Faster Unfolding of General Petri Nets

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

We propose two new unfolding semantics for general Petri nets based on the concept of prime event structures. The definitions of the two unfolding models are motivated by algorithmic aspects. We develop a construction algorithm for both unfolding models. The unfolding models employ the idea of token flows developed in [JLD05] and are much smaller than the standard unfolding model. We show that they still represent the complete partial order behaviour of the given net. Since the models are smaller, they can be constructed much faster.

1 Introduction

Non-sequential Petri net semantics can be coarsely classified into unfolding semantics, process-oriented semantics and algebraic semantics [MMS94]. While the latter is not widely known and process models do not provide semantics of a net as a whole, but specify only single, deterministic computations, unfolding models are a popular approach to describe the complete behaviour of nets and thus account for the fine interplay between concurrency and nondeterminism. To study the behaviour of Petri nets primarily two models for unfolding semantics were retained: labelled occurrence nets and event structures. In this work we focus on algorithmic aspects of unfolding semantics. We propose two new event structure-based unfoldings of general Petri nets and compare them to the existing unfolding semantics. For both new unfolding models, we sketch a construction algorithm and argue, that they can be constructed in many cases very efficient compared to the standard unfolding approach.

The standard unfolding semantics for general Petri nets (p/t-nets) was developed in [BD87, MMS97, MMS94]. This approach generalizes the unfolding semantics originally introduced in [NPW81] for safe nets, and extended in [Eng91] to initially one marked p/t-nets without arc weights by viewing the tokens as individualized "coloured" entities. In contrast to [Eng91], the unfolding approach for p/t-nets even individualizes tokens having the same "history". This guarantees that one can analogously as in [Eng91] define a single occurrence net (or more precisely a branching process), called the unfolding of the system, capturing the complete behaviour of the p/t-net (see Figure 2). The unfolding construction is well analyzed and several algorithmic improvements are proposed in literature. By restricting the relations of causality and conflict of the resulting occurrence net to events, one obtains a labelled prime event structure (with binary conflict relation) representing the causality of the unfolding (see Figure 3). This labelled prime event structure modelling the behaviour of a p/t-net is called PES0 in the following. In figures of prime event structures, e.g. Figure 3, the colouring of the events illustrates the sets of consistent events, i.e. events not being in conflict. A set of consistent events of a prime events structure defines a partially ordered *run*.

As explained in [HKT96, MMS94, CPW04] this standard unfolding approach for p/t-nets, in particular the concept of coloured tokens, yields several problems. Consequently there have been many attempts to define alternative unfolding semantics for p/t-nets: using so called local event structures [HKT96] instead of prime event structures, allowing arbitrarily valued and non-safe occurrence nets [CPW04], grouping conditions into families [MMS97] (decorated occurrence nets), translating general nets into safe nets by introducing places for reachable markings [Haa98] or

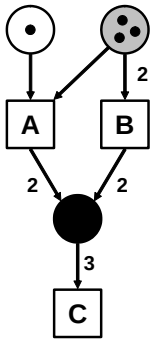

Figure 1: Example net N.

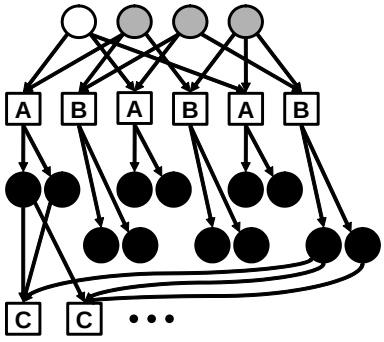

Figure 2: Standard unfolding of N.

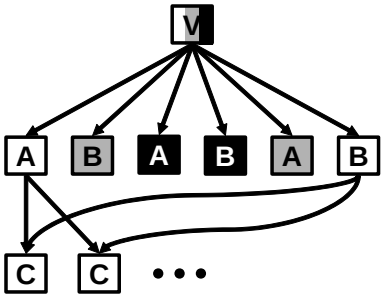

Figure 3: Prime event structure PES0 of N.

considering a swapping equivalence to introduce a collective token view [BD87]. All of these works focus on algebraic and order-theoretic properties, but the approaches are not to of algorithmic interest.

In this paper we are interested in algorithms to efficiently construct unfoldings. With regard to this, the standard unfolding procedure for p/t-nets has the drawback, that the individualized tokens cause multiple conditions referring to the same place in the net and having the same pre-event (or being initial events), e.g. the three initial grey conditions in Figure 2. This artificially blows up the size of the unfolding and the size of PES0. The individualized tokens in the worst case increases the number of events exponentially for every step of depth of the unfolding. For example one has to regard all three possibilities of the transition b to consume two of the three individualized tokens in the grey place (compare the three b events in Figure 2). That means the individualized tokens (condition) cause PES0 to contain multiple instances of so-called *identical* events, which have the same label, the same pre-events (pre-events are events from which the considered event is causally dependant in a prime event structure) and are in conflict (e.g. the three a- or b-events in Figure 3 are identical, also the two c-events in Figure 4).[1] In other words two events in PES0 are identical, if the pre-sets (of conditions) of the two events in the respective unfolding have the same pre-set of events and share at least one condition. Two cases of identical events can be distinguished. First the two presets of the identical events in the unfolding can contain for each place label the same number of conditions having this label and a common pre-event (see the identical a-events in Figure 2 respectively 3). This phenomenon of generating identical events we abbreviate as (IDENT1). But it is also possible, that there are identical events in PES0, whose presets in the unfolding contain for some place label a different number of conditions having this label and a common pre-event. For example one of the two depicted c-events in Figure 2 has two black pre-conditions generated by the most left a-event and one black pre-condition generated by the most right b-event, while the other depicted c-event has two black pre-conditions generated by the most right b-event and one black pre-condition generated by the most left a-event. This phenomenon is called (IDENT2). The described problem (IDENT1) causes the standard unfolding to be an inefficient representation of the set of Goltz-Reisig processes [GR83] of the given p/t-net. The unfolding includes several copies of the same process, because each (IDENT1)-identical event introduces additional isomorphic processes (e.g. in Figure 2, all three conflict-free combinations of a and b events yield isomorphic processes). Moreover the calculation of the processes from the unfolding is inefficient. In contrast to (IDENT1)-identical events, (IDENT2)-identical events may (usually but not always) constitute different Goltz-Reisig processes (the two identical c-events in Figure 4 define two different processes). But for a compact unfolding representation of the net behaviour, it is also not necessary to consider these multiple instances of (IDENT2)-identical events, because such events produce isomorphic partially ordered runs.

Altogether individualizing tokens causes the standard unfolding and the associated labelled prime event structure PES0 to become unnecessary large. PES0 contains huge numbers of identical events, although identical events are never necessary to capture the causal behaviour of a system by a labelled prime event structure. Unfolding models ensuring less nodes could significantly decrease the construction time, because a construction algorithm in some way always has to test all co-sets of events or conditions of the so-far constructed model to append further events. For example, in each algorithm to construct the standard unfolding and PES0, all co-sets of constructed conditions have to be checked (e.g. in [DJL03]). Since the number of conditions in the unfolding is linearly dependant on the number of events, reducing the number of events could significantly improve the runtime of a construction algorithm. In this paper we propose two unfolding approaches reducing the number of events in contrast to the standard unfolding by neglecting identical events.

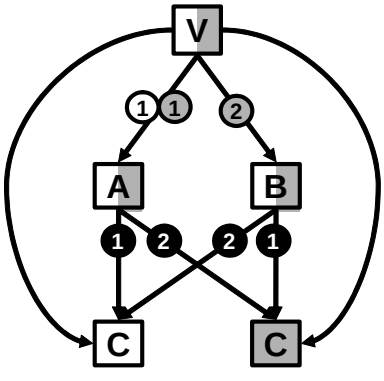

Figure 4: Prime event structure PES1 of N.

We show two methods to directly unfold a p/t-net to a labelled prime event structure using the concept of *token flow* presented in [JLD05]. Token flows were developed for a compact representation of process nets. Namely, token flows abstract from the individuality of conditions of the process net and encode the flow relation of the process by natural numbers. For each place a natural number is assigned to the edges of the partial order relation between events defined by the process net. Such a natural number assigned to an edge (v, v') represents the number of tokens produced by the transition occurrence v and consumed by the transition occurrence v' in the respective place. This idea is generalized to unfoldings in this paper. Since we abstract from the individuality of conditions, both presented unfolding methods generate event structures having significantly less events than PES0. To append events in a construction algorithm generating directly an event structure, co-sets of events or more precisely prefixes (defined by a co-set) of consistent events have to be checked. Minimizing the number of events, the two proposed approaches are candidates to yield fast construction algorithms.

The first method constructs a labelled prime event structure enriched with the full token flow information from a p/t-net. The resulting event structure, called PES1 in the following, basically coincides with PES0, except (IDENT1) being resolved, i.e. respective identical events are neglected (compare Figure 3 and 4). That means the number of

[1] Note that multiple instances of concurrent events with the same label and the same pre-events are necessary to model the causalities of the net, but identical events are not necessary.

events is usually a lot smaller, while the token flow information is preserved. Note here that omitting identical events disables the possibility of applying prime event structures with a binary conflict relation. We use the general prime event structures with a consistency set as introduced in [Win86] (in a slightly simplified version). That means for each process in the unfolding we have to store a set of respective events in a so called consistency set. Because the number of processes represented by an unfolding might be exponential, this could lead to some performance problems of the construction algorithm. On the one hand it could lead to a higher memory consumption for saving the consistency set, although the number of stored events is decreased. On the other hand this also leads to some minor runtime problems explained later on, so that it is not clear in advance, in which cases the method is faster than the standard unfolding method. It remains to mention that the token flow information of PES1 guarantees that the set of Goltz-Reisig processes of the p/t-net is still completely reflected in PES1, but the number of isomorphic processes is a lot smaller than in the standard unfolding. Isomorphic processes occur in PES1 only in specific auto-concurrency situations. The set of processes can directly be deduced from PES1 by the consistency sets. We implemented an algorithm to construct PES1.

The second method constructs a labelled prime event structure PES2 from a p/t-net, which additionally to PES1 resolves (IDENT2), i.e. respective identical events are also neglected (the two (IDENT2)-identical c-events in Figure 4 do not occur in Figure 5). That means PES2 contains no two identical events at all. Therefore it is the smallest labelled prime event structure capturing the complete behaviour of a p/t-net. Notice, that the token flow information of PES2 is not any more "complete" (in contrast to PES1). That means not each possible token flow distribution is represented in PES2. Instead "example token flows" are stored for each partially ordered run. Consequently, while the construction algorithm deals with smaller event structures and thus less possibilities to append further events, the procedure of appending one event is more complicated, because the token flow eventually has to be recalculated. There is an efficient method for this recalculation based on the ideas in [JLD05]. Finally, note that the set of Goltz-Reisig processes is of course not completely represented in PES2, since there are different processes generating the same run, and every run exists exactly once in PES2. For each run associated to some process, one example process generating the run is given by a respective example token flow in PES2. We did not implement a construction algorithm for PES2 so far.

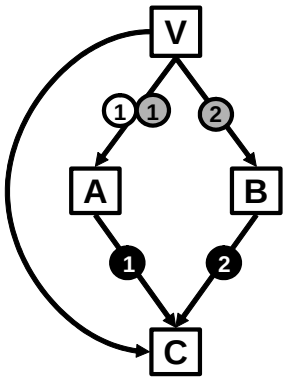

Figure 5: Prime event structure PES2 of N.

Lastly the question for the practical usefulness of the two event structures PES1 and PES2 has to be answered. First they can in many cases be constructed more efficient than the standard unfolding. A comparison of runtime and size of constructing PES0 and PES1 follows in Section 4. In particular the new methods may lead to a more efficient computation of the set of processes of a p/t-net. Furthermore they can still be applied almost analogously as the standard unfolding and PES0: they form a compact representation of the complete behaviour of a p/t-net accounting at the same time for concurrency and nondeterminism and they offer a possibility to calculate and represent the set of Goltz-Reisig processes. Most of the model checking algorithms working on standard unfoldings and PES0 can also be adapted to PES1 and PES2. The two event structures PES1 and PES2 are even more compact than the standard unfolding. Consequently applying PES1 and PES2 could further improve the efficiency of such automatic verification methods. In this context it is also important that the concepts of creating a finite complete prefix of the standard unfolding of a bounded p/t-net using adequate partial orders and cut-off events can analogously be transferred to PES1 and PES2. Having this in mind we do not discuss these application topics in this paper any further, but concentrate only on the runtime of construction algorithms.

The two prime event structure unfolding methods will be explained and discussed in detail in the following two sections. In particular they will be compared with each other and with the standard unfolding method on a conceptual level. We omit technical definitions or proofs in this workshop paper. In Section 4 we finally compare the runtime of our implementation of a construction algorithm for PES1 and of the standard unfolding algorithm used in [DJL03] with experimental results.

2 Algorithm 1

PES1: The first method constructs a general labelled prime event structure PES1 enriched with fixed token flow tuples for every edge of the structure (see Figure 4). The consistency set of the prime event structure stores sets of events forming a run (or process) of the p/t-net. The algorithm will only consider so called left-closed sets of consistent events (forming prefixes of the prime event structure), since only such sets represent runs. Moreover only maximal (w.r.t. set inclusion) runs are stored having in mind that prefixes of runs are also runs. Within one run, i.e. one set of consistent events, the ordering relation of the prime event structure models the causal dependencies of the events. That means ordered events of one run necessarily occur in the timeline of the respective computation in this ordering, i.e. the first event occurs before the second one, while unordered events of one run occur concurrently in the respective computation. The token flow tuples assigned to the edges modelling the causal dependency relation of the

prime event structure have one non-negative integer entry for every place in the given p/t-net (compare Figure 4). These values represent the numbers of tokens produced by the event at the beginning of the edge and consumed by the event at the end of the edge in the respective place. Such flow of tokens is responsible for the causal dependencies of events in a p/t-net. Thus such tuple assigned to an edge of the skeleton of the event structure is never the zero-tuple, since a dependency between events always comes with a non-zero flow of tokens or it is caused by the transitivity of the dependency relation. A zero flow of tokens means, that there is no direct dependency between events. In particular, there are no token flows between concurrent events of the event structure. The token flows also determine the construction of PES1. PES1 contains one copy of each run of the net for each possible token flow distribution within the run. As mentioned in the introduction identical events w.r.t. (IDENT1) having the same ingoing flows of tokens are not duplicated. But for each possible distribution of the ingoing token flows, one event is generated, i.e. there are identical events in PES1, but the token flows assigned to the ingoing edges of the events differ (e.g. the two c-events in Figure 4). Since in the standard unfolding, there may be exponentially many identical events having the same ingoing token flows, PES1 may have significantly less events than PES0.

Construction algorithm: We implemented a construction algorithm for PES1. It starts introducing an initial event v representing the initial marking of the p/t-net. This event constitutes the first consistency set, and the residual token flow of this event within the consistency set coincides with the initial marking. The residual token flow of an event within a consistency set represents the numbers of tokens produced by the event and not consumed by some other event in the consistency set. Transition occurrences are step by step appended in a fixed order. Given a consistency set and residual token flows for each event within this consistency set, one can append events consuming less tokens than given by the sum of the residual token flows. For each appended transition occurrence, all possibilities to distribute the residual token flows onto the ingoing token flow of this transition occurrence are considered. This is a combinatoric problem. Having solved this problem yields the candidates of events together with their ingoing token flows, that have to be appended to PES1 within the consistency set. Each such event leads to a new consistency set with new residual token flows for each event. The new residual token flows can easily be computed from the old ones. The only remaining challenge here is that some event, that should be appended in some stage of the algorithm, was possibly already added in some previous stage. Then it must not be duplicated. This has to be checked and leads to a recalculation of some events.

Performance: Altogether the algorithm should have a significantly faster runtime than the standard unfolding algorithm in many cases, because the number of events is a lot smaller (the difference can be of exponential size) and the procedure of appending events is similar: The stored token flows of a run determine the residual token flows, that can be used to append an event. The computation of the residual token flows is a simple summation operation. It is approximately as costly as the stepwise calculation of co-sets of conditions, which is necessary in the standard unfolding. Given the residual token flows, computing all possibilities to append an event is a combinatoric problem. This problem is of similar complexity as the procedure of testing all computed co-sets of conditions in order to append an event in the standard unfolding. But the construction algorithm for PES1 may also have a worse runtime than the standard unfolding in some special cases, because the application of a consistency set in the prime event structure PES1 causes the construction algorithm to recalculate some events (as already mentioned above), that do not have to be recalculated in the standard unfolding algorithm working with a binary conflict relation. That means the absence of a local binary conflict relation in PES1 disables the possibility to directly classify an appended event to runs. Therefore some events have to be considered by the algorithm more than once to determine a classification of the events to certain runs. Altogether there are two opposite effects concerning the runtime of the construction algorithm of PES1 in comparison to the standard unfolding: Less events are considered, but some events have to be recalculated, whereas the first effect seems to be more significant.

For the sake of completeness, it is not exactly true that PES1 constructed in this way, coincides with PES0 except for neglecting (IDENT1)-identical events (as mentioned in the introduction). There is also a very specific auto-concurrency situation, in which PES1 neglects some further events compared to PES0. But these events only generate isomorphic processes. Therefore it is even desirable to neglect these events. It is algorithmically possible not to neglect these events, but this is more costly and therefore makes no sense. On the contrary, it is an interesting extension approach to neglect all events generating isomorphic processes. We considered such extension of our construction algorithm for PES1, but not implemented it yet. The removal procedure for respective events is relatively costly, but in some cases it may significantly reduce the size of the constructed prime event structure. Note that this topic only plays a role in certain auto-concurrency situations.

3 Algorithm 2

PES2: The second method constructs a labelled prime event structure PES2 enriched with example token flow tuples for every edge of the structure (see Figure 5). The intuition behind the causal dependency relation and the consistency set of the prime event structure as well as the token flows assigned to edges of the structure is the same as for PES1. In contrast to PES1, PES2 does not contain identical events. That means, only one event having a certain set of pre-events is introduced (except for concurrent events in one run), although there are different possible distributions of the token flows on ingoing edges of the event (compare the c-events in Figure 4 and Figure 5). Only one example distribution of these possible token flows is stored. That means, that abstracting from the token flows, each run occurs

only once in PES2, while PES1 contains one instance of a run for each possible token flow distribution of the run. This drastically reduces the number of events of PES2 in contrast to PES1, the difference may even be of exponential size. Moreover as in PES1, PES2 only contains events, for which a token flow distribution being positive on each ingoing skeleton arc exists.

Construction algorithm: A construction algorithm for PES2 is as follows: Introducing an initial event v, defining the first consistency set, constitutes the starting point of the algorithm. Events are then step by step appended in a certain order. The procedure to append an event is difficult: Given a consistency set, all prefixes of the set are candidates to append an event labelled by a certain transition. Methods from flow theory can be used to calculate, if an event having certain pre-events can be appended (see [JLD05]). These methods calculate an appropriate token flow, which is a quite complex procedure. The methods utilize an existing example token flow of the consistency set and yield a new token flow, which replaces the example token flow for further calculations. A further difficulty, that has to be regarded, is that skeleton arcs are not allowed to have a zero token flow in all possible token flow distributions. Finally having computed an event that should be appended, there exists the same problem already described in the construction algorithm of PES1, that a respective event was possibly already added in some previous stage of the algorithm. Then the event must not be duplicated. Therefore some events are calculated more than once by the algorithm.

Performance: As explained, in this algorithm it is obviously possible that the stored example distribution of the token flows is not appropriate to append some event to some prefix of a run, but some other possible token flow distribution would allow to append this event. Therefore, to append an event in the construction algorithm of PES2, first a prefix of some run has to be chosen and then the token flows of this run have to be recalculated. Choosing a prefix is a similar even slightly less costly operation as the combinatorial problems occurring in the construction algorithms of PES1 and PES0. But an elaborate recalculation of token flows is not necessary in the other two approaches. Fortunately, there are efficient methods for the recalculation. In [JLD05] a polynomial method to test, if there exists an appropriate token flow to append an event, is shown. This method works without any additional information, but having already stored an example token flow distribution of the run significantly improves the runtime of this method. The already existing example token flows can then be redistributed to extend the token flows to the additional edges connected with the new event. It is not clear if this method is more or less efficient than the standard unfolding algorithm or the first presented method. On the one hand the number of events of PES2 is significantly smaller than the number of events of PES1 (and therefore a lot smaller than the number of events of PES0), but on the other hand appending one event requires a redistribution of the token flows and is therefore very costly compared to the standard unfolding or PES1. Moreover using a consistency set instead of a binary conflict relation in PES2 yields the same runtime problem of the construction algorithm as explained in the last section for PES1: Some events have to be recalculated. Since the reduction of the number of events may be exponential and the method for redistributing the token flows is polynomial, the construction of PES2 should at least in some cases be more efficient than constructing PES1 or the standard unfolding.

4 Comparison of the Algorithms and Experimental Results

In this section we experimentally test our implementation of the construction algorithm of PES0 having the standard unfolding algorithm constructing PES1 as a benchmark. To construct PES1, we use an adapted version of the unfolding algorithm in [DJL03]. When interpreting the results, one has to pay attention that this unfolding algorithm is not completely runtime optimized, but the remaining improvement potential should be very limited. We compare the runtime and the size of the resulting event structures. The upper table in Figure 7 shows a test of the parameterized version of the example net of Figure 1 shown in Figure 6. We computed PES0 and PES1 for several arc weights. The lower table in Figure 7 shows a test of the lower net in Figure 6 modelling for example a coffee automata. In this net the initial marking is varied.

The experimental results indicate that our new unfolding approach is superior to the standard approach. For the tested examples, the runtimes and the sizes of the resulting structures are a lot better in our new algorithm. It is clear, that PES1 is at least as big as PES0, but usually a lot smaller, if the net contains arc weights or a non-safe initial marking. In these cases our new algorithm is also significantly faster than the standard algorithm. We did not check memory consumption. While the size of PES1 argues for the new algorithm, the need to store consistency sets increases the memory consumption of the new algorithm. This still has to be tested.

5 Conclusion

In this paper we propose two new unfolding semantics for p/t-nets based on the concept of prime event structures. The definitions of the two unfolding models are motivated by algorithmic aspects. We develop a construction algorithm for both unfolding models. We show that there are many cases in which our implemented algorithms is significantly more efficient than standard unfolding methods for p/t-nets. We also justify the applicability of the two newly developed unfolding methods.

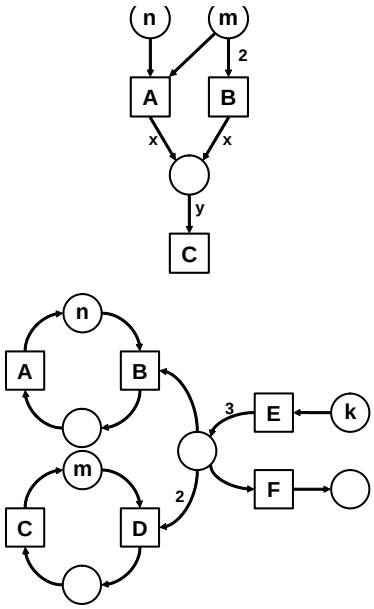

Figure 6: Parameterized test nets.

				PES0			PES1		
n	m	x	y	E	P	time	E	P	time
1	3	2	3	18	12	0,36s	4	2	0,08s
2	4	2	3	266	133	10,99s	14	6	0,05s
1	3	4	2	90	315	14,20s	10	3	0,04s
1	3	4	3	173	840	157,20s	8	6	0,05s
3	4	2	3	798	612	813,04s	21	10	0,06s
3	4	5	3	-	-	-	44	104	3,45s
	n	m	k						
	1	1	1	40	22	0,20s	12	6	0,09s
	1	2	1	46	28	0,63s	12	6	0,09s
	2	1	1	70	58	2,67s	16	9	0,13s
	2	2	1	76	67	4,63s	16	9	0,13s
	1	1	2	-	-	-	178	150	1,92s
	1	2	2	-	-	-	182	174	2,13s
	2	1	2	-	-	-	234	385	13,69s
	2	2	2	-	-	-	239	413	15,81s

Figure 7: Experimental results: E shows the number of events and P the number of processes in the constructed structure.

It could be interesting to adapt the two prime event structure unfolding methods presented in this paper in such a way that the nondeterminism can still be described by a binary conflict relation. Throughout this paper we discussed some reasons justifying that such approach could lead to an improvement of runtime as well as space consumption.

References

[BD87] Best, E. ; Devillers, R.R.: Sequential and Concurrent Behaviour in Petri Net Theory. In: *Theoretical Computer Science* 55 (1987), Nr. 1, S. 87–136

[CPW04] Couvreur, J.-M. ; Poitrenaud, D. ; Weil, P.: Unfoldings for General Petri Nets. In: *http://www.labri.fr/perso/weil/publications/depliage.pdf* University de Bordeaux I (Talence, France), University Pierre et Marie Curie (Paris, France) (2004)

[DJL03] Desel, J. ; Juhás, G. ; Lorenz, R.: *VipTool-Homepage.* 2003. – http://www.informatik.ku-eichstaett.de/projekte/vip/

[Eng91] Engelfriet, J.: Branching Processes of Petri Nets. In: *Acta Informatica* 28 (1991), Nr. 6, S. 575–591

[GR83] Goltz, U. ; Reisig, W.: The Non-Sequential Behaviour of Petri Nets. In: *Information and Control* 57 (1983), Nr. 2/3, S. 125–147

[Haa98] Haar, S.: Branching Processes of general S/T-Systems and their properties. In: *Electr. Notes Theor. Comput. Sci.* 18 (1998)

[HKT96] Hoogers, P.W. ; Kleijn, H.C.M. ; Thiagarajan, P.S.: An Event Structure Semantics for General Petri Nets. In: *Theoretical Computer Science* 153 (1996), Nr. 1&2, S. 129–170

[JLD05] Juhás, G. ; Lorenz, R. ; Desel, J.: Can I Execute My Scenario in Your Net?. In: Ciardo, G. (Hrsg.) ; Darondeau, P. (Hrsg.): *ICATPN* Bd. 3536, Springer, 2005 (Lecture Notes in Computer Science), S. 289–308

[MMS94] Meseguer, J. ; Montanari, U. ; Sassone, V.: On the Model of Computation of Place/Transition Petri Nets. In: Valette, R. (Hrsg.): *Application and Theory of Petri Nets* Bd. 815, Springer, 1994 (Lecture Notes in Computer Science), S. 16–38

[MMS97] Meseguer, J. ; Montanari, U. ; Sassone, V.: On the Semantics of Place/Transition Petri Nets. In: *Mathematical Structures in Computer Science* 7 (1997), Nr. 4, S. 359–397

[NPW81] Nielsen, M. ; Plotkin, G.D. ; Winskel, G.: Petri Nets, Event Structures and Domains, Part I. In: *Theoretical Computer Science* 13 (1981), S. 85–108

[Win86] Winskel, G.: Event Structures. In: Brauer, W. (Hrsg.) ; Reisig, W. (Hrsg.) ; Rozenberg, G. (Hrsg.): *Advances in Petri Nets* Bd. 255, Springer, 1986 (Lecture Notes in Computer Science), S. 325–392