University of Pisa

MSc in Computer Engineering

PROCESS MINING

http://www.iet.unipi.it/m.cimino/wdis/

Mario G.C.A. Cimino

Department of Information Engineering

✓ The focus of process mining is to discover, monitor and improve **real** processes (i.e., not assumed processes) by extracting knowledge from **event logs** readily available in today's information systems.

✓ Process mining techniques includes:

a) automated **process discovery** (extracting process models from an event log)

b) **conformance checking** (monitoring deviations by comparing model and log)

c) **model enhancement** (model extension and repairing)

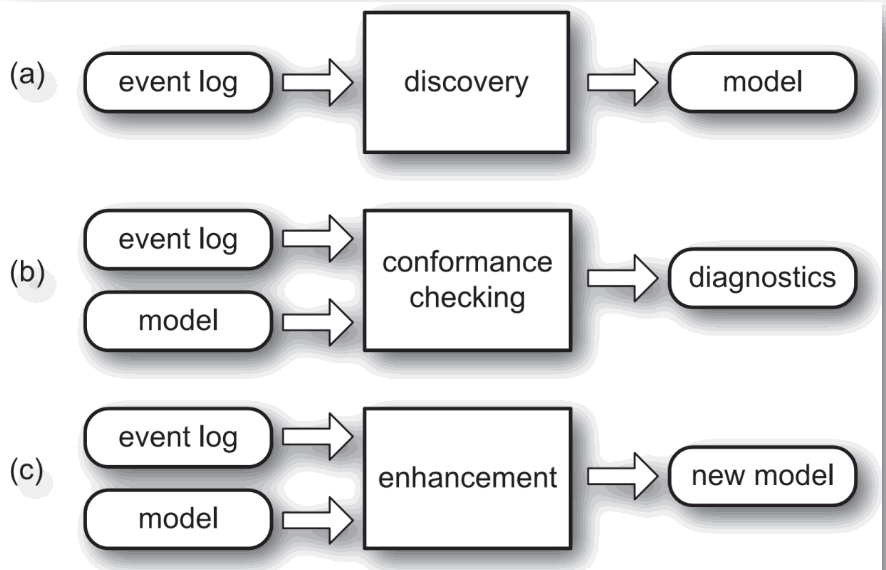


Figure : The three basic types of process mining explained in terms of input and output: (a) discovery, (b) conformance checking, and (c) enhancement.

✓ Drivers: growth of digital world + continuous process changes over time

✓ Process models are expressed via process notation: Petri net, causal nets, process trees, EPCs, BPMN, or UML activity diagrams.

✓ **BPMN 2.0** is a de-facto standard for modeling.

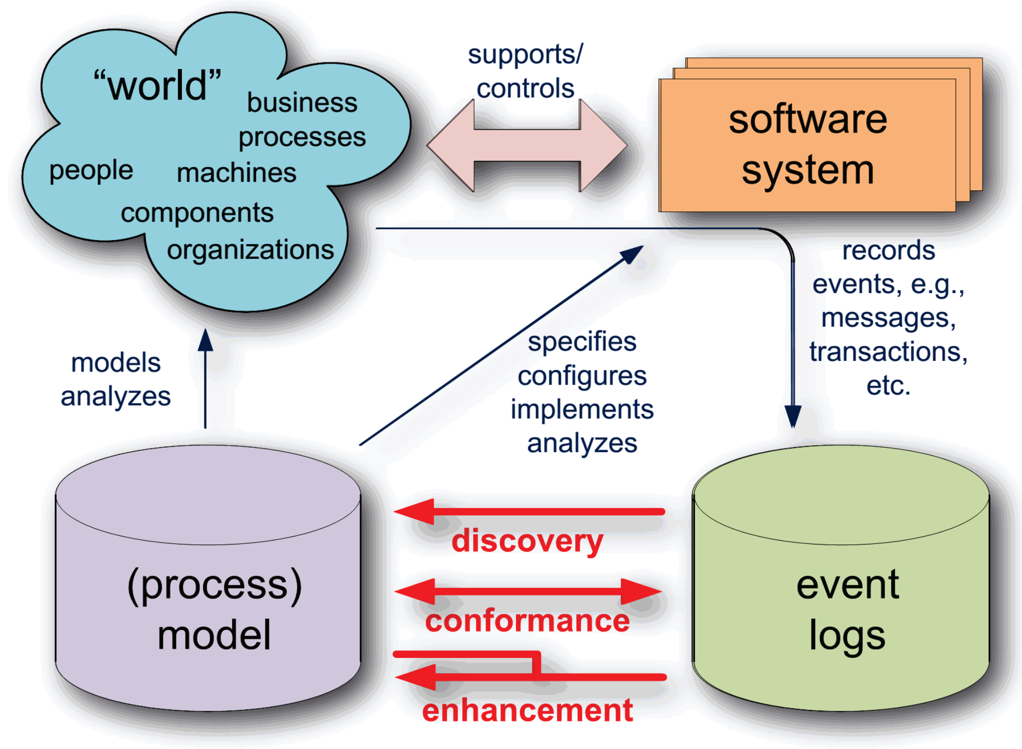


Figure: Positioning of the three main types of process mining: (a) *discovery*, (b) *conformance* checking, and (c) *enhancement*.

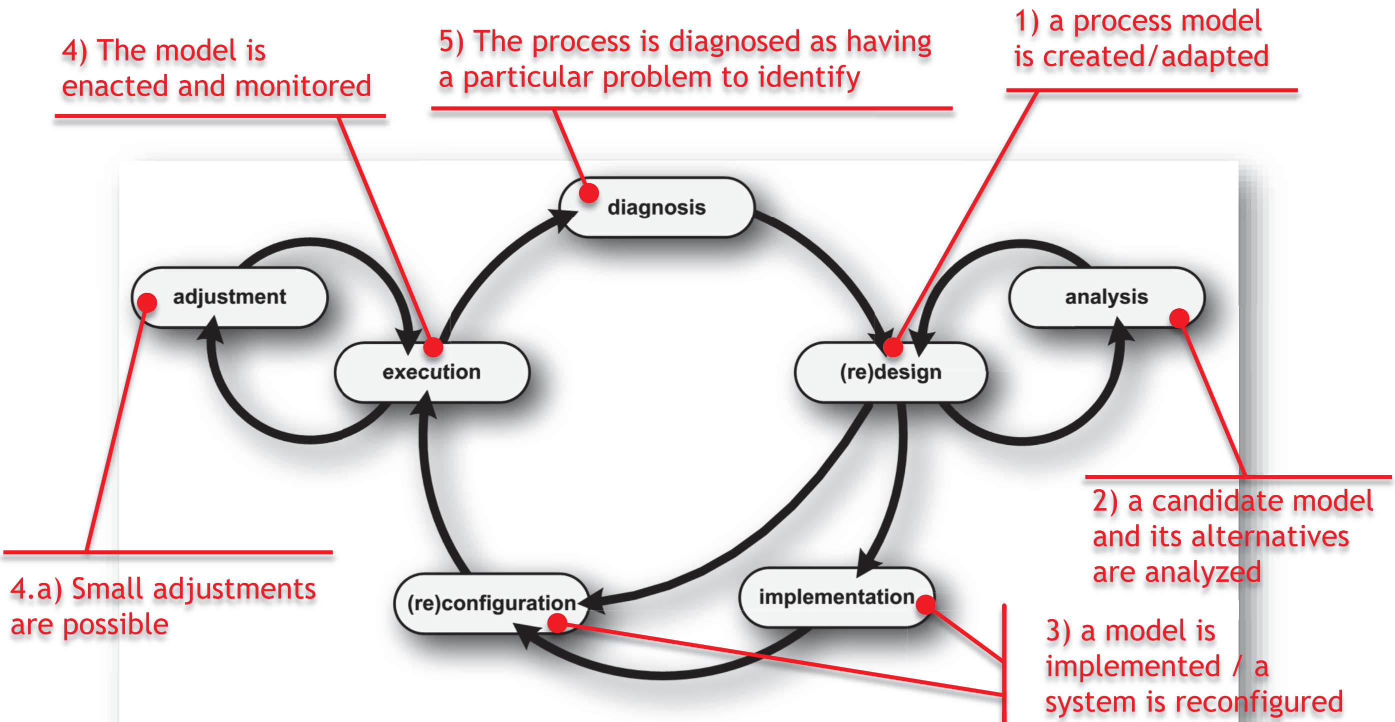
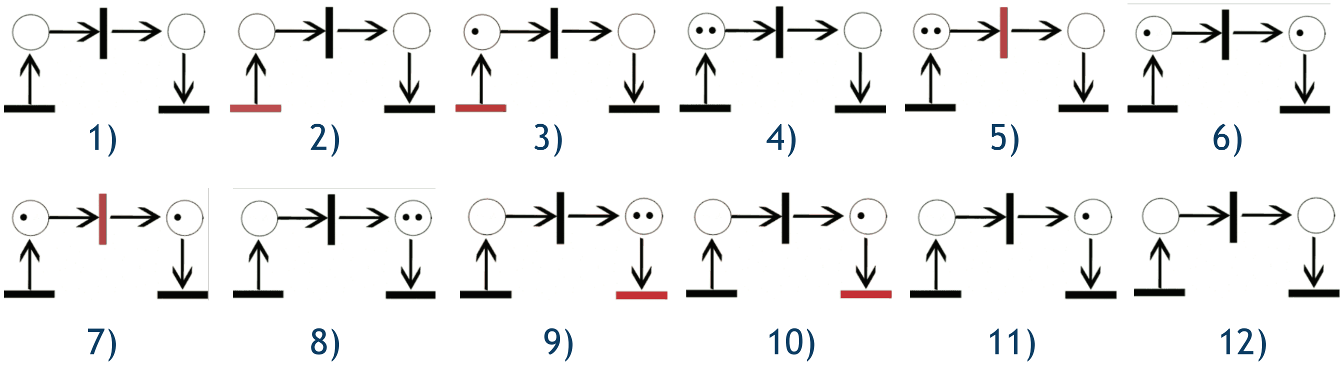


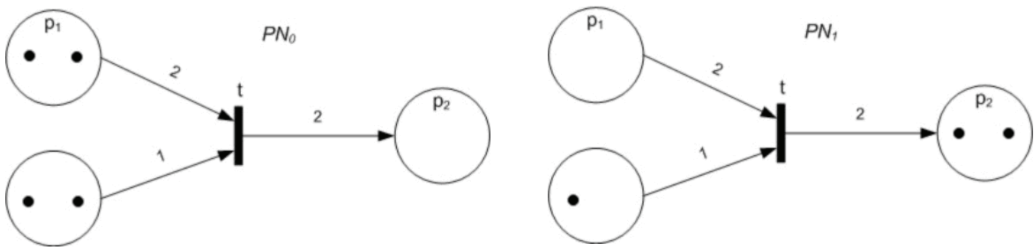
Figure: The BPM life-cycle identifying the various phases of a business process and its corresponding information system(s); process mining (potentially) plays a role in all phases (except for the implementation phase).

● **Petri net**, a mathematical modeling language for describing distributed systems. It is a bipartite graph in which nodes represents **transitions** and **places**, connected by **arcs.**



● a **transition** is an event that may occur, and is represented by a bar
● a **place** is a condition and is represented by a circle
● **arcs** run from a place to a transition (input place) or vice versa (output place), never between places or between transitions
● places may contain a discrete number of marks called **tokens**. A distribution of tokens over the places represents a configuration (marking) of the net
● a transition is fired if it is enabled, i.e., there are sufficient tokens in all of its input places. When the transition fires it consumes the required input tokens and created tokens in its output places. A firing is atomic (single non interruptible step).
● Unless otherwise specified, when multiple transitions are enabled at the same time, any one of them may fire (non determinism).

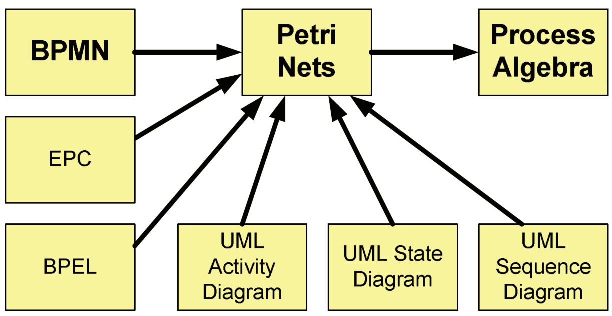
● In **pure petri nets** the behavior can be specified with **arc multiplicities**: a transition *t* is enabled at marking *m* if every input place *p* of *t* contains at least as many tokens as the multiplicity of the arc from *p* to *t* is.

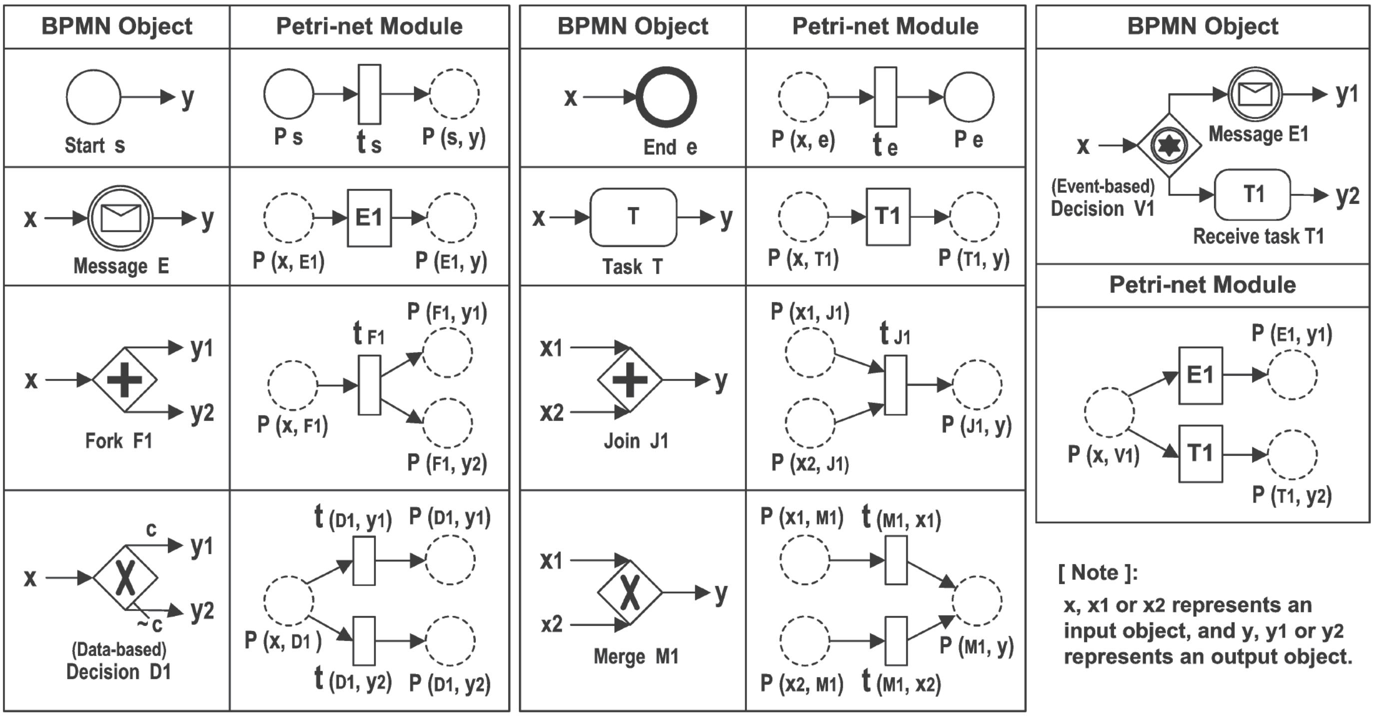


A Petri net with an enabled transition.

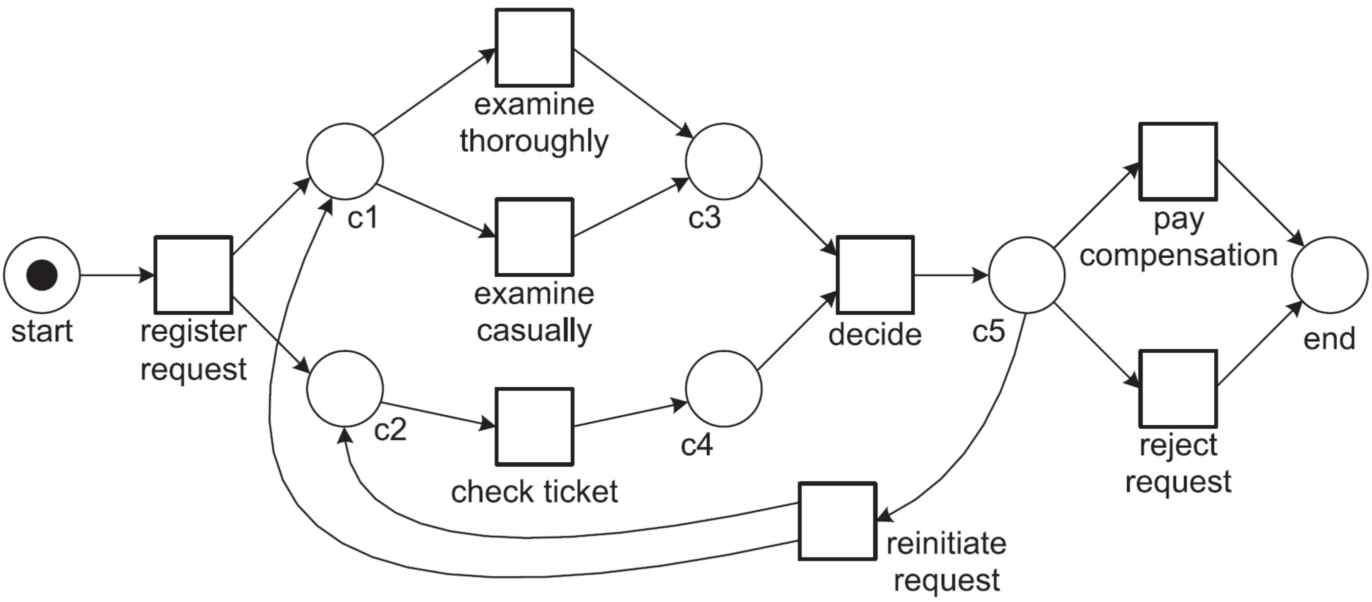
The Petri net that follows after the transition fires (Initial Petri net in the figure above).

● Extended Petri exist, e.g, coloured petri nets are a backward compatible extension allowing the distinction between tokens with data values attached.
● **Model transformations** provide a wide range of techniques for model analysis on business processes used in industry. **Process algebra** is a family of formal approaches used in computer science to model concurrent systems. It provides algebraic laws to manipulate, analyze and permit formal reasoning about processes, and are at the core of process software engines.

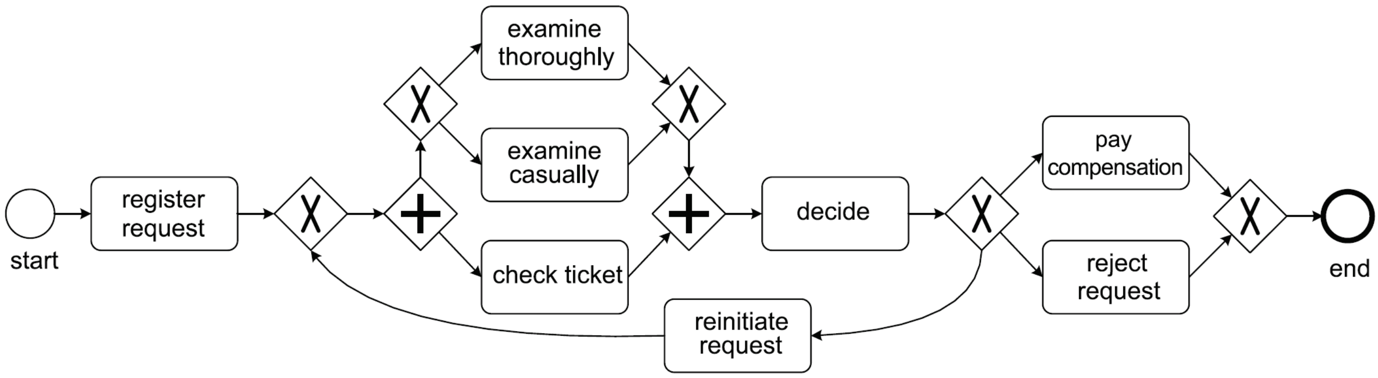




Mapping task, events, and gateways onto Petri-net modules



A Petri net modeling the handling of compensation requests



The same process modeled in terms of BPMN

PM is not limited to workflow, it covers different perspectives:

a) **control-flow perspective**: focuses on the ordering of activities, with the purpose of finding a good characterization of all possible paths.

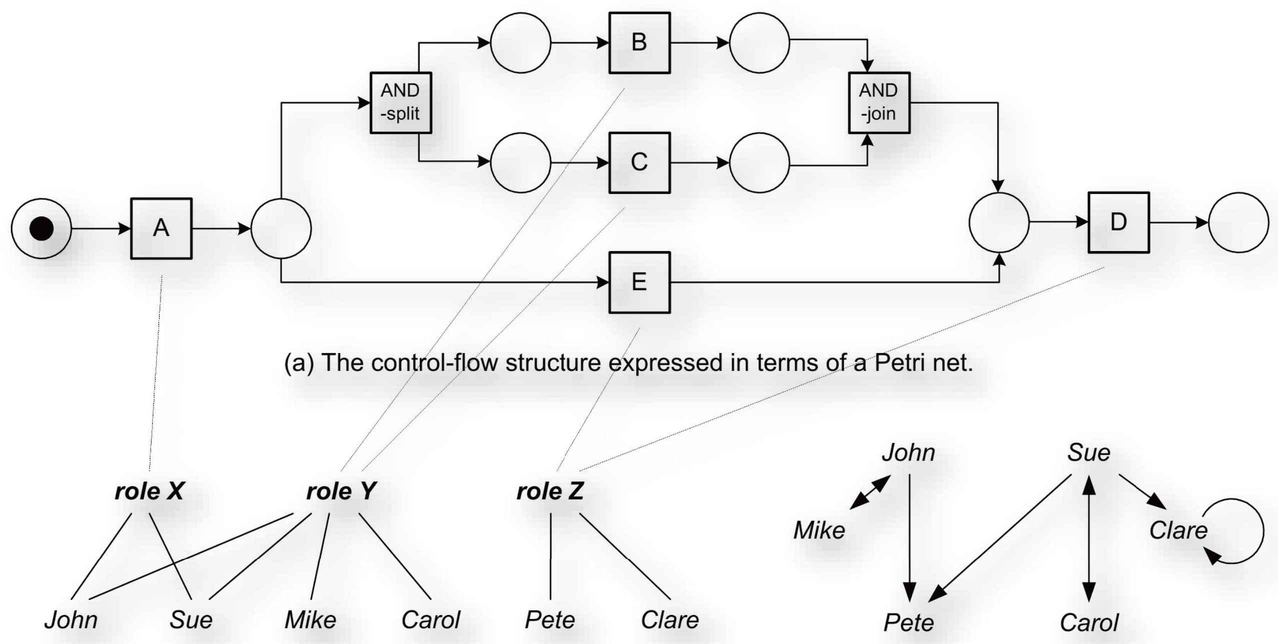
b) **organizational perspective:** focuses on resources, the goal is to either structure the organization by classifying people in terms of roles and organizational units or to show the business network.

c) **case perspective:** focuses on properties of cases: the case path in the process, the actors working on it, the values of the corresponding data objects.

d) **time perspective** is concerned with discovering bottlenecks, measuring service levels, monitoring resources utilization, predicting the remaining processing time of running cases.

An event log

case id	activity id	originator	time stamp
case 1	activity A	John	9-3-2004:15.01
case 2	activity A	John	9-3-2004:15.12
case 3	activity A	Sue	9-3-2004:16.03
case 3	activity B	Carol	9-3-2004:16.07
...	...	...	...



(a) The control-flow structure expressed in terms of a Petri net.

(b) The organizational structure expressed in terms of an activity-role-performer diagram.

(c) A sociogram based on transfer of work.

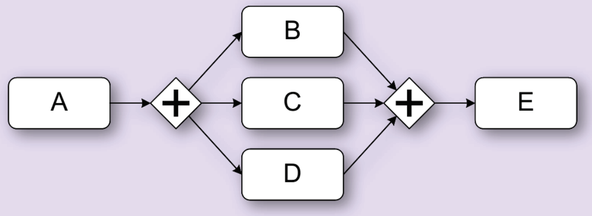
Fig. Some mining results for the process perspective (a) and organizational (b and c) perspective based on the event log shown.

- Starting point for PM is a collection of events **(events log)**, which may be stored in database tables, message logs, mail archives, transaction logs, and other data sources. More important than the storage is their **quality**.

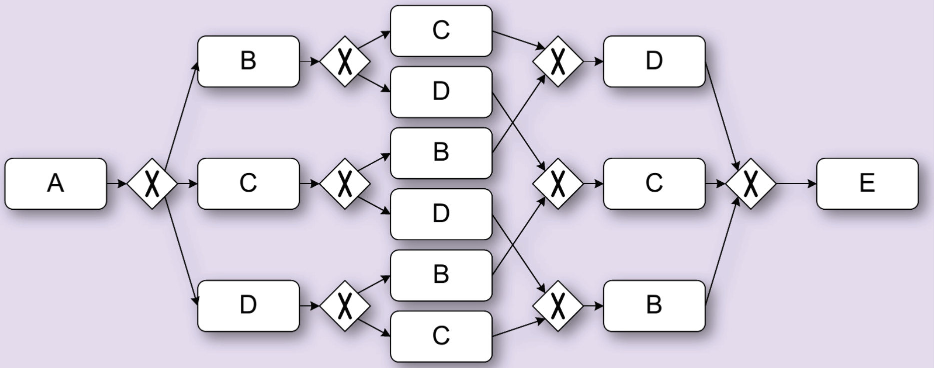
Level	Characterization	Examples
★★★★★	Highest level: the event log is of excellent quality (i.e., trustworthy and complete) and events are well-defined. Events are recorded in an automatic, systematic, reliable, and safe manner. Privacy and security considerations are addressed adequately. Moreover, the events recorded (and all of their attributes) have clear semantics. This implies the existence of one or more ontologies. Events and their attributes point to this ontology.	Semantically annotated logs of BPM systems.
★★★★	Events are recorded automatically and in a systematic and reliable manner, i.e., logs are trustworthy and complete. Unlike the systems operating at level ★★★, notions such as process instance (case) and activity are supported in an explicit manner.	Events logs of traditional BPM/ workflow systems.
★★★	Events are recorded automatically, but no systematic approach is followed to record events. However, unlike logs at level ★★, there is some level of guarantee that the events recorded match reality (i.e., the event log is trustworthy but not necessarily complete). Consider, for example, the events recorded by an ERP system. Although events need to be extracted from a variety of tables, the information can be assumed to be correct (e.g., it is safe to assume that a payment recorded by the ERP actually exists and vice versa).	Tables in ERP systems, event logs of CRM systems, transaction logs of messaging systems, event logs of high-tech systems, etc.
★★	Events are recorded automatically, i.e., as a by-product of some information system. Coverage varies, i.e., no systematic approach is followed to decide which events are recorded. Moreover, it is possible to bypass the information system. Hence, events may be missing or not recorded properly.	Event logs of document and product management systems, error logs of embedded systems, worksheets of service engineers, etc.
★	Lowest level: event logs are of poor quality. Recorded events may not correspond to reality and events may be missing. Event logs for which events are recorded by hand typically have such characteristics.	Trails left in paper documents routed through the organization ("yellow notes"), paper-based medical records, etc.

Table: Maturity levels for event logs.

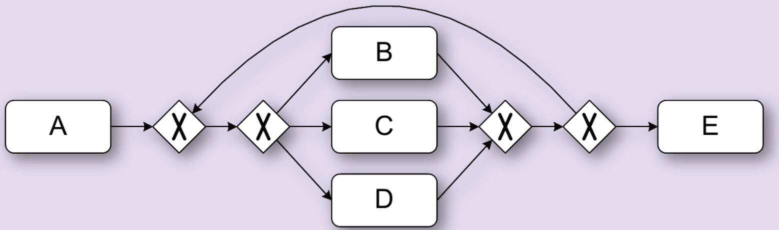
- Starting PM need to be driven by **questions**. Without concrete questions it is very difficult to extract meaningful events from tables of a database.
- As event logs contain only sample behavior, they should not be assumed to be complete. **Open world assumption**: the fact that something did not happen does not mean that it cannot happen.
- The **simplest** model that can explain the behavior seen in the log is the best model (**Occam's Razor principle**)
- Consider an event log L = {
<A, B, C, D, E>,
<A, B, D, C, E>,
<A, C, B, D, E>,
<A, C, D, B, E>,
<A, D, B, C, E>,
<A, D, C, B, E> }



B, C, and D can be executed in any order



B, C, and D can be executed in any order



B, C, and D can be executed in any order

- But cases such as <A, B, B, B, E> are possible according to the model but are not likely according to the event log.

- Noise and incompleteness make process discovery a challenging problem. There are four competing model quality dimensions:
(a) **fitness**: to allow for most of the behavior seen in the event log,
(b) **simplicity**: to allow for the most compact representation of the log
(c) **precision**: it does not allow for too much extra behavior w.r.t the log
(d) **generalization**: it does not restrict behavior to just the log

- A model that is not precise is "underfitting". A model that does not generalize is "overfitting".

- Balancing fitness, simplicity, precision and generalization is challenging. This is the reason that most of the more powerful process discovery techniques provide various parameters.

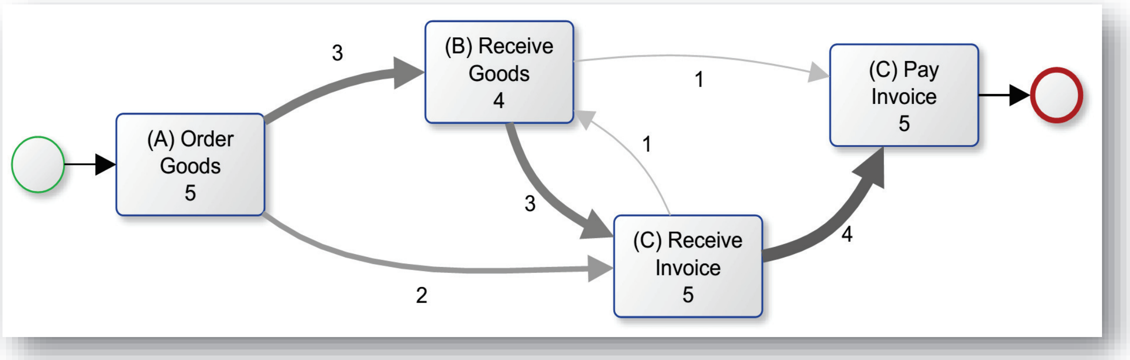
- Three important categories of mining algorithms:

i) **Deterministic** mining algorithm: it always delivers the same result for the same input. e.g. the alpha-algorithm (Van der Aalst et al. 2002).

ii) **Heuristic** mining algorithm: it also uses deterministic algorithms but they incorporate frequencies of events and traces for reconstructing a process model, disregarding infrequent paths in order to manage complexity.

iii) **Genetic** mining algorithm: it uses an evolutionary approach to find a satisfactory solution by iteratively selecting individuals and reproducing them by crossover and mutation over different generations.

E.g. **Heuristic** Fuzzy Miner algorithm (Günther and Van der Aalst 2007), which does not follow the BPMN notation but uses a **dependency graph:** it does not contain gateways but shows only dependencies between activities.



- **ProM**: a major academic tool providing many different mining algorithms, analysis, conversion and export modules.

Process Mining Tools

Product Name	Type[6]	Link
ARIS Process Performance Manager	C	www.softwareag.com
celonis business intelligence (SAP)	C	www.celonis.de
Disco	C	www.fluxicom.com
Genet/Petrify	O	www.lsi.upc.edu
Interstage Business Process Manager	C	www.fujitsu.com
QPR ProcessAnalysizer	C	www.gqr.com
ProM	O	www.processmining.org
ProcessGold	C	www.processgold.de
Rbminer/Dbminer	O	www.lsi.upc.edu
ReflectOne	C	www.pallas-athena.com
ServiceMosaic	O	soc.cse.unsw.edu.au

C = commercial, O = open source

- **Disco:** intuitive and usabile application. It provides integrated functionality for filtering and loading of event logs. It is especially suited for novel users. Non-commercial license is for academic institutions.

- Most used formats: **CSV**(comma separated values), **MXML** (Mining eXtensible Markup Language) and its successor, **XES** (eXtensible Event Stream)
- No support for the extraction of event data from source systems: data has to be extracted with specialized data extraction software or by using export functionalities of the source systems.

Process Discovery: introduction

• Event logs contain a set of events. A single event has a unique event ID, it refers to one individual case, it has a timestamp, and it shows which resources executed which task. It is a minimum requirement that the events refer to (i) one case, (ii) one task, and (iii) a point in time.

Case ID	Event ID	Timestamp	Activity	Resource
1	Ch-4680555556-1	2012-07-30 11:14	Check stock availability	SYS1
1	Re-5972222222-1	2012-07-30 14:20	Retrieve product from warehouse	Rick
1	Co-6319444444-1	2012-07-30 15:10	Confirm order	Chuck
1	Ge-6402777778-1	2012-07-30 15:22	Get shipping address	SYS2
1	Em-6555555556-1	2012-07-30 15:44	Emit invoice	SYS2
1	Re-4180555556-1	2012-08-04 10:02	Receive payment	SYS2
1	Sh-4659722222-1	2012-08-05 11:11	Ship product	Susi
1	Ar-3833333333-1	2012-08-06 09:12	Archive order	DMS
2	Ch-4055555556-2	2012-08-01 09:44	Check stock availability	SYS1
2	Ch-4208333333-2	2012-08-01 10:06	Check materials availability	SYS1
2	Re-4666666667-2	2012-08-01 11:12	Request raw materials	Ringo
2	Ob-3263888889-2	2012-08-03 07:50	Obtain raw materials	Olaf
2	Ma-6131944444-2	2012-08-04 14:43	Manufacture product	SYS1
2	Co-6187615741-2	2012-08-04 14:51	Confirm order	Conny
2	Em-6388888889-2	2012-08-04 15:20	Emit invoice	SYS2
2	Ge-6439814815-2	2012-08-04 15:27	Get shipping address	SYS2
2	Sh-7277777778-2	2012-08-04 17:28	Ship product	Sara
2	Re-3611111111-2	2012-08-07 08:40	Receive payment	SYS2
2	Ar-3680555556-2	2012-08-07 08:50	Archive order	DMS
3	Ch-4208333333-3	2012-08-02 10:06	Check stock availability	SYS1
3	Ch-4243055556-3	2012-08-02 10:11	Check materials availability	SYS1

Process Discovery: the α-algorithm

• It is a basic deterministic mining algorithm. Assumptions:
(1) the events in the log are chronologically ordered.
(2) each event refers to a single case.
(3) each event relates to a specific activity of the process.
(4) each activity of the process is included in the log.
(5) the log is behaviorally complete in the sense that if an activity *a* can be directly followed by an activity *b*, then there is at least one case in the log where we observe *ab*.

Case ID	Event ID	Timestamp	Activity
1	Ch-468	2012-07-30 11:14	Check stock availability
1	Re-597	2012-07-30 14:20	Retrieve product from warehouse
1	Co-631	2012-07-30 15:10	Confirm order
1	Ge-640	2012-07-30 15:22	Get shipping address
1	Em-655	2012-07-30 15:44	Emit invoice
1	Re-418	2012-08-04 10:02	Receive payment
1	Sh-465	2012-08-05 11:11	Ship product
1	Ar-383	2012-08-06 09:12	Archive order
2	Ch-405	2012-08-01 09:44	Check stock availability
2	Ch-420	2012-08-01 10:06	Check materials availability
2	Re-466	2012-08-01 11:12	Request raw materials
2	Ob-326	2012-08-03 07:50	Obtain raw materials
2	Ma-613	2012-08-04 14:43	Manufacture product
2	Co-618	2012-08-04 14:51	Confirm order
2	Em-638	2012-08-04 15:20	Emit invoice
2	Ge-643	2012-08-04 15:27	Get shipping address
2	Sh-727	2012-08-04 17:28	Ship product
2	Re-361	2012-08-07 08:40	Receive payment
2	Ar-368	2012-08-07 08:50	Archive order

Letter	Actvities
a	Check stock availability
b	Retrieve product from warehouse
c	Check materials availability
d	Request raw materials
e	Obtain raw materials
f	Manufacture product
g	Confirm order
h	Get shipping address
i	Ship product
j	Emit invoice
k	Receive payment
l	Archive order

Workflow Log
a,b,g,h,j,k,i,l
a,c,d,e,f,g,j,h,i,k,l
...

• The starting point is to build the workflow log

Process Discovery: the α-algorithm

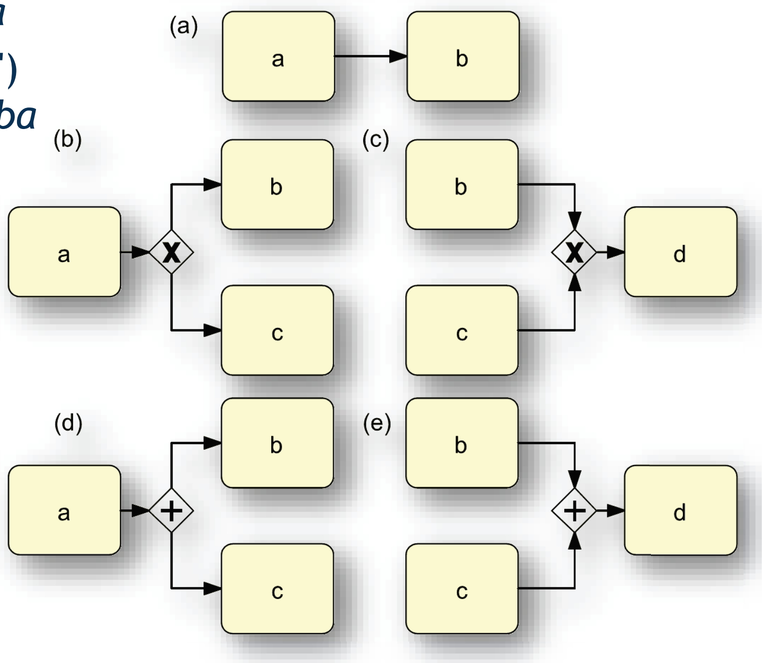
- A **workflow log** is a collection of all unique execution sequences observed in the log. The α-algorithm does not distinguish how often a specific execution sequence was observed in a workflow log.
- 1^{st} **phase:** extract three order relations from the workflow of L:

i) **causality:** $a \rightarrow b$ ("*a then b*") holds if we observe *ab* and not *ba*

ii) (potential) **parallelism:** $a \| b$ ("*a parallel b*") holds if we observe both *ab* and *ba*

iii) **non-succession:** $a \# b$ ("*a hash b*") holds if we observe neither *ab* nor *ba*

- 2^{nd} **phase:** map combinations of order relations to control flow patterns:



a) *sequence*: if $a \rightarrow b$

b) *xor-split*: if $a \rightarrow b \wedge a \rightarrow c \wedge b \# c$

c) *xor-join*: if $b \rightarrow d \wedge c \rightarrow d \wedge b \# c$

d) *parallel-split*: if $a \rightarrow b \wedge a \rightarrow c \wedge b \| c$

e) *parallel-join*: if $b \rightarrow d \wedge c \rightarrow d \wedge b \| c$

Process Discovery: the α-algorithm

- Order relations are easily derived by using the footprint matrix of the log:

- **Fig.** Footprint represented as a matrix of the workflow log L=[$\langle a,b,g,h,j,k,i,l \rangle$ $\langle a,c,d,e,f,g,j,h,i,k,l \rangle$];

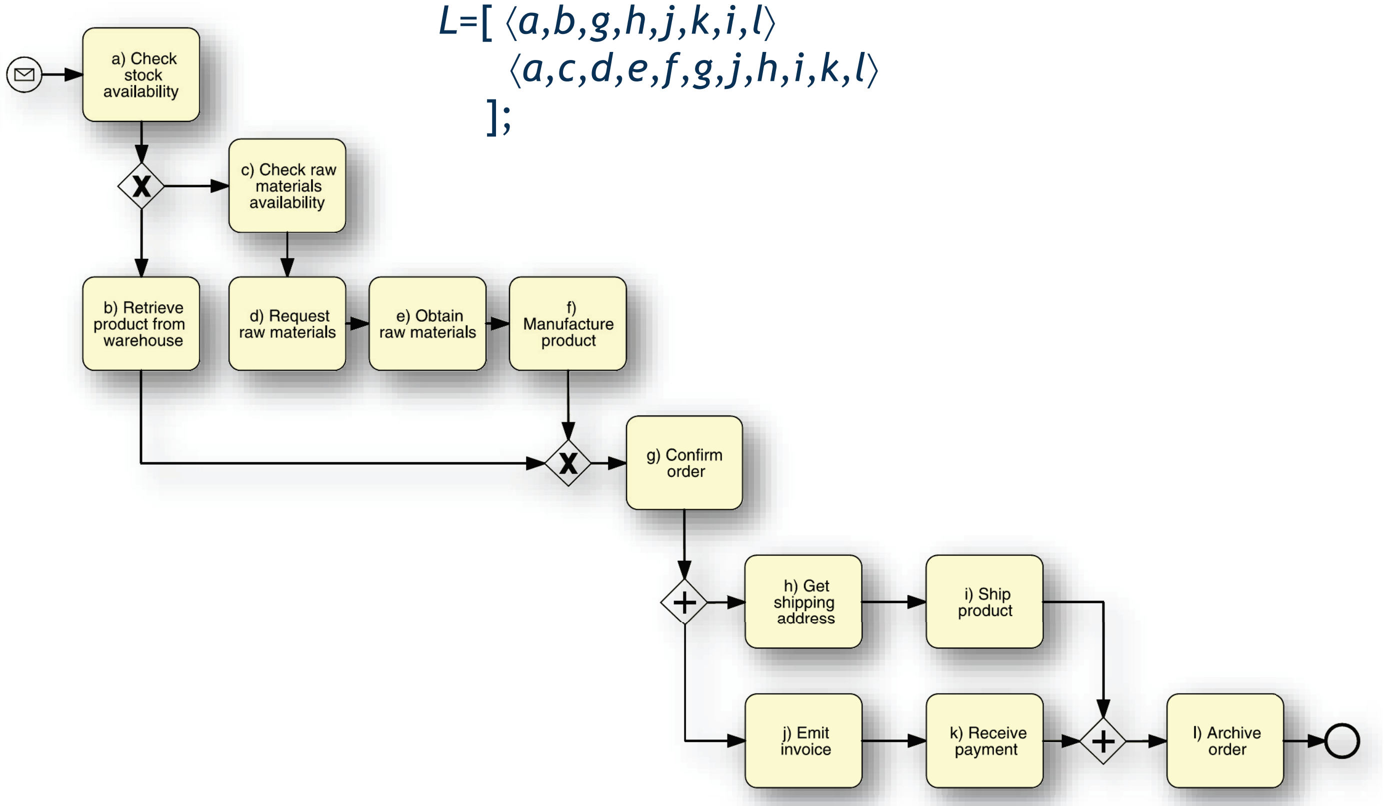
	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*
a	#	→										
b	←											
c												
d												
e												
f												
g												
h												
i												
j												
k									‖			
l												

- Order relations are easily derived by using the footprint matrix of the log:

- **Fig.** Footprint represented as a matrix of the workflow log
L=[
⟨a,b,g,h,j,k,i,l⟩
⟨a,c,d,e,f,g,j,h,i,k,l⟩
],

	a	b	c	d	e	f	g	h	i	j	k	l
a	#	→	→	#	#	#	#	#	#	#	#	#
b	←	#	#	#	#	#	→	#	#	#	#	#
c	←	#	#	→	#	#	#	#	#	#	#	#
d	#	#	←	#	→	#	#	#	#	#	#	#
e	#	#	#	←	#	→	#	#	#	#	#	#
f	#	#	#	#	←	#	→	#	#	#	#	#
g	#	←	#	#	#	←	#	→	#	→	#	#
h	#	#	#	#	#	#	←	#	→	‖	#	#
i	#	#	#	#	#	#	#	←	#	#	‖	→
j	#	#	#	#	#	#	←	‖	#	#	→	#
k	#	#	#	#	#	#	#	#	‖	←	#	→
l	#	#	#	#	#	#	#	#	←	#	←	#

- Process model constructed by the α-algorithm from workflow log
L=[⟨*a,b,g,h,j,k,i,l*⟩
⟨*a,c,d,e,f,g,j,h,i,k,l*⟩
];



- *Limitations* of the α-algorithm: it is not able to distinguish *short loops* from true parallelism:

b

I: abcd, acbd *II: abd, abcd, abcbd* *III: ad, abd, abcd, acbd*

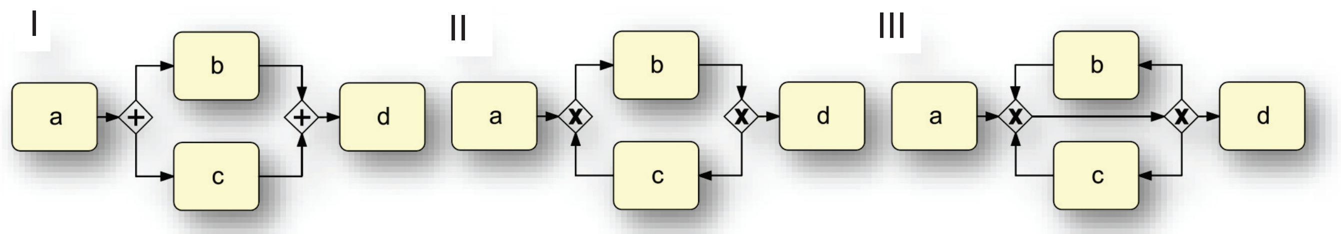


Fig. Examples of two short loops, which are problematic for the α-algorithm

- All three models above can produce the workflow logs that yield *b*||*c*, because both *bc* and *cb* can be observed.
- To distinguish (I) and (II), in the α+algorithm extension *b*||*c* is only included if there is no sequence *bcb* in the logs. Preprocessing can be also used to collapse repetitions (like *aa* and *bb*) to a single execution.
- Further problems for the α-algorithm are *incompleteness* and *noise*. E.g. to determine potential parallelism in 10 concurrent tasks, the number of required different cases to observe is 10! = 3,268,800. Moreover, event logs often include cases with missing head, tail, intermediate episode, logging errors with events being swapped or recorded twice.

✓ Conformance checking is concerned with the question whether or not the execution of a process (i.e., event logs) follows constraints.

✓ We focus on constraints expressed as a *normative process model*. If a particular constraint does not hold, we speak of a *violation*.

✓ The idea is to *replay* each trace of the log recording at each step whether an activity is allowed to be executed according to the model.

✓For example replay the case ⟨*a*,*b*,*g*,*l*,*j*,*k*,*l*⟩ on the model shown in figure.

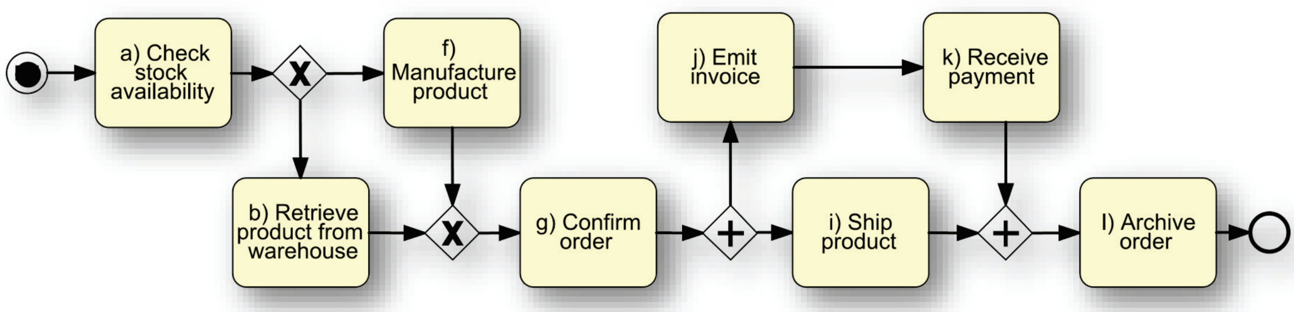


Fig. BPMN model with token on start event for replaying the case $\langle a, b, g, i, j, k, l \rangle$

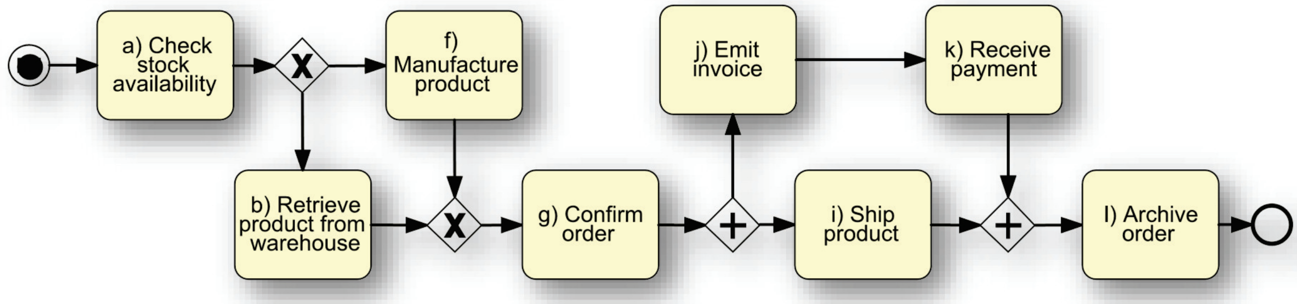
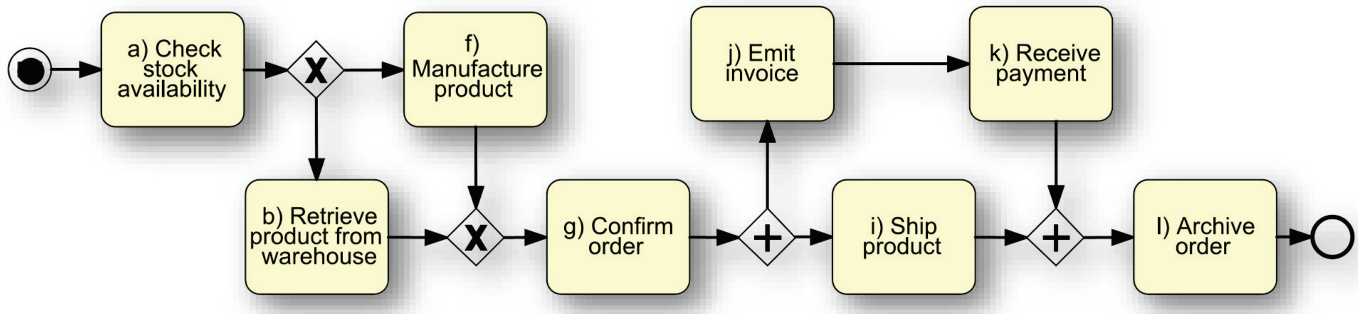


Fig. BPMN model with token on start event for replaying the case $\langle a, b, g, i, j, k, l \rangle$

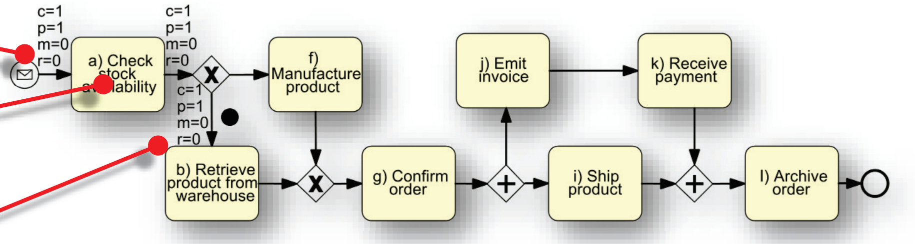
✓ In the initial state the process has a token on the start event, which leads to activity *a*. Then, the XOR-split is activated, which allows to either continue with *b* or with *f*. For the considered case, we can continue with *b*. Then, we can continue with *g*, after which the AND-split enables both *i* and *j*. These activities are concurrent. In order to replay the case, we first execute *i* and *j* afterwards. Once *i* and later *k* is completed, the AND-join is allowed to proceed. One token on each its input arcs is required for that. Since both of these tokens are consumed, a single token can be created to enable *l*, which can be finally executed. Thus, the case can be totally replayed on the model.



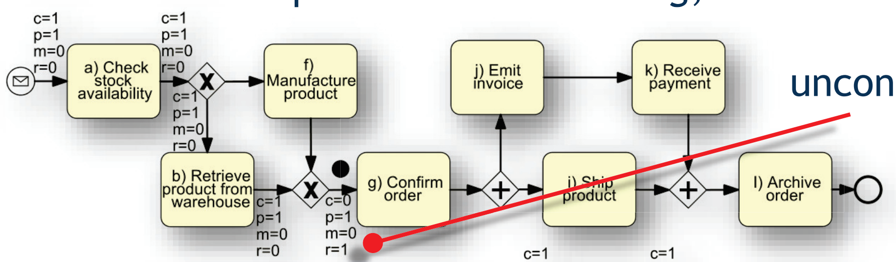
✓ Based on the concept of token replay, we can also assess the conformance of a trace to a process model.
✓ The idea is to compare at each step the number of tokens that are required for replaying an activity with the actually available tokens.
✓ At each step, we might observe situations of conformance and non-conformance. In case of conformance, we count the following four facts:

- *p: # of output tokens correctly produced by a model element*
- *c: # of input tokens correctly consumed by a model element*
- *m: # of missing (unproduced) output tokens, e.g. because something did not occur*
- *r: # of input tokens remaining unconsumed, e.g. because something did not occur although the model expected it to happen*

✓ Example. Consider the case ⟨*a*,*b*,*i*,*j*,*k*,*l*⟩. Immediately after the start event, one token is correctly produced and consumed. The same situation after replaying *a*.



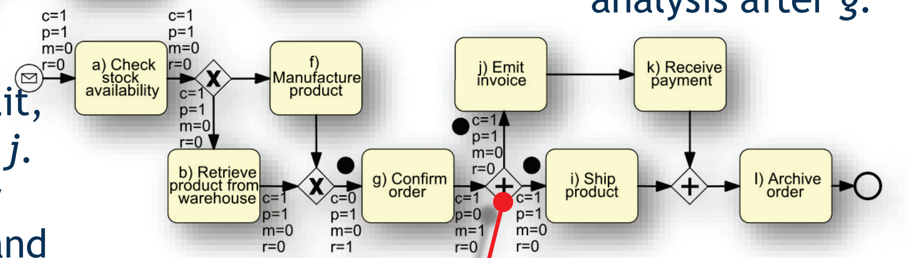
✓ Let us continue with *b* to check the considered case. After replaying *b*, there is a token produced to execute *g*, but it is not consumed by the case.

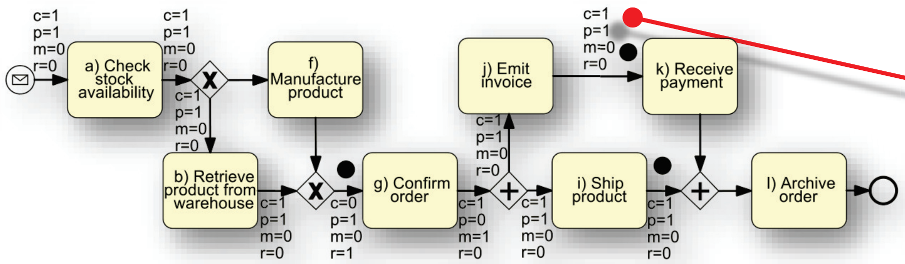


Thus, one token remains unconsumed before *g*, since *g* is not in the log for replay

Let us continue with the analysis after *g*.

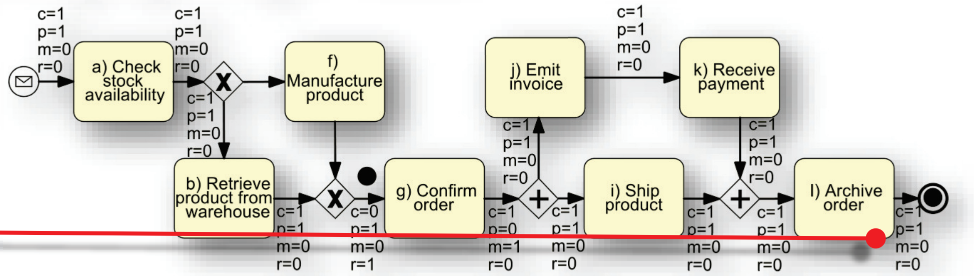
After *g* there is a missing token for firing the AND-split, which would activate *i* and *j*. The AND-split can correctly consume the input token, and correctly produce the output tokens: both are correctly consumed in the log





✓ The token delivered by *j* can also be replayed by *k in the case* ⟨*a*,*b*,*i*,*j*,*k*,*l*⟩.

✓ The AND-join can synchronize *k* and *i*, and finally *l* can be also replayed in the log.



✓ We calculate the *fitness* of the case by using the fractions *missing-to-consumed* and *remaining-to-produced:*

$$fitness = \frac{1}{2}\left(1-\frac{m}{c}\right)+\frac{1}{2}\left(1-\frac{r}{p}\right)$$

- *p*: # of output tokens correctly *produced*
- *c*: # of input tokens correctly *consumed*
- *m*: # of *missing* (unproduced) output tokens
- *r*: # of input tokens *remaining* unconsumed

$p+m = c+r$

✓ With *m*=1 and *r*=1, *c*=12 and *p*=12, *fitness*=(1-1/12)/2 + (1-1/12)/2 =0.9166

✓ With a set of cases: after replaying a case, continue counting *c,p,m,r* by replaying the next case in the process model. Once all cases have been replayed, get the resulting fitness with the same formula. Example in figure

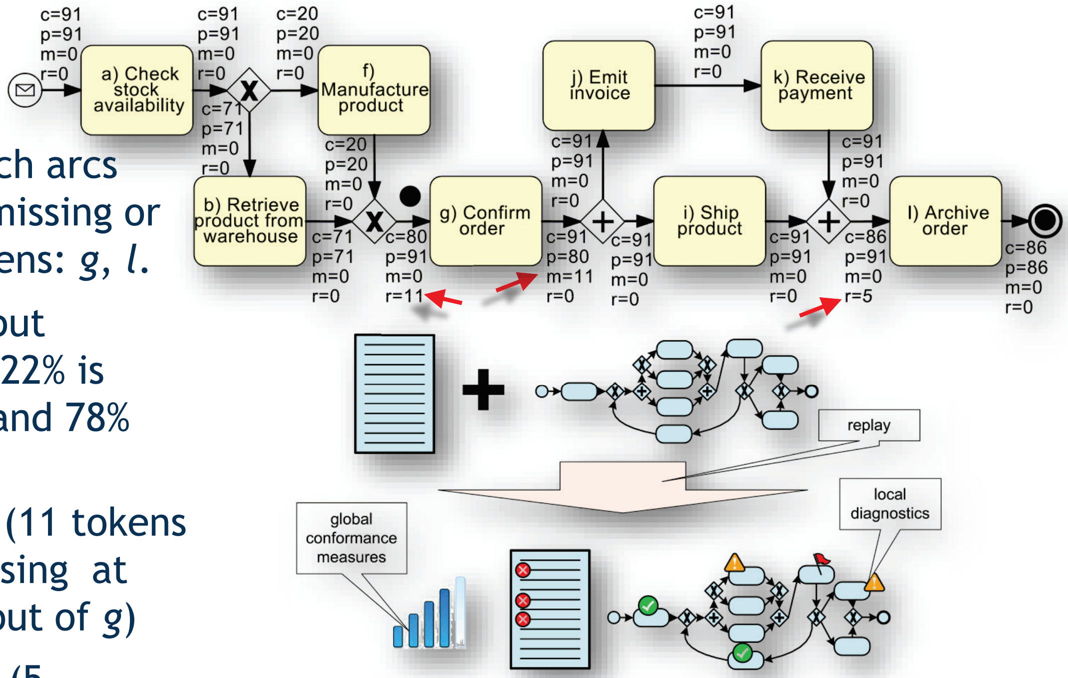
$p+m = c+r$

✓ Inspect which arcs encountered missing or remaining tokens: *g*, *l*.

✓ Given 91 input tokens, about 22% is handled via *f* and 78% via *b*.

✓ 12% omits *g* (11 tokens remaining/missing at the input/output of *g*)

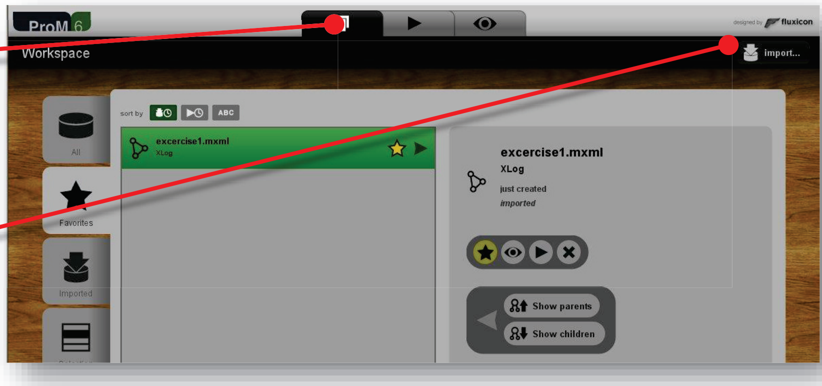
✓ 5.5% omits *l* (5 tokens remaining at the input of *l*)



Conclusion: conformance checking provides global conformance measures (like fitness) and local diagnostics

✓ *Workspace* Tab for input/output

✓ Click on import to load *exercise1.xes*



✓ *ActionTab* for executing algorithms

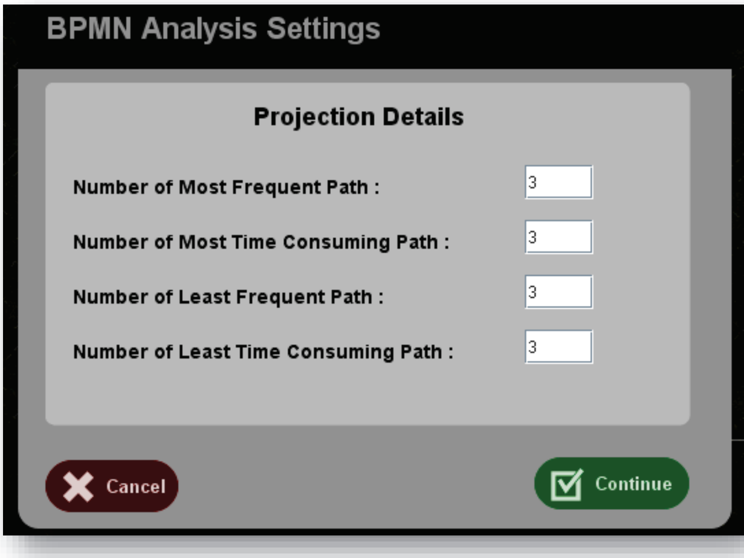
✓ Enter "BPMN" to select BPMN-based plugins

✓ Select "BPMN Analysis (using Causal Net Miner)" and press "Start"

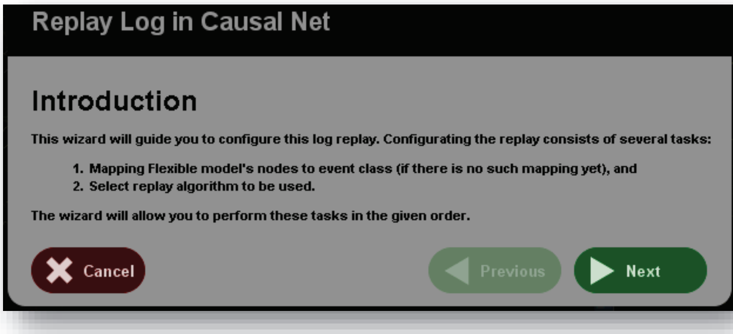
✓ BPMN Analysis using Causal (C-) Net Miner

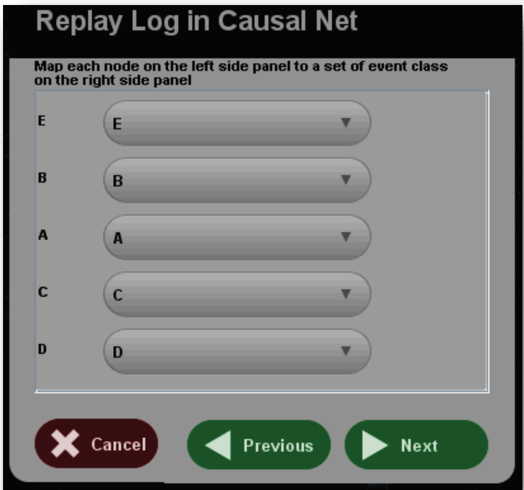
It is an advanced algorithm. How it works (in brief):

- A **C-net** is a representation of a process model as a graph, where nodes represent activities and arcs represent causal dependencies;
- **Cost-based fitness** is measured based on finding skipped/inserted activities giving minimal costs (highest possible fitness value), using **A*** algorithm (heuristic variant of Dijkstra's algorithm) to find shortest paths between two nodes in a directed graph with arc costs;
- The cost function is provided with options to specify the relative costs of skipped / inserted activities
- A **projection** method is used to split the log into pieces: it selects groups of events tightly related in the log projecting the log on these events. Leave default values and click on “Continue”.



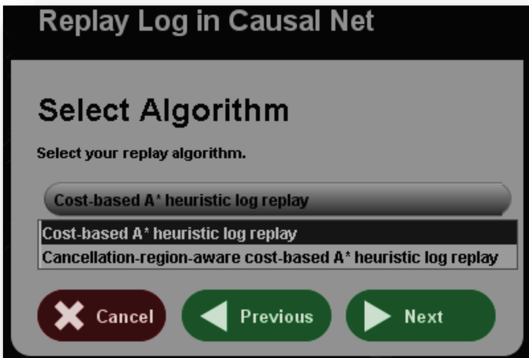
- A wizard procedure to configure the replay process.



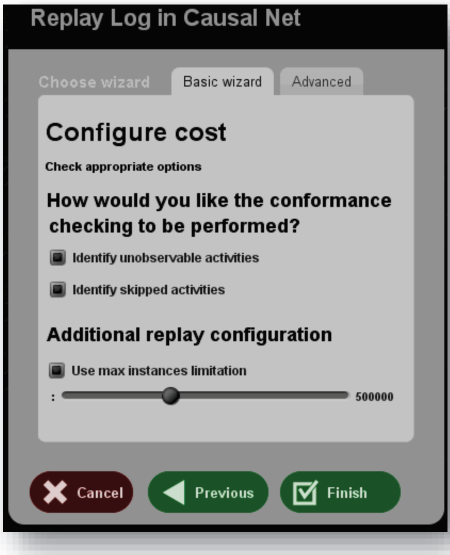


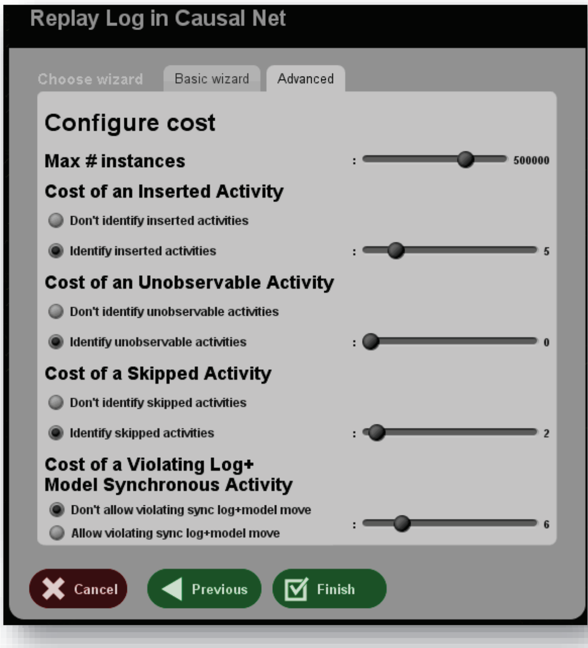
- Check the label match between node of the net and event classes. There can be additional nodes in the net, causing mismatch.

- Some algorithms are grounded on the theory of regions, which maps a model in the state based domain (automata) into the event-based domain (e.g Petri net). Leave default values and click “Next”

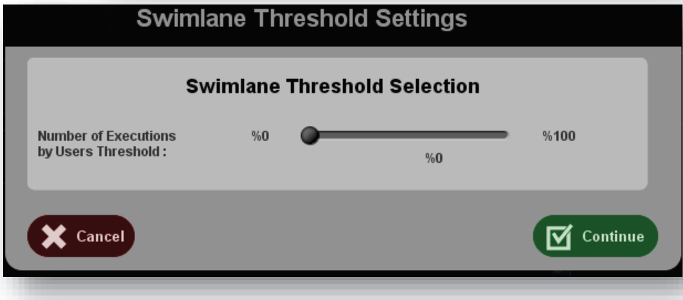


- The last part of the wizard relates to the cost configuration: leave the default values and press “Finish”.

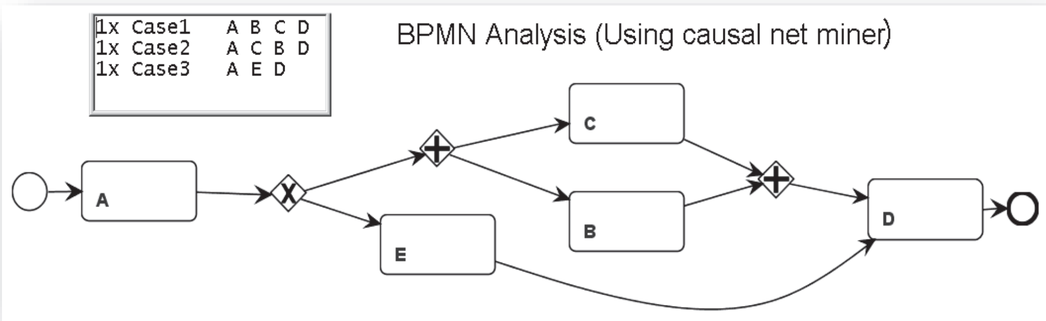




- Finally, a threshold can be set to discover also swimlanes
- Leave the default values and press “Finish”/”Continue”.



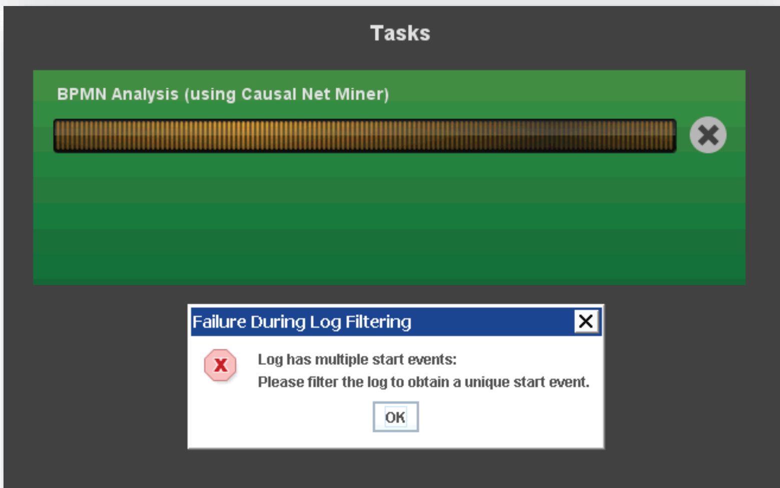
- Exercise 1



- Exercise 2

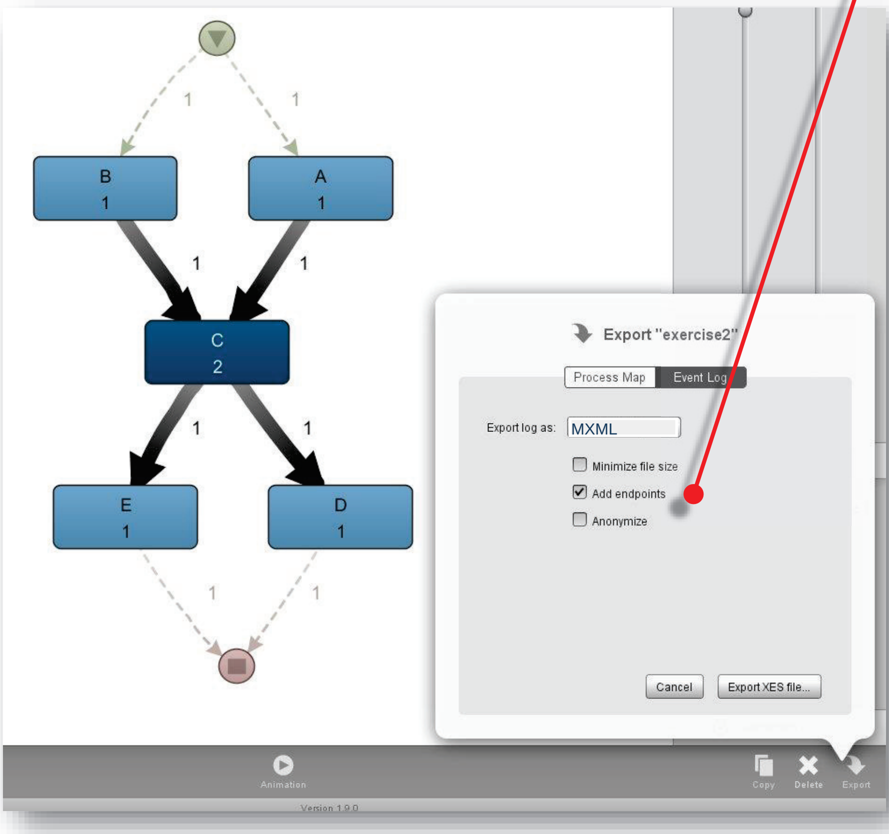
```
1x Case1   A C D
1x Case2   B C E
```

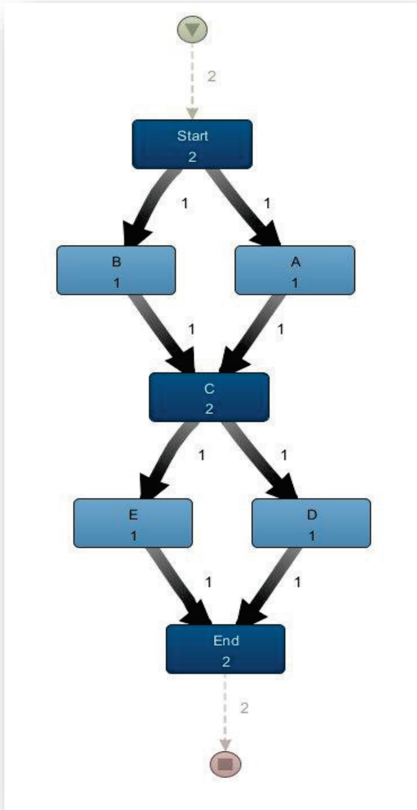
- The log cannot be processed, because it has multiple start events



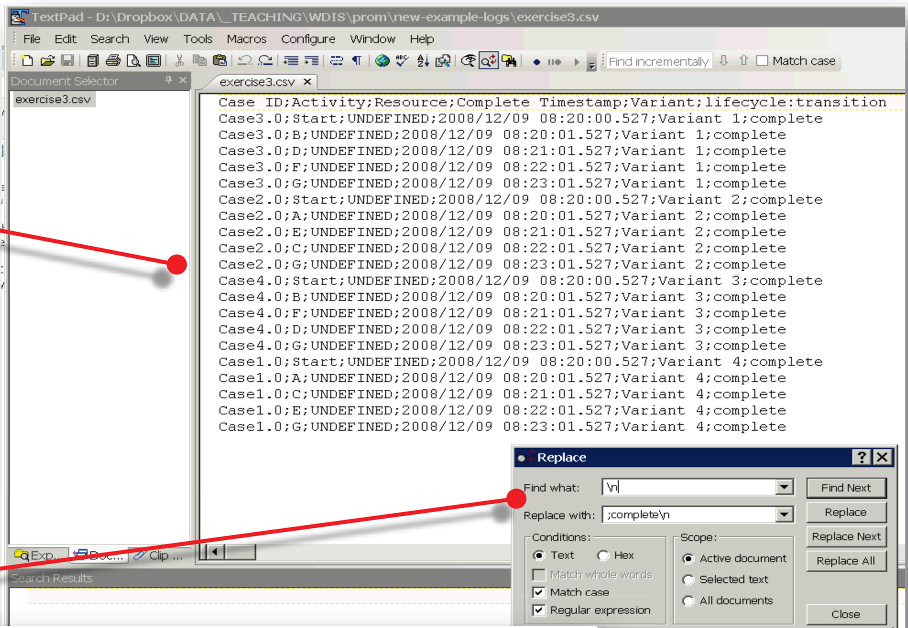
- To add the Start/End events, you can use either DISCO or TEXTPAD

- With Disco, you can export the log by adding endpoints



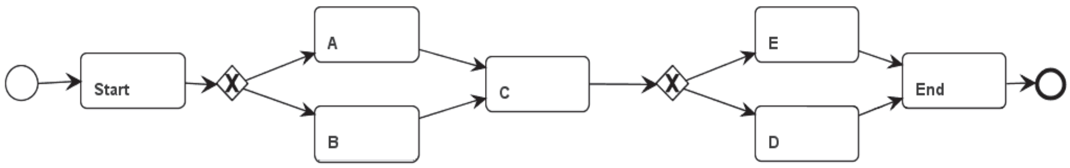


- To manually edit the log, convert it from MXML to CSV. Then edit it with Textpad or Excel and then use again Disco to convert it to MXML
  - With Textpad regular expressions can be used



A	B	C	D	E	F	G	H
Case ID	Activity	Resource	Complete Timestamp	Variant	(case) creator	(case) variant	lifecycle:transition
Case2.0	Start	Start	2008/12/09 08:20:01.527	Variant 1	Fluxicon Disco	Variant 1	complete
Case2.0	B	UNDEFINED	2008/12/09 08:20:01.527	Variant 1	Fluxicon Disco	Variant 1	complete
Case2.0	C	UNDEFINED	2008/12/09 08:21:01.527	Variant 1	Fluxicon Disco	Variant 1	complete
Case2.0	E	UNDEFINED	2008/12/09 08:22:01.527	Variant 1	Fluxicon Disco	Variant 1	complete
Case2.0	End	End	2008/12/09 08:22:01.527	Variant 1	Fluxicon Disco	Variant 1	complete
Case1.0	Start	Start	2008/12/09 08:20:01.527	Variant 2	Fluxicon Disco	Variant 2	complete
Case1.0	A	UNDEFINED	2008/12/09 08:20:01.527	Variant 2	Fluxicon Disco	Variant 2	complete
Case1.0	C	UNDEFINED	2008/12/09 08:21:01.527	Variant 2	Fluxicon Disco	Variant 2	complete
Case1.0	D	UNDEFINED	2008/12/09 08:22:01.527	Variant 2	Fluxicon Disco	Variant 2	complete
Case1.0	End	End	2008/12/09 08:22:01.527	Variant 2	Fluxicon Disco	Variant 2	complete

- Exercise 2



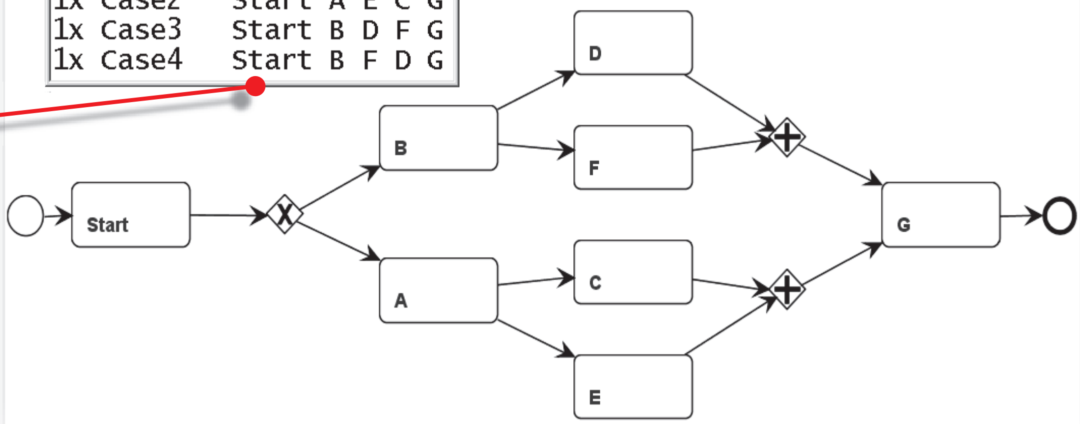
1x	Case1	Start A C D End
1x	Case2	Start B C E End

BPMN Analysis (Using causal net miner)

- Exercise 3 (the Start event must be added)

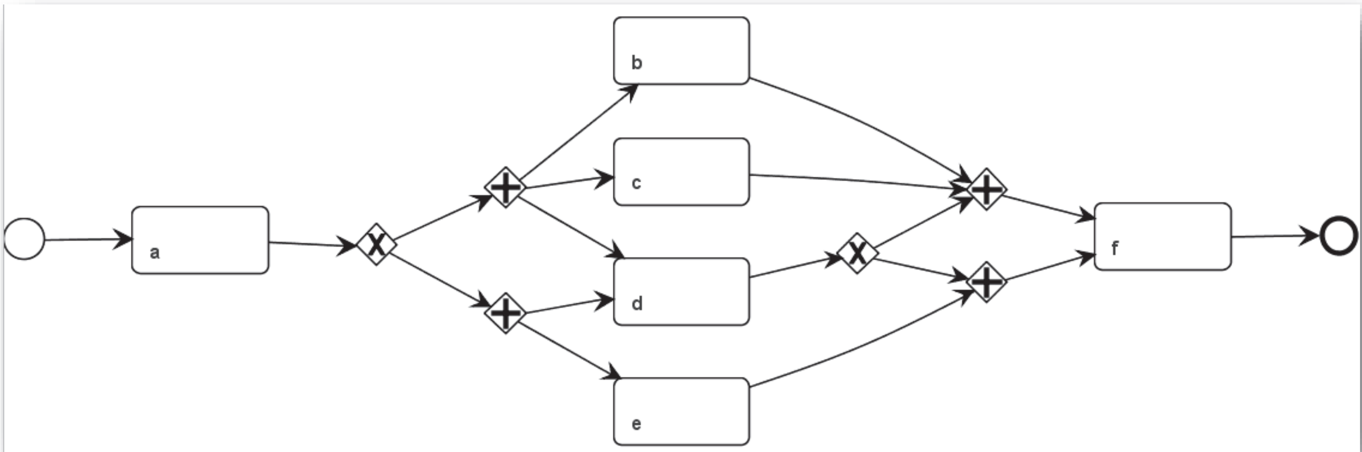
1x	Case1	Start A C E G
1x	Case2	Start A E C G
1x	Case3	Start B D F G
1x	Case4	Start B F D G

BPMN Analysis (Using causal net miner)



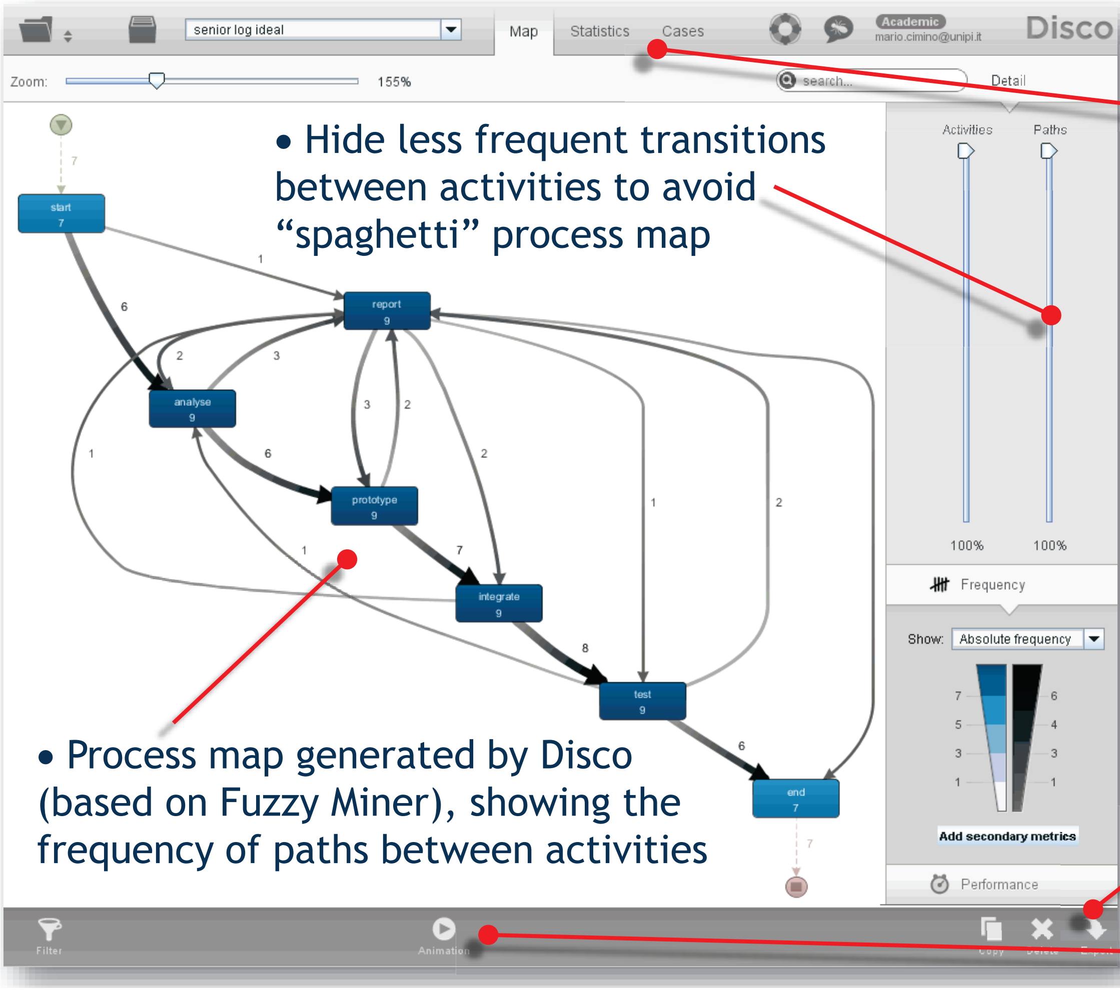
- Exercise 4

1x	Case1	a b c d f
1x	Case2	a c b d f
1x	Case3	a b d c f
1x	Case4	a c d b f
1x	Case5	a d e f
1x	Case6	a e d f

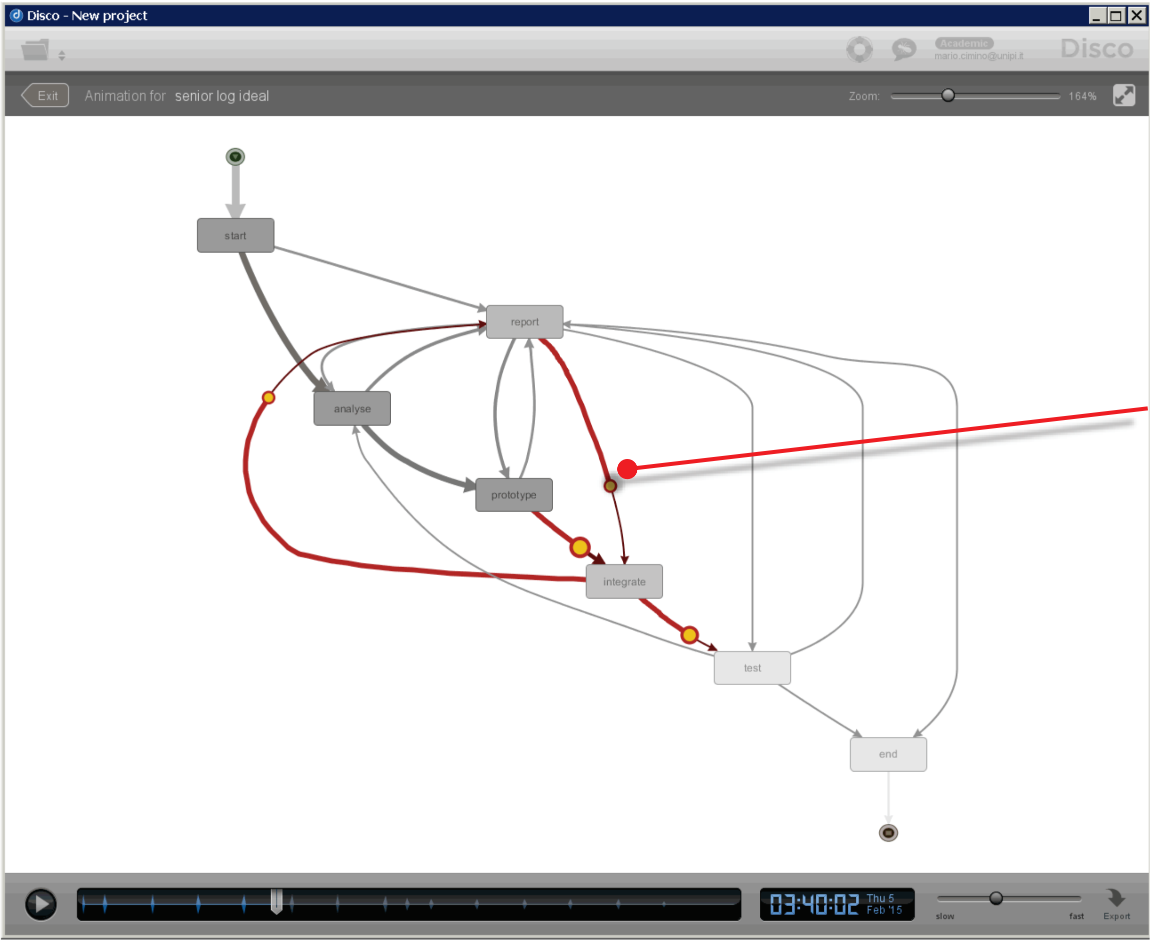


- SENIOR log (Software ENgineering for Input/Output pRoblems)
- It contains different cases of problem solving. Each case involves a worker improving the input/output (i/o) of a Java-based software application
- Synthetic log: cases without violations; the base pattern, iterated one or many times is $(a \to p \to i \to t) \parallel r$

Case ID	Timestamp	Activity
0	05/02/2015 00.00	**start**
0	05/02/2015 01.00	report
0	05/02/2015 02.00	analyse
0	05/02/2015 03.00	prototype
0	05/02/2015 04.00	integrate
0	05/02/2015 05.00	test
0	05/02/2015 06.00	end
1	05/02/2015 00.00	**start**
1	05/02/2015 01.00	analyse
1	05/02/2015 02.00	report
1	05/02/2015 03.00	prototype
1	05/02/2015 04.00	integrate
1	05/02/2015 05.00	test
1	05/02/2015 06.00	end
2	05/02/2015 00.00	**start**
2	05/02/2015 01.00	analyse
2	05/02/2015 02.00	prototype
2	05/02/2015 03.00	report
2	05/02/2015 04.00	integrate
2	05/02/2015 05.00	test
2	05/02/2015 06.00	end
3	05/02/2015 00.00	**start**
3	05/02/2015 01.00	analyse
3	05/02/2015 02.00	prototype
3	05/02/2015 03.00	integrate
3	05/02/2015 04.00	report
3	05/02/2015 05.00	test
3	05/02/2015 06.00	end
4	05/02/2015 00.00	**start**
4	05/02/2015 01.00	analyse
4	05/02/2015 02.00	prototype
4	05/02/2015 03.00	integrate
4	05/02/2015 04.00	test
4	05/02/2015 05.00	report
4	05/02/2015 06.00	end
5	05/02/2015 00.00	**start**
5	05/02/2015 01.00	analyse
5	05/02/2015 02.00	prototype
5	05/02/2015 03.00	integrate
5	05/02/2015 04.00	test
5	05/02/2015 05.00	report
5	05/02/2015 07.00	analyse
5	05/02/2015 08.00	report
5	05/02/2015 09.00	prototype
5	05/02/2015 10.00	integrate
5	05/02/2015 11.00	test
5	05/02/2015 12.00	end
6	05/02/2015 00.00	**start**
6	05/02/2015 01.00	analyse
6	05/02/2015 02.00	report
6	05/02/2015 03.00	prototype
6	05/02/2015 04.00	integrate
6	05/02/2015 05.00	test
6	05/02/2015 06.00	analyse
6	05/02/2015 07.00	prototype
6	05/02/2015 08.00	report
6	05/02/2015 09.00	integrate
6	05/02/2015 10.00	test
6	05/02/2015 11.00	end

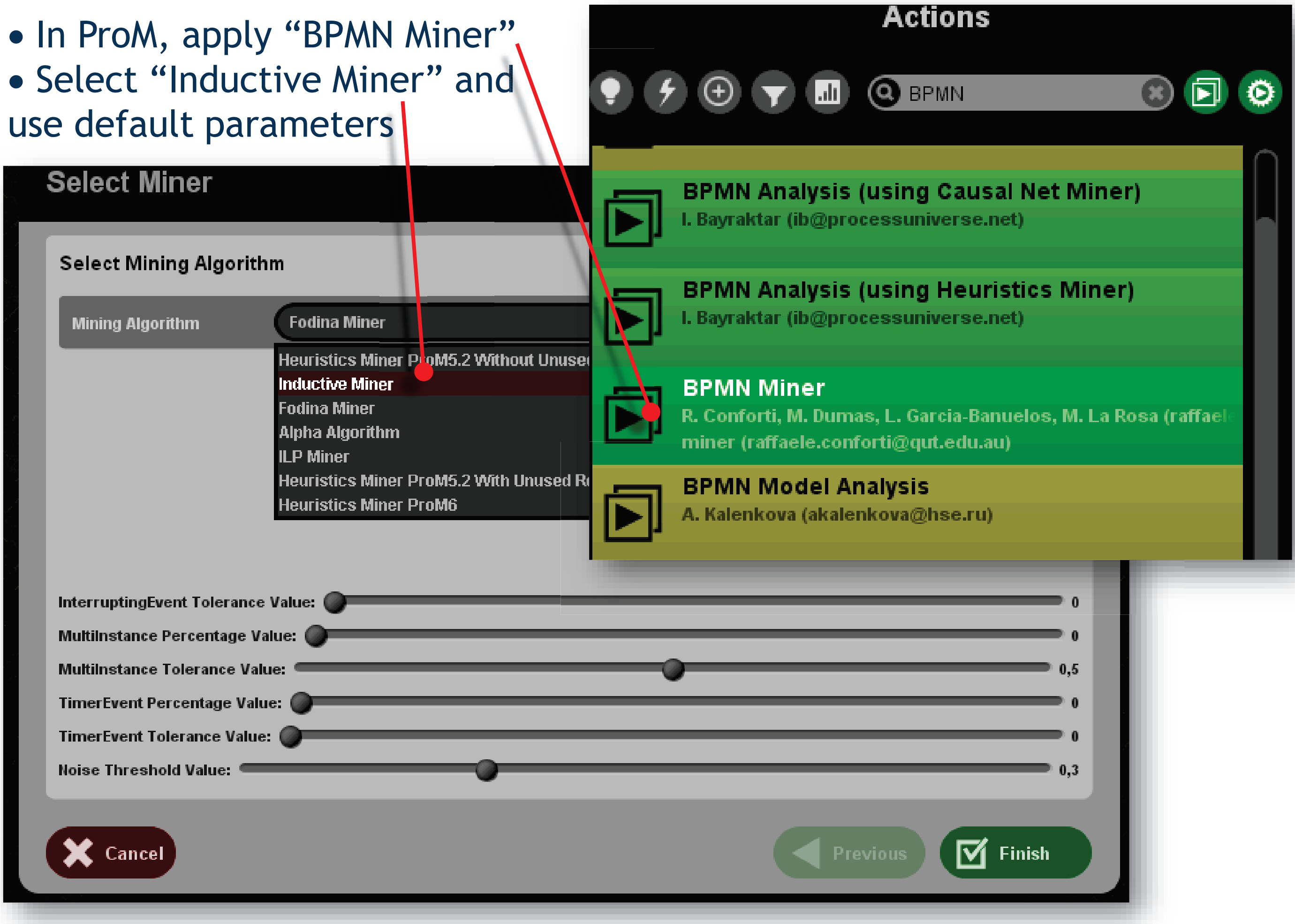


- Hide less frequent transitions between activities to avoid "spaghetti" process map
- Process map generated by Disco (based on Fuzzy Miner), showing the frequency of paths between activities
- See statistics and Cases
- Export data in a number of formats
- Animation

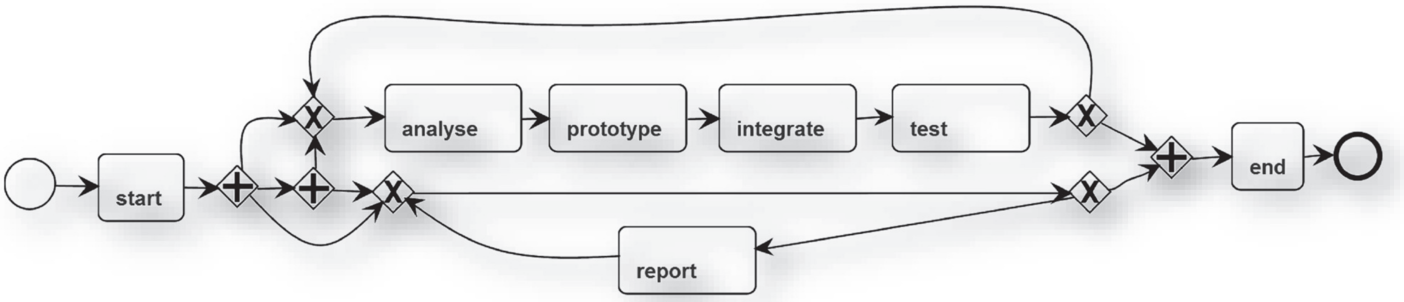


- Tokens flows animation

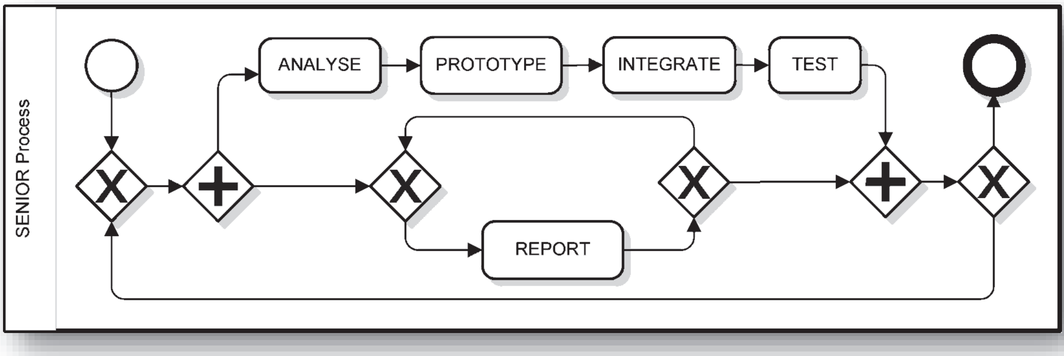
- In ProM, apply "BPMN Miner"
- Select "Inductive Miner" and use default parameters



- Model generated by the inductive miner algorithm



- Since there is no violation in the event log, the generated model is very similar to the normative process:



- In brief: the **Inductive Miner** aims to discover block-structured process models fitting the behavior represented in event log. IM partitions the activities, select the most important process constructs, splits the log and recurses until a base case is encountered.

- A *process tree* is the hierarchical representation of a block-structured workflow net. The leaves of the tree are activities, representing transitions. The nodes of the tree, *operators*, describe how their children are combined: exclusive choice ($\times$), sequential composition ($\rightarrow$), parallel composition ($\wedge$), and loop ($\circlearrowleft$).

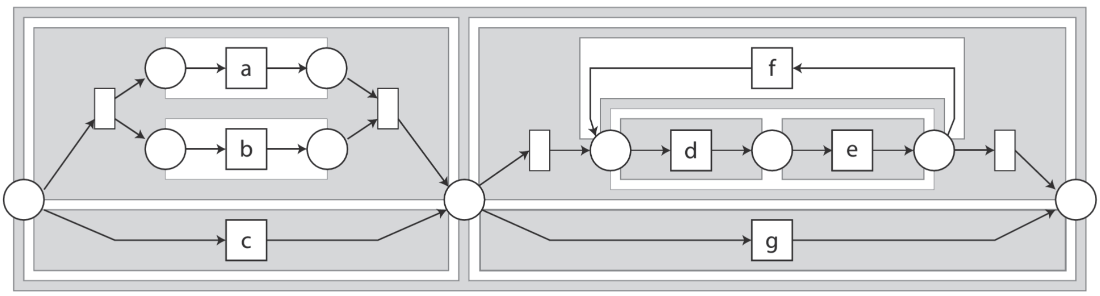
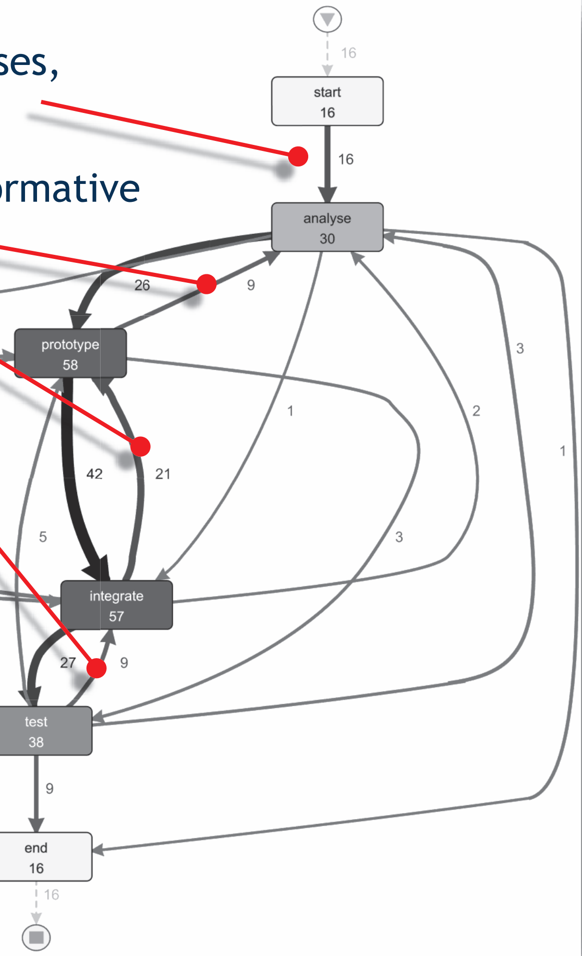
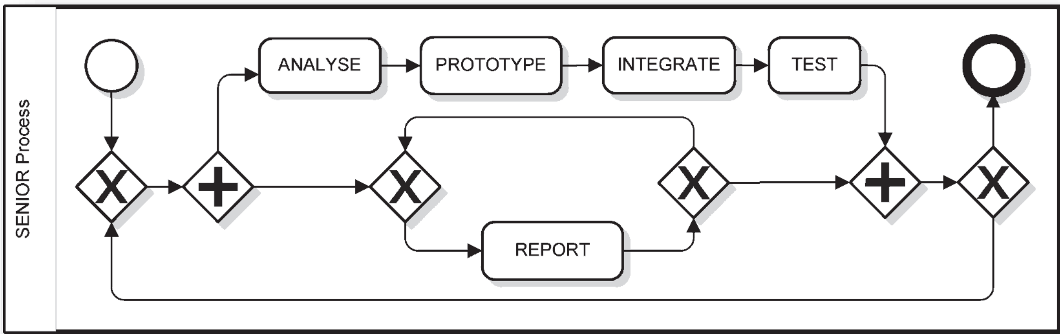


Figure: A block-structured workflow net M_E; filled regions denote the block-structure; process tree $\rightarrow(\times(\wedge(a, b), c), \times(\circlearrowleft(\rightarrow(d, e), f), g))$ corresponds to this net.

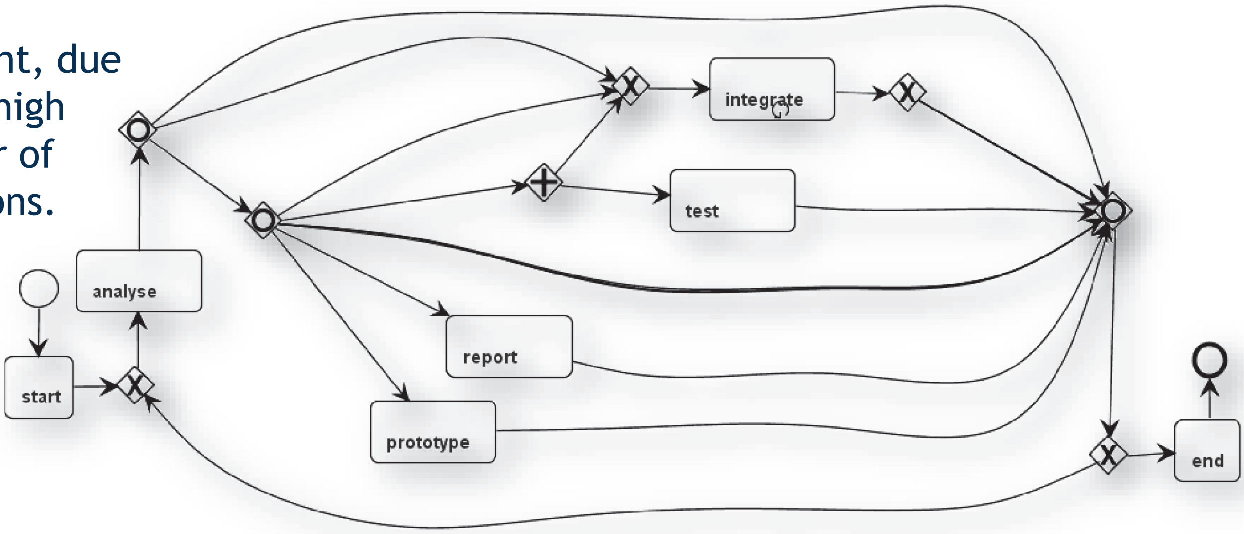
- Real log: 16 different cases, 241 total events.
- Many violations of the normative process.



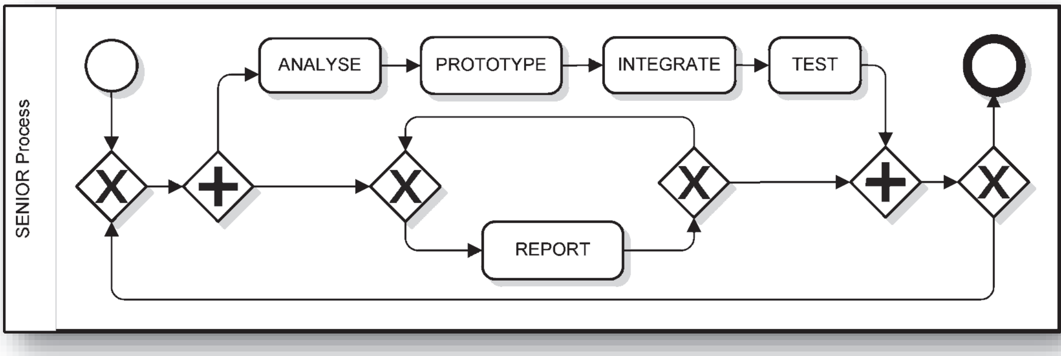
- Normative process:



- The "spaghetti" model generated by the inductive miner algorithm is very different, due to the high number of violations.



- Conformance checking: to check the real log against the normative process



- Steps on ProM:

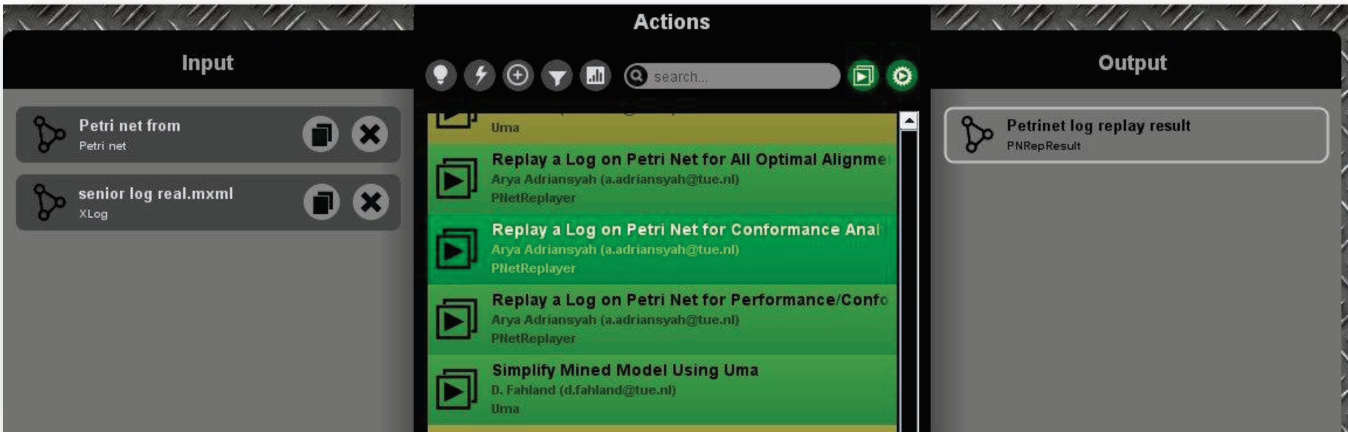
1: Import both the normative model (senior bpmn.xml) and the real log (senior log real.mxml)

2: In the action tab, click on “Select BPMN Diagram”; the normative process model appears.

3: Select the BPMN Diagram in the workspace tab, and click the action button; then select “Convert BPMN Diagram to Petri net (control-flow)”; a Petri net appears: you do not need to inspect it.

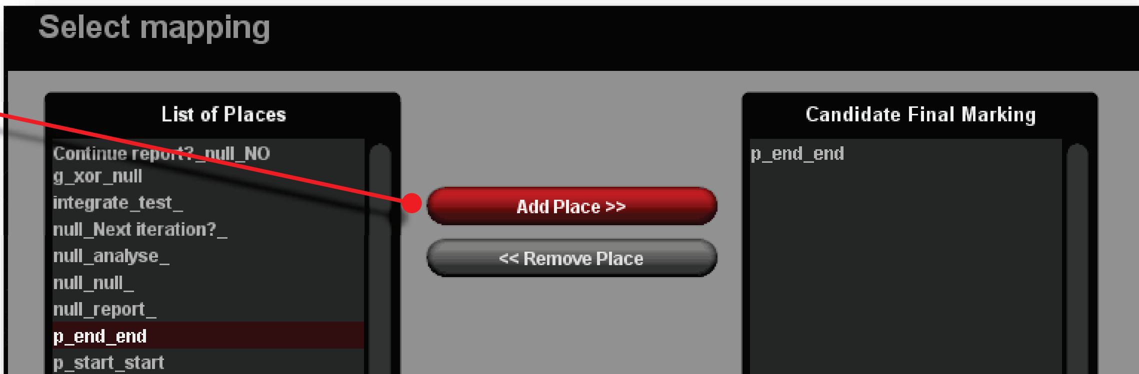
4: In the workspace tab, select the Petri net, and click the action button; then click to add input object and select “senior log real.mxml”

5: In the action tab, select “Replay Log on Petri Net for Conformance Analysis”

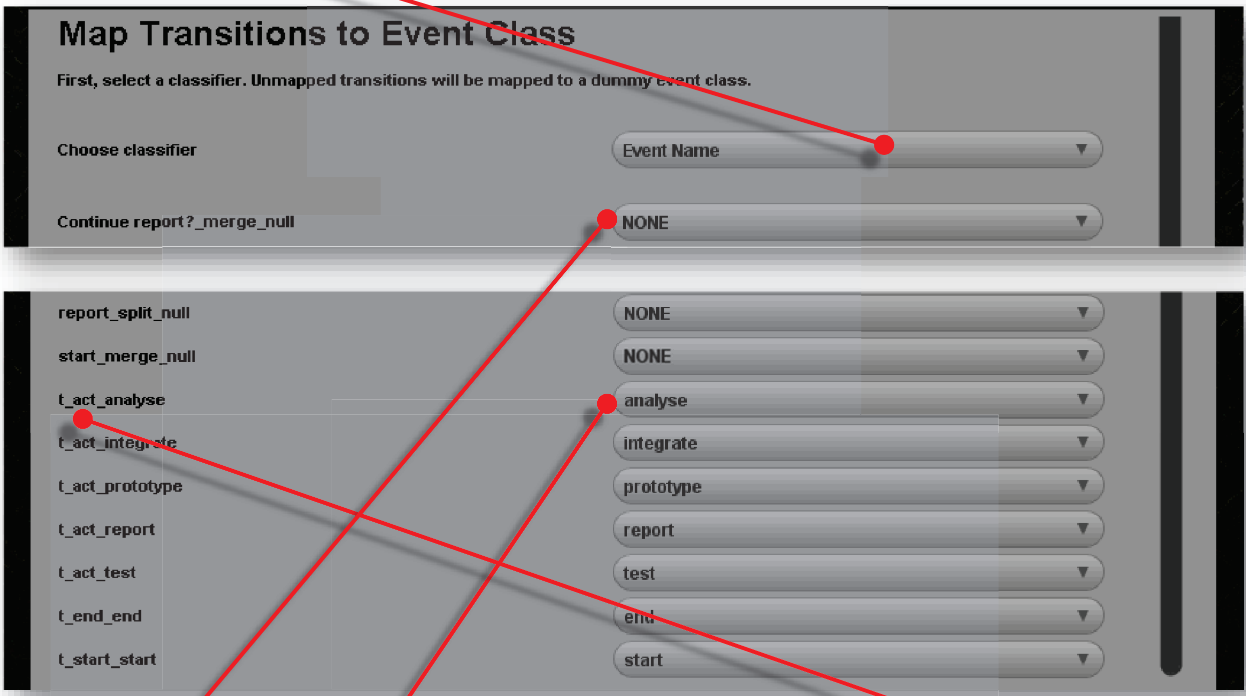


6: Answer YES to the question “No final marking is foud on this model. Do you want to create one?”; select “p_end_end” as a candidate final marking;

Click on “Add place”.



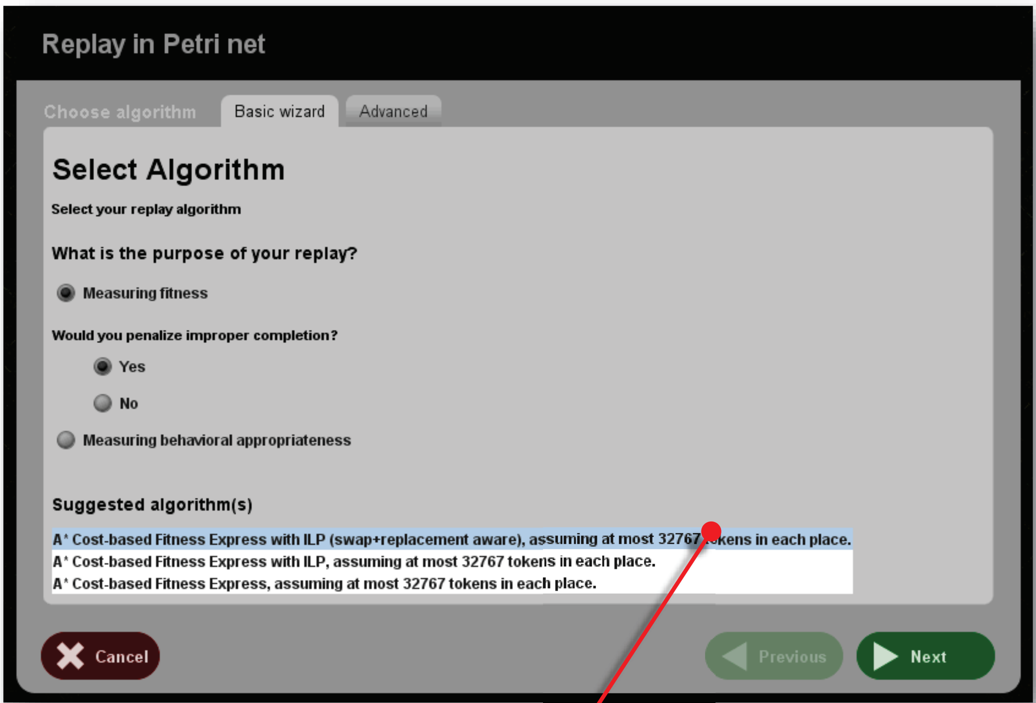
7: Select “Event Name” as a classifier



8: Select NONE for all transitions, except for those with “t_act”,”t_end”, “t_start” as a name prefix, whose mapping must be accurately checked

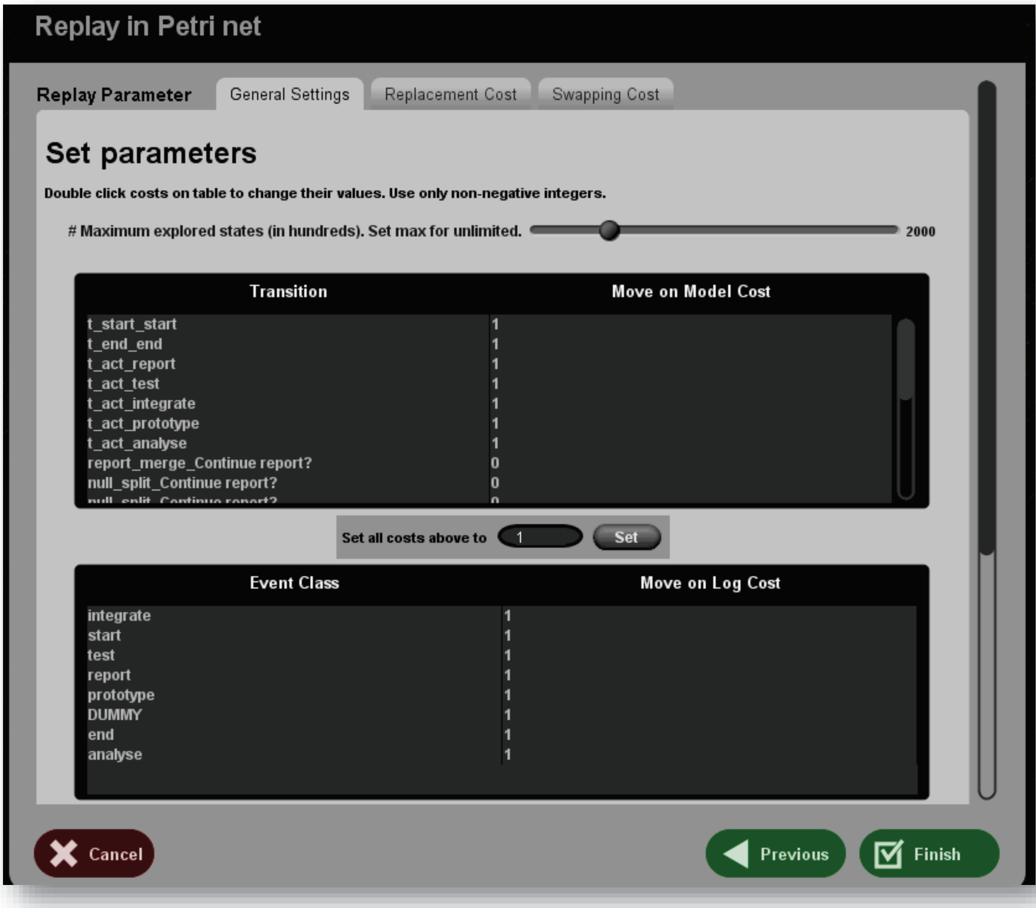
9: Select “YES, set them to invisible” in the dialog windows on the visibility of unmapped transitions

10: Wait the processing for abut a half a mninute

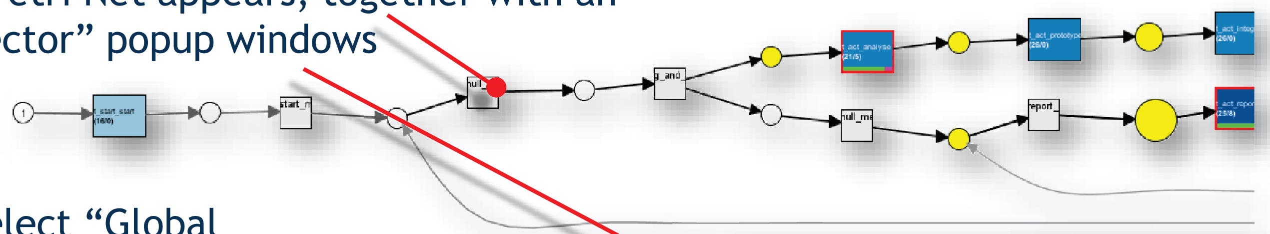


11: Leave the default algorithm for measuring fitness

12: Leave the default parameters, and click on “Finish”. Wait some seconds



13: A Petri Net appears, together with an “Inspector” popup windows



14: Select “Global Statistics” to see a table with the most important average fitness properties.

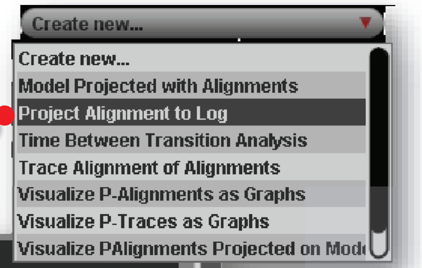
- In particular, the average *Trace Fitness* is shown: 0.737457...

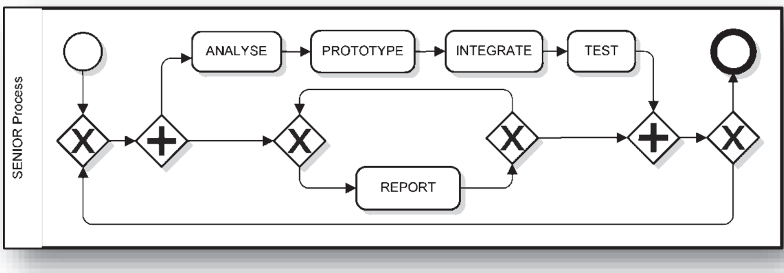
- The Trace-Fitness value represents the fitness value of the Petri Net with the log, and indicates how well the event log can be replayed in the discovered Petri Net.

- A fitness value of 1 means that the log can be successfully replayed, whereas a value of 0 means that this is completely not the case.

15: Select *Create new* ⇒ *Project Alignment to log*

16: The individual trace fitness values can be inspected





← The normative process

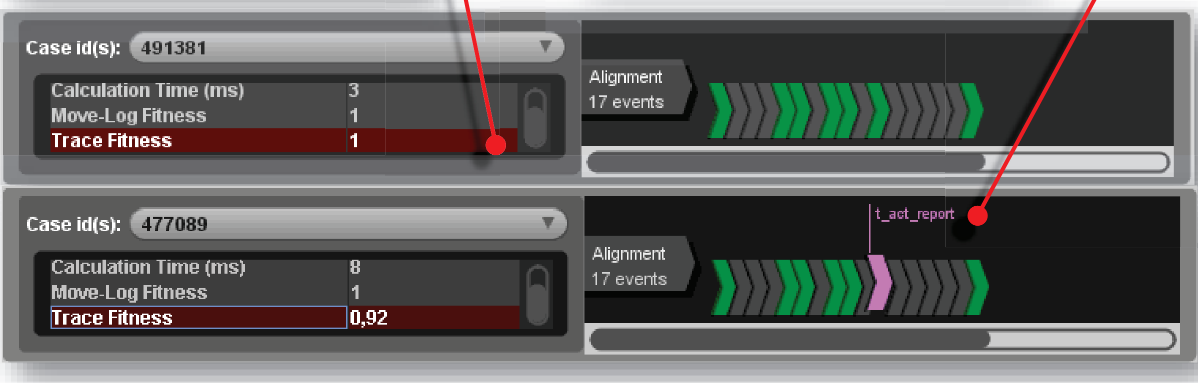
LEGEND
- Synchronous move (move log+model)
- Unobservable move (move model only)
- Skipped event class (move model only)
- Inserted event class (move log only)
- Replaced violation (move log+model)
- Swapped violation (move log+model)

- Case 491381: Fitness=1

491381, 11/12/2014 22.11, start
491381, 11/12/2014 22.12, analyse
491381, 15/12/2014 17.05, prototype
491381, 06/01/2015 12.07, integrate
491381, 22/01/2015 11.22, test
491381, 24/01/2015 09.58, report
491381, 24/01/2015 16.47, end

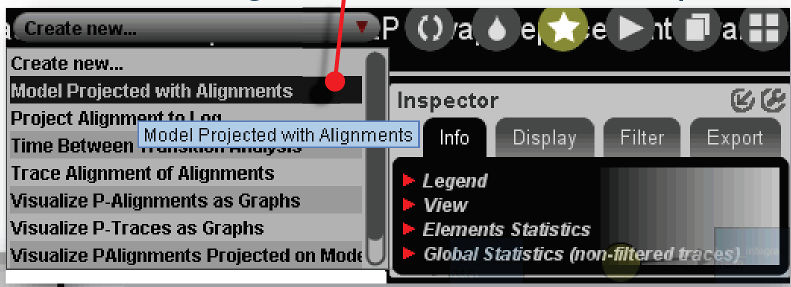
- Case 477089: Fitness = 0.92 (report event skipped)

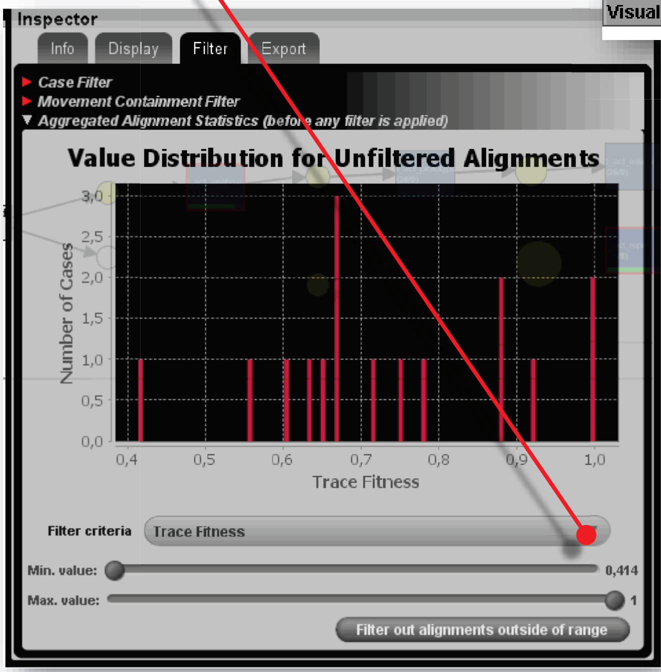
477089, 16/12/2014 21.59, start
477089, 16/12/2014 22.00, analyse
477089, 26/12/2014 14.49, prototype
477089, 08/01/2015 21.22, integrate
477089, 11/01/2015 15.34, test
477089, 11/01/2015 18.27, end



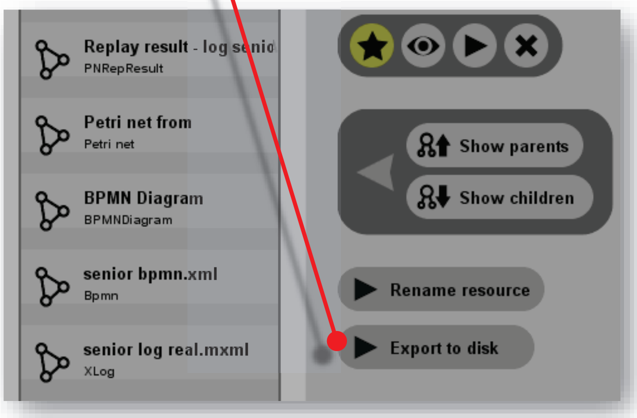
Select *Create new…> Model Projected with Alignments* to show the Inspector

Select *Filter tab >*
Aggregated Alignment Statistics
> Trace Fitness

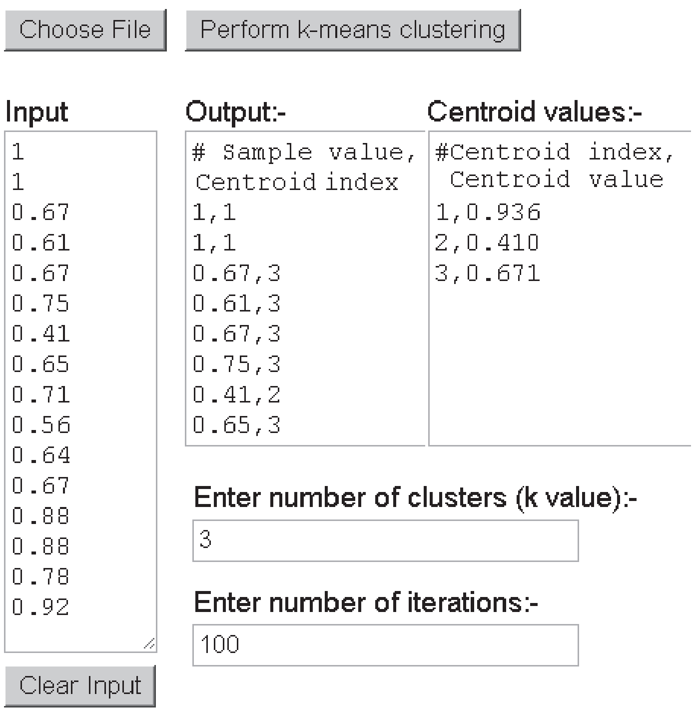




1. Go to the workspace tab
2. Select the Replay result in the list
3. Click on "export to disk"
4. Select "Export result report as CSV(.csv)"

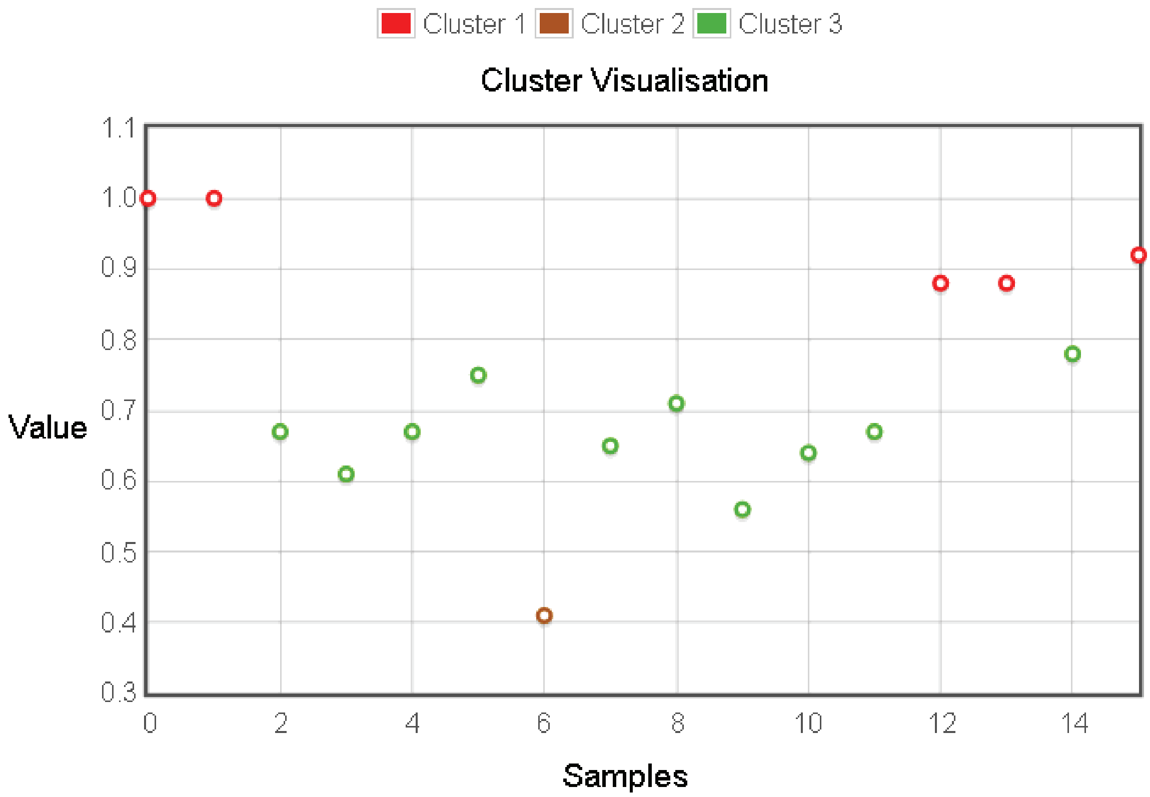


Separate cases into clusters according to fitness
https://tinyurl.com/pdis-k-means
http://scistatcalc.blogspot.it/2014/01/k-means-clustering-calculator.html



Index	Case ID	Trace Fitness
1	491381	1
2	477447	1
3	493534	0,67
4	490259	0,61
5	492746	0,67
6	493724	0,75
7	483505	0,41
8	476904	0,65
9	479570	0,71
10	492998	0,56
11	490756	0,64
12	475754	0,67
13	485552	0,88
14	490875	0,88
15	456762	0,78
16	477089	0,92

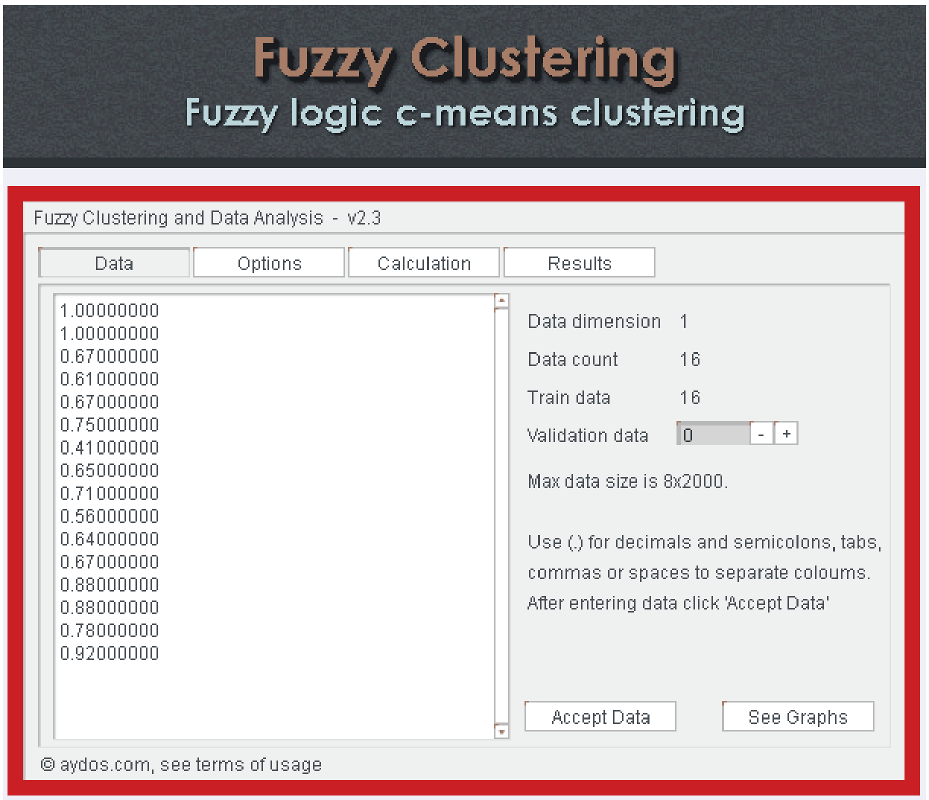
Separate cases into clusters according to fitness



Separate cases into clusters according to fitness

https://tinyurl.com/pdis-fcm

https://aydos.com/fcm/



Index	Case ID	Trace Fitness
1	491381	1
2	477447	1
3	493534	0,67
4	490259	0,61
5	492746	0,67
6	493724	0,75
7	483505	0,41
8	476904	0,65
9	479570	0,71
10	492998	0,56
11	490756	0,64
12	475754	0,67
13	485552	0,88
14	490875	0,88
15	456762	0,78
16	477089	0,92

Accept data > Options > Calculation > Results

Cluster centers:

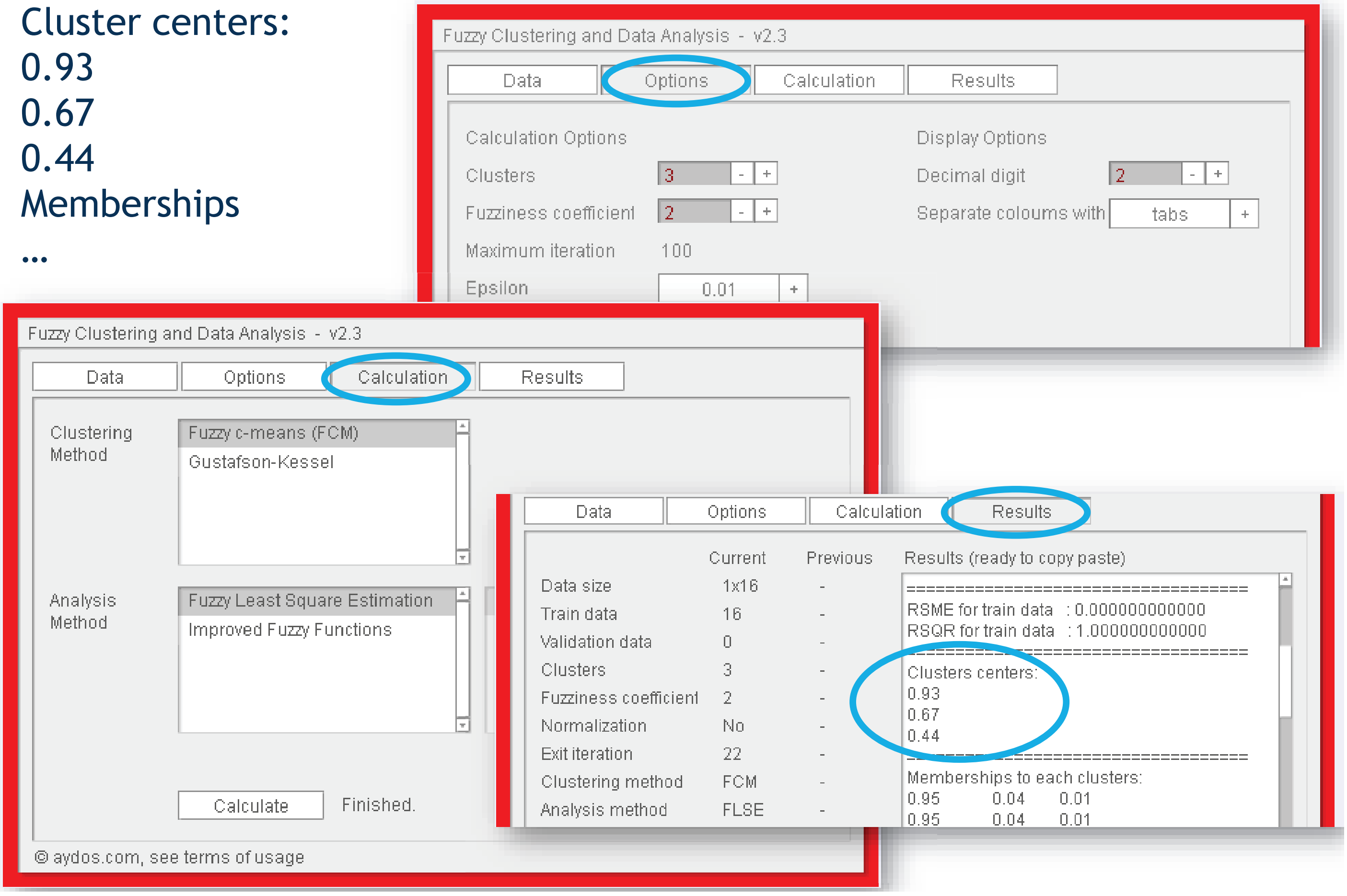
0.93

0.67

0.44

Memberships

...



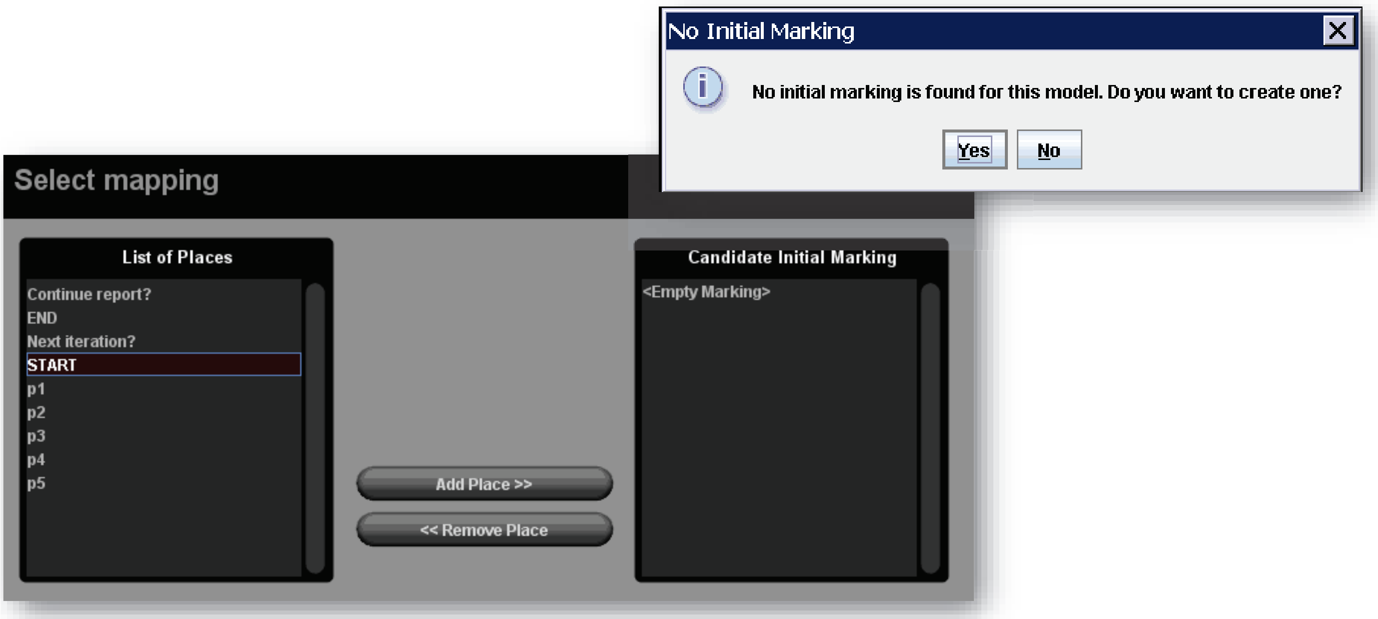
Exercise: Simplified map transitions to event classes

a) Import the senior bpmn.xml file in Visual Paradigm and remove the pool

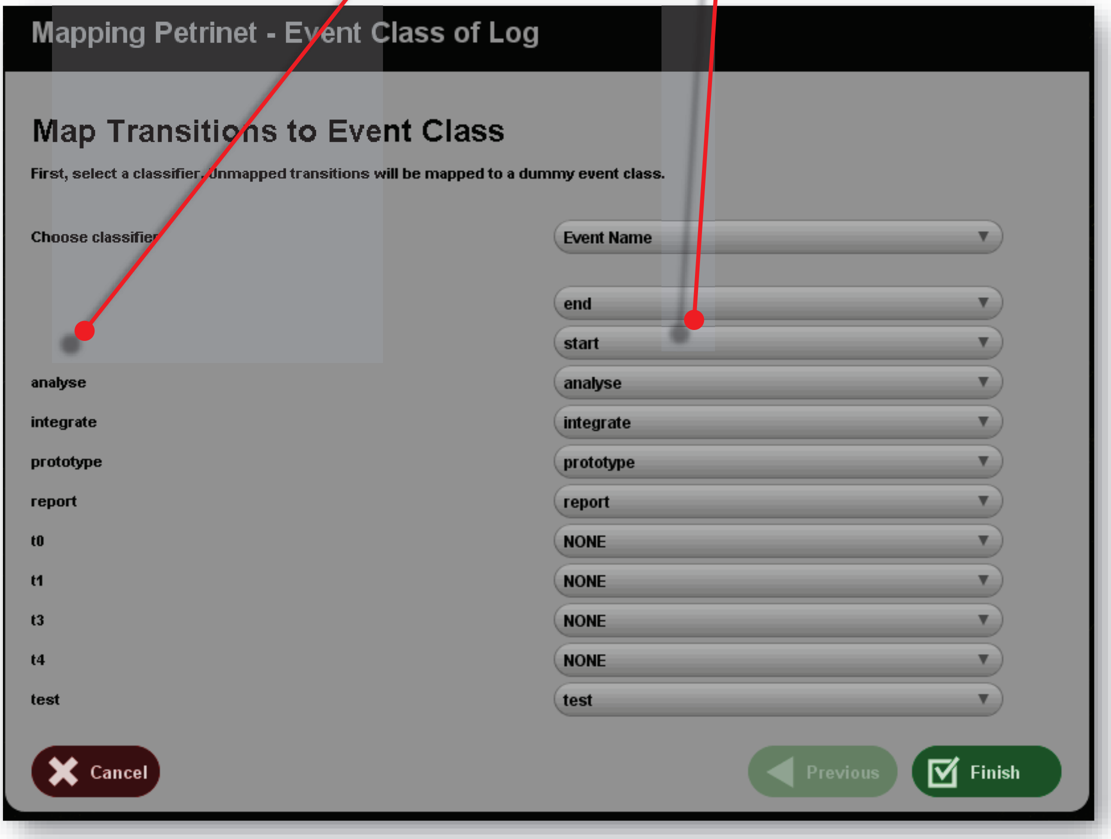
b) Export the new BPMN as senior bpmn nopool.xml

c) Convert the model to Petri Net via “**Convert BPMN to Petrinet**”
the resulting Petri Net is more compact

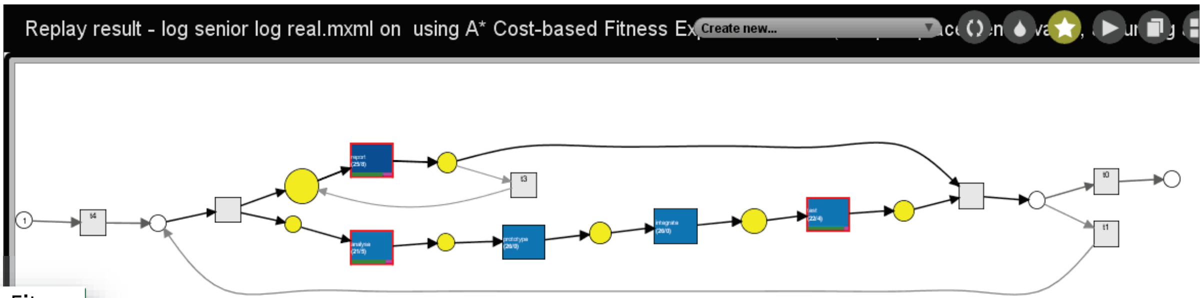
d) Carry out the “Replay a Log on Petri Net for Conformance Analysis”
The map transitions to event classes is simplified.



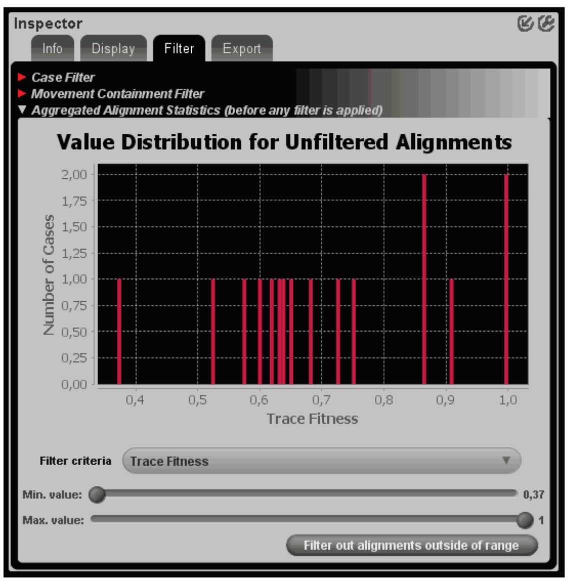
The transition name may not appear due to label errors in the BPMN



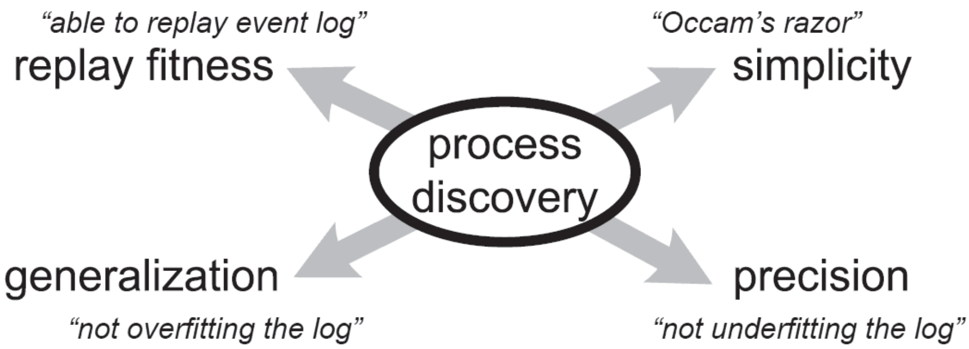
Inspect individual cases, compare the trace fitness w.r.t. the other Petri Net



Case IDs	Trace Fitness
477447	1
491381	1
477089	0,91
485552	0,87
490875	0,87
456762	0,75
493724	0,73
479570	0,68
493534	0,65
492746	0,64
475754	0,63
476904	0,62
490756	0,6
490259	0,58
492998	0,52
483505	0,37



The four quality dimensions for process discovery



The **Fitness** of a mined Model (M_m) meaures how much of the observed behavior in the log (L) is captured by the process model. Good fitness (close to 1) allows the replay of (most of) the behavior seen in the event log

E.g. the fraction of event patterns represented by the model, the fraction of cases that can be replayed in the model)

$$f(L, M_m) = \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{k} \mathrm{n_i}\, \mathrm{m_i}}{\sum_{i=1}^{k} \mathrm{n_i}\, \mathrm{c_i}}\right) + \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{k} \mathrm{n_i}\, \mathrm{r_i}}{\sum_{i=1}^{k} \mathrm{n_i}\, \mathrm{p_i}}\right)$$

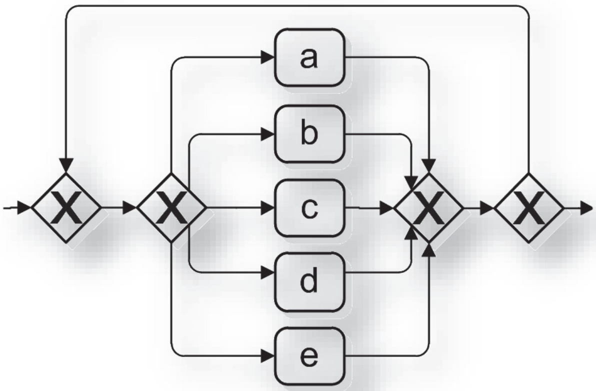
k # of different traces.
$\mathrm{n_i}$ # of traces of type k in log L.
$\mathrm{m_i}$ # of missing tokens (artificially added) parsing i.
$\mathrm{r_i}$ # of remaining tokens parsing i.
$\mathrm{c_i}$ # of consumed tokens parsing i.
$\mathrm{p_i}$ # of produced tokens parsing i.

● **Simplicity** is measured by comparing the number of elements (i.e., #**gateways + #sequence flows + #activities**) of the model M with the number of activities (cardinality of) in the log.
In general it is measured by complexity metrics for process models such as model size or degree of structuredness.

● Fitness and simplicity are not enough to judge the quality of a discovered process model, as clearly shown in the following exercise.

- Exercise: Draw a model in BPMN with which you can replay any execution sequence that includes tasks *a, b, c, d, e*. Furthermore, discuss the fitness, simplicity, precision, and generalization of such a model for the trace $\langle a, b, c, d, e \rangle$.

- Solution:

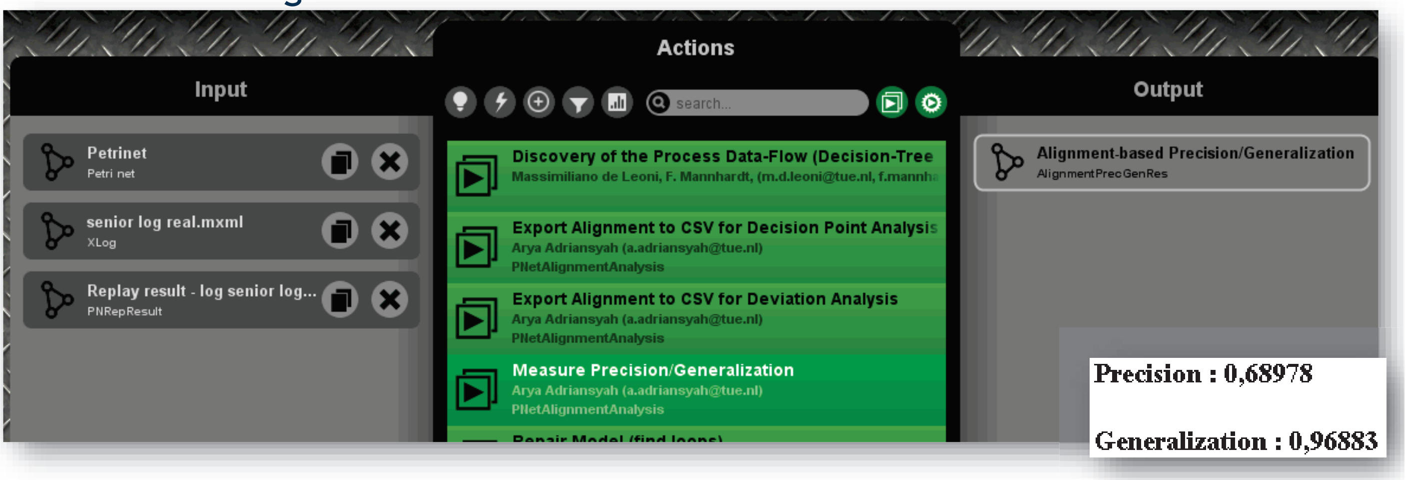


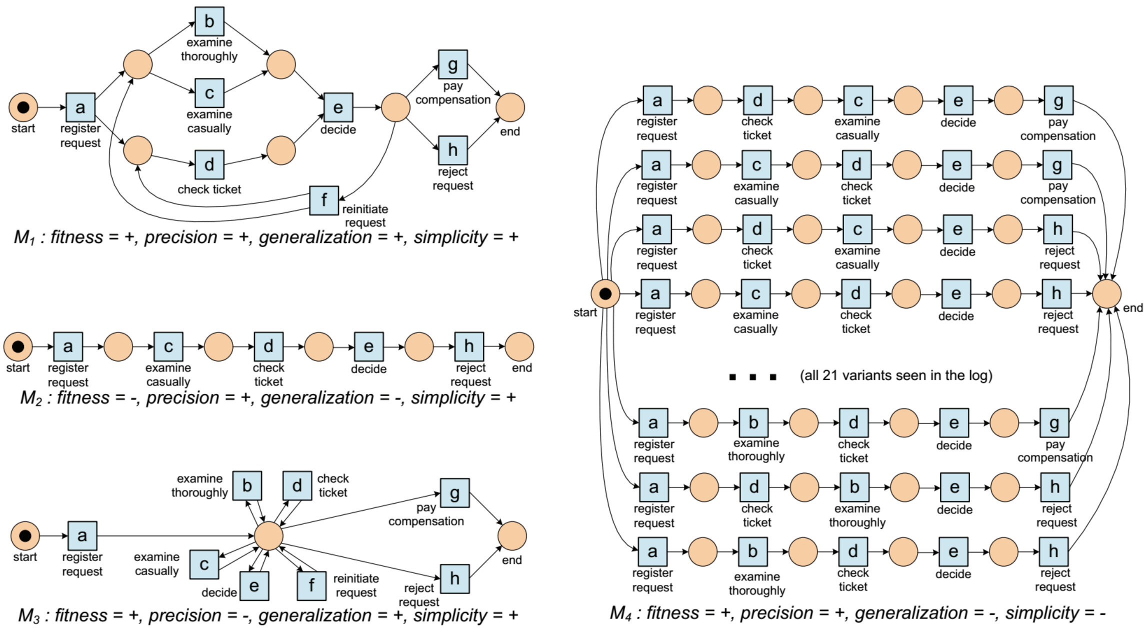
- Fitness: it is perfect to the execution sequence as it is able to replay the occurrence of *a* to *e*
- Simplicity: the model is not completely simple as it includes four gateways for characterizing the behavior of five activities.
- **Precision**: The discovered model should not allow for behavior very different from what was seen in the event log. The solution is not very precise because it does not introduce specific constraints on the behavior: any occurrence of *a* to *e* is allowed at any stage.
- A simplified formula for the precision of a mined model (M), assuming that {enL} is included in {enM}:
  if the enM >> enL then the P is low (too much extra behavior)

$$\mathrm{P}\,(L, M) = \frac{1}{|\varepsilon|} \sum_{e \in \varepsilon} \frac{|\mathrm{en}_{\mathrm{L}}(e)|}{|\mathrm{en}_{\mathrm{M}}(e)|}$$

ε	collection of *unique events* in a context of the log.
$\mathrm{en}_{\mathrm{M}}(e)$	enabled activities in the model M.
$\mathrm{en}_{\mathrm{L}}(e)$	observed activities actually executed in a similar context in L.

- **Generalization**: as opposed to precision, this criterion focus on avoiding overly precise models, since event logs are often far from complete.
- It refers to the ability of the model to abstract. The solution model does not constrain the behavior, and then there is hardly any general insight that we can learn from it.
- To calculate precision and generalization with ProM:
  after the Replay result generated by conformance cheking, in the workspace select Replay Result, Petri net and the Log
  Then select "Measure Precision/Generalization" and continue with default settings





trace
acdeh
abdeg
adceh
abdeh
acdeg
adceg
adbeh
acdefdbeh
adbeg
acdefbdeh
acdefbdeg
acdefdbeg
adcefcdeh
adcefdbeh
adcefbdeg
acdefbdefdbeg
adcefdbeg
adcefbdefbdeg
adcefdbefbdeh
adbefbdefdbeg
adcefdbefcdefdbeg

Other useful ProM plugins (alternative to Disco)

a) Log filtering
- *"CSV File (XES Conversion with Log package)"*: to import csv logs
- *"Convert CSV to XES"*: to transform the csv log into a XES log
- *"Move trace level attributes from events to trace (In Place)"*: to set trace attributes in the XES
- *"Filter Log on Trace Attribute Values"*: to filter trace attributes
- *"Filter Log using Simple Heuristic"*: to remove cases on the basis of the activities frequency
- *Add Artificial Events*, to add start/end tasks when missing

b) Model miners
- *"Mine for a Fuzzy Model": it generates a transition map*
- *"BPMN Miner" > "Heuristics Miner ProM6": it extracts compact models*