Modelling and reasoning with business processes and workflows

Verification of Workflow Nets

Jens Kohler, jkohler@hs-mannheim.de
Emanuele Storti, e.storti@univpm.it
Raffaele Dell'Aversana, r.dellaversana@gmail.com
Babak Bagheri Hariri, bagheri@inf.unibz.it
Emilio Sanfilippo, emiliosanfilippo@gmail.com

Mentor: Diego Calvanese

Outline of the Presentation

1. Introduction
    a. Definition of Workflow Systems
    b. Research problem
2. Approach
    a. Petri-Nets and their properties
    b. Workflow Nets
    c. Transformation Rules
3. Discussion

Reference: Wil M. P. van der Aalst: *Verification of Workflow Nets.* ICATPN 1997

Introduction

Workflow Management Systems: systems to

- define,
- create and
- manage the execution of workflows

(e.g., BPs in enterprises, Scientific Workflows in research projects)

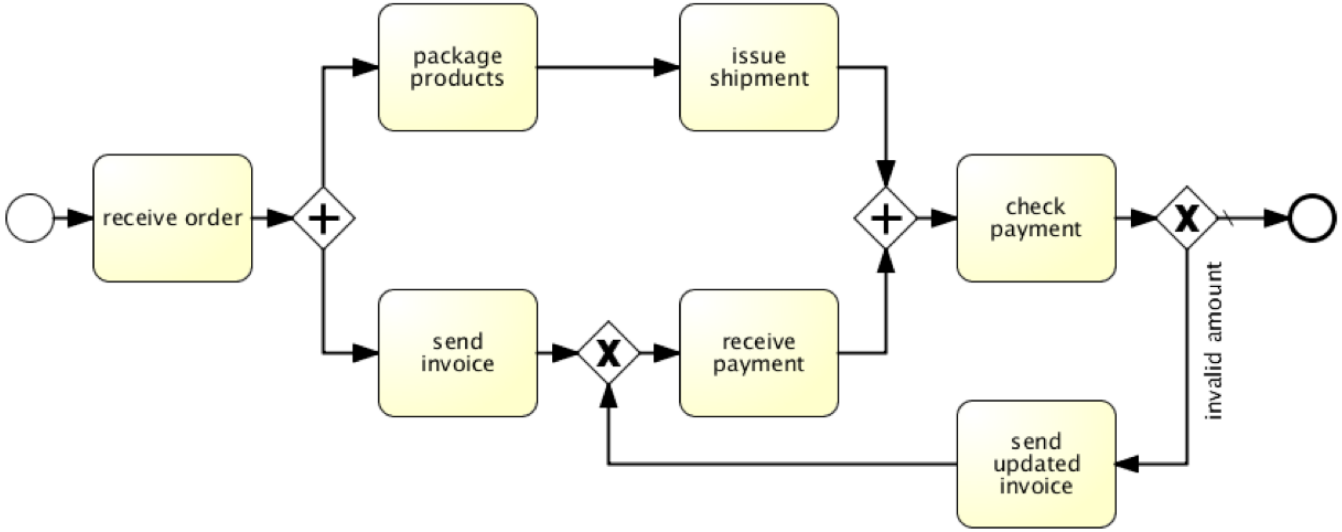
Extensively used in organizations to process ***cases*** (e.g., claims, orders) by linking **procedures** to resources:

- **procedure**: a partially ordered set of **tasks**, routed through **operators** (e.g., AND-split, OR-split, AND-join, OR-join)
- **resource**: the organization unit or the role in charge to execute a task
- **data**: information processed by the system

Workflow (WF) example

BPMN 2.0: Object Management Group (OMG) standard

- Widely adopted (77 compliant implementations)



Motivation:
Checking Correctness of WF

Efficient automated **analysis** of the **properties** of WFMSs required

Properties such as:

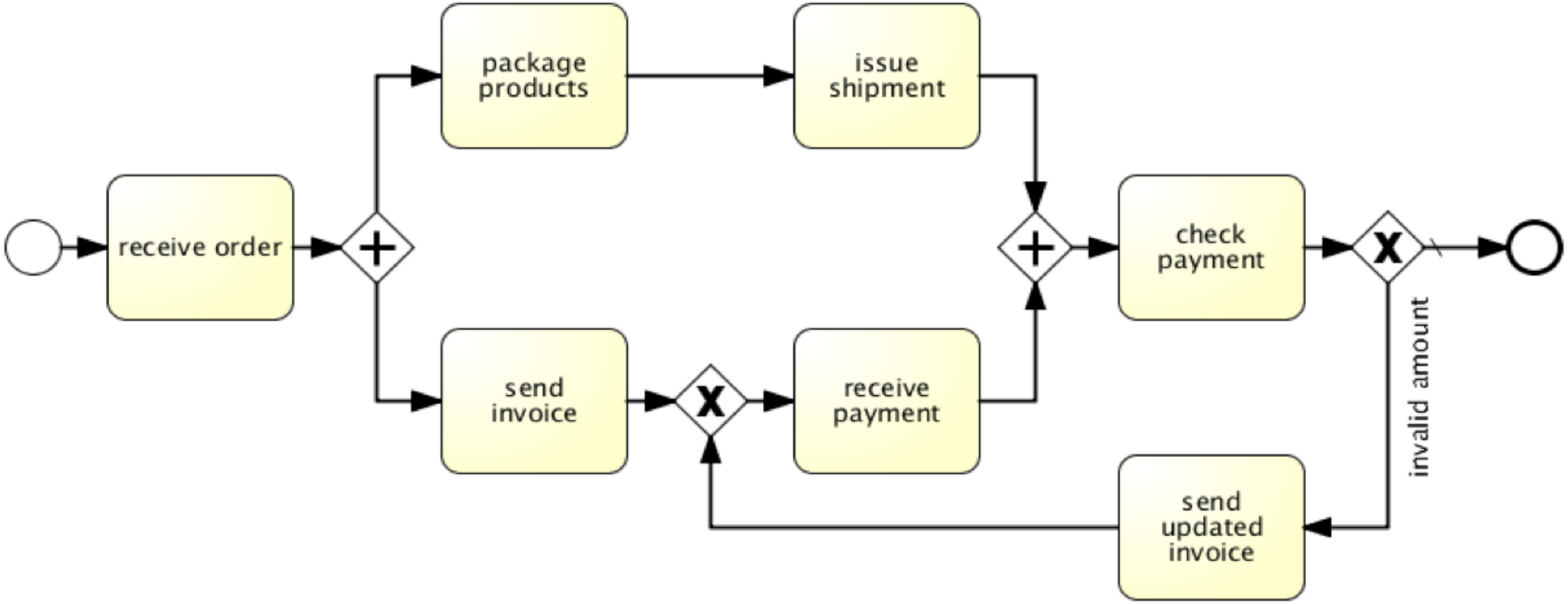
- Deadlock and Livelock free
- Boundedness
- Liveness
- Soundness

No theoretical foundation for the analysis of WFs, but for Petri Nets!

Solution:

Use **Petri Nets** for the **representation, validation and verification** of WFs

Example: BPMN Workflow - Deadlock



The Problem: Soundness of WFs

Check if the system can terminate properly in every state:

- the procedure will terminate eventually;
- After termination the system is in the appropriate final state!

Approach

Workflow procedures can be represented by Petri Nets:

- expressive enough to represent workflows
- well-known tools/techniques available for modeling, validation and verification

Solution:

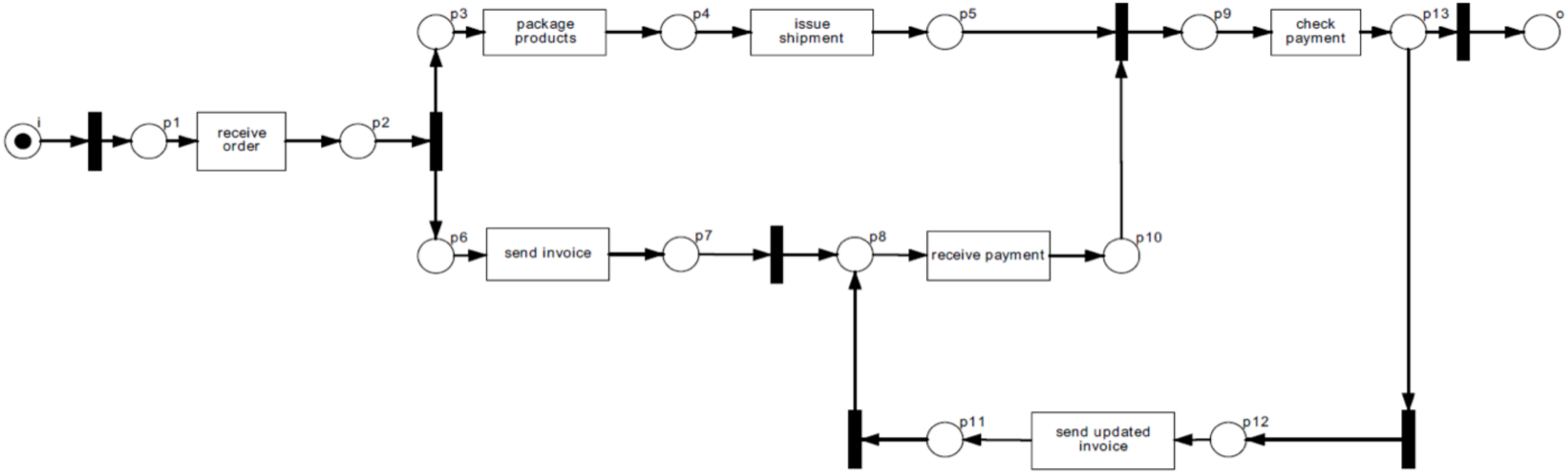
- using Workflow Nets, a class of Petri Nets, suitable for:
  - representation, validation of workflow procedures
  - verification of soundness

- definition of transformation rules to construct and modify procedures

Definition: Petri Net

Definition 1 (Petri net). A *Petri net* is a triple (P, T, F):

- P is a finite set of *places*,
- T is a finite set of *transitions* ($P \cap T = \emptyset$),
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (flow relation)

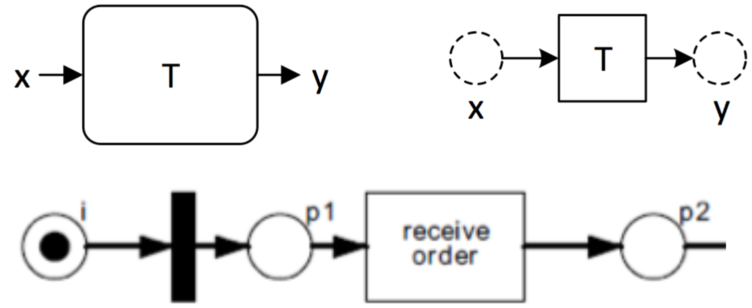


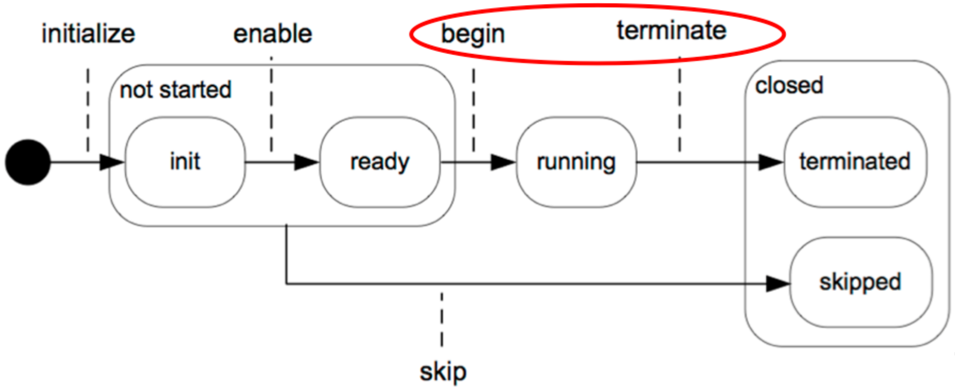
Definition: Petri Net - Transitions

Given a Petri net (P, T, F) and an initial state M_1, we have the following notations:

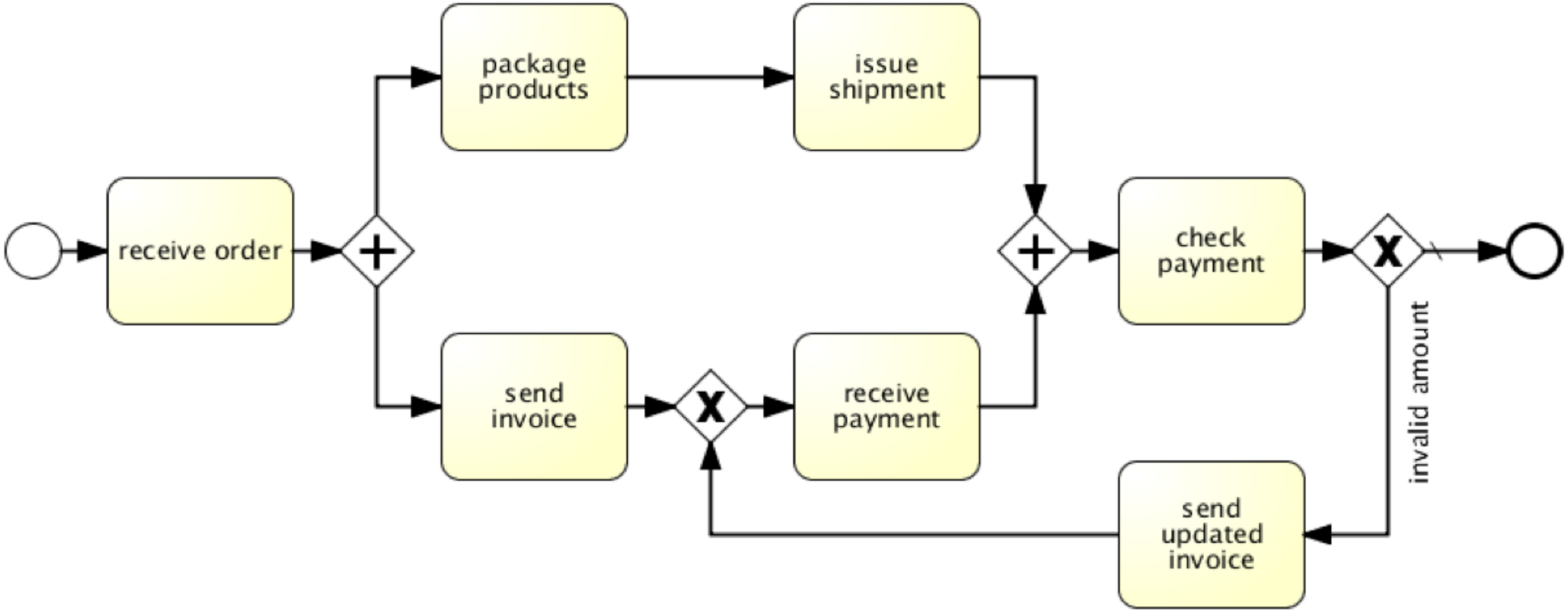
- $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2
- $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \xrightarrow{t} M_2$
- $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ leads from state M_1 to state M_n, i.e. $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$

A state M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) iff there is a firing sequence $\sigma = t_1 t_2 \ldots t_{n-1}$ such that $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$.

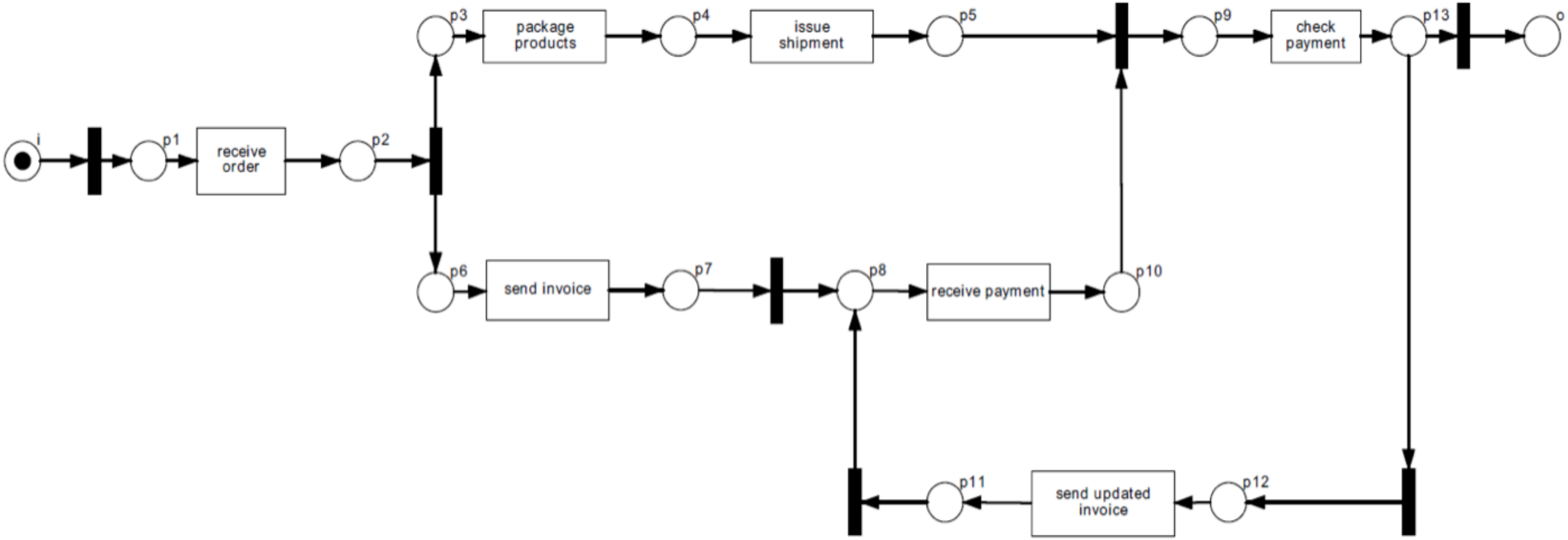




Example: BPMN Workflow - Deadlock

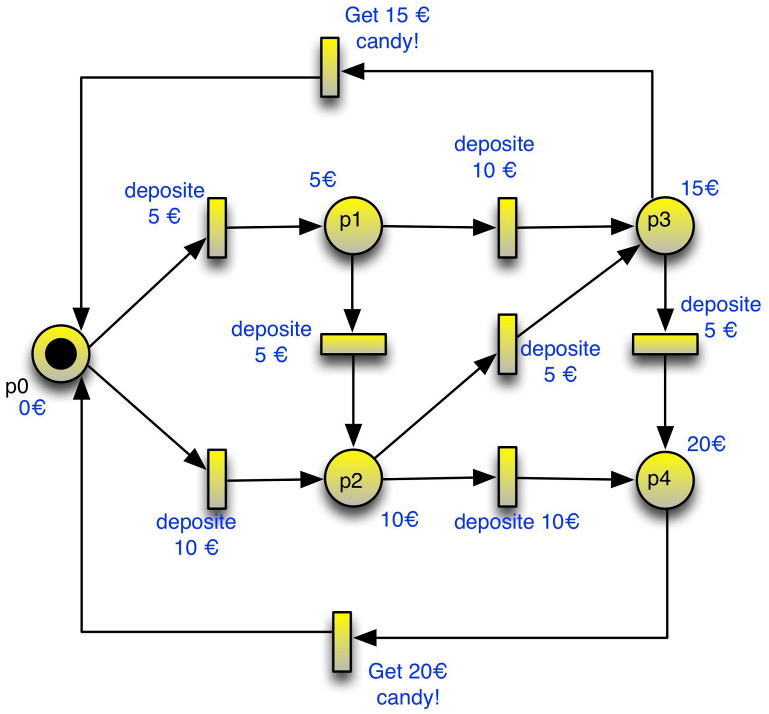


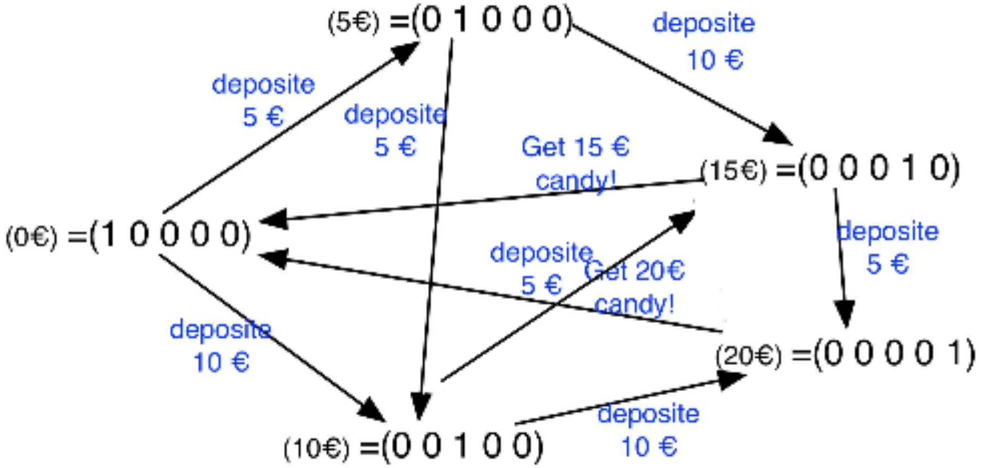
Example: Petri Net



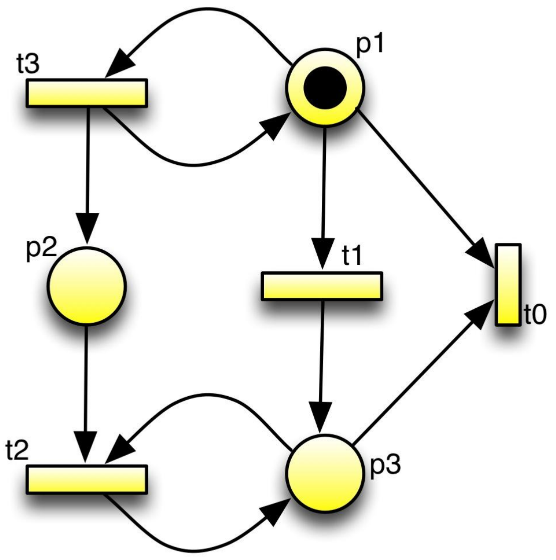
Bounded Petri nets

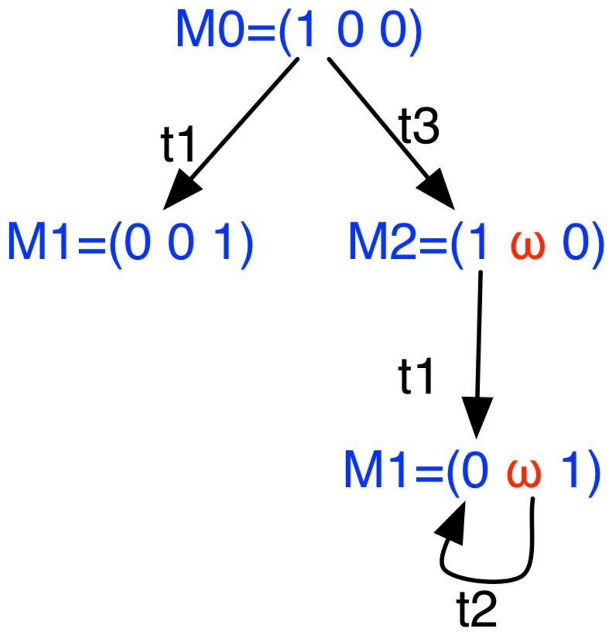
A Petri net is *bounded* iff, for every reachable state and every place *p* the number of tokens in *p* is bounded.





Unbounded Petri net!




M0=(1 0 0)

t1

t3

M1=(0 0 1)

M2=(1 ω 0)

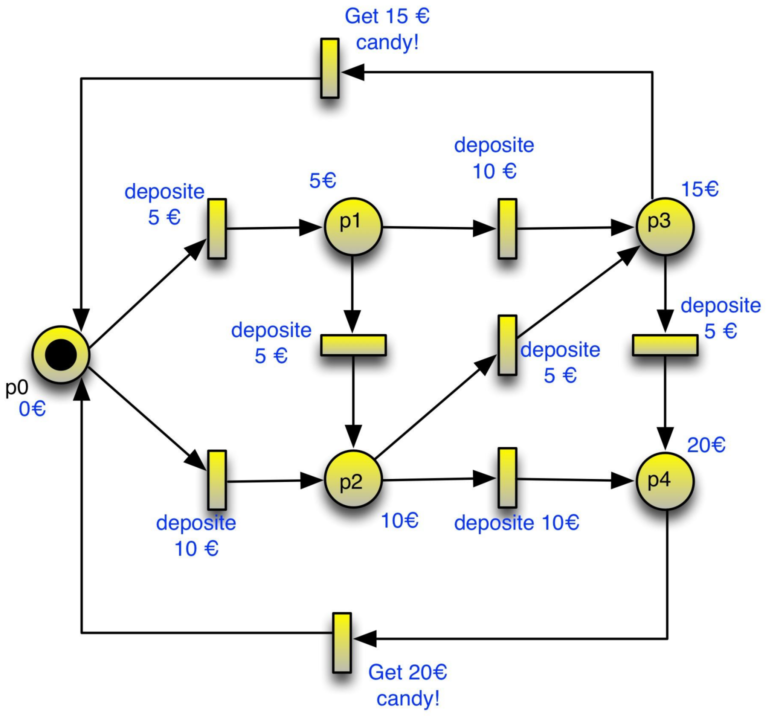
t1

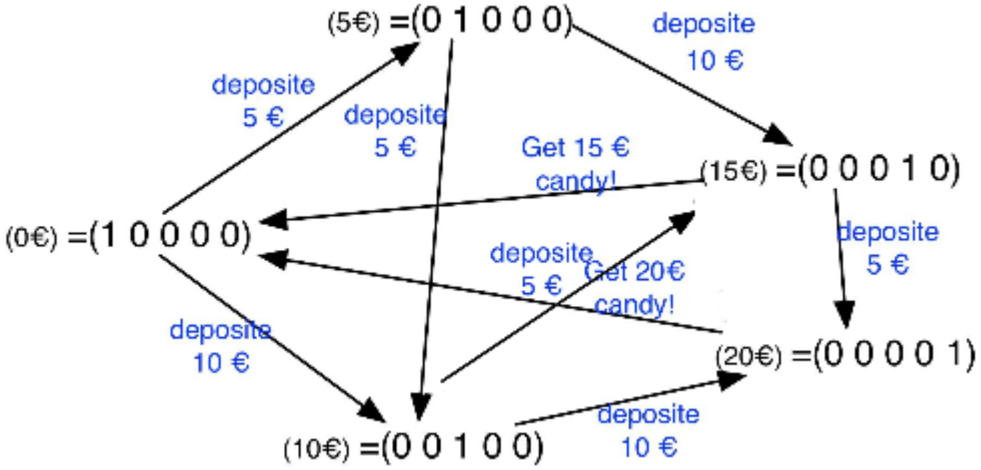
M1=(0 ω 1)

t2

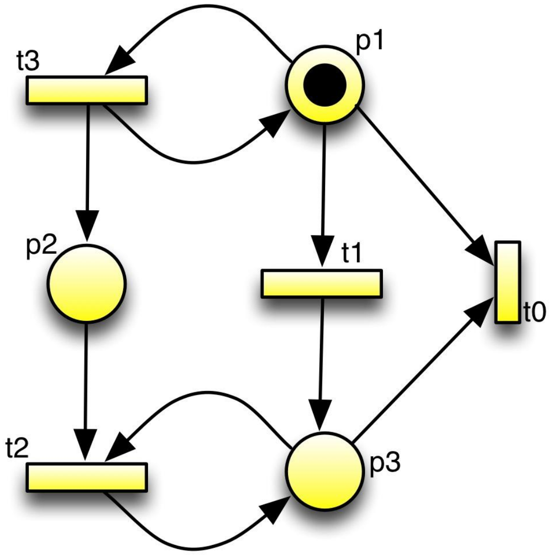
Live Petri nets

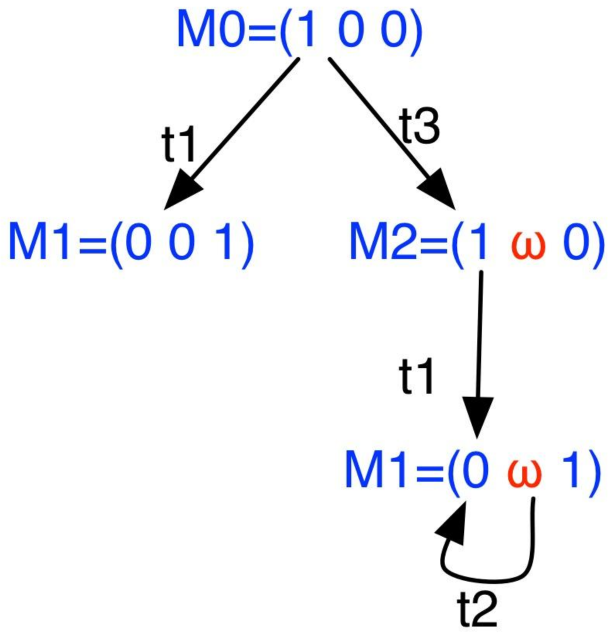
A Petri net is *live* iff, for every reachable state *M'* and every transition *t* there is a state *M''* reachable from *M'* which enables *t*.





Not live Petri net!




M0=(1 0 0)

t1

t3

M1=(0 0 1)

M2=(1 ω 0)

t1

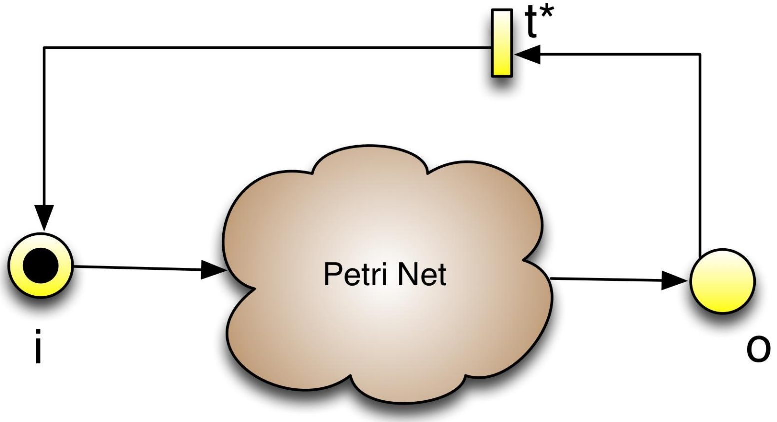
M1=(0 ω 1)

t2

Definition: WF-net (static)

A workflow-net is a petri net that:

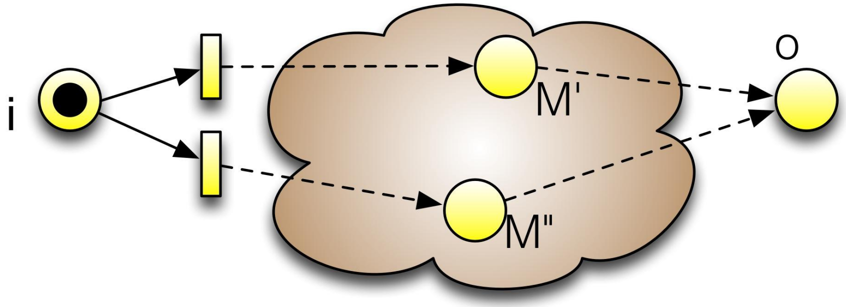
1. has a special place i with no input transitions.
2. has a special place o with no output transitions.
3. If we add a transition t from o to i the resulting petri-net is strongly connected.



Sound WF-nets (dynamic)

A WF-net is sound if and only if:

1. from any reachable state it is possible to reach a state with a token in the sink place (option to complete)
2. any reachable state having a token in the sink place does not have a token in any of the other places (proper completion)
3. for any transition there is a reachable state enabling it (absence of dead parts)



Coverability graph for soundness!

If the coverability graph of a petrinet has an ω edge, then it is not sound!

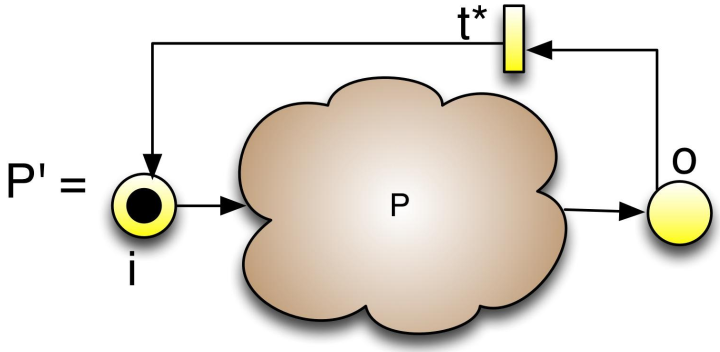
Otherwise, there is an easy algorithm to check the soundeness.

The complexity of construction of the coverability graph:

- WF-nets: primitive recursive space hard!
- Free choice Petri nets: EXPSPACE-hard.

However, in most of practical cases, the soundness can be checked in polynomial time!

Checking soundness



Lemma 1. If a WF-net P′ is live and bounded, then P is sound!
Lemma 2. If a WF-net P is sound, then P′ bounded!
Lemma 3. If a WF-net P is sound, then P′ is live!

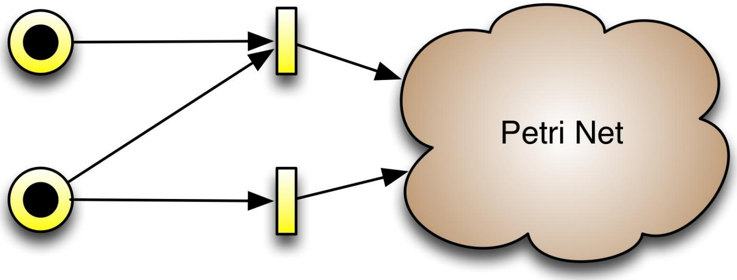
Theorem. A WF-net P is sound if and only if P′ is live and bounded.

Free Choice Petri nets

For every two places:

- either they do not have any common outgoing transitions, or
- they have the same set of outgoing transitions.

Example of non-free choice:



Free-choice WF-nets capture most of the models behind existing WFMSs.

Theorem. Checking soundness for free-choice WF-nets is in polynomial time.

Transformation Rules (TR)

Managing change:

- Changes in organization practices lead to modifications to BPs
- Model changing is error prone
- Previous approach is useful to check the soundness of the new procedure

An alternative approach is the usage of **Transformation Rules**:

- correspond to basic routing constructs identified by WFMS
- useful to modify a WF-net by preserving soundness

Transformation Rules (TR)

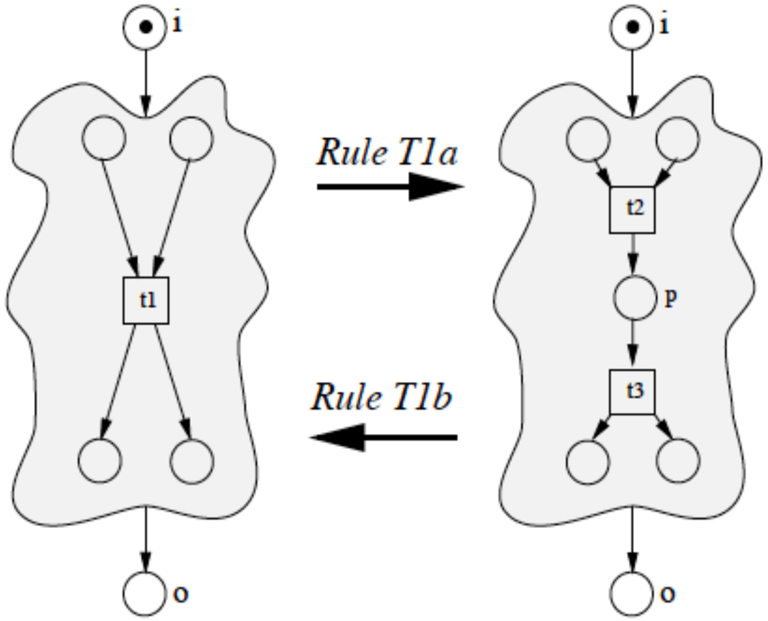
Eight basic TRs:

- T1a-T1b: division/aggregation
- T2a-T2b: specialization/generalization
- T3a-T3b: parallelization
- T4a-T4b: iteration

Transformation Rules (TR)

Transformation T1

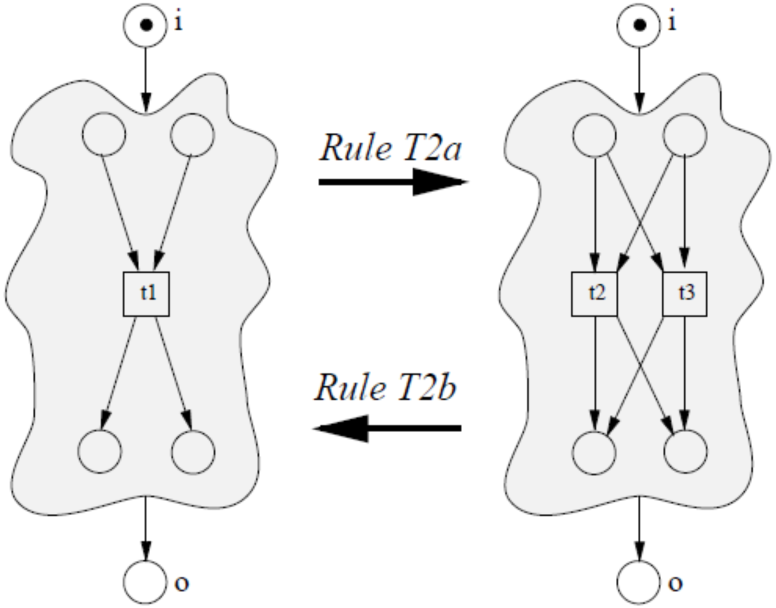
- **T1a (division):** Task t1 is replaced by two consecutive tasks t2 and t3. A complex task is divided into two tasks which are less complicated
- **T1b (aggregation):** Two consecutive tasks t2 and t3 are replaced by one task t1. Two tasks are combined into one task.



Transformation Rules (TR)

Transformation T2

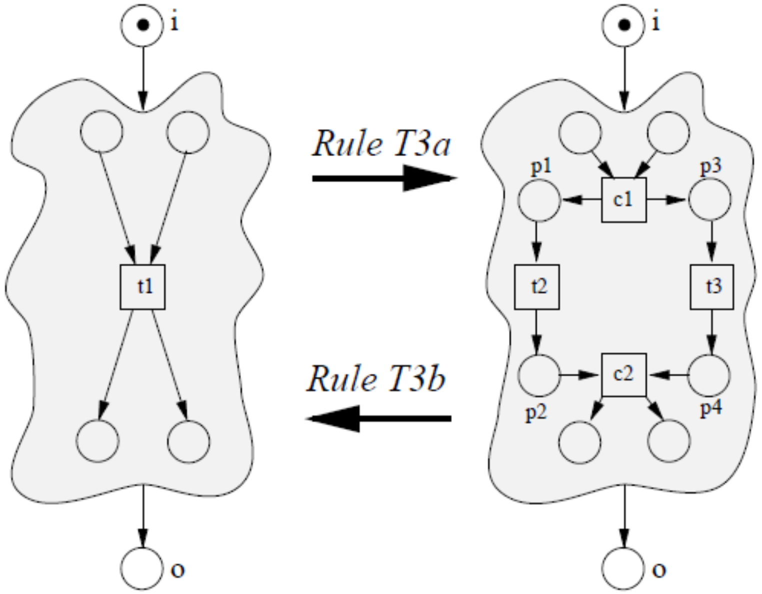
- **T2a (specialization):** Task t1 is replaced by two conditional tasks t2 and t3. One generic task is replaced by two more specialized tasks.
- **T2b (generalization):** Two conditional tasks t2 and t3 are replaced by one task t1. Two rather specific tasks are replaced by one more generic task.



Transformation Rules (TR)

Transformation T3

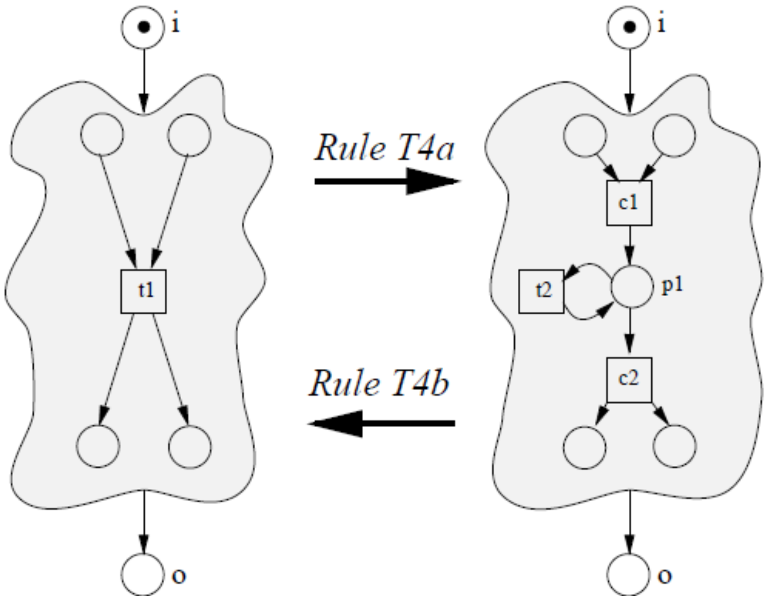
- **T3a (parallelization):** Task t1 is replaced by two parallel tasks t2 and t3 that achieve the same effect of the execution of t1. The transitions c1 and c2 represent control activities to fork and join two parallel threads.
- **T3b:** The opposite of transformation rule T3a: two parallel tasks t2 and t3 are replaced by one task t1.



Transformation Rules (TR)

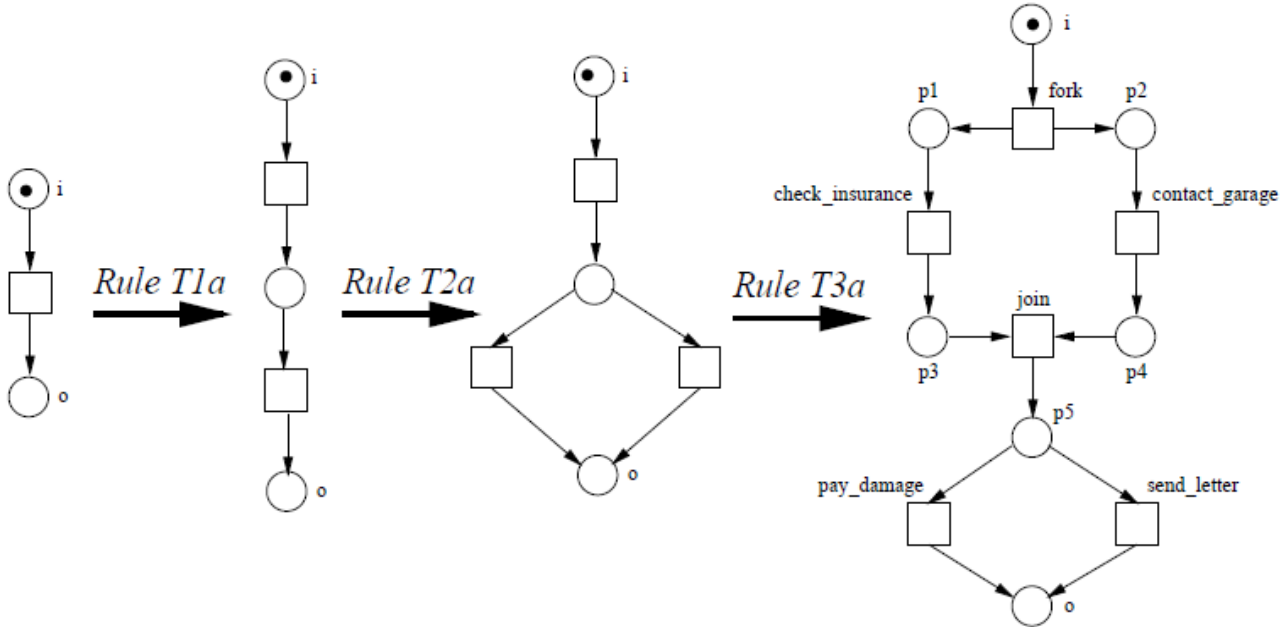
Transformation T4

- **T4a (iteration):** Task t1 is replaced by an iteration of task t2. The transitions c1 and c2 represent control activities that mark the begin and end of a sequence of ‘t2-tasks’. Typical examples of situations where iteration is required are quality control and communication.
- **T4b:** The opposite of transformation rule T4a: the iteration of t2 is replaced by task t1.



Transformation Rules (TR)

Theorem: The TRs preserve soundness, i.e. if a WF-net is sound, then the WF-net transformed by one of these rules is also sound



Advantages

+ formal semantics
+ graphical language
+ enough expressive to represent most workflow procedure
+ widely studied from a theoretical perspective
+ many tools and techniques available
+ WF system independent

Extensions

- Reasoning also about data, not only the control flow
  - The interconnection of data and process make the systems infinite-state
  - Most of the known techniques for verification of finite-states systems are not applicable!
- Reasoning with semantics (ontologies)
- Checking for properties related to domain knowledge


THANK
YOU
Obrigado
Grazie
Takk
Спасибо
Gracias
Danke
Paldies
Kiitos
Merci
Благодарам
Добре
Multumesc
Tesekkurler
Grazie
Gracias
Danke
Paldies
Takk
Obrigado
Kiitos

If a WF-net P′ is live and bounded, then P is sound!

Proper termination conditions:

P’ is live:

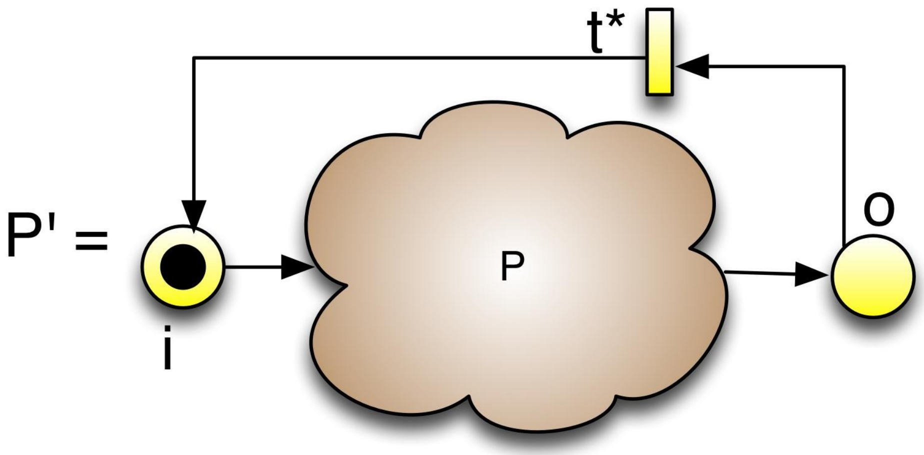
From any reachable state M, there is a reachable state M’ = M” + O, in which t* is fireable

i=>* M’

M’ => M” + i

P’ is bounded: M” should be finite.

i =>* o => i



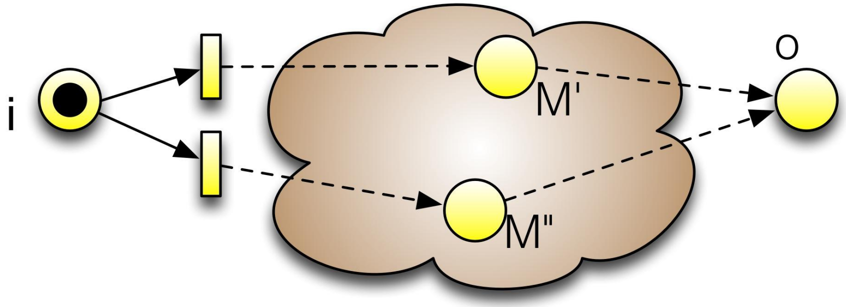
No dead transition:

P’ is live: P has no dead transition!

Sound WF-nets (dynamic)

Proper termination conditions:

1. For every state M reachable from state i, there exists a firing sequence leading from state M to state O.
2. State O is the only state reachable from state i with at least one token in place O.



No dead transition:

1. There are no dead transitions in the petri-net.