Conformance Checking of Processes Based on Monitoring Real Behavior
Seminar - Multimedia Retrieval and Data Mining

Aljoscha Steffens

Data Management and Data Exploration Group
RWTH Aachen University
Germany
`Aljoscha.Steffens@Rwth-Aachen.de`

Abstract. Process models are used to depict complex processes in a structured way. Processes generally consist of two different parts: the idealized model version that is often created before the execution and the execution of the process in reality. Companies nowadays possess not only the knowledge about the process model, but also about the real world execution by logging events. This poses the question if the model conforms to reality. The paper "Conformance Checking of Processes Based on Monitoring Real Behavior" from A. Rozinat and W.M.P. van der Aalst [1] presents ideas and metrics of how to measure this conformance. The first measure is called *Fitness* and shows how good the logged events fit to the underlying model, that means if they could have been generated by a process that follows the model. It is shown that this measure alone is important but not sufficient to measure the conformance so the concepts of *Behavioral Appropriateness* and *Structural Appropriateness* are introduced. They evaluate if the model represents the process in a suitable way, that means if the model represents the behavior that actually takes place and if it is minimal in structure.

1 Introduction

The usage of Business Process Models in corporations increases due to the wide range of applications they can be employed for and information they offer. They can be used to provide insight into a process, to help discussing the process with stakeholders, for documentation purposes, performance analysis, to find errors in systems or procedures and much more [3]. Due to the mentioned grows in application, these models are often developed by non-expert modelers [5] or generated artificially. Algorithms for the automated generation of process models derive them with the help of event logs (see 2.1). So there is a process that is monitored on the one hand and a model that describes the process on the other hand. This poses the question if they align properly. The reasons for a misalignment are manifold but can be seen from two different perspectives: Either the model is wrong or the process wasn't executed in a proper manner. The model can be wrong for different reasons. The process itself might have changed over time due to changes of laws or changes in the management. It could be that the process wasn't thoroughly planned and thus wrong from the beginning. In this cases the model can be repaired [1] or constructed anew. If the process wasn't executed correctly, the reason could be a schedule that is too tight for workers to fulfill, or that workers didn't behave how they should which could identify fraud. The reasons for both viewpoints are numerous and may not be easily visible but require the help of domain experts. The metrics and ideas that are proposed in [1] can help to indicate discrepancies between model and reality and show how it may be possible to localize them.

2 Preliminaries

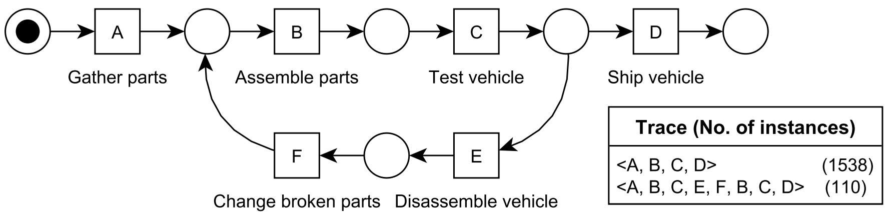
The following two sections explain the concepts of event logs and Petri-Nets with the help of a simplified example of a car assembly. The third section shows the basic functionality of the process discovery algorithms that were used in this paper to construct example models, namely the *Alpha-Algorithm* and *ILP-Miner*.

2.1 Event log

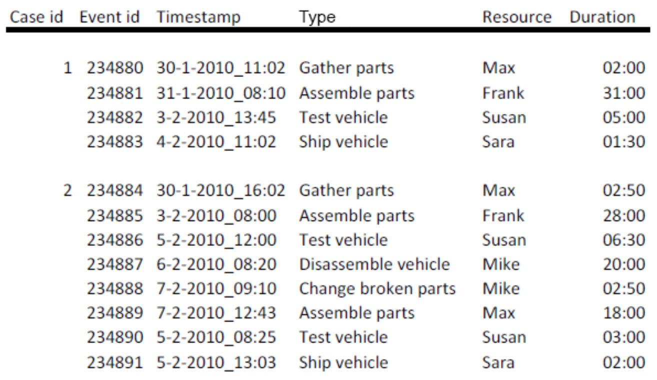
A process generally consists of several activities, i.e. steps that have to be executed for the process to complete. A strongly simplified example of the assembly of a car is: Individual components need to be gathered, then they need to be assembled and finally the vehicle is tested. If everything works as expected, the car gets shipped, otherwise some parts need to be changed (the event log and Petri-Net that correspond to this example are depicted in figure 1). The event log captures information about this and typically consists of three layers. The first layer lists all different cases. A case contains all activities/events that are part of one distinct execution, so in the example it would consist of all activities that were executed in order to produce one car. And an activity itself combines all pieces of information that are gathered in one step of execution. The amount of information within an activity can vary a lot, generally it contains information like the task that was executed, start- and end-time, persons that were involved and the case-id it belongs to.

Typically not all information is needed at the same time. So for example the automatic generation of process models from event logs just needs information about the type of task (in the following this is represented by a letter) and the order of execution. Since a lot of differentiating attributes are left out, a lot of cases become indistinguishable and are thus referred to as *traces*. So the condensed version of a event log just lists all different traces and shows how often they were executed.

In addition to *"normal"*, that means *visible* tasks, a Business Process Model can also contain *invisible tasks*. Invisible tasks are those which don't have a corresponding log event. They can appear if, for example, the task is not observable (or logable) in the real world. An example could be a telephone call: If the caller does not log the event him-/herself, there will be no entry in the log. In process models invisible tasks can be used for routing purposes (e.g. to bypass a visible task) or to delay visible tasks.



Trace	(No. of instances)
<A, B, C, D>	(1538)
<A, B, C, E, F, B, C, D>	(110)



Case id	Event id	Timestamp	Type	Resource	Duration
1	234880	30-1-2010_11:02	Gather parts	Max	02:00
	234881	31-1-2010_08:10	Assemble parts	Frank	31:00
	234882	3-2-2010_13:45	Test vehicle	Susan	05:00
	234883	4-2-2010_11:02	Ship vehicle	Sara	01:30
2	234884	30-1-2010_16:02	Gather parts	Max	02:50
	234885	3-2-2010_08:00	Assemble parts	Frank	28:00
	234886	5-2-2010_12:00	Test vehicle	Susan	06:30
	234887	6-2-2010_08:20	Disassemble vehicle	Mike	20:00
	234888	7-2-2010_09:10	Change broken parts	Mike	02:50
	234889	7-2-2010_12:43	Assemble parts	Max	18:00
	234890	5-2-2010_08:25	Test vehicle	Susan	03:00
	234891	5-2-2010_13:03	Ship vehicle	Sara	02:00

Fig. 1: Simplified process model and event log for the assembly of a car

2.2 Petri-Net

Petri-Nets are bipartite graphs that can be used to depict Business Process Models (see figure 1). They consist of four different elements: places (depicted as circles), transitions (depicted as squares), arcs (depicted as arrows) and tokens (depicted as black dots). Transitions represent the different steps that the model consists of, so they are linked to activities in the real world. The model itself is static, but tokens can *flow* through the network: Places can contain tokens and a transition is enabled if all preceding places contain tokens. When a transition is enabled, it may *fire*, whereas one token of each input place is consumed and one token on each output places is produced. The distribution of tokens represents the *state* that the process is in, i.e. it shows which steps can possibly be executed. A formal definition of Petri-Nets is given in [3] as following:

Definition 1 (Petri-Net) *A Petri-Net is a triplet $N = (P, T, F)$ where P is a finite set of* places*, T is a finite set of* transitions *such that $P \cap T = \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the* flow relation*. A* marked *Petri-Net is a pair (N, M) where $N = (P, T, F)$ is a Petri-Net and $M \in \mathbb{B}(P)$ is a* multi-set *over P denoting the* marking *of the net.*

A subset of Petri-Nets are the so called *Workflow-Nets*. They have a dedicated *start*- and *end*-place which is convenient for describing Business Process Models. Also every node is on a path from source to think. Most of the models that are shown in this paper are Workflow-Nets. A formal definition of Workflow-Nets is given in [3] as following:

Definition 2 (Workflow-Net) *Let $N = (P, T, F, A, l)$ be a (labeled) Petri-Net and $\bar{t}$ a fresh identifier not in $P \cup T$. N is a* Workflow-Net *if and only if (a) P contains an input place i (also called source place) such that $\bullet i = \emptyset$, (b) P contains an output place o (also called sink place) such that $\bullet o = \emptyset$ and (c) $\bar{N} = (P, T \cup \bar{t}, F \cup \{(o, \bar{t}), (\bar{t}, i)\}, A \cup \{\tau\}, l \cup \{(\bar{t}, \tau)\})$ is strongly connected, i.e., there is a directed path between any pair of nodes in N.*

2.3 Process discovery algorithms

This section gives an overview of the process discovery algorithms that were used in this paper to generate process models from logs. The goal was not to generate models that fit as good as possible to the log files. They were just used to generate a lot of different models from different logs to identify strength and weaknesses of the quality measures that are presented in [1].

α-algorithm
The α-algorithm is one of the oldest algorithms that is used for process discovery. Based on four relations between tasks it generates a Workflow-Net from an event log (W). The relations are:

- $a{>}b$ iff there is a trace $\sigma = t_1 t_2 t_3 ... t_{n-1}$ and $i \in \{1, ...n-2\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$
- $a \longrightarrow b$ iff $a{>}b$ and $b \not> a$
- $a \# b$ iff $a \not> b$ and $b \not> a$
- $a || b$ iff $a{>}b$ and $b{>}a$

The algorithm consists of 8 steps:

1. Generate the set of transitions based on the log file
2. Generate the set of start transitions (transitions that appear first in a trace)
3. Generate the set of end transitions (transitions that appear last in a trace)
4. Determine which transitions are connected and derive the possible places in the Workflow-Net (based on the relations given above)
5. Remove places that appear more than once between transitions (remove all ”non-maximal pairs”)

6. Generate source and sink place
7. Generate arcs
8. Return Workflow-Net

In the original version the α-algorithm is not able to generate loops with two elements or less and was thus improved in later versions (for example in the α^+-algorithm that is part of the ProM software) [9].

ILP-Miner
Another process discovery algorithm that is part of the ProM software is the ILP-Miner [6]. The ILP-Miner is an algorithm, that returns a Petri-Net which is optimal with respect to the *fitness*-metric. It uses Integer Linear Programming to decide whether adding a place to the model would reduce the fitness. If so, the place won't be added. A problem of this algorithm is the runtime: Since basically every combination is tested the runtime is exponential in the number of transitions. So even for an input with view activities (<20) the algorithm takes a lot longer to finish than for example the α-algorithm.

3 Quality Measures

At first glance, the question of whether a model and a log conform could be answered by checking if the model is able to generate the traces in the log. This is an important property and is measured by the *Fitness* metric. Even though that would technically be enough, in practice the term *conform* implies more. It is very easy to construct simple models that are able to generate the desired log traces as seen in figure 2. The problem is that they don't describe the underlying process very well. The *Flower-Model* is able to generate the log traces, but also allows for other traces of unrestricted length with an arbitrary execution order of the tasks. That means it is too general. The *Sequential-Model* is the complete opposite: it just lists the log traces. Even if this may be all the log traces that the model should allow for, this is still not a desired structure of a model as it does not reveal any information about the corresponding process. That means it does not help to gain insight about the process nor is it of any more use than the process log itself. This shows that another dimension is needed to check whether model and log conform, which is called *Appropriateness* since it measures if the model describes the process appropriately.

4 Fitness

As presented above, the fitness metric evaluates if the traces in a log could actually be generated by the model. A first approach could be to divide the number of possible traces by the number of all traces. This is a valid idea and can be used to measure fitness but it neglects the fact, that two traces can be false to a different degree. A trace that's completely false should be punished more

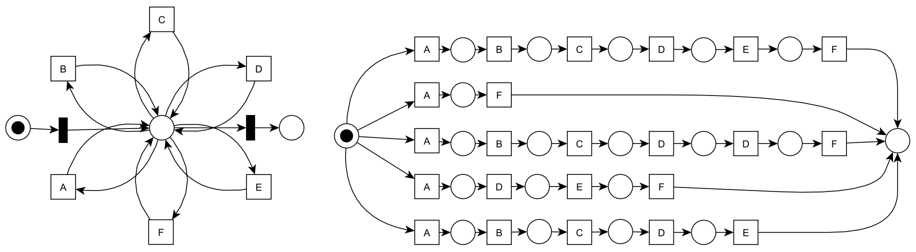


Fig. 2: Left: *Flower-Model* (black bars indicate invisible tasks), Right:*Sequential-Model*, both possible models for the traces ⟨A, B, C, D, E, F⟩, ⟨A, F⟩, ⟨A, B, C, D, D, F⟩, ⟨A, D, E, F⟩ and ⟨A, B, C, D, E⟩

than a trace that's just missing a single transition. To incorporate the amount of correctness, each trace is *replayed* with the model. Replay means, that one tries to generate the trace with the model. That however is done in a non-blocking, log-based manner. That means every transition that should be executed will be executed. If the previous places in the Petri-Net are not occupied, or if the following places are, this defectiveness will be captured but won't obstruct the replay. This leads to four counters that capture the behavior during replay: *produced tokens*, *consumed tokens*, *missing tokens* and *remaining tokens*. If a transition can be executed without errors, i.e. all previous places are occupied and all following places are free, the counter for *consumed tokens* increases by the number of incoming arcs and the counter for *produced tokens* increases by the number of outgoing arcs. If previous places are empty, they will be filled and the counter for *missing tokens* increases accordingly. The counter for *remaining tokens* counts all tokens that are still in the Petri-Net when the replay is finished. An example for the replay of a log is given in figure 3. The process model that is shown here was automatically constructed by the Alpha-Algorithm [8] that is part of the ProM software. The traces that were used are: ⟨A, B, D, E, F⟩, ⟨A, D, B, E, G⟩, ⟨A, D, C, E, H, C, D, E, H, B, D, E, F⟩ and ⟨A, C, D, E, F⟩. Some intermediate steps were left out for simplicity reasons.

The following metric can be constructed from the counters to quantify the *Fitness* of the model.[1].

Metric 1 (Fitness)

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

In the metric k is the number of different traces, n_i the amount of occurrences of the current trace, m_i, r_i, c_i and p_i the counters for missing, remaining,

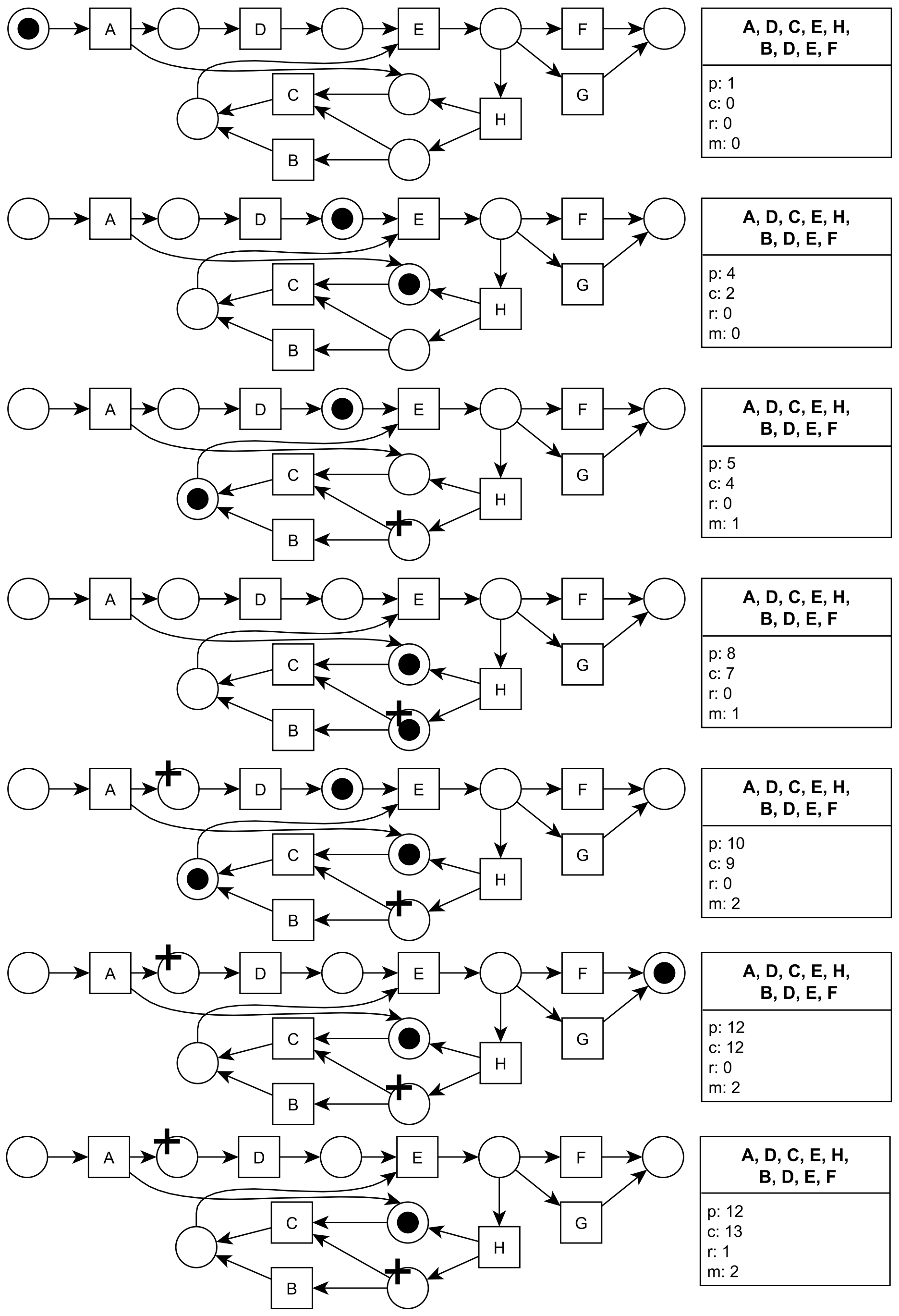

A
B
C
D
E
F
G
H
+
A, D, C, E, H, B, D, E, F
p: 1 c: 0 r: 0 m: 0
A, D, C, E, H, B, D, E, F
p: 4 c: 2 r: 0 m: 0
A, D, C, E, H, B, D, E, F
p: 5 c: 4 r: 0 m: 1
A, D, C, E, H, B, D, E, F
p: 8 c: 7 r: 0 m: 1
A, D, C, E, H, B, D, E, F
p: 10 c: 9 r: 0 m: 2
A, D, C, E, H, B, D, E, F
p: 12 c: 12 r: 0 m: 2
A, D, C, E, H, B, D, E, F
p: 12 c: 13 r: 1 m: 2

Fig. 3: Example steps of the log replay with corresponding counters. The "+"-symbol marks places where a token was missing.

consumed and produced tokens. Note that there is always at least one token produced at the *start*-place and one token consumed at the *end*-place, so the denominators can't be zero. Also for all i, $m_i \leq c_i$ and $r_i \leq p_i$ so that $0 \leq f \leq 1$.

Beside a metric that allows a more sophisticated view than just counting the number of invalid traces, the replay of a log may offer meaningful insight in the shortcomings of the model. If after a replay one can observe for example that before a certain transition there is a place with a lot of remaining tokens and afterwards a place with a lot of missing ones, one can assume that there is a scenario in the real world that makes the event being omitted for a certain reason. A domain expert can now use the information to investigate further and decide for example that the deviation is correct and that the model should be adapted to allow for it. Or that the step cannot be neglected so that the execution of the process should be controlled more frequently. As for example in figure 3 there are two modifications of the model that achieve a better *Fitness*. In the current state, it is not possible to generate any valid log since transition E can't be executed. For E to be executed, either B or C need to be executed and for them the execution of H and thus E is necessary. It would be possible though, if there was only one place that connects A and H to both B and C. The other modification is a connection between H and the place before D. This of course is a rather simple model and thus the problems can easily be spotted. In reality those models are typically a lot bigger and more complex. In case that the model was created thoroughly at the beginning and only changed gradually over time, the structural problems should be limited to certain parts of the model and therefore be relatively easy to spot and repair after an exhaustive replay of the log. If, on the other hand, the model was generated from an incomplete log file by an algorithm and the underlying process included a lot of concurrency and loops, the resulting model can become very hard to read and even with the information about missing or remaining tokens it can be hard to locate errors and enhance the model.

Other than conformance checking, a replay can be used to detect bottlenecks by extending the model with frequencies and temporal information. It can be used to construct predictive models and to offer operational support [3].

5 Appropriateness

As mentioned above, it is easy to create models that have a *Fitness* of one, the structural extreme examples are the *Flower-Model* that allows for arbitrary behavior and is thereby too general and the *Sequential-Model* that just lists the different log traces and is thus too specific. So it's the common discrepancy between under- and overfitting. In [1] those properties are measured separately in terms of *Behavioral Appropriateness* and *Structural Appropriateness.*

5.1 Behavoral Appropriateness

Behavioral Appropriateness addresses the issue of underfitting. It measures how much behavior is allowed by the model and compares it to how much can actually be found in the log. If the log does not include the behavior that is possible according to the model, either the log is incomplete or the model allows for behavior that is not necessary, that means it's too general. The first approach of measuring this property that is proposed in [1] also relies on replaying the log entries with the model. The idea is, that if a model is very general, it will have many alternative routes and thus many transitions will be enabled during replay (for the *Flower-Model* every transition will be enabled at any time). The metric is defined as follows:

Metric 2 (Simple Behavioral Appropriateness)

$$a_B = \frac{\sum_{i=1}^{k} n_i(|T_V| - x_i)}{(|T_V| - 1) * \sum_{i=1}^{k} n_i} \tag{2}$$

In the metric k is the number of different traces, n_i the amount of occurrences of the current trace, x_i the mean number of enabled transitions during log replay of the current trace and $|T_V|$ is the number of visible tasks. The problem with this metric is, that it will rank those models high that are as specific as possible which is the *Sequential-Model*. The author of [1] suggests another metric that is less structural dependent, but analyses how much behavior is actually needed by the model to describe the log entries. The metric uses the concepts of tasks $following$ and *preceding* each other globally. That is, if there is a task x in the trace and in the same trace there is a y afterwards, y follows x. Equivalently for *precede*. It doesn't matter if there is another x afterwards, the condition needs to be met only once. This captures global behavioral dependencies. There are three grades though: *always following/preceding*, *sometimes following/preceding* and *never following/preceding*. In [1] they are defined as following:

Definition 3 (Follows relations) *Two activities (x, y) are in "Always Follows", "Never Follows" or "Sometimes Follows" relation in the case that, if x is executed at least once, then always, never or sometimes also y is eventually executed, respectively.*

Definition 4 (Precedes relations) *Two activities (x, y) are in "Always Precedes", "Never Precedes" or "Sometimes Precedes" relation in the case that, if y is executed at least once, then always, never or sometimes also x was executed some time before, respectively.*

Both relations can be computed for both model and log. The idea is to measure how much more specific the log is compared to the model. The *sometimes*-relations play a big role in this concept, since they show concurrent or alternative behavior and thus indicate flexibility. If the model now allows two tasks to follow each other (they are in a $sometimes\text{-}follow$-relation) and in the log this relation is more strict (either $always\text{-}follow$ or $never\text{-}follow$) the model's behavior is potentially to big (always considering that the log may be incomplete). If it is the other way around (meaning the model is more specific than the log) this indicates a *Fitness* problem.

In the following metric S^l_F and S^m_F denote the sets of $follow$-relations with value *sometimes* of log and model. S^l_P and S^m_P denote the sets of *precede*-relations with the value *sometimes*. The intersection of two sets are those elements which have the value *sometimes* in both model and log.

Metric 3 (Behavioral Appropriateness)

$$a'_B = \frac{|S^l_F \cap S^m_F|}{2 * |S^m_F|} + \frac{|S^l_P \cap S^m_P|}{2 * |S^m_P|} \tag{3}$$

Note that both the $follows$- and the *precedes*-relation are taken into account since the relations are not symmetric and including them both makes the metric more stable[1]. Note further that, for a technical reason [1], artificial transitions for *Start* and *End* were created. The metric ranges from 0 (if in both relations model and log do not conform regarding the *sometimes*-elements) to 1 (they share exactly the same elements with value *sometimes*).

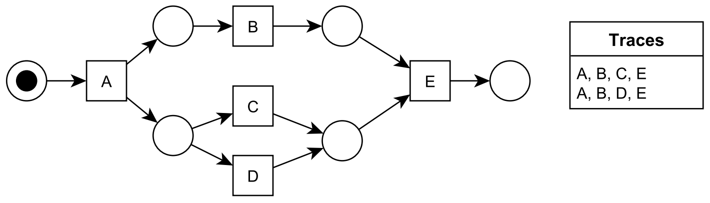
Figure 4 shows a model that allows concurrent execution of B and C or D. This concurrency cannot be observed in the log. The result of the metric is $a'_B = \frac{6}{2*8} + \frac{4}{2*8} = \frac{5}{8}$. Still, the model is not necessarily false. In reality the execution of a task is not only restricted by the execution of the previous ones, but can have other reasons that are for example linked to the availability of resources. Thus it *may* be correct, that there is a concurrent execution of tasks B and C/D even though task B is always executed first. The reason can be, that B is executed automatically and for C and D personnel is required. If the concurrency was now removed from the model it would appear as if the execution of B is mandatory for the execution of C and D which would lead to a decreased descriptiveness of the model. This of course is only a special case but it shows that it is not sufficient to rely only on the metrics - good knowledge about the process is still necessary and cannot be neglected when evaluating a process model.

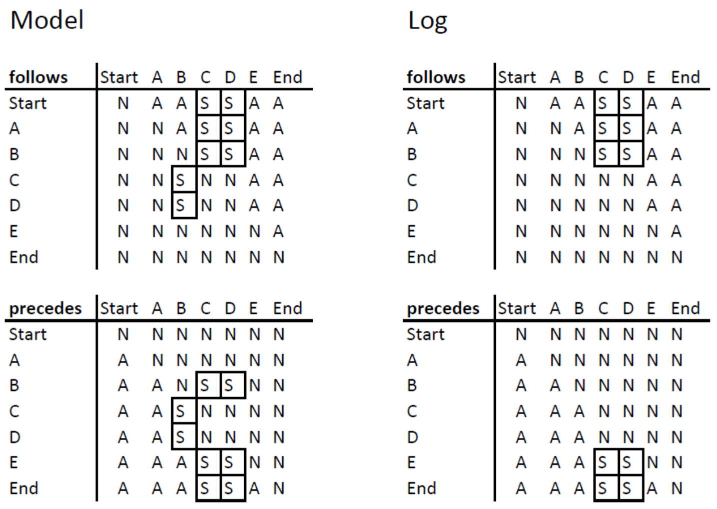
As with the *Fitness* metric, the results that are obtained from creating the $follows$- and *precedes*-relation can be used to increase the model quality. If the model allows that a task is in a *sometimes*-$follows$-relation with itself and in the log the task never follows itself, this indicates that there might be an unnecessary loop in the model. Again, the conclusions that can be drawn from this analysis are manifold and require help of a domain expert.

5.2 Structural Appropriateness

For most processes there are several different models, which are able to perform the same behavior, yet their structure may differ substantially. The problem is that there are only guidelines for developing models and those can change a lot due to personal and corporate preferences. Still there are indications for a structurally bad model such as *alternative duplicate tasks* and *redundant invisible tasks*. Those structures tend to inflate the process model and should thus be

[1] See [2]





Model

follows	Start	A	B	C	D	E	End
Start	N	A	A	S	S	A	A
A	N	N	A	S	S	A	A
B	N	N	N	S	S	A	A
C	N	N	S	N	N	A	A
D	N	N	S	N	N	A	A
E	N	N	N	N	N	N	A
End	N	N	N	N	N	N	N

precedes	Start	A	B	C	D	E	End
Start	N	N	N	N	N	N	N
A	A	N	N	N	N	N	N
B	A	A	N	S	S	N	N
C	A	A	S	N	N	N	N
D	A	A	S	N	N	N	N
E	A	A	A	S	S	N	N
End	A	A	A	S	S	A	N

Log

follows	Start	A	B	C	D	E	End
Start	N	A	A	S	S	A	A
A	N	N	A	S	S	A	A
B	N	N	N	S	S	A	A
C	N	N	N	N	N	A	A
D	N	N	N	N	N	A	A
E	N	N	N	N	N	N	A
End	N	N	N	N	N	N	N

precedes	Start	A	B	C	D	E	End
Start	N	N	N	N	N	N	N
A	A	N	N	N	N	N	N
B	A	A	N	N	N	N	N
C	A	A	A	N	N	N	N
D	A	A	A	N	N	N	N
E	A	A	A	S	S	N	N
End	A	A	A	S	S	A	N

Fig. 4: A simple model that contains concurrency.

avoided [1]. Duplicate tasks can have two different origins. They may be needed if a task occurs in completely different context and thus needs to be separated. An example could be that you wipe the table before and after dinner. On the other hand, there are duplicate tasks which serve the structure of the process model, but could be replaced by other techniques. The *Sequential-Model* consists of a lot of duplicate tasks that can easily be avoided for example by splitting the model paths after the first transition. A first intuitive measure that comes to mind when examining the sequentialized model is to compare the number of transitions and places to the number of different labels. This is captured in the following metric [1].

Metric 4 (Simple Structural Appropriateness)

$$a_S = \frac{|L| + 2}{|N|} \tag{4}$$

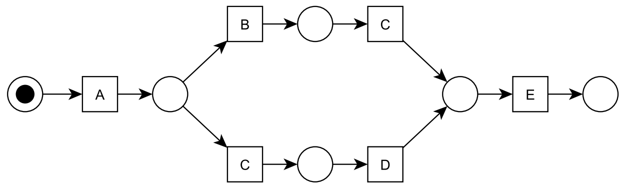
$|L|$ is the number of different labels ($|L| + 2$ accounts for the start and end place) and $|N|$ the number of places and transitions. Places are also taken into account since it is possible to inflate a model unnecessarily just by adding places. The problem here is that this metric only takes the model size into account and so just allows two models with the same behavior to be compared [1]. The following metric accounts for that by incorporating alternative duplicate tasks (T_{AD}) and redundant invisible tasks (T_{RI}). $|T|$ is the number of transitions in the Petri-Net. Note that duplicate tasks are defined to be visible and thus $|T_{AD}| + |T_{IR}| < |T|$.

Metric 5 (Structural Appropriateness)

$$a'_S = \frac{|T| - (|T_{AD}| + |T_{IR}|)}{|T|} \tag{5}$$

The difficulty here is that one needs the number of invisible and duplicate tasks to compute the metric. According to [1] the retrieval of redundant invisible tasks is fast and can be done via a *structural analysis* of the model. Alternative duplicate tasks can be retrieved by a *state space analysis* with the help of a coverability graph. However, process models can become very large, which makes the computation very expensive, since the number of paths in a model grows exponentially. Yet there are techniques available to deal with this problem such as partial order reduction and symmetry methods [1].

This metric is targeted towards identifying models that are too sequential. As an overall measure this is a good approach, but still it does not always need to be valid. In figure 5 there are two simple models that allow for the same behavior. The first one was created by hand, the second with the *ILP-Miner* that is part of the ProM software [6]. According to the *Structural Appropriateness* metric, the second model should be preferred: It does not contain any redundant invisible tasks or alternative duplicate tasks, so it results in a value of one. The upper one has a value of $a'_S = \frac{5}{6} = 0.83$. Still, the upper model should be preferred due to its simple structure. When looking at the lower model, one would think at first that, because of the occurrence of loops and concurrency, a lot of different traces can be generated by it. *Simplicity* is, next to *Replay Fitness*, *Precision* and *Generalization*, one of the four quality criterions that are generally used to evaluate results of process discovery techniques [7] and it is of great importance when the model is used as a base for discussion or documentation purposes. The given example shows that it can be necessary to neglect the *Structural Appropriateness* for the benefits of a simpler model.



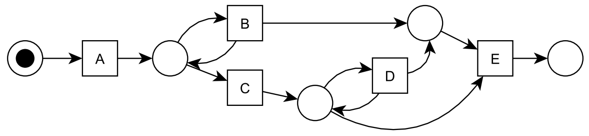


Fig. 5: Two Petri-Nets that allow for the execution of the traces ⟨A, B, C, E⟩ and ⟨A, C, D, E⟩. The upper one results in a worse *Structural Appropriateness* than the lower one, but is a lot easier to read.

6 Conclusion

The metrics that are presented in [1] give a good overview of the conformance of process model and log and tackle the quality dimensions *Replay Fitness*, *Precision* and *Generalization*. The ideas that result in the metrics are comprehensible and allow both the comparison of two models of different structure and size as well as an estimation of the absolute quality[2]. There are however examples that contradict the results of the metrics, especially when the fourth quality dimension, *Simplicity*, is taken into account. Sometimes it supports the understandability and readability of a model, if for example duplicate tasks are allowed. A Petri-Net that is shrank to the bare minimum of transitions and places may result in an (apparent) increase of complexity. And since one of the major purposes of process models is to discuss a process, gain insight in it or document it, this should not be the goal. So together with the presented metrics it is necessary to evaluate other properties of process models: In [4] the authors suggest quality measures that come from the field of software development, in [5] the metrics are partly based on measures from graph-theory and give a good estimation of the structural quality. Also as in most topics that have more than one quality dimension, it is not enough to solely rely on the numbers of metrics; a good knowledge about the process itself also plays a fundamental role in the evaluation.

Besides the ability to evaluate a model according to the *Fitness* measure, it was shown that the replay technique is also able to help localizing errors in the model. However, the extent to which the replay is useful for *repairing* a model is

[2] All three metrics are between 0 and 1

limited by the complexity of it, which can increase very easily, especially if the model was not created by hand.

The *Fitness* and *Behavioral Appropriateness* metric can be computed quickly and don't require complex algorithms and according to [1] there are also algorithms available that speed up the computation of redundant invisible tasks and alternative duplicate tasks.

All in all, it can be said, that the presented metrics can give a good overview of the conformance of process models and logs but it requires the help of domain experts to evaluate the metrics and interpret the results.

References

1. A. Rozinat, W.M.P. van der Aalst *Conformance Checking of Processes Based on Monitoring Real Behavior*, Eindhoven University of Technology, Netherlands, 2008.
2. A. Rozinat, W.M.P. van der Aalst *Conformance Testing: Measuring the Alignment Between Event Logs and Process Models*, Eindhoven University of Technology, Netherlands.
3. W.M.P. van der Aalst *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, Published by Springer, Berlin Heidelberg, Germany, 2011.
4. I. Vanderfeesten, J. Cardoso, J. Mendling, H. Reijers, W.M.P. van der Aalst *Quality Metrics for Business Process Models*, Eindhoven University of Technology, Netherlands 2007.
5. Laura Sánchez-González, Flix Garca, Jan Mendling, Francisco Ruiz, Mario Piattini *Prediction of Business Process Model Quality based on Structural Metrics*, University of Castilla La Mancha, Spain.
6. T. van der Wiel *Process Mining using Integer Linear Programming*, Eindhoven University of Technology, Netherlands, 2010.
7. J.C.A.M. Buijs, B. F. van Dongen, and W. M.P. van der Aalst *On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery*, Eindhoven University of Technology, Netherlands.
8. W.M.P. van der Aalst, T. Weijters, L. Maruster *Workflow Mining: Discovering Process Models from Event Logs*, IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No.9, 2004
9. A.K.A. de Medeiros, B.F. van Dongen, W.M.P. van der Aalst, A.J.M.M. Weijters *Process Mining: Extending the α-algorithm to Mine Short Loops*, BETA Working Paper Series, WP 113, Eindhoven University of Technology, Netherlands, 2004