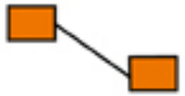
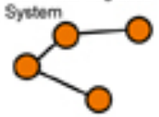
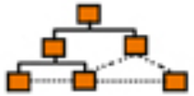
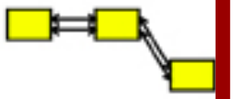
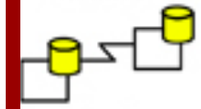
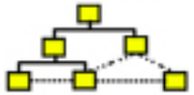
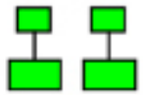
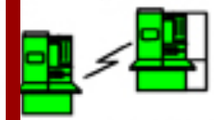
02291 System Integration

Behavioral Models with Petri Nets

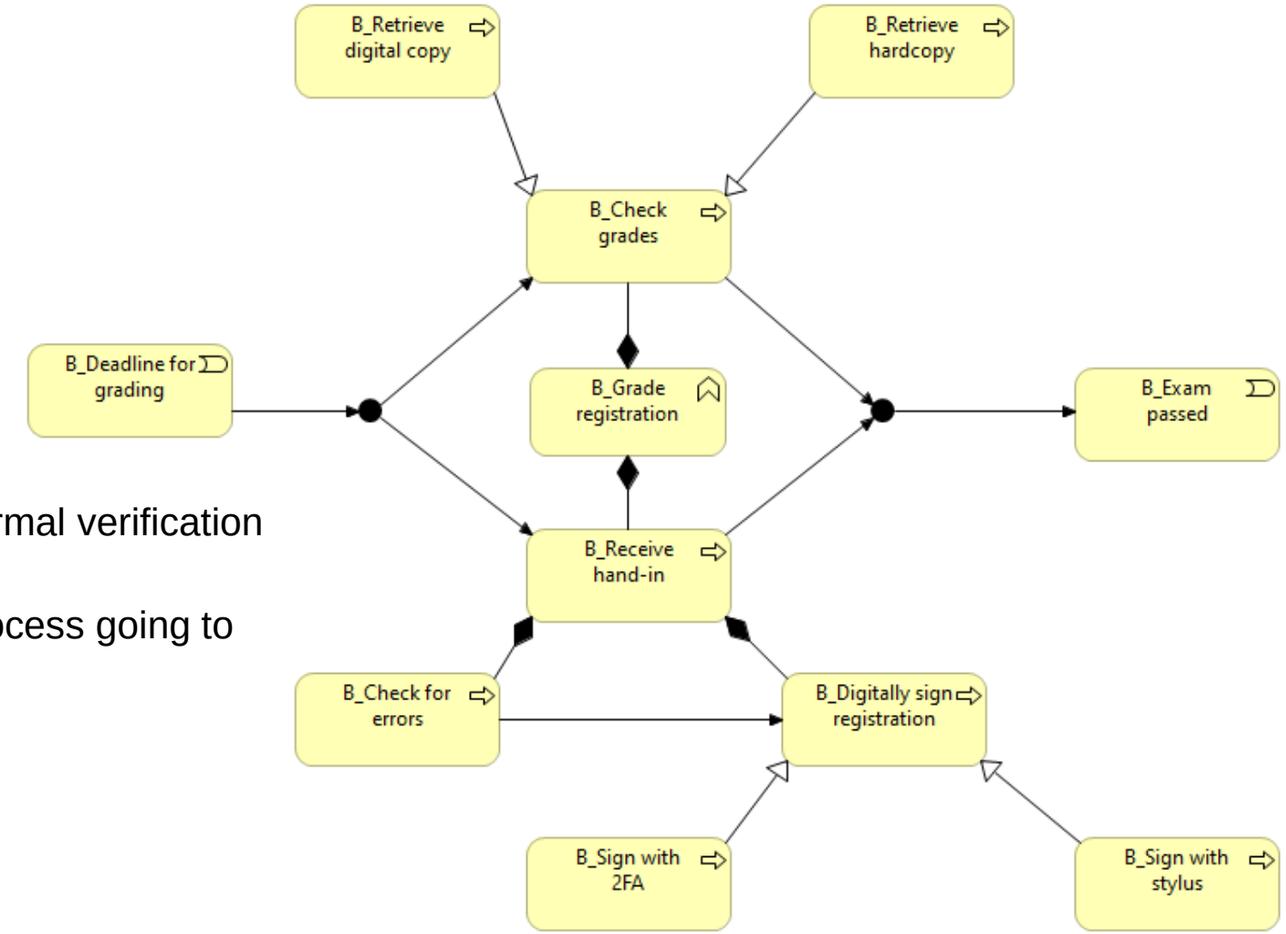
© Giovanni Meroni

Slides based on previous versions by Prof. Pierluigi Plebani (Politecnico di Milano)

	DATA *What*	FUNCTION *How*	NETWORK *Where*	PEOPLE *Who*	TIME *When*	MOTIVATION *Why*	
SCOPE (CONTEXTUAL) *Planner*	List of Things Important to the Business  ENTITY = Class of Business Thing	List of Processes the Business Performs  Process = Class of Business Process	List of Locations in which the Business Operates  Node = Major Business Location	List of Organizations Important to the Business  People = Major Organization Unit	List of Events/Cycles Significant to the Business  Time = Major Business Event/Cycle	List of Business Goals/Stratgies  Ends/Means = Major Business Goal/Strategy	SCOPE (CONTEXTUAL) *Planner*
BUSINESS MODEL (CONCEPTUAL) *Owner*	e.g. Semantic Model  Ent = Business Entity Reln = Business Relationship	e.g. Business Process Model  Proc. = Business Process I/O = Business Resources	Business Logistics System  Node = Business Location Link = Business Linkage	e.g. Work Flow Model  People = Organization Unit Work = Work Product	e.g. Master Schedule  Time = Business Event Cycle = Business Cycle	e.g. Business Plan  End = Business Objective Means = Business Strategy	BUSINESS MODEL (CONCEPTUAL) *Owner*
SYSTEM MODEL (LOGICAL) *Designer*	e.g. Logical Data Model  Ent = Data Entity Reln = Data Relationship	e.g. Application Architecture  Proc. = Application Function I/O = User Views	Distributed System Architecture  Node = I/S Function (Processor, Storage, etc) Link = Line Characteristics	e.g. Human Interface Architecture  People = Role Work = Deliverable	e.g. Processing Structure  Time = System Event Cycle = Processing Cycle	e.g., Business Rule Model  End = Structural Assertion Means = Action Assertion	SYSTEM MODEL (LOGICAL) *Designer*
TECHNOLOGY MODEL (PHYSICAL) *Builder*	e.g. Physical Data Model  Ent = Segment/Table/etc. Reln = Pointer/Key/etc.	e.g. System Design  Proc. = Computer Function I/O = Data Elements/Sets	Technology Architecture  Node = Hardware/Systems Software Link = Line Specifications	e.g. Presentation Architecture  People = User Work = Screen Format	e.g. Control Structure  Time = Execute Cycle = Component Cycle	e.g. Rule Design  End = Condition Means = Action	TECHNOLOGY MODEL (PHYSICAL) *Builder*
DETAILED REPRESEN-TATIONS (OUT-OF-CONTEXT) *Sub-Contractor*	e.g. Data Definition  Ent = Field Reln = Address	e.g. Program  Proc. = Language Statement I/O = Control Block	e.g. Network Architecture  Node = Address Link = Protocol	e.g. Security Architecture  People = Identity Work = Job	e.g. Timing Definition  Time = Interrupt Cycle = Machine Cycle	e.g. Rule Specification  End = Sub-condition Means = Step	DETAILED REPRESEN-TATIONS (OUT-OF CONTEXT) *Sub-Contractor*
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

	DATA *What*	FUNCTION *How*	NETWORK *Where*	PEOPLE *Who*	TIME *When*	MOTIVATION *Why*	
SCOPE (CONTEXTUAL) *Planner*		Behavior			List of Events/Cycles Significant to the Business  Time = Major Business Event/Cycle	List of Business Goals/Stratgies  Ends/Means = Major Business Goal/Strategy	SCOPE (CONTEXTUAL) *Planner*
BUSINESS MODEL (CONCEPTUAL) *Owner*				**Business**	e.g. Master Schedule  Time = Business Event Cycle = Business Cycle	e.g. Business Plan  End = Business Objective Means = Business Strategy	BUSINESS MODEL (CONCEPTUAL) *Owner*
SYSTEM MODEL (LOGICAL) *Designer*				**Application**	e.g. Processing Structure  Time = System Event Cycle = Processing Cycle	e.g., Business Rule Model  End = Structural Assertion Means =Action Assertion	SYSTEM MODEL (LOGICAL) *Designer*
TECHNOLOGY MODEL (PHYSICAL) *Builder*				**Technology**	e.g. Control Structure  Time = Execute Cycle = Component Cycle	e.g. Rule Design  End = Condition Means = Action	TECHNOLOGY MODEL (PHYSICAL) *Builder*
DETAILED REPRESEN-TATIONS (OUT-OF-CONTEXT) *Sub-Contractor*	e.g. Data Definition  Ent = Field Reln = Address	e.g. Program  Proc = Language Statement I/O = Control Block	e.g. Network Architecture  Node = Address Link = Protocol	e.g. Security Architecture  People = Identity Work = Job	e.g. Timing Definition  Time = Interrupt Cycle = Machine Cycle	e.g. Rule Specification  End = Sub-condition Means = Step	DETAILED REPRESEN-TATIONS (OUT-OF CONTEXT) *Sub-Contractor*
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

Behavioral models in ArchiMate



ArchiMate does not provide formal verification techniques:

- Is the grade registration process going to terminate?
- Can all tasks be executed?

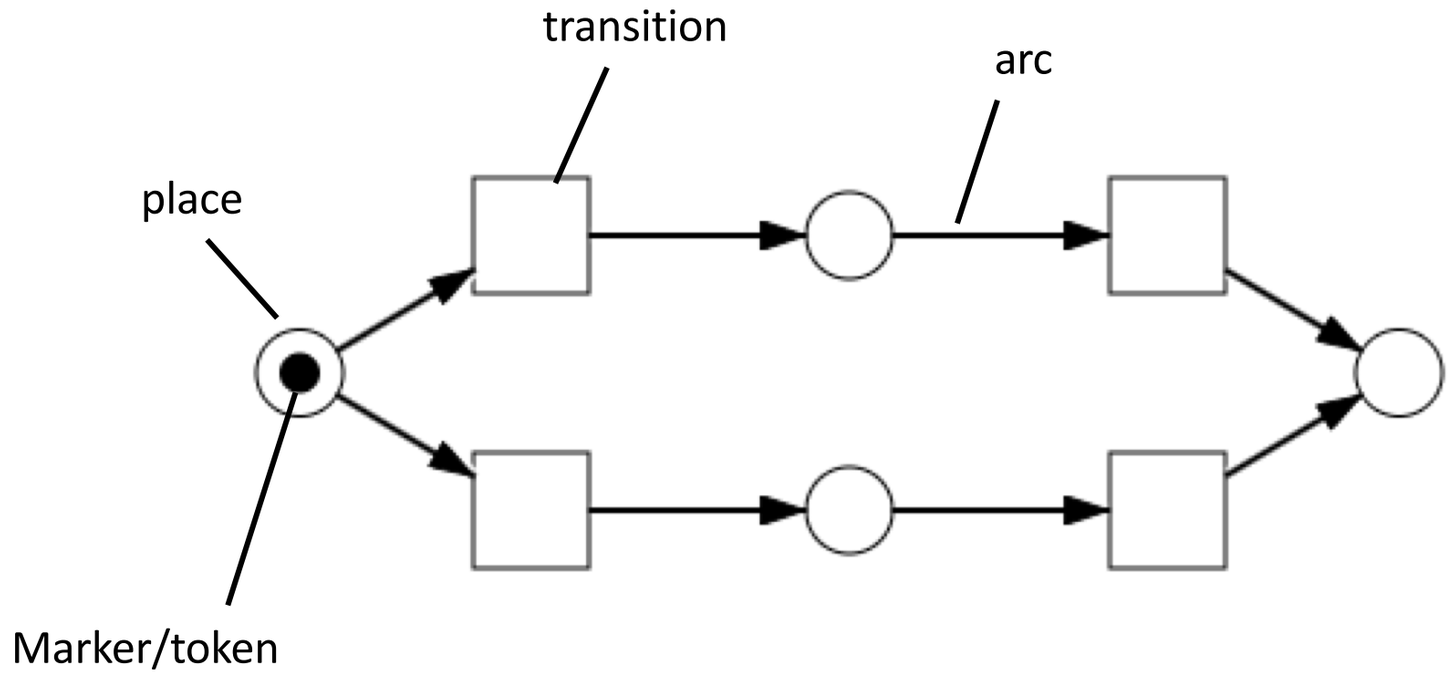
Petri Nets

Petri Nets (a very short introduction)

- Developed in the 60s by Carl Adam Petri
- Allows the modeling of concurrent, distributed, asynchronous systems
- Many variants exist to model additional aspects of a system
  - Temporal PN
  - Colored PN
  - Stochastic PN
  - …

- Here we introduce, intuitively and formally, the basic elements
- For an exhaustive definition please refer to one of the thousands books available. We suggest:
  - Van der Aalst, Modeling Business Process: A Petri-Net Approach, MIT press, 2011

Petri Nets: an informal introduction



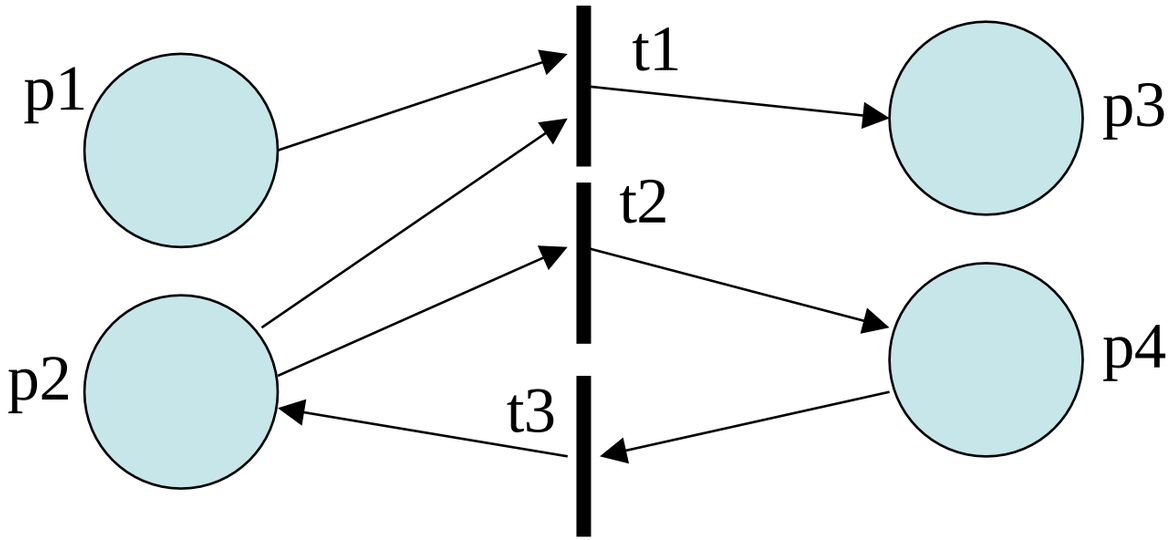
- Bipartite oriented graph connecting places and transitions
- Transitions and places define the structure of the net
- Tokens define the behaviour of the net

Petri Nets: a formal definition of the structure

- A Petri Net **N** is a triple (**P**, **T**, **F**), where
  - P is a finite set of places
  - T is a finite set of transitions
  - F $\subset$ (P x T $\cup$ T x P) is a flow relation

- Given a Petri Net N = (P, T, F)
  - *Pre-sets* can be defined for places and transitions
    - Preset for a place *p* is defined as $\bullet p = \{t \in T \mid (t, p) \in F\}$
    - Preset for a transition *t* is defined as $\bullet t = \{p \in P \mid (p, t) \in F\}$
  - Similarly, places and transitions *post-sets* are defined
    - Post-set for a place *p* is defined as $p \bullet = \{t \in T \mid (p, t) \in F\}$
    - Post-set for a transition *t* is defined as $t \bullet = \{p \in P \mid (t, p) \in F\}$

Example



P = ?

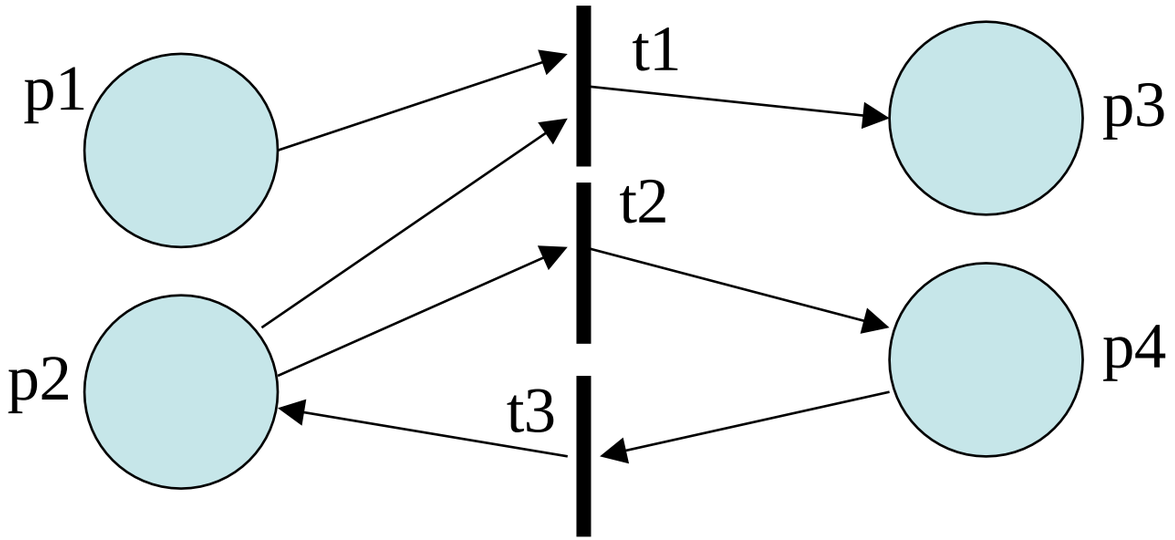
T = ?

F = ?

$t_1 \bullet$ = ?; $\bullet t_1$ = ?; $\bullet p_2$ = ?; $\bullet p_1$ = ? ; $p_2 \bullet$ = ?

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example



$P = \{p_1, p_2, p_3, p_4\}$

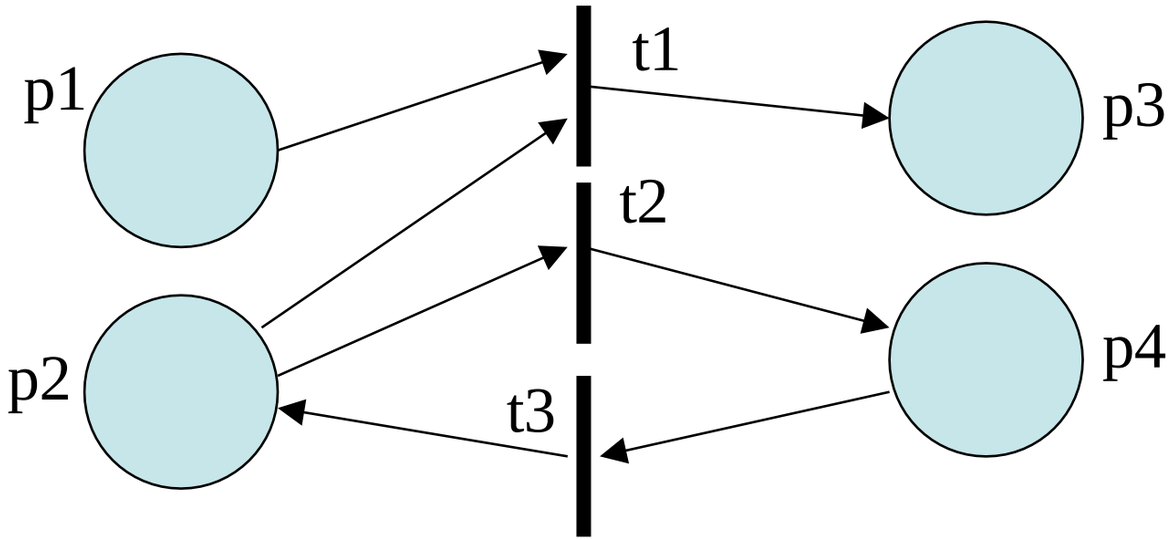
$T = ?$

$F = ?$

$t_1 \bullet = ?; \bullet t_1 = ?; \bullet p_2 = ?; \bullet p_1 = ?; \ p_2 \bullet = ?$

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example



$P = \{p_1, p_2, p_3, p_4\}$

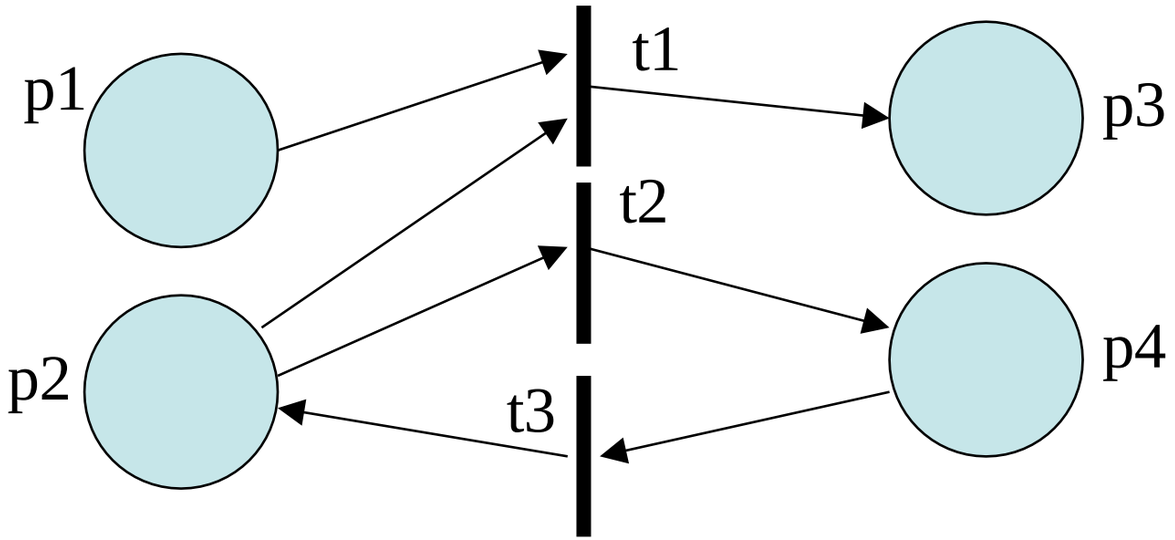
$T = \{t_1, t_2, t_3\}$

$F = ?$

$t_1 \bullet = ?;\ \bullet t_1 = ?;\ \bullet p_2 = ?;\ \bullet p_1 = ?;\ p_2 \bullet = ?$

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example



$P = \{p_1, p_2, p_3, p_4\}$

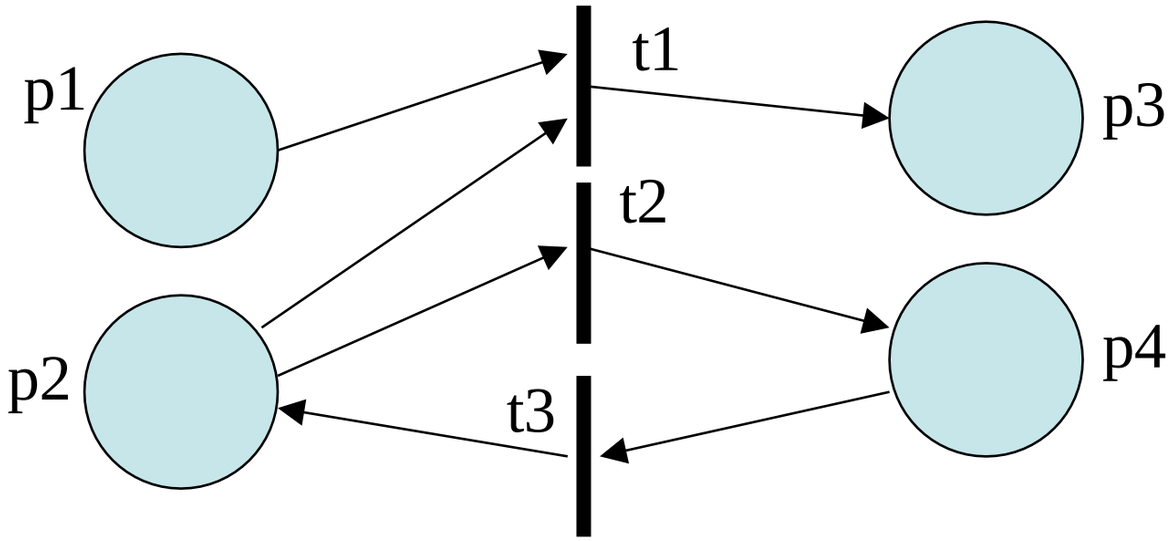
$T = \{t_1, t_2, t_3\}$

$F = \{(p_1, t_1), (p_2, t_1), (t_1, p_3), (p_2, t_2), (t_2, p_4), (p_4, t_3), (t_3, p_2)\}$

$t_1 \bullet = ?; \bullet t_1 = ?; \bullet p_2 = ?; \bullet p_1 = ?; \ p_2 \bullet = ?$

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example



$P = \{p_1, p_2, p_3, p_4\}$

$T = \{t_1, t_2, t_3\}$

$F = \{(p_1, t_1), (p_2, t_1), (t_1, p_3), (p_2, t_2), (t_2, p_4), (p_4, t_3), (t_3, p_2)\}$

$t_1 \bullet = \{p_3\}; \bullet t_1 = \{p_1, p_2\}; \bullet p_2 = \{t_3\}; \bullet p_1 = \varnothing;\ p_2 \bullet = \{t_1, t_2\}$

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

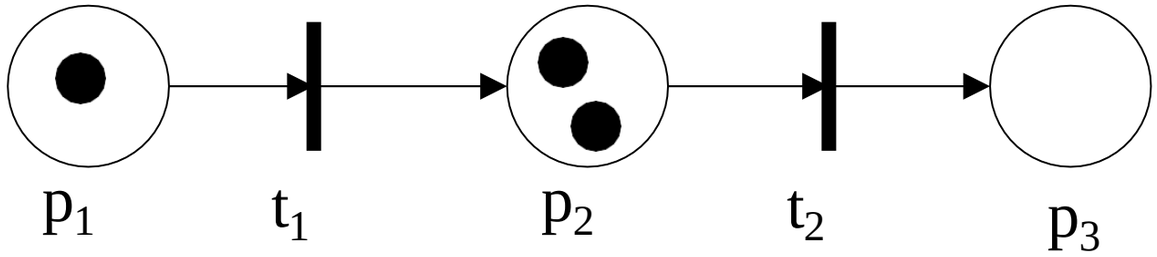
Petri Nets: a formal definition of the behaviour

- Marking is the key element describing the behaviour of a Petri Net
- Marking is represented by tokens and determines a state of the Petri Net

- Given a Petri Net N = (P, T, F)
  - m: P $\rightarrow$ N
  - Marking assigns to each place p $\in$ P a number of tokens m(p)
  - The set M = {$\forall$p $\in$ P | m(p) }

- For a Petri net an initial marking M_0 needs to be specified

Example

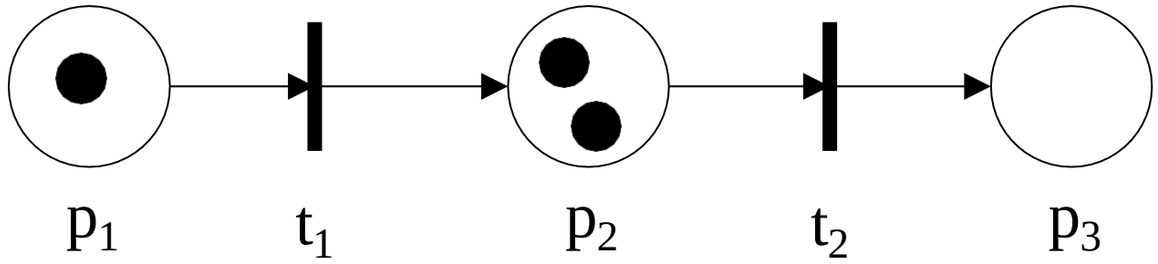


- The marking below is formally captured by:
  - $\{(p_1,1),(p_2,2),(p_3,0)\}$
- Alternative notation
  - p_1+2p_2.

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Enabled transitions

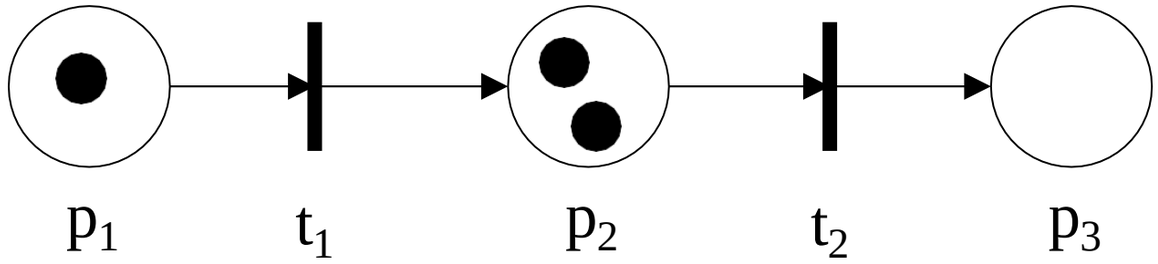
- Given a Petri Net N = (P, T, F) a transition t ∈ T is enabled in a marking M iif ∀p ∈ •t, m(p) > 0



- Is t_1 enabled?
- Is t_2 enabled?

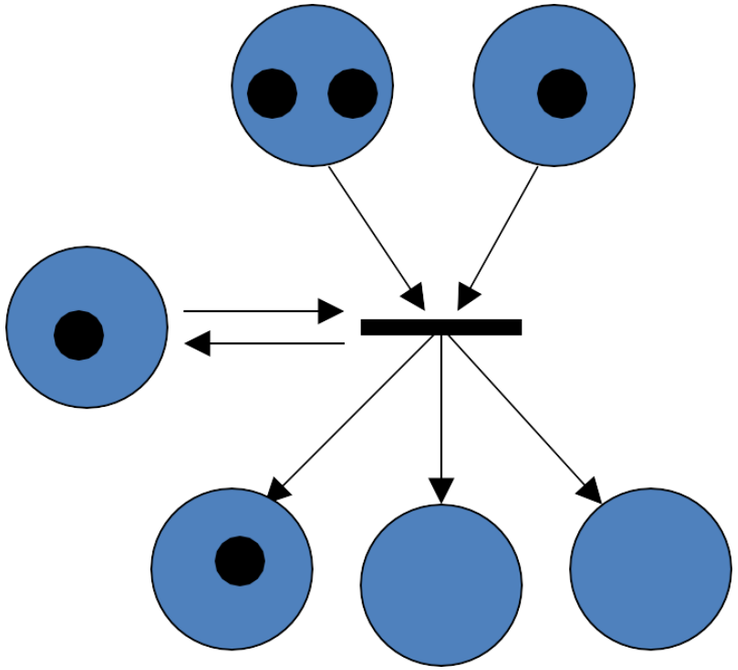
Enabled transitions

- Given a Petri Net N = (P, T, F) a transition t ∈ T is enabled in a marking M iif ∀p ∈ •t, m(p) > 0



- Is t_1 enabled? Yes! $m(p_1) = 1 > 0$
- Is t_2 enabled? Yes! $m(p_2) = 2 > 0$

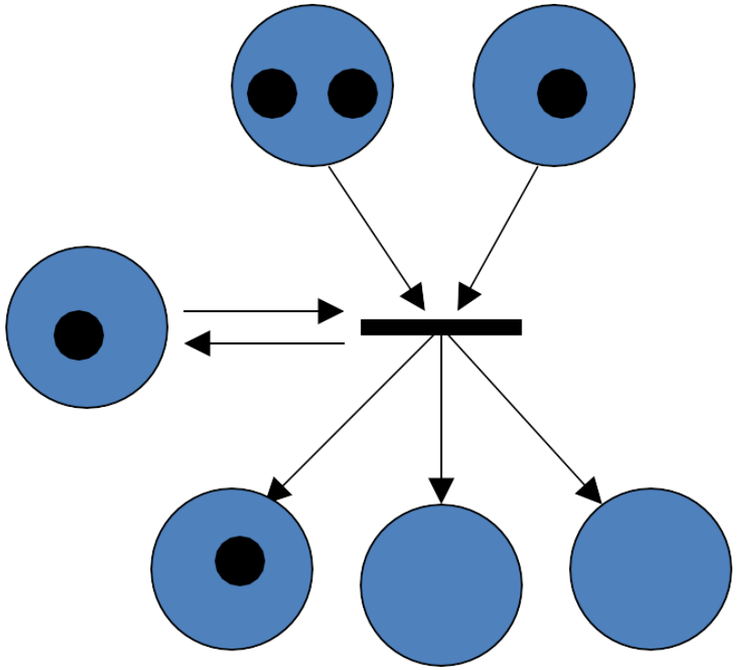
Example



Is the transition enabled?

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Example



Is the transition enabled?

Yes! We have a token in each place belonging to the preset

Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Firing transition

- Given a marking M of a Petri Net N = (P, T, F), any enabled transition may fire
- When transition fires:
  - a token is removed from each of the input places
  - a token is produced for each of the output places

- To better define the transition firing we firstly introduce the weight function *w*
  - w: (P X T) ∪(T X P) → N
    - w(x,y) != 0 if (x,y) ∈ F
    - w(x,y) = 0 if (x,y) ∉ F
- In case range of w is [0,1] then the Petri Net is named *ordinary Petri Net*

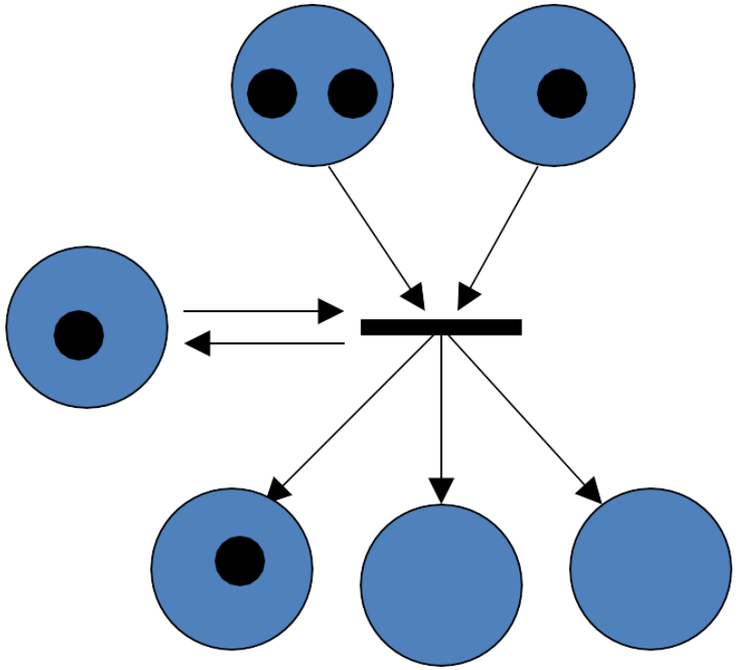
Firing transition

- We can formally define the firing transition as follows:

- Given a Petri Net N = (P, T, F)
  - the current marking M and the weight function *w,* a transition t ∈ T can fire iif *t* is enabled in M
  - the firing of *t* yields to a new marking M’ where

    ∀p ∈ P, m’(p) = m(p) – w(p,t) + w(t,p)

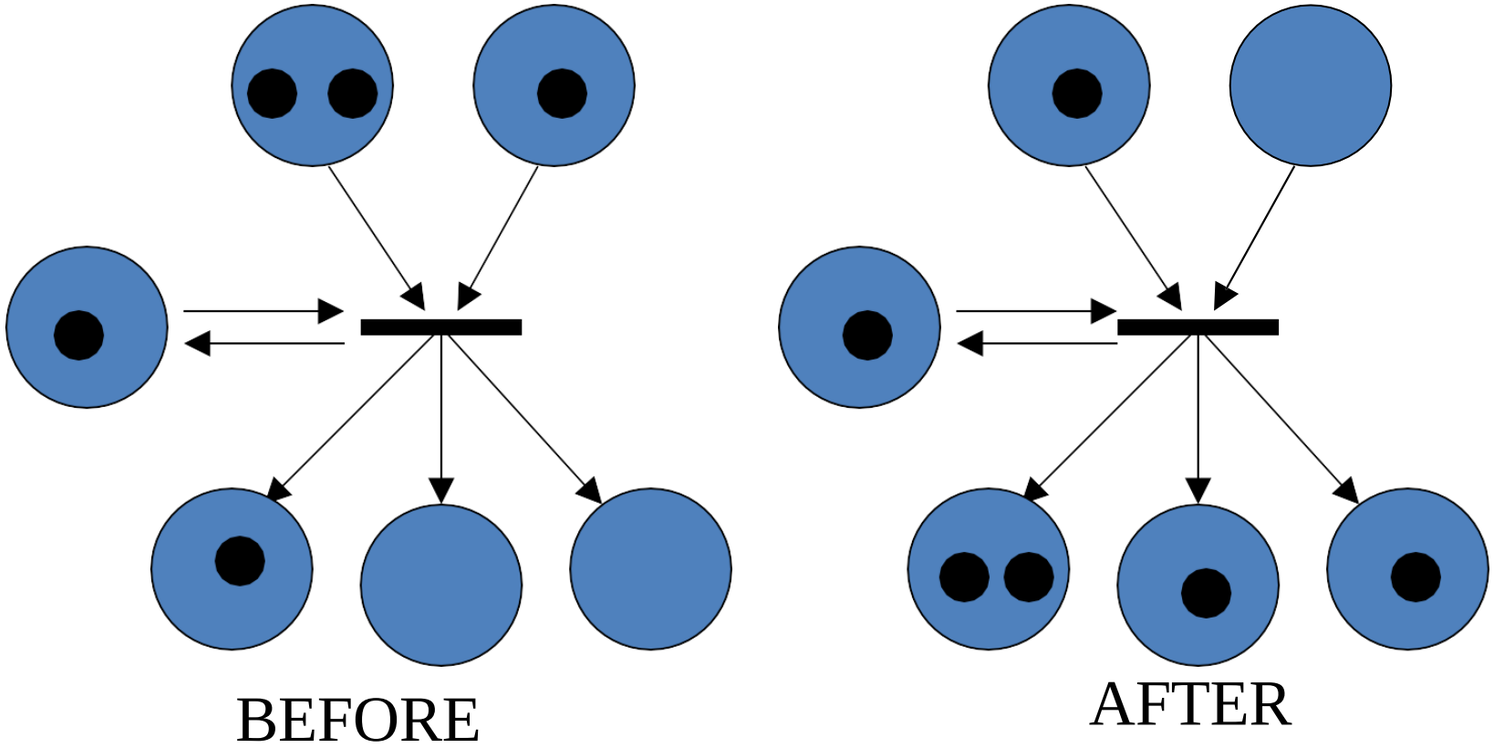
Example



BEFORE

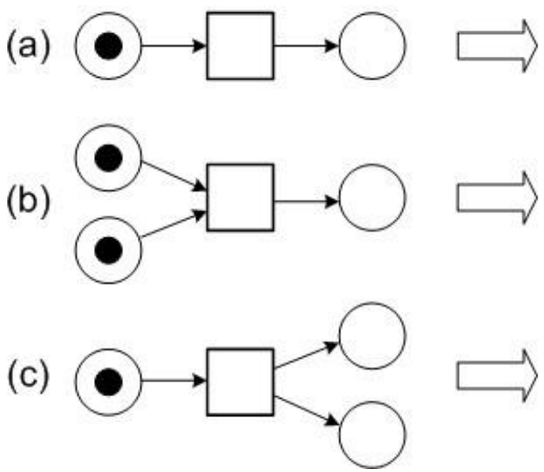
Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

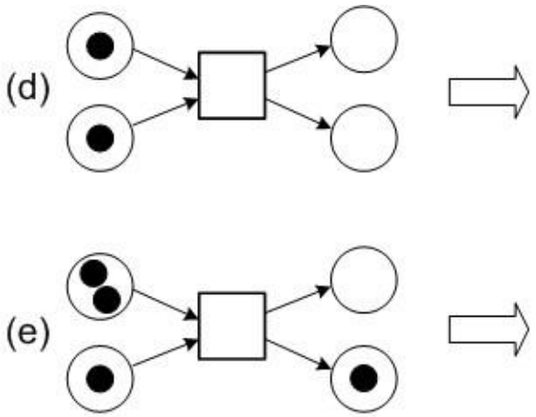
Example



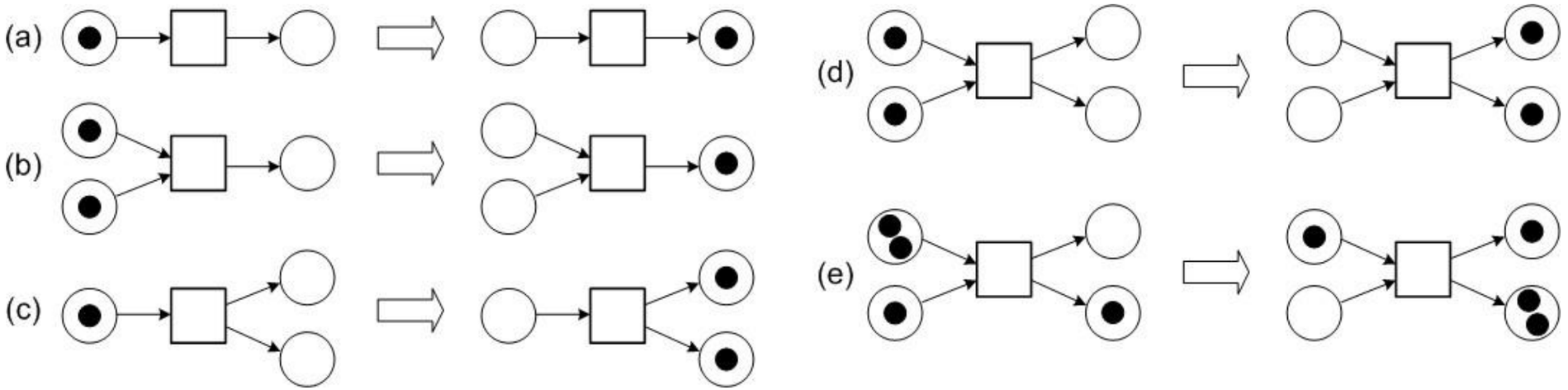
Source: A.H.M. ter Hofstede, W. van der Aalst, M. Adams, N. Russel, Modern Business Process Automation: YAWL and its support environment, Springer, 2009

Examples



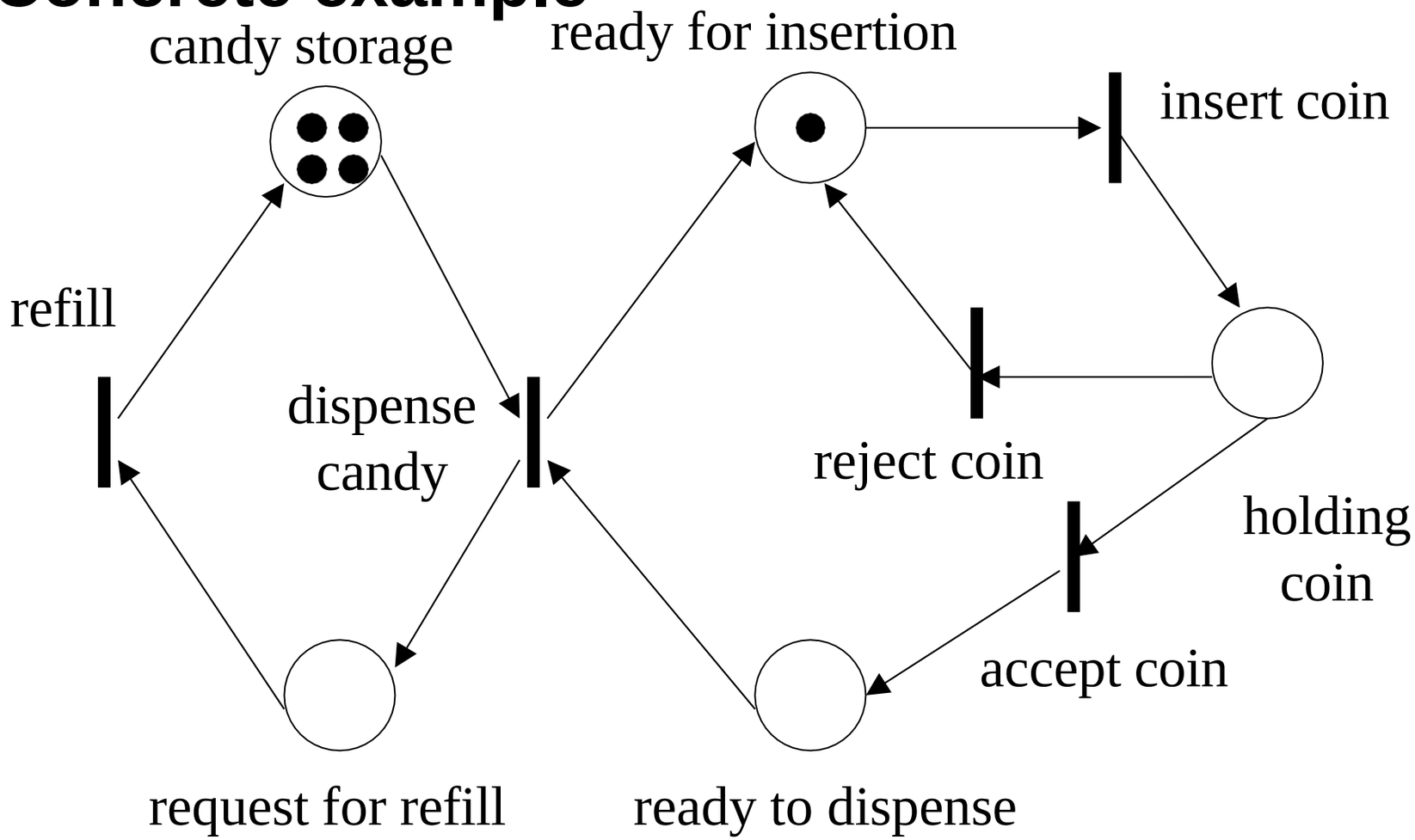


Examples

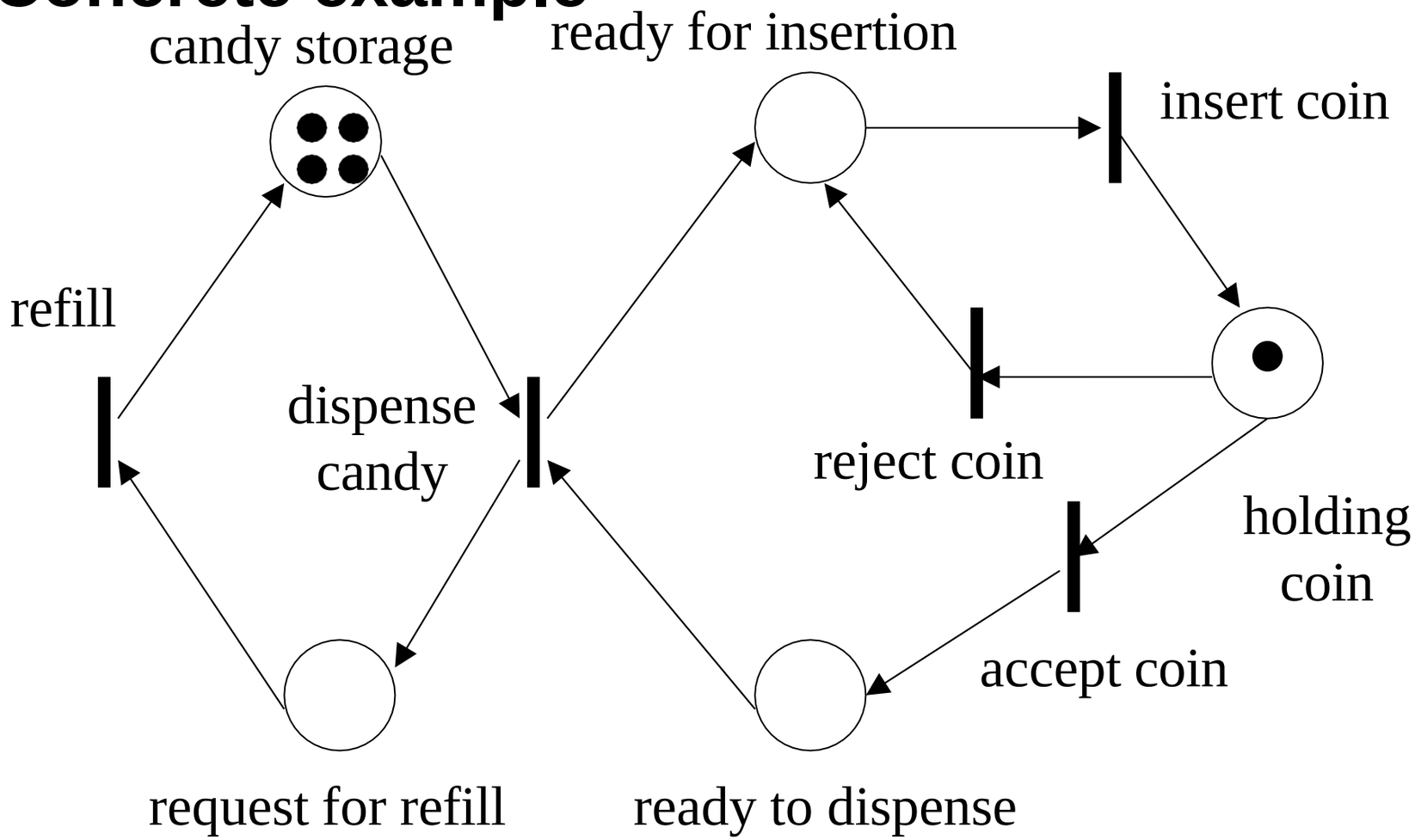


Concrete example

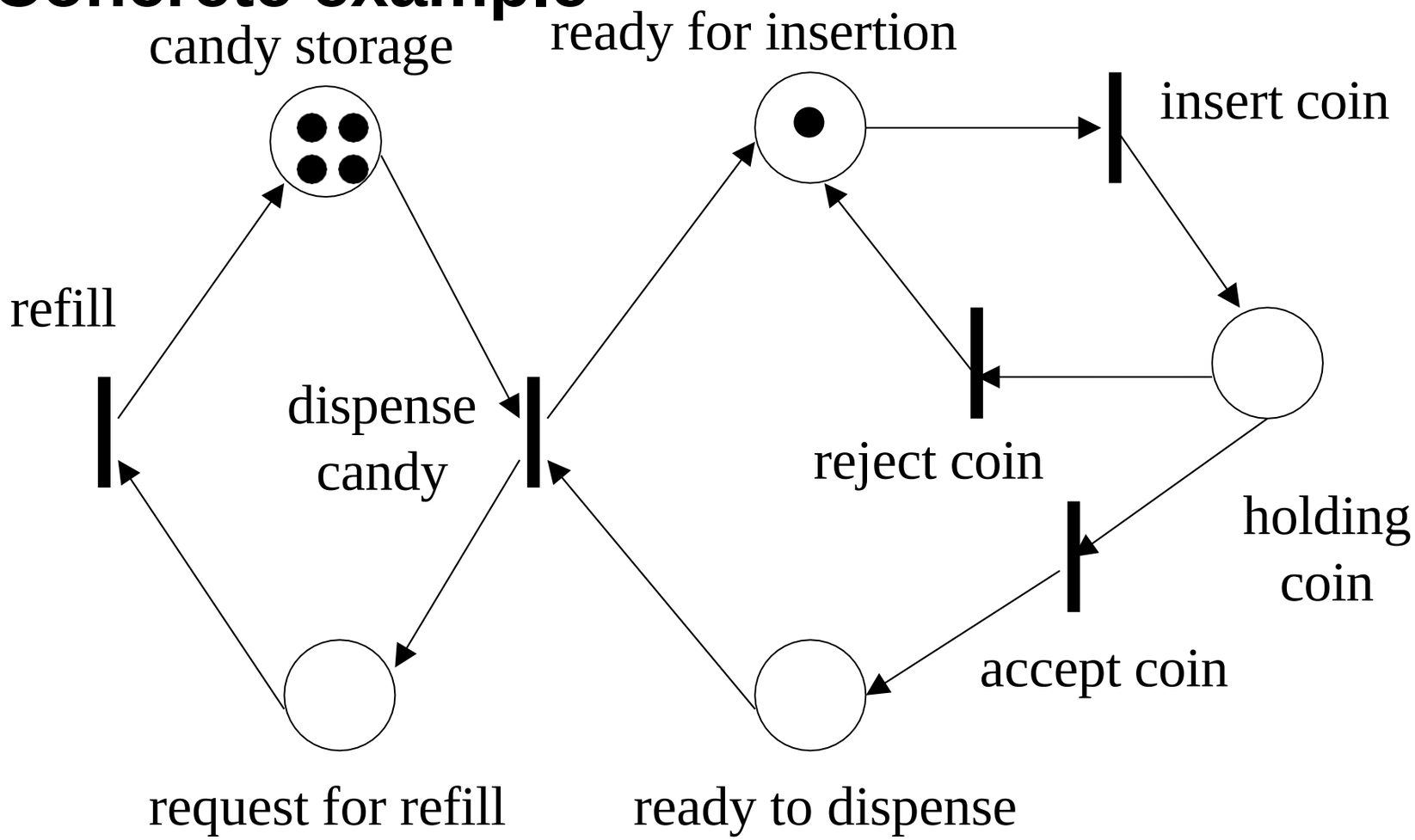
Concrete example



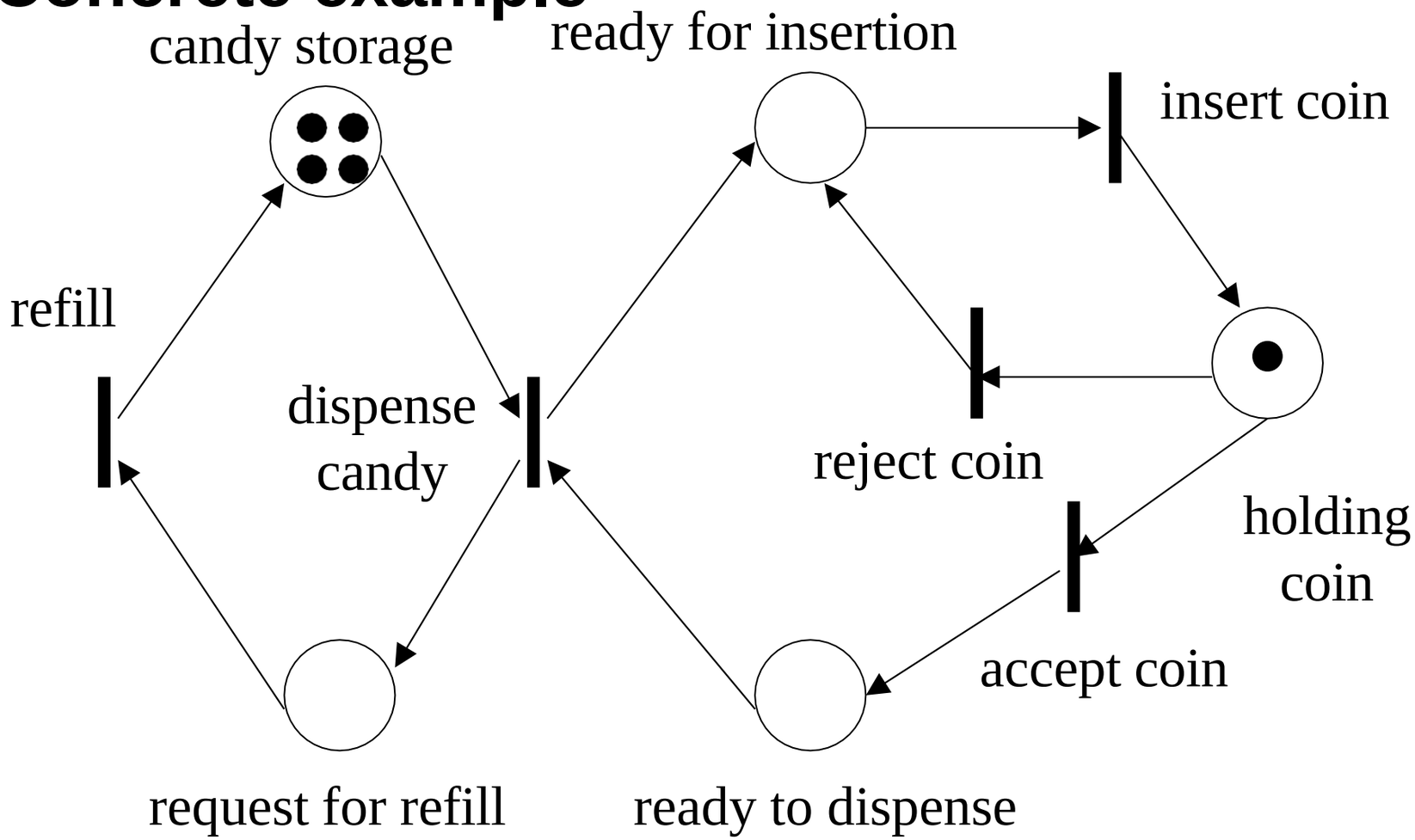
Concrete example



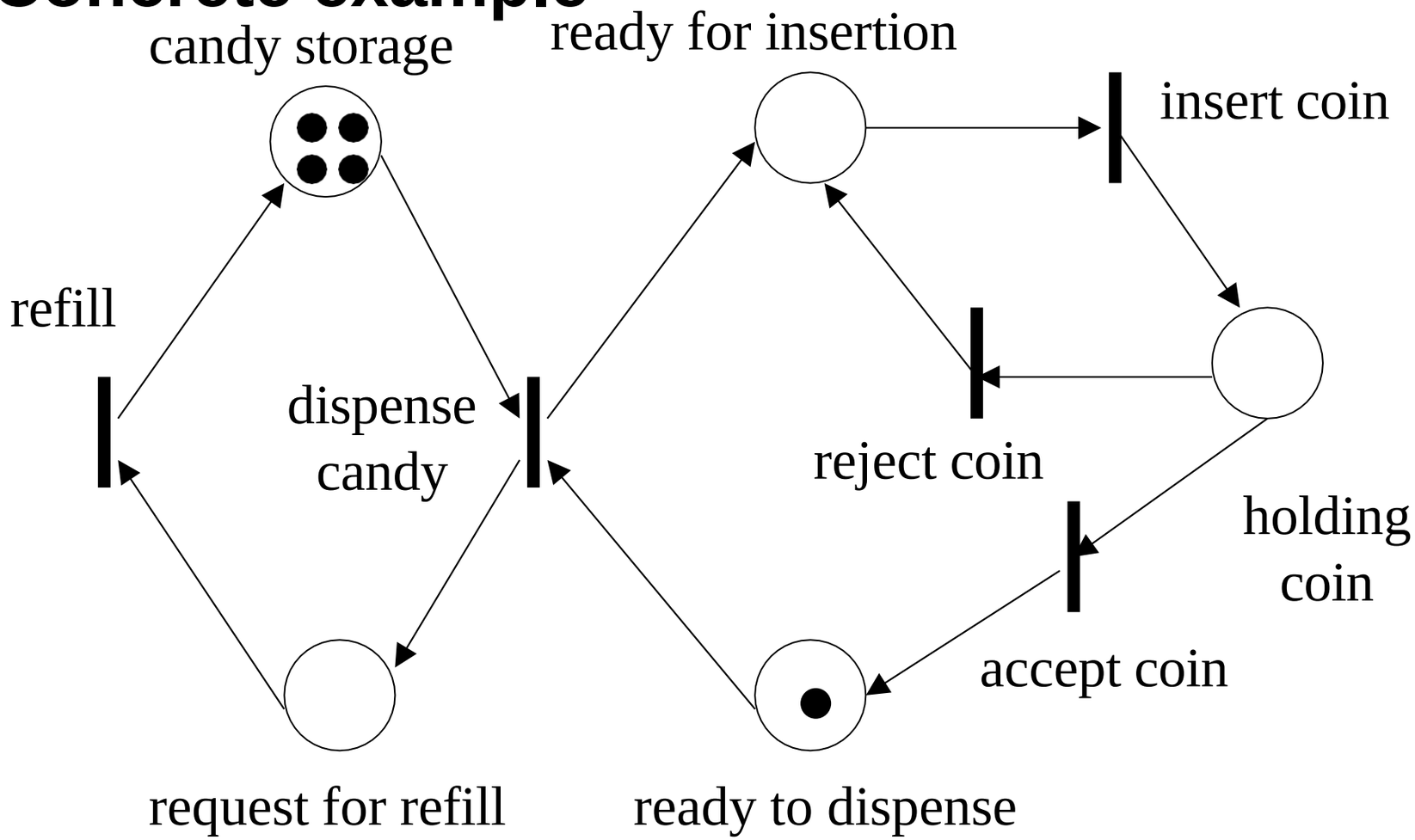
Concrete example



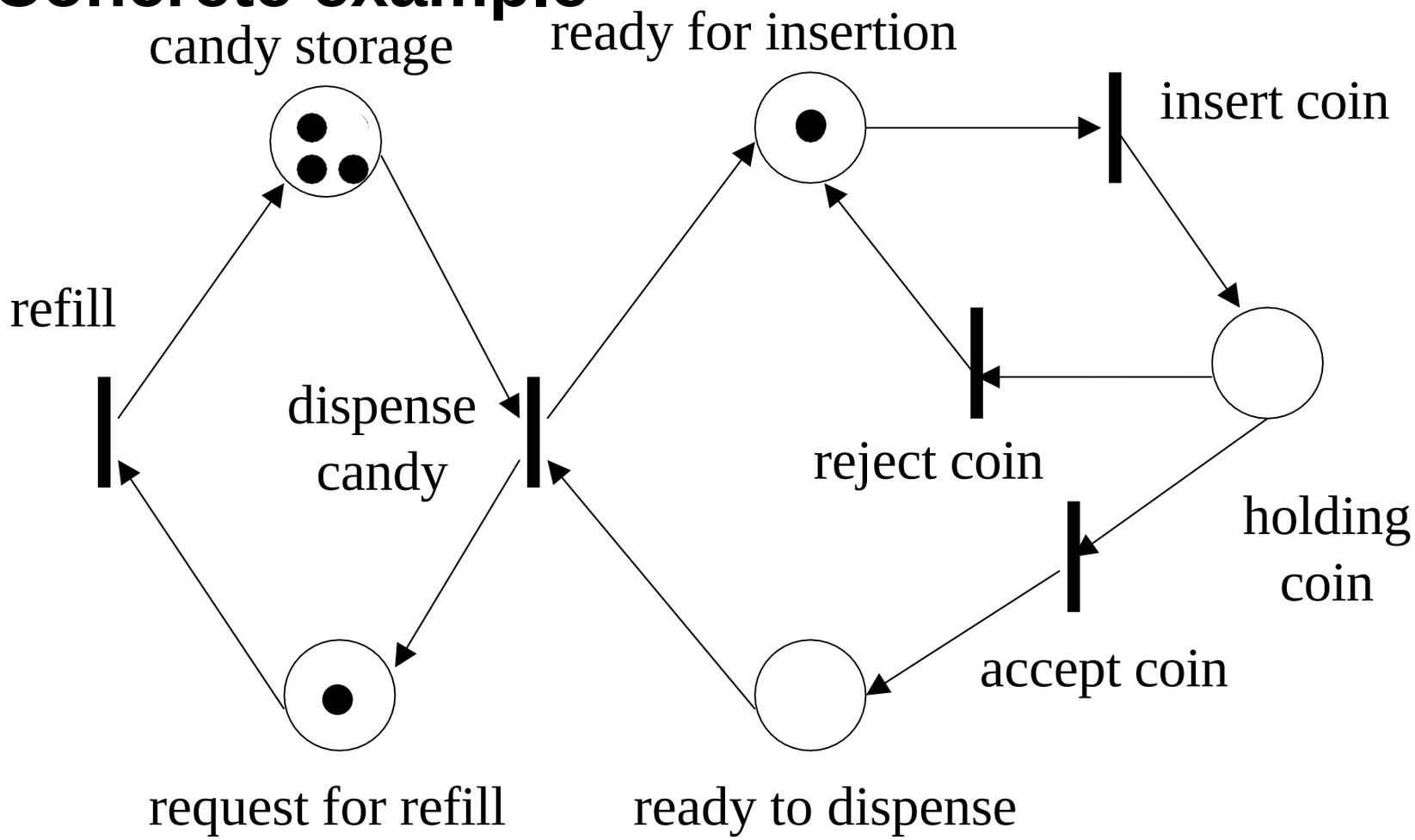
Concrete example



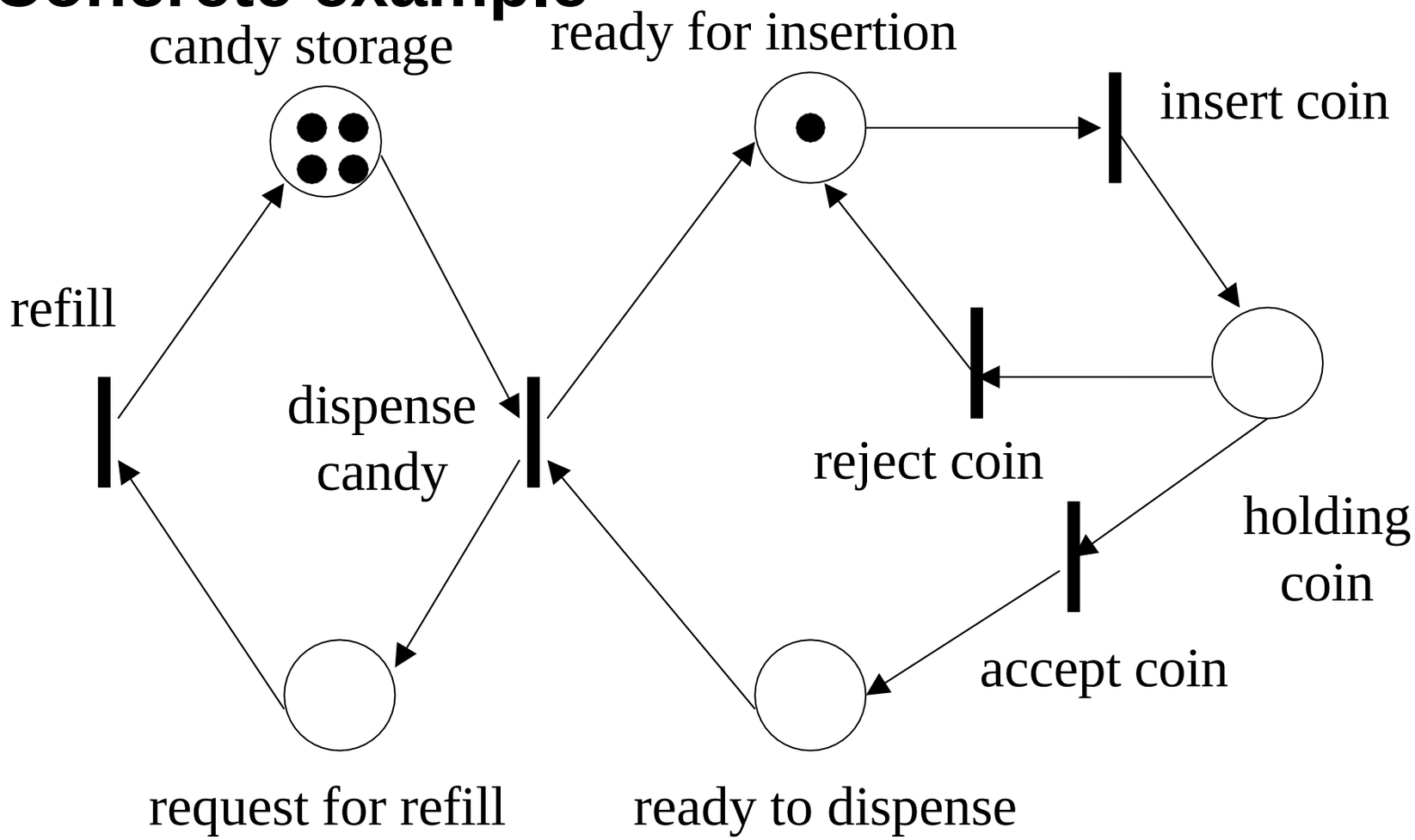
Concrete example



Concrete example



Concrete example



Nondeterminism

- For a given marking, more than one transitions could be enabled
- If there are multiple enabled transitions, any one of them may fire
  - for execution purposes, it is assumed that **they cannot fire simultaneously.**
- It is assumed that the firing of a transition is an atomic action that occurs instantaneously and cannot be interrupted

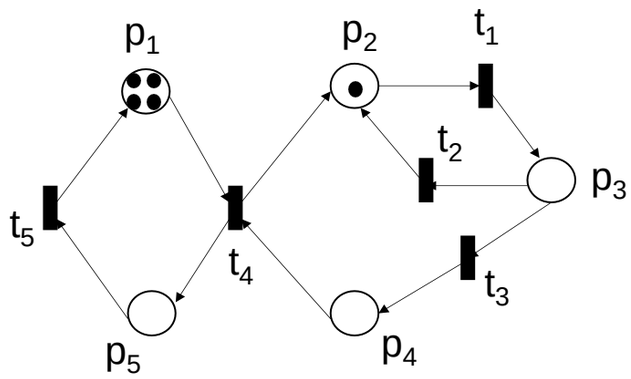
Reachable marking

- Given a Petri Net (P, T, F) and a state M_1:
  - M_1 [t⟩ M_2 : a transition t is enables in state M_1 and firing t in M_1 results in state M_2
  - M_1 [⟩ M_2 : there is a transition t such that M_1 [t⟩ M_2
  - M_1 [σ⟩ M_n : the firing sequence $\sigma=t_1,t_2, \ldots ,t_n$ leads from state M_1 to state M_n, i.e., M_1 [t_1⟩ M_2 [t_2⟩ M_2 … M_{n-1} [t_n⟩ M_n

- A state M_n is called reachable from M_1 if and only if there is a firing sequence $\sigma=t_1,t_2, \ldots ,t_n$ such that M_1 [σ⟩ M_n
  - An empty firing sequence is also allowed M_1 [ε⟩ M_1

Reachability graph

1. **Label the initial marking m_0 as the root and tag it "new".**
2. While "new" markings exists, do the following:
   a) Select a new marking m.
   b) If no transitions are enabled at m, tag m "dead-end".
   c) While there exist enabled transitions at m, do the following for each enabled transition t at m:
      I. Obtain the marking m’ that results from firing t at m.
      II. If m’ does not appear in the graph add m’ and tag it "new".
      III. Draw an arc with label t from m to m’ (if not already present).
3. Output the graph

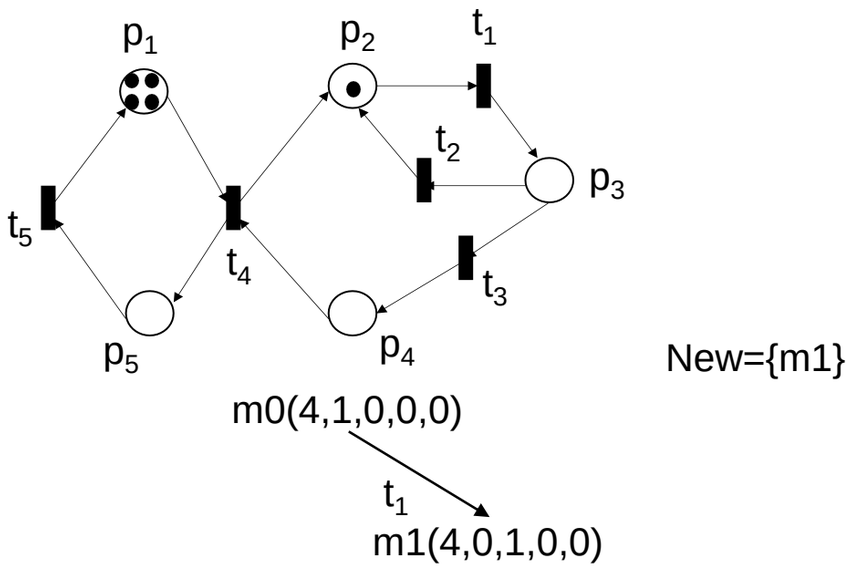


m0(4,1,0,0,0)

New={m0}

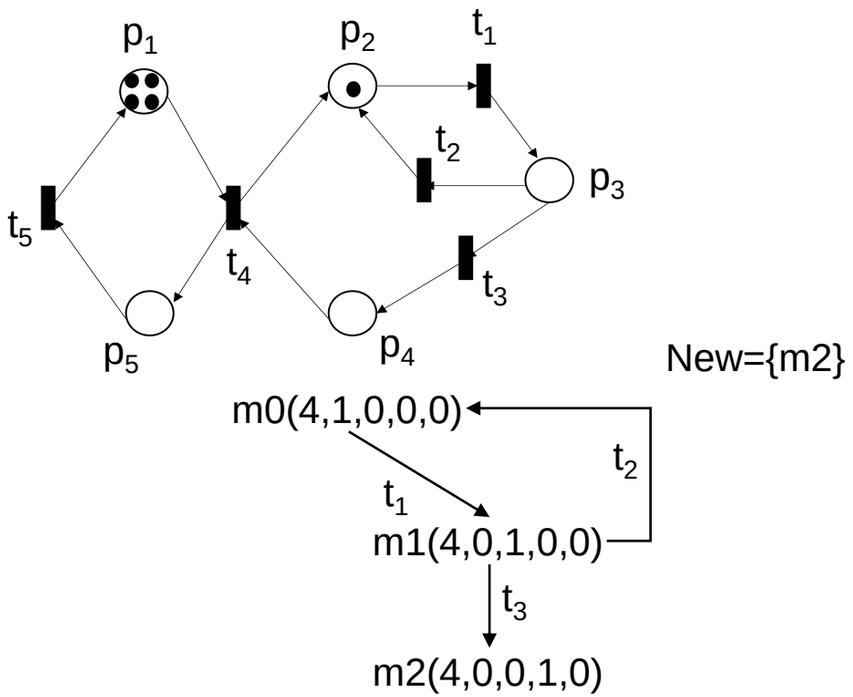
Reachability graph

1. Label the initial marking m_0 as the root and tag it "new".
2. **While "new" markings exists, do the following:**
   a) Select a new marking m.
   b) If no transitions are enabled at m, tag m "dead-end".
   c) While there exist enabled transitions at m, do the following for each enabled transition t at m:
      I. Obtain the marking m' that results from firing t at m.
      II. If m' does not appear in the graph add m' and tag it "new".
      III. Draw an arc with label t from m to m' (if not already present).
3. Output the graph



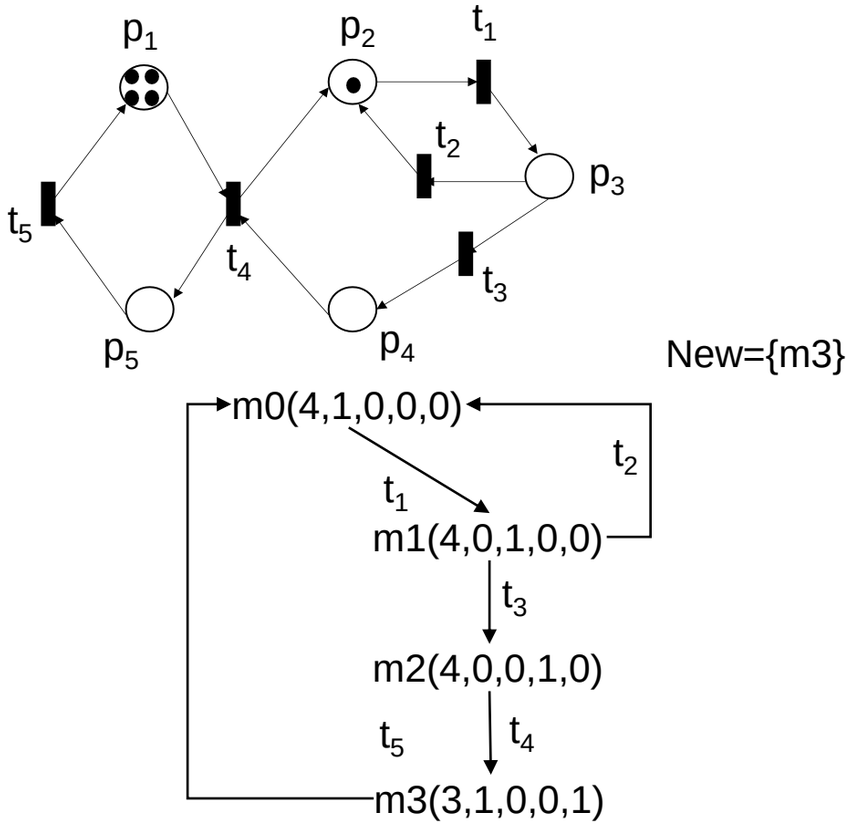
Reachability graph

1. Label the initial marking m_0 as the root and tag it "new".
2. **While "new" markings exists, do the following:**
   a) Select a new marking m.
   b) If no transitions are enabled at m, tag m "dead-end".
   c) While there exist enabled transitions at m, do the following for each enabled transition t at m:
      I. Obtain the marking m' that results from firing t at m.
      II. If m' does not appear in the graph add m' and tag it "new".
      III. Draw an arc with label t from m to m' (if not already present).
3. Output the graph



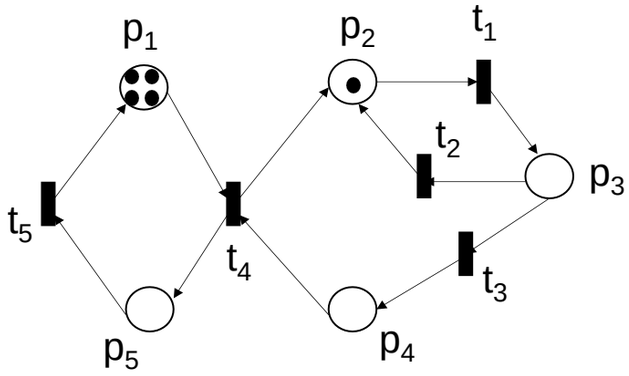
Reachability graph

1. Label the initial marking m_0 as the root and tag it "new".
2. **While "new" markings exists, do the following:**
   a) Select a new marking m.
   b) If no transitions are enabled at m, tag m "dead-end".
   c) While there exist enabled transitions at m, do the following for each enabled transition t at m:
      I. Obtain the marking m' that results from firing t at m.
      II. If m' does not appear in the graph add m' and tag it "new".
      III. Draw an arc with label t from m to m' (if not already present).
3. Output the graph

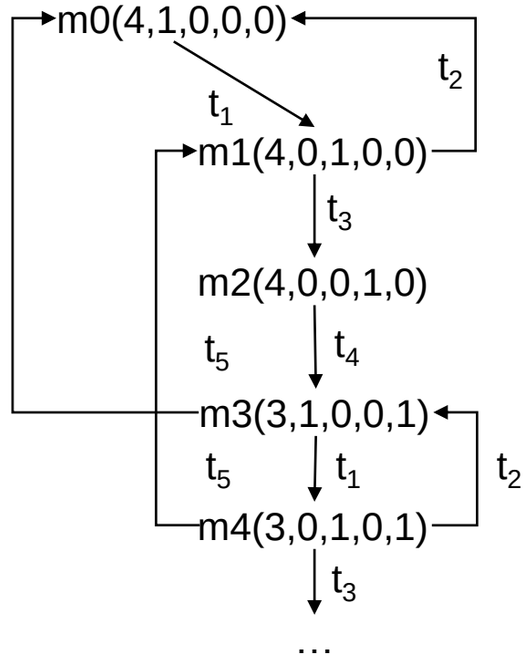


Reachability graph

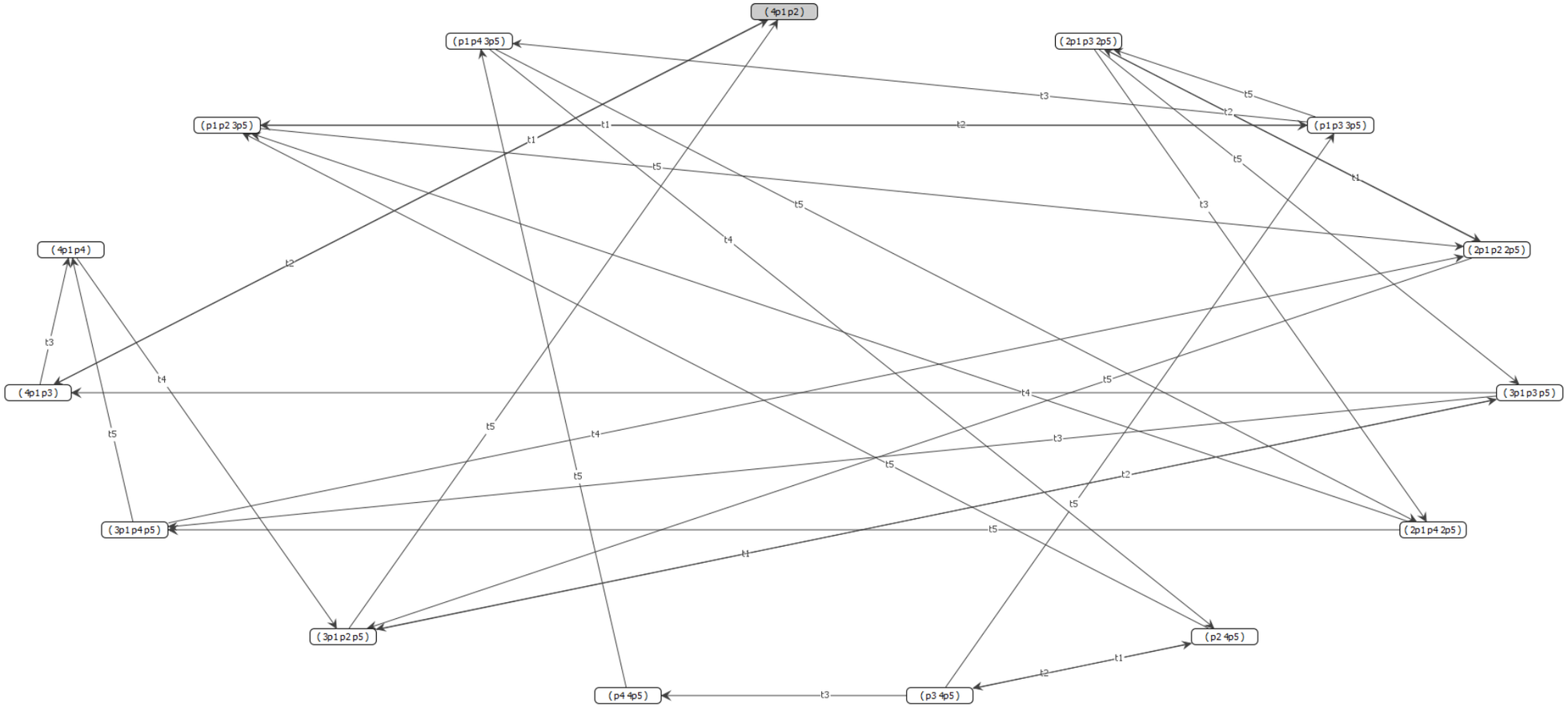
1. Label the initial marking m_0 as the root and tag it "new".
2. **While "new" markings exists, do the following:**
   a) Select a new marking m.
   b) If no transitions are enabled at m, tag m "dead-end".
   c) While there exist enabled transitions at m, do the following for each enabled transition t at m:
      I. Obtain the marking m' that results from firing t at m.
      II. If m' does not appear in the graph add m' and tag it "new".
      III. Draw an arc with label t from m to m' (if not already present).
3. Output the graph



New={m4}



Reachability graph

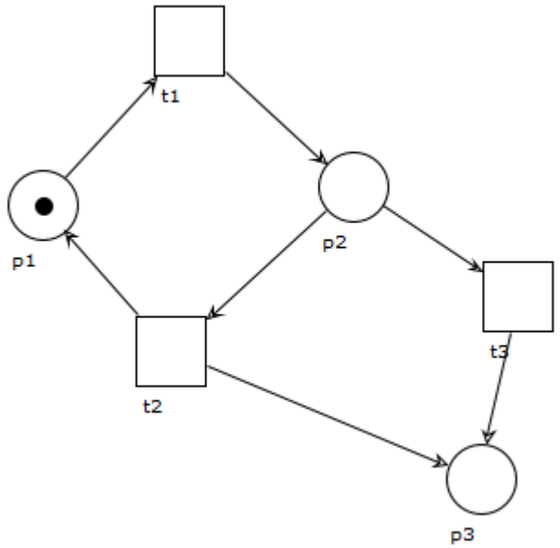


Some Petri Nets properties

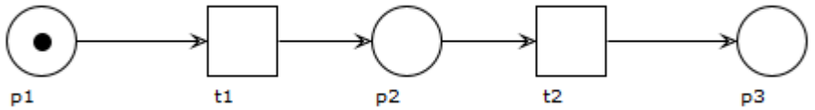
- Boundness
  - Does the net manage a limited number of tokens regardless of the reached marking?
  - Implies a finite state space
- Liveness
  - Does the net have a marking with none of the transitions are enabled?
  - Implies the absence of deadlocks

Boundness

- A Petri Net N = (P, T, F) with initial marking M_0 is k-bounded iif for each reachable markings M_0 [σ⟩ M_i
  - ∀ p ∈ P, m_i (p) ≤ k
- If k = 1 then the Petri Net is safe



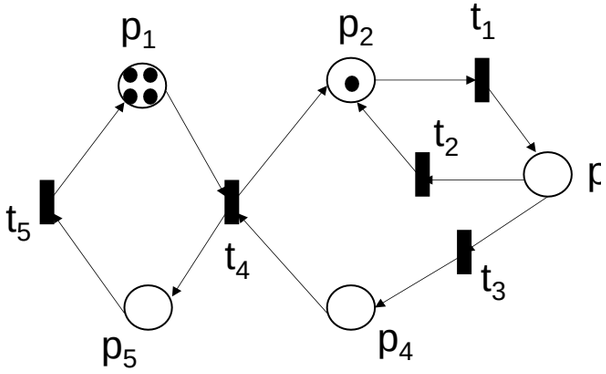
Unbounded: reachability graph grows infinitely



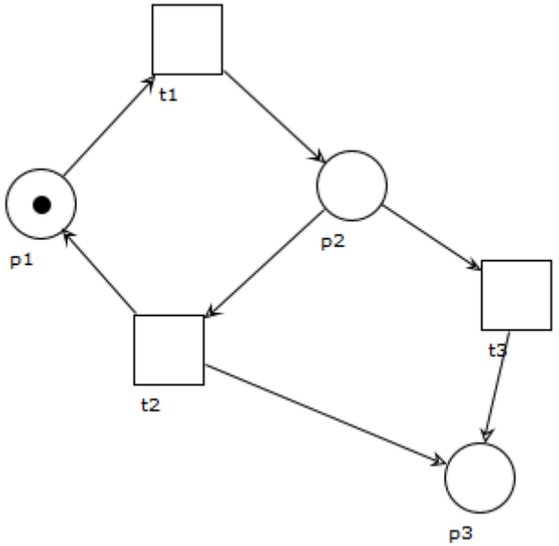
1-Bounded (safe)

Termination

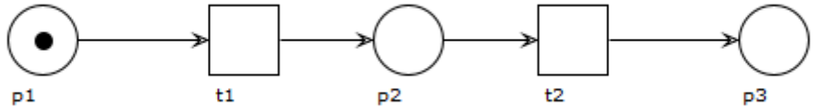
- Intuition: The Petri net always reaches a terminal marking
- A Petri net is terminating if every run is finite.
- Petri net with a finite and acyclic reachability graph is terminating
- A terminating Petri net has only finitely many runs



Non-terminating: reachability graph is cyclical



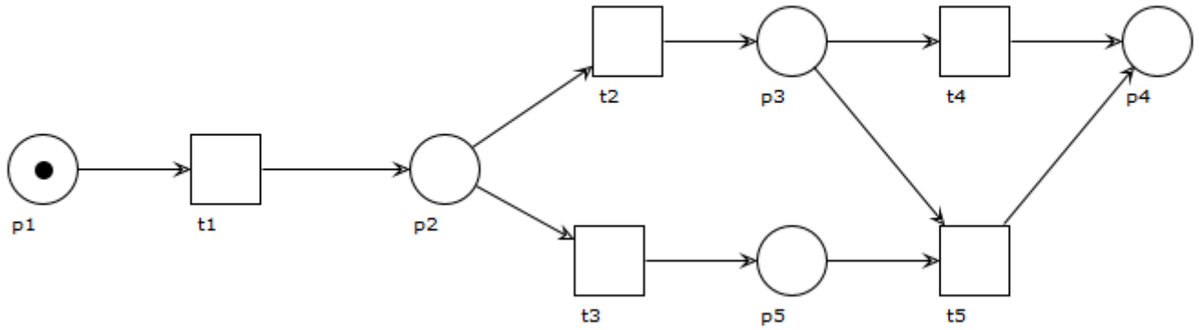
Non-terminating: infinite reachability graph



Terminating: marking (0,0,3) is terminal

Deadlock freedom

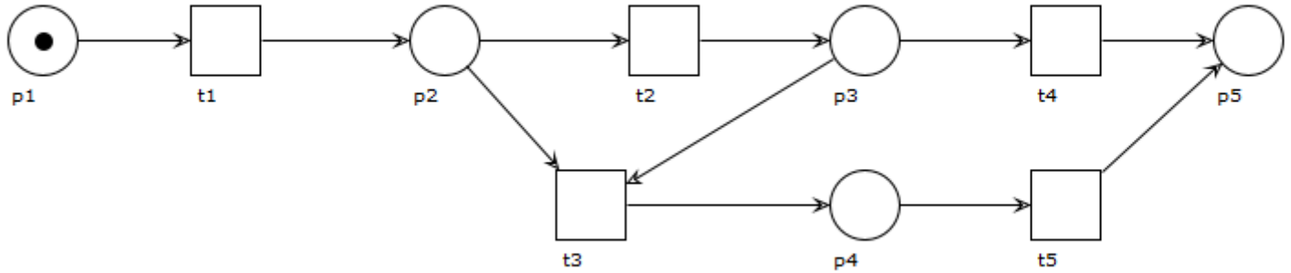
- Intuition: The Petri net can reach a terminal marking
- A Petri net is deadlock free if at least one transition is enabled at every reachable marking



Deadlock: no transition can fire from (0,0,0,0,1)

Dead transition

- Intuition: A transition can in principle occur (its implemented functionality can be used)
- A transition t of a Petri net is dead if t is not enabled at any reachable marking.



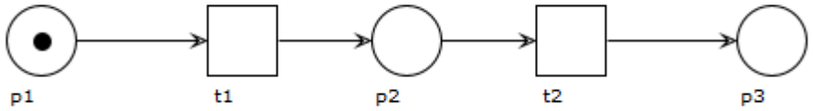
t_3 and t_5 are dead: they can never fire

Liveness

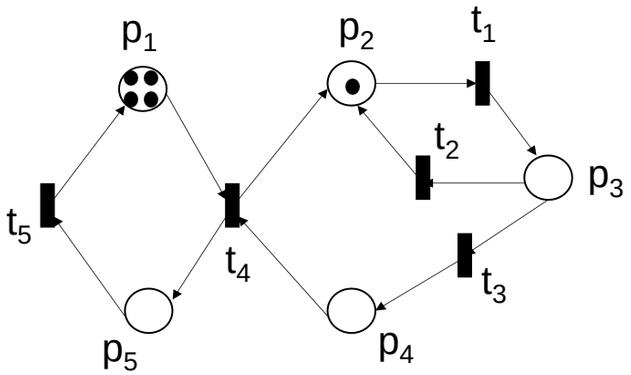
- Live transition
  - A transition t is live iif from every reachable marking M there is a marking M’ such that t is enabled
  - This property ensures that the transition can always become enabled again
- Live net
  - A Petri Net is live iif $\forall t \in T$, t is live

- Liveness and termination exclude each other
  - Termination holds for Petri net that always reach a marking where no transition is enabled.
  - A Petri Net is terminating iif every run is finite: i.e., the reachability graph is finite and acyclic

Liveness



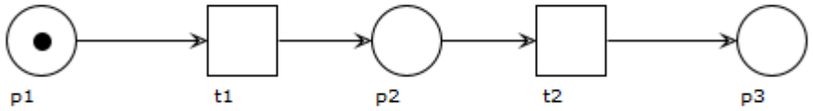
Not live: t_1 and t_2 can fire only once



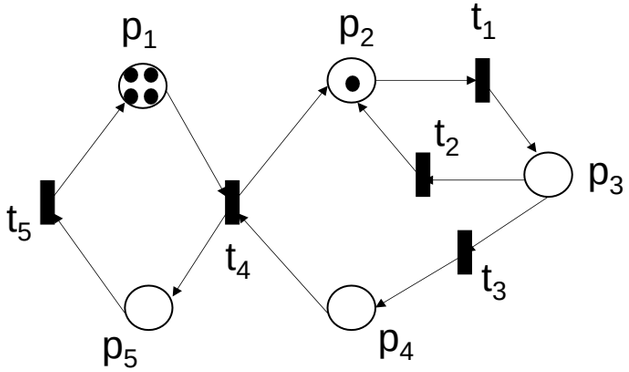
Live: all transitions can fire

Home-marking

- Intuition: A marking can always be reached again
- A marking m is a home-marking if from any reachable marking we can reach m. A Petri net is reversible if its initial marking is a home-marking.
- A Petri net is reversible if and only if its reachability graph is strongly connected.



Not reversible: $m_0(1,0,0)$ is not a home-marking



Reversible: we can always reach $m_0(4,1,0,0,0)$

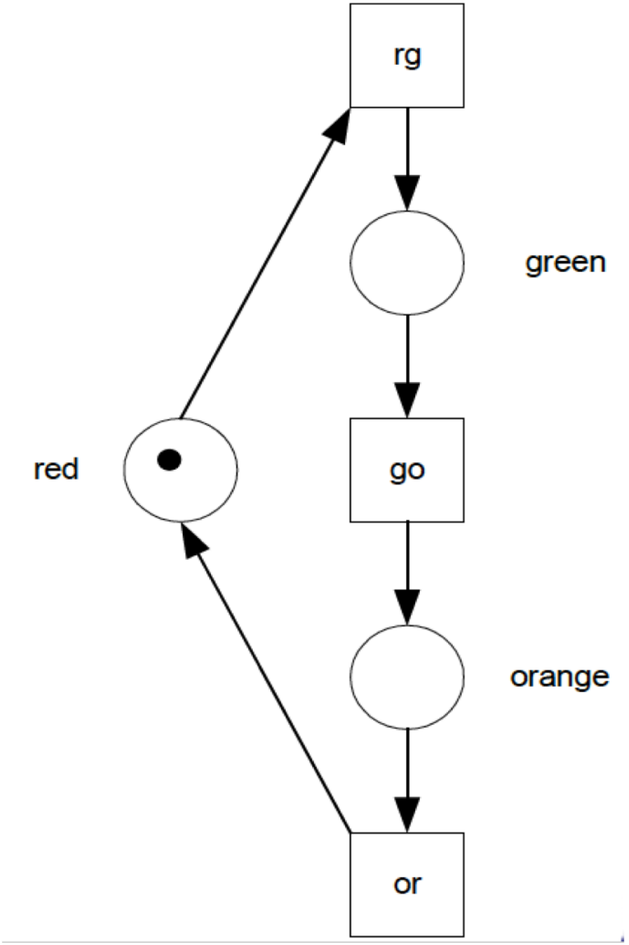
Example

- We want to model the behaviour of a traffic light
  - Light can be green, yellow, or red

Source: W. van der Aalst, C. Stahl, Modeling Business Processes: A Petri net approach, MIT Press, 2011

Example

- We want to model the behaviour of a traffic light
  - Light can be green, yellow, or red

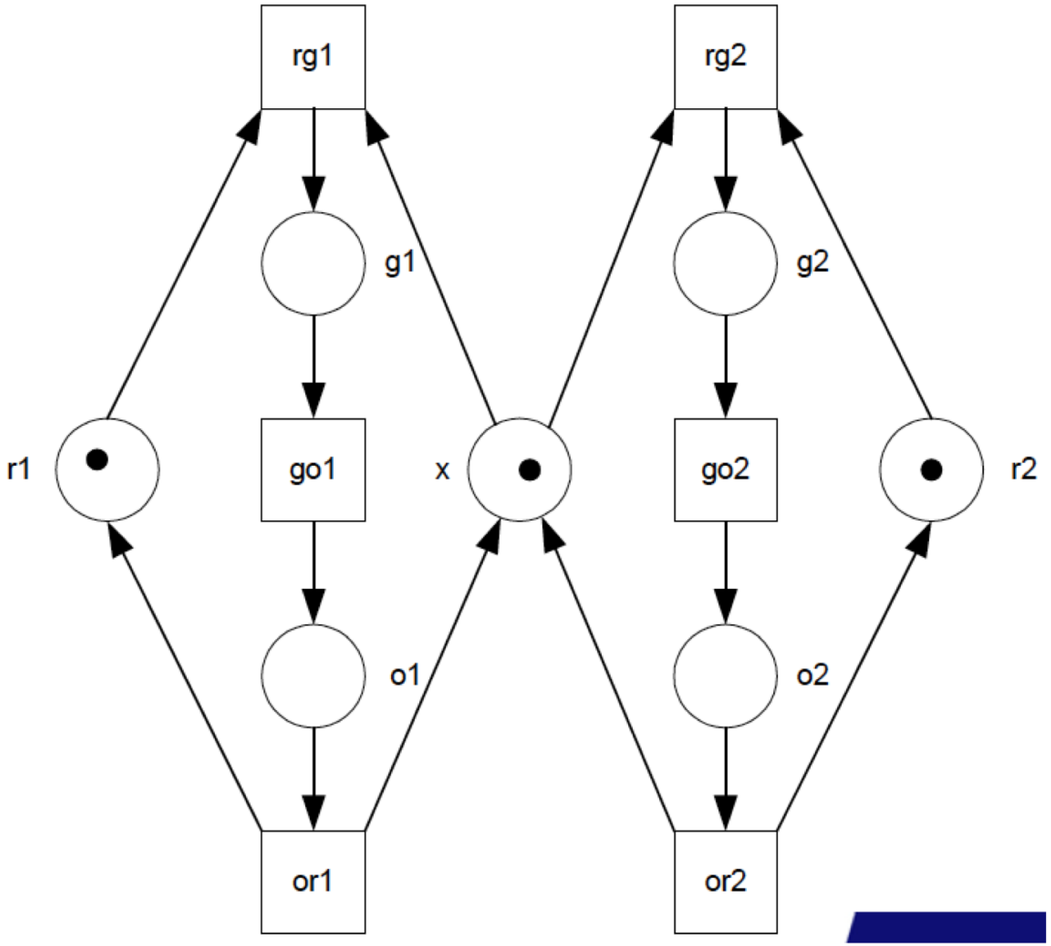


Source: W. van der Aalst, C. Stahl, Modeling Business Processes: A Petri net approach, MIT Press, 2011

Example

- We want to model with a Petri Net the behaviour of two traffic lights at an intersection, in a way that they cannot be green or yellow at the same time
- Conversely, they are allowed to signal red at the same time

Solution



Source: W. van der Aalst, C. Stahl, Modeling Business Processes: A Petri net approach, MIT Press, 2011

Workflow nets

Workflow net

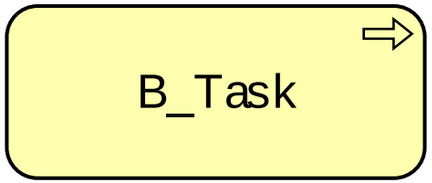
- Workflow net is defined as:
  - A Petri Net that models a workflow process definition (i.e., the life cycle of one case in isolation)

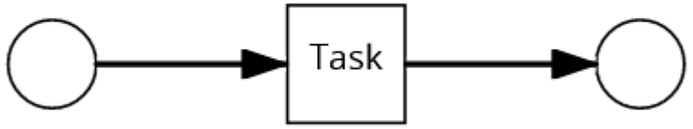
- A task
  - Generic piece of work (defined for a type of cases)
  - Corresponds to a transition
- A work item
  - Task enabled for a specific case
  - Corresponds to an enabled transition
- An activity
  - Actual execution of a work item
  - Correspond to a transition firing

Workflow net

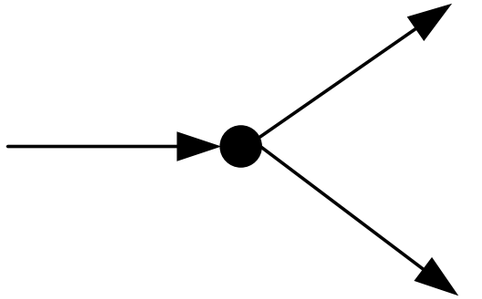
- A Petri net N = (P, T, F) is a workflow net if
- Object creation: N contains an input place i (the source place) such that $\bullet i = \varnothing$.
- Object completion: N contains an output place o (the sink place) such that $o \bullet = \varnothing$.
- Connectedness: Every node in N is on a path from i to o.

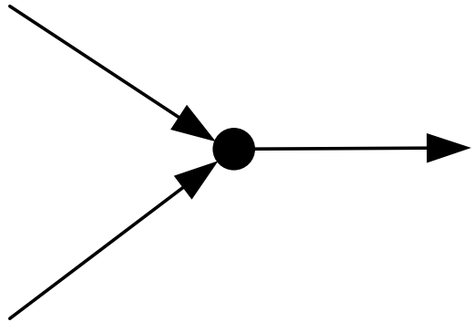
Workflow patterns with PN

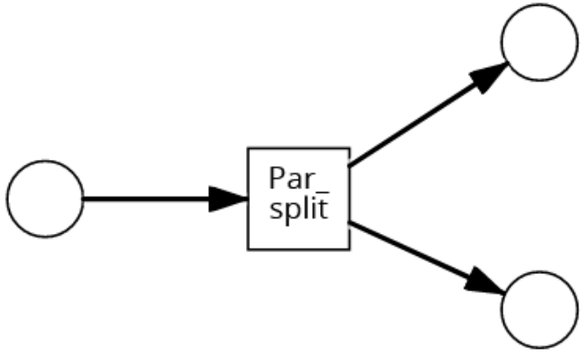


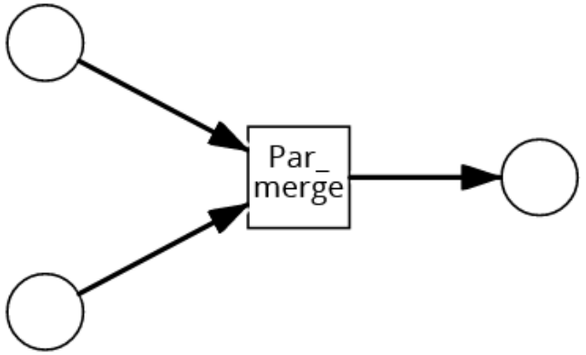


Workflow patterns with PN

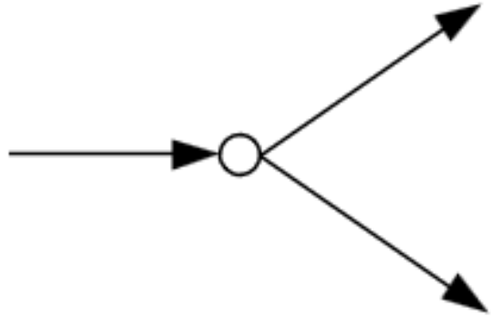


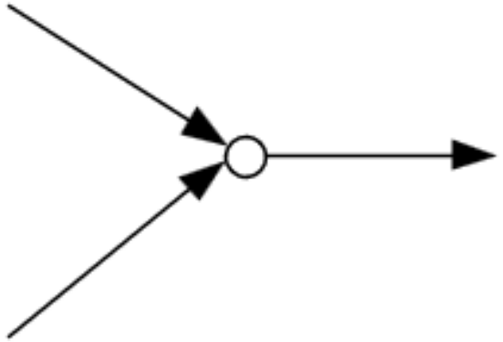


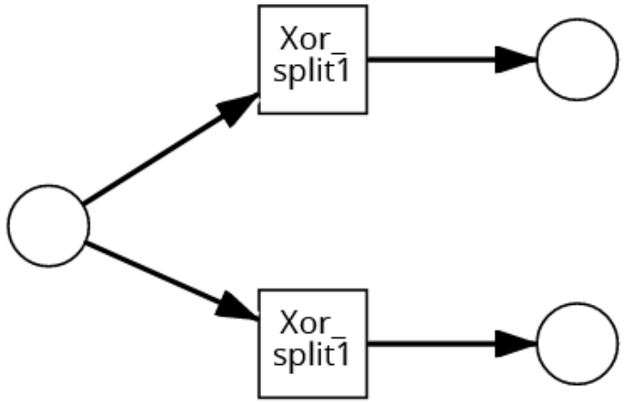


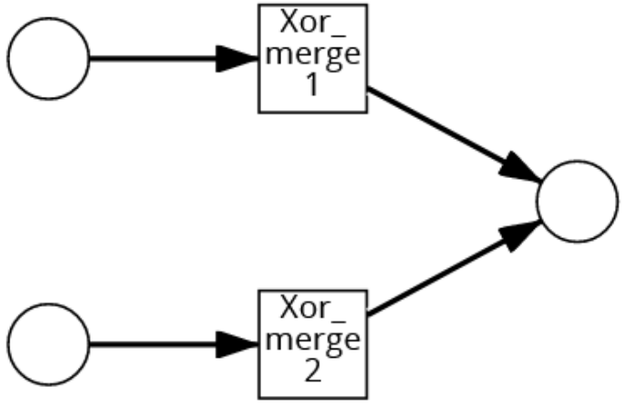


Workflow patterns with PN

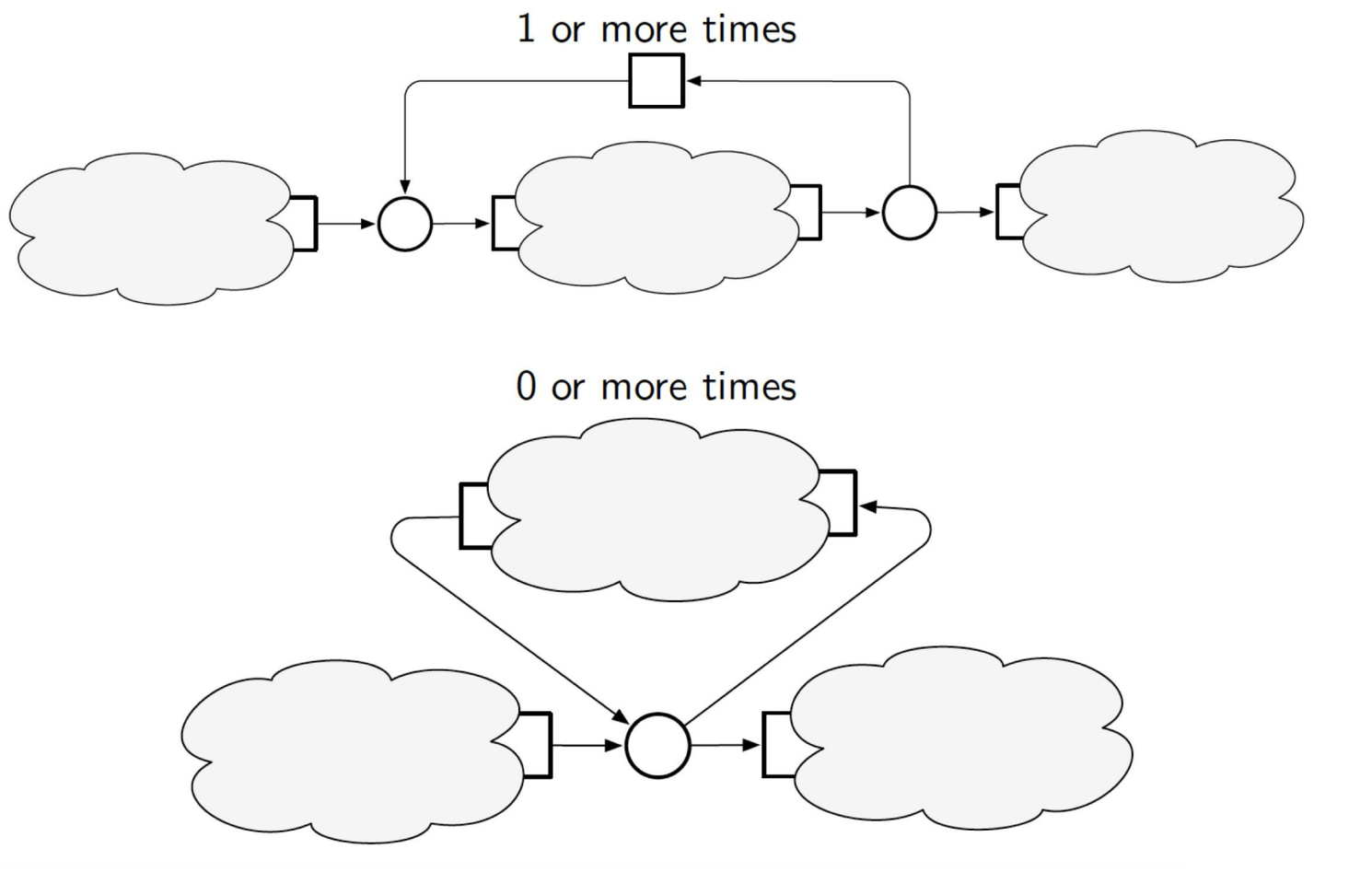








Workflow patterns with PN



Source: M. Montali, Conceptual Modeling form Information Systems, slides on Process Analysis – Petri Nets, Properties, A.Y. 2011/2012

Soundness

- A WF-Net (corresponding to a BP) is sound iif
  - “for any case, the procedure will terminate eventually, and at the moment the procedure terminates there is a token in place ‘o’ and all the other places are empty”

[from W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, Application and Theory of Petri Nets 1997, LNCS1248, pages 407–426. Springer-Verlag, 1997.]

- A little bit more formally:
  - For every state *M* reachable from state *i* there exists a firing sequence σ so that *M* [σ⟩ *o*
  - State *o* is the only state reachable from state *i* with at least one token in place *o*
  - There are no dead transitions in the workflow net in state *i*

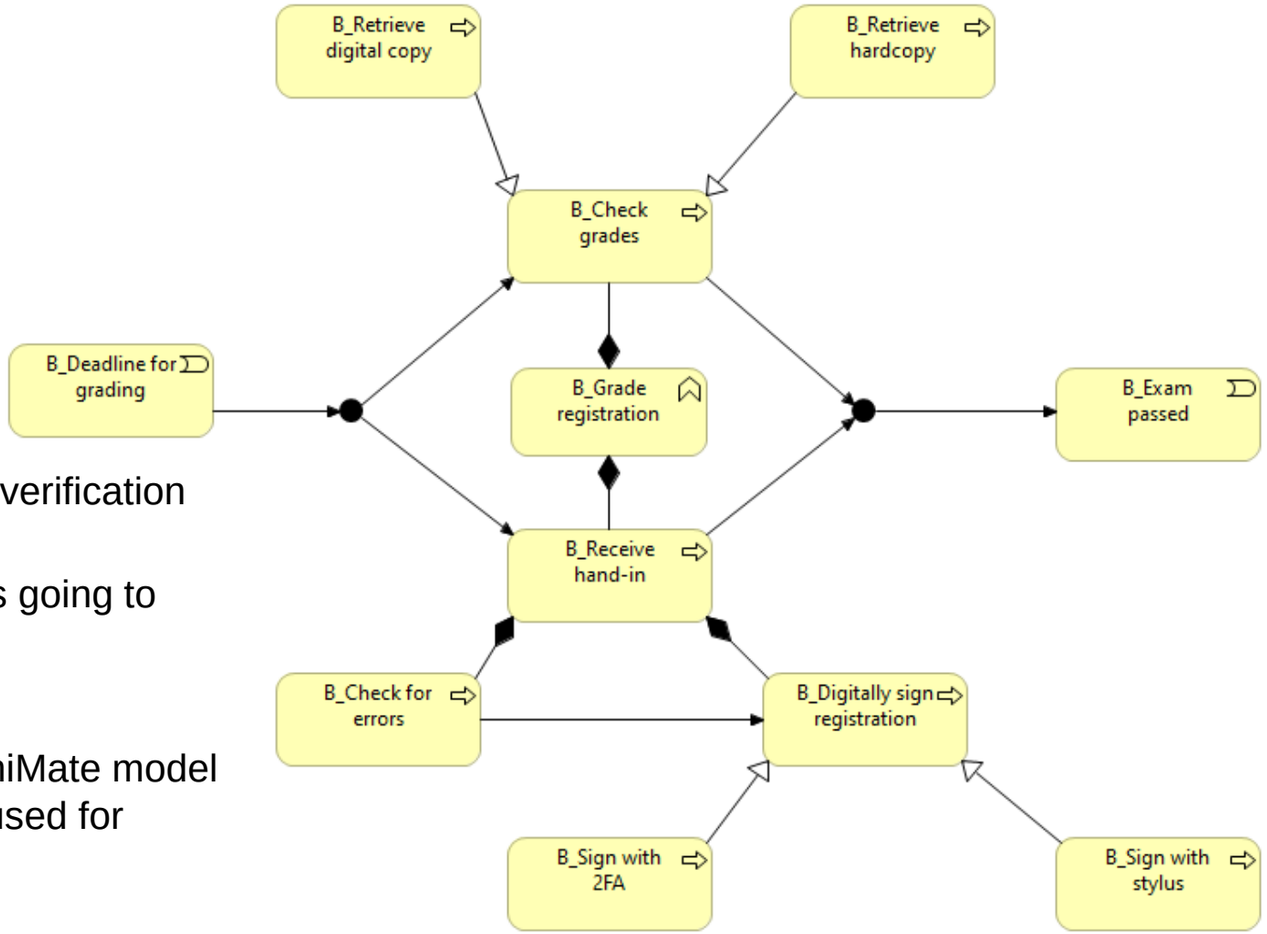
- Reachability graph can be used to verify the soundness of a WF-net

Soundness

- Van der Aalst proves that a WF-net is sound iif for the extended WF-Net the following properties hold:
  - Liveness
  - Safeness (1-boundness)

- The extended WF-Net is obtained adding to the WF-net an additional transition which has *o* as its input place and *i* as its output place

Back to our example in ArchiMate

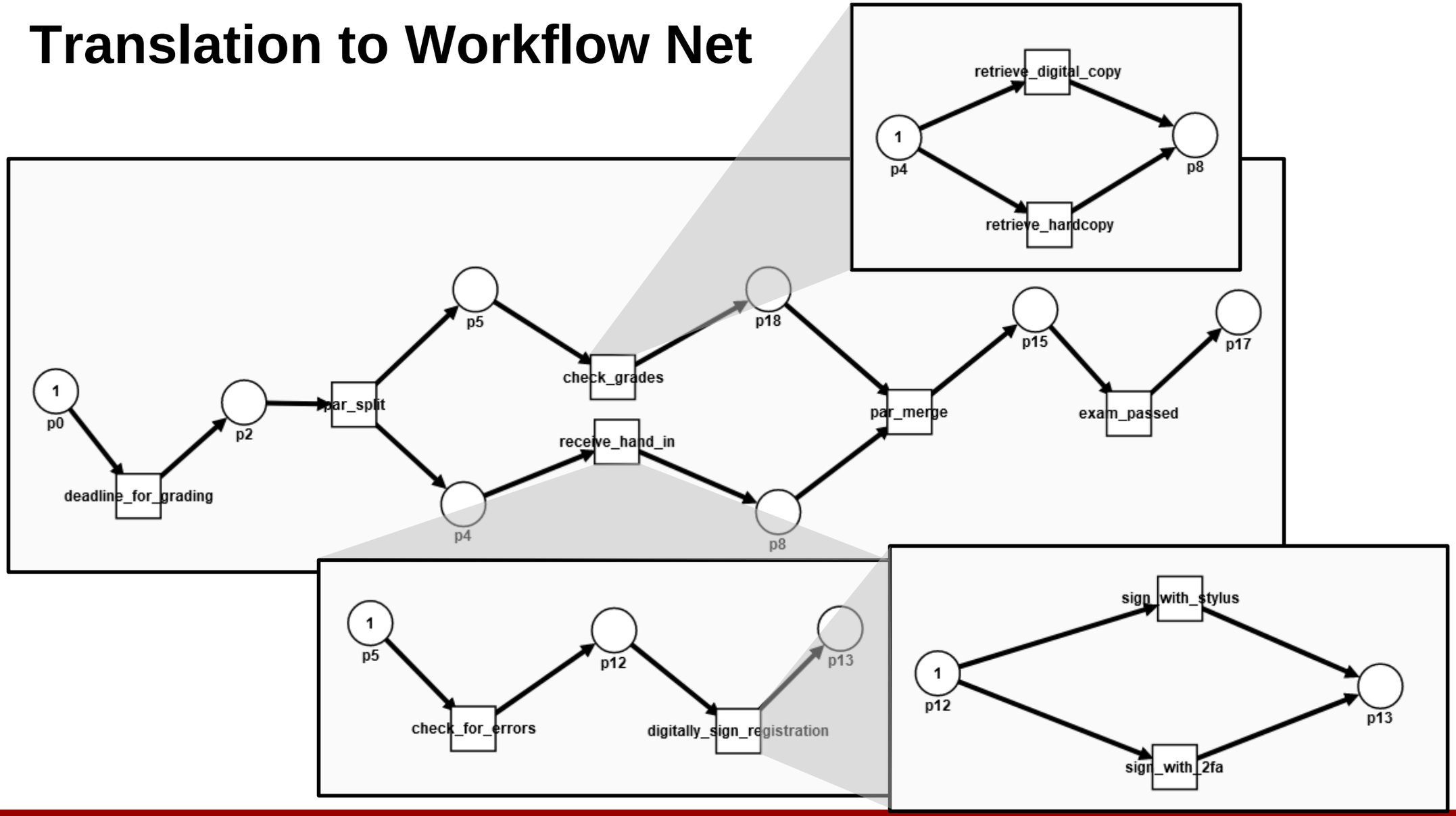


ArchiMate does not provide formal verification techniques:

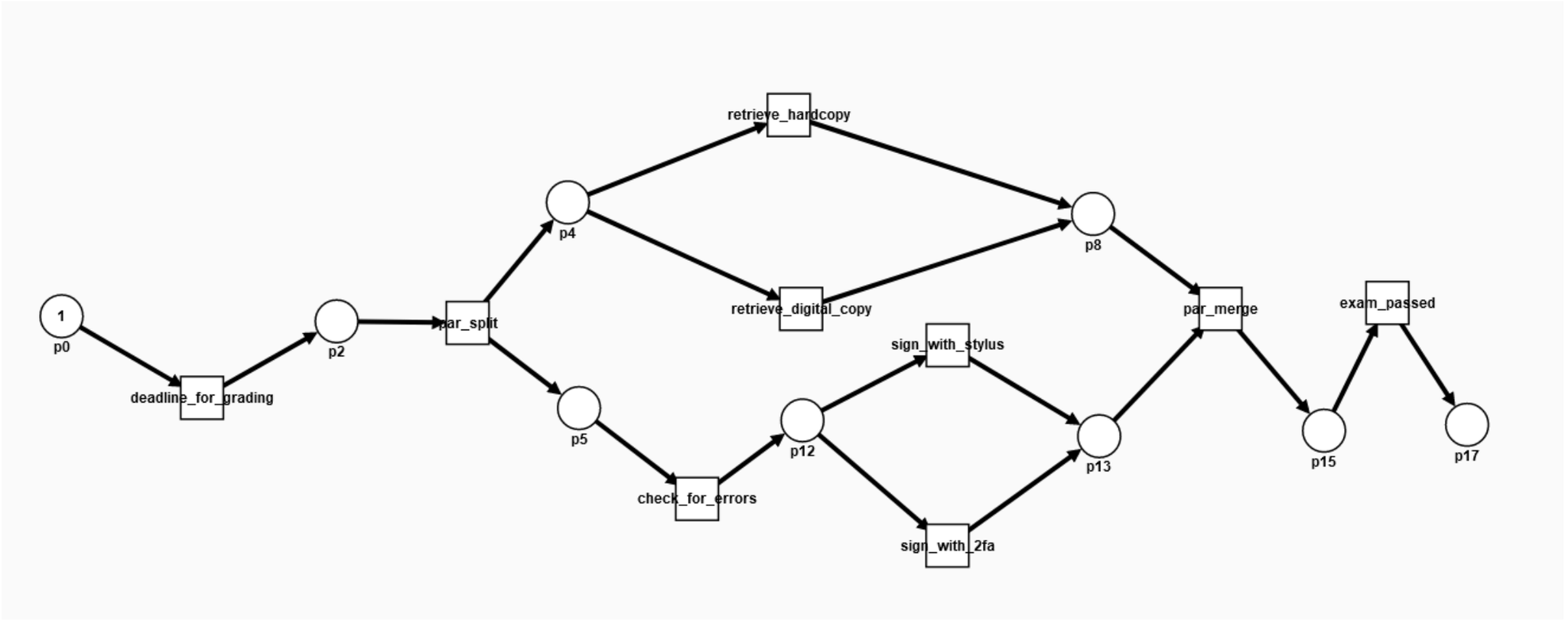
- Is the grade registration process going to terminate?
- Can all tasks be executed?

However, we can translate the ArchiMate model into a Workflow Net, which will be used for verification

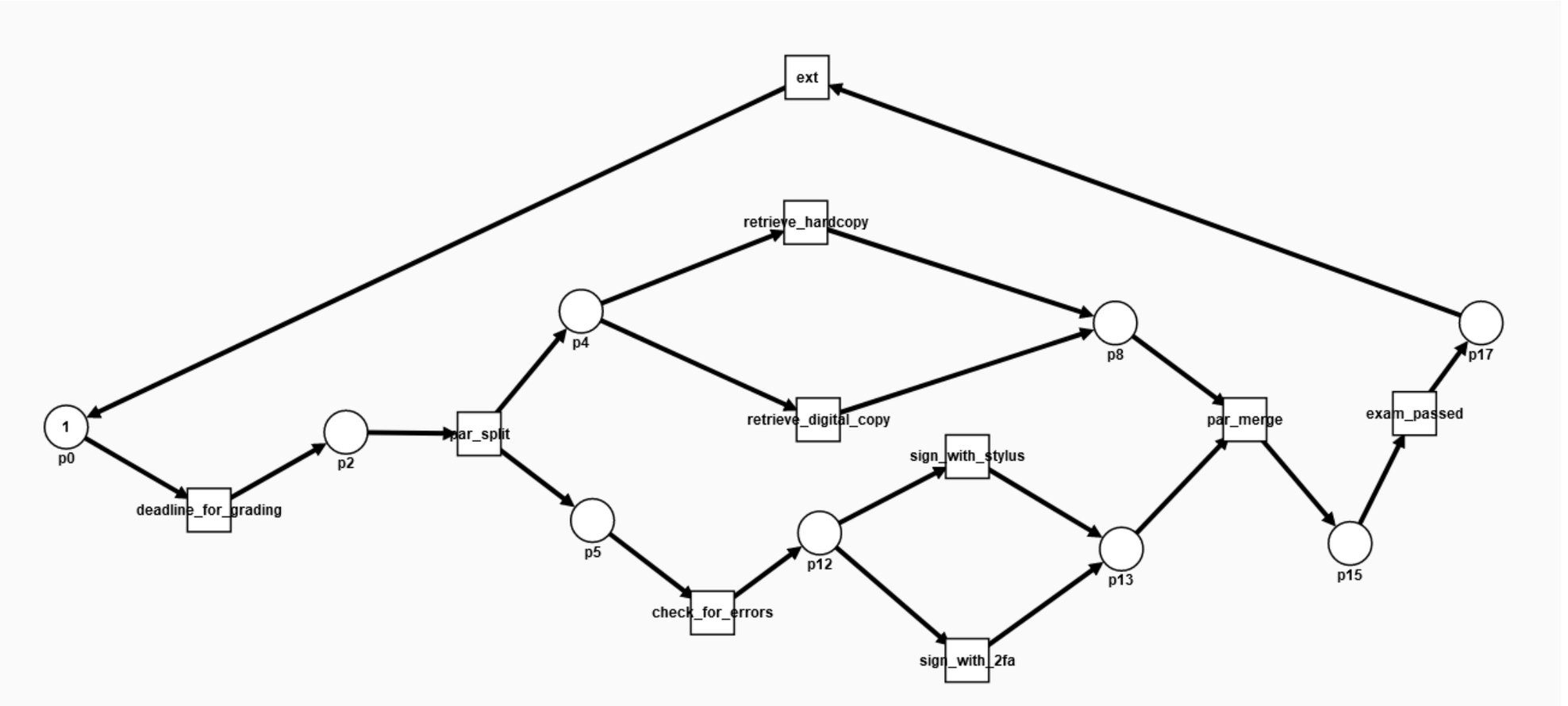
Translation to Workflow Net



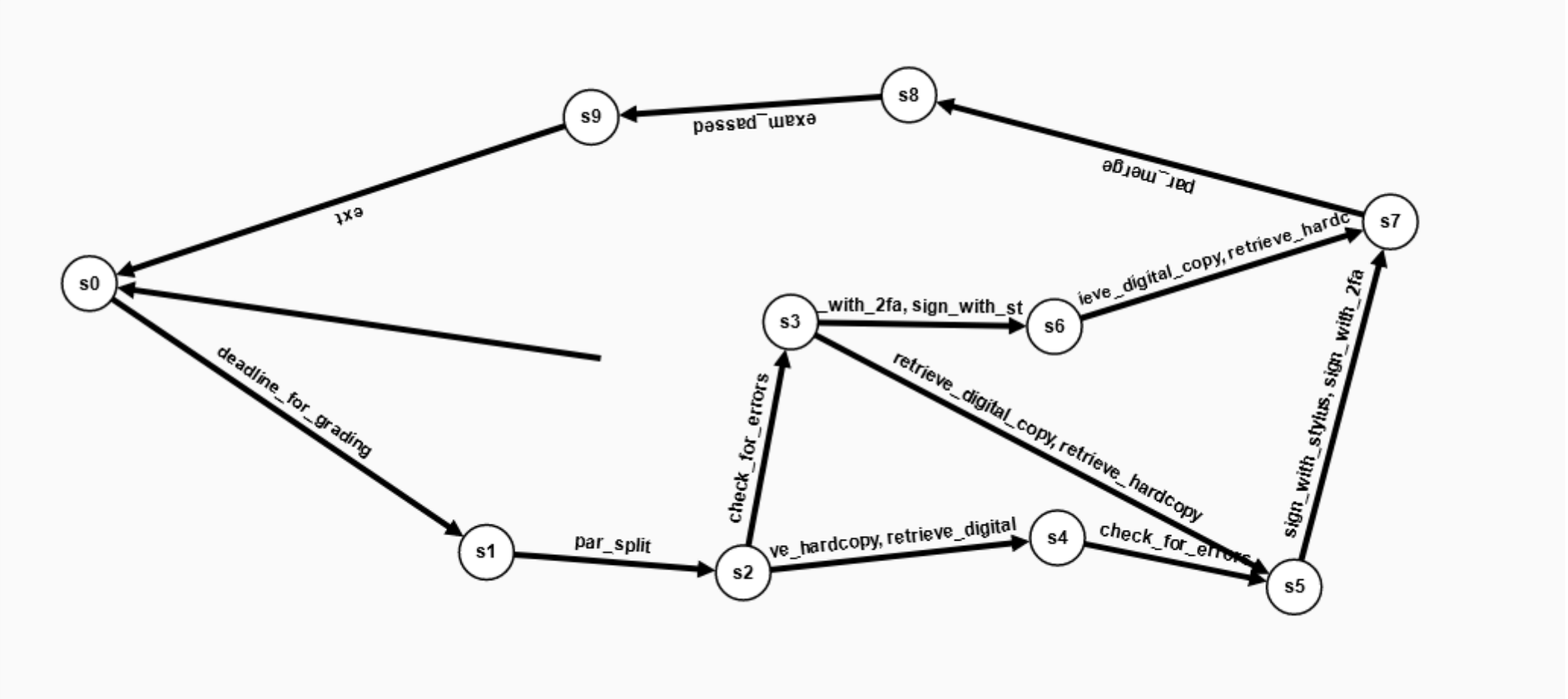
Equivalent Workflow Net



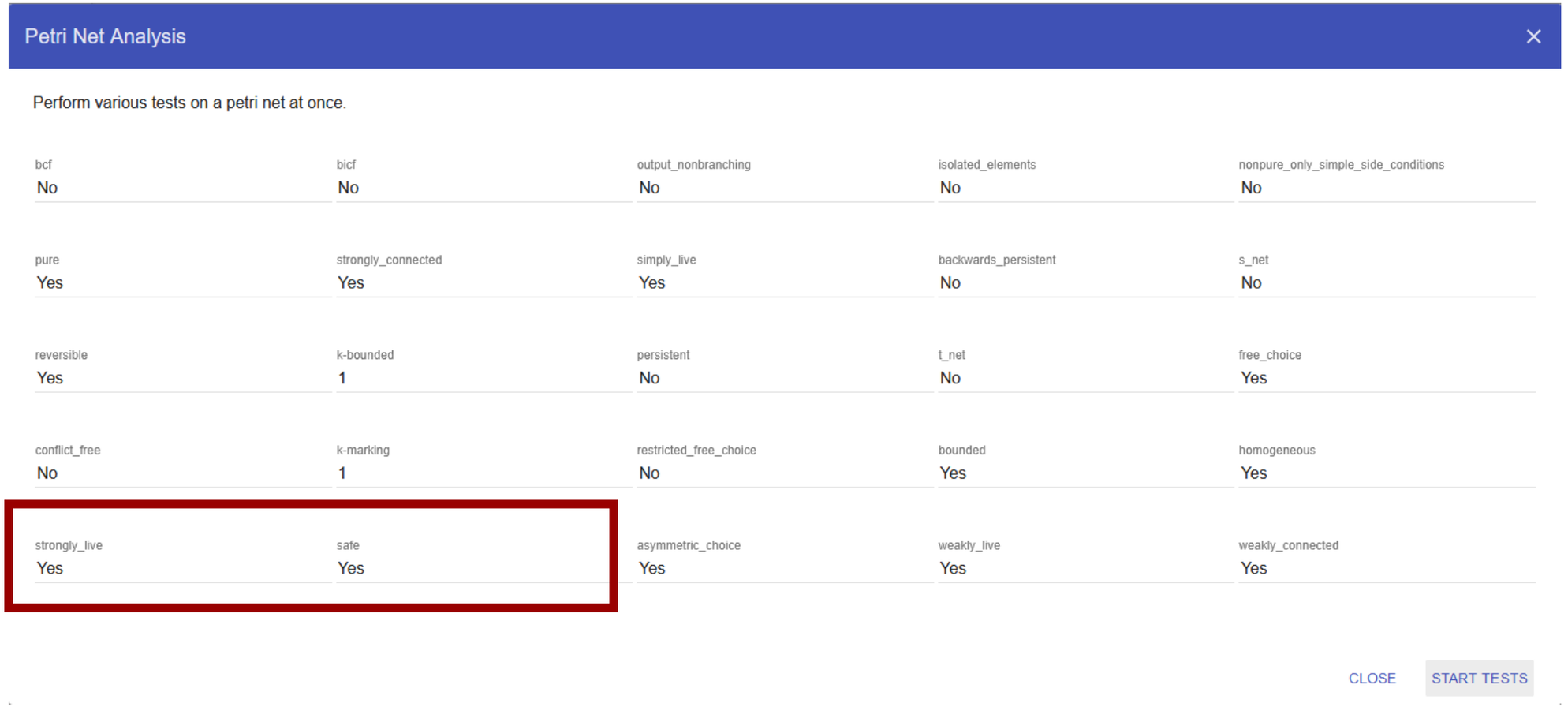
Extended Workflow Net



Reachability graph

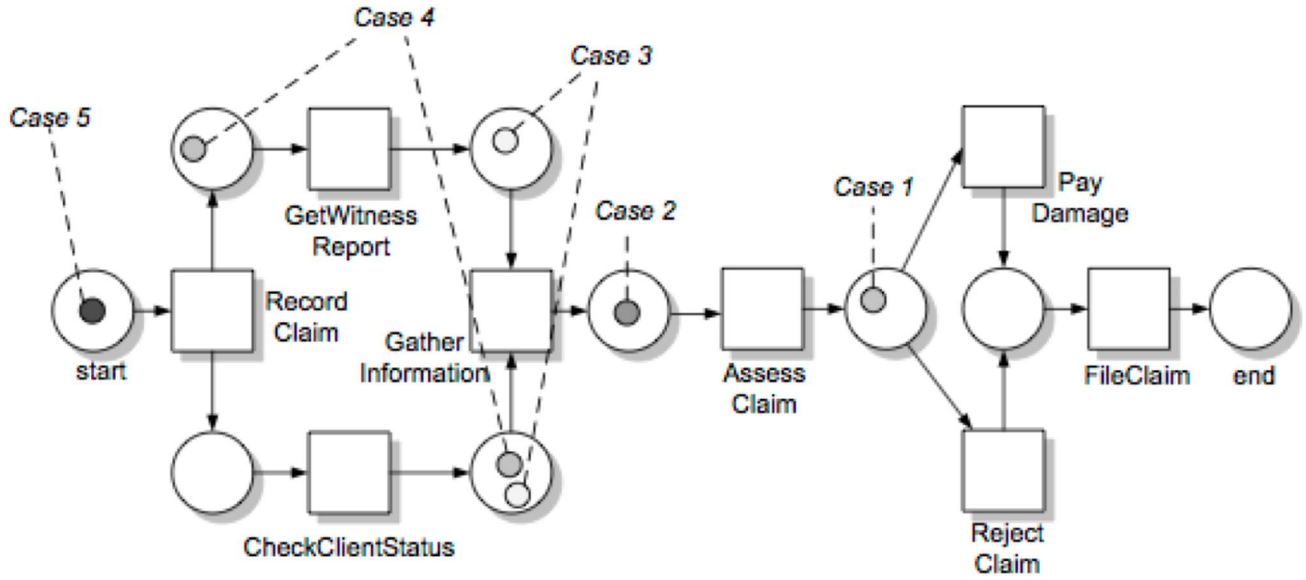


Analysis results



Managing multiple instances

- Classical Petri Nets are suitable to manage only one instance
- All the tokens have the same “meaning”
- Coloured Petri Nets are used to manage multi-instance processes
- Can also model other properties: resources, data, etc.



Weske, Business Process Management:
Concepts, Languages, and Architectures (2nd ed.), Springer, 2012

Study material

- Books and articles:
  - Van der Aalst et al. - Workflow Management: Models, methods and systems
  - Available at: https://pure.tue.nl/ws/files/2456322/543561.pdf
  - Chapter 2.2, 2.3 (not 2.3.3), 4.1 to 4.3, A.1 to A.3
- Modeling tools:
  - APO: https://apo.adrian-jagusch.de/#!/Sample%20Net
  - WoPeD: https://github.com/woped/WoPeD/releases

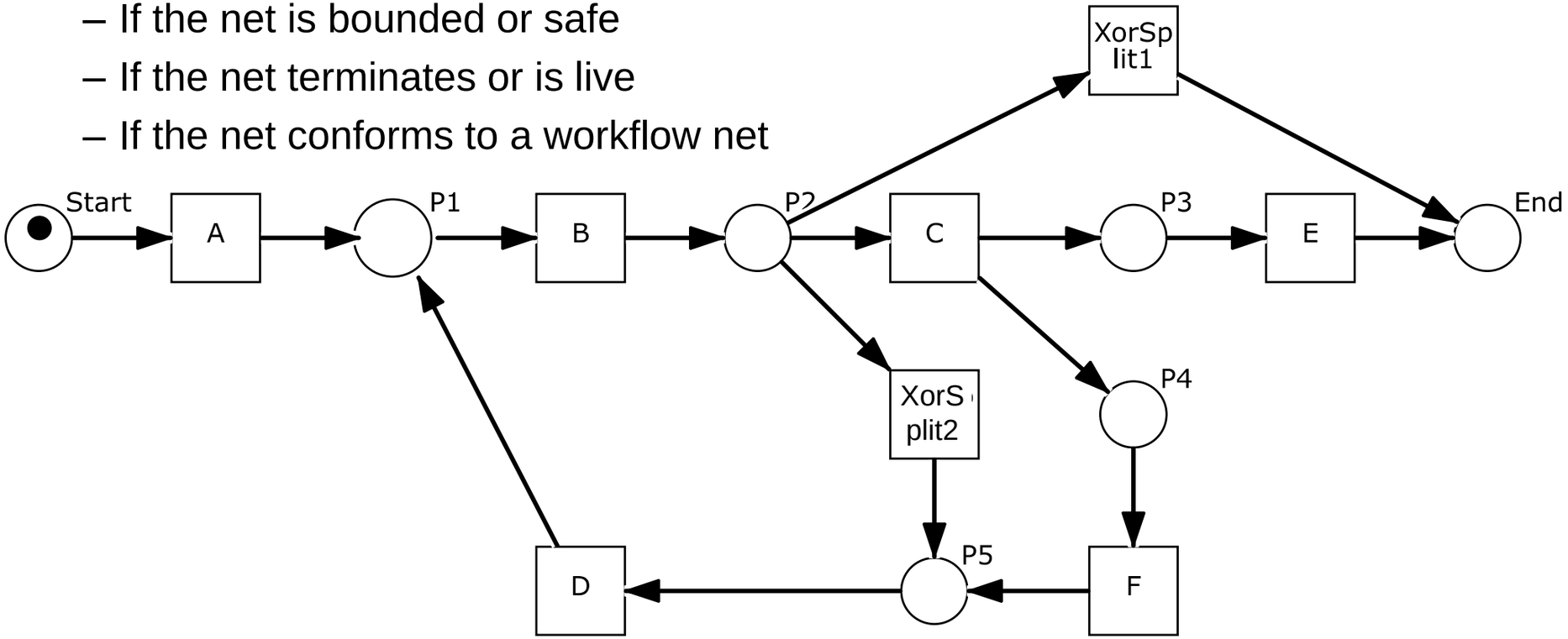
Exercises

Please answer all exercises to demonstrate your skills.

Solutions will be available at 11:45

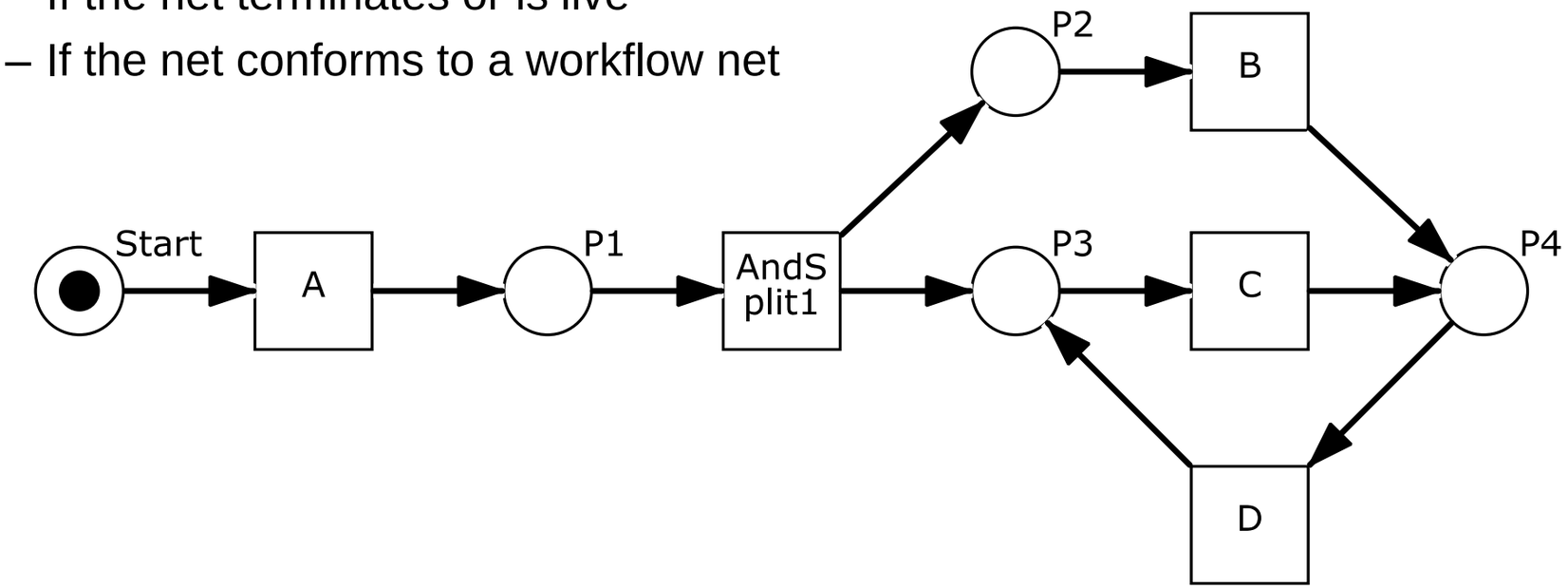
Exercise 1

- Given the following Petri Net, compute:
  - P, T, F sets
  - The reachability graph
  - If the net is bounded or safe
  - If the net terminates or is live
  - If the net conforms to a workflow net



Exercise 2

- Given the following Petri Net, compute:
  - P, T, F sets
  - The reachability graph
  - If the net is bounded or safe
  - If the net terminates or is live
  - If the net conforms to a workflow net



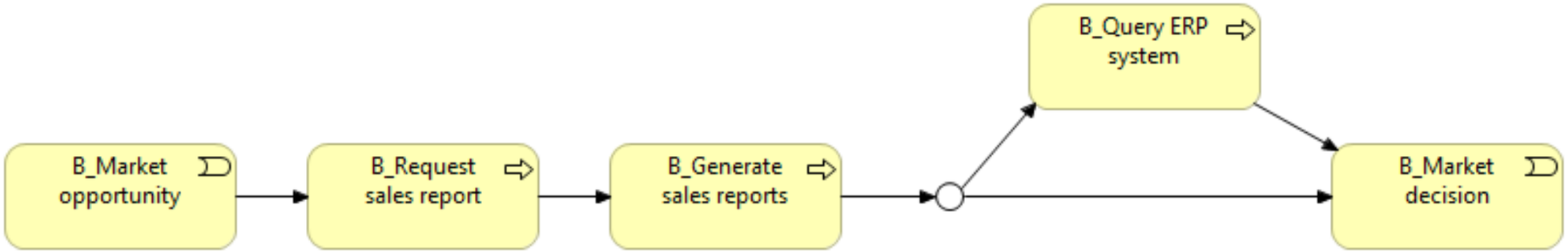
Exercise 3

Speedy is a delivery company that wants to create a new reporting system for the top management. After inspecting the sales reports, the top management may also need to query the existing ERP system, based on Oracle Fusion, to get detailed sales and HR information.

An ArchiMate model representing the process is enclosed below.

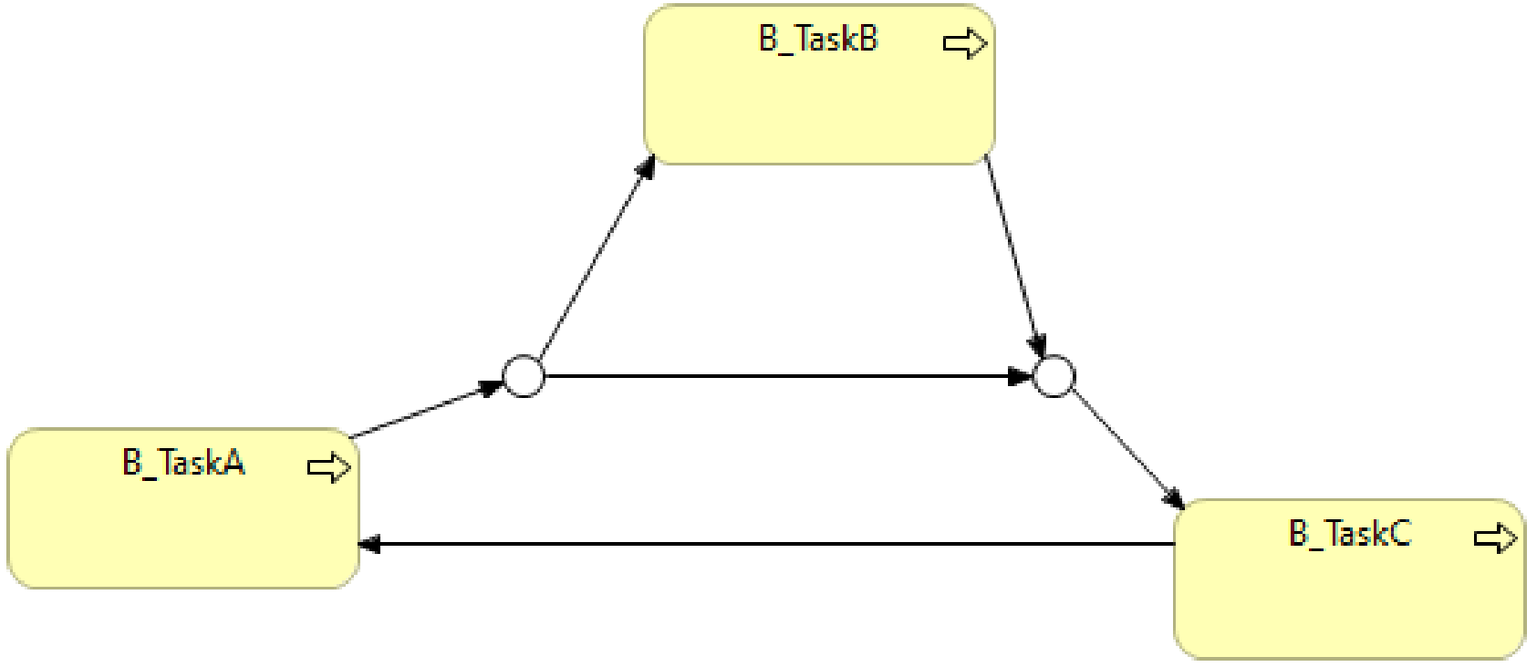
Starting from this model, create an equivalent Petri Net and try to answer the following questions:

- What is the reachability graph of the net?
- Is the net sound?



Exercise 4

- Translate the following process model into a Petri Net.
- Check if the resulting Petri Net is sound. If not, propose an action to repair the process, to make it sound.



Exercise 5

- Translate the following process model into a Petri Net.
- Check if the resulting Petri Net is sound. If not, propose an action to repair the process, to make it sound.