Synthesis of Petri Nets from Infinite Partial Languages with VipTool

Robin Bergenthum and Sebastian Mauser

Catholic University of Eichstätt-Ingolstadt, Germany
prename.name@ku-eichstaett.de

Abstract. In this paper we show an implementation of an algorithm to synthesize a place/transition Petri net (p/t-net) from a possibly infinite partial language, which is given by a term over a finite set of labelled partial orders (LPOs).
The implementation is integrated as an extension of our Petri net toolset VipTool. The new extension comprises two plug-ins. The first plug-in offers editing features to specify term based partial languages. The specification of terms is graphically supported by a visualization similar to structograms as well as by a representation in the form of UML activity diagrams. The second new plug-in provides the algorithmic computation of a p/t-net synthesized from a term specification. This algorithm is in principle based on ideas already presented in the paper [4], but a so called separation computation is applied instead of the basis representation used in [4].

1 Introduction

Synthesis in the field of Petri net theory means algorithmic construction of a Petri net satisfying a given behavioural description. Often labelled partial orders (LPOs) are considered the most natural model to describe the non-sequential behaviour of Petri nets. An LPO represents a run of a place/transition Petri net (p/t-net) if it is enabled w.r.t the net in the sense that the events of the LPO modelling transition occurrences can fire in the net respecting the concurrency and dependency relations given by the LPO. In [4] we presented an algorithm to synthesize a finite unlabelled p/t-net from a partial language, i.e. a set of LPOs, which is given in a term based representation. It was shown that the set of runs of the synthesized net coincides with the set of LPOs represented by the term – if such a net exists. A term of LPOs is built from a finite alphabet of LPOs and composition operators for union, parallel composition, sequential composition and iteration. Due to the iteration operator the partial language represented by such term may be infinite. The synthesis approach in [4] is based on the theory of regions. Each transition of the synthesized net is given by a label appearing in the term, and the places of the net are computed by so called token flow regions. Since the set of all regions is infinite, the synthesis algorithm calculates a finite set of so called basis regions.

It was shown in [3] that such basis representation is often inappropriate for practical applications, because the set of places corresponding to basis regions is usually very large causing unreadable nets. Therefore, in this paper we change the synthesis algorithm from [4] by computing so called separating regions, which proved to yield good synthesis results [3]. Moreover we use the region type called transition regions instead of the token flow regions considered in [4]. But this is only a design decision, because we showed in [3] that both region types have advantages in different settings. While exchanging the region type of token flow regions by transition regions only requires minor changes of the synthesis algorithm, it is difficult to use the principle of computing a separating representation of all regions instead of a basis representation. The problem in particular lies in the infinity of the considered partial languages. Tackling this problem to get a synthesis algorithm from terms of LPOs which uses separating regions is the main theoretical contribution of this paper.

On the practical side we implemented this algorithm as an extension of our toolset VipTool [1] (viptool.ku-eichstaett.de). VipTool offers a user-friendly framework for integrating plug-ins. The focus of VipTool is on partial order behaviour of Petri nets. In particular, VipTool already offers functionalities for editing, visualizing and storing p/t-nets and LPOs. The new extension comprises a plug-in to specify terms of LPOs. It allows to compose stored LPOs by respective composition operators. The composition of LPOs is graphically supported by a visualization of the term in a style analogous to structograms. This representation of such terms was already suggested in [2] as a nice possibility to compose LPOs. The plug-in also offers an alternative visualization of terms of LPOs in the form of UML activity diagrams which might be more intuitive for some users. The second plug-in is an implementation of the actual synthesis algorithm developed in this paper. It computes a p/t-net from a term of LPOs which can then be layouted and displayed by VipTool.

2 Synthesis Algorithm

In this section the new synthesis algorithm is described. Due to lack of space, we omit formal details here, and try to give a high-level explanation of the algorithm. We provide a toy example concerning a workflow of processing a claim in an insurance company.

The input to the synthesis algorithm is a term over a finite alphabet of LPOs serving as building components. Terms are constructed inductively by iteration (*), parallel composition (||), sequential composition (;) and union (+). A term defines a possibly infinite set of LPOs, called the partial language of the term, by combining the LPO components according to the composition operators. For example the infinite partial language of the term *Registration; (((PosEvaluation; ((Queries)*); Payment) + NegEvaluation)* || *Reserves)* build of the LPOs given in Figure 1 is illustrated in Figure 2. For formal definitions see [4].

The aim now is to compute a p/t-net, such that the set of enabled LPOs of the net coincides with the partial language of the specified term (or more precisely with its prefix- and sequentialization closure) - if such net exists. The synthesis algorithm proposed in this paper follows standard techniques known from the theory of regions. It roughly works as follows: The set of transitions of the synthesized net is the finite set of labels appearing in the term. One restricts the behaviour of this net by creating causal dependencies between the transitions through addition of places. Places are defined by their initial marking and the arc weights connecting them to transitions. Two kinds of places can be distinguished. In the case that there is an LPO in the partial language of the term which is not enabled in the net which has only the one considered place, this place restricts the behaviour too much. Such places are non-feasible. In the other

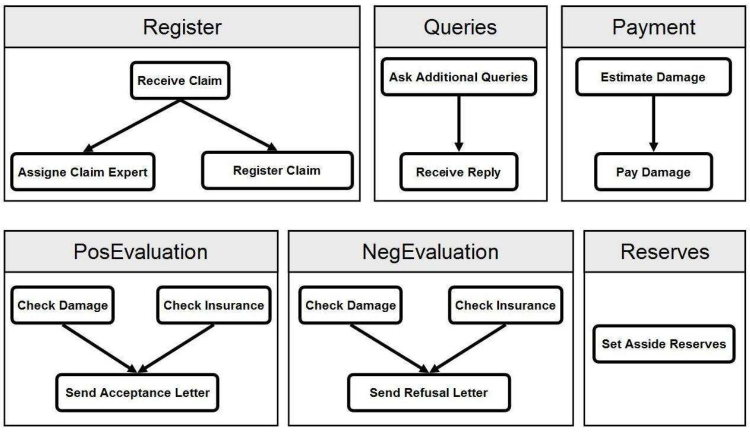
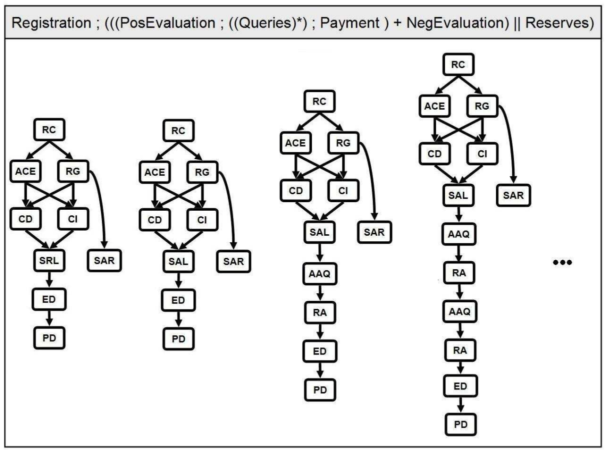


Fig. 1. Example LPOs modelling parts of a claim handling process in an insurance company.



Registration ; (((PosEvaluation ; ((Queries)*) ; Payment) + NegEvaluation) || Reserves)
RC
ACE
RG
CD
CI
SRL
SAL
SAR
AAQ
RA
ED
PD
•••

Fig. 2. Example term over the LPOs given in Figure 1 and the corresponding infinte partial language (transition names are abbreviated).

case, the considered place is feasible. The set of feasible places corresponds to the set of regions of the term. The definition of regions considered in this paper is similar to the one developed in [4]. But the idea of considering token flows is replaced by using so called transition regions (compare [3]). Since in general there are infinitely many regions, in [4] a finite set of so called basis regions representing all feasible places is computed. The approach in this paper is different. The aim is to iteratively compute feasible places prohibiting LPOs not in the language of the term from being enabled, such that after a finite number of steps all behaviour not given by the term is not any more possible in the calculated net. The basic idea is to append events to LPOs given by the language of the term such that the resulting behaviour, called a wrong continuation of the term, is not specified by the term. Then for each wrong continuation a region is computed such that the corresponding feasible place separates the wrong continuation (compare [3]), if such a place exists. In the positive case the place is added to the net and the next wrong continuation is considered, in the negative case it is directly proceeded with the next wrong continuation. This solves the synthesis problem, because the finite set of wrong continuations is defined in such a way that, if all wrong continuations are separated, then the infinite set of all LPOs not in the language of the term is prohibited, or it is not possible to prohibit all such behaviour. The problem is first to define the notion of regions of a term appropriately and second to define wrong continuations of a term in such a way that the set of wrong continuations is finite, although the language of a term may be infinite.

The idea in [4] to define regions of a term is to consider a finite set of representation-LPOs R and a finite set of iteration-LPOs I (see Figure 3 for an example). The region definition ensures that the LPOs in R are enabled, and that the LPOs in I can be iterated, which means that if such LPO is enabled in a marking then it is again enabled after having fired all events of the LPO. Consequently the LPOs in R are enabled w.r.t. a place corresponding to a region (i.e. the place is feasible w.r.t. R) and the LPOs in I produce at least as many tokens in a place corresponding to a region as they consume. It is shown in [4] that a place is feasible w.r.t. the language of a term if and only if it satisfies these constraints. In this paper we consider regions of a term by applying these constraints to transition regions. Then similar as in [4] the set of feasible places of the language of a term corresponds to the set of regions of the term where the set of regions is given by the integer solutions of an inequality system.

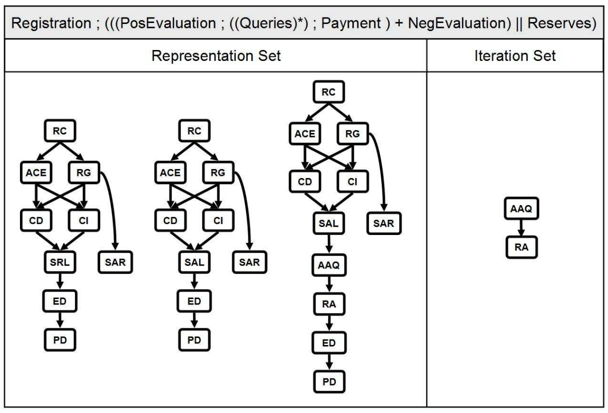


Fig. 3. Example term of Figure 2 and the corresponding finite representation and iteration sets.

Having defined regions we have to tackle the problem of defining wrong continuations. The basic idea is to use the notion of wrong continuations of a finite partial language developed in [3] on the finite set R. Note that due to iteration it may happen that a wrong continuation of an R-LPO is an element of the partial language of the term, but this does not matter, because in this case there is of course no feasible place prohibiting this wrong continuation and thus the step is skipped. The aim is that if we separate all wrong continuations of R, then also all wrong continuations of LPOs that can be generated by iterating the I-part of an R-LPO are separated. More precisely, a feasible place separating a wrong continuation of an R-LPO should also separate the respective wrong continuation of an LPO arising from this R-LPO through iteration of I-LPOs. But we found examples showing that this is not automatically guaranteed, because separating places may be filled with tokens by iterating I-LPOs, such that these places do not any more separate behaviour after enough iterations. Therefore, we need some constraints ensuring that iterating certain I-LPOs does not pump up the tokens in a separating place, i.e. such I-LPOs are not allowed to produce more tokens in the place than they consume from the place. But there are examples showing that we cannot introduce such restriction for each I-LPO and each separating place, because sometimes only an unsafe place can prohibit a certain wrong continuation. The solution is to identify for each feasible place separating a wrong continuation of an R-LPO an appropriate subset of I-LPOs, for which we consider a ”non-pump-up” constraint. This subset is roughly speaking given by the I-LPOs that can occur before the wrong continuation within the R-LPO, because each such I-LPO could otherwise pump up the place so that the wrong continuation is not separated after a certain number of iterations of the I-LPO.

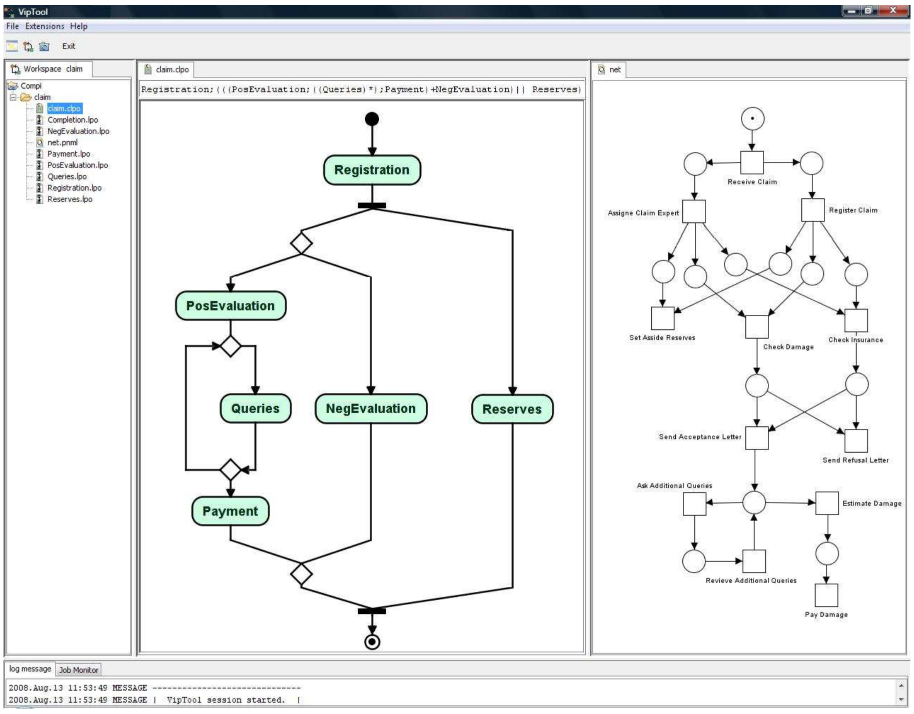
The algorithm described semi-formally in this section synthesizes a p/t-net from a term of LPOs. For the term of Figure 2 the resulting net is shown on the right part of Figure 4. Without providing a formal proof in this workshop paper, we claim that if there is a p/t-net having the partial language of the given term as its set of enabled LPOs, then the computed net is such net. The decidability if the computed net actually has the given behaviour is an open problem (see also [4]).

3 Implementation

We implemented the algorithm presented in the last section as an extension of our toolset VipTool [1]. VipTool was originally designed as a tool for modelling, simulation, validation and verification of business processes using (partial order behaviour of) Petri nets. It now offers a flexible xml-based open plug-in architecture to integrate methods concerned with causality and concurrency modelled by partially ordered runs of Petri nets. In particular, it already provides plug-ins to synthesize p/t-nets from finite partial languages. The new extension to synthesize p/t-nets from terms of LPOs now offers two important advancements: On the one hand it is for the first time possible to compute nets from infinite sets of LPOs (finitely represented by terms), and on the other hand terms allow a modular specification of the input partial language. The support of modularity and the possibility to specify infinite behaviour are two innovative steps towards practical applicability of our synthesis framework in industrial settings, where our focus remains on modelling of business processes.

The new synthesis extension can nicely build on VipTool's existing plug-ins for editing, visualizing and storing p/t-nets and LPOs. A first new plug-in of VipTool supports the design of terms of LPOs. Using LPOs drawn with VipTool and stored as xml-files, the plug-in provides an editor to specify terms over such LPOs. The composition of single LPOs to terms of LPOs by using the provided composition operators is supported by two graphical views. The first visualization illustrates the inductive composition of the LPOs in a natural way by building a block structure illustrating the composition operators between the blocks (as shown in [2]). This is very similar to the visualization of algorithms by structograms. The second visualization translates the term structure into an UML activity diagram (see Figure 4). The alphabet of single LPOs underlying the term form the activities of the diagram. In this sense, they have to be interpreted as abstract activities that are refined by a behavioural description through the respective LPOs. The sequential composition operator determines the path dependencies between the LPO-activities. The parallel and alternative composition operators yield balanced parallel and alternative splits and joins. The iteration operator is implemented as a respective cyclic structure. Having designed a specification of partial order behaviour of Petri nets by a term, a description of the term of LPOs is stored in an xml-file.

A second plug-in loads such xml-file and xml-files of the LPOs occurring in the term. It computes a p/t-net from the term specification by applying the described algorithm. The resulting net is stored as a pnml-file, which can be loaded, visualized, layouted and analyzed by already existing plug-ins of VipTool. Figure 4 shows that in our running example the computed net nicely models the workflow specified by the considered term.



```
Registration;(((PosEvaluation;((Queries)*);Payment)+NegEvaluation)|| Reserves)
```

```
2008.Aug.13 11:53:49 MESSAGE -----------------------------
2008.Aug.13 11:53:49 MESSAGE |  VipTool session started.  |
```

Fig. 4. Screenshoot of VipTool showing the example term of Figure 2 represented as an activity diagram and the net resulting from the synthesis algorithm.

References

1. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri Nets from Scenarios with Viptool. In *ICATPN 2008, LNCS 5062*, pages 388–398.
2. R. Bergenthum, J. Desel, and S. Mauser. Synthesis of Petri Nets for Business Process Design. In *Proceedings of workshop Verhaltensmodellierung: Best Practices und neue Erkenntnisse @Modellierung 2008*.
3. R. Bergenthum and S. Mauser. Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language. In *Proceedings of workshop CHINA @ICATPN 2008*.
4. R. Lorenz, R. Bergenthum, S. Mauser, and J. Desel. Synthesis of Petri Nets from Infinite Partial Languages. In *Proceedings of ACSD 2008, IEEE*.