Modular Model Checking of Reference Nets: MoMoC

Sven Willrodt, Daniel Moldt, and Michael Simon

University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences,
Department of Informatics http://www.informatik.uni-hamburg.de/TGI/

Abstract Reference nets provide a high expressiveness for modeling. However, like other high-level Petri nets, they are hard to analyze. In this contribution, we present our model checking tool for Reference nets: Modular Model Checker (MoMoC). Its two main purposes are to provide a proof of concept and to lay the foundation for a modular framework for Reference net model checking and verification in general.
To allow CTL model checking, novel atomic propositions are added to the specification language that cover the unique Reference net concepts. For its specific main purpose, the application in the context of teaching, we provide a dynamic coloration of the reachability graph according to results of the extended CTL algorithm.
MoMoC is the first working tool in this area. The main emphasis is on the overall architecture as our intention is to extend this framework in the future to cover more features and to become more efficient.

Keywords: High-level Petri nets, Nets-within-nets, Reference nets, Tool, CTL Model Checking

1 Introduction

Low-level Petri nets have a large repertoire of methods like partial order reduction [30], symmetries [25], the sweep-line method [6] or alternative ways to represent the state space [10,22] to cope with large model sizes. However, for high-level Petri nets, research is still needed. Information about tools and links to Petri net topics can be found at http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/ and http://www.petrinet.de.

The lack of verification methods holds especially for Reference nets with their special features of synchronous channels and the nets-within-nets concept. We consider model checking as a subdiscipline to be a strong candidate for the verification of Reference nets.

An idea that we have worked on for quite a while now is the partitioning of the whole Reference net system into its sub-nets. It is a central hypothesis that

a partitioning of the net system based on the nets-within-nets concept will aid in developing efficient model checking methods. Therefore, we developed concepts, methods and tools. In this contribution we present our current prototype for a modular model checking framework called Modular Model Checker (MoMoC). A first step was to use MoMoC for our theory teaching lessons in bachelor courses which was performed successfully this year.

The overarching goal is to provide a flexible framework that can easily be modified to allow for experiments with different kinds of verification techniques. A module structure provides the necessary flexibility. Modules are largely independent from each other and can be individually optimized and exchanged for a specific purpose. The current prototype implements explicit CTL model checking. As a basis, we use the reachability graph generation functionality from [27]. The reachability graph is a central means for the model checking and was developed as the first step. Therefore, the Reference net reachability graph generation is also discussed in this contribution.

In the following we briefly present the basic notions in Section 2. The main goal and the purpose of MoMoC as well as our requirements for and features of this prototype are covered in Section 3. The architecture and some basic behavior are sketched in Section 4. Section 5 demonstrates the usage of MoMoC by a simple example, before we conclude in Section 6. Several text passages in this contribution originate from [31], which provided first insights.

2 Background

Inspired by the nets-within-nets formalism of [29], the idea of object- and agent-oriented Petri nets [4,20] and object-oriented programming in general, Reference nets as a modeling technique and Renew as the accompanying tool have been developed [16,17].

Reference nets allow the use of net instances as tokens and recognize Java expressions as inscriptions. Templates can be considered as classes and instances as objects. Each net instance can technically be replaced by an object and vice versa. Communication between net instances is realized via synchronous channels [5,16]. Due to the dynamic instantiation of net instance tokens, markings may contain an arbitrary number of instances of each net template.

Although Renew is developed by our group since the end of the 90s, verification has only been addressed for simple versions of nets and only small studies have been performed. Tools like Maria [18], LoLA [32], GreatSPN [1], Maude [8] and others have been (partially) integrated to provide verification options for traditional net variants like P/T nets. For high-level Petri nets several interesting model checking tools exist that are not integrated [9,11,14]. However, the mapping from Renew's Reference net formalism to a formalism that can be read by these tools poses a very difficult problem in itself. The unique Reference net concepts cannot be easily translated to other CPN concepts and it is only feasible to translate a subset of all possible Java inscriptions to other inscription languages. Therefore, we pursue a different model checking approach based on

the exploration of reachable markings using Renew's own simulation core. A central benefit is that this approach retains all of the Reference net formalism's expressive power.

Model checking is a method to verify systems that are modeled as state-transition systems [3,7]. With their capability to express concurrency and distribution, Petri nets are a popular modeling language for which model checking is an appropriate method [12]. Every year, the Model Checking Contest [15] is held, where tools can compete in verifying Petri nets. There, highly optimized tools are tested which use efficient algorithms to perform different kinds of model checking. However, there is no tool for formalisms where nets can contain other marked nets as tokens.

Due to the unique concepts of the Reference net formalism, the reachability graph is constructed and presented differently by MoMoC compared to other tools for Coloured Petri nets, e.g. CPN/Tools. The reachability graph is explored based on a single *root net instance* in the initial marking of the *root net template*. In general a marking in the reachability graph is always associated with a single root net instance state. However, this state can recursively contain other marked net instances. The state of the overall system can be explored by traversing the contained net instances in the UI. A working prototype, the Reference Net Reachability Graph (Rnrg) plugin, is presented in [27].

The Rnrg plugin is a reachability graph generator prototype for Reference nets. Its primary design philosophy is the minimalism of the implementation. MoMoC has a different design philosophy: the central aspects are modularity and extensibility of the implementation. We took the reachability graph generation functionality from the Rnrg plugin, broke it apart into flexible components and added the architecture for the integration of different model checking methods as modules. Thus, the MoMoC plugin contains all of the Rnrg plugin's functionality and builds upon it. We consider it as its successor. The original Rnrg functionality can still be accessed as a procedure module. Working model checking functionality is also already provided in the form of a CTL procedure module. It is a proof-of-concept of this architecture and a prototype for future model checking methods. Furthermore, MoMoC adds visualization functionality that is essential for its practical usability and deployment in teaching.

In the future, it is of particular interest to develop optimized model checking methods that make use of the concepts that distinguish Reference nets from other CPN formalisms and to combine these with optimizations that are known from the other formalisms. This version of MoMoC does not offer this. However, the architecture is build with the integration of such methods in mind and is an essential step into this direction.

3 Objectives

The overall objective is to improve the tool support for Reference net verification. To provide an extendable basis for research on model checking, the first step is to

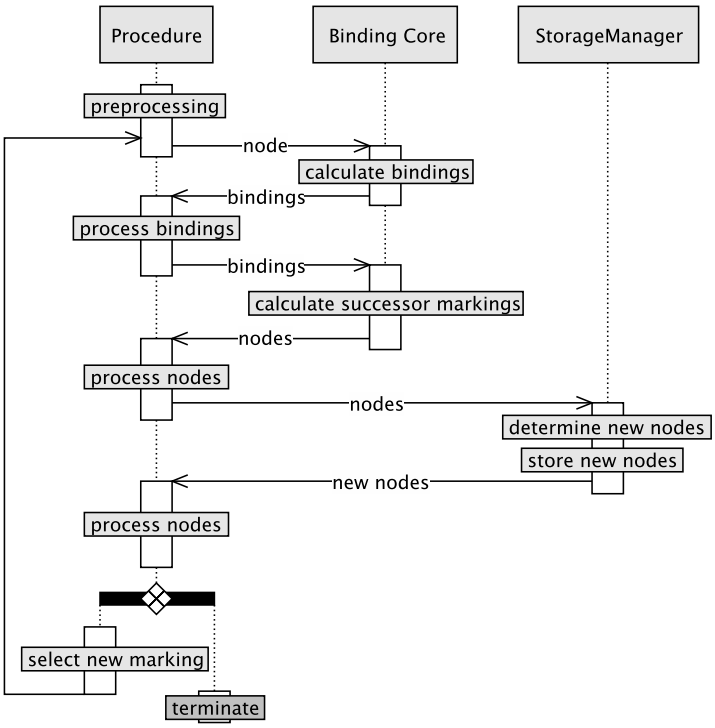

Figure 1. An exemplary interaction between a procedure, binding core and storage manager module

change RNRG from a single purpose, monolithic tool to a modular environment that allows quick prototyping.

The second objective is to implement the basic CTL model checking algorithm [3,7]. It serves both as a foundation for future research and tools, as well as a prototype to identify the difficulties and bottlenecks that need to be handled when model checking Reference nets. Based upon this prototype, more efficient model checking techniques can be developed that utilize the advantages of the Reference net formalism, like the distributed structure of net templates.

Further, with a working model checking routine, MOMOC can be used for teaching purposes in our bachelors theory course, which includes Petri net model checking. Because the examples are relatively small in size, performance is not essential. Rather, the focus shifts to result visualization and general feedback, helping students to gain insight into the CTL model checking algorithm.

4 Architecture

MoMoC features a modular architecture (thus its name), where the processing of a query is determined by mainly three types of modules that can freely be exchanged to alter the behavior; *binding cores*, *storage managers* and *procedures*. These three module types directly result from the modularization of the Rnrg plugin. The CTL model checking implementation is fully contained in a single procedure. Additionally, MoMoC provides a framework for common tasks like parsing, result visualization and user interaction.

The *binding core* finds bindings for a given marking and can calculate the resulting markings. Thus, it represents the formalism that is being verified. It can be set up to exclude certain types of transitions, e.g. transitions with inscriptions that have side-effects. For the majority of potential extensions we expect that it is not necessary to modify the binding core, however it is possible to integrate new formalisms that way. The binding core is realized by integrating Renew's simulator, which is further described in [27].

Storage managers keep track of found markings. They store the reachability graph, insert new nodes into it and detect duplicate markings. Storage managers are of special interest, because it is not trivial to store Reference nets efficiently. By exchanging the storage manager, it is possible to test new data structures that encode markings or storage heuristics. Techniques such as the sweepline method, symmetry and bloomfiltering are examples that can be implemented as storage managers.

Procedures contain the logic and steps to process a query. They define the overall purpose of the query and can be considered as the active components which utilize all other modules. The generation of a reachability graph, CTL model checking and model checking with partial order reduction are examples of procedures. Procedures can also use other procedures, to prevent basic algorithms from being implemented twice. A good example is the reachability graph procedure, originally from Rnrg, which generates the reachability graph for the explicit CTL model checking procedure. Optimization techniques that select which transitions are fired, like the stubborn set method or a depth-first search, need to be realized as a procedure.

Figure 1 shows a typical interaction between the three described modules on an abstract level. It is similar to how MoMoC generates the reachability graph, but entirely different interactions are imaginable.

To provide tasks that resolve around parsing, MoMoC uses the parser generator ANTLR [21]. This includes the parsing itself, the normalization of formulas, as well as the resolution of redundancies. It further provides a simple encoding of labels for the CTL model checking algorithm.

Lastly, the framework provides a UI that has multiple features. It presents the user available procedures as well as optional parameters that are defined by the procedure. The UI can take multiple formulas at once and it is up to the procedure to decide on how to process them. Currently, the CTL model checking algorithm constructs a single reachability graph and then verifies all formulas on it, reusing the labels.

Complementing the UI, the *result visualizer* takes on the task of presenting the outcome of a query to the user. It dynamically displays available results from a procedure. Besides a list of arguments that the procedure returned, the user can also inspect a hierarchical representation of the normalized formula. In addition, for label-based approaches like CTL model checking, the reachability graph can be colorized to show which nodes fulfill the currently selected (sub-)formula from the hierarchical view.

Language Features

When formulating atomic propositions for Reference nets one faces the problem that net elements in net instances are not uniquely identifiable by their name and net template. This is based on the fact that an arbitrary number of net instances of the same template may exist during simulation. We call this problem *net instance ambiguity*. To approach this problem, MoMoC introduces two novel operators, called *net instance quantifiers*. As their name suggests, they quantify over net instances and require either all, or at least one net instance of a given type to fulfill a marking predicate.

With a given net template name *net* and a marking predicate m, the term $!(net, m)$ expresses that *all* net instances of *net* fulfill m, and $?(net, m)$ that *at least one* net instance of *net* fulfills m. Expressions with net instance quantifiers are still atomic propositions and thus do not change syntax or semantics of CTL. For nets that only consist of one net instance (like P/T nets or CPNs), both operators effectively result in the semantics familiar from other tools for those Petri net formalisms.

While net instance quantifiers are a first solution to the net instance ambiguity, many properties remain that cannot be expressed with them.

5 Examples and Evaluation

This section briefly demonstrates the usage of MoMoC. For a detailed description and an executable demo see https://paose.informatik.uni-hamburg.de/paose/wiki/MoMoC.

Figure 2 is a Reference net representation of a sender-receiver system with a buffer. There exist four different net templates, of which net instances are created for simulation; a *sender* that can select and send a message, a *receiver* that can receive a message, a *buffer* which can receive and store up to two messages that it can then send on, as well as a system net that serves as a platform for communication between all net instances. All net instances communicate via synchronous channels. In this example, channels exist such that messages can be transmitted either directly between sender and receiver (channels with *direct*), or by using the buffer (channels with *transmit* and *fetch*). This Reference net representation of the system is very easy to scale, as the amount of each net instance can be increased via the inscription in the system net. It is thus possible to add more sender, receiver or buffer nets without adding any new places or

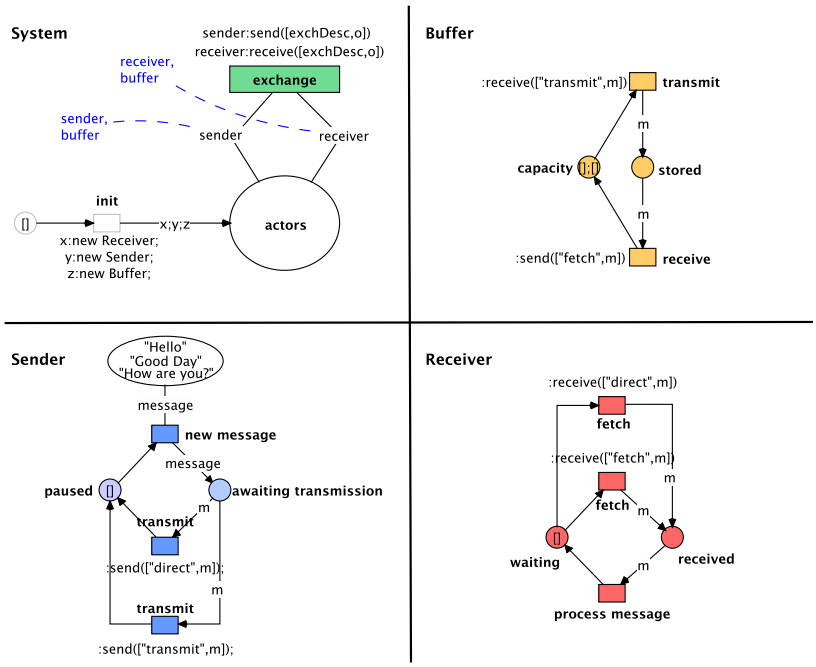

Figure 2. Sender-Receiver with Buffer

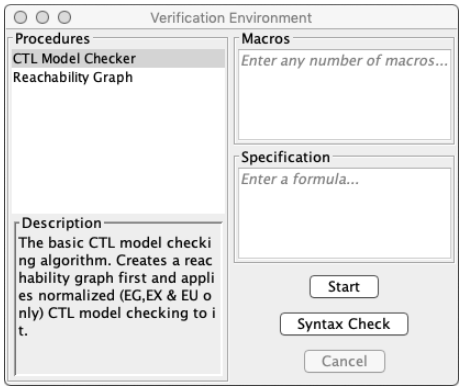

Figure 3. MoMoC's UI

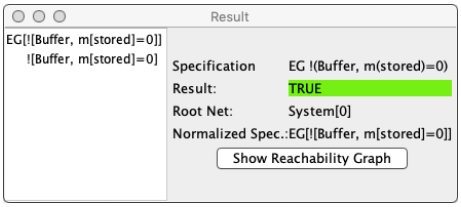

Figure 4. Result panel

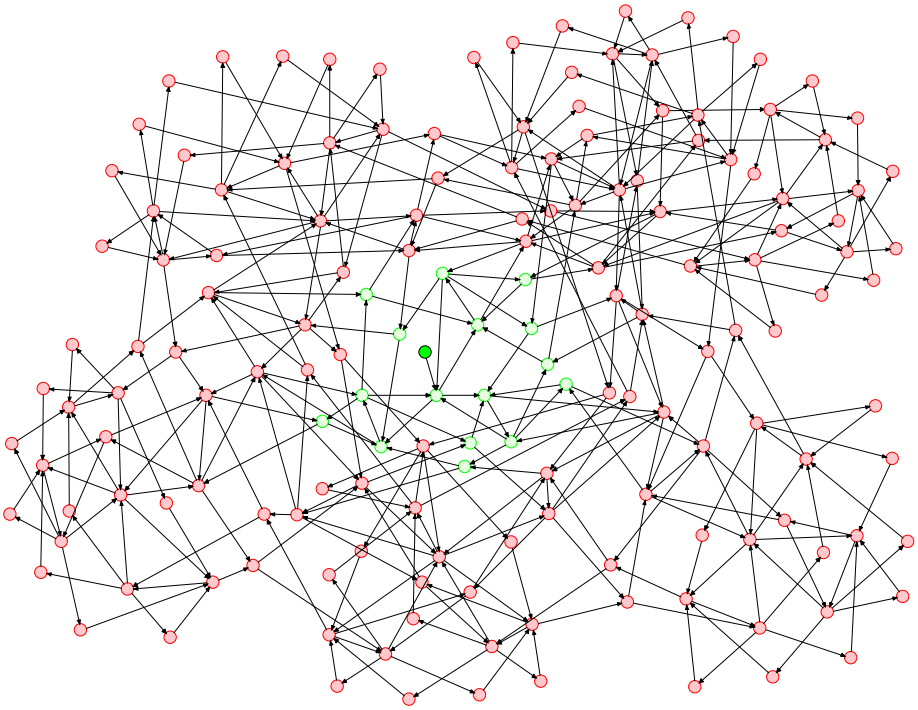

Figure 5. Reachability graph, colored with $EG!(\text{Buffer}, m(\text{stored}) = 0)$

transitions. The depicted system net in Figure 2 instantiates exactly one instance of each net template.

In the following, the CTL formula $EG!(\mathit{Buffer}, m(\mathit{stored}) = 0)$ is used to prove that communication does not necessarily include the buffer. Using the net instance quantifier !, the formula expresses that there exists a path, on which all instances of the net template *Buffer* have zero tokens in the place *stored*. Since the predicate is required for all instances, the formula is independent of the number of instantiated *Buffer* nets.

Figure 3 shows MoMoC's UI. The left panel displays all available procedures and a short description. The right panel receives user input. A formula can be entered in the lower text area, and macros can be defined in the upper text area for better readability. From top to bottom, the three buttons start the selected procedure, perform a syntax check and terminate all ongoing procedures.

Figure 4 presents the result of the query. Besides the overall result, it displays additional information like the root net name. The left panel displays the normalized formula in a hierarchical view. Additionally, the result panel has a button to display the reachability graph. The first time it is clicked, the graph layout algorithm is triggered, which currently takes significantly longer than the actual model checking of the query.

The reachability graph in Figure 5 was arranged automatically. Edge labels are hidden for a better readability. Each node is colored green or red, representing if the formula $EG!(\mathit{Buffer}, m(\mathit{stored}) = 0)$ holds in this node. The initial marking is represented by a strong green node (red in a negative case). Any path on only green nodes constitutes an example path that verifies the given formula.

The state space in Figure 5 consists of 161 nodes and 367 edges. Using an Intel i7-8550U CPU, the state space generation and model checking took an average of 0.24 seconds, not including the graph layouting. When adding two more *Buffer* net instances, the state space increases to 3521 nodes and 12277 edges, for which MoMoC requires 4.8 seconds on average. At state spaces of this size, bottlenecks in the current algorithms of MoMoC become evident. A main bottleneck is the currently inefficient encoding of Reference nets, but also the lack of optimization techniques requires all states to be explored and kept in memory. The addition of such techniques and a more sophisticated encoding will further increase the efficiency of MoMoC. For models that we currently use in teaching, the efficiency is already sufficient and the dynamic graph colorization is a helpful feature to understand the CTL model checking approach.

6 Conclusion

MoMoC provides a foundation for model checking in Renew and allows explicit CTL model checking of Reference nets and other Petri net formalisms that can be expressed with Reference nets, like P/T nets or CPNs. While its performance cannot compare to tools that participate in the Model Checking Contest at verifying classic Petri net formalisms, MoMoC is the first tool that can apply model checking to the Reference net formalism. With its modular structure, components can be independently exchanged to create new behavior. The procedure module can be exchanged to apply partial order reduction or similar techniques during the state space generation. By exchanging the storage manager, data structures and storage techniques can be altered. The binding core can be exchanged to effectively replace the semantics of the underlying Petri net. Thus, MoMoC provides a well-extendable basis to conduct further research on model checking algorithms and data structures designed for Reference nets.

For its main purpose, the comprehensive visualization of results, the reachability graph can be colorized interactively for each subformula with the result of the CTL model checking algorithm. With this feature, the framework is well suited to teach the basics of Petri net model checking with CTL. MoMoC has been successfully used in a bachelor theory course for third and fifth semester students this winter term. It was both used during lecture for demonstrations and in exercises for students.

Outlook

Several improvements are planned for MoMoC in the short term. In addition to a colorization of nodes, a trace representation is planned to display positive

or negative example traces. More atomic propositions are planned to increase the expressiveness with respect to data types and net instances. Especially net instance expressions are an interesting research topic as they are a balancing act between expressiveness and computational complexity of their evaluation. Finally, a more efficient encoding of markings can significantly increase the performance of MoMoC.

An interesting research topic is the combination of CPN model checking and the verification of inscribed code. A promising approach is to use MoMoC with the *Curry-Coloured Petri Net* (*CCPN*) formalism [26,28]. Especially when considering future extensions integrating hierarchical nets and Reference net concepts. The CCPN framework integrates the purely functional logic programming language Curry as an inscription language. It is side-effect free, has a formally defined semantics and thus offers great potential for code verification as evidenced by existing research [2,13]. This verification would make model checking possible on markings that are parametric on some tokens. This would be particularly useful to provide results that are valid for a whole range of external Reference net channel calls at once.

As another important task besides the transfer of algorithms and methods from traditional Petri net analysis, we plan to increase the efficiency and thus the practical feasibility of Reference net analysis in two steps: the first step is to use the pragmatic structure of the model which is build in by the modeler. The nets-within-nets concept supports object- or agent-oriented modeling which can be used to partition the models. It is a current research topic to investigate how net templates that follow those modeling paradigms can be verified independently. This will greatly relieve the model checker and allows the application of different techniques for each net template. The second step is to distribute the analysis, based on this partitioning, for which we can utilize our research on Kubernetes in the context of Renew [23,24]. Special treatment of the composition is required by algorithms and methods that are the object of further research.

Due to the modular architecture, MoMoC will continue to improve our teaching: other verification techniques can easily be added (like in [19]), the visualization of relations between components in a software system (modeled as net instances) is possible and students can experiment with verification algorithms during our teaching project courses to gain deeper insights into software architecture, verification and algorithms at the same time.

In the context of teaching it is also a topic of interest to increase the readability of large reachability graphs. Central to this is a dynamic visibility of states in the reachability graph, so that only relevant parts are displayed. A suitable approach would be to display only traces that verify or disprove a formula and surrounding states to give enough context. With sophisticated techniques to comprehensively display larger reachability graphs, MoMoC will also become more interesting for real world applications.

References

1. Amparore, E.G., Balbo, G., Beccuti, M., Donatelli, S., Franceschinis, G.: 30 years of GreatSPN. In: Principles of Performance and Reliability Modeling and Evaluation, pp. 227–254. Springer (2016)
2. Antoy, S., Hanus, M., Libby, S.: Proving non-deterministic computations in Agda. In: WFLP 2016, Proceedings. Electronic Proceedings in Theoretical Computer Science, vol. 234, pp. 180–195. Open Publishing Association (2017)
3. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press (2008)
4. Becker, U., Moldt, D.: Object-oriented concepts for coloured Petri nets. In: Systems, Man and Cybernetics, 1993. 'Systems Engineering in the Service of Humans', Conference Proceedings., IEEE, International Conference on. vol. 3, pp. 279–285. IEEE, Le Touquet, France (Oct 1993). https://doi.org/10.1109/ICSMC.1993.385024
5. Christensen, S., Hansen, N.D.: Coloured petri nets extended with channels for synchronous communication. In: Valette, R. (ed.) Petri Nets 1994. LNCS, vol. 815, pp. 159–178. Springer (1994). https://doi.org/10.1007/3-540-58152-9_10
6. Christensen, S., Kristensen, L.M., Mailund, T.: A sweep-line method for state space exploration. In: Margaria, T., Yi, W. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 450–464. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
7. Clarke, Jr., E.M., Grumberg, O., Kroening, D., Peled, D.A., Veith, H.: Model Checking. MIT Press, Cambridge, MA, USA, third edn. (2018)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L. (eds.): All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic, LNCS, vol. 4350. Springer (2007). https://doi.org/10.1007/978-3-540-71999-1
9. Colange, M., Baarir, S., Kordon, F., Thierry-Mieg, Y.: Crocodile: A Symbolic/Symbolic Tool for the Analysis of Symmetric Nets with Bag. In: Kristensen, L.M., Petrucci, L. (eds.) Applications and Theory of Petri Nets. pp. 338–347. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21834-7_20
10. Esparza, J., Heljanko, K.: Unfoldings: a partial-order approach to model checking. Springer Science & Business Media (2008)
11. Evangelista, S.: High Level Petri Nets Analysis with Helena. In: Ciardo, G., Darondeau, P. (eds.) Applications and Theory of Petri Nets 2005. Lecture Notes in Computer Science, vol. 3536, pp. 455–464. Springer, Berlin, Heidelberg (2005). https://doi.org/10.1007/11494744_26
12. Girault, C., Valk, R.: Petri nets for systems engineering: a guide to modeling, verification, and applications. Springer Science & Business Media (2013)
13. Hanus, M.: Combining static and dynamic contract checking for Curry. In: Fioravanti, F., Gallagher, J.P. (eds.) Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017, Namur, Belgium, October 10-12, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10855, pp. 323–340. Springer (2017)
14. Hostettler, S., Marechal, A., Linard, A., Risoldi, M., Buchs, D.: High-level Petri net model checking with AlPiNA. Fundamenta Informaticae **113**(3-4), 229–264 (Aug 2011)
15. Kordon, F., Garavel, H., Hillah, L.M., Hulin-Hubard, F., Amparore, E., Beccuti, M., Berthomieu, B., Ciardo, G., Dal Zilio, S., Liebke, T., Li, S., Meijer, J., Miner,

A., Srba, J., Thierry-Mieg, Y., van de Pol, J., van Dirk, T., Wolf, K.: Complete Results for the 2019 Edition of the Model Checking Contest. http://mcc.lip6.fr/ (Apr 2019)
16. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002), http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=0035&lng=eng&id=
17. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L., Haustermann, M., Mosteller, D.: Renew – User Guide (Release 2.5). University of Hamburg, Faculty of Informatics, Theoretical Foundations Group, Hamburg (Jun 2016), http://www.renew.de/
18. Mäkelä, M.: Maria: Modular reachability analyser for algebraic system nets. In: Esparza, J., Lakos, C. (eds.) Petri Nets 2002. LNCS, vol. 2360, pp. 434–444. Springer, Springer (2002). https://doi.org/10.1007/3-540-48068-4_25
19. Mascheroni, M., Wagner, T., Wüstenberg, L.: Verifying reference nets by means of hypernets: A plugin for Renew. In: Duvigneau, M., Moldt, D. (eds.) PNSE'10 2010, Proceedings. pp. 39–54. No. FBI-HH-B-294/10 in Bericht, Univerity of Hamburg (Jun 2010), http://epub.sub.uni-hamburg.de/informatik/volltexte/2010/148/
20. Moldt, D., Wienberg, F.: Multi-agent-systems based on coloured Petri nets. In: Azéma, P., Balbo, G. (eds.) Petri Nets 1997, Proceedings. pp. 82–101. No. 1248 in LNCS, Springer Verlag, Berlin Heidelberg New York (1997)
21. Parr, T.: The definitive ANTLR 4 reference. Pragmatic Bookshelf (2013)
22. Pastor, E., Roig, O., Cortadella, J., Badia, R.M.: Petri net analysis using boolean manipulation. In: Valette, R. (ed.) Petri Nets 1994. pp. 416–435. Springer Berlin Heidelberg (1994)
23. Röwekamp, J.H., Feldmann, M., Moldt, D., Simon, M.: Simulating Place / Transition Nets by a distributed, web based, stateless service. In: Moldt, D., Kindler, E., Wimmer, M. (eds.) PNSE'19, Proceedings. CEUR Workshop Proceedings, vol. 2424, pp. 163–164. CEUR-WS.org (2019), http://CEUR-WS.org/Vol-2424
24. Röwekamp, J.H., Moldt, D.: RenewKube: Reference net simulation scaling with Renew and Kubernetes. In: Donatelli, S., Haar, S. (eds.) Petri Nets2019. LNCS, vol. 11522, pp. 69–79. Springer (2019). https://doi.org/10.1007/978-3-030-21571-2_4
25. Schmidt, K.: Symmetries of Petri Nets. Citeseer (1994)
26. Simon, M.: Curry-Coloured Petri Nets: A Concurrent Simulator for Petri Nets with Purely Functional Logic Program Inscriptions. Master thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (Apr 2018)
27. Simon, M., Moldt, D., Engelhardt, H., Willrodt, S.: A first prototype for the visualization of the reachability graph of reference nets. In: Moldt, D., Kindler, E., Wimmer, M. (eds.) PNSE'19 2019, Proceedings. CEUR Workshop Proceedings, vol. 2424, pp. 165–166. CEUR-WS.org (2019), http://CEUR-WS.org/Vol-2424
28. Simon, M., Moldt, D., Schmitz, D., Haustermann, M.: Tools for Curry-Coloured Petri nets. In: Donatelli, S., Haar, S. (eds.) Petri Nets 2019. LNCS, vol. 11522, pp. 101–110. Springer (2019). https://doi.org/10.1007/978-3-030-21571-2_7
29. Valk, R.: Petri nets as token objects - an introduction to elementary object nets. In: Desel, J., Silva, M. (eds.) Petri Nets, 1998. pp. 1–25. No. 1420 in LNCS, Springer-Verlag, Berlin Heidelberg New York (1998). https://doi.org/10.1007/978-3-540-27793-4_29
30. Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design **1**(4), 297–322 (12 1992). https://doi.org/10.1007/BF00709154

31. Willrodt, S., Moldt, D.: Discussion of a Renew implementation of a modular model checking framework for reference nets. In: Bergenthum, R., Kindler, E. (eds.) AWPN 2019, Proceedings. pp. 12–17. Fernuniversität in Hagen, Germany (2019). https://doi.org/https://doi.org/10.18445/20191003-092114-0
32. Wolf, K.: Petri net model checking with lola 2. In: Khomenko, V., Roux, O.H. (eds.) Petri Nets 2018. pp. 351–362. Springer International Publishing, Cham (2018)