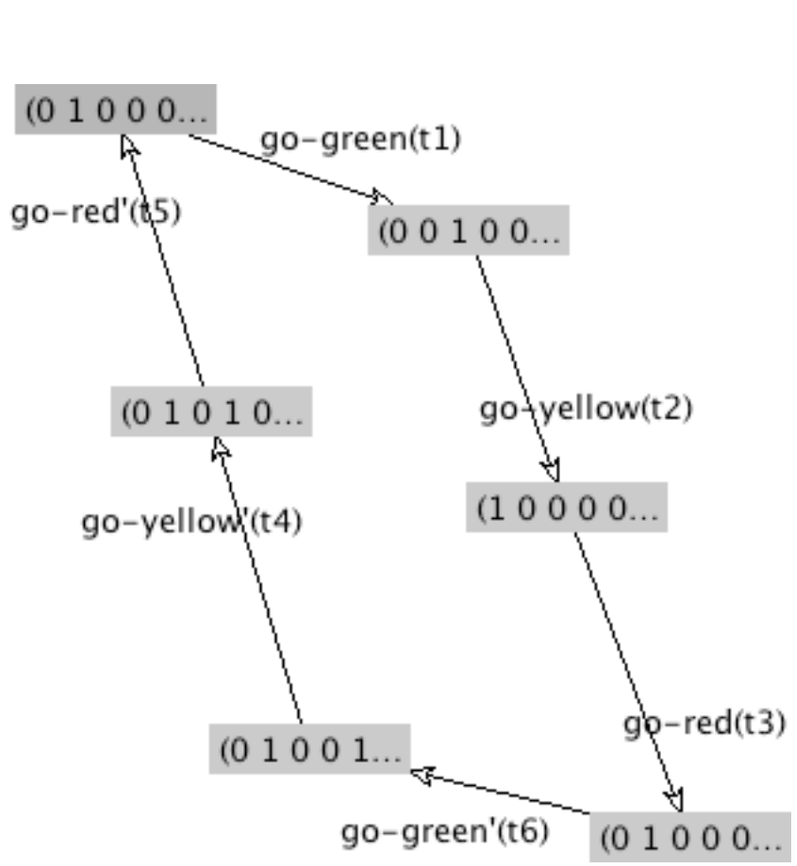


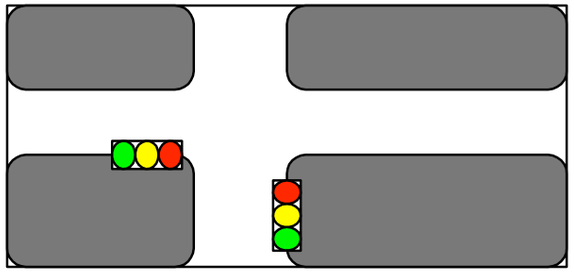


Exercises

Modify the net so to guarantee that green alternate on the two traffic lights and then draw the reachability graph

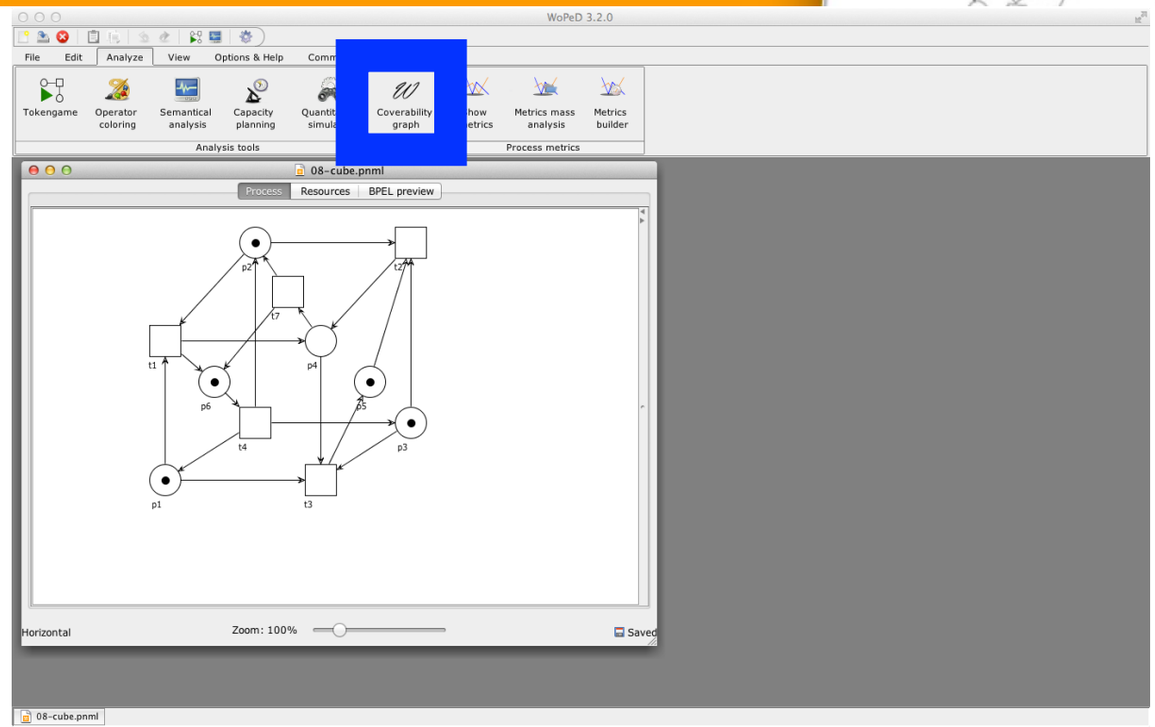


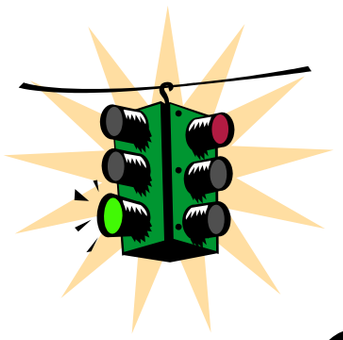




Exercises

Play the “token games” on the previous nets using Workflow Petri net Designer: http://www.woped.org





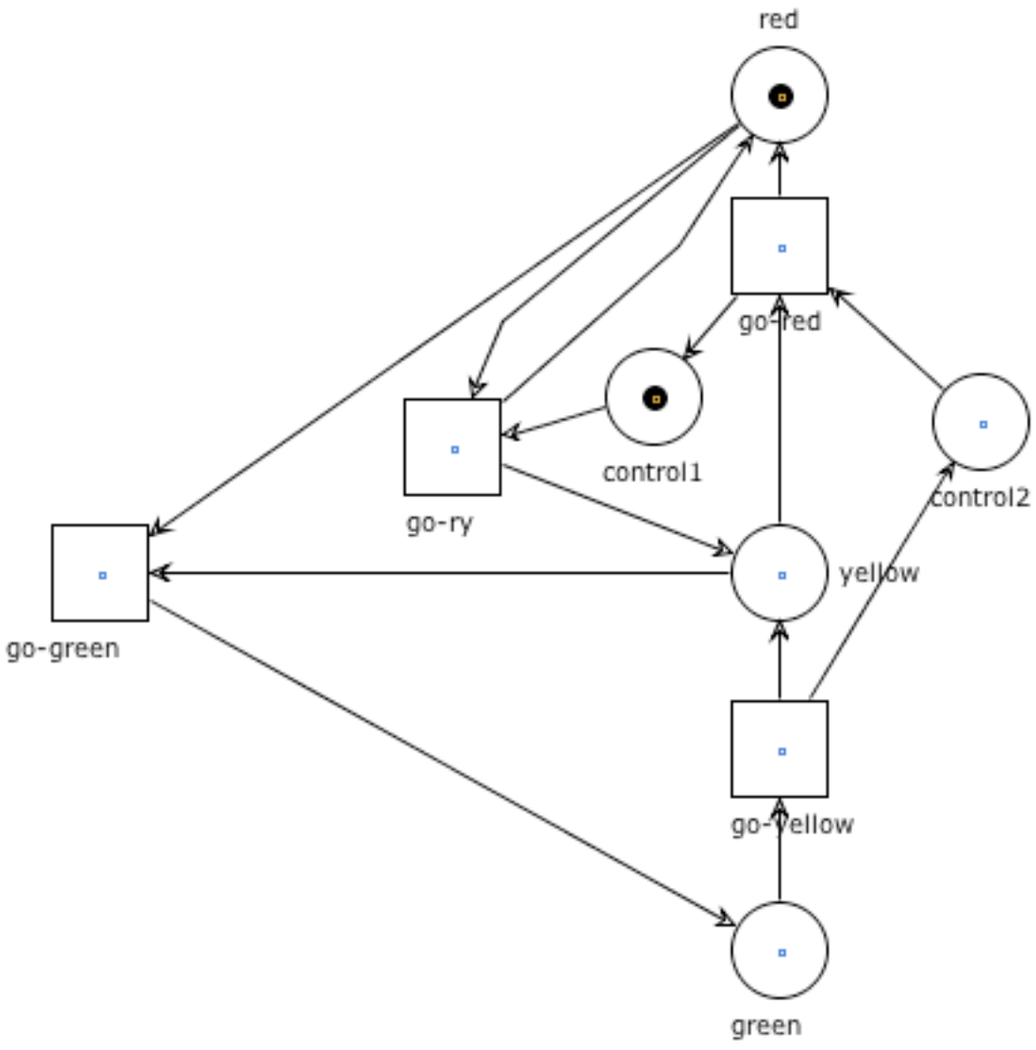
Exercise: German traffic lights

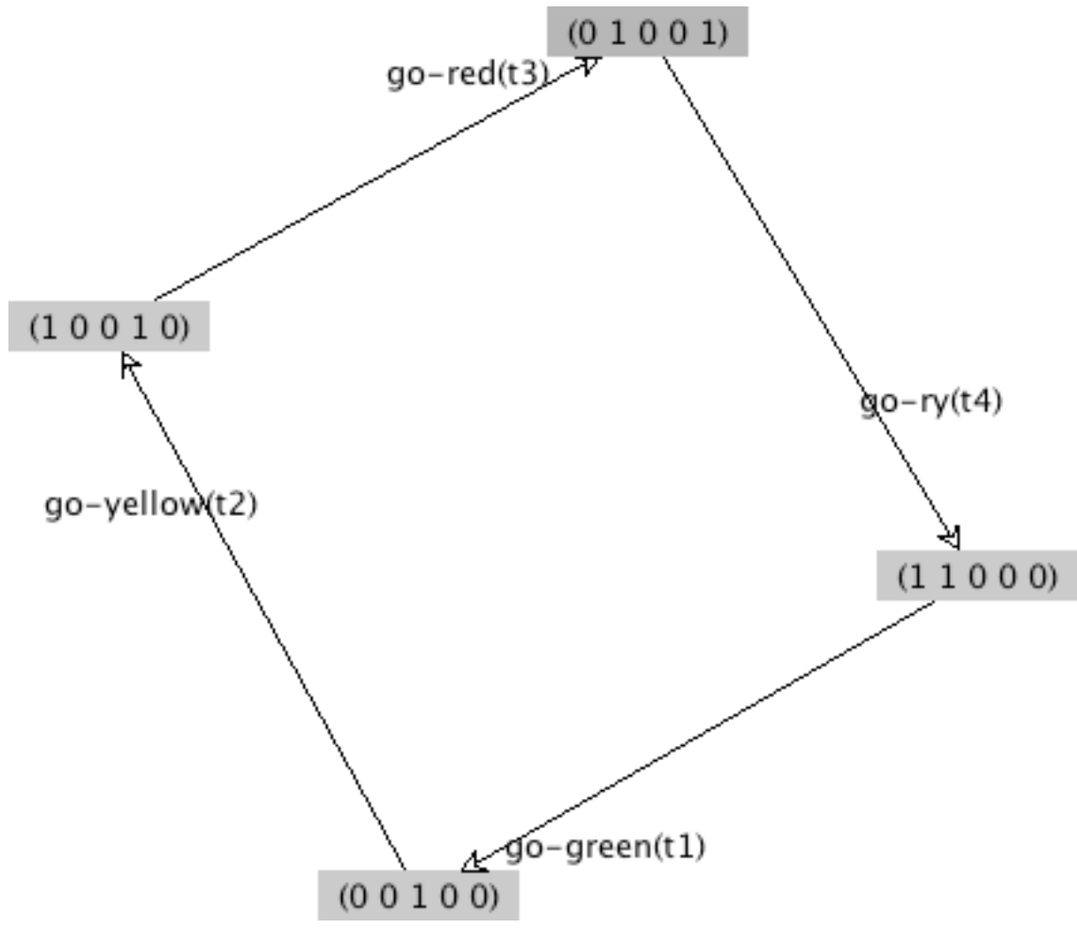
German traffic lights have an extra phase:
traffic lights do not turn suddenly from red to green but
give a red light together with a yellow light before turning to green.

Identify the possible states and model the automaton that lists all possible states and state transitions.

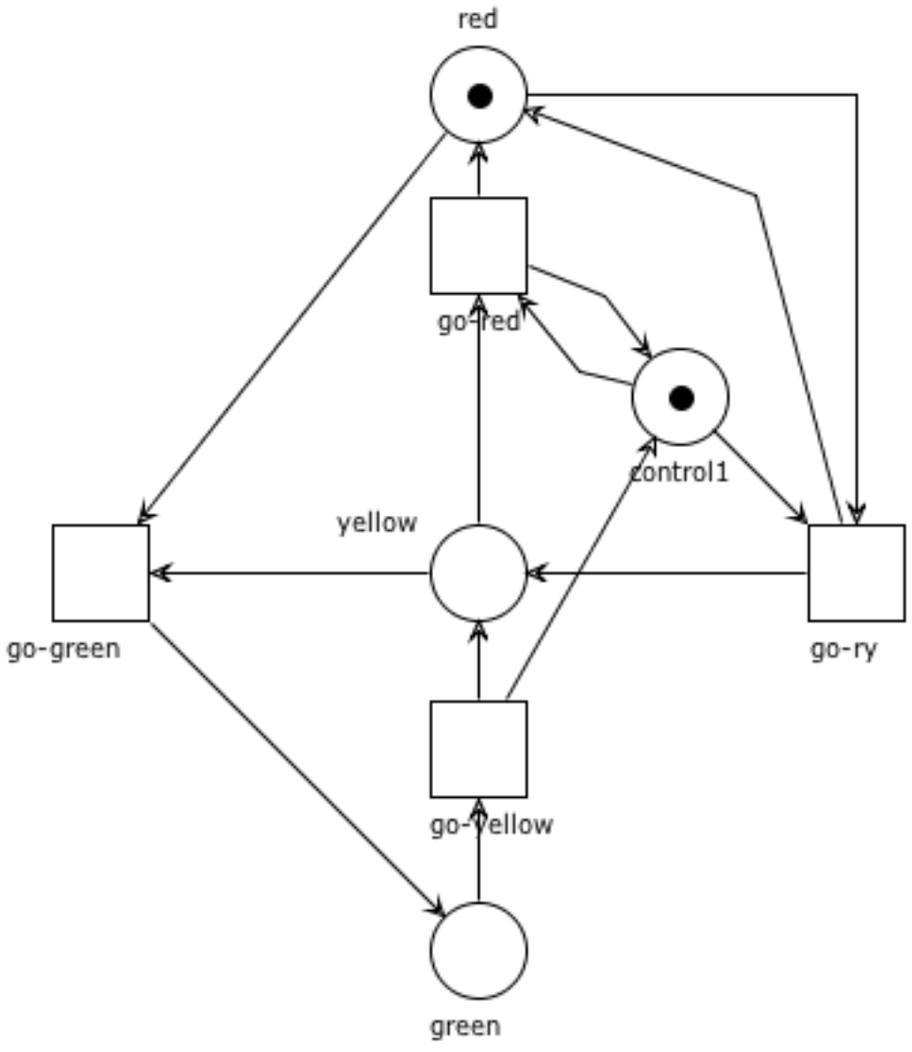
Design a Petri net that behaves exactly like a German traffic light. There should be three places indicating the state of each light and make sure that the Petri net does not allow state transitions which should not be possible.

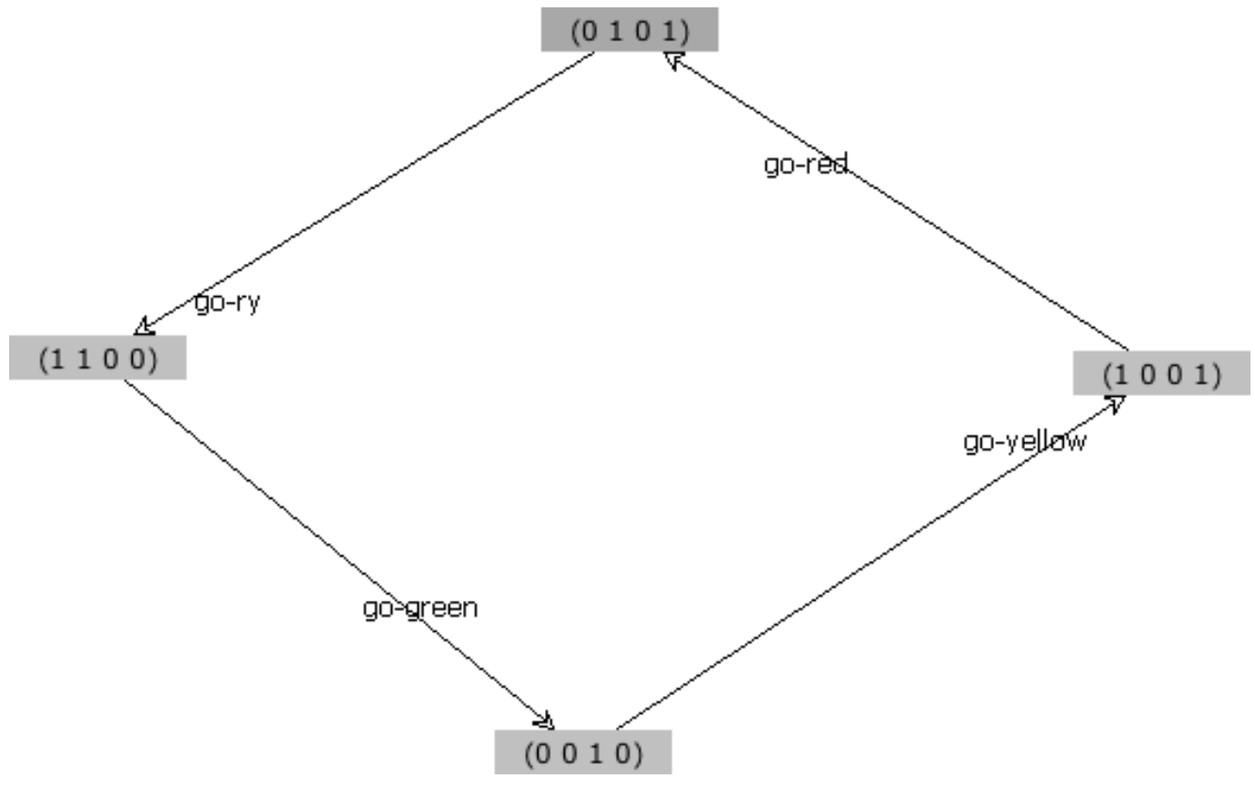
German traffic lights





German traffic lights





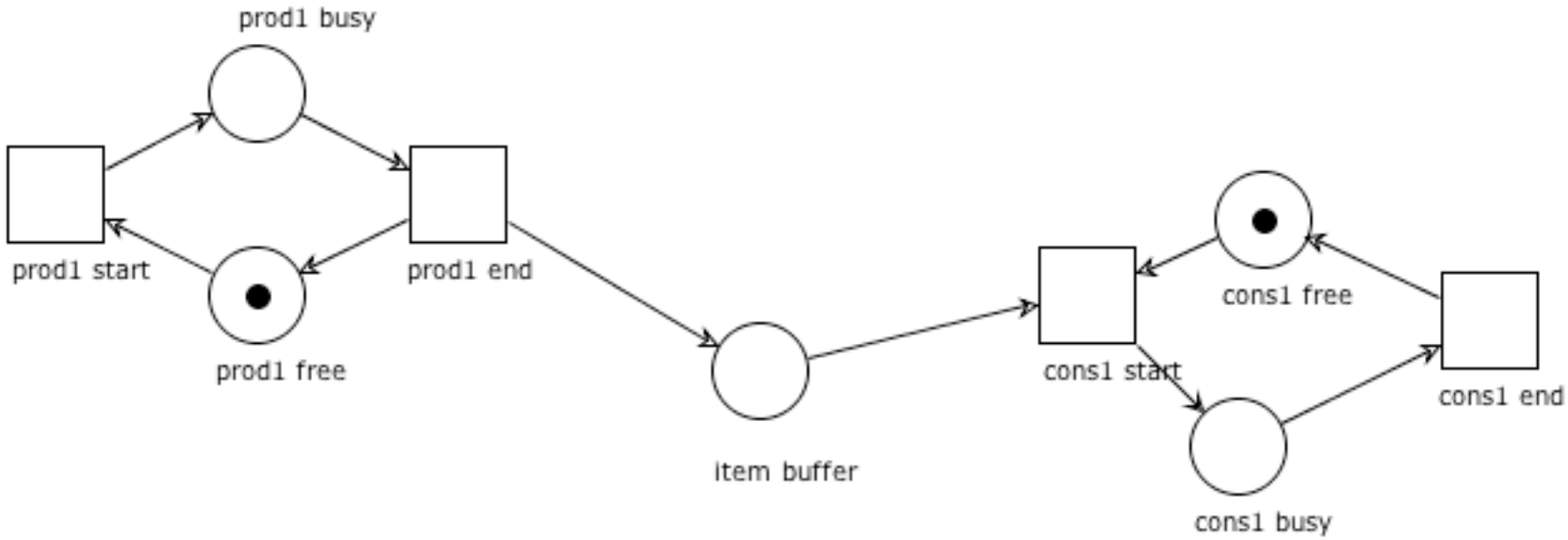
Exercise:

Producer and consumer

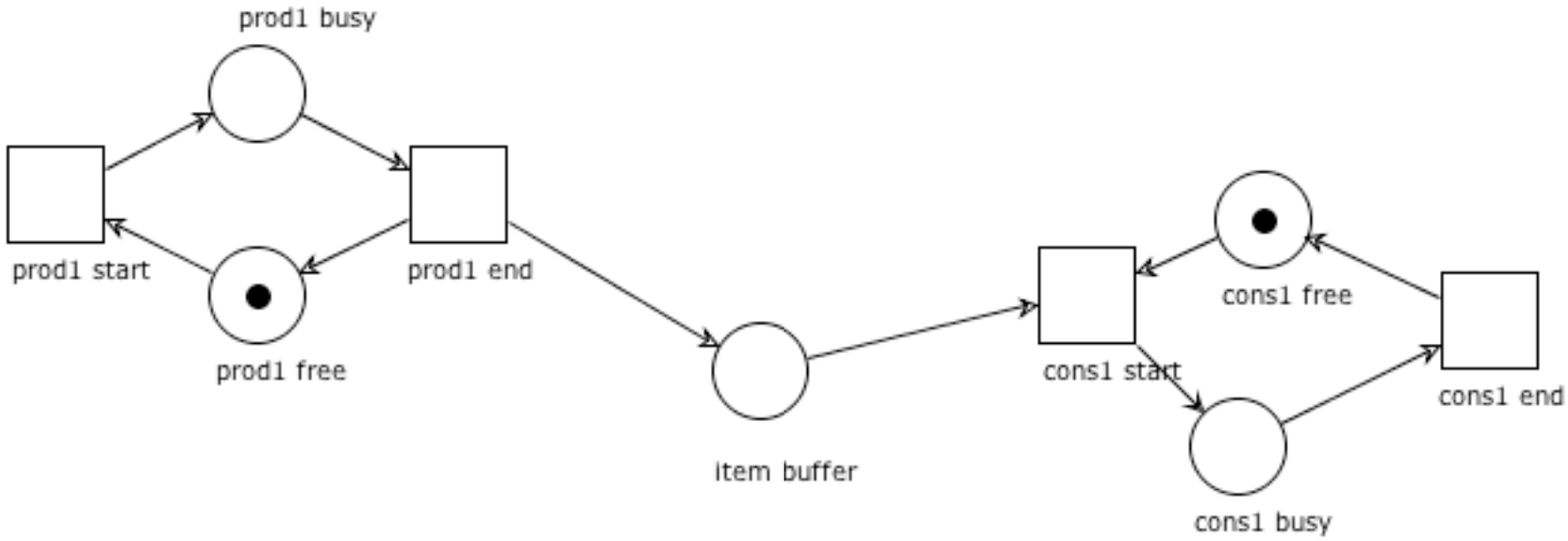
Model a process with one producer and one consumer:
Each one is either busy or free.
Each one alternates between these two states
After every production cycle the producer puts a product in a buffer and the consumer consumes one product from this buffer (when available) per cycle.

How to model 4 producers and 3 consumers connected through a single buffer?
How to limit the size of the buffer to 2 items?
Draw the reachability graph

Producer and consumer

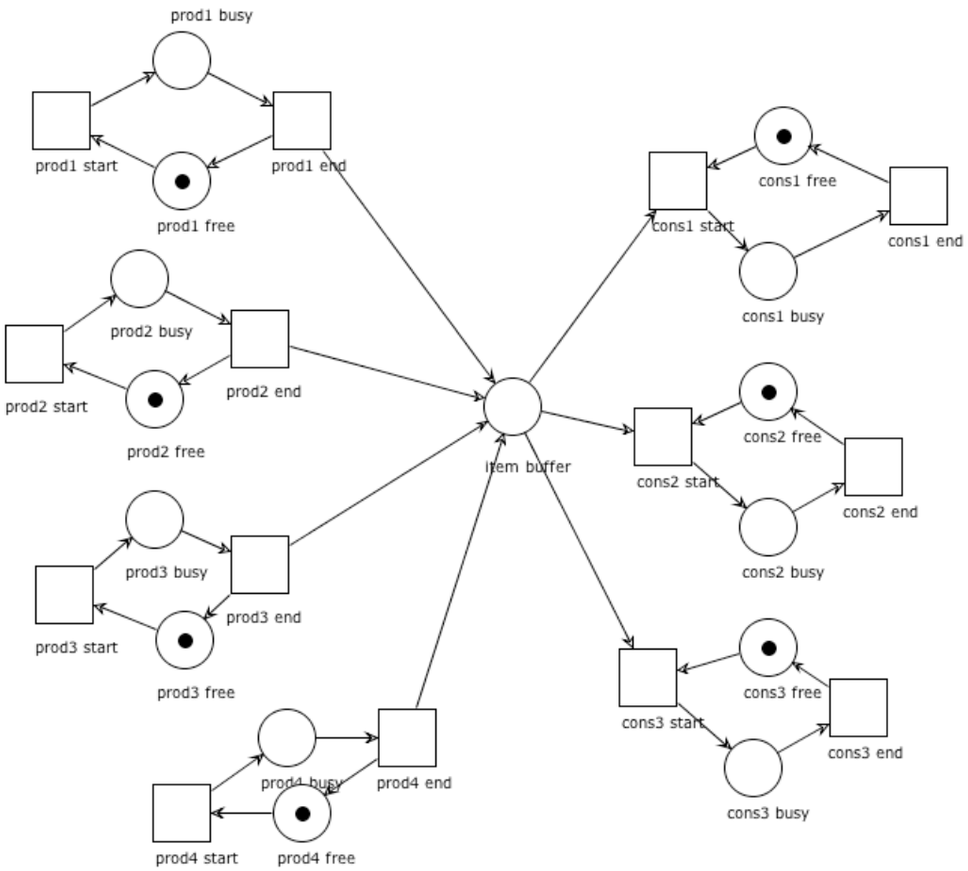


Producer and consumer

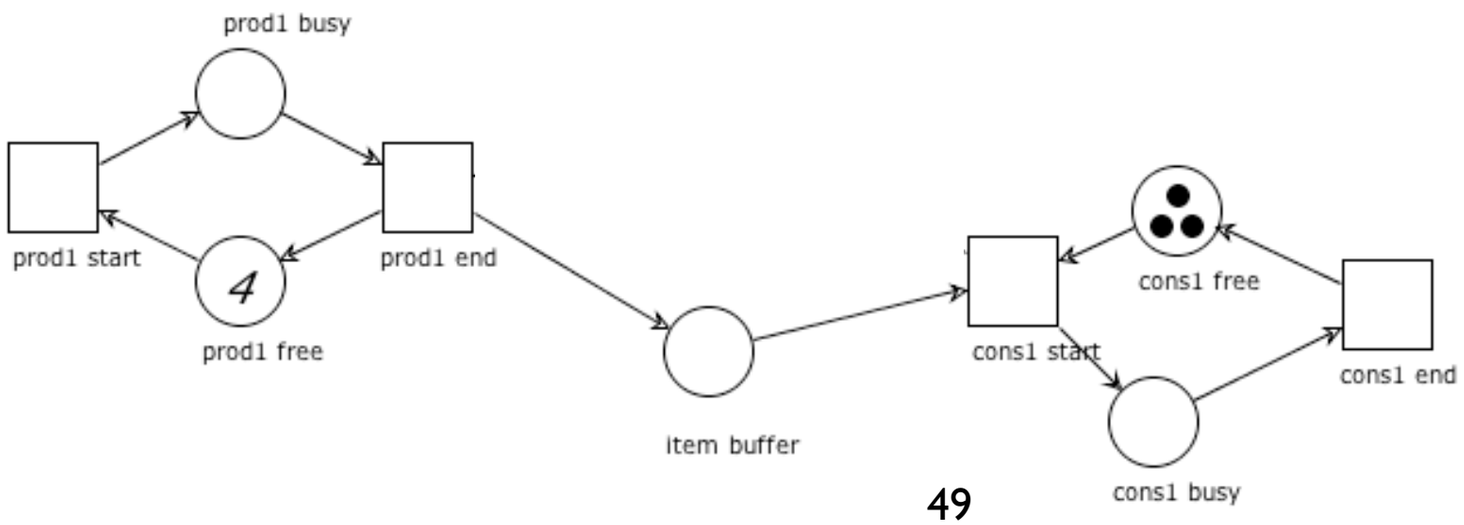


How to model 4 producers and 3 consumers connected through a single buffer?

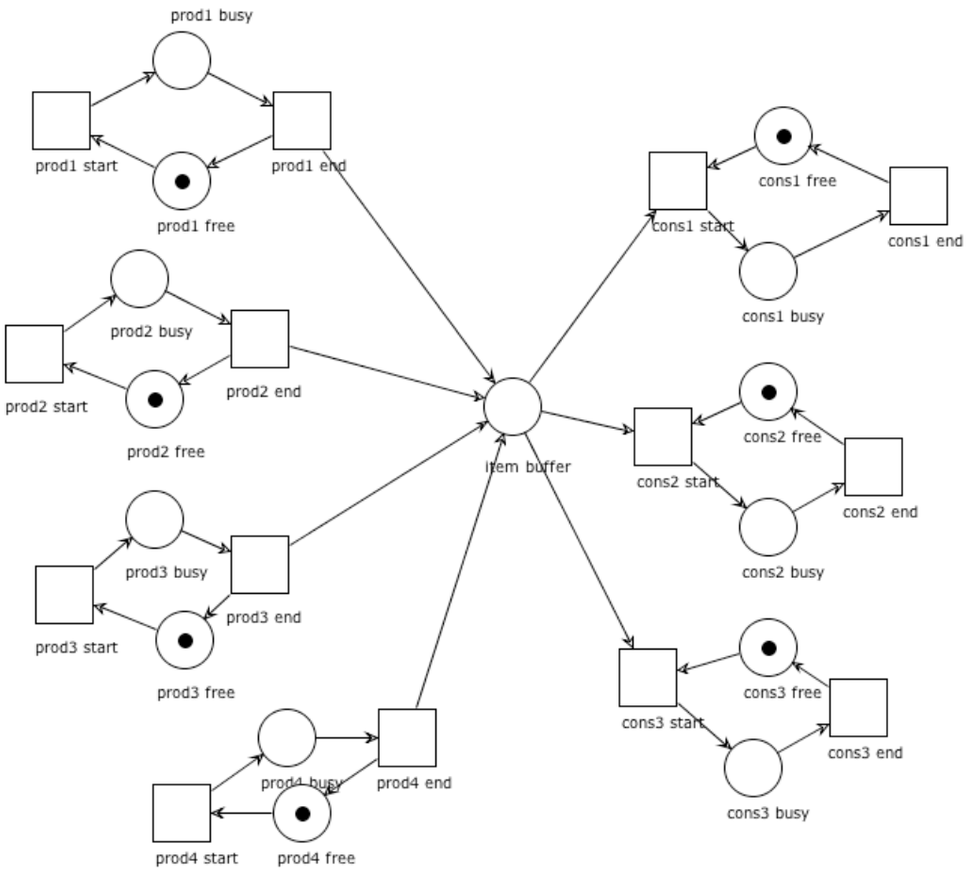
Producers and consumers



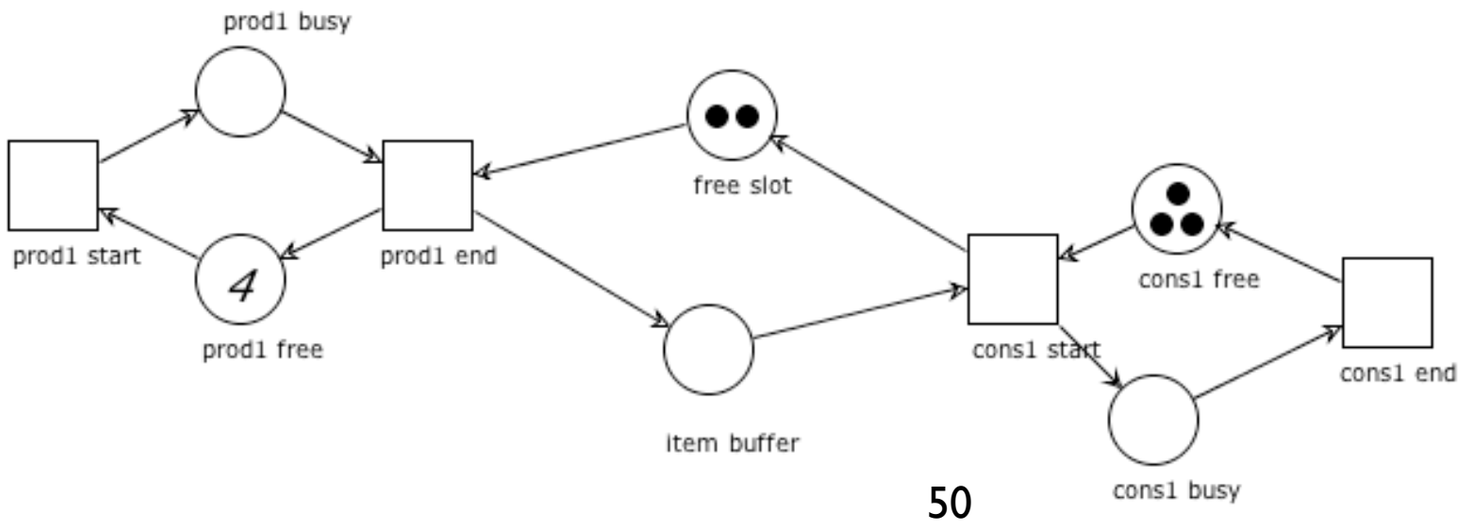
How to limit the size of the buffer to 2 items?



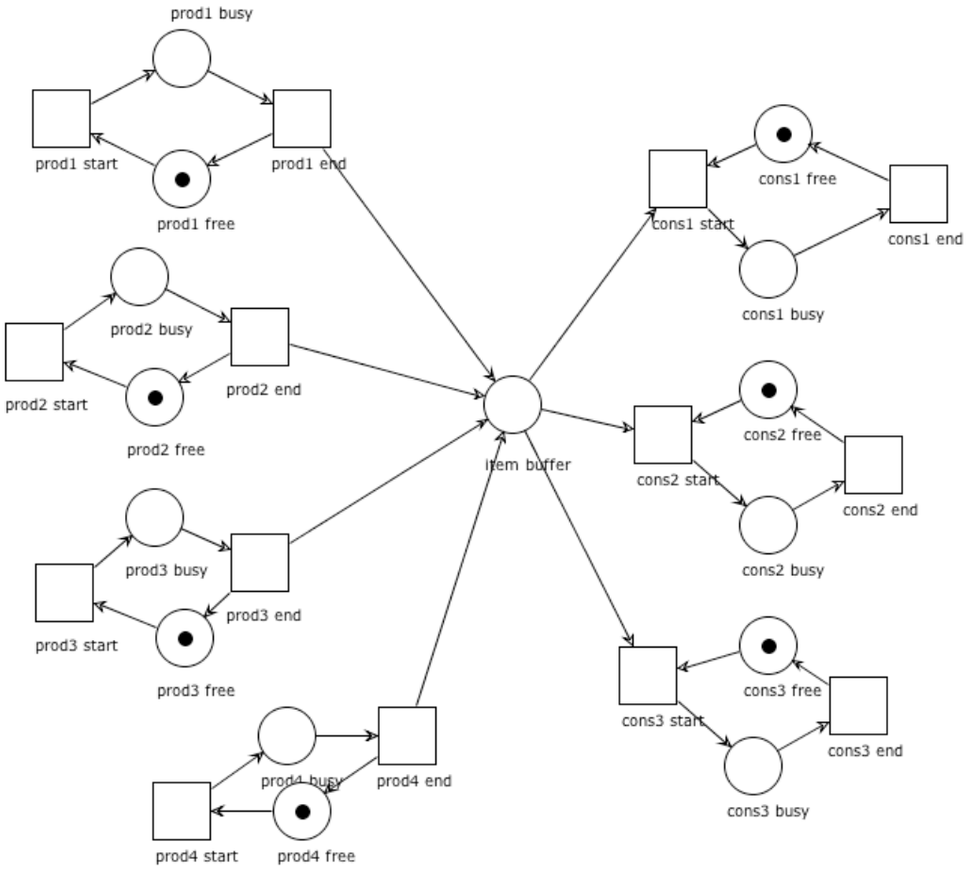
Producers and consumers

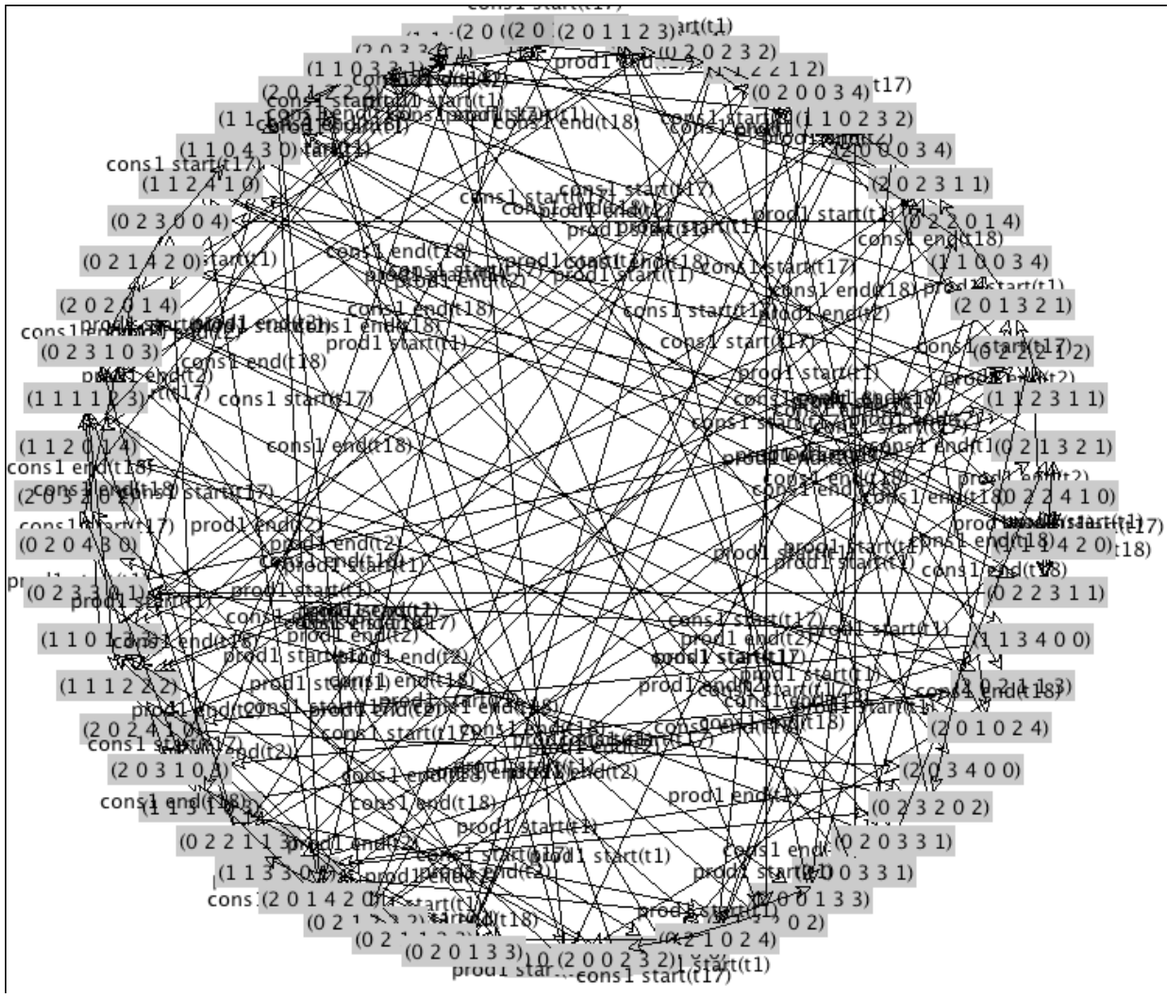


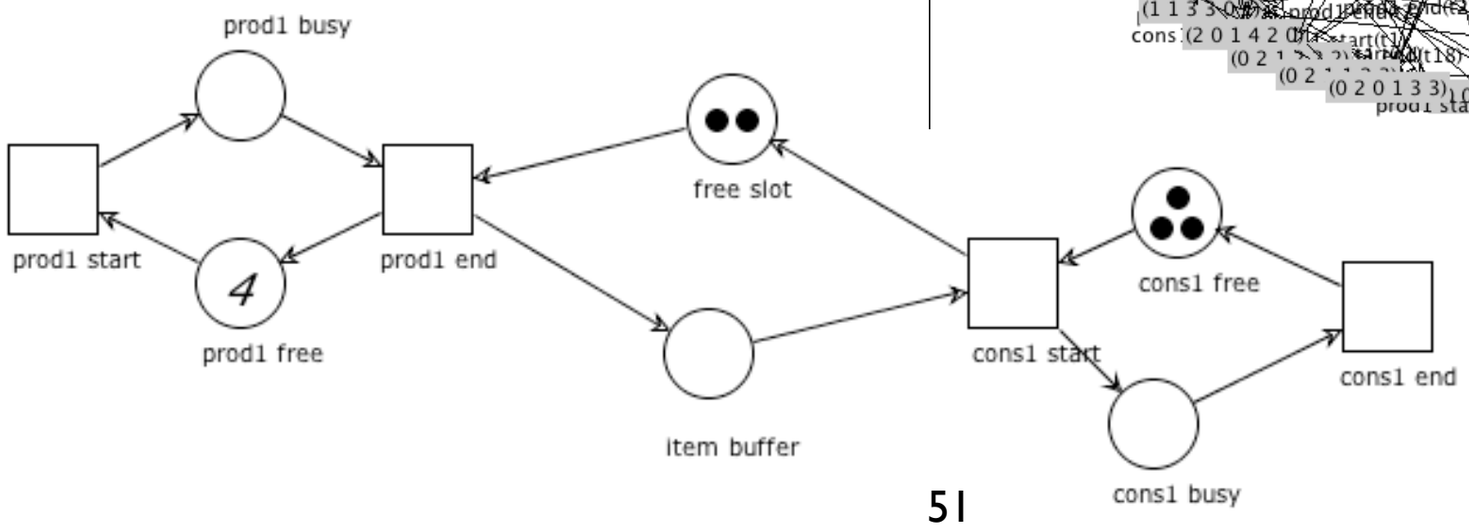
Draw the reachability graph

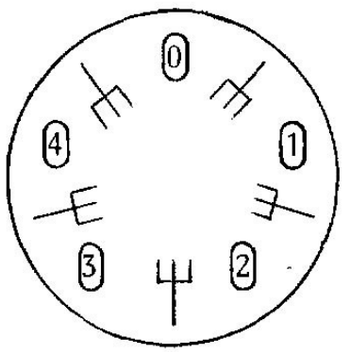


Producers and consumers





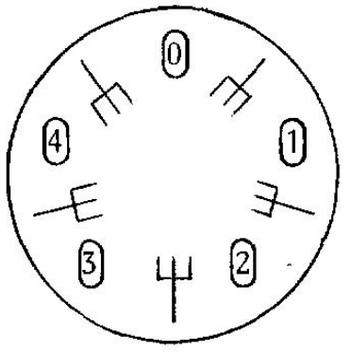




Exercise: Dining philosophers

The problem is originally due to E.W. Dijkstra (and soon elaborated by T. Hoare) as an examination question on a synchronization problem where five computers competed for access to five shared tape drive peripherals.

It can be used to illustrate several important concepts in concurrency (mutual exclusion, deadlock, starvation)

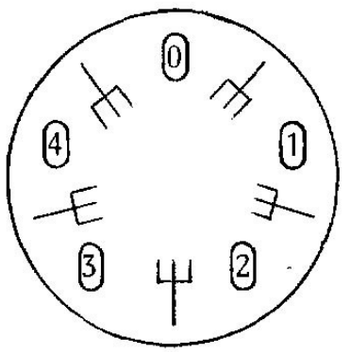


Exercise: Dining philosophers

The life of a philosopher consists of an alternation of thinking and eating

Five philosophers are living in a house where a table is laid for them, each philosopher having his own place at the table

Their only problem (besides those of philosophy) is that the dish served is a very difficult kind of spaghetti, that has to be eaten with two forks. There are two forks next to each plate, so that presents no difficulty: as a consequence, however, no two neighbours may be eating simultaneously.



Exercise: Dining philosophers

Design a net for representing the dining philosophers problem, then use WoPeD to compute the reachability graph

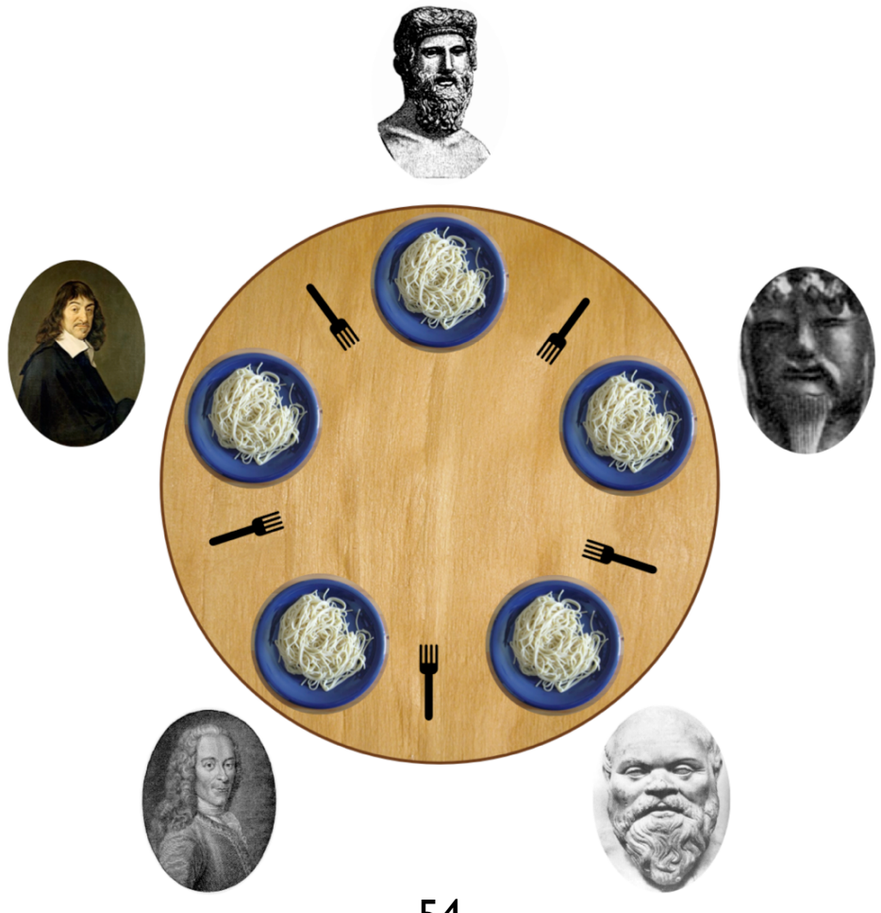
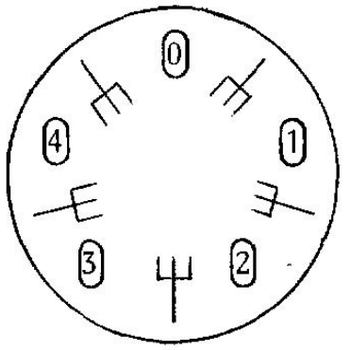
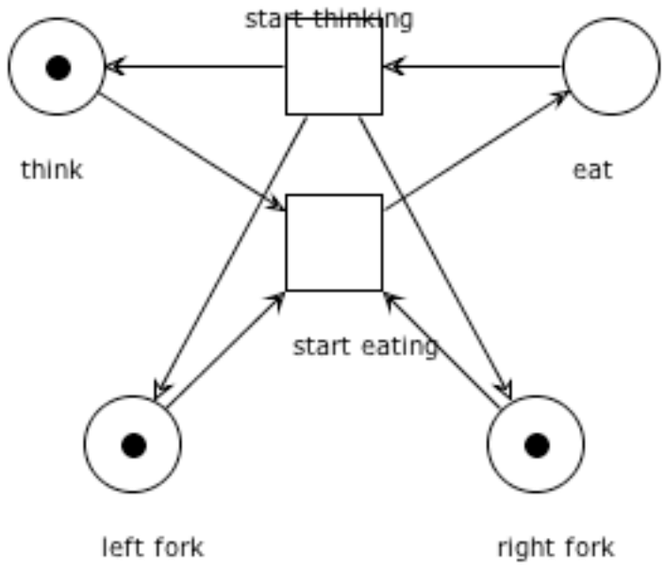
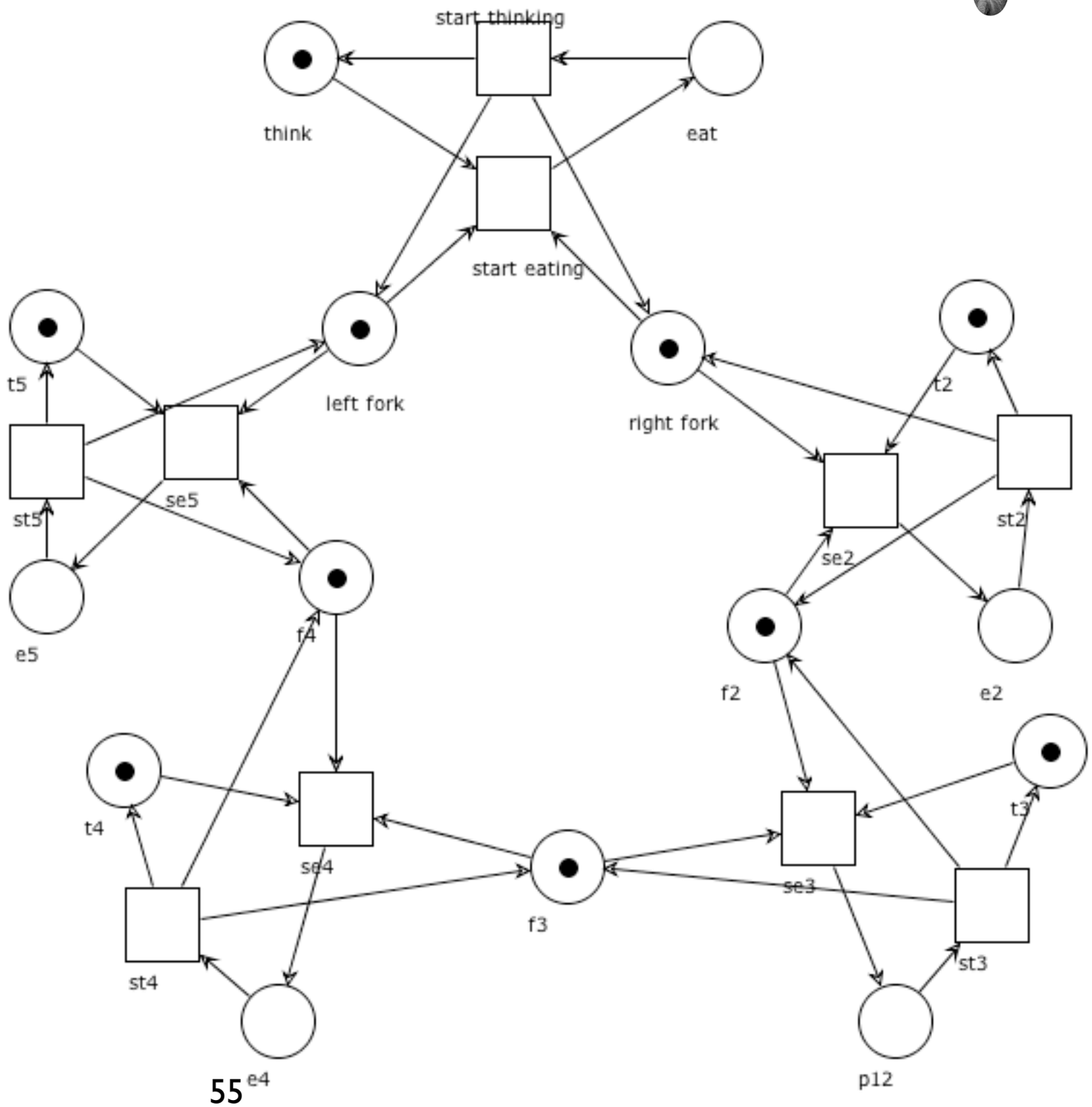


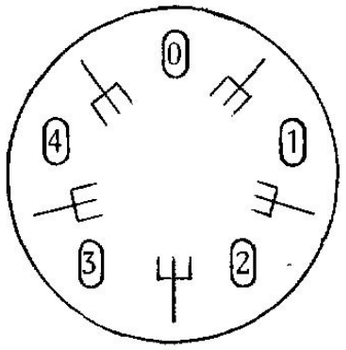
image taken from wikipedia
philosophers clockwise from top:
Plato, Konfuzius, Socrates,
Voltaire and Descartes



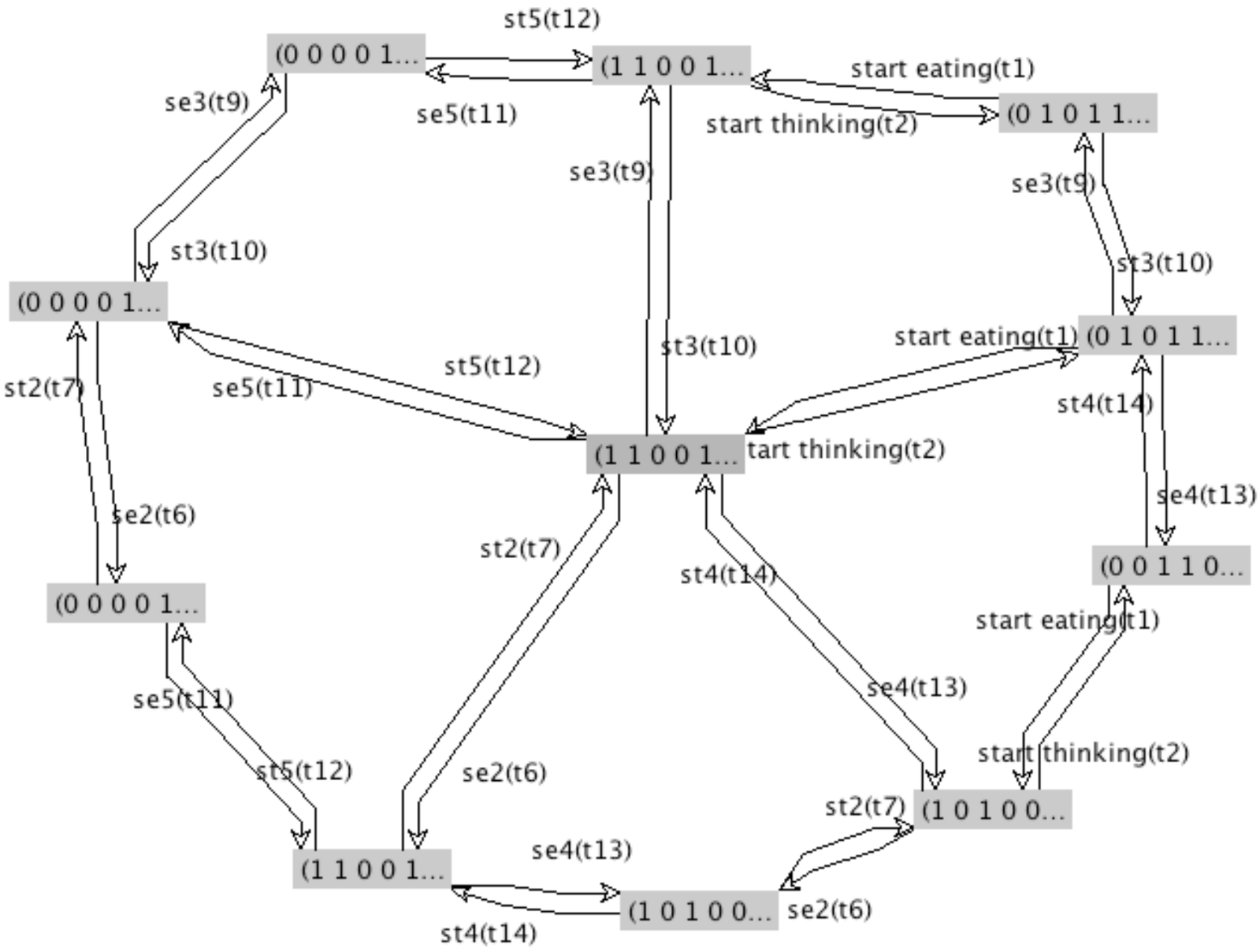
Dining philosophers







Dining philosophers



Exercise:
Railway system

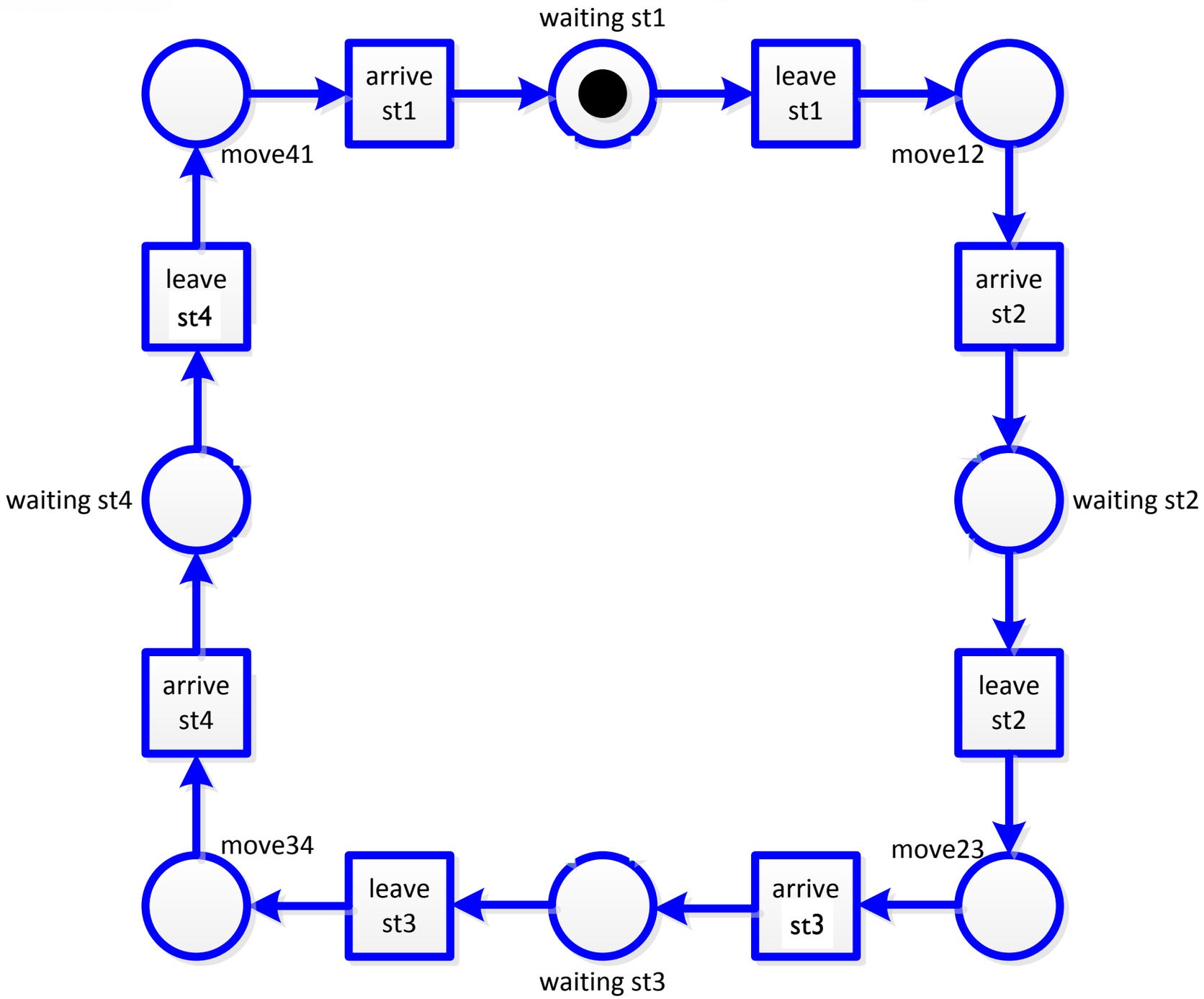
Use a Petri net to model a circular railway system with four stations (st_1, st_2, st_3, st_4) and one train

At each station passengers may "hop on" or "hop off" (this is impossible when the train is moving)

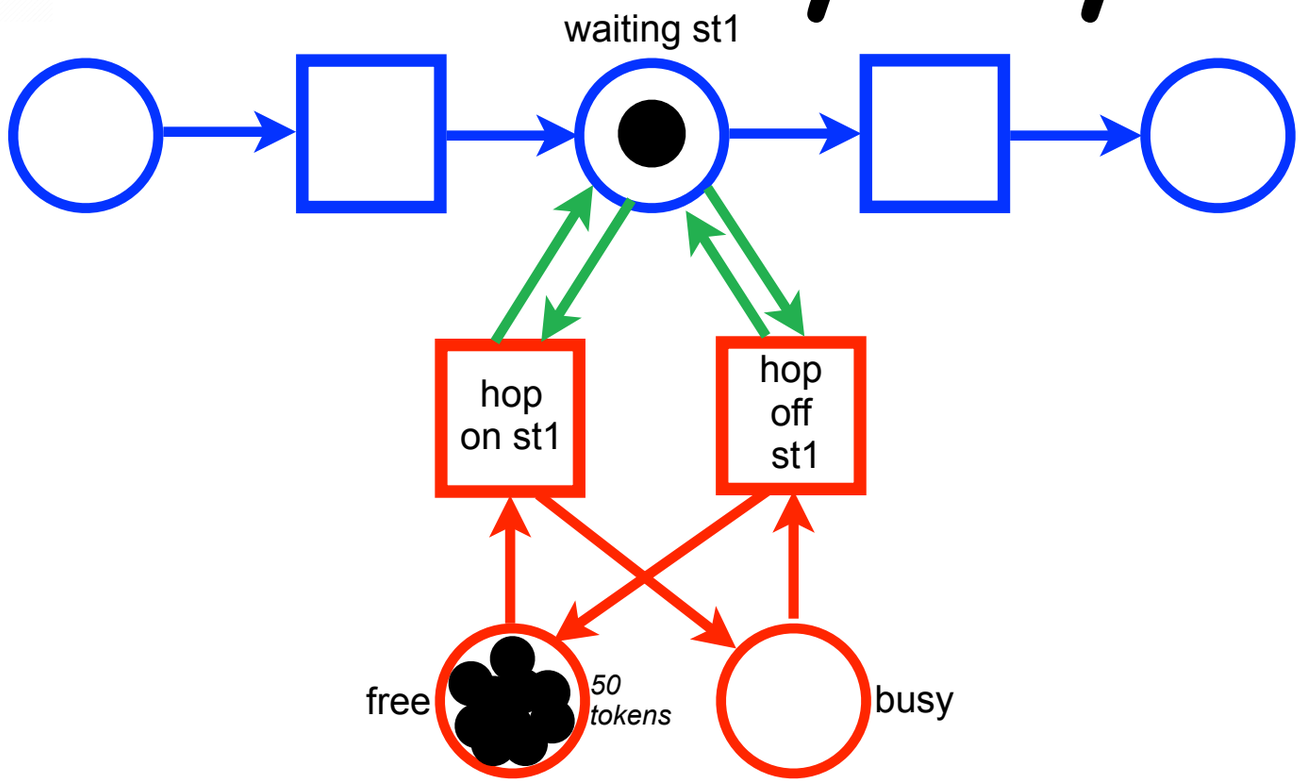
The train has a capacity of 50 persons (if the train is full no passenger can hop on, if the train is empty no passenger can hop off)

What is the number of reachable states?

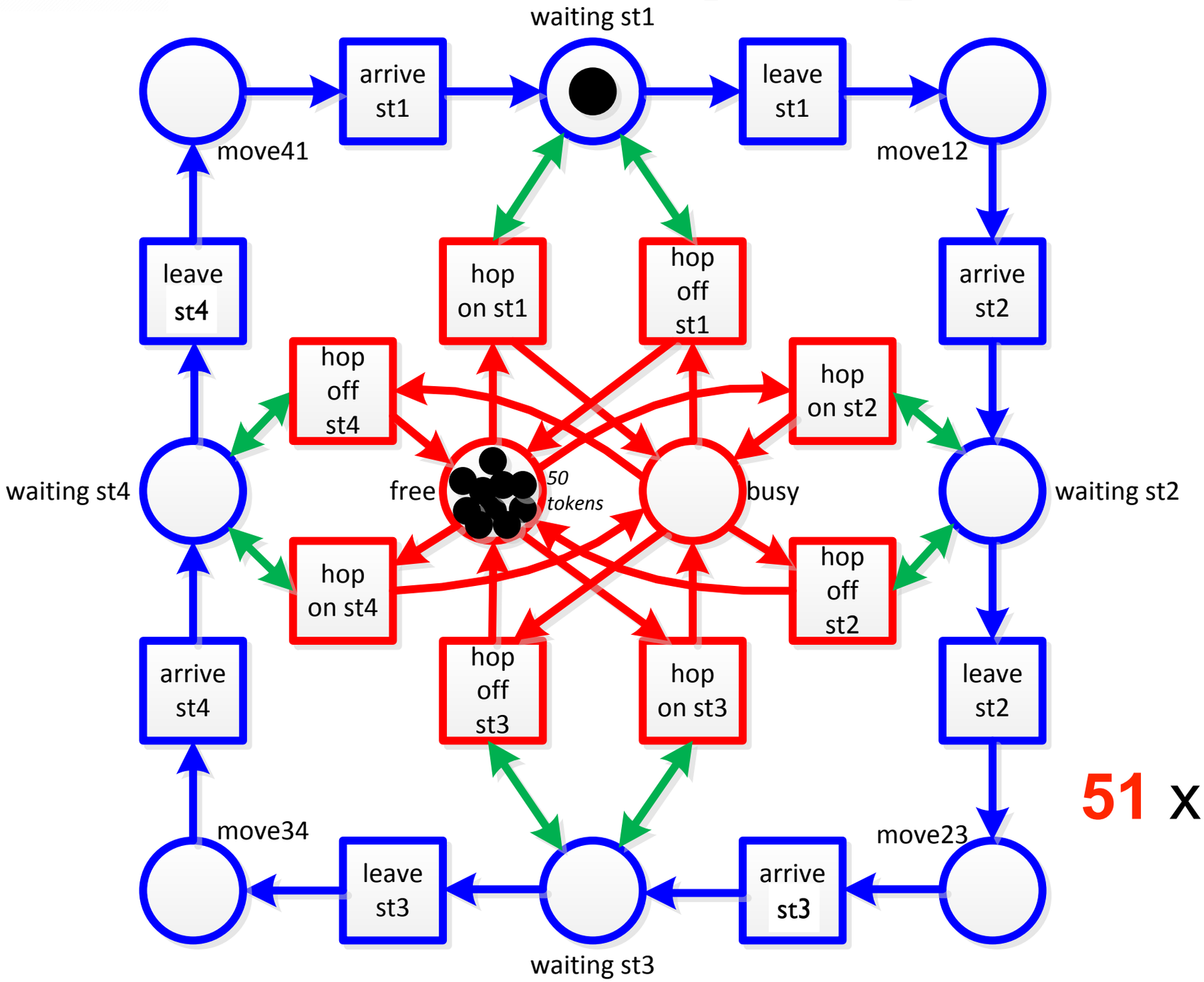
Railway System



Railway System



Railway System



51 x **8** = **408** states

Boundedness

k-Boundedness

Let k be a natural number

A place p is **k-bounded** if no reachable marking has more than k tokens in place p

A net is **k-bounded** if all of its places are k-bounded

In other words, if a net is k-bounded, then k is a capacity constraint that can be imposed over places without any risk of causing “overflow”

Safe nets

A place p is **safe** if it is 1-bounded

A net is **safe** if all of its places are safe

In other words, if the net is safe, then we know that, in any reachable marking, each place contains one token at most

Boundedness

A place p is **bounded** if it is k-bounded for some natural number k

A net is **bounded** if all of its places are bounded

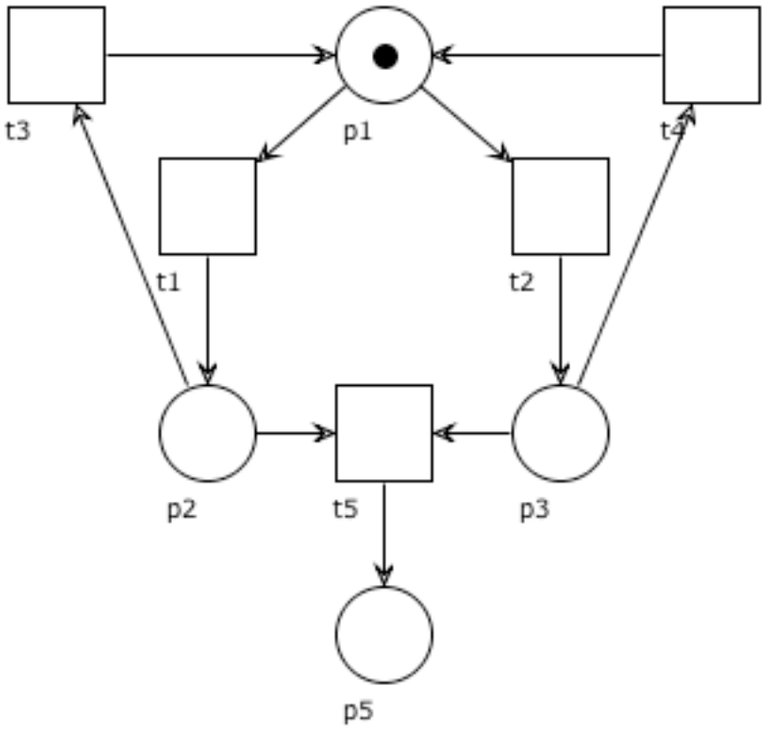
A net is **unbounded** if it is not bounded
(i.e., there is at least one place in which any number of tokens can appear)

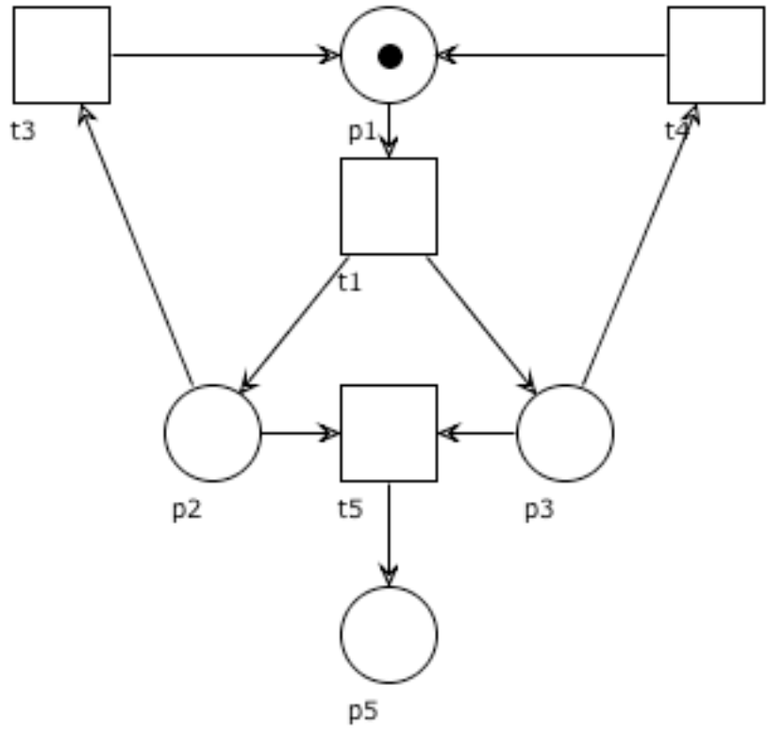
Boundedness, formally

$$(P, T, F, M_0)$$

$$\exists k \in \mathbb{N}, \quad \forall M \in [\, M_0 \,\rangle, \quad \forall p \in P, \quad M(p) \leq k$$

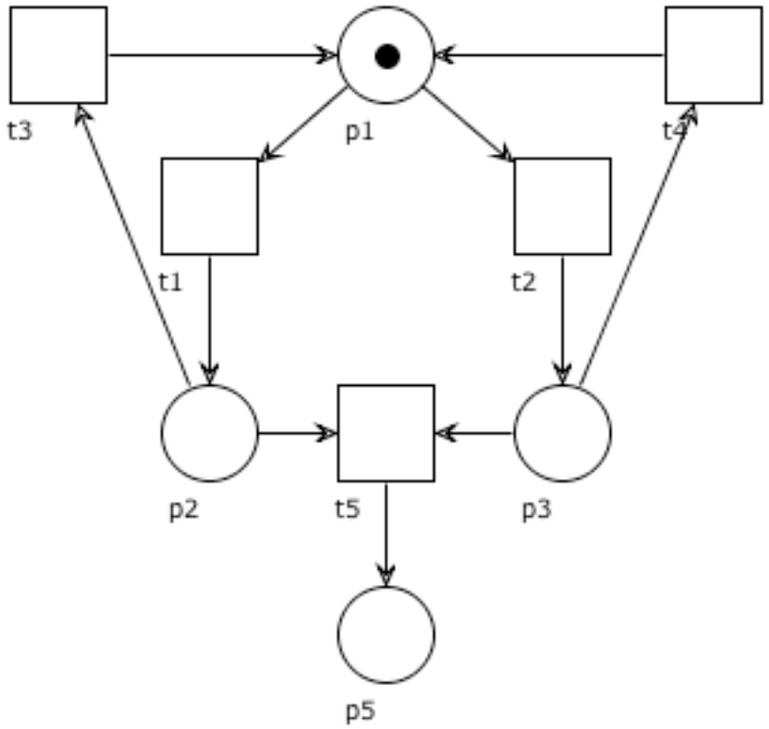
Boundedness: example

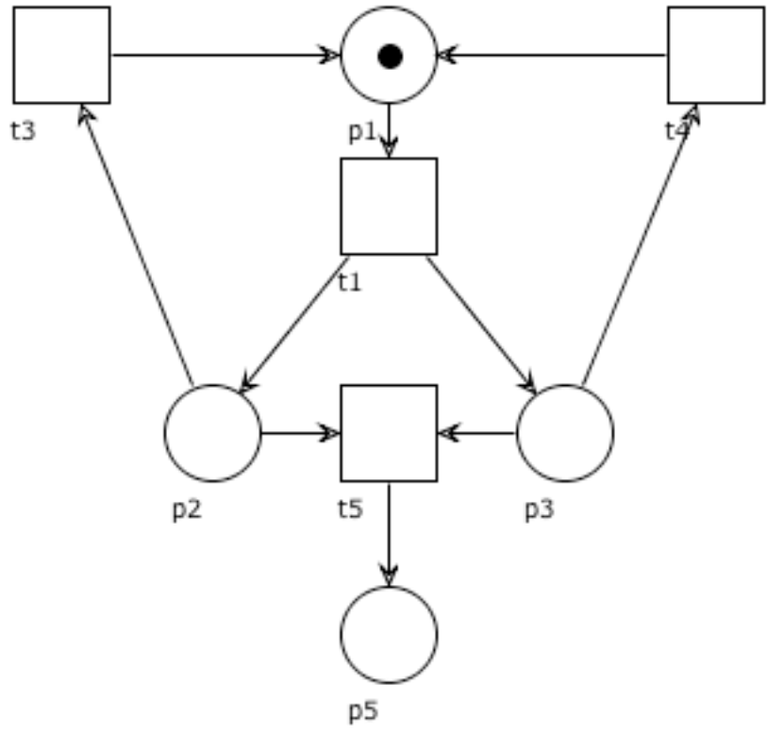




Which places are bounded?
Is the net bounded?
Which places are safe?
Is the net safe?

Boundedness: example





all	Which places are bounded?	none
Yes	Is the net bounded?	No
all	Which places are safe?	none
Yes	Is the net safe?	No

Boundedness and the reachability graph

A system is bounded
iff
its reachability graph is finite

Boundedness implies finiteness

Theorem: If a system is **bounded** then its reachability graph is finite

Proof: if the system is bounded there exists k such that each place contains at most k tokens.

If there are n places it means that there are at most $(k+1)^n$ reachable markings.

Hence the occurrence graph has a finite number of nodes

Finiteness implies boundedness

Theorem: A system is **bounded** if its reachability graph is finite

Proof: for each node M we take k_M be the maximum number of tokens in the same place.

Then we let k be the largest among all k_M

$k = \max \{k_M \mid M \text{ is a node of the graph}\}$

(k exists because the reachability graph is finite)

Clearly the system is k-bounded and thus bounded

Consequence

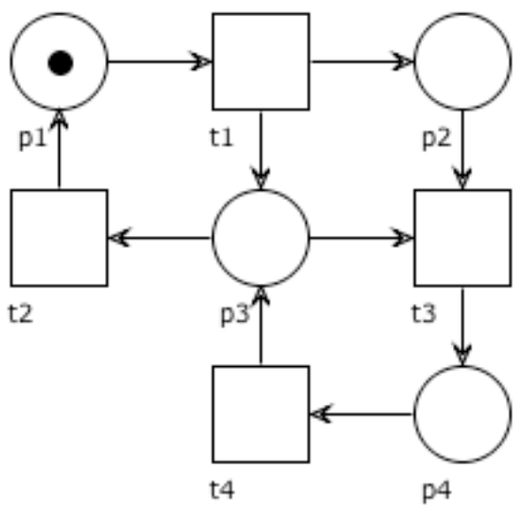
A net is unbounded

if and only if

its reachability graph is not finite

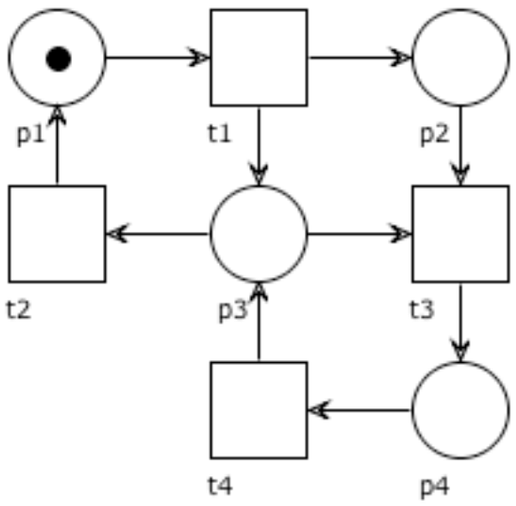
Exercises

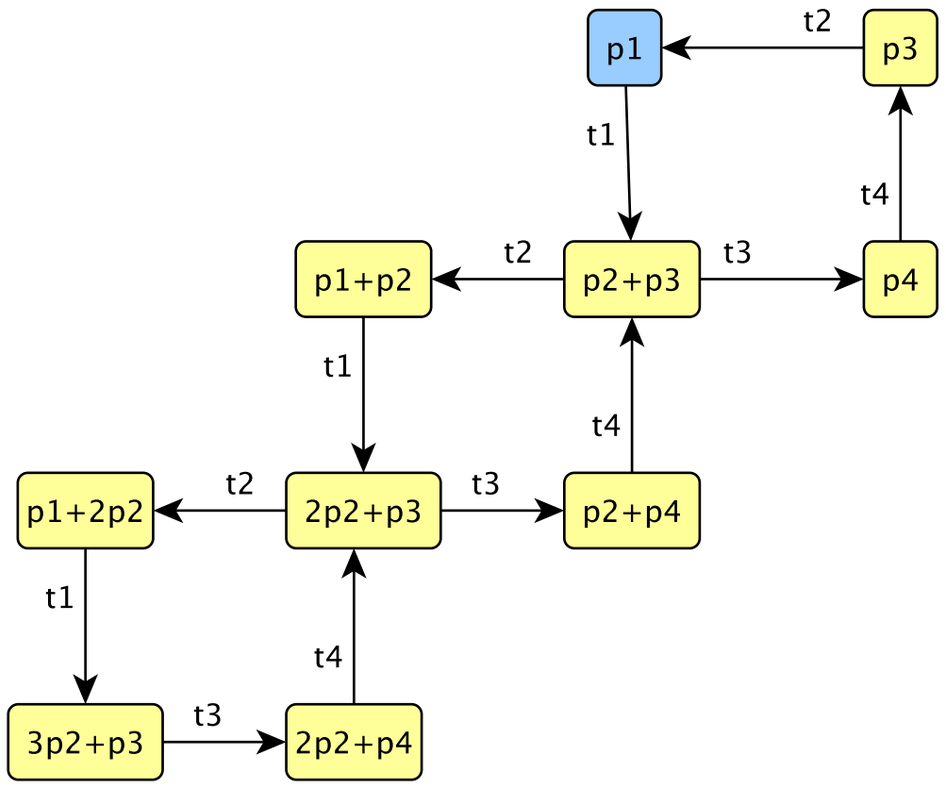
Which places are bounded?
Which ones are safe?
Is the net bounded?



Exercises

Which places are bounded?
Which ones are safe?
Is the net bounded?

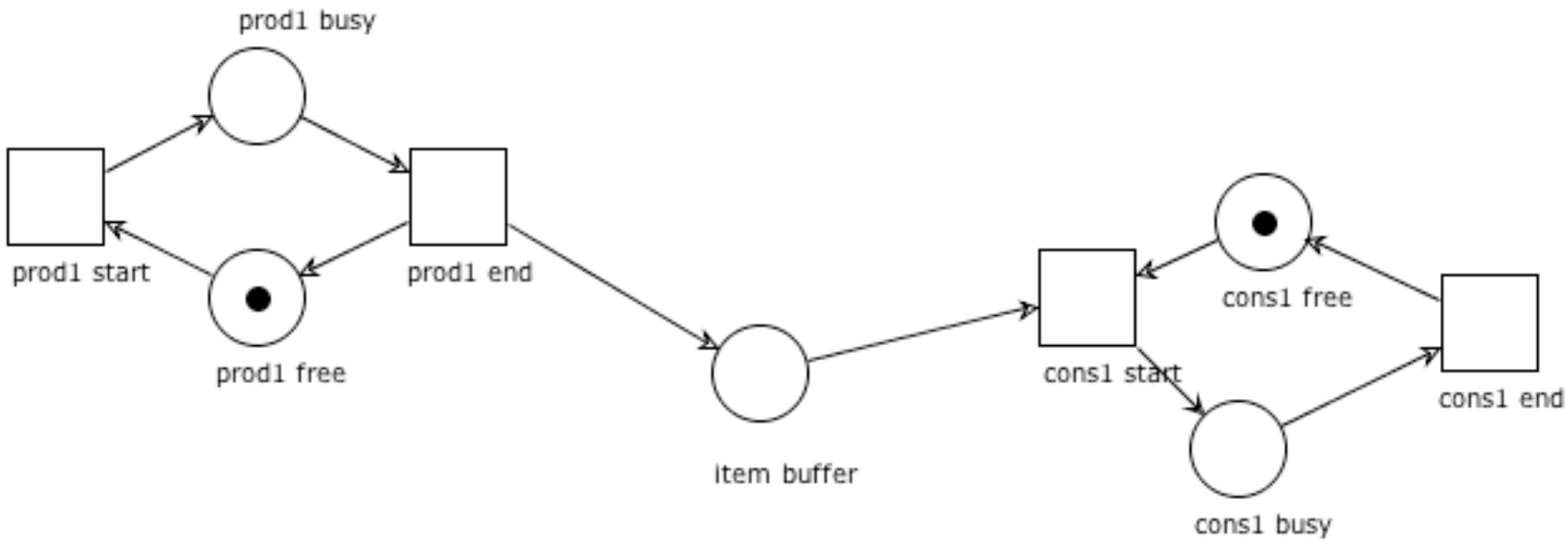




• • •

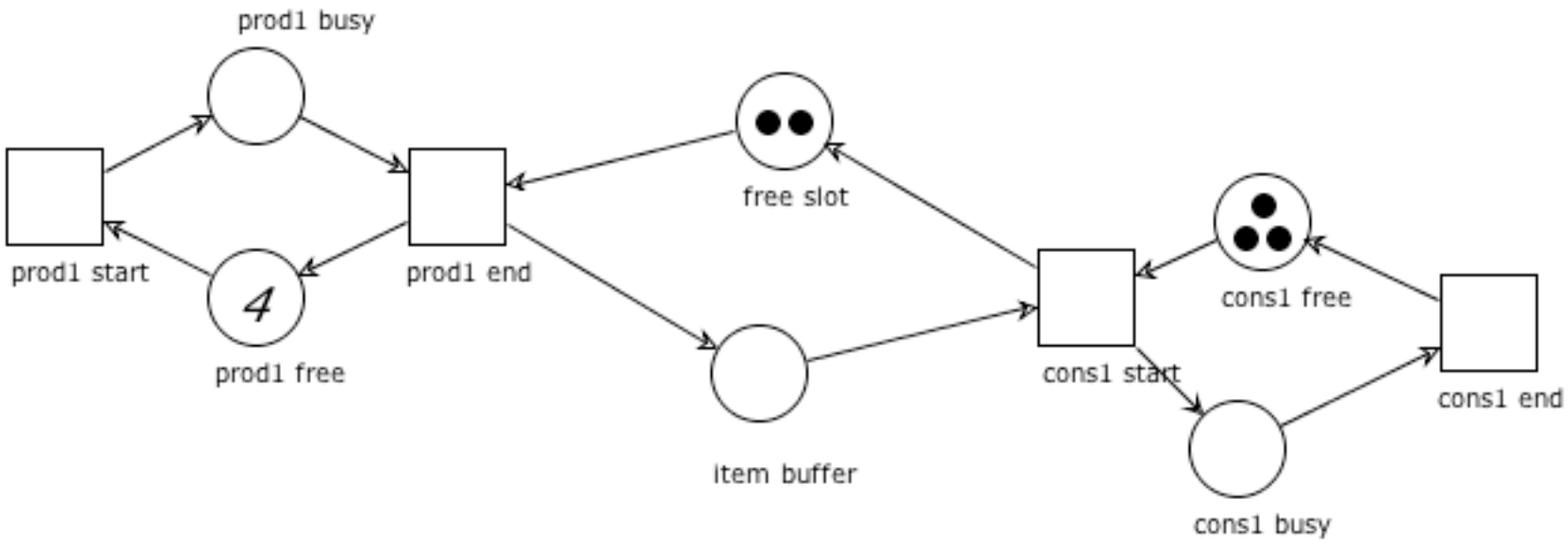
Question time

Which places are bounded?
Which ones are safe?
Is the net bounded?



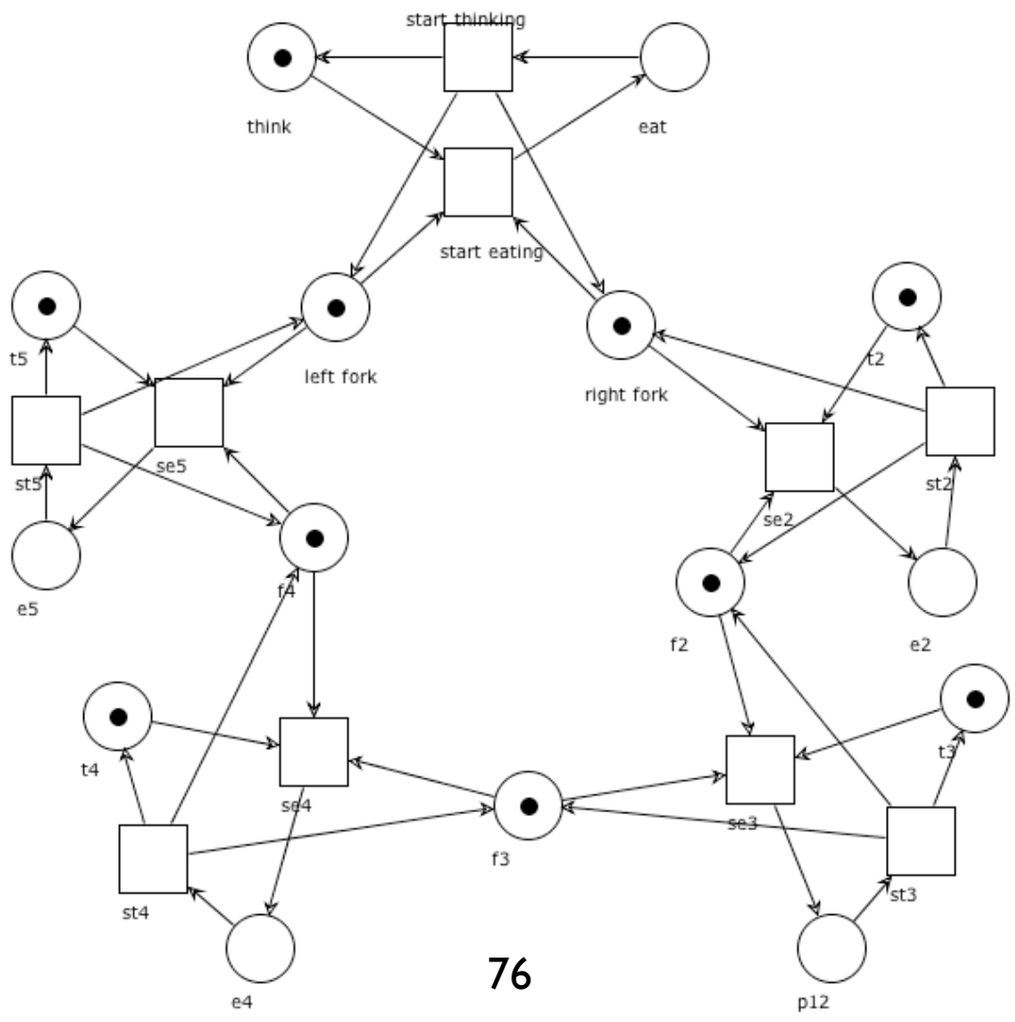
Question time

Which places are bounded?
Which ones are safe?
Is the net bounded?



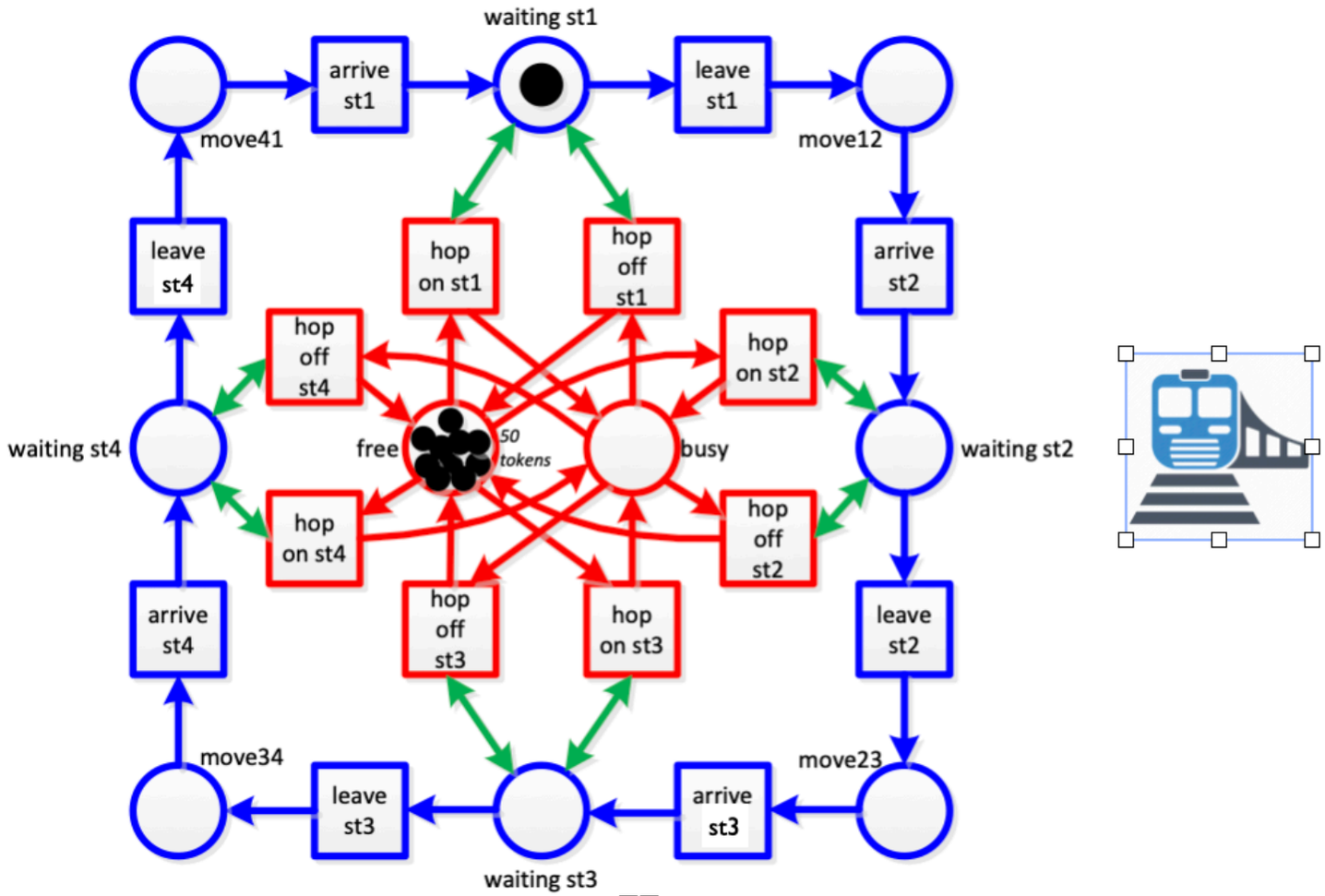
Question time

Which places are bounded?
Which ones are safe?
Is the net bounded?



Question time

Which places are bounded?
Which ones are safe?
Is the net bounded?



Coverability

Coverability graph

A **coverability graph** is a finite over-approximation of the reachability graph

It allows for markings with infinitely many tokens in one place (called extended bags)

$$B : P \longrightarrow \mathbb{N} \cup \{\infty\}$$

Discover unbounded places

Suppose

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \ldots \xrightarrow{t_i} M_i \ldots \xrightarrow{t_j} M_j$$

with $M_i \subset M_j$

Let $M = M_i$ and $M' = M_j$ and $L = M' - M$

By the monotonicity Lemma we have, for any $n \in \mathbb{N}$:
$M \to^* M + L \to^* M + 2L \to^* \ldots \to^* M + nL$

Hence all places p marked by L (i.e. if $L(p) > 0$) are unbounded

Account for unbounded places

Idea:
When computing the RG, if M' is found s.t.

$M_0 \to^* M \to^* M'$ with $M \subset M'$

Add the extended bag B (instead of M') to the graph

$$\text{where} \quad B(p) = \begin{cases} M'(p) & \text{if } M'(p) - M(p) = 0 \\ \infty & \text{otherwise} \end{cases}$$

A few remarks

Idea: mark unbounded places by ∞

Remind: $M \subset M'$ means that $M \subseteq M' \wedge M \neq M'$, i.e.,

1. for any $p \in P$, $M'(p) \geq M(p)$
2. there exists at least one place $q \in P$ such that $M'(q) > M(q)$

Remark:
Requiring $M_0 \rightarrow^* M \rightarrow^* M'$ is different than
requiring $M, M' \in [\, M_0 \,\rangle$

Operations on extended bags

Inclusion: Let $B, B' : P \to \mathbb{N} \cup \{\infty\}$
We write $B \subseteq B'$ if for any p we have
$B'(p) = \infty$ or $B(p), B'(p) \in \mathbb{N} \wedge B(p) \leq B'(p)$

Sum: Let $B, B' : P \to \mathbb{N} \cup \{\infty\}$

$$(B + B')(p) = \begin{cases} \infty & \text{if } B(p) = \infty \text{ or } B'(p) = \infty \\ B(p) + B'(p) & \text{if } B(p), B'(p) \in \mathbb{N} \end{cases}$$

Difference: Let $B : P \to \mathbb{N} \cup \{\infty\}$ and $M : P \to \mathbb{N}$ with $M \subseteq B$

$$(B - M)(p) = \begin{cases} \infty & \text{if } B(p) = \infty \\ B(p) - M(p) & \text{if } B(p) \in \mathbb{N} \end{cases}$$

Operations on extended bags: examples

$$2a + b + \infty c \subseteq 2a + 2b + \infty c \subseteq 2a + \infty b + \infty c$$

$$2a + b + \infty c \not\subseteq a + 2b + \infty c \not\subseteq 2a + \infty b + 3c$$

$$(3a + 2b + \infty c) + (2a + \infty b + \infty c) = (5a + \infty b + \infty c)$$

$$(5a + \infty b + \infty c) - (3a + 2b + \infty c) = ?$$

must be a marking!

$$(5a + \infty b + \infty c) - (3a + 2b + 4c) = (2a + \infty b + \infty c)$$

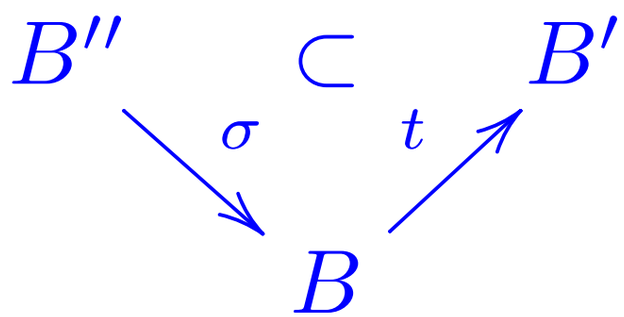
Compute a reachability graph

1. Initially N = { M_0 } and A = ∅
2. Take a bag B ∈ N and a transition t ∈ T such that
   1. B enables t and there is no arc labelled t leaving from B
3. Let B' = B - •t + t•
4. Add B' to N and (B,t,B') to A
5. Repeat steps 2,3,4 until no new arc can be added

(all bags are finite in this case)

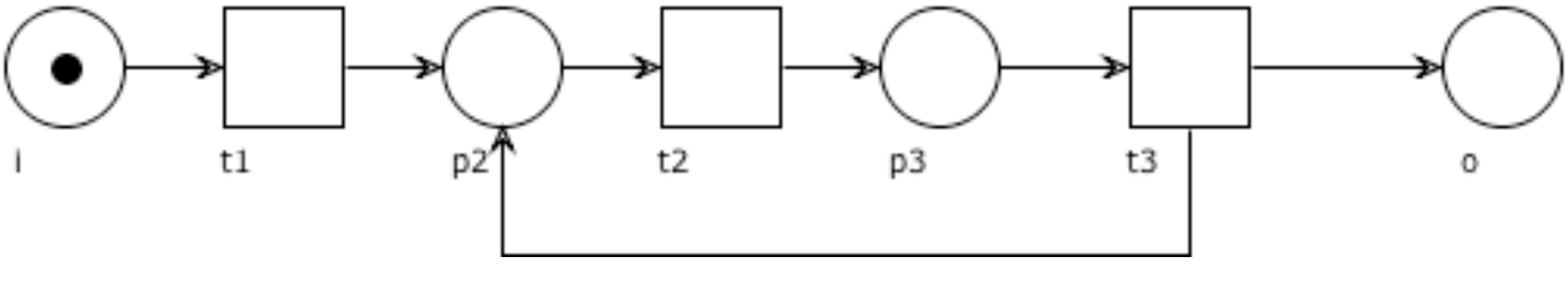
Compute a coverability graph

1. Initially N = { M_0 } and A = ∅
2. Take a bag B ∈ N and a transition t ∈ T such that
   1. B enables t and there is no arc labelled t leaving from B
3. Let B' = B - •t + t•
4. Let B_c' such that for any p ∈ P
   1. $B_c'(p) = \infty$ if there is a node B'' ∈ N such that
      1. there is a directed path from B'' to B in the graph (N,A)
      2. B'' ⊂ B',
      3. B''(p) < B'(p)
   2. $B_c'(p)$ = B'(p) otherwise

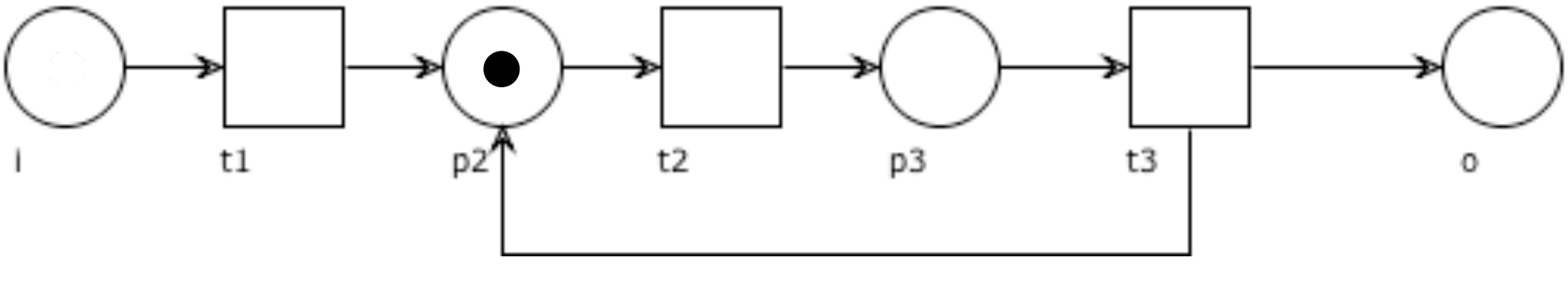


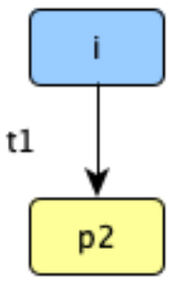
5. Add B_c' to N and (B,t,B_c') to A
6. Repeat steps 2,3,4,5 until no new arc can be added

Example

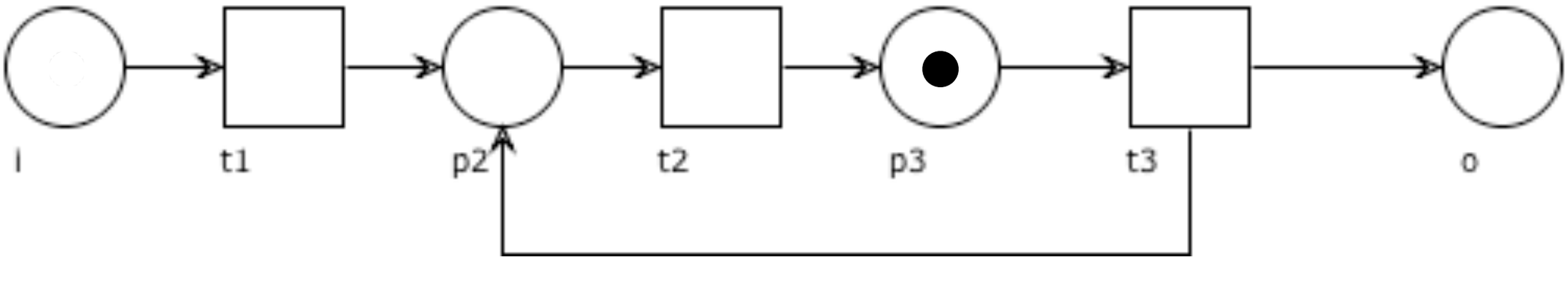


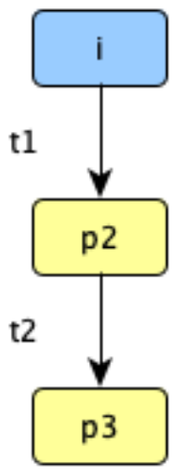
Example



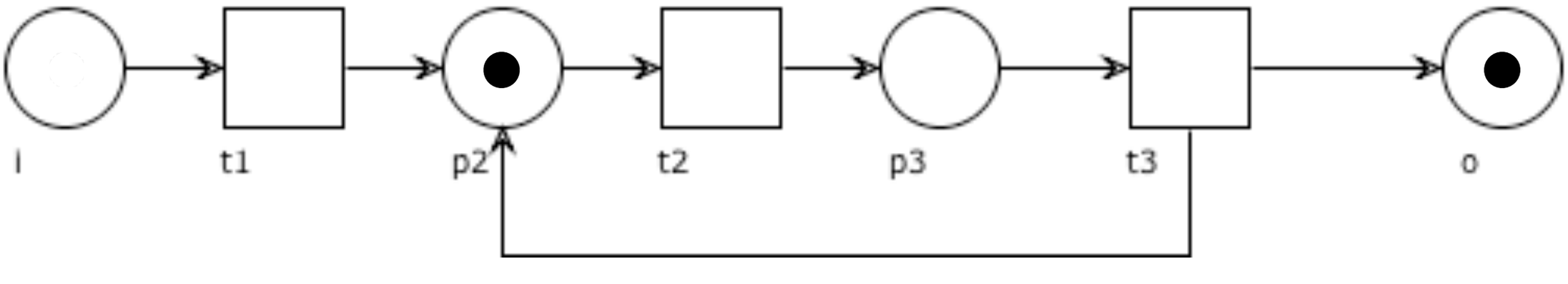


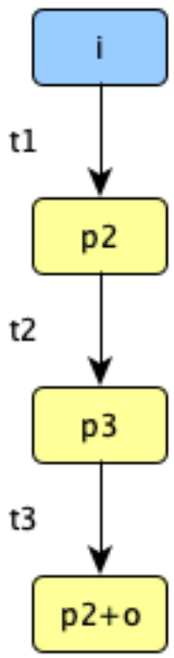
Example



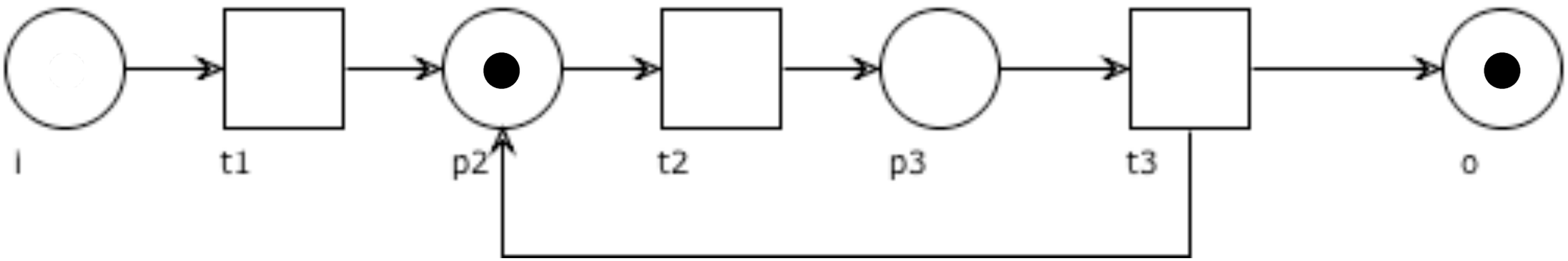


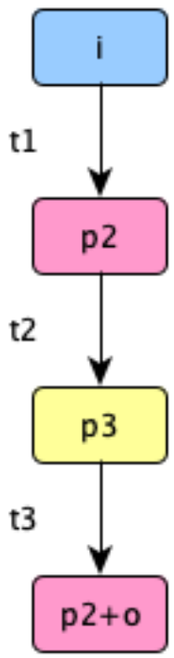
Example



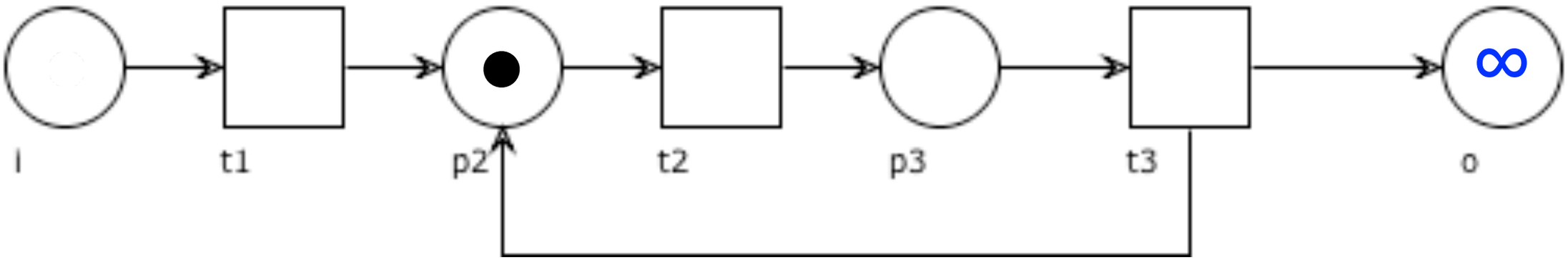


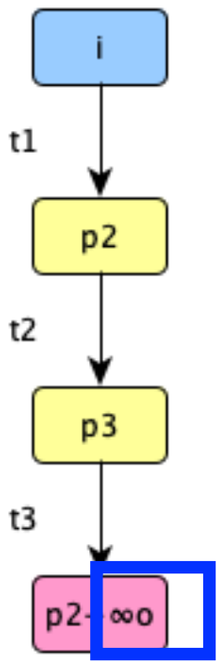
Example



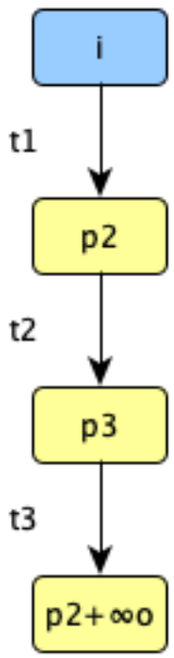


Example

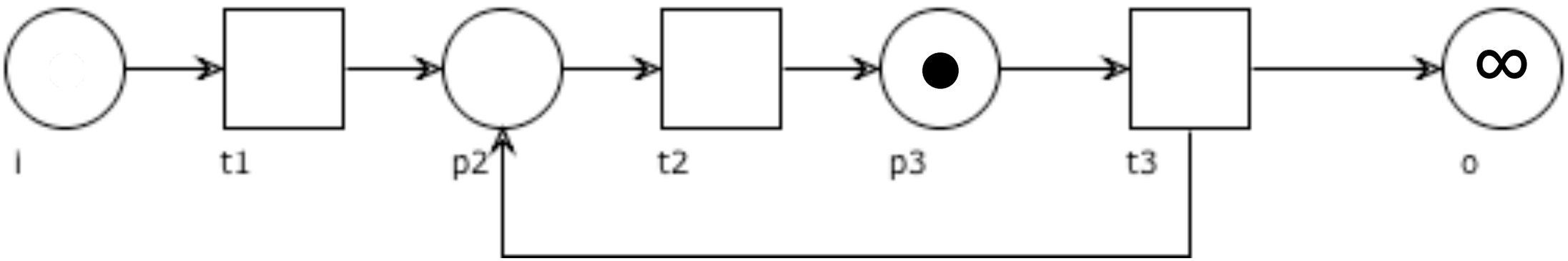


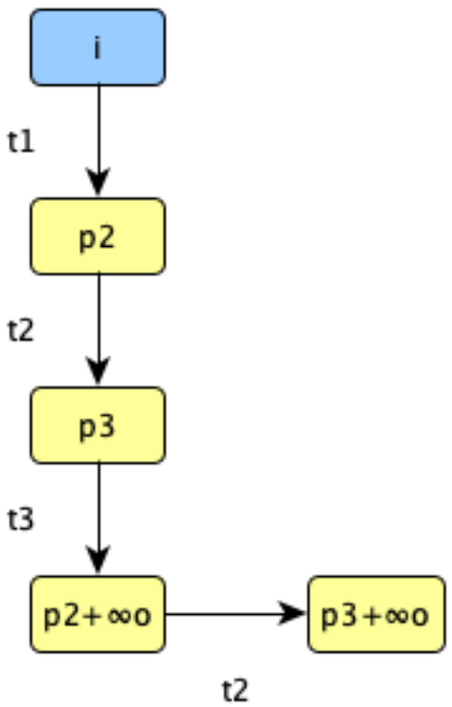


Example

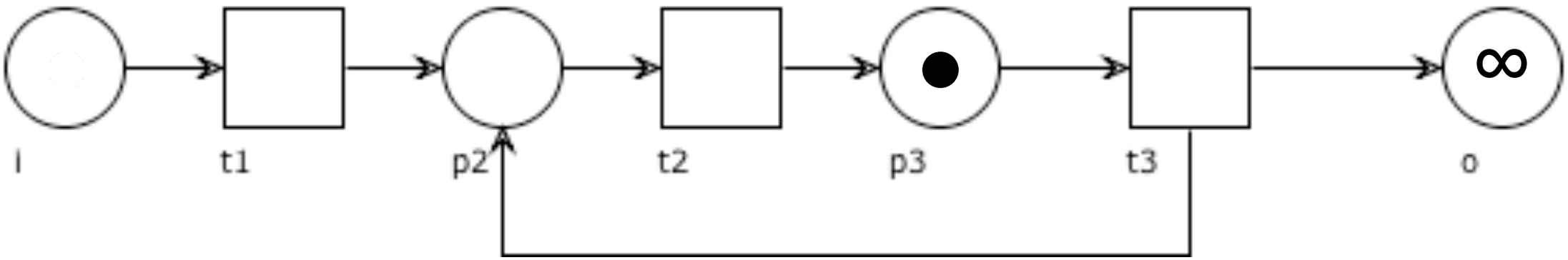


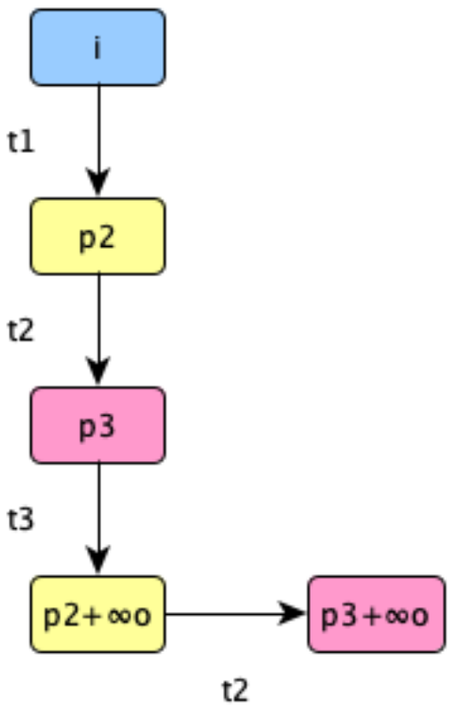
Example



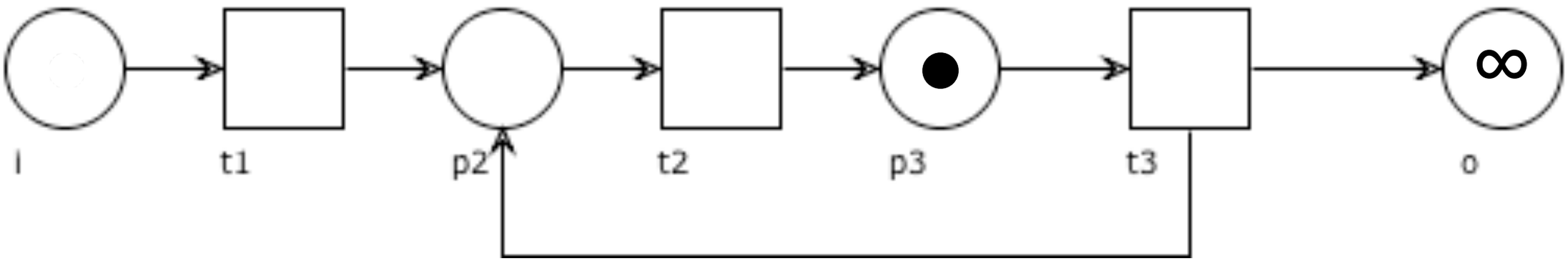


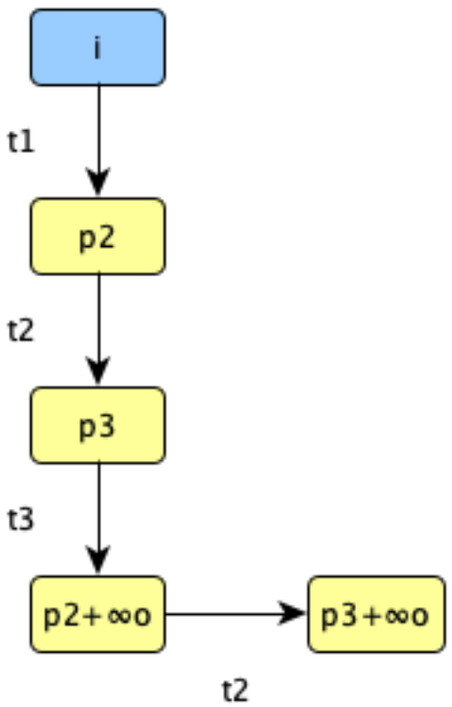
Example



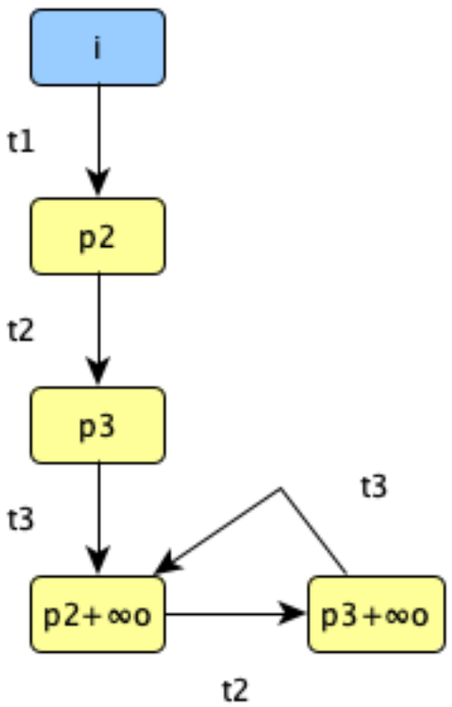


Example





Example



Properties of coverability graphs

A coverability graph is always finite,
but in general it is not uniquely defined
(it depends on which B and t are selected at step 2)

Every firing sequence has a corresponding path in the CG
(the converse is not necessarily true)

Any path in a CG that visits only finite markings
corresponds to a firing sequence

If the RG is finite, then it coincides with the CG

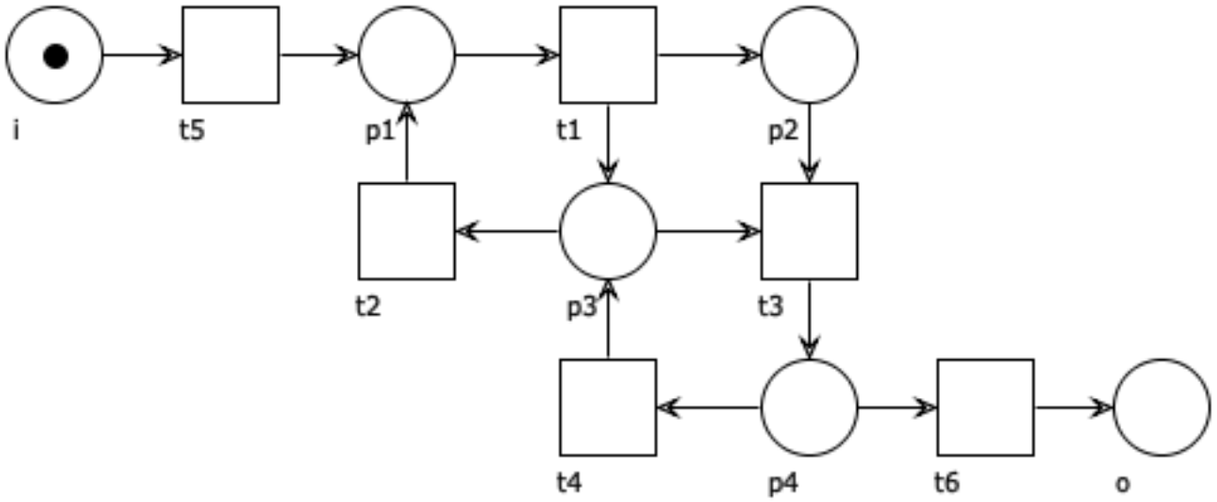
Reachability analysis by coverability

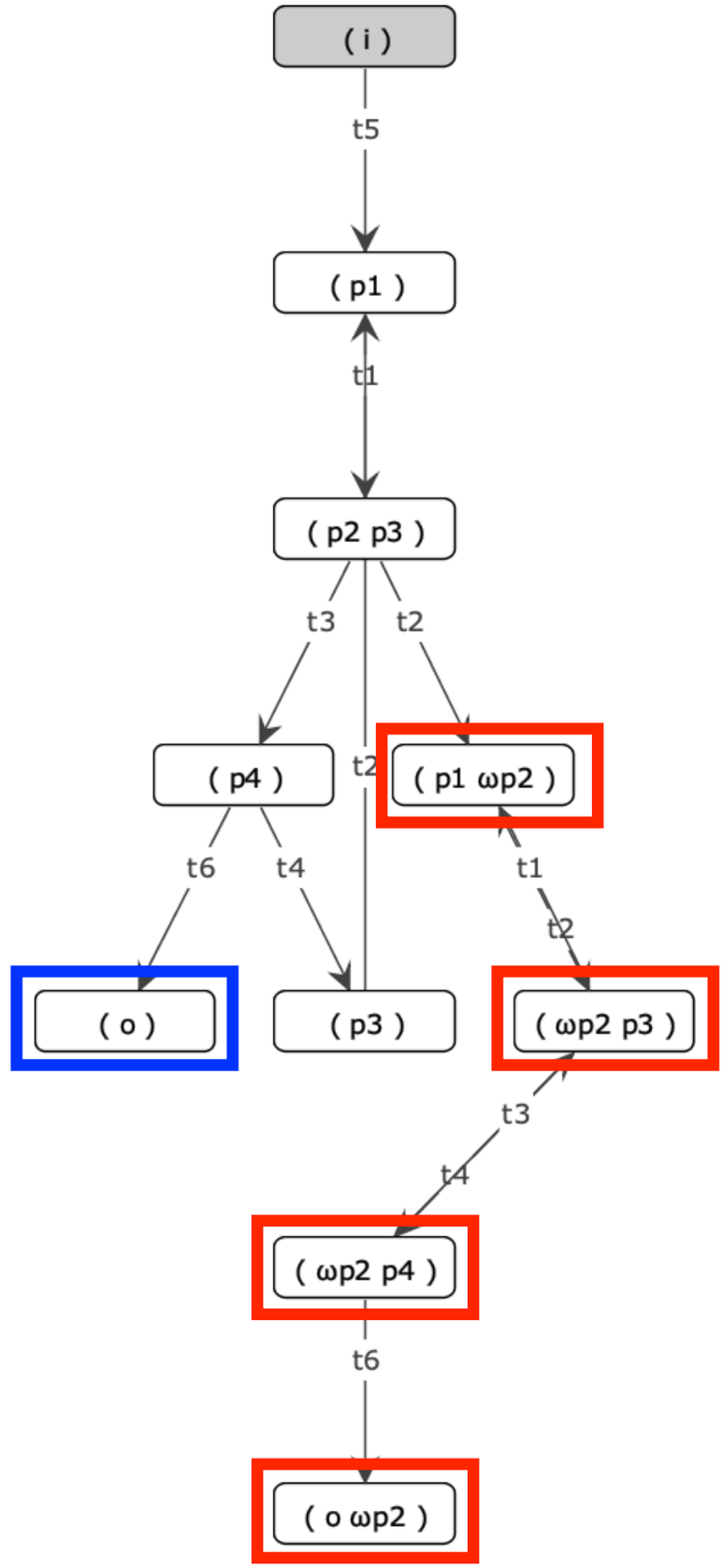
All possible behaviours of a workflow net are represented exactly in the Reachability Graph (if finite)

We use Coverability Graph when necessary (RG not finite)

WoPeD computes a Coverability Graph

Example


i
t5
p1
t1
p2
t2
p3
t3
t4
p4
t6
o


(i)
t5
(p1)
t1
(p2 p3)
t3
t2
(p4)
t2
(p1 ωp2)
t6
t4
t1
t2
(o)
(p3)
(ωp2 p3)
t3
t4
(ωp2 p4)
t6
(o ωp2)