How to Verify Aggregating Global Properties

Karsten Wolf

Universität Rostock, Universitätsplatz 1, 18051 Rostock, Germany

Abstract

Some global properties of Petri nets, such as liveness or k-boundedness, simply aggregate corresponding properties of the individual nodes. It is recommendable to break down their verification to the individual node-based properties. That investigation can be organized as a portfolio of different verification techniques. For Petri nets with symmetric structure, we propose to apply symmetries as a separate portfolio member. This way of applying symmetries differs from the usual use of symmetries as a tool for state space reduction. We discuss some nice advantages of this approach.

Keywords

Petri net standard property, portfolio approach to verification, symmetry

1. Introduction

We can roughly distinguish three categories of Petri net verification. In *structural verification*, we detect patterns in the net topology itself. We may, for instance, enumerate siphons. A siphon is a set S of places such that every transition that has a post-place in S, also has a pre-place in S. [1] discusses an example where knowledge about siphons can directly be used for solving a relevant problem. A second category of verification is *model checking* [2]. Here, we decide whether the state space of a Petri net satisfies a property that is specified in some temporal logic. The third category is the *verification of global properties* (sometimes called standard properties). We consider a list of pre-defined properties and corresponding decision procedures. Examples of global properties are reversibility (can we reach the initial marking from every reachable marking), deadlock freedom (does every reachable marking enable at least one transition), k-boundedness (do all reachable markings have at most k tokens on every place), or liveness (does every reachable marking enable some transition sequence that contains all transitions). There exists a huge body of results in Petri net theory that involve global properties [3]. Recently, the Petri net model checking contest (MCC, [4]) added a separate competition on global properties.

We contribute to the third category of verification problems. In particular, we study global properties that aggregate properties of the individual places or transitions of the net. We call such a property an *aggregating property*. A place p is k-bounded if, for every reachable marking m, $m(p) \leq k$. The net is k-bounded if and only if every place is k-bounded. A transition t is live if every reachable marking enables a transition sequence that contains t. The net is live if and only if every transition is live. That is, properties such as k-boundedness and liveness are aggregating. In contrast, properties like reversibility and deadlock-freedom are not aggregating since no appearant separation into properties of individual places or transitions is known.

After setting up our terminology in Section 2, we argue in Section 3 that a node-by-node approach is recommendable for the verification of aggregating global properties. For every individual node, several methods exist to solve the node-based subproblem. These methods can be organized as a verification portfolio. We give examples of suitable portfolios for several aggregating properties. In Section 4, we recall the concept of symmetry. In Section 5, we propose to use symmetries as a separate portfolio member rather than a vehicle for state space reduction. We discuss the particular merits of this approach.

PNSE'24: International Workshop on Petri Nets and Software Engineering, 2024

karsten.wolf@uni-rostock.de (K. Wolf)

0009-0003-9877-6973 (K. Wolf)

2. Basic Terminology

We consider standard place/transition nets in the following notation.

Definition 1 (Petri net). *A* Petri net $N = [P, T, F, W, m_0]$ *consists of a finite set* P *of* places*, a finite set* T *of* transitions*, disjoint from* P*, a set* F *of* arcs *(*$F \subseteq (P \times T) \times (T \times P)$*), a* weight function $W : (P \times T) \times (T \times P) \to \mathbb{N}$ *where* $W(x, y) = 0$ *if and only if* $[x, y] \notin F$*, and a marking* m_0*, the* initial marking*. A* marking *is a mapping* $m : P \to \mathbb{N}$*.*

The behavior of a Petri net is defined as follows.

Definition 2 (Firing rule). *Transition* t *is* enabled *in marking* m *if, for all places* p*,* $[p, t] \in F$ *implies* $m(p) \geq W(p, t)$*. If* t *is enabled in* m*,* t *may* fire *and yield marking* m' *where, for all places* p*,* $m'(p) = m(p) - W(p, t) + W(t, p)$*. This* firing relation *is written as* $m \xrightarrow{t} m'$*.*

We lift relation $\xrightarrow{t}$ *to transition sequences by setting* $m \xrightarrow{\varepsilon} m$ *for all markings* m *and the empty sequence* ε*, and by setting* $m \xrightarrow{wt} m''$ *if and only if there is a marking* m' *such that* $m \xrightarrow{w} m'$ *and* $m' \xrightarrow{t} m''$*. Transition sequence* w *is* enabled *in marking* m *if there exists a marking* m *with* $m \xrightarrow{w} m'$*.*

We shall study the following properties of Petri nets.

Definition 3 (Properties).

- *Marking* m' *is* reachable *from marking* m *if there is a transition sequence* w *where* $m \xrightarrow{w} m'$*.*
- *Place* p *is* k*-bounded if, for all markings* m *reachable from* m_0*,* $m(p) \leq k$*. A Petri net is* k*-bounded if all its places are* k*-bounded.*
- *Place* p *is constant if, for all markings* m *reachable from* m_0*,* $m(p) = m_0(p)$*. A Petri net has the constant-place property if it contains a constant place.*
- *Transition* t *is* quasi-live *if there exists a transition sequence* w *such that* wt *is enabled in* m_0*. A Petri net is quasi-live if all its transitions are quasi-live.*
- *Transition* t *is* live *if, for every marking* m *reachable from* m_0*, there exists a transition sequence* w *such that* wt *is enabled in* m*. A Petri net is live if all its transitions are live.*

Observe that most properties listed above have a local version (place p is …, transition t is …) and an aggregating version (the Petri net is …). In the sequel, we shall therefore distinguish local standard properties and global standard properties. It is also obvious that answering the local problems for all nodes of a Petri net amounts to a solution of the corresponding global problem.

The global versions of 1-boundedness, constant place, quasi-liveness and liveness have been part of the "global properties" competition in recent model checking contests.

3. Verifying aggregating global properties

The brute-force approach to verify an aggregating global property is to compute the reachability graph of the Petri nets. The vertices of the graph are the markings reachable from m_0, and the edges correspond to the firing relation (considering only single transitions). k-boundedness and existence of constant places can be easily verified by inspecting the vertices of a reachability graph. Transition t is quasi-live if there is an edge labeled with t. Transition t is live if every terminal strongly connected component of the reachability graph contains an edge labeled with t. A strongly connected component is an equivalence class in the set of markings with respect to the equivalence relation "m is reachable from m' and m' is reachable from m". A strongly connected component is terminal if no other strongly connected component can be reached from it.

Although all standard properties can be analyzed using the reachability graph, its use generally is infeasible even if it is finite, due to the well-known state explosion problem. State space reduction

techniques may alleviate the state explosion problem to a certain degree, but work much better on the local version of a standard property than on the global version. This can be explained as follows. The most powerful state space reduction technique for Petri nets is the stubborn set method [5]. It observes that transitions that cannot enable or disable each other can be fired in various orders, and it tries to execute them in only some of the orders, yet taking care that the property under investigation is preserved. There exist stubborn set methods for the global versions of k-boundedness [6] or liveness [5]. However, related methods for the local versions are much stronger since the order of transitions that are not in the immediate vicinity of the considered place or transition are less sensitive to their respective order and permit stronger reduction. Another powerful state space reduction technique is the symmetry method. It also performs better on the local versions of problems. We postpone further discussion on the symmetry method to the next section. Apart from state space reduction, the local versions of standard properties permit the application of powerful net reduction in the whole net except the vicinity of the considered place or transition [3, 7, 8, 9].

For these reasons, our tool LoLA [10] verifies the global versions for standard properties by rigorously tracing them back to their local versions, applied to each place or transition separately. Since memory is more limited than run time in verification, a large number of small state spaces is easier to handle than just one large state space. In addition, our experience in the MCC tells us that, for most places or transitions, the local property can be easily proven for most places or transitions while only few places or transitions require complex computations. Altogether, even the sum of all efforts for verifying the local problems is typically smaller than approaching the global problem monolithically.

We can substantially reduce the costs of considering all places or all transitions of the net by considering *bycatch*. Bycatch means that, when approaching a local problem for one node, we may get sufficient information for answering the problem for other nodes. If, for instance, we search for an executable firing sequence that proves some transition to be quasi-live, all members of that sequence, and all members of sequences that have been produced in unsuccessful attempts, are clearly quasi-live. Many properties permit similar bycatch.

The effort for verifying the individual local problems can be reduced by running a portfolio approach that applies different verification techniques in parallel. In LoLA, the following verification techniques are used.

First, we may produce a *reduced state space* using the stubborn set method. There exist specialized approaches for all local problems mentioned so far. The article [6] discusses them in detail. A reduced state space produces a lot of bycatch. Every occurring transition t is quasi-live regardless of whether the state space was produced for t. Every place p with $m(p) > k$, for some computed marking m, is not k-bounded. Every place p with $m(p) \neq m_0(p)$, for some computed marking m, is not constant. Among the mentioned aggregating properties, the liveness problem permits virtually no bycatch. A transition is live if and only if it occurs in every terminal strongly connected component of the reachability graph. Due to stubborn set reduction, this works only for the investigated transition. For other transitions, neither presence nor absence in terminal components tell anything about their liveness. Due to this lack of bycatch, we added a single global verification task to our portfolio for liveness as an exception to our generally local approach. We compute a reduced state space where a version of stubborn sets is used that preserves all terminal strongly connected components of the original reachability graph. This reduced graph therefore preserves liveness of all transitions.

Random walks (i.e. repeated random execution of firing sequences) may provide information for some local problems. If some sequence contains a transition t, t is quasi-live. If such a walk enters a marking with more than k tokens on a place, this place in not k-bounded. If the number of tokens changes on a place, it is not constant. Using only random walks, we can, however, never be sure that a transition is not quasi-live. Similarly, random walks may disprove but never prove k-boundedness of a place and they can show that a place is non-constant. For liveness of a transition, random walk do not provide any useful information. We can restrict the set of transitions to choose from to a stubborn set that preserves some local verification problem. This way, we can significantly increase the success rate of the random walk approach. Random walks may also deliver some bycatch similar to state space generation.

A *place invariant* is a solution of $C^T \cdot \overline{y} = \overline{0}$ where C is the incidence matrix of the net holding

$C(p,t) = W(t,p) - W(p,t)$. If $\bar{i}$ is a place invariant, we have $\bar{i}^T \cdot m = \bar{i}^T \cdot m_0$, for all markings m reachable from m_0. The right hand side of this equation is a constant. Place invariants are mainly used for k-boundedness. If we find a place invariant $\bar{i}$ with $i(p) = 1$ and $i(q) \in \{0,1\}$ for all other places, such that $\bar{i}^T \cdot m_0 \leq k$ then p is k-bounded. As a bycatch, all other places q with $\bar{i}(q) > 0$ are k-bounded as well.

The *state equation* refers to the well-known fact that, for all markings m and m' and transition sequences w, $m + C \cdot \Psi(w) = m'$ where $\Psi(w)$ is the T-indexed Parikh vector of w. In the Parikh vector, $\Psi(w)(t)$ is the number of occurrences of t in w. Let $\overline{x}$ be a T-indexed vector of variables and $\overline{y}$ be a P-indexed vector of variables. If the integer linear program (LP) $m_0 + C \cdot \overline{x} = \overline{y}$, $\overline{x} \geq \overline{0}$, $\overline{y} \geq \overline{0}$, $x(t) \geq 1$ is infeasible then t is not quasi-live. If the system is feasible and $\overline{x}$ can be arranged into an executable transition sequence then t and all transitions t' with $x(t') > 0$ are quasi-live. If none of that happens, we have to rely on the other portfolio members. The state equation can be used for other local standard properties as well. For k-boundedness of place p, we replace constraint $x(t) \geq 1$ with $y(p) \geq k + 1$. For the constant place problem, we operate with two separate LP problems where we use $y(p) \geq m_0(p) + 1$ (resp. $y(p) \leq m_0(p) - 1$) instead. According to [9], one can further improve this approach by using an SMT-solver instead of an LP solver.

The considerations so far suggest the following portfolios. For k-boundedness, we run, for every place p, a reduced state space generation, a random walk task, a state equation task and a place invariant task. For quasi-liveness we run, for every transition t, a reduced state space generation, a random walk task, and a state equation task. For the constant place problem, we run reduced state space generation, a random walk task, and a state equation task for every place p. In fact, we run two copies of each task where one copy checks whether the number of tokens on p can be increased while the other one checks whether that number can be decreased. This separation simplifies both the stubborn set generation and the ILP problem to be studied. For liveness, we generate a global reduced state space that preserves all terminal strongly connected components and thus the liveness of all transitions. In addition, we generate separate reduced state spaces for each individual transition.

The portfolio manager takes care that the number of active tasks corresponds to the number of available cores. It also takes care that verification tasks are canceled as soon as some portfolio member was able to find the answer for the same place or transition.

4. Symmetries

Symmetries have been introduced in [11, 12] for high-level nets and later adapted to place/transition nets [13]. Their application is not at all restricted to Petri nets [14, 15, 16]. We follow the approach for place/transition nets. It considers a symmetry to be a graph automorphism, i.e. a bijection on the places and transitions that respects the node type, the arc relation, and the initial marking. A symmetry σ maps a marking to a marking $\sigma(m)$ such that the number of tokens in m on p coincides with the number of tokens on $\sigma(p)$ in $\sigma(m)$.

Definition 4 (Symmetry). *A* symmetry *is a bijection $\sigma : (P \cup T) \to (P \cup T)$ such that, for all nodes x and y and all places p,*

- *$\sigma(x) \in P$ if and only if $x \in P$ (respects node type);*
- *$W(x,y) = W(\sigma(x), \sigma(y))$ (respects arc relation and weights);*
- *$m_0(p) = m_0(\sigma(p))$ (respects initial marking).*

For a symmetry σ and a marking m, the marking $\sigma(m)$ is defined by $\sigma(m)(\sigma(p)) = m(p)$, for all places p.

The identity σ_{Id} with $\sigma(x) = x$, for all x, is always a symmetry. If σ is a symmetry, so is σ^{-1}, and for two symmetries σ_1 and σ_2, the composition $\sigma_1 \circ \sigma_2$ is a symmetry as well. That is, the set of all symmetries is a group under composition. A Petri net may have exponentially many symmetries which

can be represented by a generating set with a polynomial number of elements (in the number of nodes of the net) [17, 18].

Computing the generating set for the symmetries is closely related to the graph isomorphism problem. The graph isomorphism is in NP, and until now there is neither a known polynomial algorithm nor a proof of NP-completeness. Computing generators for the symmetry group amounts to a systematic search [19]. As long as every descent in the search tree ends in an actual generator, the runtime is polynomial. If, however, there are dead ends in the search tree that require backtracking without delivery of a generator, run time may become exponential. In the MCCs until 2021, there has been only one model where dead ends occur, For this model, they occur in every instance of that model. That is, the computation of the generating set is possible in polynomial time in most but not all cases. This observation accords to the complexity considerations above. Nevertheless, the absolute amount of required time for computing symmetries may be substantial even in those cases where it is polynomial in principle. The extreme example is a net in the MCC where the generating set for the symmetry has millions of elements and the total number of symmetries is beyond the numbers that can be expressed by double preecision floating point numbers in the C programming language.

This is why the use of symmetries is quite risky in situations where only limited time is available (such as the MCC). The computation of the generating set for the symmetries may take substantial time (as discussed above), and the use of large generating sets in reduced state space generation may also slow down the verification speed. For this reason, the most competitive configuration of LoLA in the MCC does not apply the symmetry reduction in its state space explorations. For state spaces in local verification problems, the use of symmetries for actual state space reduction is even more prohibitive since, for every individual node, the use of a specific subgroup of the symmetries may be required, as explained below.

We call two nodes x and y equivalent if there is a symmetry σ holding $\sigma(x) = y$. Accordingly, we call two markings m_1 and m_2 equivalent if there is a symmetry σ with $\sigma(m_1) = m_2$. Both relations are indeed equivalence relations. A symmetrically reduced reachability graph is permitted to omit markings if it can be guaranteed that at east one member of its equivalence class is contained. There are many ways to achieve this [18, 20]. The most popular implementation is to use the generating set to transform a considered marking into a "small" member of its equivalence class and to store and explore that one. "Small" usually refers to the lexicographic order on the marking vectors. This transformation can be achieved in polynomial time. The problem of transforming a marking into the smallest instead of some small member of its equivalence class is known to be equivalent to the graph isomorphism problem for which, as already mentioned, no polynomial solution is known. That is, a symmetrically reduced graph is often slightly larger than it could be in theory.

With the symmetrically reduced graph, we can approach some, but not all global versions of standard properties.

Proposition 1 (Preservation of standard properties under symmetry). *Place p is k-bounded if and only if the symmetrically reduced reachability graph does not contain a markimg m where, for some place q equivalent to p, $m(q) > k$.*

Place p is constant if and only if, for all markings m contained in the reachability graph and all places q equivalent to p, $m(q) = m_0(p)$.

Transition t is quasi-live if the symmetrically reduced reachability graph contains an arc annotated with some transition t' that is equivalent to t.

There is no straightforward way to exhaustively analyse liveness of all transitions using the symmetrically reduced graph. Remember that transition t is live if and only if it occurs in every terminal strongly connected component of the unreduced graph. If some transition t' equivalent to t occurs in some terminal strongly connected component of the symmetrically reduced graph, then the corresponding terminal strongly connected components in the unreduced graph may be the same for t and t', or may be different (just equivalent). That is why we cannot see whether all transitions occur in all terminal strongly connected components. Only if there is a terminal strongly connected component in the reduced graph that neither contains t nor any transition equivalent to t, we know that t is not live.

We can, however, compute a symmetrically reduced reachability graph to solve some particular local liveness problem [18]. If we want to study the liveness of t, we just need to use only symmetries σ with $\sigma(t) = t$. These symmetries form a subgroup of the group of all symmetries. This approach, however, requires computation of symmetries for every local problem and is therefore too expensive.

5. Using symmetries in a portfolio context

We are now proposing an alternative way to benefit from symmetries. We consider the following context. We are verifying an aggregating global standard property by separating it into its local problems, one for each place or transition. These local properties are tackled by a portfolio of verification methods, as discussed above. For the verification, a machine with multiple physical cores is available.

In this context, we may add another thread that runs concurrently to the remaining portfolio. It computes a generating set for the symmetries. Whenever some generator has been found, the equivalence relation on places and transitions is updated. Whenever some portfolio member reports a result for some place or transitions, this result is immediately propagated to all equivalent places or transitions.

This approach has some decisive advantages, compared to state space reduction using symmetries. First, the actual portfolio can start to do its work and does not need to wait until the generating set for the symmetries has been computed. That is, we do not lose too much time for getting the actual verification done. In the worst case, only the computing power of one core is siphoned away from the actual verification. Second, every single generator allows us to reason about equivalent nodes. So symmetries start to take effect long before the computation of the generating set has finished. Third, after having used a generator to update equivalences, the generator does not need to be stored permanently (which is the case for state space reduction as the generators are applied for transforming markings into their small representatives). Fourth, if symmetries are detected, we may skip a large number of local verification tasks since values can be inferred from equivalent nodes. Our tool LoLA has recently been quite successful in the verification of global properties, and we blame the combination of local property verification and our novel symmetry application for that.

For organizing the integration of the symmetries into the regular portfolio, the portfolio manager of our tool has implemented the following communication.

The portfolio manager records the local results and checks whether the global result is determined. Taking quasi-liveness as an example, the global result is fixed if one transition is reported not to be quasi-live, or if all transitions have been reported to be quasi-live. The portfolio manager additionally maintains the equivalence relation. To this end, it uses the data structure of Tarjan's union/find algorithm [21] that handles equivalence relations in the most efficient way. It starts with singleton equivalence classes for all nodes. Whenever a symmetry pours in, it unifies the class containing x with the class containing $\sigma(x)$, for all nodes x. Depending on the nature of the property, it only maintains the place classes or only the transition classes.

The regular tasks ask the portfolio manager for an unsolved node they can process (and have not processed yet), and finally deliver their result (true, false, or inconclusive). That result is then propagated to all nodes that are equivalent to the reported one. The symmetry task is started at the beginning. Whenever it finds a generator of the symmetry group, it reports this generator to the portfolio manager for updating the equivalences. If equivalence classes change, known results are propagated to the new members of the equivalence class. Using this strategy, we can evaluate every global property for the n dining philosophers problem with $5n$ places and $4n$ transitions by solving only up to 5 local problems. This is a dramatic speed up, especially for large n.

6. Conclusion

We studied the verification of aggregating global properties. The proposed approach has a few similarities to state-of-the-art model checking for Petri nets. For both types of verification tasks, we run a portfolio

offering various verification methods, including methods that strongly benefit from Petri net structure theory. For both types of verification problems, we try to separate the problem into as many as possible subproblems such that solving the subproblems requires less memory.

There are, however, remarkable differences between model checking and the verification of global properties. The first difference is the sheer number of subproblems we end up with. In model checking, we may be able to rewrite the formula under investigation into a Boolean combination of a few sub-formulas and verify them separately. An aggregating global property can be split into as many subproblems as we have places or transitions in the net. For this reason, we need to squeeze out as much bycatch as possible from the solved subproblems. While the subproblems in model checking tend to be heterogeneous, subproblems for global properties are homogeneous. That is why it is possible to apply symmetries in a different manner. Instead of using them for state space reduction, we may apply them to generate additional bycatch by propagating results from solved problem instances to unsolved problem instances.

References

[1] M. Heiner, R. Donaldson, D. Gilbert, Petri Nets for Systems Biology, in Iyengar, M.S. (ed.), Symbolic Systems Biology: Theory and Methods; Chapter 21, 2010.

[2] E. M. Clarke, O. Grumberg, D. A. Peled, Model checking, MIT Press, London, Cambridge, 1999.

[3] T. Murata, Petri nets: Properties, analysis and applications, Proc. IEEE 77 (1989) 541–580. URL: https://doi.org/10.1109/5.24143. doi:10.1109/5.24143.

[4] E. G. Amparore, B. Berthomieu, G. Ciardo, S. Dal-Zilio, F. Gallà, L. Hillah, F. Hulin-Hubard, P. G. Jensen, L. Jezequel, F. Kordon, D. L. Botlan, T. Liebke, J. Meijer, A. S. Miner, E. Paviot-Adet, J. Srba, Y. Thierry-Mieg, T. van Dijk, K. Wolf, Presentation of the 9th edition of the model checking contest, in: D. Beyer, M. Huisman, F. Kordon, B. Steffen (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III, volume 11429 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 50–68. URL: https://doi.org/10.1007/978-3-030-17502-3_4. doi:10.1007/978-3-030-17502-3_4.

[5] A. Valmari, The state explosion problem, in: W. Reisig, G. Rozenberg (Eds.), Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996, volume 1491 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 429–528. URL: https://doi.org/10.1007/3-540-65306-6_21. doi:10.1007/3-540-65306-6_21.

[6] K. Schmidt, Stubborn sets for standard properties, in: S. Donatelli, H. C. M. Kleijn (Eds.), Application and Theory of Petri Nets 1999, 20th International Conference, ICATPN '99, Williamsburg, Virginia, USA, June 21-25, 1999, Proceedings, volume 1639 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 46–65. URL: https://doi.org/10.1007/3-540-48745-X_4. doi:10.1007/3-540-48745-X_4.

[7] G. Berthelot, G. Roucairol, Reduction of Petri-nets, in: A. W. Mazurkiewicz (Ed.), Mathematical Foundations of Computer Science 1976, 5th Symposium, Gdansk, Poland, September 6-10, 1976, Proceedings, volume 45 of *Lecture Notes in Computer Science*, Springer, 1976, pp. 202–209. URL: https://doi.org/10.1007/3-540-07854-1_175. doi:10.1007/3-540-07854-1_175.

[8] F. M. Bønneland, J. Dyhr, P. G. Jensen, M. Johannsen, J. Srba, Stubborn versus structural reductions for Petri nets, J. Log. Algebraic Methods Program. 102 (2019) 46–63. URL: https://doi.org/10.1016/j.jlamp.2018.09.002. doi:10.1016/J.JLAMP.2018.09.002.

[9] Y. Thierry-Mieg, Structural reductions revisited, in: R. Janicki, N. Sidorova, T. Chatain (Eds.), Application and Theory of Petri Nets and Concurrency - 41st International Conference, PETRI NETS 2020, Paris, France, June 24-25, 2020, Proceedings, volume 12152 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 303–323. URL: https://doi.org/10.1007/978-3-030-51831-8_15. doi:10.1007/978-3-030-51831-8_15.

[10] K. Wolf, Petri net model checking with LoLA 2, in: V. Khomenko, O. H. Roux (Eds.), Application and Theory of Petri Nets and Concurrency - 39th International Conference, PETRI NETS 2018, Bratislava, Slovakia, June 24-29, 2018, Proceedings, volume 10877 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 351–362. URL: https://doi.org/10.1007/978-3-319-91268-4_18. doi:10.1007/978-3-319-91268-4_18.

[11] E. A. Emerson, A. P. Sistla, Symmetry and model checking, Formal Methods Syst. Des. 9 (1996) 105–131. URL: https://doi.org/10.1007/BF00625970. doi:10.1007/BF00625970.

[12] K. Jensen, Condensed state spaces for symmetrical coloured Petri nets, Formal Methods Syst. Des. 9 (1996) 7–40. URL: https://doi.org/10.1007/BF00625967. doi:10.1007/BF00625967.

[13] P. H. Starke, Analyse von Petri-Netz-Modellen, Leitfäden und Monographien der Informatik, Teubner, 1990.

[14] E. M. Clarke, S. Jha, R. Enders, T. Filkorn, Exploiting symmetry in temporal logic model checking, Formal Methods Syst. Des. 9 (1996) 77–104. URL: https://doi.org/10.1007/BF00625969. doi:10.1007/BF00625969.

[15] C. N. Ip, D. L. Dill, Better verification through symmetry, Formal Methods Syst. Des. 9 (1996) 41–75. URL: https://doi.org/10.1007/BF00625968. doi:10.1007/BF00625968.

[16] E. A. Emerson, A. P. Sistla, Symmetry and model checking, Formal Methods Syst. Des. 9 (1996) 105–131. URL: https://doi.org/10.1007/BF00625970. doi:10.1007/BF00625970.

[17] D. E. Knuth, Efficient representation of perm groups, Comb. 11 (1991) 33–43. URL: https://doi.org/10.1007/BF01375471. doi:10.1007/BF01375471.

[18] K. Schmidt, Integrating low level symmetries into reachability analysis, in: S. Graf, M. I. Schwartzbach (Eds.), Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings, volume 1785 of *Lecture Notes in Computer Science*, Springer, 2000, pp. 315–330. URL: https://doi.org/10.1007/3-540-46419-0_22. doi:10.1007/3-540-46419-0_22.

[19] K. Schmidt, How to calculate symmetries of Petri nets, Acta Informatica 36 (2000) 545–590. URL: https://doi.org/10.1007/s002360050002. doi:10.1007/S002360050002.

[20] T. A. Junttila, Computational complexity of the place/transition-net symmetry reduction method, J. Univers. Comput. Sci. 7 (2001) 307–326. URL: https://doi.org/10.3217/jucs-007-04-0307. doi:10.3217/JUCS-007-04-0307.

[21] R. E. Tarjan, J. van Leeuwen, Worst-case analysis of set union algorithms, J. ACM 31 (1984) 245–281. URL: https://doi.org/10.1145/62.2160. doi:10.1145/62.2160.