SMPT: A Testbed for Reachability Methods in Generalized Petri Nets

Nicolas Amat(✉) and Silvano Dal Zilio

LAAS-CNRS,
Université de Toulouse, CNRS, INSA, Toulouse, France
nicolas.amat@laas.fr

Abstract. SMPT (for Satisfiability Modulo Petri Net) is a model checker for reachability problems in Petri nets. It started as a portfolio of methods to experiment with symbolic model checking, and was designed to be easily extended. Some distinctive features are its ability to benefit from structural reductions and to generate verdict certificates. Our tool is quite mature and performed well compared to other state-of-the-art tools in the Model Checking Contest.

Keywords: Model checking · Reachability problem · Petri nets

1 Introduction

SMPT is an open source model checker designed to answer reachability queries on generalized Petri nets, meaning that we do not impose any restrictions on the marking of places or the weight on the arcs. We can in particular handle unbounded nets. We also support a generalized notion of reachability properties, in the sense that we can check if it is possible to reach a marking that satisfies a combination of linear constraints between places. This is more expressive than the reachability of a single marking and corresponds to the class of formulas used in the reachability category of the Model Checking Contest (MCC), a yearly competition of formal verification tools for concurrent systems [7,27].

The tool name is an acronym that stands for *Satisfiability Modulo Petri Net*. This choice underlines the fact that, for most of the new features we implemented, SMPT acts as a front-end to a SMT solver; but also that it adds specific knowledge from Petri net theory, such as invariants, use of structural properties, etc.

The design of SMPT reflects the two main phases during its development process. The tool was initially developed as a testbed for symbolic model checking algorithms that can take advantage of structural reductions (see e.g. [2,3]). This explains why it includes many "reference" implementations of fundamental reachability algorithms, tailored for Petri nets, such as Bounded Model Checking (BMC) [8,17,22] or k-induction [31]. It also includes new verification methods, such as adaptations of Property Directed Reachability (PDR) [15,16] for Petri nets [6]. One of our goal is to efficiently compare different algorithms, on a level playing field, with the the ability to switch on or off optimizations. This motivates our choice to build a tool that is highly customizable and easily extensible.

In a second phase, since 2021, we worked to make SMPT more mature, with the goal to improve its interoperability, and with the addition of new verification methods that handle problems where symbolic methods are not the best suited. We discuss the portfolio approach implemented in SMPT in Sect. 3. This second set of objectives is carried by our participation in the last two editions of the MCC [25,26], where we obtained a 100% confidence level (meaning SMPT never returned an erroneous verdict). With this last evolution, we believe that SMPT left its status of prototype to become a tool that can be useful to other researchers. This is what motivates the present paper.

There are other tools that perform similar tasks. We provide a brief comparison of SMPT with two of them in Sect. 5, ITS-Tools [32,33] and Tapaal [19]. All tools have in common their participation in the MCC and the use of symbolic techniques. They also share common input formats for nets and formulas. We can offer two reasons for users to use SMPT instead of—or more logically in addition to—these tools. First, SMPT takes advantage of a new approach, called *polyhedral reduction* [2,3], to accelerate the verification of reachability properties. This approach can be extremely effective in some cases where other methods do not scale. We describe this notion in Sect. 2. Another interesting feature of SMPT is the ability to return a *verdict certificate*. When a property is invariant, we can return a "proof" that can be checked independently by a SMT solver.

2 Technical Background

We briefly review some theoretical notions related to our work. We assume basic knowledge of Petri net theory [30]. In the following, we use P for the set of places of a net N. A marking, m, is a mapping associating a non-negative integer, $m(p)$, to every place p in P. SMPT supports the verification of *safety properties* over the reachable markings of a marked Petri net (N, m_0). Properties, F, are defined as a Boolean combination of literals of the form $\alpha \sim \beta$, where $\sim$ is a comparison operator (one of $=$, $\leqslant$ or $\geqslant$) and α, β are linear expressions involving constants or places in P. For instance, $(p + q \geqslant r) \vee (p \leqslant 5)$ is an example property.

We say that property F is valid at marking m, denoted $m \models F$, if the ground formula obtained by substituting places, p, by $m(p)$ is true. As can be expected, we say that F is *reachable* in (N, m_0) if there is m reachable such that $m \models F$. See [2,3,6] for more details. We support two categories of queries: $\mathrm{EF}\, F$, which is true only if F is reachable; and $\mathrm{AG}\, F$, which is true when F is an invariant, with the classic relationship that $\mathrm{AG}\, F \equiv \neg\, (\mathrm{EF}\, \neg F)$. A *witness* for property $\mathrm{EF}\, F$ is a reachable marking such that $m \models F$; it is a *counterexample* for $\mathrm{AG}\, \neg F$. Examples of properties we can express in this way include: checking if some transition t is enabled (quasi-liveness); checking if there is a deadlock; checking whether some linear invariant between places is always true; etc.

SMPT implements several methods that combine SMT-based techniques with a new notion, called polyhedral reduction. The idea consists in computing structural reductions [10,11] of the form $(N_1, m_1) \rhd_E (N_2, m_2)$, where (N_1, m_1) is the (initial) Petri net we want to analyse; (N_2, m_2) is a reduced version; and E is a system of linear equations relating places in N_1 and N_2. The goal is to preserve enough information in E so that we can reconstruct the reachable markings of (N_1, m_1) by knowing only those of (N_2, m_2). Given a starting net, we can automatically compute a polyhedral reduction using the tool Reduce, which is part of Tina [12]. (But obviously there are many irreducible nets.)

Polyhedral reductions are useful in practice. Given a property F_1 on the initial net N_1, we can build a property F_2 on N_2 [2,3] such that checking F_1 on N_1 (whether it is reachable or an invariant) is equivalent to checking F_2 on N_2. We have observed very good speed-ups with this approach, even when we only have a moderate amount of reductions. This notion is also “compatible” with symbolic methods. In SMPT, we recast all constraints and relations into formulas of Quantifier Free Linear Integer Arithmetic (the QF-LIA theory in the SMT-LIB standard [9]) and pass them to SMT solvers.

Another important notion is that of *inductive invariant*. We say that R is an inductive invariant of property F if it is: (i) valid initially ($m_0 \models R$); (ii) inductive (if $m \rightarrow m'$ and $m \models R$ then $m' \models R$, for all markings m, even those that are not reachable); and (iii) $R \supseteq F$. Given a pair (F, R) we can check these three properties automatically using a SMT solver (and with only one formula in each case). In some conditions, when property F is an invariant, SMPT can automatically compute an inductive invariant from F. This provides an independent certificate that invariant F holds.

3 Design and Implementation

SMPT is open-source, under the GNU GPL v3.0 licence, and is freely available on GitHub (https://github.com/nicolasAmat/SMPT). The repository also provides examples of nets, formulas, and scripts to experiment with the tool. SMPT is a Python project of about 4 000 lines of code, and is fully typed using the static type checker mypy. The code is heavily documented (4 500 lines) and we provide many tracing and debugging options that can help understand its inner workings. The project is packaged in libraries, and provides abstract classes to help with future extensions. We describe each library and explain how they can be extended.

The ptio library defines the main data-structures of the model checker, for Petri nets (`pt.py`), reachability formulas (`formula.py`), and reduction equations (`system.py`). It also provides the corresponding parsers, for different formats.

The interface library includes interfaces to external tools and solvers. For example, we provide an integrated interface to z3 [14] built around the SMT-LIB format [9]. We can also interface with MiniZinc [29], a solver based on constraint programming techniques, and with a random state space explorer, WALK, distributed with the TINA toolbox. New tools can be added by implementing the abstract class `Solver` (`solver.py`).

The exec library provides a concurrent "jobs scheduler" that helps run multiple verification tasks in parallel and manage their interactions.

The checker library is the core of our tool. It includes a portfolio of methods intended to be executed in parallel. All methods implement an abstract class (`abstractchecker.py`) which describes the abstract method `prove`. We currently support the following eight methods:

(1) **Induction**: a basic method that checks if a property is an inductive invariant (see Sect. 2). This property is "easy" to check, even though interesting properties are seldom inductive. It is also useful to check verdict certificates.
(2) **BMC**: Bounded Model Checking [13] is an iterative method to explore the state space of systems by unrolling their transitions. This method is only useful for finding counterexamples.
(3) **k-induction**: [31] is an extension of BMC that can also prove invariants.
(4) **PDR**: Property Directed Reachability [15,16], also known as IC3, is a method to strengthen a property that is not inductive, into an inductive one. This method can return a verdict certificate. We provide three different methods of increasing complexity [6] (one for coverability and two for general reachability).
(5) **State Equation**: is a method for checking that a property is true for all "potentially reachable markings" (solution of the state equation). This is a semi-decision method, found in many portfolio tools, that can easily check for invariants. We implement a refined version [32,33] that can over-approximate the result with the help of trap constraints [20] and other structural information, such as NUPN specifications [21].
(6) **Random Walk**: relies on simulation tools to quickly find counterexamples. It is also found in many tools that participate in the MCC [27]. We currently use WALK, distributed with the TINA toolbox, but we are developing a new tool to take advantage of polyhedral reductions.
(7) **Constraint Programming**: is a method specific to SMPT in the case where nets are "fully reducible" (the reduced net has only one marking). In this case, reachable markings are exactly the solution of the reduction equations (E) and verdicts are computed by solving linear system of equations.
(8) **Enumeration**: performs an exhaustive exploration of the state space and relies on the TINA model checker. It can be used as a fail-safe, or to check the reliability of our results.

4 Commands, Basic Usage and Installation

SMPT requires Python version 3.7 or higher. The easiest method for experimenting with the tool is to directly run the `smpt` module as a script, using a command such as `python3 -m smpt`. Our repository includes a script to simplify the installation of the tool and all its dependencies. It is also possible to find disk images with a running installation in the MCC website and in artifacts archived on Zenodo [4,5]. As usual, option `--help` returns an abridged description of all the available options. We list some of them below, grouped by usage.

Input Formats. We accept Petri nets described using the Petri Net Markup Language (PNML) [23] and can also support colored Petri nets (using option `--colored`) by using and external unfolder [18]. For methods that rely on polyhedral reductions, it is possible to automatically compute the reduction (`--auto-reduce`) or to provide a pre-computed version (with option `--reduced-net <path>`). It is also possible to save a copy of the reduced net with the option `--save-reduced-net <path>`.

Verification Methods. We support the verification of three predefined classes of safety properties: *deadlock detection* (`--deadlock`), which is self-descriptive; *quasi-liveness* (`--quasi-liveness t`), to check if it is possible to fire transition `t`; and *reachability* (`--reachability p`), to check if there is a reachable marking where place `p` is marked (it has at least one token). It is also possible to check the reachability of several places, at once, by passing a comma-separated list of names, `--reachability p1,...,pn`; and similarly for liveness. Finally, SMPT supports properties expressed using the MCC property language [28], an XML format encoding the class of formulas described in Sect. 2. Several properties can be checked at once.

Output Format. Results are printed in the text format required by the MCC, which is of the form `FORMULA <id> (TRUE/FALSE)`. There are also options to output more information: `--debug` to print the SMT-LIB input/output code exchanged with the SMT solver; `--show-techniques`, to return the methods that successfully computed a verdict; `--show-time`, to print the execution time per property; `--show-reduction-ratio`, to get the reduction ratio; `--show-model`, to print the counterexample if it exists; `--check-proof`, to check verdict certificates (when we have one); `--export-proof`, to export verdict certificates (inductive invariants, traces leading to counterexamples, etc.).

Tweaking Options. We provide a set of options to control the behaviour of our verification jobs scheduler. We can add a timeout, globally (`--global-timeout <int>`) or per property (`--timeout <int>`). We can also restrict the choice of verification methods (`--methods <method_1> .... <method_n>`). Finally, option `--mcc` puts the tool in "competition mode".

5 Comparison with Other Tools

We report on some results obtained by SMPT, ITS-Tools [32,33], and Tapaal [19] during the 2022 edition of the MCC [26]. We created a public repository [1] containing the scripts used to generate the statistics and oracles for the 2022 edition of the Model Checking Contest for the Reachability category.

SMPT provides a default competition mode that implements a basic strategy that should be effective in the conditions of the MCC. Basically, we start by running the Random Walk and State Equation methods in parallel with a timeout of 120 s, on all formulas, in order to catch easy counterexamples and invariants as quickly as possible. Then we run more demanding methods: BMC, k-induction, PDR, etc. The rationale is that queries used in the reachability competition are randomly generated and usually exhibit a bias towards "counterexamples" (CEX), meaning false AG properties or true EF ones. Also, when the formula is an invariant (INV), for instance a "true AG property", it can often be decided with the State Equation method.

Our tool is quite mature. It achieved a perfect reliability score (all answers are correct) and ranked at the third position, behind TAPAAL and ITS-TOOLS. We display the results in a Venn diagram where we make a distinction between CEX and INV properties. There is a total of 50 187 answered queries (with almost 60% CEX). We observe that a vast majority of these queries (41 006) are computed by all tools, and can be considered "easy". Conversely, we have 9 181 difficult queries, solved by only one or two tools (Fig. 1).

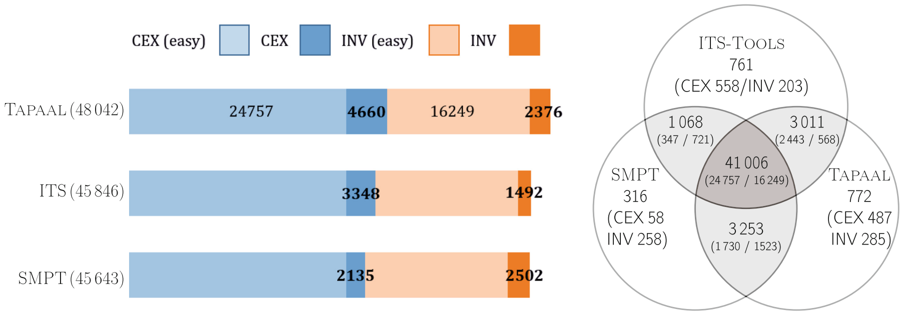

Fig. 1. Comparison of tools on all computed queries

We also provide a bar chart where we distinguish between easy/difficult, and CEX/INV queries. We observe that, while SMPT ranks last in the number of unique queries, it behaves quite well with invariants (INV); which is the category we target with our most sophisticated methods. Overall, we observe that SMPT performs well compared to other state-of-the-art tools in the Model Checking Contest and that it is a sensible choice when we try to check invariants.

6 Future Work

Work on SMPT is still ongoing. At the moment, we concentrate on methods to quickly discover counterexamples. The idea is to combine polyhedral reductions and random exploration in order to find counterexamples directly in the reduced net. We also plan to improve our use of the "state equation" method, in particular by identifying new classes of Petri nets for which all potentially reachable markings are indeed reachable. A problem we already started to study in a different setting [24].

References

1. Amat, N.: Oracles and report for the reachability category of the model checking contest (2022). https://github.com/nicolasAmat/MCC-Reachability
2. Amat, N., Berthomieu, B., Dal Zilio, S.: On the combination of polyhedral abstraction and SMT-based model checking for petri nets. In: Buchs, D., Carmona, J. (eds.) PETRI NETS 2021. LNCS, vol. 12734, pp. 164–185. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-76983-3_9
3. Amat, N., Berthomieu, B., Dal Zilio, S.: A polyhedral abstraction for petri nets and its application to SMT-based model checking. Fund. Inform. **187**(2–4), 103–138. IOS Press (2022). https://doi.org/10.3233/FI-222134
4. Amat, N., Dal Zilio, S.: Artifact for FM 2023 paper: SMPT: a testbed for reachability methods in generalized petri nets, November 2023. https://doi.org/10.5281/zenodo.7341426
5. Amat, N., Dal Zilio, S., Hujsa, T.: Artifact for TACAS 2022 paper: property directed reachability for generalized petri nets, January 2022. https://doi.org/10.5281/zenodo.5863379
6. Amat, N., Zilio, S.D., Hujsa, T.: Property directed reachability for generalized petri nets. In: TACAS 2022. LNCS, vol. 13243, pp. 505–523. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_28
7. Amparore, E., et al.: Presentation of the 9th edition of the model checking contest. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) TACAS 2019. LNCS, vol. 11429, pp. 50–68. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17502-3_4
8. Armando, A., Mantovani, J., Platania, L.: Bounded model checking of software using SMT solvers instead of SAT solvers. In: Valmari, A. (ed.) SPIN 2006. LNCS, vol. 3925, pp. 146–162. Springer, Heidelberg (2006). https://doi.org/10.1007/11691617_9
9. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB standard: version 2.6. Technical report, Department of Computer Science, The University of Iowa (2017). http://www.smt-lib.org/
10. Berthelot, G.: Transformations and decompositions of nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) ACPN 1986. LNCS, vol. 254, pp. 359–376. Springer, Heidelberg (1987). https://doi.org/10.1007/978-3-540-47919-2_13
11. Berthomieu, B., Le Botlan, D., Dal Zilio, S.: Counting Petri net markings from reduction equations. Int. J. Softw. Tools Technol. Transfer **22**(2), 163–181 (2019). https://doi.org/10.1007/s10009-019-00519-1
12. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool TINA-construction of abstract state spaces for Petri nets and time Petri nets. Int. J. Prod. Res. **42**(14), 2741–2756 (2004)
13. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49059-0_14
14. Bjørner, N.: The Z3 Theorem Prover (2020). https://github.com/Z3Prover/z3/

15. Bradley, A.R.: SAT-based model checking without unrolling. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 70–87. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18275-4_7
16. Bradley, A.R.: Understanding IC3. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 1–14. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_1
17. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods Syst. Des. **19**, 7–34 (2001). https://doi.org/10.1023/A:1011276507260
18. Dal Zilio, S.: MCC: A tool for unfolding colored petri nets in PNML format. In: Janicki, R., Sidorova, N., Chatain, T. (eds.) PETRI NETS 2020. LNCS, vol. 12152, pp. 426–435. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51831-8_23
19. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K.Y., Møller, M.H., Srba, J.: TAPAAL 2.0: integrated development environment for timed-arc petri nets. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 492–497. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_36
20. Esparza, J., Melzer, S.: Verification of safety properties using integer programming: beyond the state equation. Formal Methods Syst. Des. **16**(2), 159–189 (2000). https://doi.org/10.1023/A:1008743212620
21. Garavel, H.: Nested-unit Petri nets. J. Log. Algebr. Methods Program. **104**, 60–85 (2019). https://doi.org/10.1016/j.jlamp.2018.11.005
22. Heljanko, K.: Bounded reachability checking with process semantics. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 218–232. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44685-0_15
23. Hillah, L.M., Kordon, F., Petrucci, L., Trèves, N.: PNML framework: an extendable reference implementation of the petri net markup language. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 318–327. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13675-7_20
24. Hujsa, T., Berthomieu, B., Dal Zilio, S., Le Botlan, D.: Checking marking reachability with the state equation in Petri net subclasses. arXiv preprint: arXiv:2006.05600 (2020)
25. Kordon, F., et al.: Complete results for the 2021 edition of the model checking contest, June 2021. http://mcc.lip6.fr/2021/results.php
26. Kordon, F., et al.: Complete results for the 2022 edition of the model checking contest (2022). http://mcc.lip6.fr/2022/results.php
27. Kordon, F., Hillah, L.M., Hulin-Hubard, F., Jezequel, L., Paviot-Adet, E.: Study of the efficiency of model checking techniques using results of the MCC from 2015 To 2019. Int. J. Softw. Tools Technol. Transfer **23**(6), 931–952 (2021). https://doi.org/10.1007/s10009-021-00615-1
28. LIP6: model checking contest property language (manual). Petri Nets (2020)
29. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74970-7_38
30. Reisig, W.: Petri Nets: An Introduction, vol. 4. Springer Science & Business Media, Cham (2012). https://doi.org/10.1007/978-3-642-69968-9
31. Sheeran, M., Singh, S., Stalmarck, G.: Checking safety properties using induction and a SAT-solver. In: Hunt, W.A., Johnson, S.D. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 127–144. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-40922-X_8

32. Thierry-Mieg, Y.: Symbolic model-checking using ITS-tools. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 231–237. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_20
33. Thierry-Mieg, Y.: Structural reductions revisited. In: Janicki, R., Sidorova, N., Chatain, T. (eds.) PETRI NETS 2020. LNCS, vol. 12152, pp. 303–323. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51831-8_15