Generating Petri Net State Spaces

Karsten Wolf

Universität Rostock, Institut für Informatik,
18051 Rostock, Germany
karsten.wolf@informatik.uni-rostock.de

Abstract. Most specific characteristics of (Place/Transition) Petri nets can be traced back to a few basic features including the *monotonicity* of the enabling condition, the *linearity* of the firing rule, and the *locality* of both. These features enable "Petri net" analysis techniques such as the invariant calculus, the coverability graph technique, approaches based on unfolding, or structural (such as siphon/trap based) analysis. In addition, most verification techniques developed outside the realm of Petri nets can be applied to Petri nets as well.

In this paper, we want to demonstrate that the basic features of Petri nets do not only lead to additional analysis techniques, but as well to improved implementations of formalism-independent techniques. As an example, we discuss the explicit generation of a state space. We underline our arguments with some experience from the implementation and use of the Petri net based state space tool LoLA.

1 Introduction

Most formalisms for dynamic systems let the system state evolve through reading and writing variables. In contrast, a Petri net marking evolves through the consumption and production of resources (tokens). This fundamental difference has two immediate consequences. First, it leads to a *monotonous enabling condition*. This means that a transition enabled in some state is as well enabled in a state that carries additional tokens. Second, it leads to the *linearity of the firing rule*. This means that the effect of a transition can be described as the addition of an integer vector to a marking vector.

The Petri net formalism has another fundamental property that it shares with only some other formalisms: *locality*. This means that every transition depends on and changes only few components of a state. In Petri nets, these components (places) are explicitly visible through the arc (flow) relation.

Many specific Petri net analysis techniques can be directly traced back to some of these characteristic features of Petri nets. For instance, the *coverability graph* generation [KM69, Fin90] is closely related to monotonicity and linearity. The *invariant* calculus [LS74, GL83, Jen81] clearly exploits linearity of the firing rule. For structural analysis such as *Commoner's Theorem* [Com72] or other methods based on siphons and traps, monotonicity and locality may be held responsible. Petri net *reduction* [Ber86] is based on the locality principle. Analysis

J. Kleijn and A. Yakovlev (Eds.): ICATPN 2007, LNCS 4546, pp. 29–42, 2007.

based on *unfoldings* [McM92,Esp92] (or partial ordered runs) involves the locality principle as well. This list could be continued.

On the other hand, there are several verification techniques that do not depend on a particular modeling language. An example for this class of techniques is *state space* based analysis, and—build on top of it—*model checking* [CE81,QS81]. The availability of specific techniques like the ones mentioned above has long been observed as a particular advantage of Petri nets as a modeling language. Beyond this insight, we want to discuss the implementation of a Petri net based state space generator (LoLA [Sch00c]) in order to show that the basic features of Petri nets mentioned in the beginning of this section can be used for an efficient implementation of—otherwise formalism-independent—explicit state space generation.

We start with a short introduction to the tool LoLA and give, in Sec. 3 a brief overview on some case studies conducted with this tool. In Sec. 4, we consider the implementation of fundamental ingredients of a state space generator such as firing a transition or checking for enabledness. Finally, in Sec. 5, we informally discuss a few Petri net specific issues of state space reduction techniques.

2 The State Space Generation Tool LoLA

The LoLA project started in 1998. Its original purpose was to demonstrate the performance of state space reduction techniques developed by the author. Most initial design decisions were driven by prior experience with the state space component of the tool INA [RS98].

LoLA can read place/transition nets as well as high-level nets. High-level nets are immediately unfolded into place/transition nets So, LoLA is indeed a place-transition net tool while the high-level input can be seen as a shorthand notation for place/transition nets.

LoLA generates a state space always for the purpose of verifying a particular property. It implements the in-the-fly principle, that is, it stops state space generation as soon as the property being verified is determined. The properties that can be verified include

- Reachability of a given state or a state that satisfies a given state predicate;
- Boundedness of the net or a given place;
- Quasiliveness of a given transition;
- Existence of deadlocks;
- Reversibility of the net;
- Existence of home state;
- Liveness of a state predicate;
- Validity of a CTL formula;
- Validity of the LTL formulas $F\phi$, $GF\phi$, and $FG\phi$, for a given state predicate ϕ.

LoLA can build a state space in depth-first or breadth-first order, or it can just generate random firing sequences for an incomplete investigation of the state

space. While depth-first search is available for the verification of all properties listed above, the other search methods are restricted to reachability, deadlocks and quasiliveness.

The following reduction techniques are available in LoLA:

- Partial order reduction (the stubborn set method);
- The symmetry reduction;
- The coverability graph generation;
- The sweep-line method;
- Methods involving the Petri net invariant calculus;
- a simplified version of bit hashing

The techniques are employed only if, and in a version that, preserves the property under investigation. If applicable, reduction techniques can be applied in combination.

Using LoLA amounts to executing the following steps.

1. Edit a particular file `userconfig.H` in the source code distribution for selecting the class of property to be verified (e.g., "boundedness of a place" or "model check a CTL formula") and the reduction techniques to be applied.
2. Translate the source code into an executable file `lola`.
3. Call `lola` with a file containing the net under investigation, and a file specifying the particular instance of the property (e.g., the name of the place to be checked for boundedness, or the particular CTL formula to be verified).
4. LoLA can return a witness state or path, an ASCII description of the generated state space, and some other useful information.

Instead of a stand alone use of LoLA, it is also possible to rely on one of the tools that have integrated LoLA, e.g.,

- CPN-AMI [KPA99],
- The Model Checking Kit [SSE03]
- The Petri net Kernel [KW01]

3 Some Applications of LoLA

As LoLA can be downloaded freely, we do not have a complete overview on its applications. In this section, we report on a few case studies conducted by ourselves, and studies we happen to know about.

Validating a Petri Net Semantics for BPEL [HSS05]
WS-BPEL (also called BPEL or BPEL4WS, [Cur+03]) is an XML-based language for the specification of web services. Due to an industrial standardization process involving several big companies, the language contains a lot of non-orthogonal features. It was thus adequate to give a formal semantics to BPEL

in order to reason about consistency and unambiguity of the textual specification. Several formal semantics have been proposed in various formalisms, among them two which are based on Petri nets. One of the Petri net semantics has been reported in [HSS05]. It translates every language construct of BPEL into a Petri net fragment. The fragments are glued together along the syntactical structure of a BPEL specification. As the fragments interact in a nontrivial way, it was necessary to validate the semantics. The validation was done by translating several BPEL specifications into Petri nets and letting LoLA verify a number of crucial properties concerning the resulting nets.

The most successful setting in the application of LoLA was the combination of partial order reduction (stubborn sets) with the sweep-line method. Partial order reduction performed well as BPEL activities can be executed concurrently (availability of a significant number of concurrent activities is essential for the partial order reduction). Furthermore, the control flow of a BPEL specification follows a pattern of progress towards a fixed terminal configuration. Such progress can be exploited using the sweep-line method. LoLA detects the direction of progress automatically (this issue is further discussed in Sec. 5).

It turned out that LoLA was able to solve verification tasks for services with up to 40 BPEL activities while it ran out of memory for a service with little more than 100 activities. The nets that could be successfully verified consisted of about 100 to 400 places and 250 to 1.000 transitions. Reduced state spaces had up to 500.000 states. With the capability of handling BPEL specifications with 40 or 50 activities, LoLA is well in the region of practical relevance. So, LoLA has become part of a tool chain for web services that is developed within the Tools4BPEL project [T4B07]. There, ideas exist to tackle larger processes through further tuning the translation from BPEL to Petri nets, and through abstraction techniques to be applied prior to state space verification.

The main lesson learned of this application was that partial order reduction in combination with the sweep-line method is a powerful combination of reduction techniques in the area of services.

Detecting Hazards in a GALS Circuit [SRK05]

GALS stands for globally asynchronous, locally synchronous circuits. A GALS design consists of a number of components. Each component has its own clock signal and works like a synchronous circuit. Communication between components is organized in an asynchronous fashion. This way, designers try to save the advantages of a synchronous design (tool support, clearly understood setting) while they tackle the major drawbacks (speed limitation and high energy consumption due to long distance clock signal paths, energy consumption in idle parts of the circuit).

In collaboration with the Institute of Semiconductor Physics in Frankfurt/-Oder, we investigated a GALS circuit for coding and decoding data of the 802.11 wireless LAN protocol. In particular, we translated a so-called GALS wrapper gate by gate into a place/transition net. A GALS wrapper is the part of a GALS design that encapsulates a component, manages incoming data, and pauses the

clock of the component during periods where no data are pending. Consequently, a wrapper as such is an asynchronous circuit. It consists of five components: an input port maintaining incoming data, an output port managing outgoing data, a timeout generator, a pausable clock, and a clock control. All in all, there are 28 gates (logical gates, flip-flops, muller-C-elements, counters, and mutex elements).

We were interested in the detection of hazards in the wrapper. A hazard is a situation where, due to concurrently arriving input signals, it is not clear whether the corresponding output signal is able to fully change its value (after arrival of the first signal) before switching back to the original value. Such an incomplete signal switch may cause undefined signal values which are then potentially propagated through the whole circuit. The Petri net model was built such that the occurrence of a hazard in a particular gate corresponds to a particular reachable marking in the Petri net model of the gate.

LoLA was applied for solving the various reachability queries. Again, a combination of partial order reduction with the sweep-line method turned out to be most successful. Nevertheless, LoLA was not able to solve all of the reachability queries on the original model (having 286 places and 466 transitions). In a second approach, the model was replaced by a series of models each modeling one of the parts of the wrapper in detail while containing abstract versions of the others. The abstraction was based on the assumption that no hazards occur in the abstracted part. LoLA was then able to solve all reachability queries. The largest state space had little more than 10.000 nodes.

We detected eight hazards and reported them to the people in Frankfurt. For each hazard, we could derive a scenario for its occurrence from the witness paths available in LoLA. Six of the scenarios could be ruled out through knowledge about timing constraints. The remaining two hazards were considered as really dangerous situations. Using LoLA again, a re-design of the wrapper was verified as being hazard-free.

The approach of modeling one part of the investigated system in detail while abstracting the others was the main lesson learned out of this application

Garavel's Challenge

Back in 2003, Hubert Garavel posted a challenge to the Petri Net Mailing List. The mail contained a place/transition net with 485 places and 776 transitions that allegedly stemmed from a LOTOS specification. Garavel was interested in quasi-liveness of all transitions of the net.

Apart from LoLA, the two symbolic state space tools SMART (by G. Chiardo and R. Siminiceanu) and Versify (by O. Roig), and the tool TINA (by B. Berthomieu and F. Vernadat) which used the covering step graph technique responded to the challenge. The symbolic tools were able to calculate the exact number of reachable states of the system which was in the area of 10^{22}.

The LoLA approach to the challenge was to generate not just one (reduced) state space but 776 of them, one for the verification of quasi-liveness of a particular transition. This way, partial order reduction could be applied very successfully. 774 of the queries could be solved this way while two queries ran out of

memory. For these transitions, we applied then the LoLA feature of generating random firing sequences. In fact, the sequences are not totally random. Instead, the probability of selecting a transition for firing is weighted according to a heuristics which is closely related to the stubborn set method. This heuristics is quite successful in attracting a firing sequence towards a state satisfying the investigated property. At least, it worked for the two problematic transitions in the challenge and we were able to report, for each transition, a path witnessing its quasi-liveness.

This challenge showed that LoLA can be competitive even to symbolic state space tools. A major reason for success was the division of the original verification problem into a large number of simpler verification tasks.

Exploring Biochemical Networks [Tal07]
In a biochemical network, a place represents a substance, tokens in a place represent presence of the substance. A transition models a known chemical reaction. A transition sequence that finally marks a place represents a chain of possible reactions that potentially generates the corresponding substance.

People at SRI use LoLA for exploring reaction paths. According to [Tal07], they "use LoLA because it is very fast in finding paths".

The examples show that LoLA can be applied in various areas. It is able to cope with models of practically relevant systems. The performance of LoLA is due to at least four reasons:

- LoLA features a broad range of state-of-the-art state space reduction techniques most of which can be applied in combination;
- LoLA offers specialized reduction techniques for every property listed in the previous section;
- LoLA uses the formalism of place/transition nets which can be handled much easier than a high-level net formalism;
- The core procedures in LoLA exploit the basic characteristics of Petri nets as mentioned in the introduction.

4 Core Procedures in a State Space Generator

In this section, we demonstrate how the basic characteristics of Petri nets can be taken care of in the implementation of a state space generator. A state space generator is basically an implementation of a search through the state graph, typically a depth-first search. The elementary steps of a depth-first search include the following steps, each discussed in a dedicated subsection. When discussing complexity, we assume the investigated system to be *distributed*. Formally, we assume that there is a fixed value k such that every transition has, independently of the size of the net, at most k pre- and post-places. Several realistic distributed systems satisfy such a requirement for a reasonably small k, for even more systems there are only few transitions violating the assumption.

Firing a Transition
By firing a transition, we proceed from one state to a successor state. In a Petri net, the occurrence of a transition t changes the marking of at most $card(\bullet t) + card(t\bullet)$ places. According to the assumption made above, this number is smaller than $2k$. By maintaining, for each transition, an array of pre- and post-places, it is indeed possible to implement the occurrence of a transition in time $O(1)$. The ability to easily implement the firing process in constant time can be contributed to *locality.* In fact, other formalisms exhibiting locality have the same opportunity (like the model checking tool SPIN [Hol91] using the guarded command style language PROMELA. In contrast, the input language of the model checker SMV [McM02] does not support explicitly a notation of locality, and it would require a lot of expensive analysis for an explicit model checker to retrieve information on locality from SMV input. Note that SMV is not an explicit model checker, so this consideration does not concern SMV as such.

Checking Enabledness
The enabling status of a transition can change only due to the occurrence of a transition t. So, except, for an initial enabledness check on the initial marking, the check for enabledness can be reduced to the transitions in $\bullet t \cup t\bullet$. This approach is again feasible for all formalisms exhibiting *locality.* For Petri nets, however, the check for enabledness can be further refined due to the *monotonicity* of the enabling condition. If t' is enabled before having fired t, and t is only adding tokens to places in $\bullet t'$, it is clear that t' is still enabled after having fired t. Likewise, a previously disabled t' remains disabled if t only removes tokens from $\bullet t'$. This way, the number of enabledness checks after a transition occurrence can be significantly reduced. In LoLA, we maintain two separate lists of transitions for each t: those that can potentially be *enabled* by t (must be checked if they have been disabled before), and those that can be potentially disabled by t (must be checked if they have been enabled before). Through an additional treatment of all *enabled* transitions as a doubly linked list (with the opportunity to delete and insert an element at any position), it is possible to retrieve a list of enabled transitions in a time linear to the number of enabled transitions (which is typically an order of magnitude smaller than the overall number of transitions).

Returning to a Previously Seen Marking
In depth-first search, we typically have a *stack* of visited but not fully explored markings. This stack actually forms a path in the state space, that is, the immediate successor of marking m on the stack is reachable from m through firing a single transition. After having fully explored marking m on top of the stack, we proceed with its immediate predecessor m' on this stack. As Petri nets enjoy the *linearity* of the firing rule, there is a strikingly simple solution to this task: just fire the transition *backwards* that transformed m' to m. This way, it takes constant effort to get back to m'.

For assessing the value of this implementation, let us discuss potential alternatives. Of course, it would be possible to maintain a stack that holds full markings. Then returning to a previous marking amounts to redirecting a single pointer. But in depth-first exploration, the search stack typically holds a substantial number of visited states, so this approach would pay space for time. In state space verification, space is, however, the by far more limited resource. Another solution suggests to maintain, for each stack element, a pointer into the repository of visited markings. This data structure is, in principle, maintained in any explicit state space verification, so this solution would not waste memory at the first glance. For saving memory, it is, however, highly recommendable to deposit visited markings in a compressed form [WL93]. Thus, calculations on a marking in this compressed form require nontrivial run time. Finally, this solution prohibits approaches of storing only some reachable markings in the repository (see Sec. 5 for a discussion on such a method).

Maintaining the Visited Markings
According to the proposal to organize backtracking in the search by through firing transitions backwards, there are only two operations which need to be performed for on the set of visited markings. One operation is to search whether a newly encountered marking has been seen before, the other is to insert that marking if it has not. All other operations, including the evaluation of state predicates, computing the enabled transitions, computing successor and predecessor markings etc. can be performed on a single uncompressed variable, call it `CurrentMarking` (in the case of LoLA: an array of integers). For searching `CurrentMarking` and inserting it in the depository, we can look up and insert its compressed version.

In consequence, it is at no stage of the search necessary to uncompress a marking! This fact can be exploited for compressions where the uncompression is hard to realize. In LoLA, we have implemented such a technique [Sch03] that is based on place invariants (thus, a benefit from the *linearity* of the firing rule). Using a place invariant I, we can express the number of tokens of one place p in $supp(I)$ in terms of the others. We can thus exempt the value of p from being stored in any marking. Given n linearly independent place invariants, the number of values to be stored can be reduced by n. The number n typically ranges between 20% and 60% of the overall number of places, so the reduction is substantial. It does not only safe space but time as well. This is due to the fact that a look up in the depository is now performed on a smaller vector.

Compressing a marking according to this technique is rather easy: we just need to throw away places marked a "dependent" in a preprocessing stage. Uncompressing would require an evaluation of the justifying place invariant. In particular, it would be necessary to keep the invariant permanently in storage! In LoLA, we do not need to keep them. Concerning space, the additional costs of the approach, beyond preprocessing, amount to one bit ("dependent") for each place. Even in preprocessing, it is not necessary to fully compute the invariants. As explained in [Sch03], the information about mutual dependency can be

deduced from an upper triangle form of the net incidence matrix, an intermediate stage of the calculation. This, way, invariant based preprocessing requires less than a second of run time even for a net with 5.000 places and 4.000 transitions. In that particular system, we would have 2.000 linearly independent place invariants (each being a vector of length 5.000!).

Breadth-First Search
While depth-first search is the dominating technique for state space exploration, breadth-first search can be used for some purposes as well, for instance for the calculation of a shortest path to some state. In breadth-first search, subsequently considered states are not connected by a transition occurrence. Nevertheless, it is possible to preserve some of the advantages of the backtracking through firing transitions. In LoLA, we mimic breadth-first search by a depth-first search with an incrementally increased depth restriction. That is, we proceed to the next marking to be considered by stepping back a few markings (through firing some transitions backwards) and then firing some other transitions forward. The average number of transitions to be fired is reasonably small as the number of states tends to grow exponentially with increased depth. This is true even for reduced state spaces, as some of the most powerful reduction techniques require the use of depth-first search.

5 Reduction Techniques

In this section, we discuss a few state space reduction techniques and show that the basic characteristics of Petri nets lead to specific solutions.

Partial Order Reduction
Roughly spoken, the purpose of partial order reduction is to suppress as many as possible interleaved firings of concurrently enabled transitions. This goal is achieved by considering, in each marking, only a subset of the enabled transitions. This subset is computed such that a given property or class of properties is preserved in the reduced state space.

It is well-known that *locality* is the major pre-requisite of the stubborn set method [Val88] and other methods of partial order reduction [Pel93, GW91, GKPP95]. Furthermore, *linearity* of the firing rule turns out to be quite beneficial. The reason is that partial order reduction is, among others, concerned with permutations of firing sequences. It is typically desired that a firing sequence reaches the same marking as the permuted sequence. Due to the *linearity* of the firing rule, this property comes free for Petri nets. For other formalisms, it needs to be enforced, as can be seen in [Val91]. This way, other formalisms have additional limitations in the application of partial order reduction.

For partial order reduction, there is another source of efficiency that is worth being mentioned. It is not related to the formalism of Petri nets itself, but with

the tradition of Petri net research. In the area of Petri nets, people have studied a broad range of singular properties such as boundedness, liveness, reversibility, reachability, deadlock freedom, etc. Following this tradition, it was apparent to support each of these properties with a *dedicated* version of partial order reduction [Sch99, Sch00d, KV00, KSV06]. In contrast, it is the tradition of model checking to support a rich language or two (such as the temporal logics CTL [Eme90] or LTL [MP92]). According this line of research, people came up with reduction techniques that support the whole language [Pel93, GKPP95]. It is evident, that a dedicated reduction technique for property X can lead to a better reduction than a generic technique for a specification language can can express X. We believe that this observation is crucial for the competitivity of LoLA in various areas.

The Symmetry Method

Symmetrically structured systems exhibit a symmetric behavior. Exploiting symmetry means to suppress consideration of a state if a symmetric state has been considered before.

Most approaches search for symmetric structures in data types of the specification. The most popular data type in this respect is the so-called *scaler set* [DDHC92] where variables can be compared for equality, used as indices in arrays and order-independent loops, while there are no constants of that type. In [CDFH90], a rather sophisticated detection of symmetric structure in data types is described.

Due to the *locality* of Petri nets, place/transition nets have a rather fine grained graphical representation. This feature enables another approach to finding symmetries in the structure: we can compute the graph automorphisms of the Petri net graph [Sta91, Sch00a, Sch00b, Jun04]. There is a polynomial size generating set of the automorphism group of a graph, and it can be computed in reasonable time (though not always in polynomial time). The generating set is sufficient for an approximated calculation of a canonical representative of a marking [Sch00b], a method for detecting previously seen symmetric states during state space calculation. The graph automorphism based approach to symmetry is a unique feature of LoLA and INA [RS98] (both implemented by the author of this article).

The main advantage of the graph automorphism approach is that it can recognize *arbitrary* symmetry groups while the data type approach is restricted to a couple of standard symmetries.

The Sweep-Line Method

The sweep-line method assumes that there is a notion of *progress* in the system evolution. That is, assigning a progress value to each state, successor markings tend to have larger progress values than their predecessors. This observation can be exploited by traversing the search space in order of increasing progress values, and to remove visited markings from the depository which have smaller

progress value than the currently considered markings. For reason of correctness, markings which are reached through a transition that decreases the progress value, are stored permanently, and their successors are encountered.

The original method [Mai03, CKM01, KM02] assumes that the progress measure is given manually. However, exploiting the *linearity* of the Petri net firing rule, it is possible to compute a progress measure automatically. The measure being calculated assigns some arbitrary progress value, say 0, to the initial state. Then, each transition t gets an offset $o(t)$ such that, if t fired in m leads to m', the progress value of m' is just the progress value of m plus $o(t)$. For correctness, it is important that different firing sequences from the initial marking to a marking m all yield the same progress value for m. This can, however, been taken care of by studying linear dependencies between transition vectors. In LoLA, we compute the measure by assigning an arbitrary offset, say 1, to each transition in a maximum size linearly independent set U of transitions. For the remaining transitions (which are linear combinations of U) the offset is then determined by the correctness requirement stated above. All applications of the sweep-line method reported in this article have been conducted with an automatically computed progress measure.

Cycle Coverage

The depository of visited markings is the crucial date structure in a state space verification. In explicit methods, the size of the depository grows with the number of visited states. The number of states to be stored can, however, be reduced in a trade that sells time for space. By simply exempting states from being stored, we obviously safe space but lose time as, in a revisit to a forgotten state, its successors are computed once again. For an implementation of this idea, it is, as for instance observed in [LLPY97], important to store at least one marking of each cycle in the state graph. This condition actually ensure termination of the approach.

Thanks to *linearity* in the Petri net firing rule, it is fairly easy to characterize a set of states such that every cycle in the state graph is covered. We know that every firing sequence that reproduces the start marking forms a transition invariant. Thus, choosing a set of transitions U such that the support of every transition invariant contains an element from U, it is evident that every cycle in the state graph contains at least one marking where a transition in U is enabled. This approach has been described in [Sch03].

Combination of Reduction Techniques

Most techniques mentioned above can be applied in combination. The combined application typically leads to additional reduction like in the case of joint application of partial order reduction with the symmetry method. For some reduction techniques, we experienced that their joint application with another technique is actually a pre-requisite for a substantial reduction as such. For instance, the sweep-line method only leads to marginal reduction for Petri nets with a lot of cycles [Sch04]. Also, the cycle coverage reduction does not perform well on such systems [Sch03]. Both methods can, however, lead to substantial (additional!) reduction when they are applied to a stubborn set reduced state space. This is due to

a particular effect of partial order reduction. If a system consists of several, mostly concurrently evolving, cyclic components, then the almost arbitrary interleaving of transitions in these components closes a cycle in almost every reachable state. This causes a tremendous number of regress transitions in the sweep-line method (and thus a huge number of states to be stored permanently) and a huge number of states to be stored with the cycle coverage reduction. Partial order reduction decouples the arbitrary interleaving of concurrent components. A partial order reduced state space contains only a fraction of the cycles of the original state space, and the remaining cycles tend to be significantly larger.

6 Conclusion

Petri nets as a formalism for modeling systems enjoy specific properties including locality, linearity and monotonicity. These properties lead to specific verification techniques such as the coverability graph, the invariant calculus, siphon/trap based analyses, or the unfolding approach. In this article we demonstrated, that the specific properties of Petri nets are as well beneficial for the implementation of techniques which are otherwise applicable in other formalisms as well. Our discussion covered explicit state space verification as such, but also a number of state space reduction techniques all of which can be applied to several modeling languages.

All mentioned methods have been implemented in the tool LoLA. LoLA is able to solve problems that have practical relevance. We hold three reasons responsible for the performance of LoLA:

- A consistent exploitation of the basis characteristics of Petri nets,
- A broad variety of reduction techniques which can be applied in many combinations, and
- The availability of dedicated reduction techniques for frequently used singular properties.

In this light, it is fair to say that LoLA is a *Petri net* state space tool.

References

[Ber86] Berthelot, G.: Checking properties of nets using transformations. Advances in Petri Nets, pp. 19–40 (1986)

[CDFH90] Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: On well–formed colored nets and their symbolic reachability graph. In: Proc. ICATPN, pp. 378–410 (1990)

[CE81] Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronisation skeletons using brnaching tim temporal logic. In: Logics of Programs. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1981)

[CKM01] Christensen, S., Kristensen, L.M., Mailund, T.: A sweep-line method for state space exploration. In: Margaria, T., Yi, W. (eds.) ETAPS 2001 and TACAS 2001. LNCS, vol. 2031, pp. 450–464. Springer, Heidelberg (2001)

[Com72] Commoner, F.: Deadlocks in Petri nets. Technical report, Applied Data Research Inc. Wakefield, Massachussetts (1972)

[Cur+03] Curbera, F., et al.: Business process execution language for web services, version 1.1. Technical report, BEA, IBM, Microsoft (2003)

[DDHC92] Dill, D.L., Drexler, A.J., Hu, A.J., Yang, C.H.: Protocol verification as a hardware design aid. In: Proc. IEEE Int. Conf. Computer Design: VLSI in Computers and Processors, pp. 522–525 (1992)

[Eme90] Emerson, E.A.: Handbook of Theoretical Computer Science. Chapter 16. Elsevier, Amsterdam (1990)

[Esp92] Esparza, J.: Model checking using net unfoldings. Technical Report 14/92, Universität Hildesheim (1992)

[Fin90] Finkel, A.: A minimal coverability graph for Petri nets. In: Proc. ICATPN, pp. 1–21 (1990)

[GKPP95] Gerth, R., Kuiper, R., Peled, D., Penczek, W.: A partial order approach to branching time logic model checking. In: Symp. on the Theory of Computing and Systems, IEEE, pp. 130–140 (1995)

[GL83] Genrich, H., Lautenbach, K.: S–invariance in Pr/T–nets. Informatik–Fachberichte 66, 98–111 (1983)

[GW91] Godefroid, P., Wolper, P.: A partial approach to model checking. In: IEEE Symp. on Logic in Computer Science, pp. 406–415 (1991)

[Hol91] Holzmann, G.: Design an Validation of Computer Protocols. Prentice-Hall, Englewood Cliffs (1991)

[HSS05] Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri nets. In: Proc. BPM, LNCS 3649, pp. 220–235 (2005)

[Jen81] Jensen, K.: Coloured Petri nets and the invariant method. Theoretical Computer Science 14, 317–336 (1981)

[Jun04] Junttila, T.: New canonical representative marking algorithms for place/transition nets. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 258–277. Springer, Heidelberg (2004)

[KM69] Karp, R.M., Miller, R.E.: Parallel programm schemata. Journ. Computer and System Sciences 4, 147–195 (1969)

[KM02] Kristensen, L.M., Mailund, T.: A generalized sweep-line method for safety properties. In: Eriksson, L.-H., Lindsay, P.A. (eds.) FME 2002. LNCS, vol. 2391, pp. 549–567. Springer, Heidelberg (2002)

[KPA99] Kordon, F., Paviot-Adet, E.: Using CPN-AMI to validate a sfae channel protocol. Tool presentation at ICATPN (1999)

[KSV06] Kristensen, L., Schmidt, K., Valmari, A.: Question-guided stubborn set methods for state properties. Formal Methods in System Design 29(3), 215–251 (2006)

[KV00] Krisensen, L.M., Valmari, A.: Improved question-guided stubborn set methods for state properties. In: Proc. ICATPN, pp. 282–302 (2000)

[KW01] Kindler, E., Weber, M.: The Petri net kernel - an infrastructure for building petri net tools. STTT 3 (4), 486–497 (2001)

[LLPY97] Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Efficient verification of real-time systems: compact data structure and state-space reduction. In: Proc. IEEE Real-Time Systems Symp., pp. 14–24 (1997)

[LS74] Lautenbach, K., Schmidt, H.A.: Use of Petri nets for proving correctness of concurrent process systems. IFIP Congress, pp. 187-191 (1974)

[McM02] McMillan, K.: The SMV homepage. `http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/`

[Mai03] Mailund, T.: Sweeping the State Space - a sweep-line state space exploration method. PhD thesis, University of Aarhus (2003)

[MP92] Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems, vol. 1: Specification. Springer, Heidelberg (1992)

[McM92] McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronious circuits. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 164–177. Springer, Heidelberg (1993)

[Pel93] Peled, D.: All from one, one for all: on model–checking using representitives. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 409–423. Springer, Heidelberg (1993)

[QS81] Quielle, J.P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: International Symposium on Programming. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1981)

[RS98] Roch, S., Starke, P.: INA Integrierter Netz Analysator Version 2.1. Technical Report 102, Humboldt–Universität zu Berlin (1998)

[Sch99] Schmidt, K.: Stubborn set for standard properties. In: Donatelli, S., Kleijn, J.H.C.M. (eds.) ICATPN 1999. LNCS, vol. 1639, pp. 46–65. Springer, Heidelberg (1999)

[Sch00a] Schmidt, K.: How to calculate symmetries of Petri nets. Acta Informatica 36, 545–590 (2000)

[Sch00b] Schmidt, K.: Integrating low level symmetries into reachability analysis. In: Schwartzbach, M.I., Graf, S. (eds.) ETAPS 2000 and TACAS 2000. LNCS, vol. 1785, pp. 315–331. Springer, Heidelberg (2000)

[Sch00c] Schmidt, K.: LoLA – a low level analyzer. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 465–474. Springer, Heidelberg (2000)

[Sch00d] Schmidt, K.: Stubborn set for modelchecking the EF/AG fragment of CTL. Fundamenta Informaticae 43 (1-4), 331–341 (2000)

[Sch03] Schmidt, K.: Using Petri net invariants in state space construction. In: Garavel, H., Hatcliff, J. (eds.) ETAPS 2003 and TACAS 2003. LNCS, vol. 2619, pp. 473–488. Springer, Heidelberg (2003)

[Sch04] Schmidt, K.: Automated generation of a progress measure for the sweep-line method. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 192–204. Springer, Heidelberg (2004)

[SSE03] Schröter, C., Schwoon, S., Esparza, J.: The Model-Checking Kit. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 463–472. Springer, Heidelberg (2003)

[SRK05] Stahl, C., Reisig, W., Krstic, M.: Hazard detection in a GALS wrapper: A case study. In: Proc. ACSD, pp. 234–243 (2005)

[Sta91] Starke, P.: Reachability analysis of Petri nets using symmetries. J. Syst. Anal. Model. Simul. 8, 294–303 (1991)

[Tal07] Talcott, C.: Personal communication. Dagstuhl Seminar (February 2007)

[T4B07] Reisig, W., et al.: The homepage of the project Tools4BPEL `http://www2.informatik.hu-berlin.de/top/forschung/projekte/tools4bpel/`

[Val88] Valmari, A.: Error detection by reduced reachability graph generation. In: ICATPN (1988)

[Val91] Valmari, A.: Stubborn sets for reduced state space generation. In: Advances in Petri Nets 1990. LNCS, vol. 483, pp. 491–511. Springer, Heidelberg (1991)

[WL93] Wolper, P., Leroy, D.: Reliable hashing without collision detection. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 59–70. Springer, Heidelberg (1993)