Towards Applicability of Language Based Synthesis for Process Mining

Robin Bergenthum, Robert Lorenz, Sebastian Mauser

Lehrstuhl für Angewandte Informatik
Katholische Universität Eichstätt-Ingolstadt, Eichstätt, Germany
e-mail: {robin.bergenthum, robert.lorenz, sebastian.mauser}@ku-eichstaett.de

Abstract

In this paper we give an overview of methods to adapt Petri net synthesis based on regions of languages for process mining. These methods are understood to supplement the basic process mining approach presented in [BDLM07].

1 Introduction

Many of today's information systems record information about performed activities of processes in so called event logs. Not only classical workflow management systems, but also web services, middleware systems, embedded systems in high-tech equipment such as medical systems and many other information systems produce event logs. Process mining techniques attempt to extract useful, structured information from the vast amount of information recorded in event logs. In this paper we focus on constructing a process model, which matches the actual workflow of the recorded information system, from an event log. This prevalent aspect of process mining is known as process or control-flow discovery. There are many process discovery techniques in literature (see e.g. [ADH + 03]), often implemented in the ProM framework [PE].

In [BDLM07] we proposed a process discovery approach based on the theory of regions of languages. It aims at constructing a Petri net from a typical event log recording the executed activities together with cases (i.e. process instances) the activities belong to. Assuming that the cases of a log are executed independently, each case defines a sequence of activities describing the ordering of the execution of the activities. Thus an event log can be interpreted as a set of traces defining a language. That means, in a natural way methods derived from the theory of regions of languages can be applied to synthesize a Petri net. In [BDLM07] we presented two basic process discovery algorithms constructing a Petri net from an event log. This process discovery approach is a precise one in the sense that it basically constructs a Petri net reproducing the traces of the event log and having *minimal* additional behaviour.

In contrast, many classical process discovery approaches are imprecise. In order to be efficient in time and memory consumption and to keep the resulting process models small, these process models allow for much more additional behaviour as necessary – they *overapproximate* the given event log. Our precise approach overcomes many of the limitations of such classical imprecise approaches. These have problems with complex control-flow structures such as non-free-choice constructs, unbalanced splits and joins, nested loops, etc., although in reality processes often exhibit such features. These problems are resolved by the precise algorithms in [BDLM07]. Of course any process discovery algorithm is in particular restricted to control-flow structures allowed by the target language for the process model. The approach in [BDLM07] allows to easily vary the target language between several classes of Petri nets. The second main problem of imprecise methods is their tendency to *underfit* the event log, i.e. the resulting model allows for much more behaviour without any indication to be reasonable real behaviour. This problem of underfitting is of course irrelevant for precise models.

On the other hand, precise methods are often criticized for their tendency to *overfit* [AG07]. Of course a log is usually incomplete, i.e. there may be possible traces not occurring in the log. On the first glance this argues against precise approaches. But without additional information it is unclear, which possible traces are missing in the log. It is hardly possible to extract information about such missing traces solely from the event log. The overapproximation performed by most existing imprecise approaches seems to be quite arbitrary, at least hardly controllable and predictable. When using the precise approach as a basis, it is possible to specifically introduce overapproximation. In this paper we develop several modular methods to adapt the basic precise approach in order to get a process model overapproximating the event log in a specific, targeted direction. Some additional information or expert knowledge may be used to determine the directions of overapproximation. This makes the overapproximation controllable and predictable, and (since the starting point is a precise algorithm) in particular leads to reliable results. The selection possibility of overapproximation methods makes the final process discovery algorithm configurable to address different needs of the user.

A problem with precise process mining methods is, that they may be inefficient in time and memory consumption. In our situation, one of the two algorithms presented in [BDLM07] has a polynomial runtime and memory consumption, while the other one may be exponential in the worst case. Since we have no experimental results so far, we cannot

say if the complexity of the two basic algorithms is sufficient for practical applications. In any case, we will propose modular implementable techniques to improve the runtime and the memory consumption of the algorithms in this paper. In particular some overapproximation techniques improve the runtime and the memory consumption.
Lastly, the key drawback of a precise process discovery method is, that it usually generates a quite large process model. The number of components of the model – in our situation mainly the number of places of the Petri net – often has to be large in order to potentiate an exact reproduction of the event log. This is the real problem arising from the overfitting of precise algorithms. The size of the model has to be in such a range, that the model can easily be interpreted by the user. Practitioners and process analysts in industry are usually interested in concise and controllable reference models. "Spaghetti"-models that can only be evaluated with computer support are mostly not satisfactory. Therefore we propose modular methods to reduce the number of places and arcs of the net, mined by our approach. Of course there is a trade-off between simplifying the mined net and maintaining a preferably precise model. Therefore many of the proposed simplification methods overlap or coincide with overapproximation methods. In order to get a compact model, also abstraction and visualization techniques for the mined model are useful. The danger of mining "spaghetti"-models is especially problematic in logs of unstructured processes in less restrictive environments [AG07]. However this is not a specific problem of our approach, but also of most classical process mining techniques.
We also propose some methods to improve the mining quality by integrating possible additional information going beyond the pure event log. That means, we are interested in exploiting specific additional information sources or expert knowledge to improve the matching of the constructed process model with the actual flow of work. Besides a better reproduction of reality, additional information is also used in the improvement methods concerning performance, compactness and overapproximation. If such additional information is existent, it is in our opinion important to offer possibilities to the user to exploit such information for the construction of the process model.
Altogether in this paper we make proposals for modularly implementable improvement methods to yield a practically useful and applicable process discovery tool from the precise algorithms in [BDLM07]. The methods primarily cover four aspects: specific targeted overapproximation, improvement of the performance, reduction and simplification of the resulting process model and better reproduction of the reality. The methods are either targeted heuristical techniques or techniques exploiting additional information sources. The methods can be modularly integrated to the process mining algorithms. The considerations of this paper will show that our precise algorithms are very well suited to integrate such methods. Thus we propose a huge number of methods all applicable to our approach. In a practical setting our process mining approach can then very well be configured to the needs of the user by offering the possibility to modularly choose an appropriate selection of these methods. The effects and biases of these methods are well predictable, such that the complete process mining approach leads to reliable, interpretable and reasonable results. The next section gives a comprehensive overview of possible improvement methods to our precise process mining algorithms. We sketch basic methods, their integration to our mining approach and their contribution to make our mining algorithm applicable and high-quality in practice. We do not explain the methods, especially their technical fundamentals, in detail in this paper.

2 Methods to Improve the Process Mining Approach Based on Regions of Languages

In [BDLM07] we described two basic process mining algorithms based on regions of languages. In this section we will restrict ourself to the more promising polynomial algorithm using separating regions and propose improvement methods towards a practical applicability. This algorithm computes a Petri net from an event log basically as follows: The transitions are given by the set of action names occurring in the given language. Each place (together with its connections to all transitions) is computed as a non-negative integer solution of some linear inequation system. This inequation system is chosen in such a way, that w.r.t. a place, which is a solution, all traces of the event log are still possible, while there is at least one trace not belonging to the log, which is prohibited by the place. Such places are computed for a finite set of such traces not belonging to the log, called *wrong continuations*. Wrong continuations extend traces from the log by one additional event.
Before we sketch the improvement methods for our process mining algorithm, we give a systematic classification of these methods. We identified four dimensions for the methods:

(A) *The aim of the improvement method*: Combination of one or more of the issues targeted overapproximation, better performance of the algorithm, more compactness of the resulting model and better reproduction of reality.

(B) *The procedure of the method*: Methods based on heuristics or additional information beyond the pure event log. Heuristics are general principles that can be applied for each log and bias the approach to a certain direction. They often try to extract probably reasonable assumptions for the resulting process model from the pure log. In contrast, additional information is typically very specific, e.g. expert knowledge about relationships of certain activities or some other external information sources. Also logged information in the event log beyond activities and cases is additional information in our context.

(C) *The phase of the process mining approach, to which the method is applied*: Pre-processing (filtering) phase of the event log, processing phase (the actual mining algorithm working on the pre-processed log), and phase

of post-processing of the resulting process model. We are not only concentrating on methods adapting the processing phase, but we also consider pre-processing and post-processing methods specifically tailored to the mining algorithm. Pre-processing and post-processing methods are especially important methods to realize the formulated aims, because these methods do in no way influence the actual precise mining algorithm. Therefore the changes made to the result of the process mining approach by pre- and post-processing methods are completely controllable.

(D) *The user interaction*: Different shapes depending on the improvement methods. Of course there may be no user interaction, but asking an experienced user for help in some problematic or unclear situations may significantly improve the process mining result. The degree of user interaction may be determined by the user. A possibility to avoid constant user interaction is to offer the user the possibility to parameterize the improvement methods. The parameter then defines to which degree a certain method is applied in the final process mining algorithm.

We organize the presentation of the methods according to their phase (C) (pre-processing, processing, post-processing). A classification of the methods to the other three dimensions is no problem: The aims covered by a method are usually obvious. It is also clear, if additional information is required for the method. The user interaction for a method can usually be varied from non to a-priori settings to a confirmation request for every step of the method.

2.1 Pre-processing phase

In this phase the given event-log is filtered to get a less complicated event log. Typically, some *activities are completely deleted* from the log. These activities are not considered in the resulting process model. Such activities are *infrequent activities* or activities classified as *unimportant activities* (e.g. detected from data fields or resources in the event log), *noise activities* or *uninteresting activities* (e.g. start events for activities, that are divided into a start and a complete event). These activities can be identified using some additional information, but also algorithms searching in the pure log for special characteristics to detect such activities are imaginable, e.g. based on frequencies or more advanced stochastic models. User interaction may be very helpful in this case.
Sometimes it is also important to *combine several activities to one activity* (e.g. because of similar names or relations to occurrences of other activities) or to *split one activity into several activities* (e.g. because of extremely different relations to occurrences of other activities). Combined activities will refer to the same transition in the process model, while events of a split activity refer to different transitions. These transformations of the log are of special importance in our approach, because our basic mining algorithm allows no label splitting. Also additional information can very well be exploited for these transformations, e.g. the coincidence of logged data fields, transactional information and originators of events or knowledge about similarity of certain activities. Even adding completely *new routing activities* to the log can be useful. That means one could try to detect patterns in the log, that provide an indication for hidden routing activities.
Furthermore *removing*, *adding* or *editing entire traces, sub-traces or parts of traces* of the log are of special interest. Since the algorithm is precise, a removed behaviour will not be included in the mined process model, while an added or edited behaviour will. Removing traces simplifies the log. As for activities, infrequent (several cases define the same trace and thus traces have frequencies w.r.t. the log), unimportant, noise, uninteresting, exceptional or even faulty traces may be identified and deleted. If it is possible to identify faulty traces, they seem to be a problem in the real system and deserve special attention. It is reasonable to not only delete them, but also store them as *undesirable behaviour*. There may also be faulty traces not occurring in the event log. These may also be detected, altogether yielding a set of explicitly unwanted traces. Such a set can very well be exploited by the actual mining algorithm as sketched later on. Adding traces supports a specific overapproximation. The aim is to add traces, that are (probably) possible in the real system, or do marginally influence the real system but simplify the mined model. Traces may be edited by aggregating sub-traces or parts of traces to one activity. Editing traces is also important to account for special dependencies of cases, e.g. by analyzing resources, originators and data of events of different cases. This is an important task of pre-processing, because the actual mining algorithm assumes, that the cases are executed independently. Editing traces also includes tasks like splitting a trace (e.g. to better reflect a recorded case) or combining traces (e.g. to collapse similar cases or to merge related cases), similarly as for single activities. The possibilities to detect candidate traces for methods described in this paragraph are similar as for respective methods concerning single activities.
For traces as well as for activities it makes sense to build *stochastic models*. That means instead of simply deleting less relevant traces or activities, one can associate each trace or activity with a certain relevance score. Such *relevance rating* differentiates in detail between less representative and highly significant traces or activities. Analogously, instead of simply combining similar traces or activities one can introduce *similarity ratings*. These rating procedures can also be refined by rating sub-traces or certain patterns of activities. Not only frequencies, but also typical patterns and interrelationships of activities or traces as well as additional information can be used to develop a stochastic model based on such ratings. As shown later on, evolved ratings can very well be used by our mining approach, possibly leading to better results than just deleting or combining traces or activities.
Pre-processing also plays a role in detecting probable specific relations of events, e.g. *causal independency/dependency* of events, *cycles* of events or events in *conflict*. Information on causally independent events can be used to transform

the log into a *partial language*, translating the traces to partially ordered scenarios. Another application is a reasonable *decomposition of the set of traces* of the log into behavioural (relatively) independent parts. Alternatively, a *decomposition of the set of activities* of the log into subsets of activities defining distributed components of the process is reasonable. There is a huge amount of possibilities to extract such relations of events from a log, e.g. frequencies and ratings (see above) of traces, activities and of certain patterns in the traces, similarities (given by ratings) of certain patterns, anomalies in the log, systematic deviations of certain patterns or dependencies of activities in the traces. Additional information in the log such as data fields of activities, originators of activities, time spans between certain events or transactional information as well as expert knowledge (e.g. given by some a-priori potentially incorrect model of the process) are particularly useful. Some information systems even directly record partially ordered cases or independencies of events (e.g. the ARIS tool). How information about such relations is integrated to the actual mining algorithm or post-processing methods, is discussed in the next two subsections.

2.2 Processing

Concerning the actual mining algorithm, first one can *vary the target language* of the process model according to the needs of the user. Our approach offers several Petri net classes, that vary in expressivity: General place-transition nets (p/t-nets), pure nets, workflow nets, nets with capacities, elementary nets, nets with unweighted arcs, etc. The structural requirements defining a target Petri net class are introduced into the algorithm by adapting the inequation systems, which are solved by our mining algorithm to define places. Many properties of the final net can easily be introduced by extending these systems with additional requirements for the places and their connections. In this way it is even possible to directly introduce some correctness properties for the final net into the algorithm. We do not refer here to correctness properties concerning the relation of the model and the log, but to the internal consistency of the model (e.g. concerning properties such as the popular soundness requirement for workflow nets).
Adaption of these inequation systems can also be used to introduce some *specific behavioural information* extracted in the pre-processing phase or from an external source (causal (in)dependencies of activities, cycles of activities, resource sharing of activities, etc.). There are three specially interesting examples of such behavioural information:

(i) *Translating the log into partially ordered behaviour*: Then arbitrary concurrency relations are reflected, also referred to as true concurrency. An adaption of our basic algorithm shown in [LBMD07], which regards flows of tokens, can be applied to yield analogous mining results in this more complex case.

(ii) *Decomposition of the set of traces into behaviourally (relatively) independent sub processes*: Mining process models from such smaller subsets of traces is more efficient than mining the whole log. The composition of the mined sub process models is an open task.

(iii) *Decomposition of the activities to distributed components of the process model*: This can easily be integrated to our algorithm by removing possible dependencies between activities from the inequation systems.

The second and third cases introduce modularity to the mining algorithm. They simplify the resulting process model by implementing a clear and concise structure. The modularity also improves the performance, since dealing with smaller problems is less costly. But not only behavioural information, also *additional state information* can be implemented to the process model. For example an information, that the state of the process after two different sequences of activities coincides, can easily be captured by adaption of the inequation systems.
Another possibility to account for special behavioural or structural information of the logged process, is to predefine certain components of the process model. In our case this means *predefining places*. Such places are not introduced to the mined net by the actual mining algorithm, but they are constructed beforehand. Examples of reasonable predefined places are initial or final places for processes, places defining known dependencies, resources or routings in the process and places given by some a-priori process model. Such an a-priori process model does not necessarily be defined as a Petri net, because most process modelling languages can be translated to Petri nets. It is possible to assume that certain places are correct. Alternatively, the mining algorithm can detect incorrect parts of an a-priori process model through a comparison to the mining result or to the traces in the event log. The mining algorithm can be used to improve the a-priori model, by *adding correct places* and by either *deleting or modifying incorrect places*. An a-priori model may be more abstract than the behaviour of system observed by the log. Then activities of the a-priori model need to be refined and the mining algorithm can use causal dependencies known from the a-priori model. Our algorithm can very well integrate predefined places and causal dependencies. The algorithm even benefits from having reasonable predefined places, because they can directly be integrated and exploited in the construction procedure of the Petri net.
There are also general procedures to *generate a more simple and concise net*. There are heuristical methods to intentionally introduce a simplified structure to the net. That means one tries to *avoid complicated routings*, to *introduce concurrency between activities* and to *ignore less significant places*. In particular highly branched places are filtered out (not allowed), simplified or decomposed. The overall number of places can be reduced by *ignoring certain wrong continuations* of some of the traces of the log (e.g. wrong continuations which are not specially defined as undesired behaviour). Some *information from the preprocessing phase also lead to simplified nets*, such as information on cycles of events, on independent or sparsely connected components or sub-processes, on split activities or on hidden

routing activities. Such information can mostly be integrated to our algorithm by adapting the considered inequation systems. A concrete example of a very promising general method to simplify the mined net is to only account for dependencies of a certain depth. That means, if one event occurs before another event in some trace, a dependency of the activities is only assumed, if there is at most a certain number of events in between the two events. If there are more events in between the activities, the activities are not allowed to be directly connected by a place. Such restriction of the dependency to a certain depth can easily be introduced to our mining algorithm, if we consider the variant considering flows of tokens [BDLM07, LBMD07]. Then we allow flows of tokens only of a certain depth. This maximal depth can also be varied, e.g. accounting for deeper dependencies in specially significant traces. This method is a very flexible method to abstract from certain dependencies (the α-algorithm [ADH+03] only considers dependencies of depth one). All these simplification methods in general lead to overapproximation. A restriction of the overapproximation can be introduced by a certain upper bound for the overapproximation defining some tolerance range. Such bound can be given by a maximal set or number of traces of the mined net or by a minimum value of some conformance measure, modelling the conformance of the process model and the log. To *control the degree of simplification*, stochastic methods based on frequencies or ratings (described in the pre-processing passage) are useful. An extensive control can be achieved by introducing *cost-functions*. Typical cost functions introduce *structural costs* for places, branching of places, arc weights, arc connections between certain distributed activities, arc connections to infrequent or low rated activities, depth of token flows respectively direct dependencies in the traces etc. and *behavioural costs* attached to certain traces or patterns not present in the log, explicitly unwanted behaviour, rare or low rated traces, etc. The structural costs ensure a concise model and the direction of overapproximation, while the behavioural costs restrict and control the overapproximation. Instead of introducing behavioural costs, it is also possible to fix an upper bound for the overapproximation. Also *fixed structural bounds* for the mined net, e.g. restricting the number of places, the maximal branching or the maximal arc weight, may be reasonable to guarantee a simple process model. Such costs and bounds can very well be integrated to our mining algorithm, because the algorithm introduces places step by step solving certain inequation systems. These steps may then turn into optimization problems or games with equilibria.

There are precise methods improving the compactness of the mined net and the performance of the mining algorithm. These methods are based on special characteristics of the mining algorithm. For a deeper insight to these methods we refer to [BDLM07]. Some of the methods are already explained in [BDLM07] in a more formal way. Remember, that places are constructed by solving certain inequation systems. The *choice of a solver* for these inequation systems is an important parameter of the algorithm. As described in [BDLM07] possible solvers are the Simplex algorithm, the Khachyan method or the method of Karmarkar and variants of these algorithms. The evolution of the inequation systems allows for example the application of an iterative Simplex variant, which is more efficient than the standard Simplex algorithm. The solvers differ in average as well as worst case runtime and may yield different solution places. The Simplex algorithm allows to define a *linear objective function* to influence the computed solution place. Typical objective functions may aim at *minimizing certain structural properties* of the net, e.g. arc weights, branching of places or conflicts. The definition of reasonable objective functions benefits from frequencies or ratings. To control the computed solution places, it is also possible to calculate the complete solution space, or at least several solutions, of the inequation systems. Then according to ones needs, one can choose one of these places to be added to the constructed net. There are several possibilities to choose an adequate place, e.g. (possibly non-linear) objective functions, places prohibiting many wrong continuations at once (thus reducing the overall number of constructed places), places respecting distributed components of the resulting net, etc. Beside the solver, the *order of the wrong continuations* is an important parameter. This order influences which inequation systems actually have to be solved. Therefore the set of constructed places, in particular the overall number of places, depends on this order. There are several possibilities to develop an order leading to a preferably concise set of places of the mined net. For example one may start with inequation systems associated with highly significant traces or one may solve several inequation systems in parallel and decide, which of the solutions should be integrated to the net first. As mentioned in the pre-processing passage, it is also possible, that in addition to the set of logged traces, there may be a *set of unwanted traces*. If simplifications are applied in the mining procedure, it may be necessary to explicitly exclude these traces in the actual mining algorithm. This can easily be done by regarding inequation systems excluding these traces, i.e. places prohibiting the unwanted traces are explicitly computed. Lastly it was mentioned in [BDLM07], that sometimes a direct wrong continuation of a trace of the log cannot be separated, but a follower wrong continuation can. *Prohibiting such follower wrong continuations (up to a certain depth)* for specially significant traces may improve the mining quality in some cases.

2.3 Post-processing

During the post-processing phase, the mined Petri net can be fine-tuned and a graphical representation can be built. The methods in this phase mostly focus on a final adaption of the mined net to the needs of the user. The most important challenge in practice is, that the model is readable and interpretable by the user. This challenging task requires well developed post-processing techniques, because, despite of several simplification methods in the pre-processing phase and the mining algorithm phase, the pure mined model can often still be quite complex.

A first possibility to a-posteriori simplify the mined model are *precise reduction techniques* for Petri nets. Such

techniques reduce the components of the net, while preserving its behaviour. Reduction techniques for Petri nets are well established. The most apparent method in our setting is identifying and deleting *redundant places*, also called implicit places. Basically it is possible to detect a minimal sized subset of the set of places of the mined model guaranteeing the same behaviour as the set of all places. Since detecting such set is very costly, more simple approaches to delete some redundant places have to be used. Many such approaches exist; some examples are explained in [BDLM07]. Besides deleting redundant places there are also more intricate reduction techniques, e.g. concerning symmetries, transitions, etc. Some of these more complex methods can also be useful for post-processing the mined net.

Furthermore there are non-precise simplification techniques. These are either pure heuristical methods or the methods exploit certain additional information. Such methods mostly aim for systematically *deleting or simplifying places*. Simplifying places means that connections to certain transitions are deleted to reduce branching or arc weights are reduced. Sometimes this requires decomposition of one place to several places, but sometimes places can also be combined. The methods are not only targeted on improving the readability of the mined net, but also on generating a clear structure of the net, e.g. divide the net into distributed components. Further *structural simplification techniques combine transitions*, *split transitions* (label splitting) or introduce *routing transitions*. Additionally, also *methods to simplify the behaviour* of the net are reasonable, e.g. *introducing additional concurrency, cycles or conflicts*. To guarantee that the model is not biased too much by such methods, the methods may be controlled by some conformance measure ensuring a certain degree of consistency between the model and the log.

For complicated, especially unstructured processes, the pure mined model may still be "spaghetti-like". The real process itself is often "spaghetti-like" and, despite of several simplification methods, an accurate process model reflecting such process is often hardly readable. Therefore *visualization techniques* for the mined models are a very important part of the post-processing phase. Examples of adequate basic visualization techniques are summarized in [AG07].

Finally there are also some post-processing methods solely trying to a-posteriori *improve the quality of the mined model*. In contrast to the previous methods, these methods are not tackling the readability of the model. Firstly, methods a-posteriori *checking correctness properties* of the mined net are of interest. These methods benefit from having a formal model, i.e. Petri nets, as the target language. There are many verification, validation and analysis techniques for Petri nets, that can be applied. Having detected certain inconsistencies of the model, it can be tried to fix them. Secondly, it is also possible to improve the quality of the mined model by a-posteriori editing the model to improve some *conformance measure*.

3 Conclusion

We proposed methods for our process mining approach to improve performance, to simplify the mined net and to overapproximate in a targeted direction. We moreover presented strategies to control the level of overapproximation and simplification. All these methods finally result in an appropriate adaption of the linear equation systems which are solved to compute places. Some methods need the application of an advanced region based synthesis method using token flows. The next steps are to implement these methods into VipTool and evaluate them using practical examples.

References

[ADH+03] Aalst, W. M. P. d. ; Dongen, B. F. ; Herbst, J. ; Maruster, L. ; Schimm, G. ; Weijters, A. J. M. M.: Workflow mining: A survey of issues and approaches. In: *Data Knowl. Eng.* 47 (2003), Nr. 2, S. 237–267

[AG07] Aalst, W. M. P. d. ; Günther, C. W.: Finding Structure in Unstructured Processes: The Case for Process Mining. In: *Proceedings of the 7th International Conference on Application of Concurrency to System Design (ACSD)*, 2007, S. 3–12

[BDLM07] Bergenthum, R. ; Desel, J. ; Lorenz, R. ; Mauser, S.: Process Mining Based on Regions of Languages. In: *Proceedings of the 5th International Conference on Business Process Management (BPM)*, 2007

[LBMD07] Lorenz, R. ; Bergenthum, R. ; Mauser, S. ; Desel, J.: Synthesis of Petri Nets from Finite Partial Languages. In: *Proceedings of the 7th International Conference on Application of Concurrency to System Design (ACSD)*, 2007, S. 157–166

[PE] ProcessMiningGroup ; EindhovenTechnicalUniversity: *ProM-Homepage*. – http://is.tm.tue.nl/ cgunther/dev/prom/