A First Glimpse at Petri Net Regions

Robin Bergenthum[1] and Jakub Kovář[2]

[1] *Fakultät für Mathematik und Informatik, FernUni in Hagen, Germany*
[2] *Lehrgebiet Programmiersysteme, FernUni in Hagen, Germany*

Abstract

Synthesis is automatically producing a process model from specified behavior. If the desired process model is a Petri net, synthesis is tackled by so-called region theory. Region-based synthesis has been extensively explored for cases where the specification language is a transition system, a language or even a partially ordered language. Although the ideas of region-based synthesis are the same for every kind of specification, every type of specification has its own region definition and uses different representations of the set of all regions to synthesize a finite result. Up to this point, state-based and language-based regions are just two different concepts.

In this paper, we introduce a new region definition we call Petri net regions and reason that every state-based and every language-based region is a Petri net region as well. Thus, there is no need to distinguish language-based and state-based regions anymore. With the help of Petri net regions every concept from one of the older region definitions can be directly applied to the other. Using Petri net regions, we introduce an implementation of a synthesis algorithm that handles state-based as well as language-based input. Furthermore, this algorithm can synthesize a Petri net from a set of labeled Petri nets.

Keywords

Petri Nets, Synthesis, Region Theory, State-Based Regions, Language-Based Regions

1. Introduction

Complex systems are often modeled by Petri nets [1, 2, 3, 4]. Petri nets have formal semantics, an intuitive graphical representation, and can express concurrency among the occurrences of actions of a system. However, constructing a Petri net model for a real-world process is a costly and error-prone task [3, 5]. Fortunately, whenever we model a system, there are often some associated descriptions or even specifications of the desired process behavior. There may be log-files of recorded behavior, example runs, and product specifications describing use cases. We can model these specifications by a language, a transition system, or a partial language. If a specification reflects the desired behavior faithfully, we can automatically synthesize the best fitting process model. The synthesis problem is to compute a process model so that: (A) the specification is behavior of the generated model and (B) the generated model has minimal additional behavior.

Looking at the literature, the theory behind Petri net synthesis is called region theory [6, 7]. Region theory has been extensively explored for transition systems, languages, and even partial languages. There are many non-trivial theoretical results, notions, case studies, as well as tool support by tools like for example ProM [8], Genet [9], APT [10], and Viptool [11].

Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2022, Bergen, Norway
EMAIL: robin.bergenthum@fernuni-hagen.de; jakub.kovar@fernuni-hagen.de
ORCID: 0000-0003-0464-8843 (Bergenthum); 0000-0002-7775-3698 (Kovář)


Today, region theory has two main branches. If the input is state-based, a transition system for example, we apply the theory of state-based regions [6, 7,12]. Here, a region is a multi-set of states, and we construct a set of minimal regions to generate a finite set of valid places for the resulting Petri net. If the input is language-based, an event log, a language, or a partial language for example, we apply the theory of language-based regions [13, 14]. Here, a region is a multi-set of tokens produced by prefixes of the language and we calculate valid places by solving a related integer linear programming problem (ILP). To generate a finite result, we calculate a basis of the ILP or we use the concept of wrong continuations [15]. Both branches of region theory follow the same ideas, but in detail use different techniques, definitions, and algorithms. To just give one very illustrative example, we can compare the two prominent process discovery algorithms based on region theory. The ILP-miner [16] is language-based, the region-miner [17] uses state-based techniques.

This workshop paper presents a glimpse at a new and very intuitive definition we call Petri net regions. This definition can handle state-based and language-based input at once. We define Petri net regions for transition systems, languages, partial languages, branching processes, and labeled Petri nets. We get our new definition simply by lifting the notion of compact tokenflow regions [14] from partial languages to labeled Petri nets. We lift the notion of state-based regions [6, 7,12] from transition systems to labeled Petri nets as well. We show that if we lift both notions, they will match perfectly. Furthermore, we will argue that Petri net regions can handle any given labeled Petri net as an input, even if this net is neither a transition system nor a (partial) language.

2. Preliminaries

In the following preliminaries, we present compact token flow regions, state-based regions, and Petri nets by instructive examples and refer the reader to the literature for formal definitions. We assume the reader is familiar with the basic concepts of region theory.

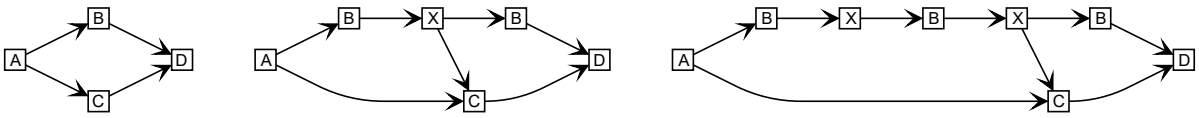

Figure 1: A partial language.

Figure 1 depicts a set of three labeled Hasse diagrams, i.e., a partial language. Every diagram models a run of the intended system's behavior. In every run, every node is called an event and models the occurrence of a transition of the Petri net to be synthesized. The set of arcs defines a later-than relation on the set of events. Thus, a partial language can directly express concurrency.

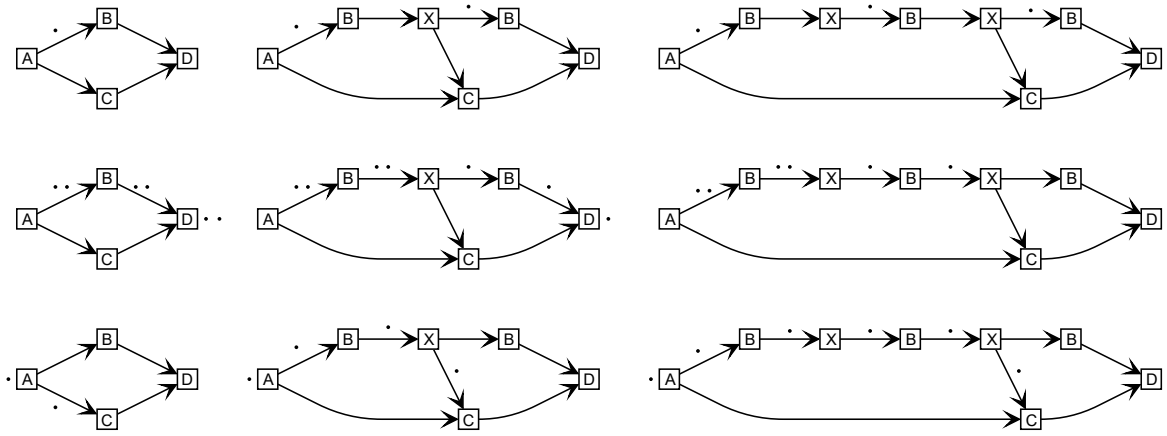

Figure 2: Three compact tokenflow regions.

To synthesize a Petri net model from a partial language we use the concept of compact tokenflow regions. Every compact tokenflow region is a distribution of tokens on the arcs of the specification and defines a valid place for the synthesis result. That means, every region describes the production, passing, and consumption of tokens between events in the related valid place and thus, adding this place to the synthesis result will generate a net which is still able to execute the specified behavior. Figure 2 depicts three copies of the specification of Figure 1 and three different compact tokenflow regions. The first region defines the valid place p2 of Figure 3. The second region defines the valid place p3. The third region defines the valid place p5.

A compact tokenflow based synthesis algorithm defines an ILP, so that every solution of the ILP is a compact tokenflow region. The solution-space of this ILP is not finite, but the set of all places related to the finite set of integer basis solutions will solve the synthesis problem. A second approach to receive a finite result from the ILP, is to use the concept of wrong continuations. Roughly speaking, the set of wrong continuations is the border between the specified and all other behavior. The set of wrong continuations is finite if the specification is finite as well. Figure 3 depicts the Petri net synthesized from the specification depicted in Figure 1 using compact tokenflow regions, an ILP synthesis algorithm, the concept of wrong continuations and deleting some implicit places as a last step.

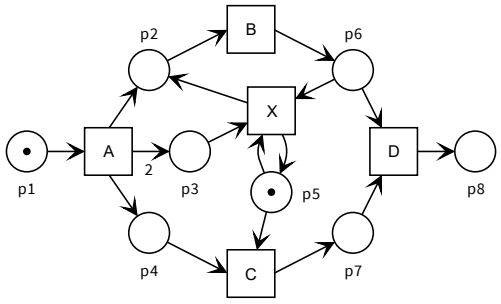

Figure 3: A Petri net.

Figure 4 depicts a transition system. A transition system is a set of states connected by a set of labeled arcs. Every arc models an event changing the state of the desired Petri net model. Partial languages can model concurrency, but they need to add extra runs whenever there is conflict. Transition systems can model conflict but are not able to model concurrency. Thus, depending on the specific application using one specific specification language can be more adequate than using the other.

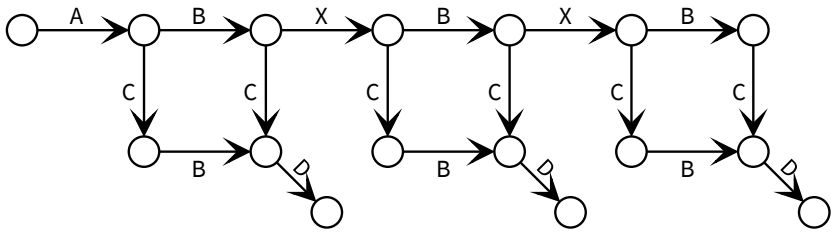

Figure 4: A transition system.

To synthesize a Petri net model from a transition system we use the concept of state-based regions. Such region is a multi-set of states so that every pair of equally labeled events has the same gradient. That means, every label is entering, not-crossing, or exiting the region with the same weight. Thus, every event has a constant effect on the marking of the synthesized result. We add the related places to construct a Petri net so that its reachability graph will be isomorphic to the input if such a net exists.

Figure 5 depicts three copies of the specification of Figure 4 and three different state-based regions. The first region is the set of grey states and defines the valid place p2 of Figure 3. The second region is twice the set of black states plus the grey states and defines the valid place p3. The third region is the

set of grey states and defines the valid place p5 without the short-loop to X. State-based regions usually do not generate short-loops because transition systems can not specify concurrency.

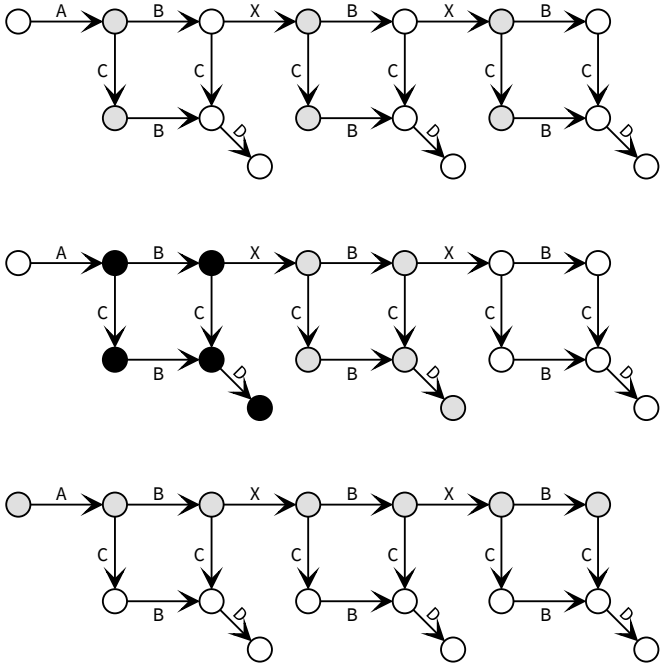

Figure 5: Three state-based regions.

Most state-based region synthesis algorithms construct a set of so-called minimal regions. They start by a set of minimal candidate regions and repair these regions by adding states until labels have a constant gradient to get a finite Petri net result.

3. Petri Net Regions

This section introduces Petri net regions. A Petri net region is a marking of a set of labeled Petri nets so that: (I) For every pair of equally labeled transitions, the difference between the sum of tokens in the post-set and the sum of tokens in the pre-set is the same. (II) For every pair of labeled Petri nets, the sum of tokens of all places with an empty preset is the same. We highlight the idea of this definition, by translating the specification of Figure 1 into three labeled Petri nets. Every event becomes a labeled transition, and we add a place to every relation. Figure 6 depicts three copies of the translated result, and we add markings related to the three different compact tokenflow regions of Figure 2.

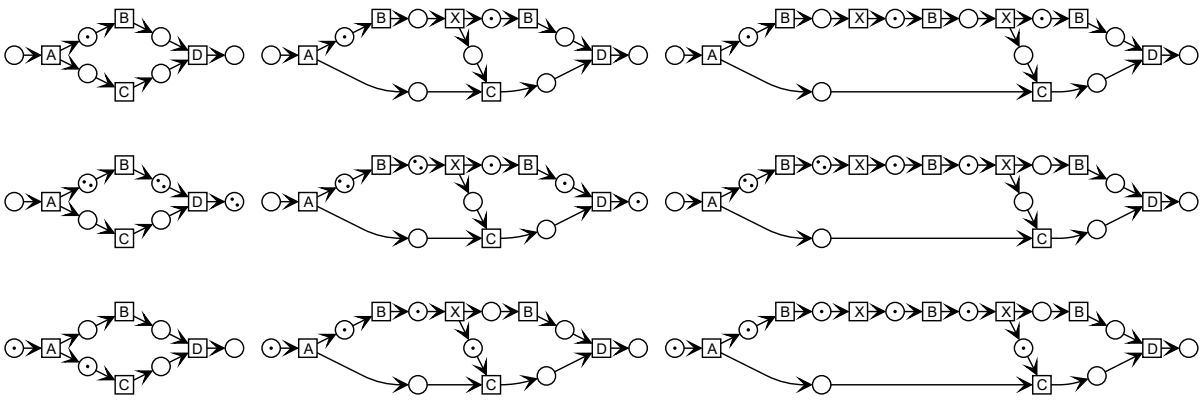

Figure 6: Three Petri net regions, equivalent to Figure 2.

It is easy to see that there is a one-to-one correspondence between a compact tokenflow region on the set of Hasse diagrams and a Petri net region on the labeled Petri nets. Thus, we propose that the following conjecture holds.

Conjecture 1. Let S be a set of labeled Petri nets and S be equivalent to a partial language L. There is a compact tokenflow region for L defining the valid place p iff there is a Petri net region for S defining the valid place p. We can solve the synthesis problem for partial languages using Petri net regions.

We translate the specification of Figure 4 into a labeled Petri net. Every state becomes a place, and every event becomes a labeled transition. Figure 7 depicts three copies of the translated result, and we add markings related to the three different state-based regions of Figure 5.

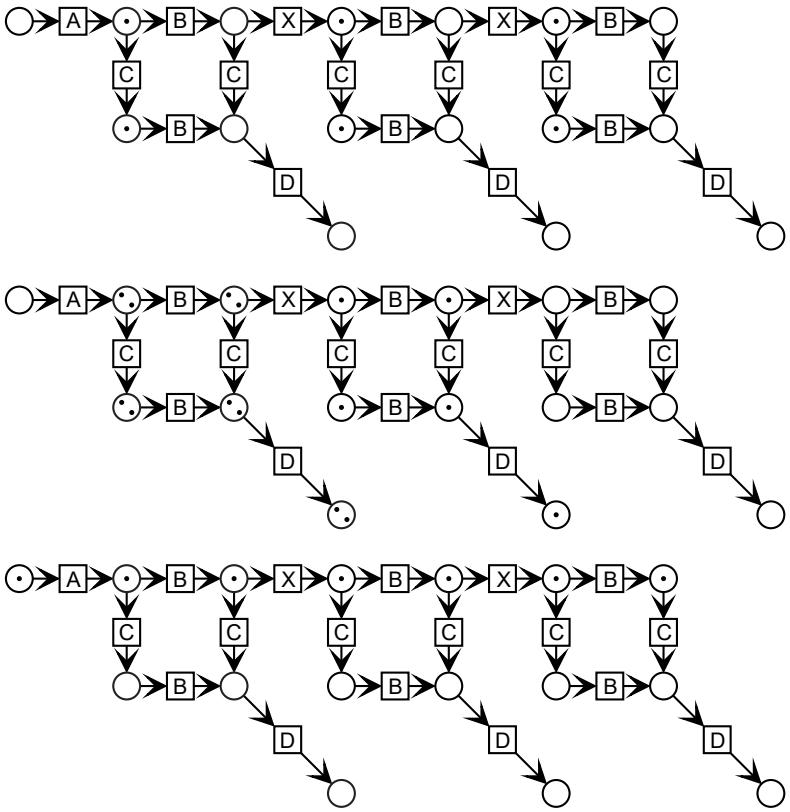

Figure 7: Three Petri net regions, equivalent to Figure 5.

Again, it is easy to see that there is a one-to-one correspondence between a state-based region on a transition system and a Petri net region on the labeled Petri net. Thus, we propose that the following conjecture holds as well.

Conjecture 2. Let S be a labeled Petri net and S be equivalent to a transition system L. There is a state-based region for L defining the valid place p iff there is a Petri net region for S defining the valid place p. We can solve the synthesis problem for transition systems using Petri net regions.

Conditions (I) and (II) are very intuitive and easy to check/implement. To get a running synthesis algorithm we can construct Petri net regions by repairing minimal candidate regions or by solving a related ILP. To get a finite result we can use the set of minimal regions, a basis of the ILP, or use the concept of wrong continuations. Petri net regions can handle state-based and language-based input. There is no need to distinguish the two concepts anymore.

Petri net regions are not only unifying previous region definitions but have very interesting application if the input is neither a transition system nor a partial language. Remark, Petri net regions can handle general labeled Petri nets as an input. Note that, we have to be careful with condition (A) of the synthesis problem in this part of the paper because of a missing semantic defining when a net is in the language of another. But we will get the idea ;)

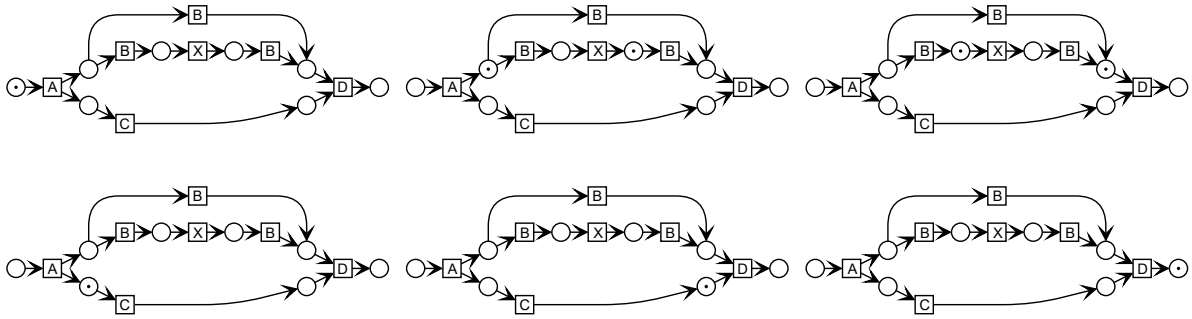

Figure 8: Six minimal Petri net regions.

Figure 8 depicts six copies of a labeled Petri net, and we use this Petri net as the specification. This Petri net is neither a transition system nor a partial language. Every depicted marking is a minimal Petri net region. The related synthesis result is depicted in Figure 9. Specifying a shared place after no and one iteration of the loop started by X, leads to a result where counting the number of loops is not possible. Thus, Figure 9 is the intended result.

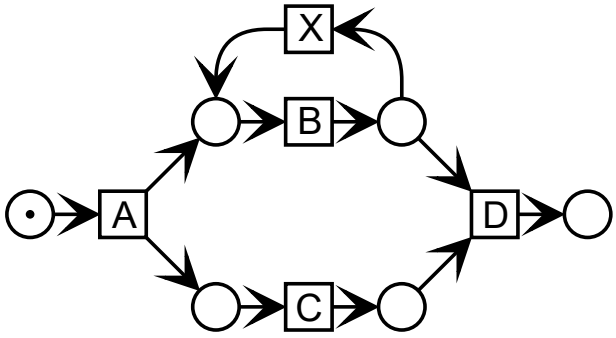

Figure 9: Synthesis result using minimal Petri net regions.

The left part of Figure 10 depicts another labeled Petri net where the state after the occurrence of the loop is not shared. The depicted marking is a minimal Petri net region. Taking the left-hand side as an input, the right-hand side of Figure 10 depicts the synthesis result. This time, the occurrence of X and the occurrence of B is restricted by the additional place.

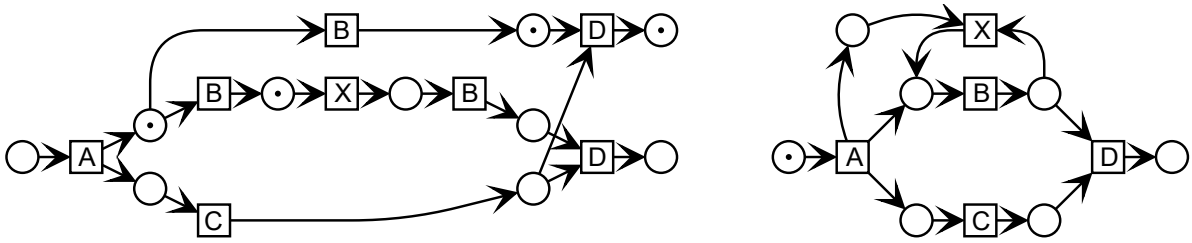

Figure 10: Petri net region for a branching process.

4. Synthesis Algorithm

In this section, we use the new concept of Petri net regions to implement a synthesis algorithm. Using regions, the region definition ensures that the output of every synthesis can execute the input. But if we implement a full-on synthesis algorithm, we must choose (a) the net class of the output, (b) the kind of regions we calculate, and (c) how regions are calculated. For the net class we can calculate nets with or without arc-weights or short-loops. We can calculate regions that are one-bounded and have at most one token in every place. To get a finite result, we can calculate a basis, a set of regions related to wrong continuations, or a set of minimal regions. Furthermore, we can construct regions by simulation, construction, or by implementing an ILP.

In this paper, to highlight the power of Petri net regions, we implement a synthesis algorithm using the new concept. The input to this algorithm is a set of labeled Petri nets. The algorithm uses an ILP to enforce conditions (I) and (II) of the Petri net region definition. Calculating the set of minimal Petri net regions, the algorithm will synthesize a Petri net without short-loops. We can toggle if regions will be one-bound or not. The synthesis algorithm is available at the I ♥ Petri Nets web toolkit (https://www.fernuni-hagen.de/ilovepetrinets/).

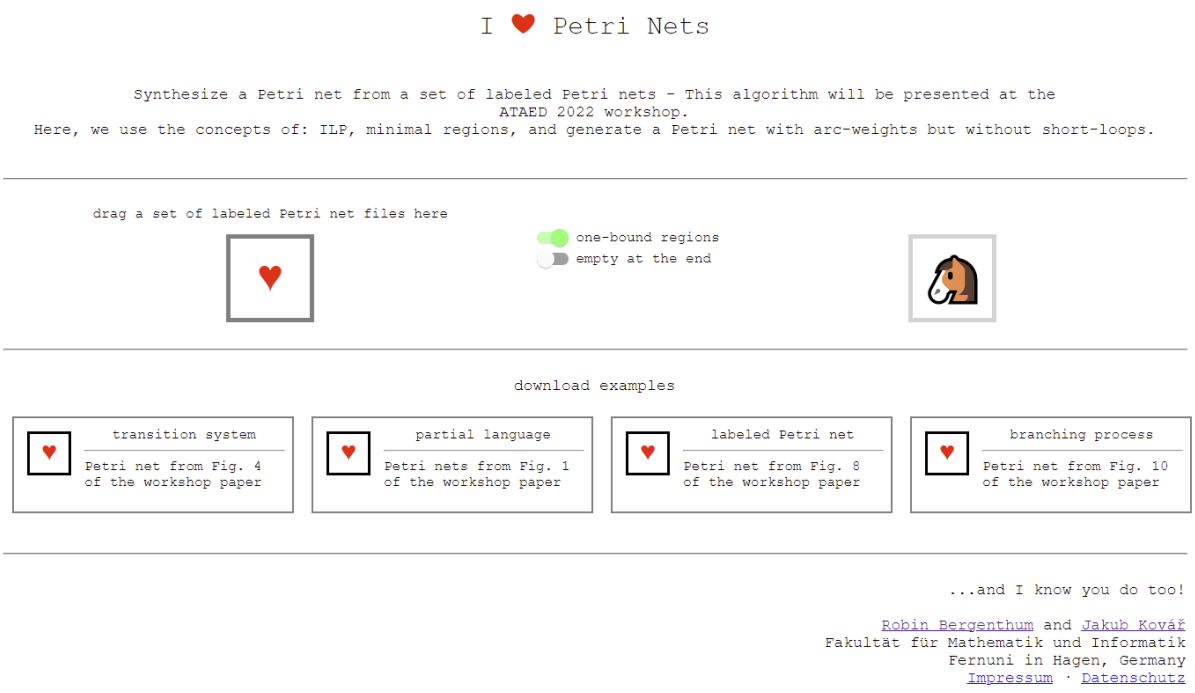

Figure 11: Synthesis using Petri net regions in the I ♥ Petri Nets web toolkit.

Figure 11 depicts a screenshot of the synthesis algorithm. Input to the algorithm is a set of text-files describing labeled Petri nets. We can use any text-file editor to specify the input, use an editor from the I ♥ Petri Nets web toolkit, or simply download example nets from the synthesis web page. Synthesis starts as we drag-and-drop files to the big ♥ button. As soon as the synthesis algorithm is done, we can download the synthesized result via the 🐴 button. We can read the files using a text-file editor or use the "show a labeled Petri net" tool from the toolkit.

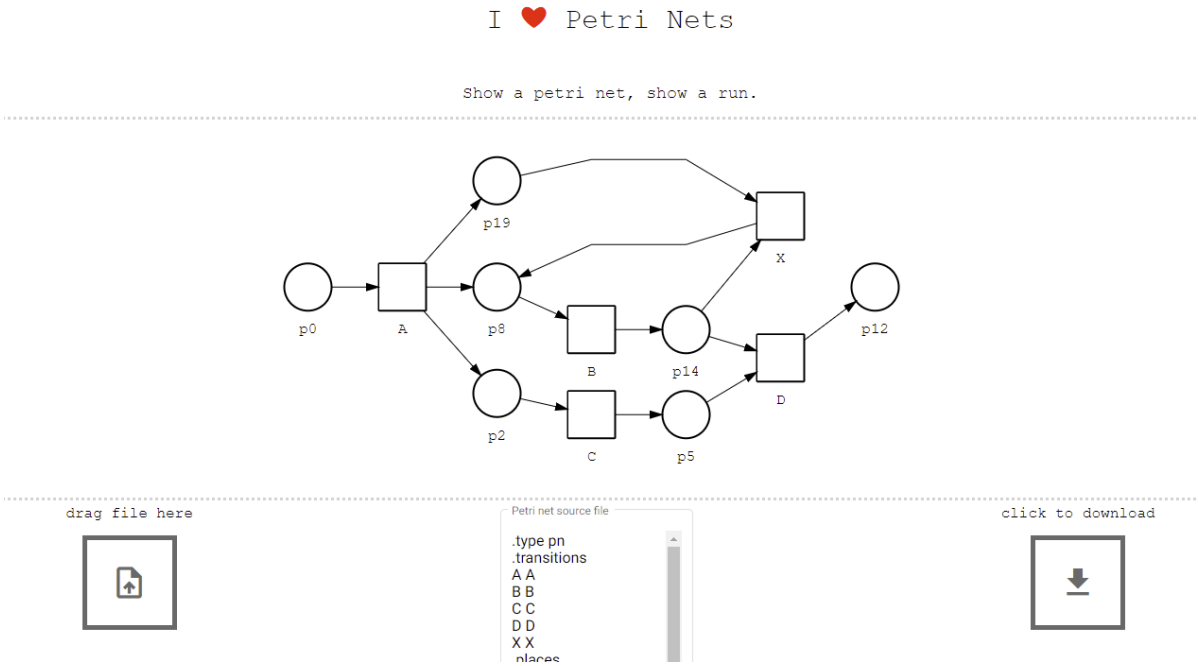

Figure 12: Displaying a labeled Petri net in the I ♥ Petri Nets web toolkit.

Figure 12 depicts the synthesis result using the example net of Figure 10 as an input for the synthesis algorithm calculating one-safe regions displayed by the show a labeled Petri net editor of the web toolkit.

5. Conclusion

Petri net regions unify state-based and language-based region theory. They can handle transition systems, languages, partial languages, branching processes (see Figure 10), and general labeled Petri nets (see Figure 9). We present a synthesis algorithm using Petri net regions in the I ♥ Petri Nets web toolkit. Here, we use an ILP to generate a set of minimal Petri net regions and calculate results without short-loops. Obviously, this workshop paper only presents a first glance at Petri net regions, the formal definitions and proofs must be submitted in future work.

References

[1] Peterson, J. L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall, 1981.
[2] Desel, J.; Juhas, G.: “What is a Petri Net?”. Advances in Petri Nets, LNCS 2128, Springer, 2001.
[3] van der Aalst, W. M. P.; van Dongen, B. F.: Discovering Petri Nets from Event Logs. ToPNoC VII, LNCS 7480, Springer, 2013, 372–422.
[4] Reisig, W.: Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies. Springer, 2013.
[5] Mayr, H. C.; Kop, C.; Esberger, D.: Business Process Modeling and Requirements Modeling. ICDS 2007, Computer Society, IEEE, 2007, 8—14.
[6] Ehrenfeucht, A.; Rozenberg, G.: Partial (Set) 2-Structures. Part I: Basic Notions and the Representation Problem, Part II: State Spaces of Concurrent Systems. Acta Inf. 27(4), 1990.
[7] Badouel, E.; Bernardinello, L.; Darondeau, P.: Petri Net Synthesis. Texts in Theoretical Computer Science, Springer, 2015.

[8] van Dongen, B. F.; de Medeiros, A.; Verbeek, H. M. W.; Weijters, A. J. M. M.; van der Aalst, W. M. P.: The ProM Framework: A New Era in Process Mining Tool Support. Petri Nets 2005, LNCS 3536, Springer, 2005, 381–390.
[9] Carmona, J.; Cortadella, J.; Kishinevsky, M.: Genet: a Tool for the Synthesis and Mining of Petri Nets. Application of Concurrency to System Design 2009, 2009, 181–185.
[10] Borde, D., Dierkes, S., Ferrari, R., Gieseking, M., Göbel, V., Grunwald, R., von der Linde, B., Lückehe, D., Schlachter, U., Schierholz, C., Schwammberger, M., Spreckels, V.: APT: analysis of Petri nets and labeled transition systems. https://github.com/CvO-Theory/apt.
[11] Bergenthum, R.; Desel, J.; Juhas, G.; Lorenz, R.: Synthesis of Petri Nets from Scenarios with VipTool. Petri Nets 2008, LNCS 5062, Springer, 2008, 388–398.
[12] Bernardinello, L.; De Michelis. G.; Petruni, K.; Vigna, S.: On The Synchronic Structure of Transition Systems. Structures in Concurrency Theory, Springer, 1995, 69–84.
[13] Bergenthum, R.; Desel, J.; Lorenz, R.; Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. Fundamenta Informaticae 88, IOS Press, 2008, 437–468.
[14] Bergenthum, R.: Synthesizing Petri Nets from Hasse Diagrams. Proceedings of Business Process Management 2017, LNCS 10445, Springer, 2017, 22-39.
[15] Bergenthum, R.; Desel, J.; Mauser, S.: Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language. ToPNoC III, LNCS 5800, Springer, 2009, 216–243.
[16] van der Werf, J.M.E.M.; van Dongen, B.F.; Hurkens, C.A.J.; Serebrenik, A.: Process Discovery Using Integer Linear Programming. Proceedings of PETRI NETS 2008, LNCS 5062, Springer, 2008.
[17] Carmona, J.; Cortadella, J.; Kishinevsky, M.: A Region-Based Algorithm for Discovering Petri Nets from Event Logs. Proceedings of BPM 2008. LNCS 5240, Springer, 2008.