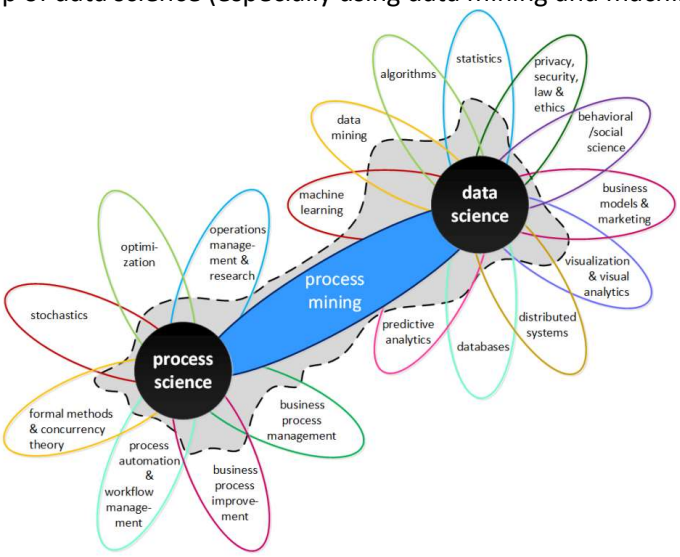
Full Summary

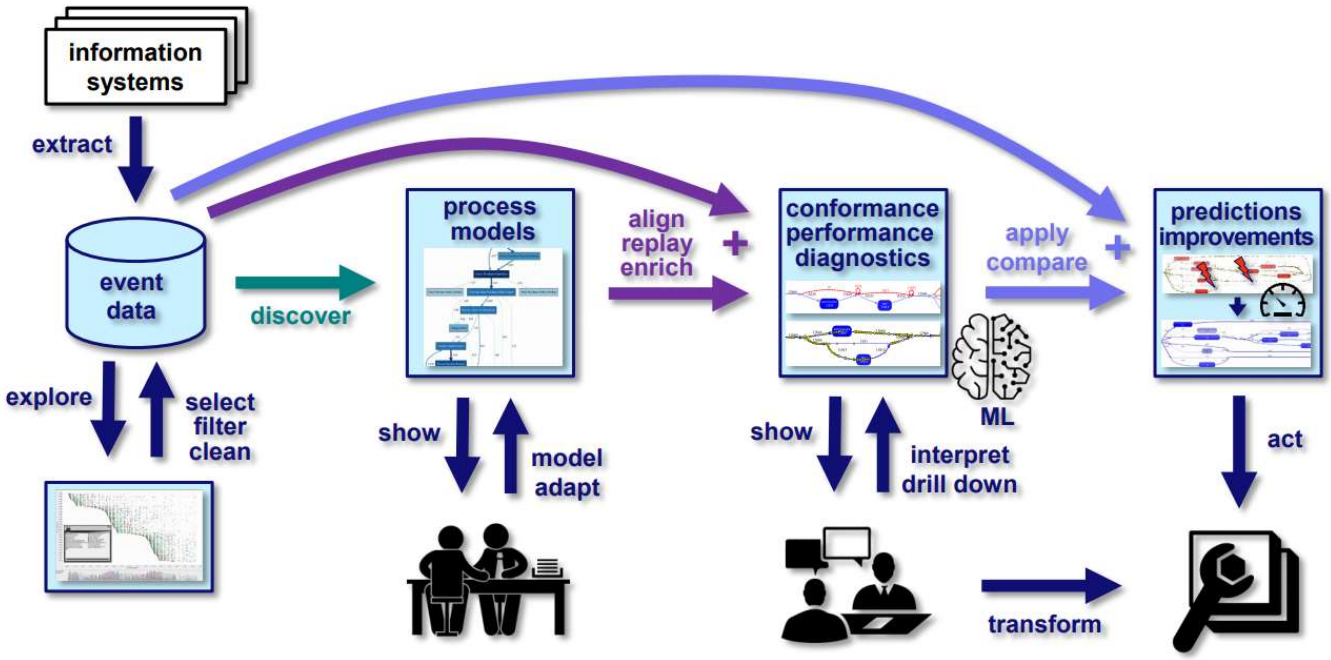
Saturday, 21 August 2021 09:53

Introduction

Process mining combines the fields of process science and data science. It aims at improving discrete processes with the help of data science (especially using data mining and machine learning).



Top down view



1. Data is **extracted** from information systems. Note that we have only observed a subset of all possible behavior
   a. This data is then **explored** and **cleaned** before it is stored as **event data** inside a database.
2. The next step is to **discover** a process model representing the event data.
   a. The model can be **normative**, showing what should happen, or they can be **discovered** showing what has really happened.
   b. The process model might need to be **adapted**. With the help of event data, the model is **replayed**, **enriched** and **aligned**.
3. We use **apply** Machine Learning on the model to **predict** future behavior and give suggestions of possible **improvements** to the information system.

Event data

The starting point is always the event data. Event data always needs to contain the following columns:

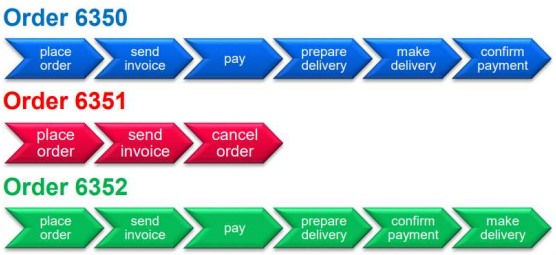
- Case ID
- Activity
- Timestamp

If those fields are present, then we can construct **traces** from the event data. The variants of different traces are called trace variants.

Trace

A **trace** is a sequence of activities referring to a particular case. The following event log contains three trace variants.

Case ID	Activity	Timestamp
6350	place order	2018/02/13 14:29:45.000
6351	place order	2018/02/13 16:17:37.000
6352	place order	2018/02/13 17:53:22.000
6352	send invoice	2018/02/19 09:20:28.000
6351	send invoice	2018/02/19 16:08:07.000
6350	send invoice	2018/02/21 09:38:16.000
6350	pay	2018/03/02 12:39:37.000
6352	pay	2018/03/05 15:46:47.000
6351	cancel order	2018/03/06 10:17:01.000
6350	prepare delivery	2018/03/07 13:50:35.000
6350	make delivery	2018/03/07 16:41:01.000
6350	confirm payment	2018/03/07 16:53:00.000
6352	prepare delivery	2018/03/07 17:05:59.000
6352	confirm payment	2018/03/07 17:59:55.000
6352	make delivery	2018/03/08 09:54:36.000

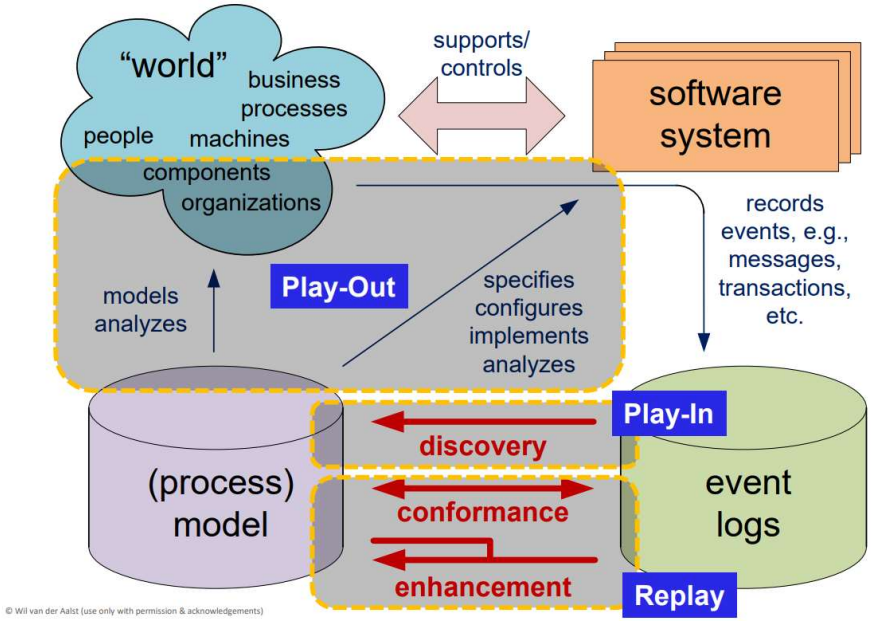


Play-In / Play-Out, Replay

Play out: *Start* from the model and *generate* data.

Play in: *Start* from the event data and *generate* the model

Replay: Start with the event data and the model and check if the traces can be generated from the model. If a trace can be generated (replayed) by the model, then we say that the trace is **fitting** the model.

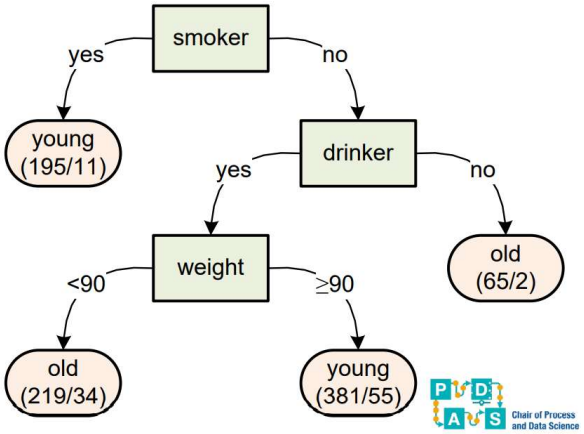


In process mining, decision trees can be used to determine why a certain path in a process model is taken.

Decision Trees

A decision tree is a model of a dataset, where each inner node is a **predictor variable** and the leaves

are the **response variables**. Each branch has a label which categorizes the predictor variable. The leaves contain labels which tell the predicted class. The leaves also tell us the total number of items which the node contains and for how many of them the wrong class was determined, usually as a (n/w) notation, n being the total amount and w the amount which are wrongly classified



Example of a decision tree with predictor variables in green and response variables in red
Notation: "young (195/11)" means 195 items were classified as young while 11 are actually old.

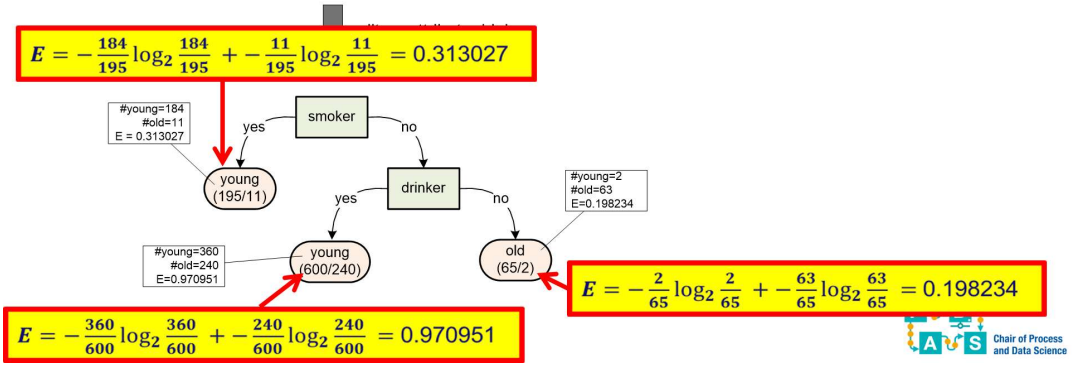
Decision tree learning

The basic idea behind decision tree learning is that you want to split the set of instances into subsets such that the variation within each subset becomes smaller. The variation is measured by **entropy**. The entropy E is defined as follows:

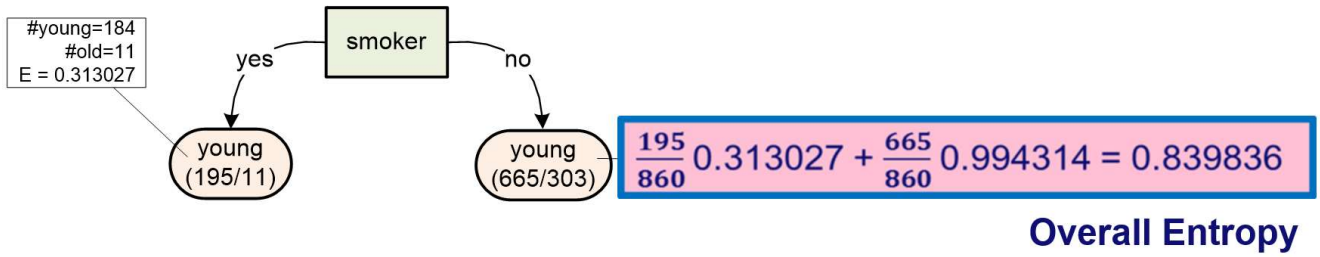
$$\mathrm{E} = -\sum_{\mathrm{i}=1}^{k} p_i \log_2(p_i)$$

With:

- k : all the possible values enumerated
- $p_i = \frac{c_i}{n}$ is the fraction of elements having value i
  - $\mathrm{c_i}$ is the number of i values
  - $\mathrm{n} = \sum_{\mathrm{i}=1}^{k} c_i$ the number of different values



The entropy for a subtree is the **weighted average** of its nodes



Information gain

The goal of a decision tree is to have the lowest entropy possible. We split a node on the attribute which **reduces** entropy the most. We call the reduction of entropy the **information gain**.

Decision tree algorithm

1. **Start with root node** corresponding to all instances.
2. **Iteratively traverse** all nodes to see whether "information gain" (i.e., reduction of entropy) is possible.
    a. For each node and for every attribute, check what the effect of splitting the node is in terms of information gain.
    b. **Select the attribute with the biggest information gain** above a given threshold.
3. **Continue until no** significant **improvement** is possible.
4. **Return** the decision tree.

Different parameters and variations are possible.

- The minimal size of a node **before or after** splitting
- Threshold for minimum information gain
- Maximal depth of the tree
- Allowing the same label to appear multiple times or not
- Alternatives to entropy, e.g., **Gini index** of diversity. $G = \sum_{i=1}^{k} p_i(1 - p_i) = 1 - \sum_{i=1}^{k}(p_i)^2$ with $p_i = \frac{c_i}{n}$
- Splitting the domain of a numerical variable.
- Post pruning: removing leaf nodes that do not significantly increase the discriminative power

Difference between process mining and data mining

Process mining techniques are process-centric, while data mining techniques are typically not. Process mining techniques cover topics such as process discovery, conformance checking and bottleneck analysis, which are not addressed by traditional data mining techniques.
End-to-end process models and concurrency are essential for process mining. Process mining assumes that the event logs contains events with timestamps referring to cases.

Data sets in Data mining

The data set consists of variable instances. Variables are often referred to as attributes, features, or data elements. There are two types of variables:

- **Categorical** variables
    - **Ordinal**: order can be established between categories
    - **Nominal**: no order can be established between different categories
- **Numerical** variables

Supervised learning

In supervised learning, we use labelled data (i.e. there is a response variable that labels each instance. The goal is to explain the **response** variable in terms of the **predictor** variables. The response variable is called the dependent variable, while the predictor variables are called the independent variables.

Classification techniques

Classification techniques assume a **categorical** response variable. The goal is to classify instances based on the predictor variables.

Regression techniques

Regression techniques assume a **numerical** response variable. The goal is to find a function that fits the data with the least error.

Unsupervised learning

Unsupervised learning assumes **unlabeled data**, which means that the variables are not split into response and predictor variable. Examples include: **clustering** and **pattern discovery**.

Association rules

An association rule is a formula $X \Rightarrow Y$, for sets X, Y. It tells us interesting rules about customer

behavior that apply in a particular context. In this case, the formula $X \Rightarrow Y$ means that customers which buy X usually also buy Y.

Support, confidence and lift

The **support** of a rule is defined as $\text{support}(X \Rightarrow Y) = \frac{N_{X \wedge Y}}{N}$, where N is the number of instances and $N_{X \wedge Y}$ the number of instances containing both X and Y

The **confidence** of a rule is defined as $\text{confidence}(X \Rightarrow Y) = \frac{N_{X \wedge Y}}{N_X}$

The **lift** of a rule is defined as $\text{lift}(X \Rightarrow Y) = \frac{N_{X \wedge Y} \cdot N}{N_X \cdot N_Y}$

We are interested in rules that have a **high** support, a confidence close to 1 and that have a lift **higher** than 1 (positive correlation)

Mining association rules

The problem with enumerating every possible rule and then calculating the support, confidence and lift for each of them is computationally expensive.
We use the following observation (**Apriori property**) to limit the search space:

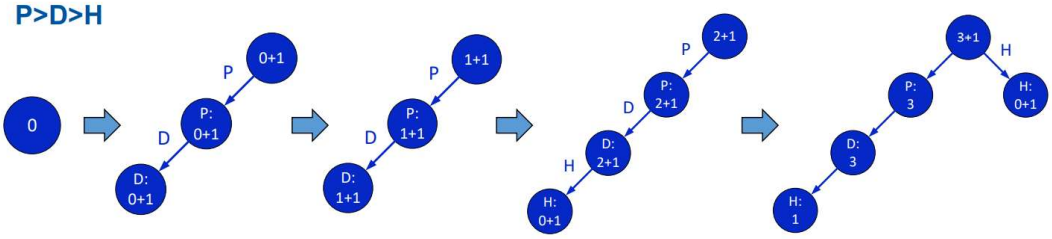
$$Y \subseteq X \Rightarrow \frac{N_Y}{N} \geq \frac{N_X}{N}$$

- If X is frequent then all **subsets** of X need to be frequent.
- If Y is infrequent, then all **supersets** of Y are infrequent.

Frequent Pattern (FP) growth

Algorithm: We build a so called FP tree.

1. **Count and remove infrequent items**: We pass through the data and calculate how frequent a certain item appears, we remove all items that do not exceed the minimal support value (which can be specified by the user)
2. **Sort by** global **frequency**: The individual items are sorted such by their frequency.
3. **Build a tree**:
   a. Take the most frequent node.
   b. Take all the projected itemsets (=take only the instances which contain the particular node).
   c. For each itemset X, let $C(X)$ be the **chain of items of X** sorted by their frequency.
      - If the chain already exists in the tree, increment the counter for each node in the chain.
      - If the chain does not exist yet, add it or extend it accordingly in the tree

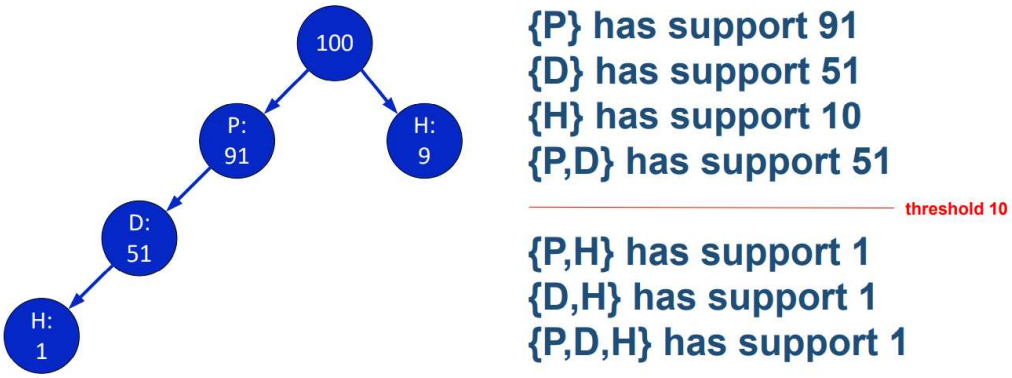


Example of FP growth tree building

To prune the FP tree, **conditional** FP trees are needed which are not covered in the lecture.

Support in FP tree

From the resulting tree, we can easily determine the support of items by summing up the frequencies in each node.
To determine the support of an itemset, we first search for the associated chain in the tree. The support is given by the last node in the chain.



Other types of pattern mining

- **Sequence mining**: Order of items is important
- **Episode mining**: Partial order of items (e.g. "Before doing X, people do Y and Z")
- **Local process models (LMP)**: Smaller process fragments, typically between three and five activity nodes. A LPM does not aim to describe all behavior in a trace completely, but instead aims to describe traces partially

  From <https://www.google.com/search?q=local+process+models&oq=local+process+models&aqs=edge..69i57j0i22i30l2.4473j0j4&sourceid=chrome&ie=UTF-8>

Clustering

Clustering aims to group items into clusters, such that items within the group are similar and items from different clusters are not. We can use clustering to split the event log and thereby might get better process models.

K-means clustering

Let k be a user defined constant which gives the number of clusters that we want to discover.
Algorithm:

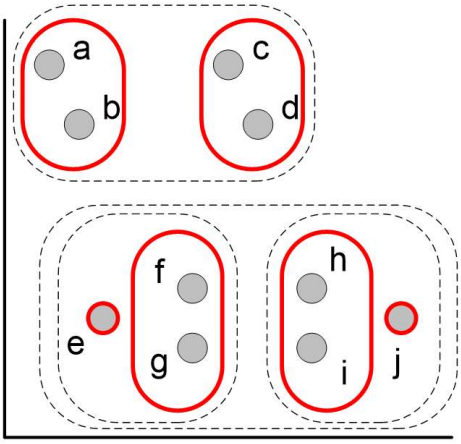
1. We **start with k centroids** (=points), which can be randomly or regularly positioned.
2. **Assign** each **item to the closest centroid** (similarity measure needs to be given, e.g. Euclidean distance). Distances for each attribute should be normalized to give equal weight to each attribute in the distance measure.
3. **Reposition the centroid** by averaging over all items assigned to the centroid
4. Go to step 2 **until a fixpoint** is reached

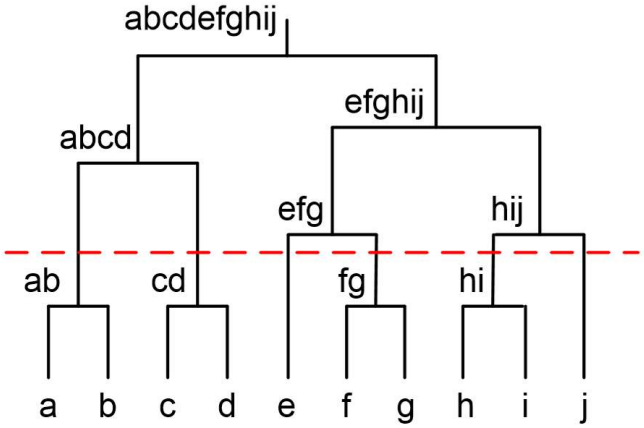
Each grouped instance represents one cluster.

A variation of k-means is **k-mediod**: Instead of taking mean values, we take the median. We start with random or regular items. The advantage is that at any step the mediod is an item of the cluster.

Other clustering techniques

- **Agglomerative hierarchical clustering** takes a bottom-up approach. We start by grouping each item with its closest neighbor. For each group, we take the mean value. We contiue grouping closest groups (by mean value of all items). The result is represented as a dendrogram. We can cut the dendrogram at a specific height to get the desired number of clusters





dendrogram

- Density-Based Spatial Clustering of Applications with Noise (**DBSCAN**): connected neighborhoods.

Evaluating mining results

Confusion matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

From <https://www.google.com/search?q=confusion+matrix&oq=Confusion+ma&aqs=edge.0.0i512j69i57j0i512l5.4823j0j1&sourceid=chrome&ie=UTF-8>

		predicted class	
		+	-
actual class	+	TP	FN
	-	FP	TN

- True Positives (TP): positive instances predicted to be positive .
- True Negatives (TN): negative instances predicted to be negative.
- False Positives (FP): negative instances predicted to be positive.
- False Negatives (FN): positive instances predicted to be negative.

Quality measures

Let K be the total number of instances, $P = TP + FN$ the number of actual positves, $P' = TP + FP$ be the number of positive predicted instances.

- $\text{error} = \frac{\text{FP} + \text{FN}}{\text{K}}$
- $\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{K}}$
- $\text{precision} = \frac{\text{TP}}{\text{P}'} = \frac{\text{TP}}{\text{TP} + \text{FP}}$
- $\text{recall} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$
- $\text{F1\_score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$

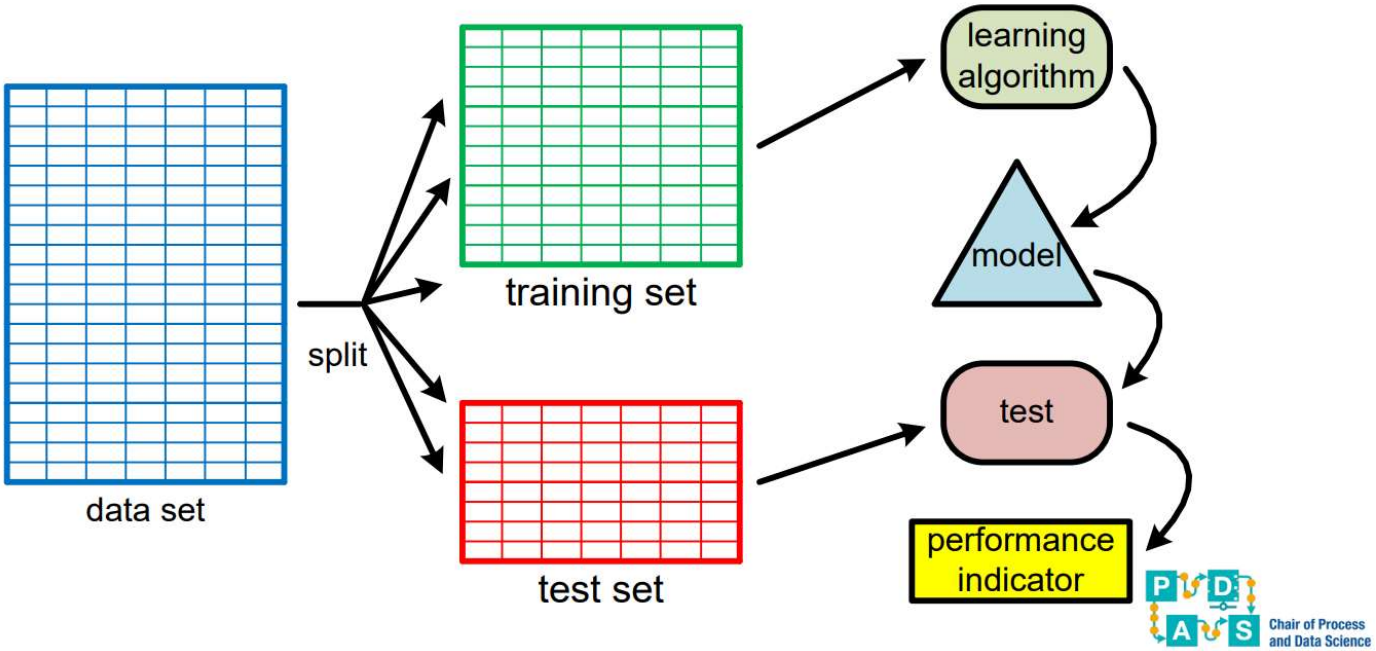
Overfitting and Underfitting

Overfitting: the model is too specific for the data set used to learn the model and performs poorly on new instances.
Underfitting: the model is too general and does not exploit the data.

We can prevent over- and underfitting by using **Cross-validation**

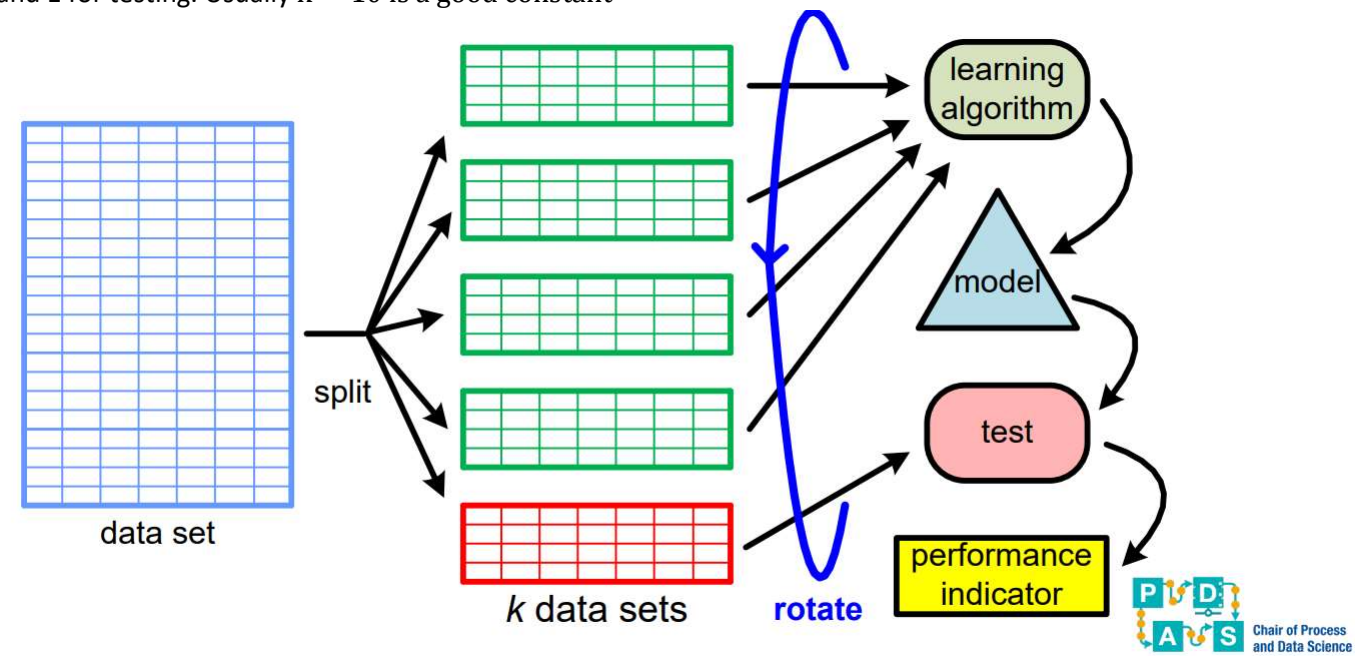
Cross-validation

Split the data set on **training** and **test** subsets. The training set is used to train the model and the model is then evaluated on the test set.



K-fold cross validation

All of the data is used for training **and** testing. Data is split data into k folds. Use k-1 folds for training and 1 for testing. Usually $k = 10$ is a good constant



Complications

We are always training on data of the past, but data might have already changed after we have created the model. Furthermore, we might only have positive examples (e.g. We only learn about sick customers that have complained)

Process discovery

The difference between case and event attributes is that case attributes don't change over time, while event attributes are related to a particular step in the process.
The event log is often stored in XES form.

Extensible Event Stream (XES)

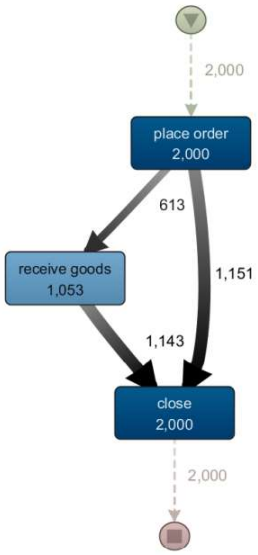
Official IEEE Standard driven by the IEEE Task Force on Process Mining. The xml syntax and OpenXES library are available under http://www.xes-standard.org/

Directly follows graph (DFG)

From the event log, we normally construct a directly follows graph (DFG) in which the nodes are the different activities. Two nodes are connected by an edge if there exists a trace in the event log, where the one node is directly followed by the other node. Note that for some processes a lot of different variants are possible. In this case, we get a spaghetti model of DFG which is difficult to analyze.

Challenges of DFG

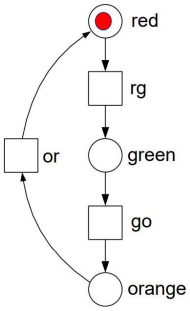
To prevent spaghetti models, commercial tools often provide sliders which modify the thresholds for number of activities or frequency of paths, but those filtering algorithms can falsify the model. Furthermore, DFGs cannot model concurrency.



Example: We see receive goods is skipped 1151. The real number should be $2000 - 1053 = 947$

Petri net notations

The network is composed of **places** and **transitions**. Places hold (one or multiple) **tokens**. Transitions produce and/or consume tokens.



- A transition is **enabled** if **each** input place contains at least one token.
- An enabled transition can **fire** by consuming a token from each input place and producing a token for each output place. The firing of a transition is atomic.
- A place p is k-bounded if there is no reachable marking with more than k tokens in p. A Petri

net is k-bounded if all places are k-bounded. It is unbounded, if no such k exists.

- A place/Petri net is bounded if there exists such a k.
- A petri net is safe if it is 1-bounded.

- A marking is dead if no transition is enabled from it.
- A Petri net has a potential **deadlock** if there is a reachable dead marking. A Petri net is deadlock-free if each reachable marking enables at least one transition
- A transition t is live if from any reachable marking it is possible to reach a marking that enables t. A Petri net is live if all transitions are live. A Petri net that is live is deadlock-free.

How to remove behavior from a petri net:

- Remove transitions
- Add places
- Remove arcs from transitions to places
- Add arcs from places to transitions

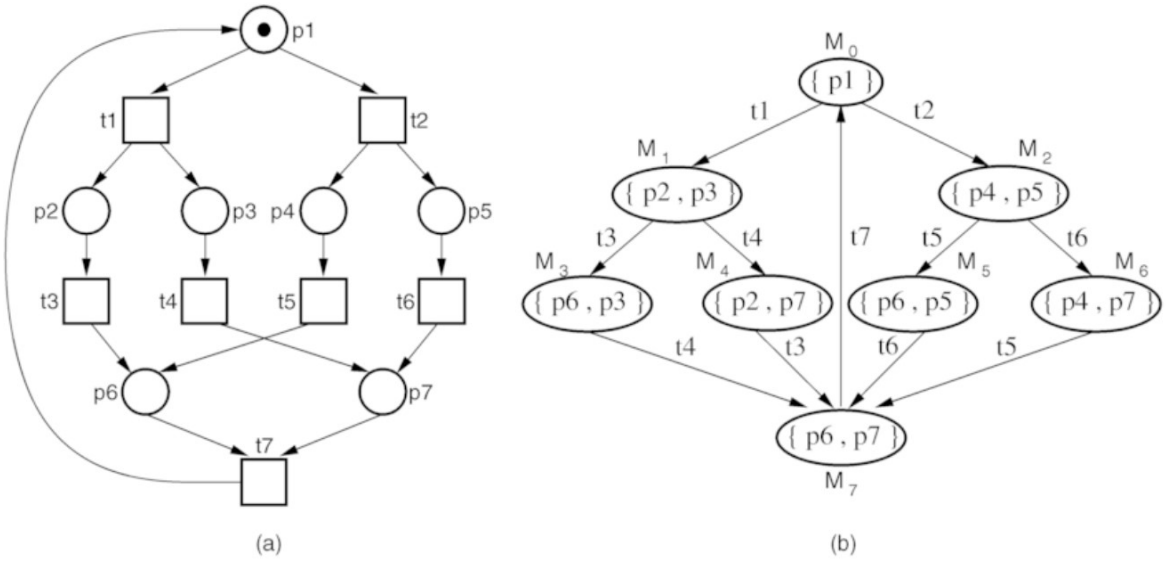
How to add behavior to a petri net:

- Add transitions
- Remove places
- Add arcs from transitions to places
- Remove arcs from places to transitions

We can construct a reachability graph from a petri net.

Reachability graph

The reachability graph is a transition system with one initial state (initial marking) and no explicit final marking. It describes the reachable markings of the petri net. The reachability graph might be infinite.



Workflow nets (WF-nets)

Workflow nets are a **subclass of Petri nets** often used in the context of workflow management and business process management. WF-nets have a well defined start **(source)** and end **(sink)**. All other nodes should be on a path from source to sink . The WF-net should be free of obvious anomalies (**soundness**).

A WF-net is sound if and only if the following properties hold:

1. **safeness**: places cannot hold multiple tokens at the same time (i.e. 1-bounded),
2. **proper completion**: if the sink place is marked, all other places are empty,
3. **option to complete**: it is always possible to reach the marking that marks just the sink place, (implies proper completion)

4. **absence of dead parts**: for any transition there is a firing sequence enabling it

Link between soundness and classical Petri net properties

A WF-net is sound if and only if the corresponding "short-circuited" Petri net is live and bounded! We get the short-circuited Petri net by adding a transition from the end to the start place.

We can use the Alpha Algorithm to discover WF-nets from event logs.

Alpha Algorithm

The first step is to simplify the event log by representing the event log as a multiset of traces where every trace is a sequence of activities ordered by their timestamp.

$$L_1 = [\langle a,b,c,d\rangle^3, \langle a,c,b,d\rangle^2, \langle a,e,d\rangle]$$

Example of an event log with six traces and three trace variants

Relationships

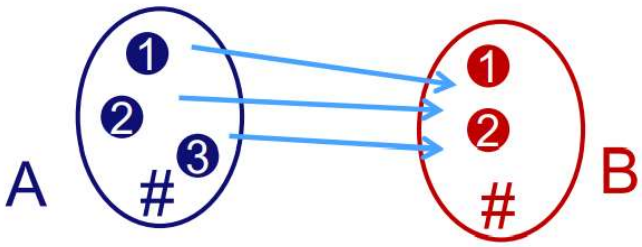
- **Direct succession**: $x > y$ iff for some case x is **directly** followed by y
- **Causality:** $x \rightarrow y$ iff $x > y$ and not $y > x$
- **Parallel:** $x||y$ iff $x > y$ and $y > x$
- **Choice:** $x\#y$ iff not $x > y$ and not $y > x$

Algorithm

The first thing that the Alpha algorithm does is determine the relationships between each pair of activities. The resulting matrix is called the **footprint matrix** of the log.

Let L be an event log over T. Then $\alpha(L)$ is defined as follows:

1. Each activity in L corresponds to a transition in $\alpha(L)$:
$$T_L = \{ t \in T \mid \exists_{\sigma \in L} t \in \sigma \}.$$
2. Fix the set of start activities:
$T_I = \{t \in T \mid \exists_{\sigma \in L}\, t = first(\sigma)\}$
3. Fix the set of end activities:
$T_O = \{t \in T \mid \exists_{\sigma \in L}\, t = last(\sigma)\}$
4. Calculate pairs (A, B) of sets of activities such that every element $a \in A$ and every element $b \in B$ are causally relatedi. e. , $a \rightarrow_L b$), all elements in A are independent $(a_1 \#_L a_2)$, and all elements in B are independent $(b_1 \#_L b_2)$:
$$X_L = \{ (A,B) | A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq TL \wedge B \neq \emptyset \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_L b \wedge \forall a_1, a_2 \in A\ a_1 \#_L a_2 \wedge \forall b_1, b_2 \in B\ b_1 \#_L b_2 \},$$



5. Delete non maximal pairs $Y_L = \{(A, B) \in X_L \mid \forall_{(A',B') \in X_L} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B')\}$.
6. Create places for every pair (A,B) as well as a start and end place:
$P_L = \{ p_{(A,B)} \mid (A,B) \in Y_L \} \cup \{i_L, o_L\}$,
7. Determine the flow relation: Connect each place $p_{(A,B)}$ with each element a of its set A of source transitions and with each element b of its set B of target transitions. In addition, draw an arc from the source place i_L to each start transition $t \in T_I$ and an arc from each end transition $t \in T_O$ to the sink place o_L.

$F_L = \{(a, p_{A,B}) \mid (A,B) \in Y_L \wedge a \in A\} \cup \{(p_{A,B}, b) \mid (A,B) \in Y_L \wedge b \in B\} \cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$

8. $\alpha(L) = (P_L, T_L, F_L)$

The alpha algorithm is a good baseline algorithm, but has a lot of limitations

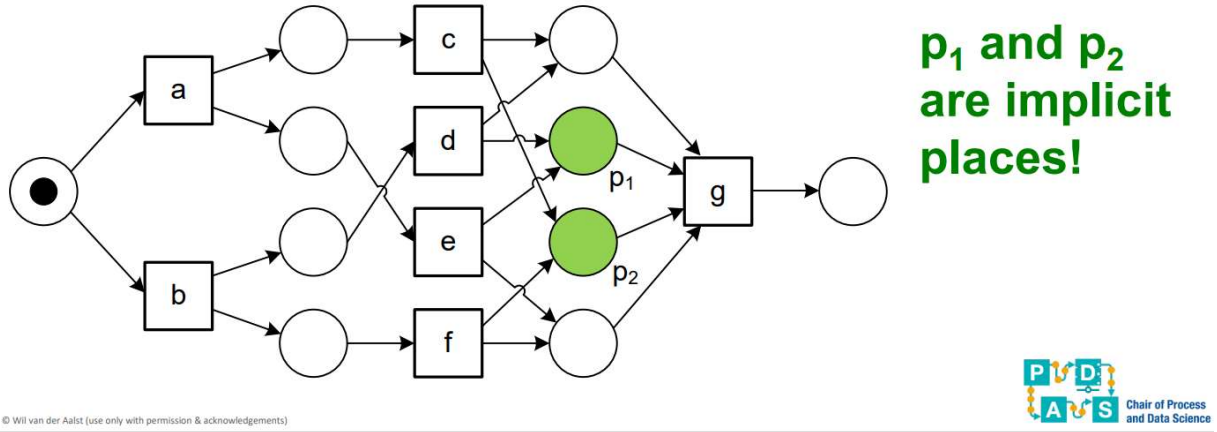
Limitations of the Alpha Algorithm

1. Implicit places
2. Loops of length 1
3. Loops of length 2
4. Non-local dependencies
5. Representation bias
6. Resulting model might not be sound
7. Noise
8. Incompleteness

Implicit places

Places in a petri net are **implicit**, if the same behavior can be simulated by the removal of those places.

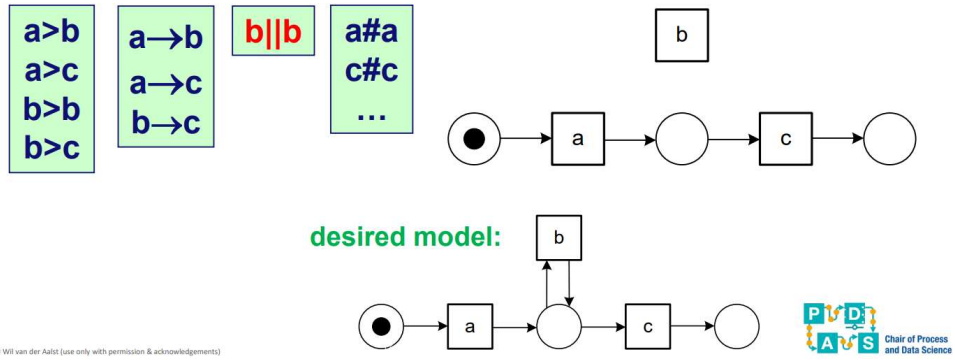
$$L_6 = [\langle a,c,e,g\rangle^2, \langle a,e,c,g\rangle^3, \langle b,d,f,g\rangle^2, \langle b,f,d,g\rangle^4]$$



This problem can be solved through postprocessing.

Loops of length 1

$$L_7 = [\langle a,c\rangle^2, \langle a,b,c\rangle^3, \langle a,b,b,c\rangle^2, \langle a,b,b,b,b,c\rangle^1]$$

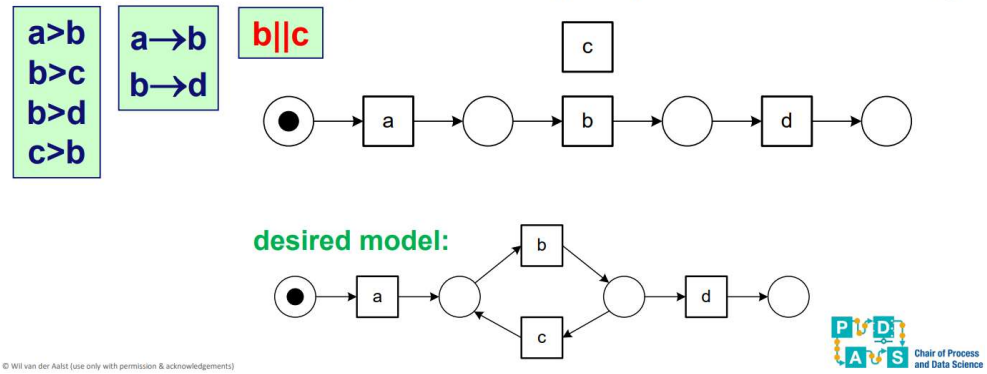


In this example the alpha miner produces the upper model, which allows for the execution of b at any time. What we would want is a model like the lower model where b can only fire after a has been fired.

This problem can be solved in multiple ways (change of algorithm or pre/post-processing).

Loops of length 2

$$L_8 = [\langle a,b,d\rangle^3, \langle a,b,c,b,d\rangle^2, \langle a,b,c,b,c,b,d\rangle]$$

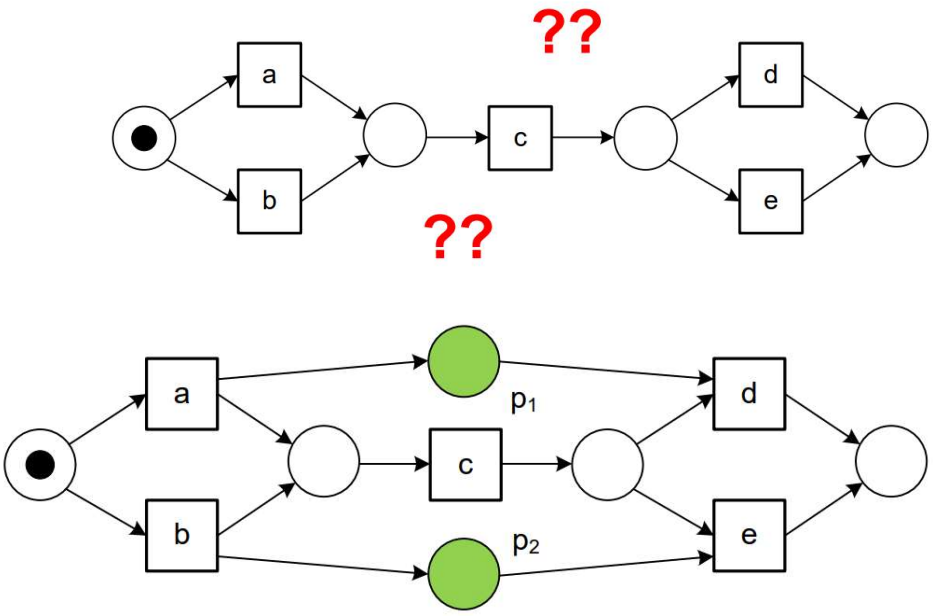


In this example the alpha miner produces the upper model, which allows c to be executed at any time. What we would want to have is a model like the lower model where c can only be executed after b.

This problem can be solved in multiple ways (change of algorithm or pre/post-processing).

Non-local dependencies

$$L_9 = [\langle a,c,d\rangle^{45}, \langle b,c,e\rangle^{42}]$$



In this example the upper model discovered by the alpha miner is able to produce the log, but it is underfitting, as it does not model the dependency between a and d. So in the upper model we could also produce a trace <a,c,e> or <b,c,d>. The lower model prevents such transitions.

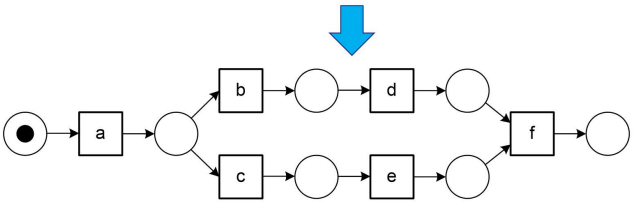
This problem is a foundational problem that is not specific to the alpha miner. If there is no such non-local dependency, we say that the net is a free choice net. The Alpha miner is able to discover free-choice nets.

Representation bias

The alpha algorithm generates a WF-net where each activity is only represented once. Thus <a,a> cannot be represented by a WF-net discovered by the alpha algorithm

Resulting model might not be sound

$$L = [\langle a, b, d, e, f \rangle^{10}, \langle a, c, e, d, f \rangle^{10}]$$



Example of an unsound WF-net produced by the alpha algorithm

General challenges of any mining technique

Noise

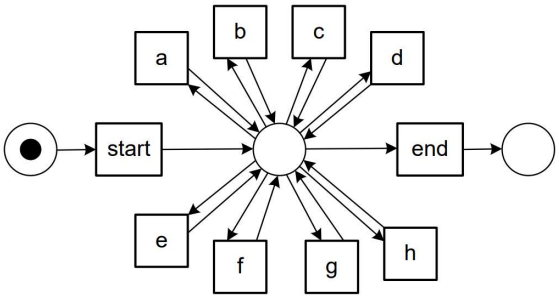
The event log contains rare and infrequent behavior not representative for the typical behavior of the process.

Incompleteness

The event log contains too few events to be able to discover some of the underlying control-flow structures

Noise and completeness are connected.

Flower Model



Any combination is possible as trace. The flower model is a model that can be easily constructed from each event log, but is too general to be useful

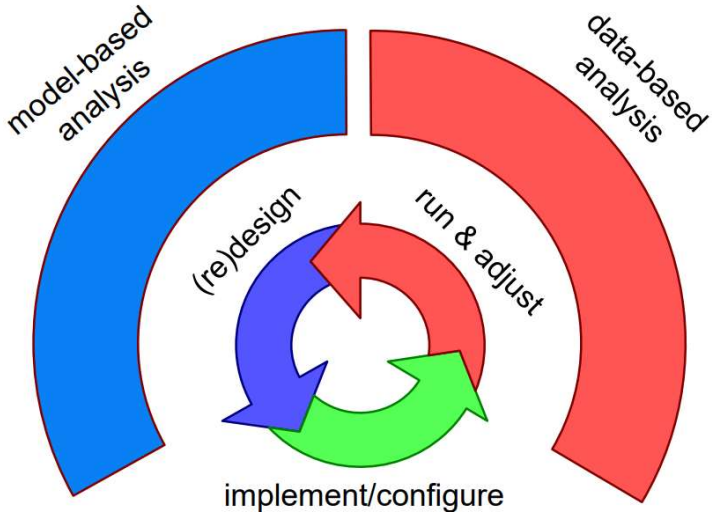
Business Process Management

Business Process Management (**BPM**) is the discipline that combines knowledge from information technology and knowledge from management sciences and applies this to operational business processes.

Robotic Process Automation (RPA)

Robotic process automation (RPA) is the practice of automating routine business practices with "software robots" that perform tasks automatically. These tasks include transaction processing, IT management and automated online assistants. These software robots could replace human beings for common tasks. Robotic process automation makes heavy use of machine learning to train these robots.

BPM lifecycle



Role of models in BPM and Workflow Management (WFM)

- reason about processes (redesign)
- make decisions inside processes (planning and control).

The Process models may be used to discuss responsibilities, analyze compliance, predict performance using simulation, and configure a WFM/BPM system.

Limitations of model-based analysis

Verification and performance analysis heavily rely on the availability of high quality models. However, if the models and reality have little in common, model-based analysis does not make much sense. Furthermore, there is often a poor alignment between hand-made models and reality.

Therefore Process mining aims to address these problems by establishing a direct connection between the models and actual event data about the process

Quality of discovered models

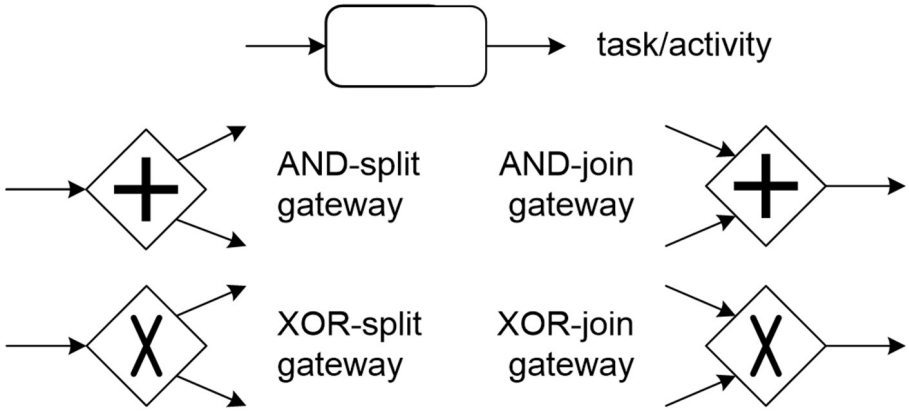
The problem with traditional data mining quality criteria is that the event log typically only shows a fraction of possible traces. Thus we cannot calculate things like recall and precision, as TP, FN,FP,... Is not known.

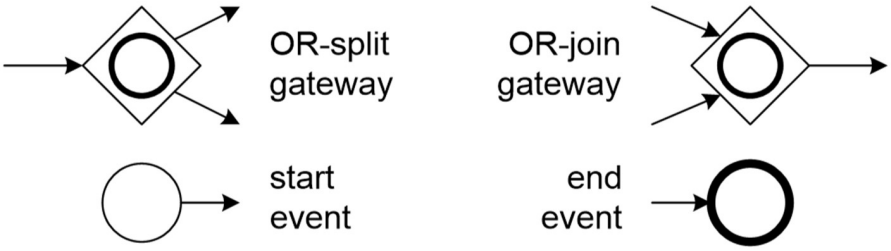
Replay fitness

$$\text{replay\_fitness} = \frac{\text{TP}'}{\text{TP}' + \text{FN}'}$$

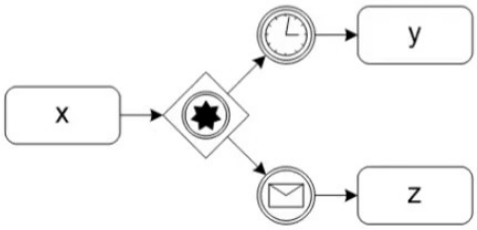
Where TP' and FN' are the True Positives resp. False Negatives taking the event log as real world behavior in the confusion matrix

Business Process Model and Notation (BPMN)





More notations are available which are not covered in the lecture:



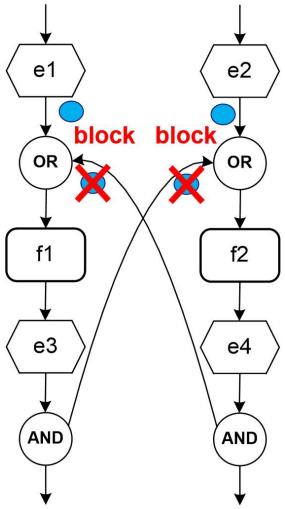
deferred choice pattern using the event-based XOR gateway

If a certain message does not arrive before a certain timeout, then y is executed.
If the message arrives on time, z is executed.

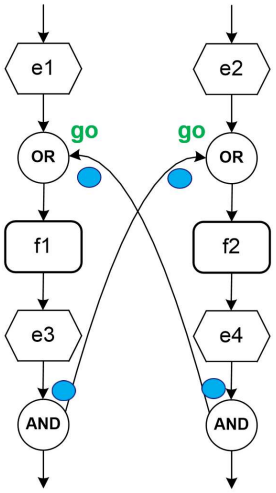
BPMN models can be converted into petri net models and vice-versa, but the conversion might be lossy.

Vicious cycle paradox

If one blocks, both should block (due to symmetry). If both block, there will never be a second token. Hence, the choice to block was wrong.



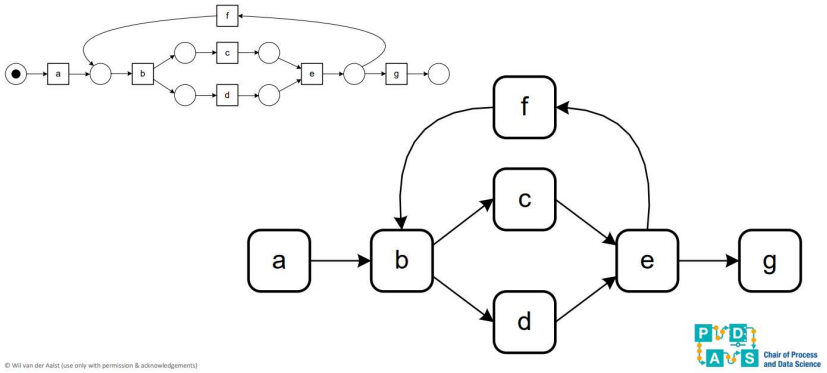
If one is not blocked, both can potentially progress (due to symmetry).
If both progress, there will be a second token. Hence, the choice to progress was wrong.



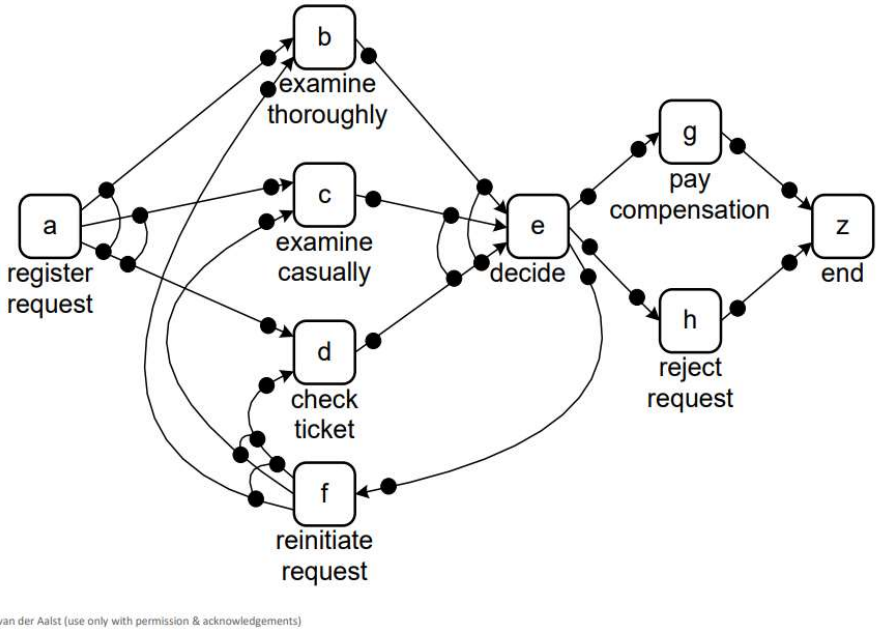
Paradox: all choices are wrong.

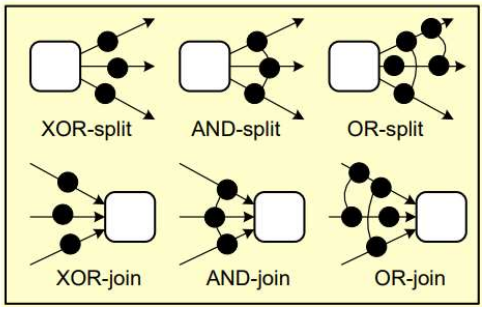
Dependency Graphs

Any footprint matrix defines a dependency graph. If we take each activity as a node and each directly follows relationship as an edge, we get the so called dependency graph. The dependency graph can also be constructed from the petri net by removing the places.

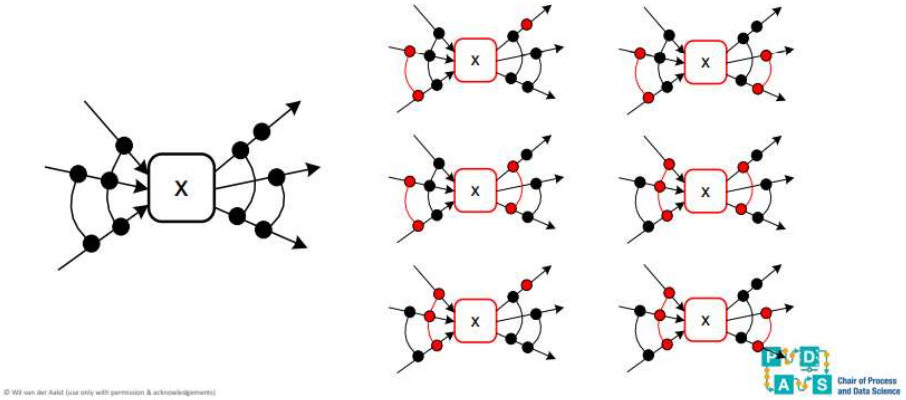


Causal nets





Connected dots define so called bindings. Each node has a set of **input bindings** and a set of **output bindings**.



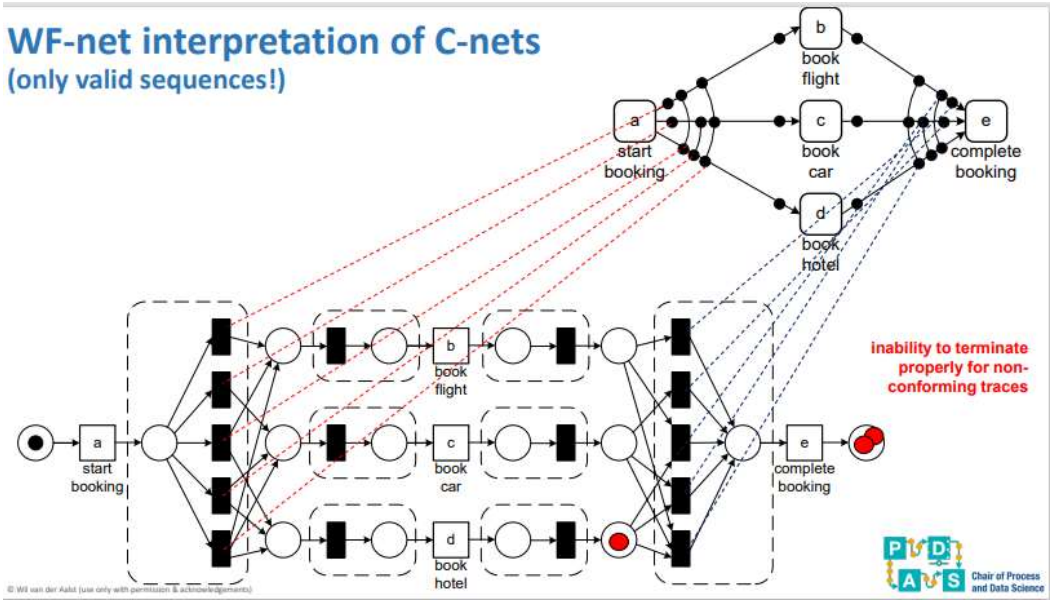
Example of all valid bindings for x

Rules of the game

Start with the start activity and end with the end activity. Those activities cannot happen in the middle.

When an activity occurs, it removes input obligations on its input edges and creates obligations on one output binding. An activity can only be activated if there are no obligations, or for each obligation, the connected bindings all have obligations.

At the end no obligations should be remaining.

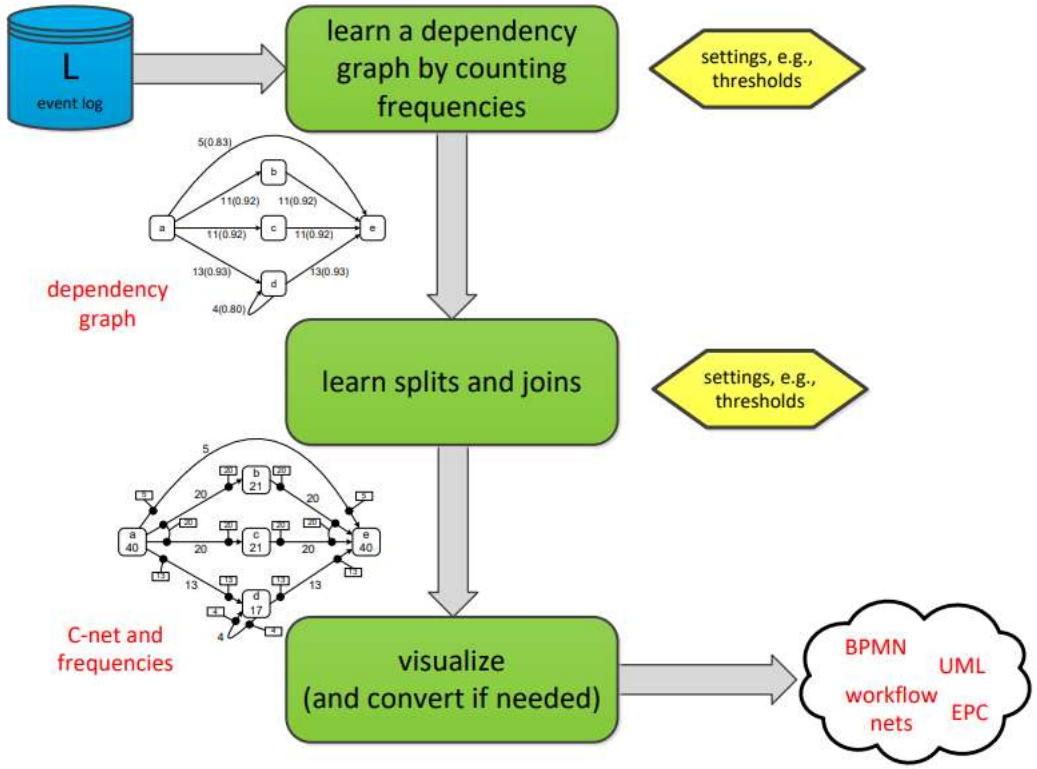


Example of a transformation into a WF-net with silent activities.

The semantics of C-nets are declarative and thereby more expressive than WF-nets. Valid binding sequences relate to valid firing sequences of the corresponding WF-net and vice-versa.

Heuristic Mining

The heuristic miner generates a C-net from the dependency graph. The resulting model can be visualized and transformed into other models if needed.



Creating the dependency graph

1. Start by counting the direct succession, which is the number of times that one activity is followed by another in the log:

$$|a >_L b| = \sum_{\sigma \in L} L(\sigma) \times |\{1 \leq i < |\sigma| \mid \sigma(i) = a \wedge \sigma(i+1) = b\}|$$

2. Then determine the dependency measure between activities:

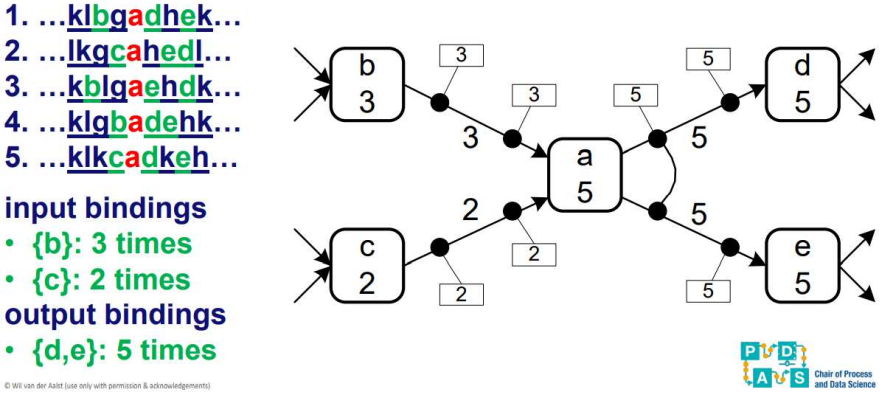
$$|a \Rightarrow_L b| = \begin{cases} \dfrac{|a >_L b| - |b >_L a|}{|a >_L b| + |b >_L a| + 1} & \text{if } a \neq b \\ \dfrac{|a >_L a|}{|a >_L a| + 1} & \text{if } a = b \end{cases}$$

3. If both the direct succession and the dependency measure are below a predefined threshold then there is no causality between the activities.
4. Draw dependency graph while including only arcs that meet **both** thresholds.

Getting the C-net

There are two classes of approaches to discover splits and joins:

1. Heuristics using a time window before and after each activity. We scan the trace in window before an activity a resp. after the activity a and try to guess what the input resp. output bindings can be based on the observed activities. By counting the sets of input and output activities the bindings can be determined. Note that we are only interested in activities which are directly connected to a in the dependency graph.



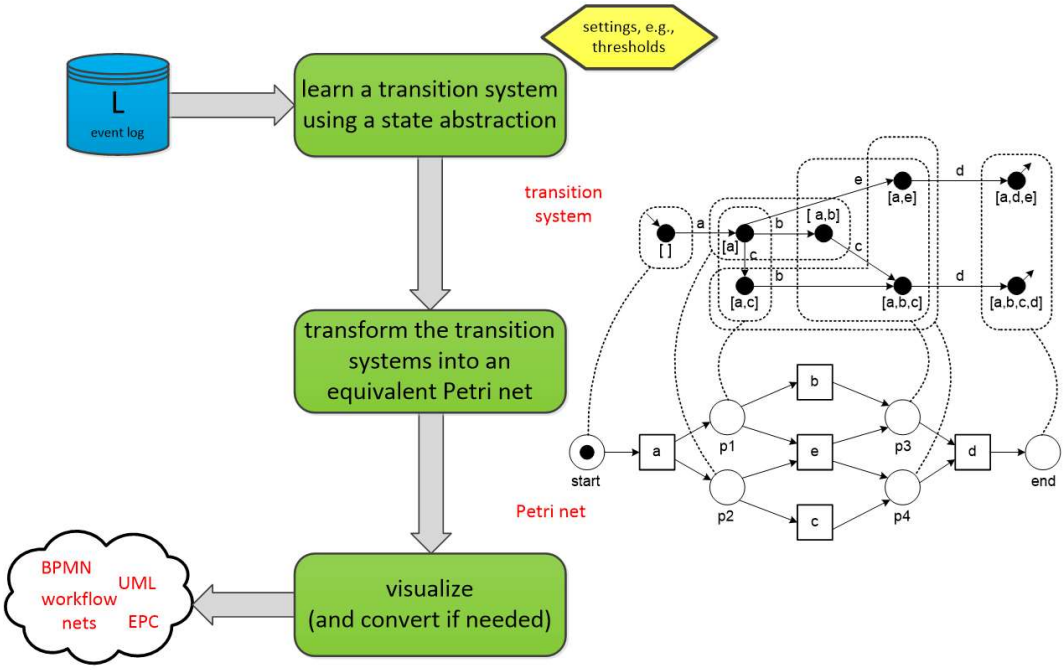
Refinements:

- Noise filtering (remove infrequent bindings).
- Handling repeating activities (e.g., cut off window size).

2. Optimization approaches based on replay. Given a set of possible input and output bindings one can see whether reality can be replayed properly. The set of possible input and output bindings is finite, so a "best set of bindings" can be determined using some goal function (e.g. Precision, generalization,...).

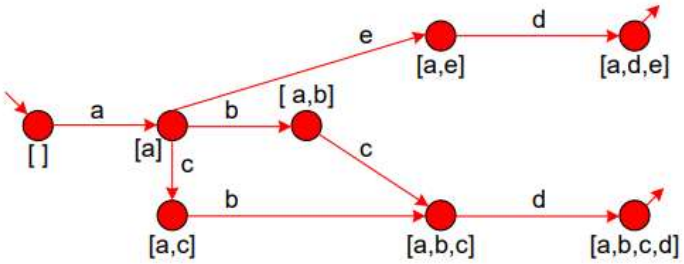
Region-based mining

Overview

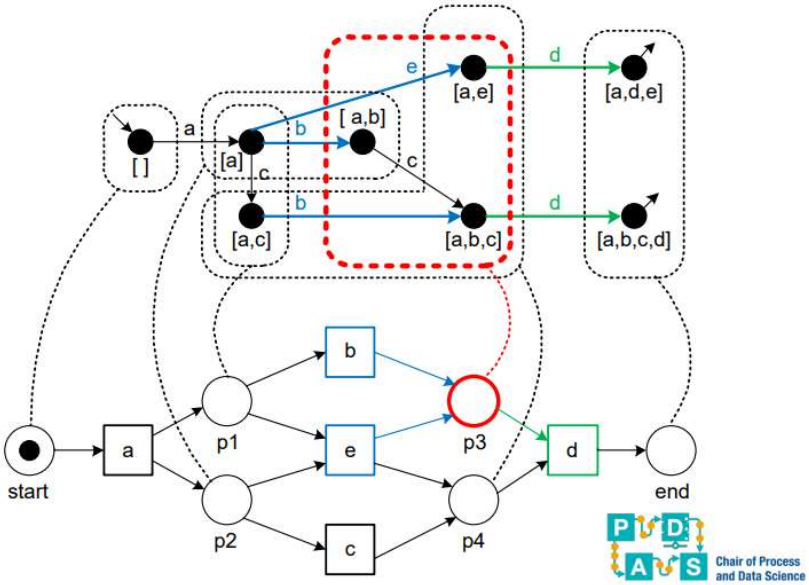


2 step approach:

1. Create a transition system from the event log



2. Convert into a petri net while detecting concurrency



As always, we might convert the result into the desired visualization format

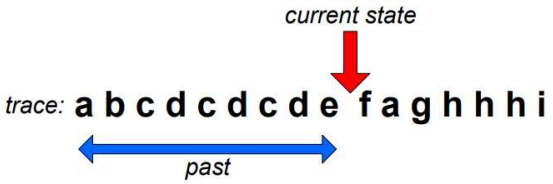
Drawbacks of this technique:

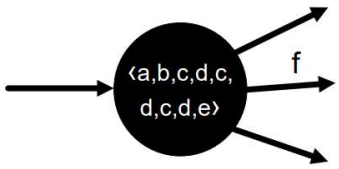
1. Computationally expensive
2. Tends to overfit the model

Creating the transition system

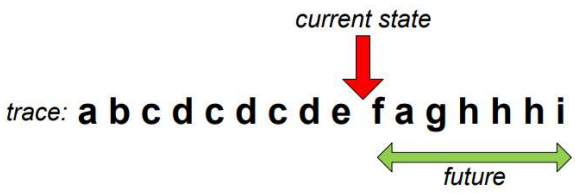
The transition system is a graph where the nodes are the reachable activities up to that point and the edges are the activities we perform to reach them. So the position in the trace determines the current state . There are multiple variations:

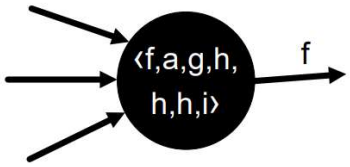
- Variation of time direction:
  - Past, also known as prefix automaton



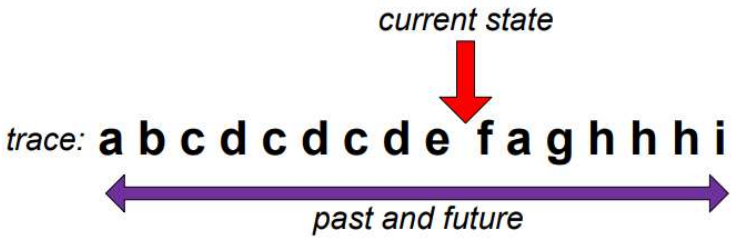


  - Future

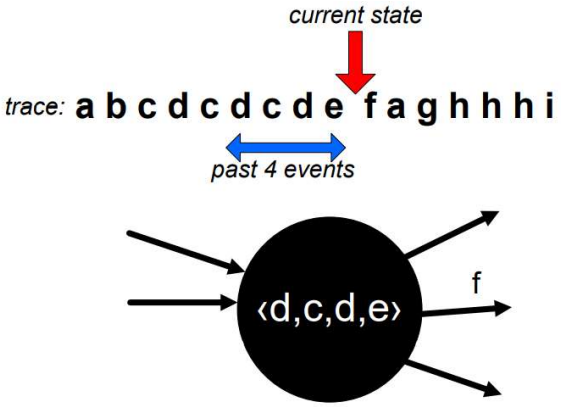




  - Past and future

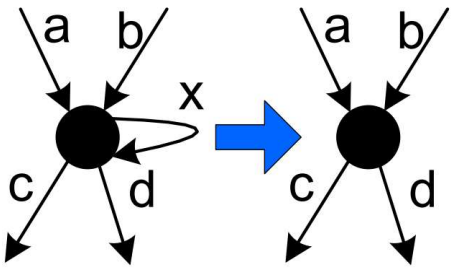


- Nodes as **sequences**: the nodes are sequences where the order and frequency is important ($< a, b > \neq < b, a >$)
- Nodes as **multisets**: only frequency matters (e.g. $[a, b^2, c]$)
- Nodes as **sets**: frequency and order don't matter ($\{a, b, c\}$)
- **Limited horizon**: only look k-steps into a particular direction:

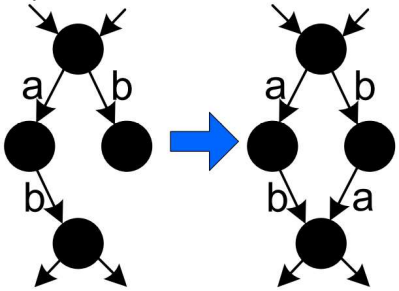


Postprocessing of transition system

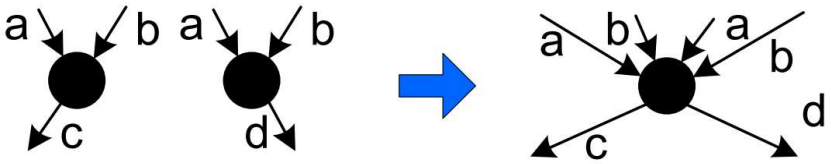
- Remove self loops



- Improve diamond structure



- Merge similar states

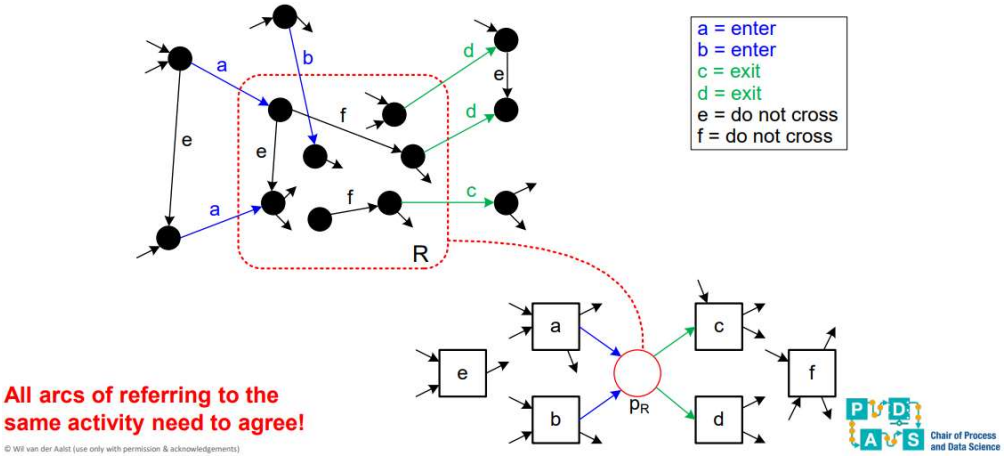


Using Regions to Discover Concurrency

Basic idea: Try to find a set of states that agree on activities, such sets are called regions.
We assume that we only have one start and one end activity. (If not we do necessary preprocessing. End activity not necessarily required) Furthermore, all states must be reachable.

Region definition

A region is a set of states, such that, if a transition exits the region, then all equally labeled transitions exit the region, and if a transition enters the region, then all equally labeled transitions enter the region. All events not entering or exiting the region do not cross the region.



In this example states a and b enter the activity set and c and c are exiting a region. Note that a,b,c,d must not appear inside the region.

Region properties

If r is a region, then S\r is a region ("exits" and "enters" are switched)

If r_1 and r_2 are regions, and r_1 is a subset of r_2, then $r_2 \setminus r_1$ is a region.

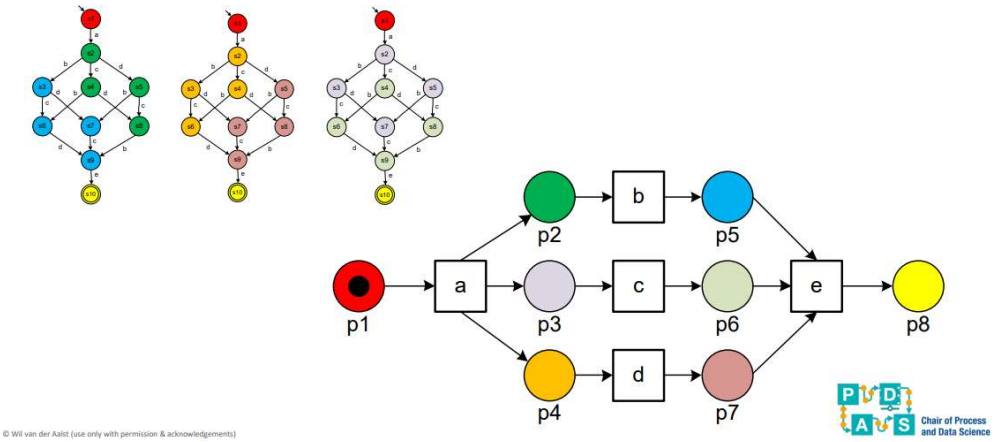
Algorithm

Only non-trivial minimal regions are selected.

Non-trivial: All sets which are not the empty set and the set of all states is non-trivial

Minimal: A region is minimal if it cannot be decomposed into smaller (non-trivial) ones

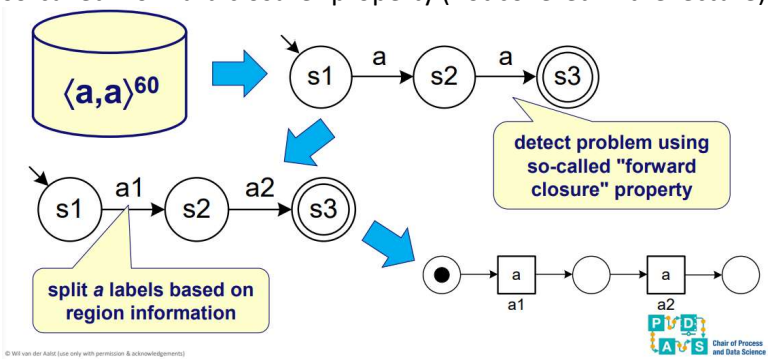
For each minimal, non-trivial region a place is added in the petri net and the corresponding arcs are generated. A token is added to each place that corresponds to a region containing the initial state. The resulting Petri net is called the minimal saturated net.



Weaknesses of approach based on state-based regions

- Inability to discover particular process constructs (can be handled through extensions of the

basic algorithm).
There is a refinement which can be used to address these problems by detecting it using the so-called "forward closure" property (not covered in the lecture) and then use label splitting



- Inability to balance the four forces (fitness, precision, generalization, and simplicity) well.

Language-based regions

Idea: encode the log and try to solve a linear equation system.
Two matrices: A,A' A encode the sequences and all non-empty prefixes of the sequences, while A' encodes the sequences and all non-empty prefixes of the sequences except the last transition. We try to solve

$$c \cdot \mathbf{1} + A' \cdot \mathbf{x} - A \cdot \mathbf{y} \geq \mathbf{0}$$

any solution (x, y, c) is a region and any region (x, y, c) is a feasible place
Example for log $L = \{\langle a, c, d\rangle, \langle b, c, e\rangle\}$:

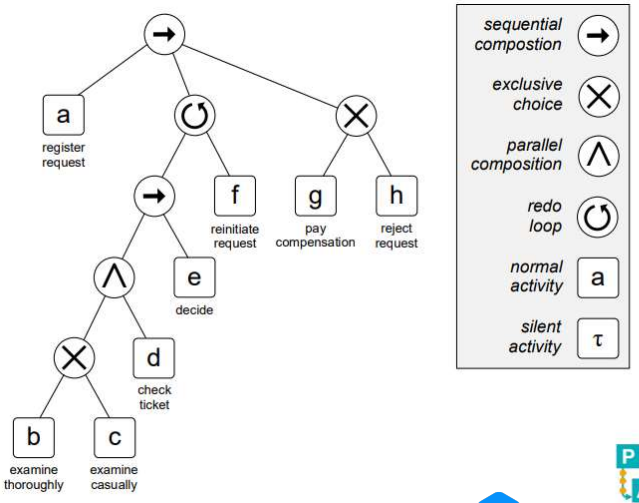
$$A = \begin{array}{c} \\ \langle a\rangle \\ \langle b\rangle \\ \langle a,c\rangle \\ \langle b,c\rangle \\ \langle a,c,d\rangle \\ \langle b,c,e\rangle \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix} \end{array} \qquad A' = \begin{array}{c} \\ \langle a\rangle \\ \langle b\rangle \\ \langle a,c\rangle \\ \langle b,c\rangle \\ \langle a,c,d\rangle \\ \langle b,c,e\rangle \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$$c \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_a \\ x_b \\ x_c \\ x_d \\ x_e \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} y_a \\ y_b \\ y_c \\ y_d \\ y_e \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

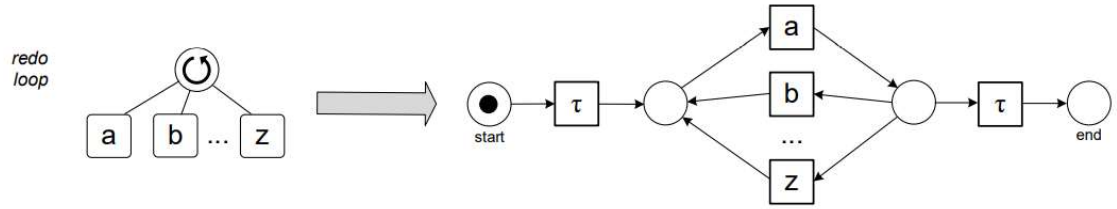
Inductive mining

Process tree

A process tree is a hierarchical representation of a process model

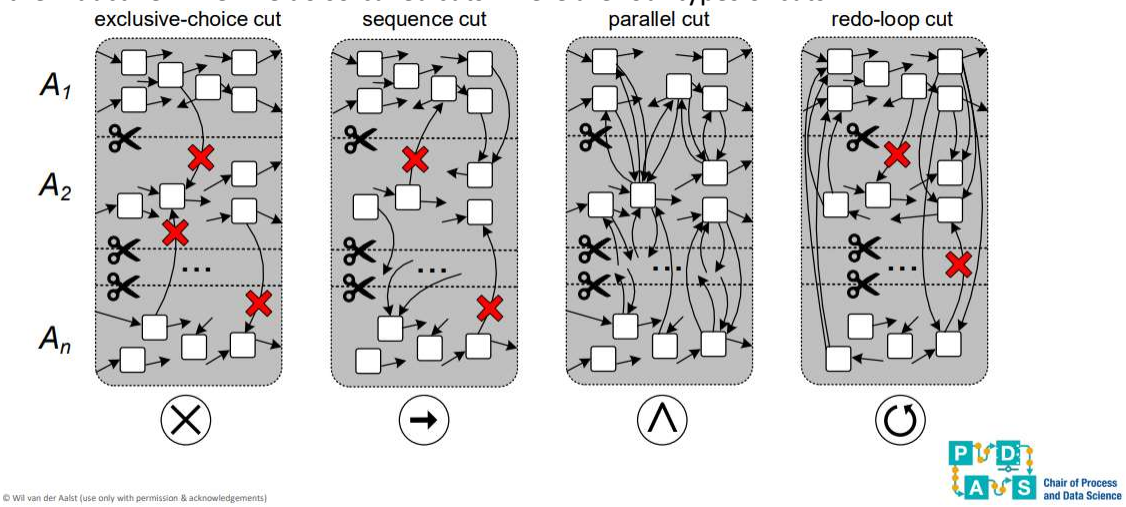


The redo loop has at least two children. The left hand child defines the forward movement. We always do that activity first. Then we decide if we stop or continue. If we continue we execute one of the right children followed again by the left child.



Cutting the DFG

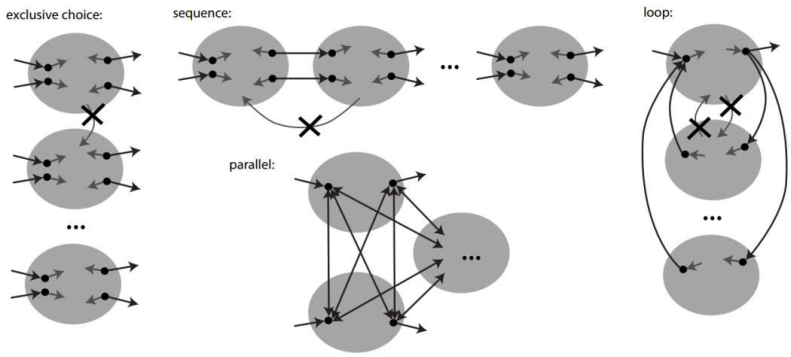
For the inductive miner we do so-called cuts. There are four types of cuts



We start with the complete log and the DFG and then cut the log into smaller subsets until we have only single activities. The types of cuts that we do create the process tree, where each inner node is a cut and the leaves are the activities.

1. **Exclusive choice**: For two subsets there should not be a directly follows relationship between the two subsets.
$$\forall_{i,j\in\{1,\dots,n\}}\forall_{a\in A_i}\forall_{b\in A_j} i \neq j \Rightarrow a \nrightarrow b$$
2. **Sequence cut**: For two subsets there should be a directly follows relationship that is not there in the opposite direction.
$$\forall_{i,j\in\{1,\dots,n\}}\forall_{a\in A_i}\forall_{b\in A_j} i < j \Rightarrow (a \rightarrow b \wedge b \nrightarrow a)$$
3. **Parallel cut:** For two subsets, between any two elements from different set there should be a directly follows relationship in both directions. Furthermore both sets need to be connected to the start and end.
$$\forall_{i,j\in\{1,\dots,n\}}\forall_{i,j\in\{1,\dots,n\}}\forall_{a\in A_i}\forall_{b\in A_j} i \neq j \Rightarrow a \rightarrow b$$
$$\forall_{i,j\in\{1,\dots,n\}} A_i \cap A_L^{start} \neq \emptyset \wedge A_i \cap A_L^{end} \neq \emptyset$$
4. **Redo-loop:** Basic idea: Loop consists of two parts the "do" and the "redo" part. The "do"-part (first child) is connected to the start and end.
   - If there is an activity in the do-part having a connection to an activity in the "redo" part then it should also be possible to go to an end activity from that activity. Furthermore, all activities in the "do" part connected to an end activity should be connected to the activity in the "redo" part.
   - If there is an activity in the "do" part having a connection to an activity in the "redo" part then it should also be possible to reach the activity in the "do" part from the start. Furthermore if we jump back from the "redo" part to an activity in the "do" part, then we should be able to jump back to any start activity in the "do" part.

The cuts need to be checked in the order 1-4.



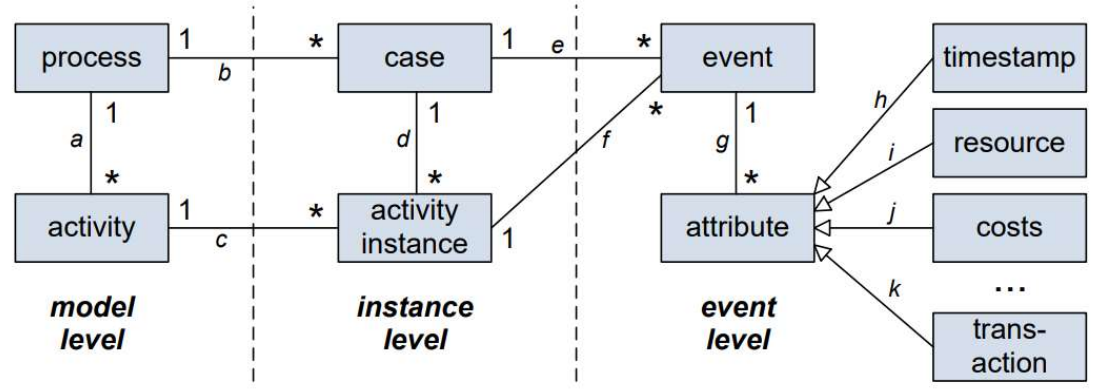
Properties of the inductive miner

- The basic algorithm formally **guarantees** that the original event log can be **replay**ed.
- **Highly scalable**, dealing with billions of events, millions of cases, and thousands of unique activities.
- Models are always **sound**. If the event log was generated from a basic process tree (no duplication of labels), then an equivalent model will be found. If not, typically a more general model is found (able to replay the log without any problems).

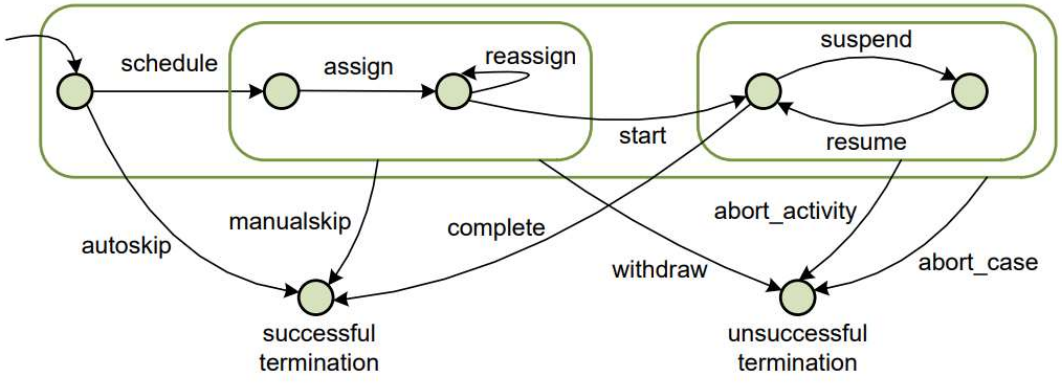
There are extensions that deal with infrequent behavior and incomplete event logs.

Event data and Exploration

Processes are composed of cases, which are in turn composed of events.
Activities are happening in a process and for a particular case, we usually have multiple activity instances. Note that activity instances always refer to a single activity, but can be composed of several events.
An event is always atomic and usually contains multiple attributes like timestamp, resource, costs, transaction,...



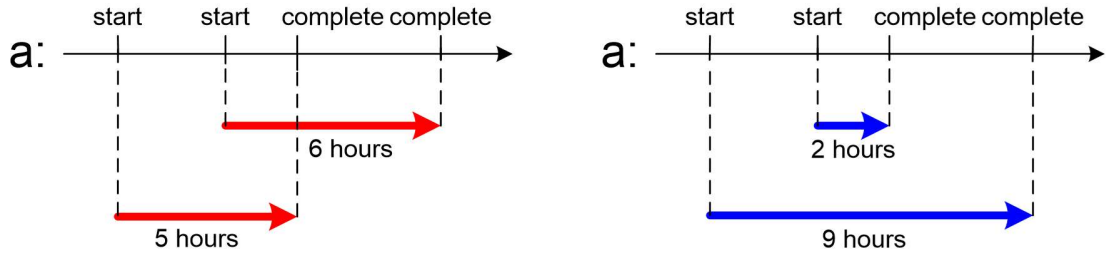
To model activity instances which take time, we need to model them with a start and complete event. Other events are also possible:



Note that sometimes, we only receive information of the complete event.

Possible confusion of activity instances

If activity instances are overlapping in time, there are some challenges.

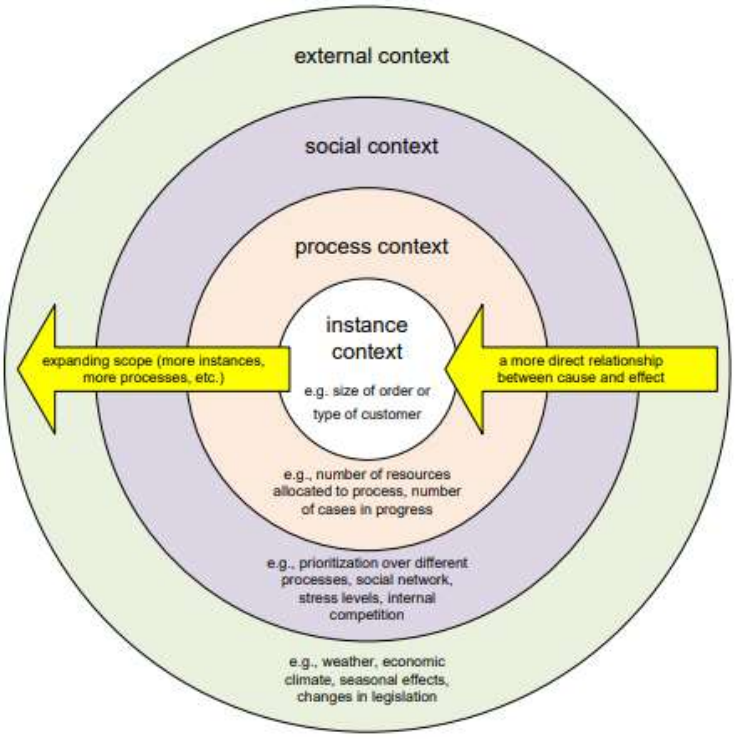


In this example two interpretations are possible. Which one is the correct one?
This problem can be addressed in two ways:

1. Events explicitly refer to activity instances. E.g. XES stores the start and complete events as tuples.
2. Activity instances are created and computed during replay (using heuristics)

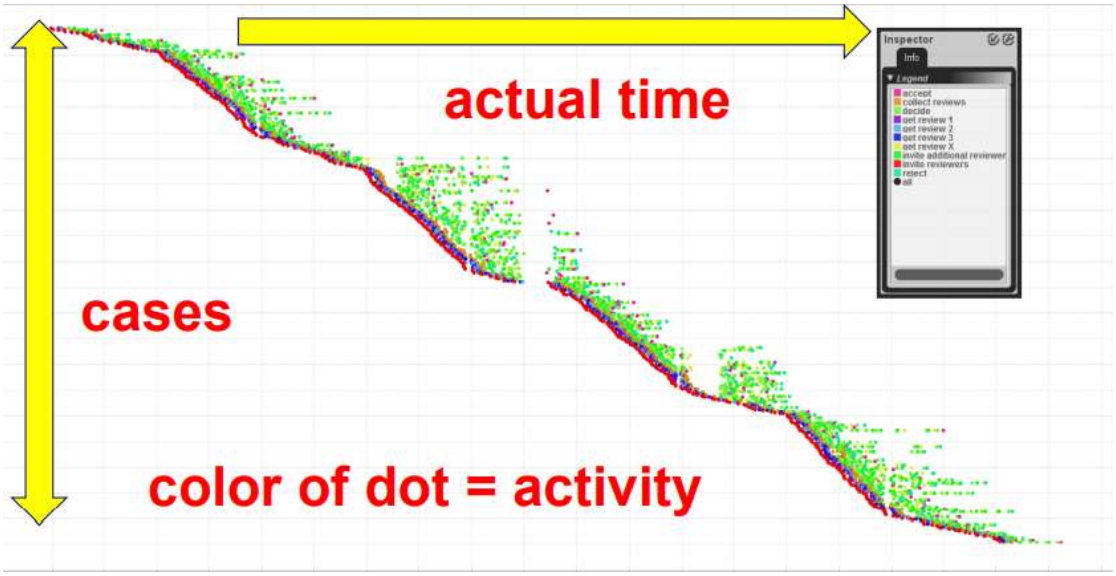
Contextual information

We might be interested in other attributes influencing the behavior of a certain resource.



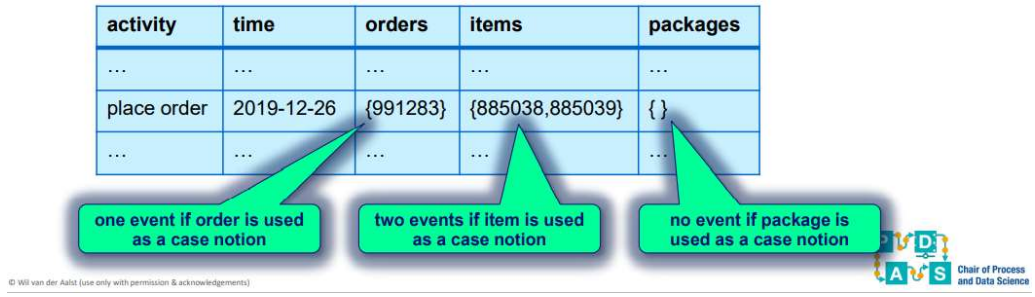
Dotted chart

To get an overview of all events we often use the dotted chart representation. In this chart every dot is corresponding to an event. Each event has a timestamp that is used to compute the X coordinate of the dot. Another attribute is used for the Y coordinate e.g. The case. The dots can be colored based on a third activity e.g. Resource or activity name.



Transformation into event log

In commercial systems, the data which is logged is not in the format of the event log. Oftentimes there are multiple objects in the data which could serve as a case identifier (e.g. Customer , product, item, order)

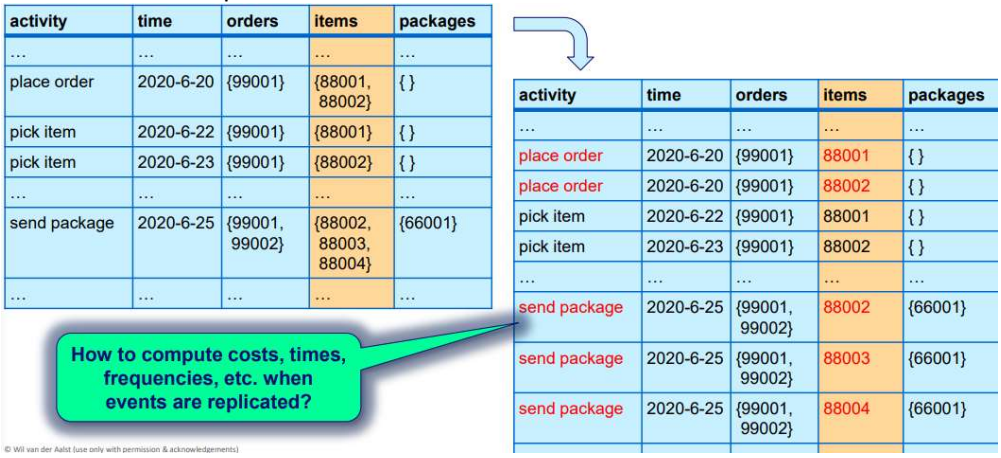


Those tables are often in a many-to-many relationship. In order to transform this into an event log, we need to flatten the data. We pick one object type as the case notion, then replicate each event for each object of the corresponding type.

Possible problems

Deficiency: If we pick one object as the case identifier and a particular event does not have any corresponding attribute for that object, then they will disappear in the flattened event log. (e.g. event where packages = {} if we take packages as case id)

Convergence: Events referring to multiple objects of the selected type are replicated, possibly leading to unintentional duplication of activities.



Divergence: Events referring to different objects of a type not selected as the case notion are considered to be causally related.

activity	time	orders	items	packages
...	...	...	...	...
place order	2020-6-20	99001	{88001, 88002, 88003}	{ }
pick item	2020-6-22	99001	{88001}	{ }
pick item	2020-6-23	99001	{88002}	{ }
pack item	2020-6-22	99001	{88002}	{ }
pack item	2020-6-23	99001	{88001}	{ }
pick item	2020-6-22	99001	{88003}	{ }
pack item	2020-6-23	99001	{88003}	{ }
...	...	...	...	...

Causal Dependency between events with the same items is lost

Data quality problems

There are four different types of quality problems:

1. **Missing** data ("things" are not recorded).
2. **Incorrect** data ("things" are recorded incorrectly).
3. **Imprecise** data ("things" are not recorded at the desired level of granularity).
4. **Irrelevant** data (relevant "things" are submerged in poorly structured data)

The 12 guidelines for logging

1. Reference and attribute names should have clear semantics, i.e., they should have the same meaning for all people involved in creating and analyzing event data. Different stakeholders should interpret event data in the same way.
2. There should be a structured and managed collection of reference and attribute names. Ideally, names are grouped hierarchically (like a taxonomy or ontology). A new reference or attribute name can only be added after there is consensus on its value and meaning.
3. References should be stable (e.g., identifiers should not be reused or rely on the context). For example, references should not be time, region, or language dependent. Some systems create different logs depending on the language settings. This is unnecessarily complicating analysis.
4. Attribute values should be as precise as possible. If the value does not have the desired precision, this should be indicated explicitly (e.g., through a qualifier). For example, if for some events only the date is known but not the exact timestamp, then this should be stated explicitly.
5. Uncertainty with respect to the occurrence of the event or its references or attributes should be captured through appropriate qualifiers. For example, due to communication errors, some values may be less reliable than usual. Note that uncertainty is different from imprecision.
6. Events should be at least partially ordered. The ordering of events may be stored explicitly (e.g., using a list) or implicitly through an attribute denoting the event's timestamp. If the recording of timestamps is unreliable or imprecise, there may still be ways to order events based on observed causalities (e.g., usage of data).
7. If possible, also store transactional information about the event (start, complete, abort, schedule, assign, suspend, resume, withdraw, etc.). Having start and complete events allows for the computation of activity durations.
8. Perform regularly automated consistency and correctness checks to ensure the syntactical correctness of the event log. Check for missing references or attributes, and reference/attribute names not agreed upon. Event quality assurance is a continuous process (to avoid degradation of log quality over time).
9. Ensure comparability of event logs over time and different groups of cases or process variants. The logging itself should not change over time (without being reported). For comparative process mining, it is vital that the same logging principles are used.
10. Do not aggregate events in the event log used as input for the analysis process. Aggregation should be done during analysis and not before (since it cannot be undone). Event data should be as "raw" as possible.

11. Do not remove events and ensure provenance. Reproducibility is key for process mining. For example, do not remove a student from the database after he dropped out since this may lead to misleading analysis results. Also: concerts are not deleted - they are canceled, employees are not deleted - they are fired, etc.
12. Ensure privacy without losing meaningful correlations. Sensitive or private data should be removed as early as possible (i.e., before analysis). However, if possible, one should avoid removing correlations. Hashing can be a powerful tool in the trade-off between privacy and analysis.

Conformance Checking

There are three main ways to do conformance checking :

1. Conformance checking using **causal footprints**.
2. Conformance checking based on **token-based replay**.
3. **Alignment**-based conformance checking.

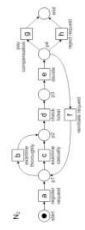
Causal footprints

We have seen that we can construct a petri net model from the footprint matrix of a log. We can also construct a footprint matrix based on all possible traces, generated from the process model. We can then compare the differences between the two footprint matrices.

L_{full}

	a	b	c	d	e	f	g	h
a	#	→	→	→	#	#	#	#
b	←	#	#	‖	→	←	#	#
c	←	#	#	‖	→	←	#	#
d	←	‖	‖	#	→	←	#	#
e	#	←	←	←	#	→	→	→
f	#	→	→	→	←	#	#	#
g	#	#	#	#	←	#	#	#
h	#	#	#	#	←	#	#	#

N_2



	a	b	c	d	e	f	g	h
a	#	→	→	#	#	#	#	#
b	←	#	#	→	#	←	#	#
c	←	#	#	→	#	←	#	#
d	#	←	←	#	→	#	#	#
e	#	#	#	←	#	→	→	→
f	#	→	→	#	←	#	#	#
g	#	#	#	#	←	#	#	#
h	#	#	#	#	←	#	#	#

Based on those differences, we can define measures.

Footprint based conformance

$1 - \frac{x}{n^2}$, where n is the number of activities and x the number of cells that are different for the two footprint matrices

Advantage

Flexible: we cannot only to model-to-log comparisons but also model-to-model, or log-to-log

Limitations

- Frequencies are not used
- The behavior is only considered indirectly
- Aims to capture fitness, precision and generalization in a single metric. But we would often like to focus on one of them first

Token-based replay

We count tokens while replaying the data. If an action is executed in the log while missing a token in the model then we add this as a "missing" token and continue the replay.
We start with one token in the source place. At the end the token is consumed in the end place (note that we might need to add a missing token)

Quantification of **fitness** at the **trace level**:

$$\text{fitness}(\sigma, N) = \frac{1}{2} \cdot \left(1 - \frac{m}{c}\right) + \frac{1}{2} \cdot \left(1 - \frac{r}{p}\right)$$

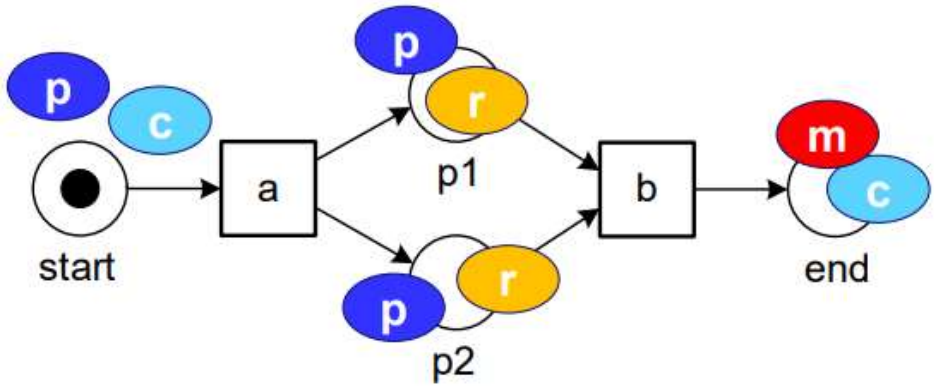
p is the number of **produced** tokens
c is the number of **consumed** tokens
m is the number of **missing** tokens
r is the number of **remaining** tokens

Quantification of **fitness** at the **log level**:

$$\text{fitness}(L, N) = \frac{1}{2}\left(1 - \frac{\sum_{\sigma \in L} L(\sigma) * m_{N,\sigma}}{\sum_{\sigma \in L} L(\sigma) * c_{N,\sigma}}\right) + \frac{1}{2}\left(1 - \frac{\sum_{\sigma \in L} L(\sigma) * r_{N,\sigma}}{\sum_{\sigma \in L} L(\sigma) * p_{N,\sigma}}\right)$$

$L(\sigma)$ is the frequency of the trace variant in the log

Often, it holds that $c = p$ and $m = r$, but this is not always the case.



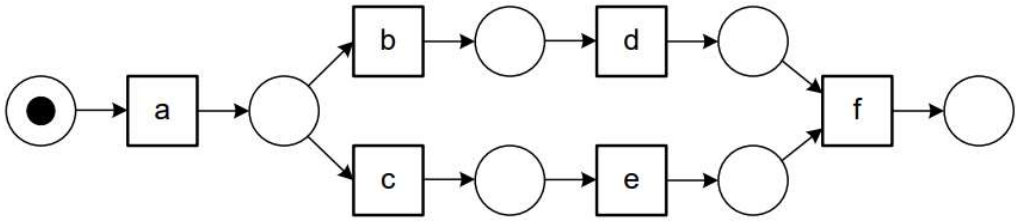
For this model and log [<a>] this does not hold ($p = 3, c = 2, m = 1, r = 2$)

Limitations

- This approach assumes visible and uniquely labeled transitions. (if there are activities with the same label and one is enabled and one not enabled then the behavior is indeterministic, but optimally we would want to fire the one which is enabled) This problem can be dealt with by using heuristics.
- Conformance values sometimes become too optimistic due to token flooding: If a token is missing we add it to the model, but adding more tokens also means that more (unwanted) behavior might be possible. This local decision making of when to add a token may lead to misleading results
- Replay technique does not provide a corresponding path through the model (vital for conformance/performance analysis and other diagnostics). What we would prefer to find the "closest path". This can be done using alignments

Measured fitness is not always a good indicator for performance, as even models where no sequence from start to end can be generated are able to have a high fitness.

$$L = [\langle a, b, d, e, f \rangle^{10}, \langle a, c, e, d, f \rangle^{10}]$$

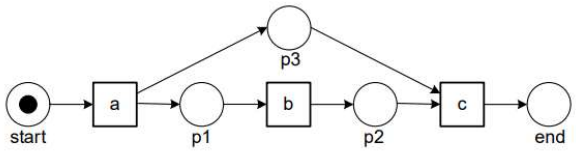


Example of a model generated by the alpha algorithm. The fitness of this model is 0.93 for the given log, even though no sequences can be generated.

To address this issue, we also require **relaxed soundness** in many cases. Relaxed soundness means that at least one sequence can be generated by the model.

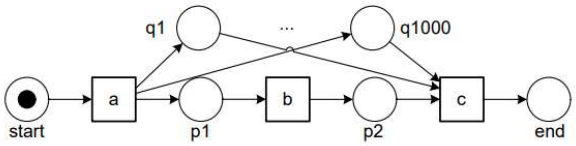
Redundant places impact fitness: A fitness close to 1 can be achieved by inserting a lot of redundant places that don't influence the behavior

$$L_{11} = [\langle a,b,c\rangle^{20}, \langle a,c\rangle^{30}]$$



$$\frac{1}{2}\left(1-\frac{30}{220}\right)+\frac{1}{2}\left(1-\frac{30}{220}\right)=\frac{19}{22}\approx 0.86$$

$$L_{11} = [\langle a,b,c\rangle^{20}, \langle a,c\rangle^{30}]$$



$$\frac{1}{2}\left(1-\frac{30}{50170}\right)+\frac{1}{2}\left(1-\frac{30}{50170}\right)=\frac{5014}{5017}\approx 0.999$$

Chair of Process and Data Science

Alignment based conformance checking

Conformance checking should not impose restrictions on the process notation (the techniques seen so far don't suppoert silent transitions and two transitions with the same label).
Furthermore, two semantically equivalent models should have the same conformance value.

We should provide a "closest matching path" through the process model for any trace in the event log. This is also required for performance analysis.

Terminology

An alignment is a pair of traces. One trace is a trace from the log and the other trace is a trace generated by the model. The traces are modified such that the activities align with each other. "No move" transitions (>>) are inserted into the places that don't align

a	»	d	e	g	h
a	b	d	e	g	»

Example of an alignment.

- Columns where both activities are matching are called **synchronous moves**. Synchronized moves must agree on the label
- Columns where we have a transition in the trace from the log, but no move in the trace from the model, are called **move in log only**.
- Accordingly, columns where we have "no move" in the trace from the log, but a transition in the trace from the model , are called **move in model only**.

The goal is to find alignments maximize the number of synchronous moves that have the least move in log / model only.

Computing fitness

The standard cost function is the number of >>'s in the alignment. Note that optimal alignments do not have to be unique. Other cost functions are also possible.

The fitness should be between 0 and 1, with 1 being perfect fitness.

Fitness at the alignment level

$$\text{fitness}(a) = 1 - \frac{c(a)}{c(\text{worst\_case\_alignment})}$$

For a cost function c

The worst case alignment is the alignment where we first have log only moves and then model only moves. Thus, the cost of the worst case alignment is the length of the trace plus the length of the shortest path through the model.

Advantages of aligning log and model

- The **observed behavior** is directly **related to modeled behavior**.
- It is very **flexible** (any cost structure is possible). Other attributes like time can also be considered
- Detailed **diagnostics** are available.
- After aligning log and model, other quality dimensions can be investigated (**separation of concerns**).

Next steps

An example approach might be that after doing performance analysis, we pick out deviating cases and create a new event log from them. From this event log, we can then repeat process discovery and then compare the new model with the model from the original log.

Conformance checking also considers precision, generalization and simplicity. Furthermore, conformance checking might also include other perspectives such as resource, time, cost. These are investigated in data-aware alignments (not covered in the lecture)

Decision mining

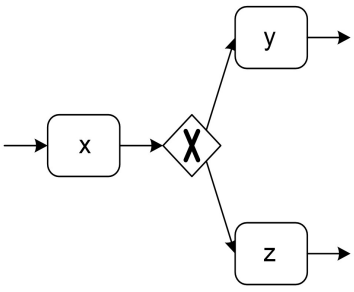
Mining decision points

For a given aligned model and log, we would like to discover why certain choices are made in the petri net

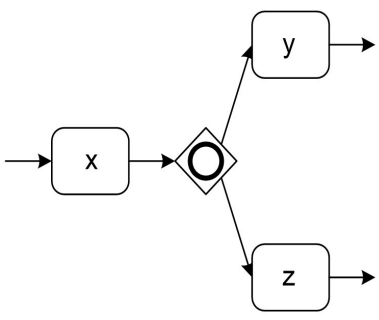
A decision point in the petri net is a place with multiple outgoing arcs. In BPM diagrams they are the **XOR-split and OR-split** gateways. Note that the AND-split gateway is **not** a decision point.

We can annotate the activities with guards. For a decision point, we can use anything related to the context up to the decision point as **predictor** variables and the type of activity which is executed as **response** variable. Then we can use a classifier to create the guards.

For an XOR-split the response variable will be either activity. For the OR-split the response variable will be the AND combinations of activities. Examples:



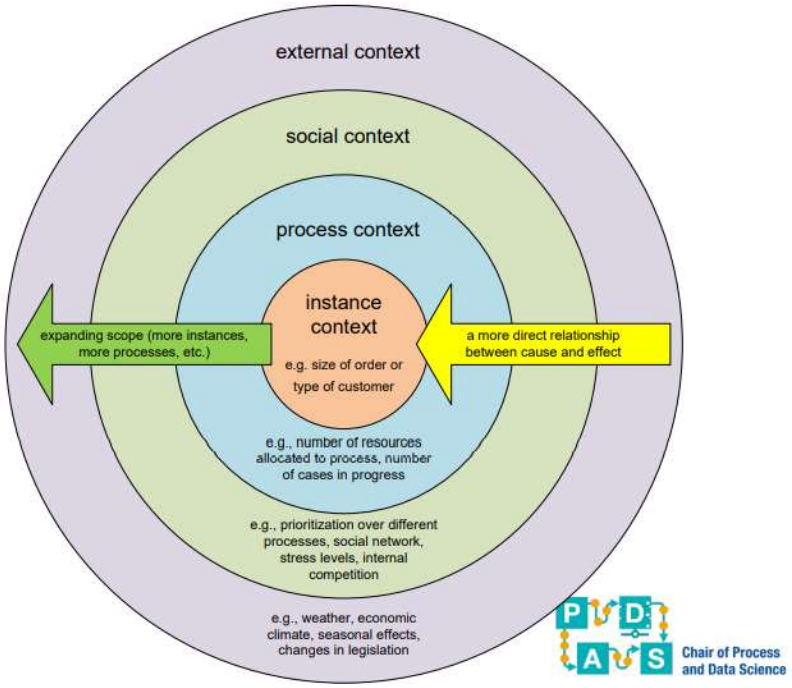
Response variables: y, z



Response variables: y, z, y AND z

Predictor variables can be taken from the last activity, the whole past or the whole trace. The predictor variables might also be based on the context of the process instances, e.g.

- Number of cases running (e.g. skip check if busy).
- Number of resources present.
- Workload of resource.
- Day of the week.
- Etc

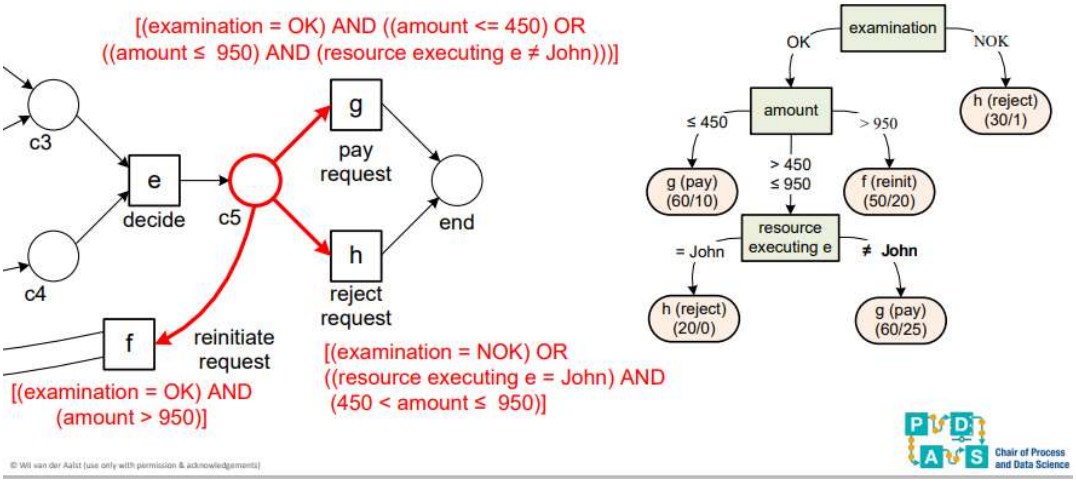


Other features can also be used as response variables:

- Remaining flow time
- Costs
- Risks
- Etc.

If we have a decision tree than the paths to the node define the guards. Paths are combined using OR.

Example



There are two leaves with the g label in the decision tree.

- The path to the top left g is $(examination = OK)$ and then $(amount \leq 450)$ so the path is an AND combination of both.
- The path to the lower g leaf is $(examination = OK)$, then $(450 < amount \leq 950)$ and then $(resource\ executing\ e \neq John)$. The path is an AND combination of the three.

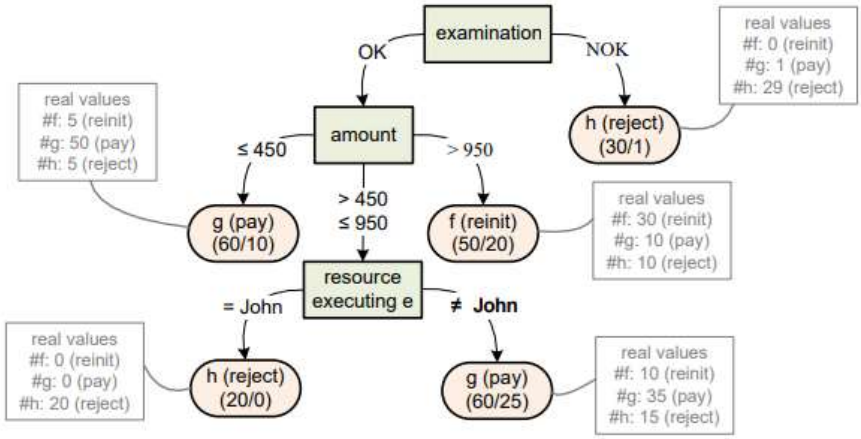
If we combine both paths by OR and simplify we get the formula $[(examination =$

Dealing with uncertainty

Any classifier will most likely classify some instances wrong. If the fraction of wrong instances is too high we can randomize the choice

Data-dependent probabilities

We can also make choices deterministic

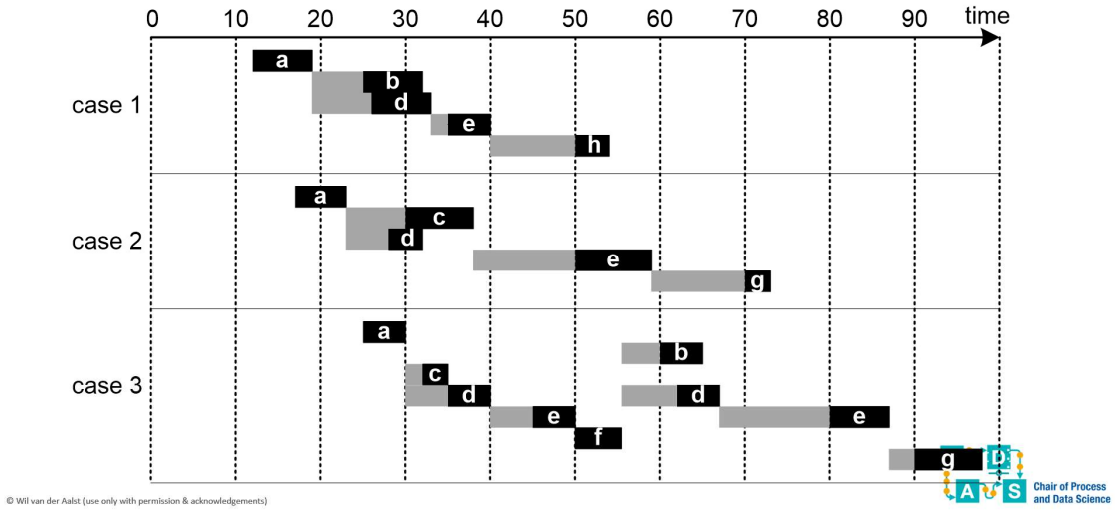


We label the decision points with the probabilities from the classifier. Note that the discovered guards are descriptive rather than prescriptive

Organizational mining

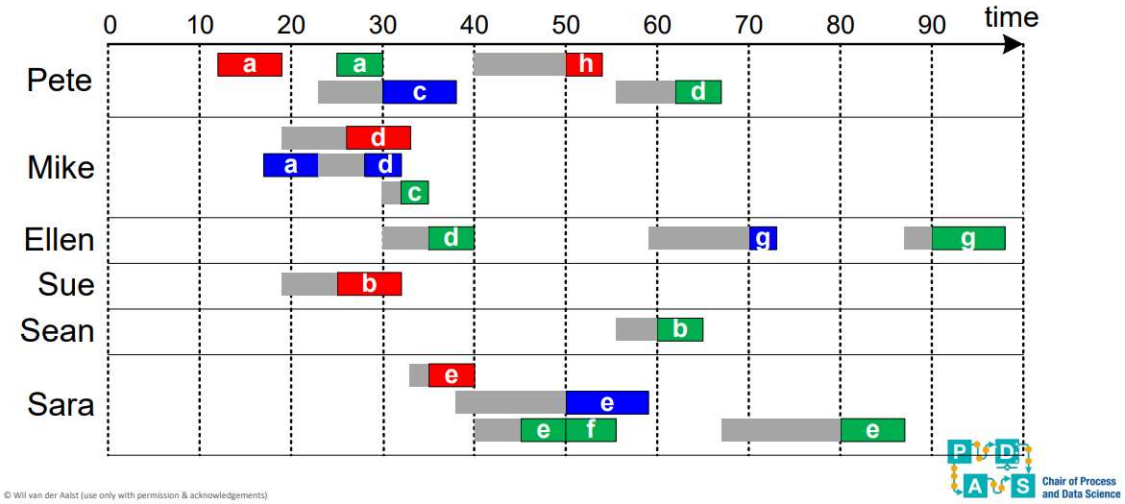
Performance analysis

As we have seen before, activities consist of activity instances (most of the time start and complete). If we take the difference between the start of one activity of one instance and the end activity of another instance, then we get the **in between time** for the two activities resp. The duration at which the second transition in the model was enabled, but has not fired yet. The duration in which we stay in a particular state is also called the **sojourn time**.



Example of a Gantt chart. The black boxes are the activity durations. The black boxes are the times from which on a transition was enabled.

We can also project the instances to resources



The colors represent different cases.

From a process model we can compute average service times, average waiting times and also compute branching probabilities.

Mining Social Networks

Sometimes, we are interested in learning about resources. We can project the log onto traces where every activity in the trace contains the label of the resource executing the activity

case id	trace
1	$\langle a^{Pete}, b^{Sue}, d^{Mike}, e^{Sara}, h^{Pete} \rangle$
2	$\langle a^{Mike}, d^{Mike}, c^{Pete}, e^{Sara}, g^{Ellen} \rangle$
3	$\langle a^{Pete}, c^{Mike}, d^{Ellen}, e^{Sara}, f^{Sara}, b^{Sean}, d^{Pete}, e^{Sara}, g^{Ellen} \rangle$
4	$\langle a^{Pete}, d^{Mike}, b^{Sean}, e^{Sara}, h^{Ellen} \rangle$
5	$\langle a^{Ellen}, c^{Mike}, d^{Pete}, e^{Sara}, f^{Sara}, d^{Ellen}, c^{Mike}, e^{Sara}, f^{Sara}, b^{Sue}, d^{Pete}, e^{Sara}, h^{Mike} \rangle$
6	$\langle a^{Mike}, c^{Ellen}, d^{Mike}, e^{Sara}, g^{Mike} \rangle$
...	...

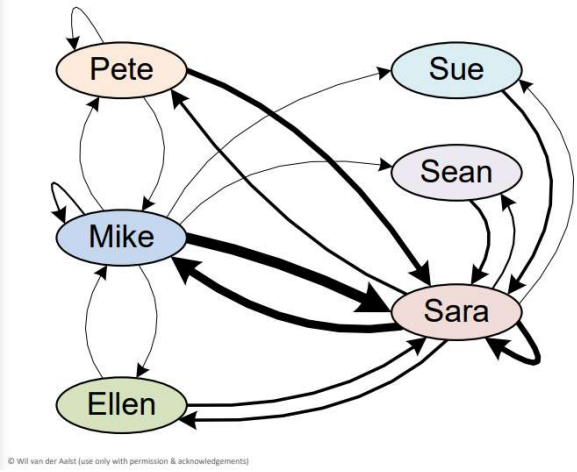
Resource-activity matrix

From the projected log we can also create resource activity matrices, where each entry represents the average number of times a resource has performed an activity per case.

	a	b	c	d	e	f	g	h
Pete	0.3	0	0.345	0.69	0	0	0.135	0.165
Mike	0.5	0	0.575	1.15	0	0	0.225	0.275
Ellen	0.2	0	0.23	0.46	0	0	0.09	0.11
Sue	0	0.46	0	0	0	0	0	0
Sean	0	0.69	0	0	0	0	0	0
Sara	0	0	0	0	2.3	1.3	0	0

Handover of work matrix

Two persons are related when one person hands over the case to another person. This is recorded in the handover of work matrix. In this matrix the entry $w_{i,j}$ corresponds to the number of times that a person i is handing over work to a person j on average per case. From this matrix we can construct a network



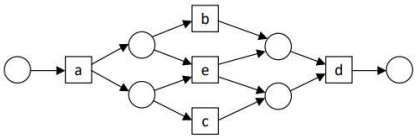
In this figure only the thickness of the arcs is based on frequencies. All nodes have the same size.

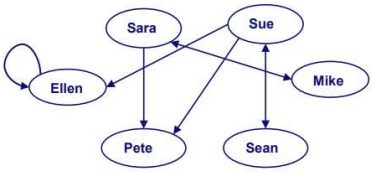
	Pete	Mike	Ellen	Sue	Sean	Sara
Pete	0.135	0.225	0.09	0.06	0.09	1.035
Mike	0.225	0.375	0.15	0.1	0.15	1.725
Ellen	0.09	0.15	0.06	0.04	0.06	0.69
Sue	0	0	0	0	0	0.46
Sean	0	0	0	0	0	0.69
Sara	0.885	1.475	0.59	0.26	0.39	1.3

Refinements

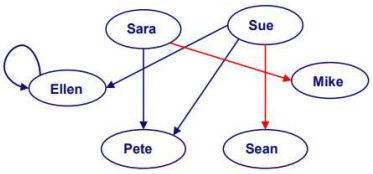
- Taking into account causality (i.e., process model) or not, it may be the case that certain arcs in the handover model are not allowed by the process model. In this case, we might consider removing those arcs.

$$L=[\langle a^{Sara},b^{Mike},c^{Sara},d^{Pete}\rangle,\ \langle a^{Sara},c^{Mike},b^{Sara},d^{Pete}\rangle,\ \langle a^{Sue},b^{Sean},c^{Sue},d^{Pete}\rangle,\ \langle a^{Sue},c^{Sean},b^{Sue},d^{Pete}\rangle,\ \langle a^{Sue},e^{Ellen},d^{Ellen}\rangle]$$





Simple handover of work



Real handover of work

Chair of Process and Data Science

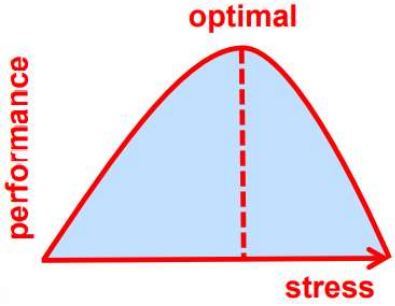
Note that we can also use the causal dependency defined in the earlier lectures

- Taking into account the number of transfer within a case or not.
- Taking into account distance.
- Using additional org. information.

Organizational models

What are the groups/roles that can be identified from a given log?
If we have the handover of work matrix, then we define Persons as the row vectors. We can apply clustering (with different distance measures e.g. Euclidean distance) to find groups. We can then extend the process model with those groups.

Yerkes-Dodson law of arousal

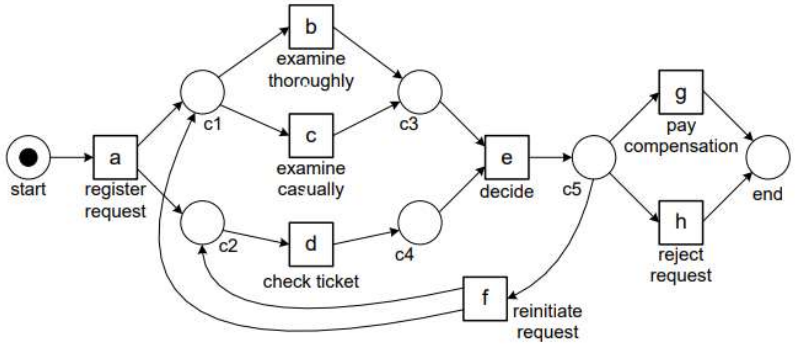


The **Yerkes–Dodson law** is an empirical relationship between pressure and performance, originally developed by psychologists Robert M. Yerkes and John Dillingham Dodson in 1908.[1] The law dictates that performance increases with physiological or mental arousal, but only up to a point. When levels of arousal become too high, performance decreases.

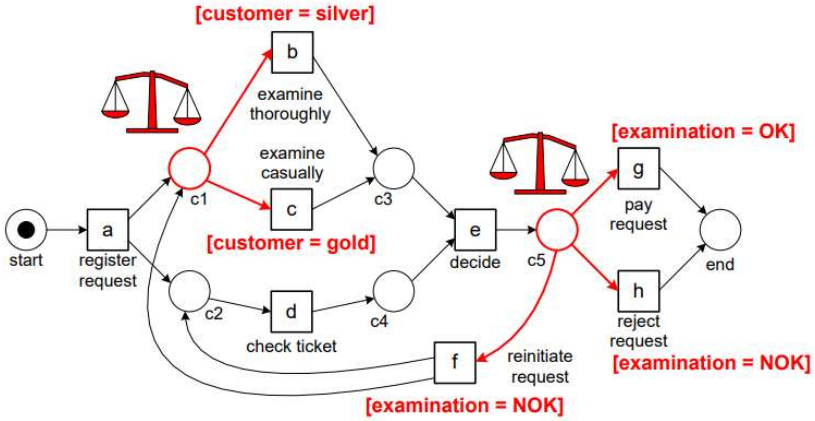
From <https://en.wikipedia.org/wiki/Yerkes%E2%80%93Dodson_law>

Combining Different Perspectives

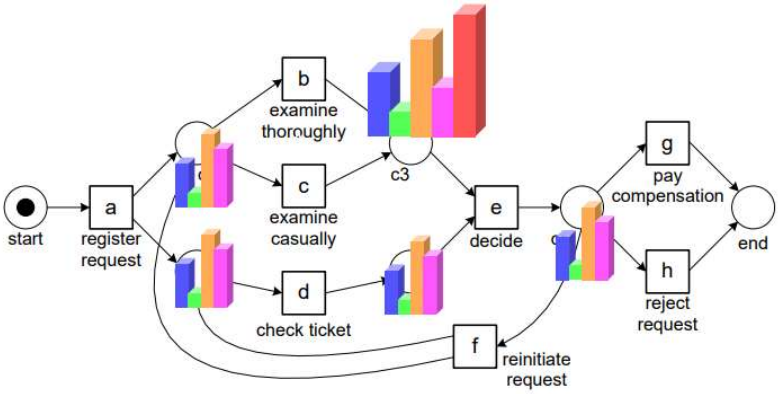
1. Starting point: **Control-flow model** discovered or made by hand, should be aligned with event log



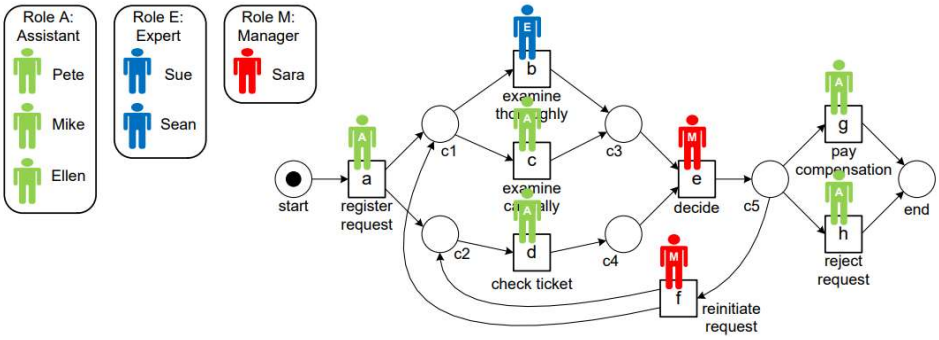
2. Adding the **data perspective**: decision tree learning can be used to create guards



3. Adding the **time perspective**: replay the event log to compute waiting times and service times. Also capture routing probabilities



4. Adding the **resource perspective**:



Refined Process Mining Framework

Business process provenance

In many cases we want the data to represent the way that things have really happened. The data should be **faithful**, **safe** and we should be able to replay the data in the future

Pre mortem and post mortem event data

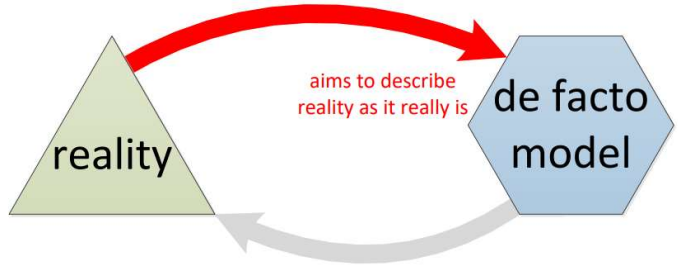
"**Post mortem**" event data refer to information about cases that have completed, i.e., these data can be used for process improvement and auditing, but not for influencing the cases they refer to.

"**Pre mortem**" event data refer to cases that have not yet completed.

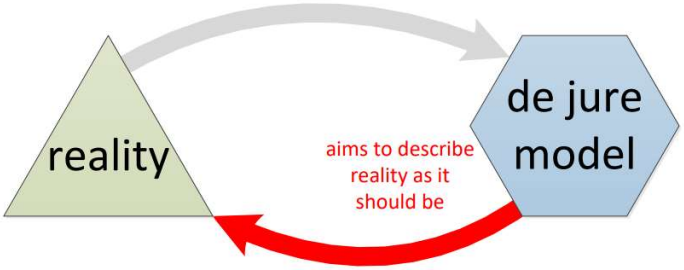
If a case is still running, i.e., the case is still "alive" (pre mortem), then it may be possible that information in the event log about this case (i.e., current data) can be exploited to ensure the correct or efficient handling of this case.

Types of models

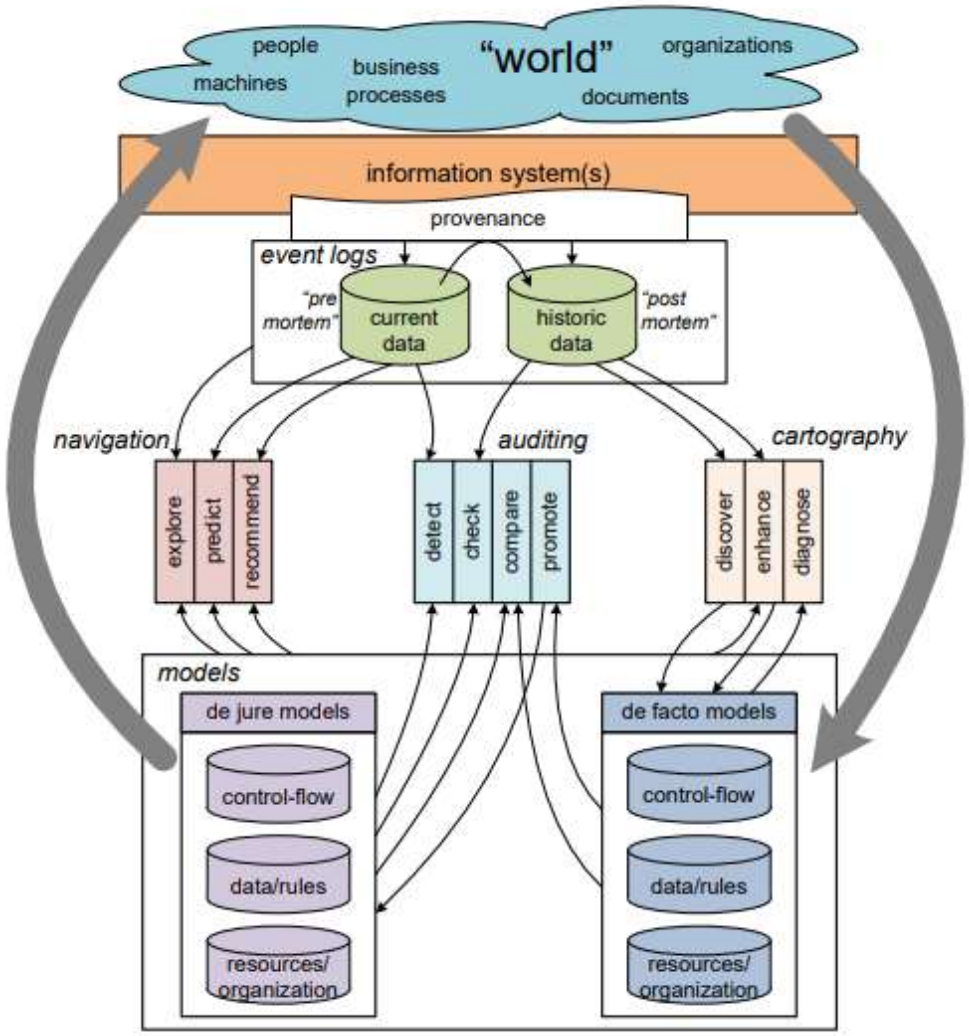
A de facto model is **descriptive**. Its purpose is to describe the "as is" process, and not to steer or control reality. De facto models aim to capture reality.



A de jure model is **normative**. It specifies how things should be done or handled

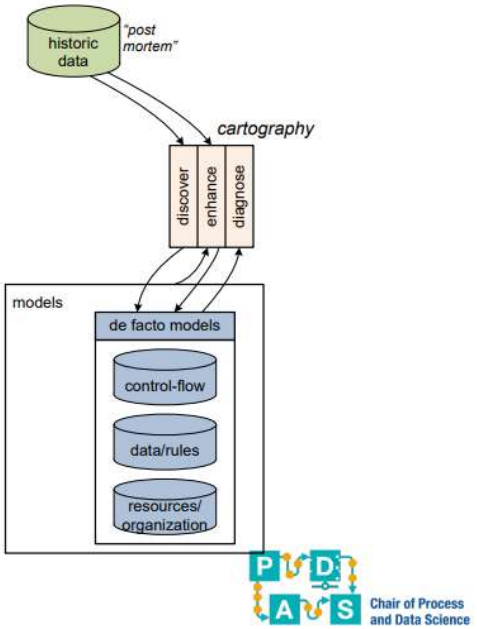


Refined process model overview



Cartography

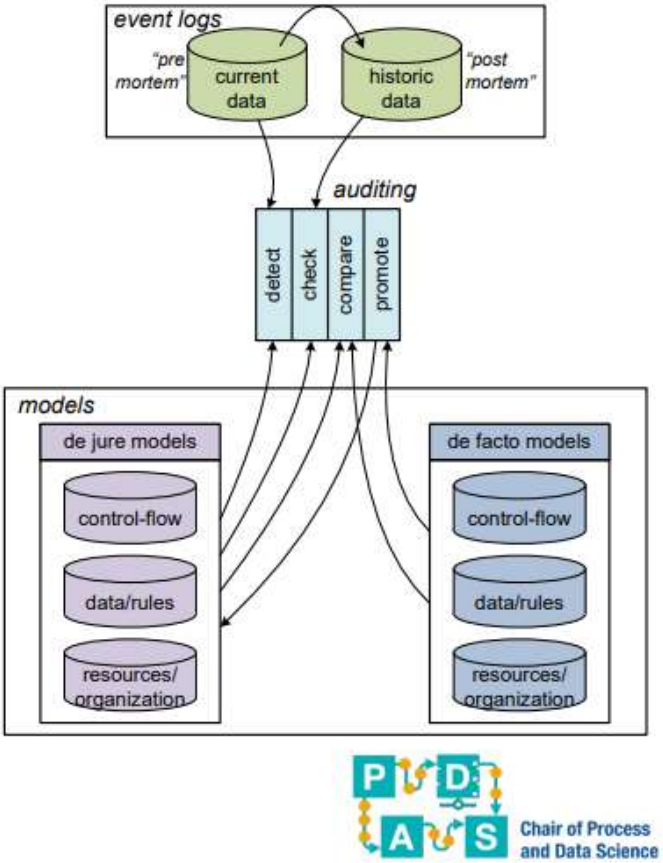
- **Discover**: This activity is concerned with the extraction of (process) models.
- **Enhance**: When existing process models (either discovered or hand-made) can be related to events logs, it is possible to enhance (extend and repair) these models.
- **Diagnose**: This activity does not directly use event logs and focuses on classical model-based analysis.



Auditing

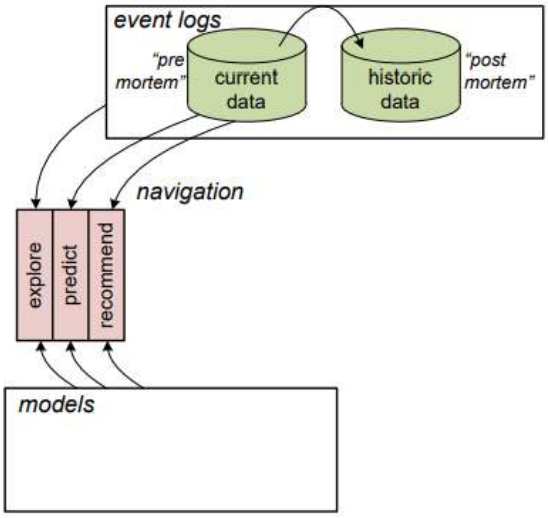
- **Detect**: Compares de jure models with current "pre mortem" data. The moment a predefined rule is violated, an alert is generated (online).

- **Check**: The goal of this activity is to pinpoint deviations and quantify the level of compliance (offline).
- **Compare**: De facto models can be compared with de jure models to see in what way reality deviates from what was planned or expected.
- **Promote**: Promote parts of the de facto model to a new de jure model.



Navigation

- **Explore**: The combination of event data and models can be used to explore business processes at run-time.
- **Predict**: By combining information about running cases with models, it is possible to make predictions about the future, e.g., the remaining flow time and the probability of success
- **Recommend**: The information used for predicting the future can also be used to recommend suitable actions (e.g. to minimize costs or time).



Operational support

Operational support focuses on the following three activities from the refined process model:

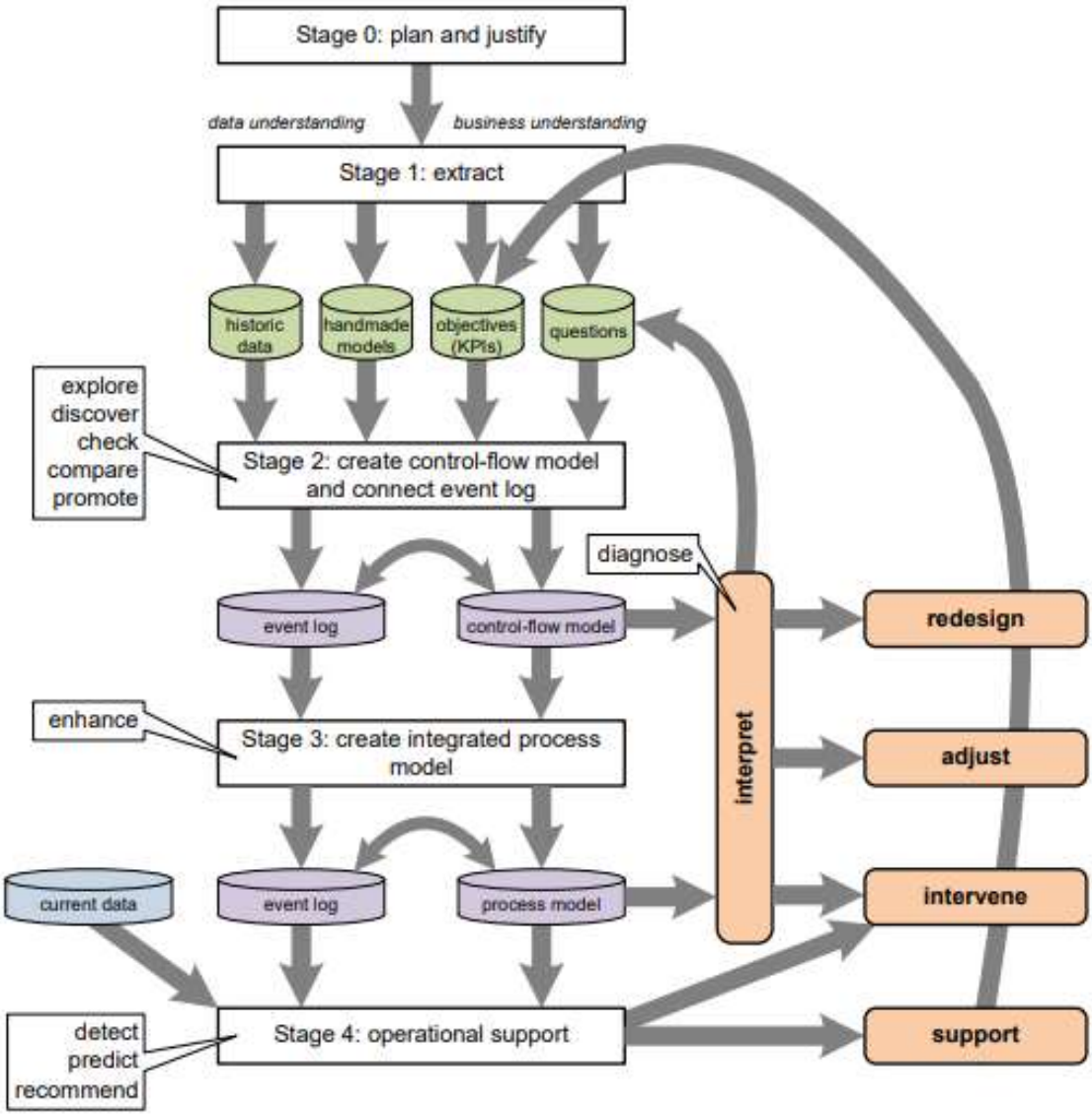
- **Detect**:
  - Something is going wrong now!

    - This case is deviating now!
    - The deadline just expired!
- **Predict**:
    - When will the case finish?
    - Will the case be rejected?
    - Will the case deviate?
- **Recommend**:
    - Which activity should be executed?
    - Who should execute it?

L* lifecycle model

There are many use cases of process mining. The goal is to improve KPIs related to **time**, **costs** or **quality** of a process. Many different actions can be employes:

- **Redesign** the process
- **Adjust** the process
- **Intervene**
- **Support**



Stage 0: plan and justify

There are three types of objectives one can pursue:

- **Data-driven**: (curiosity driven)
- **Question-driven**
- **Goal-driven**: improve a certain KPI

If we have a goal we can plan the project. The planned activities should also be justified

Stage 1: extract

We need to locate and extract the event data. We also collect models and their artifacts, objectives and questions. In this step it is favorable to exploit existing domain knowledge.

Stage 2: Create control-flow model and event log

The control flow model is the backbone of any process. Therefore, we should first create a suitable control-flow model that is well connected to the available event data. Conformance checking and alignments are key. This step might also be iterative (like other stages)

Stage 3: Create an integrated process model

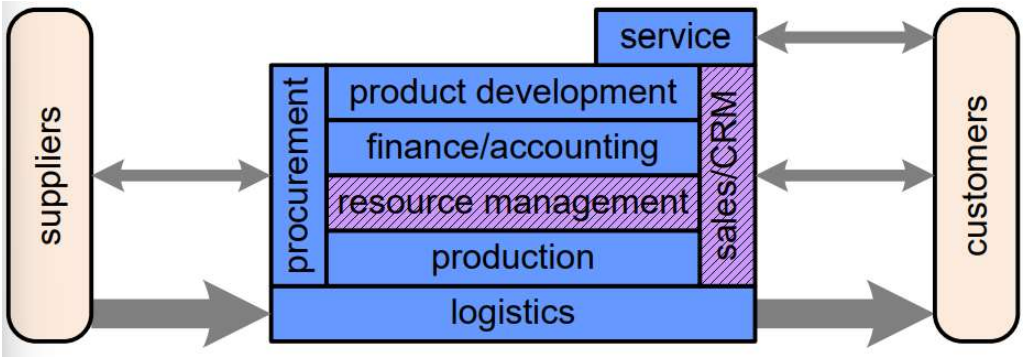
We replay the event data on the control-flow model to learn about other perspectives such as time, data ressources,… The perspectives are merged into an overall model showing the different perspectives

Stage 4: Operational support

Use current (pre-mortem) data for on the fly deviation **detection**, **predictions** and **recommendations**. Note that this is only possible for Lasagna processes.

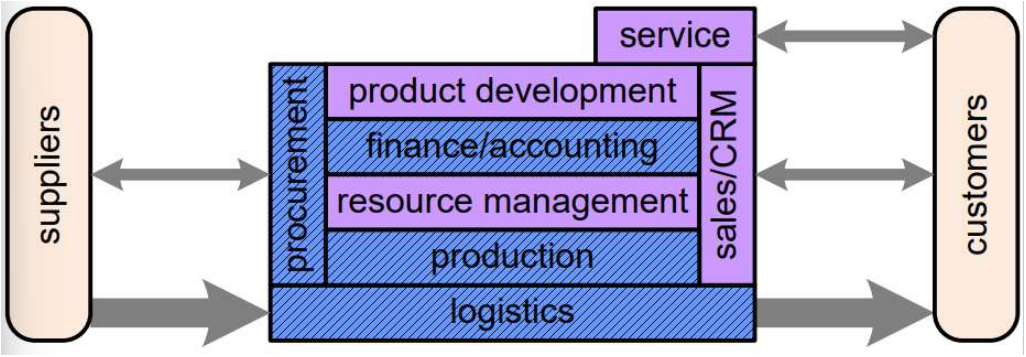
Lasagna processes vs spaghetti processes

Lasagna process: structured, regular, controllable and repetitive process



Purple sections are sections which often have lasagna processes

Spaghetti process: unstructured, irregular, uncontrollable and variable process.



Purple sections are sections which often have spaghetti processes

Dealing with Big Event Data

The volume of data might be so large that we cannot store all of them. For an event log multiple metrics can be employed

- Number of cases
- Average length of cases
- Number of distinct activities
- Number of distinct cases
- …

Pareto distribution

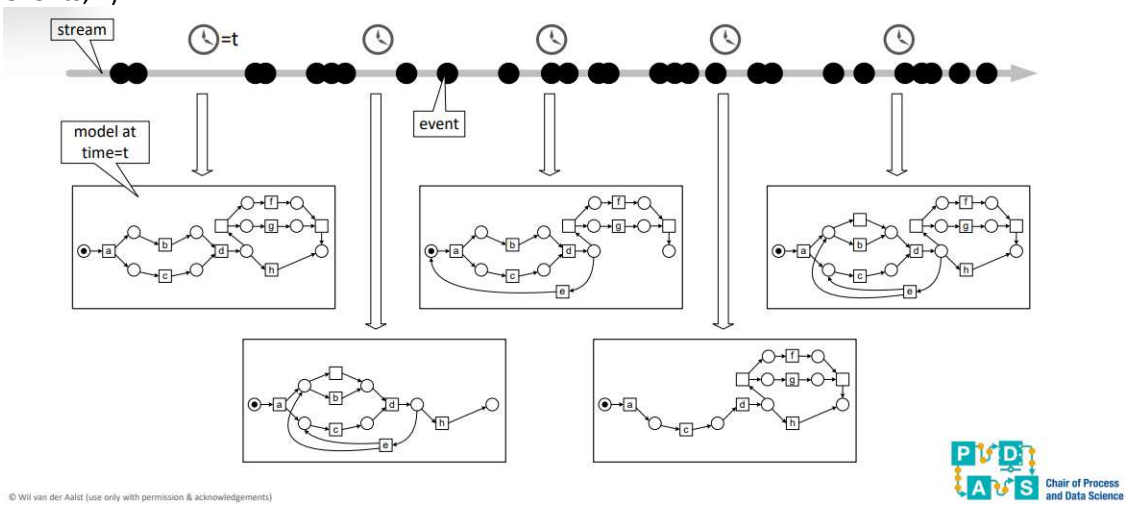
The Pareto principle or "80-20 rule" states that 80% of outcomes are due to 20% of causes

From <https://en.wikipedia.org/wiki/Pareto_distribution>

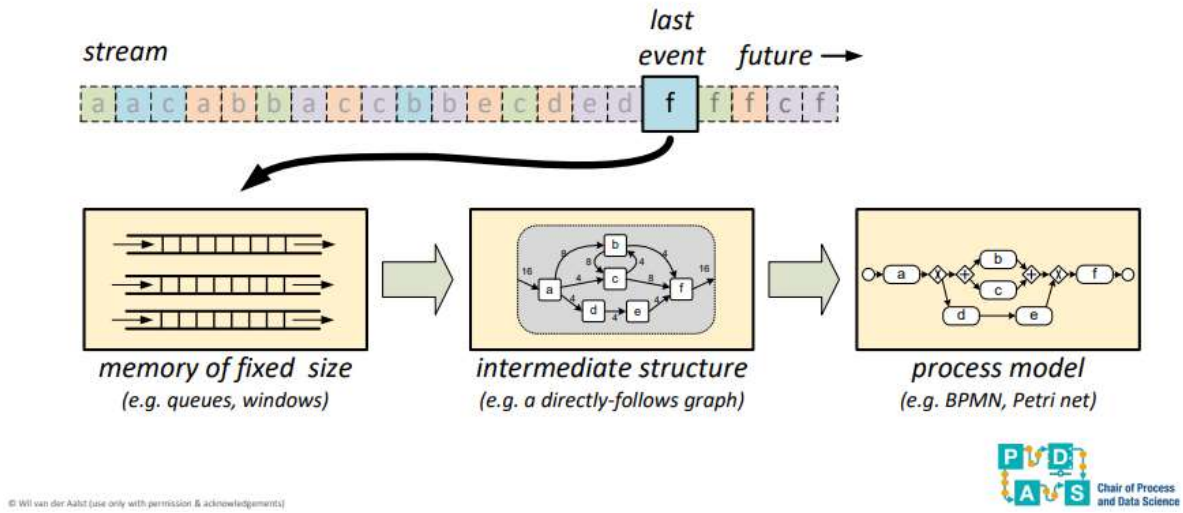
Often times, a small fraction of case variants describe most of the behaviour in a log. Thus, we can deal with spaghetti models by projecting the log on the most frequent case variants. Another method is to group cases into clusters and project the log on the clusters.

Streaming process mining

If event data is a stream of events we can create process models for a particular time point t, from the events which are still stored in memory. (We need to decide how long data is stored. Many possibilities are possible here e.g. store data for a particular time interval, for each case store k events,…)

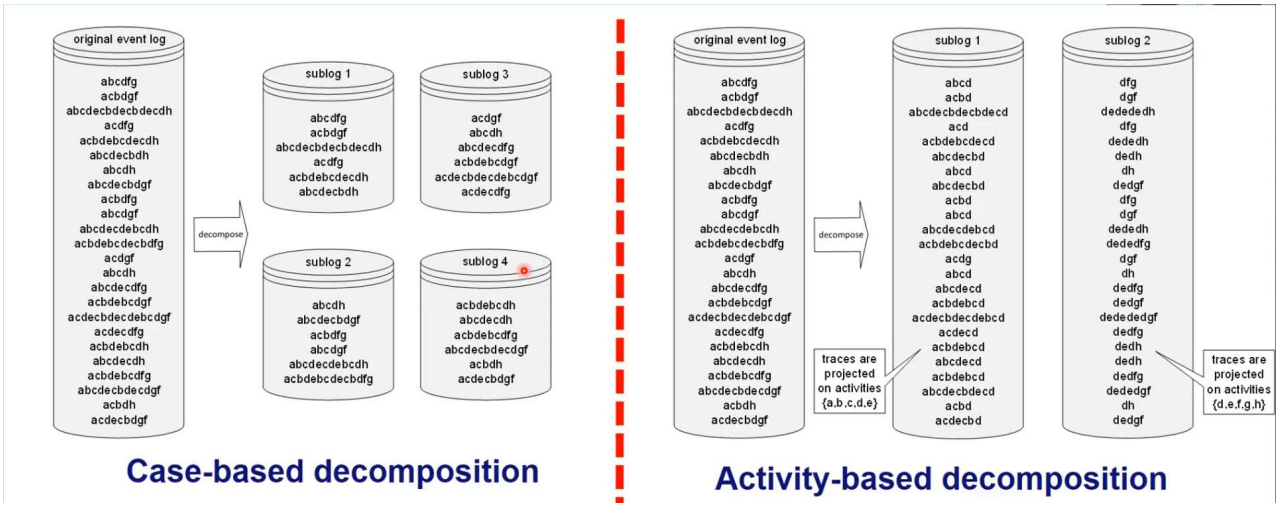


We might also store only the last event for each case



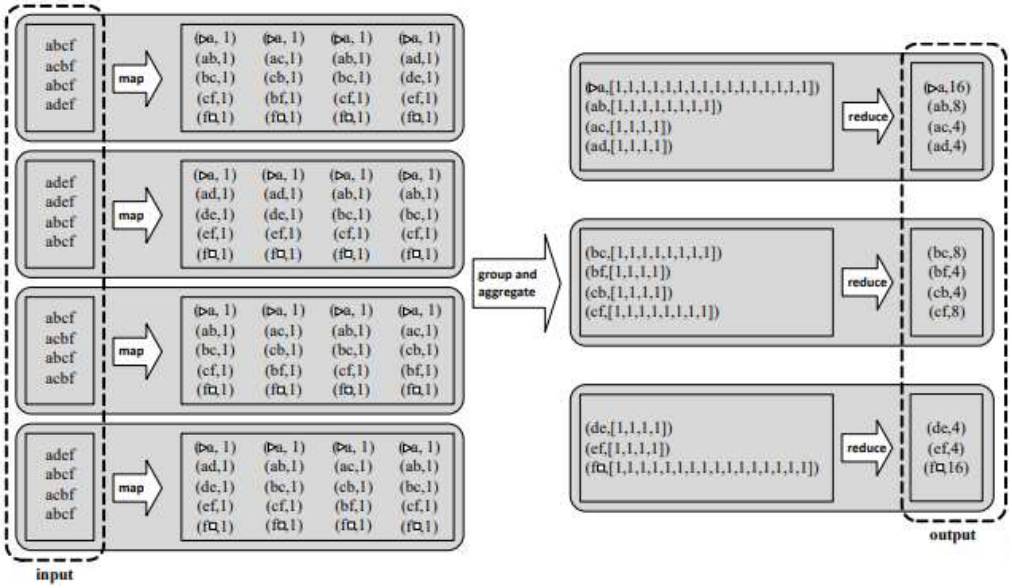
In both cases, the problem is that we never know when a case is finished (It could always be that at time $t + 1$ a new event arrives) Furthermore, it is difficult to know which will be the start and which the end activities

Distributed process mining

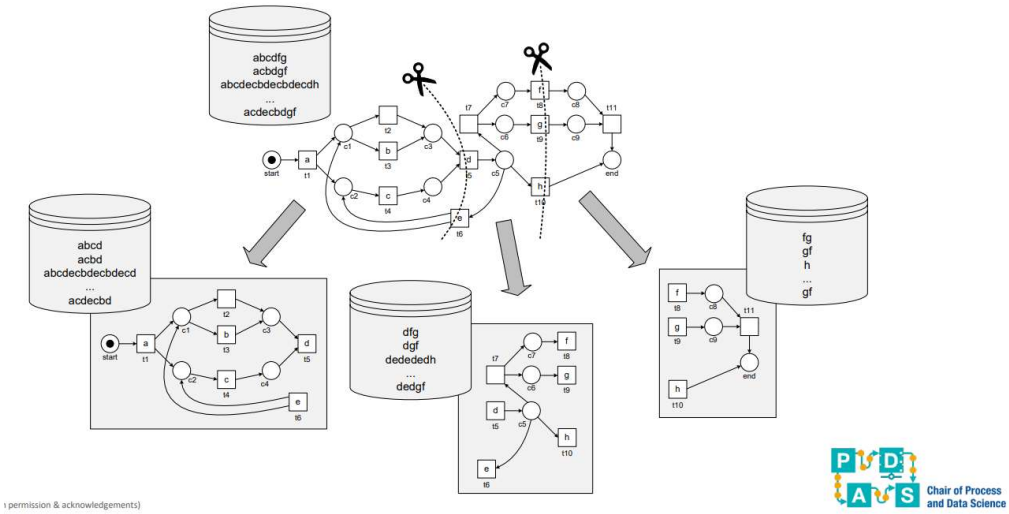


There are two main ways of decomposing data:

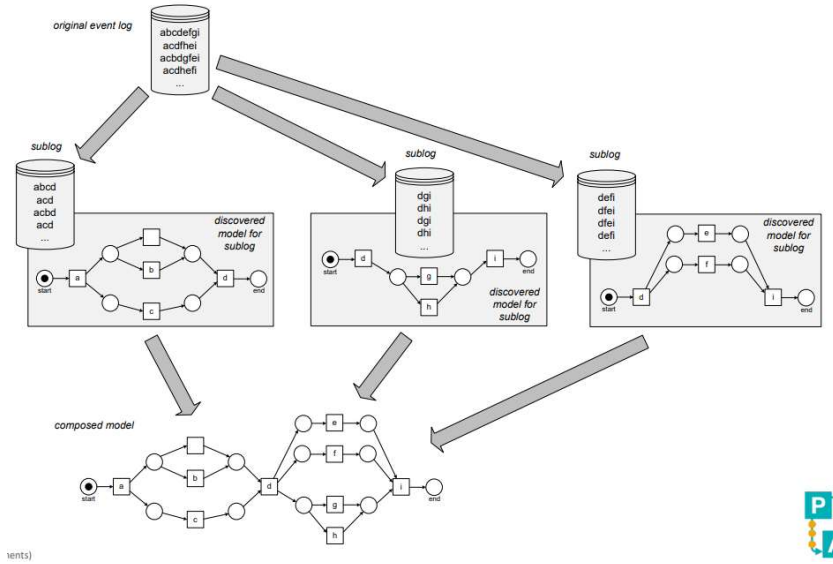
1. **Case-based**: create a DFG for each sub log and then merge them. This can be done efficiently with MapReduce. Case-based decomposition should be used whenever we want to apply algorithms that are based on counting



2. **Activity-based**: Algorithms which have an exponential factor should use activity based decomposition



Example of conformance checking using activity based distribution. Note that we can only cut on activities and not places. A trace is fitting iff its subparts are fitting in each distributed process model.



Example of process discovery using activity based decomposition