The PEP Verification System[1]

Eike Best[2], Javier Esparza[3], Bernd Grahlmann[4], Stephan Melzer[3], Stefan Römer[3] and Frank Wallner[3]

Abstract

This paper gives a short overview of the verification system PEP (a Programming Environment based on Petri nets). It focuses on some recent developments.
Keywords: Model Checking, Partial Order Semantics, Petri Nets, Programming, Verification.

1 Introduction

The PEP tool [1, 4, 5, 17] supports the development of parallel systems. It combines a set of facilities for the modelling, the editing, the simulation and the compilation of systems expressed in various languages – including an imperative programming language, Petri nets, and finite automata – with a number of algorithms for the analysis and the verification of their properties. Two of the distinctive features of the tool are that it is based on Petri net theory, and that its analysis component uses partial order semantics in order to avoid the construction of the state space, whenever possible.

This paper is structured as follows. Section 2 provides a brief account of the structure of the tool. Section 3 describes the verification component, in particular the novel items. Section 4 concludes.

2 Modelling parallel systems with PEP

The PEP tool [1, 4, 5, 17] supports the main phases of the development of parallel systems: modelling, simulation, compilation, analysis and verification. The user may choose between a spectrum of input languages and various ways of translating them into each other. There is also the possibility of editing temporal logic formulae and running automatic verification algorithms. These algorithms are comparable in performance with other existing model-checking systems [11, 14, 19, 22, 23], but are limited – as are all others – by the inherent complexity of verification. They could be used, for instance, for the verification of prototypes, intricate distributed algorithms, or communication protocols.

[1]This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) under grants Be 1267/2-1, Be 1267/2-2, Be 1267/6-1, Fl 207/1-1, Sta 450/1-1 and by project A3 (Spezifikation, Analyse, Modellierung) of the DFG-Sonderforschungsbereich SFB-342 (Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen).

[2]e.best@informatik.uni-oldenburg.de; Fachbereich Informatik, Carl-von-Ossietzky-Universität zu Oldenburg.

[3]{esparza,melzers,roemer,wallnerf}@informatik.tu-muenchen.de; Institut für Informatik, Technische Universität München.

[4]bernd@informatik.uni-hildesheim.de; Institut für Informatik, Universität Hildesheim.

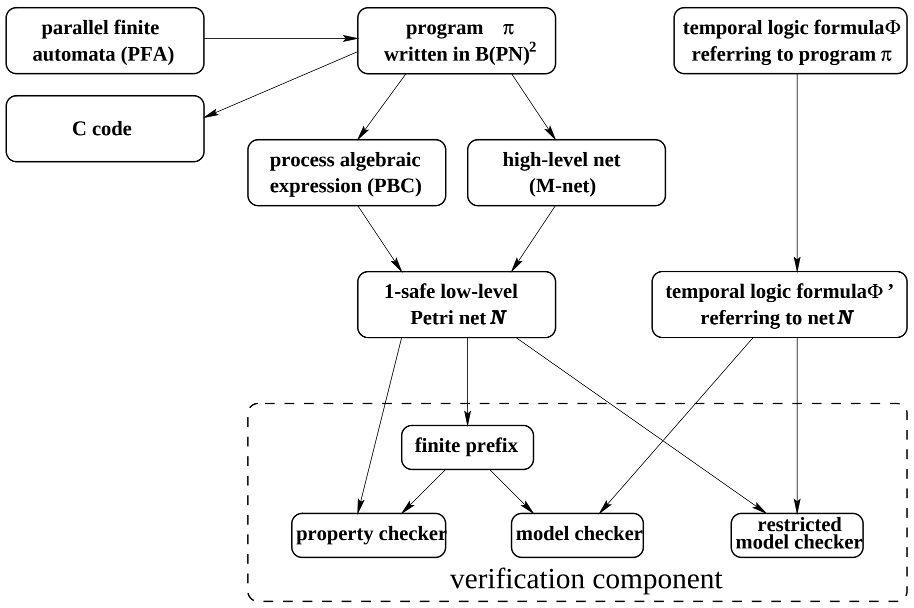
parallel finite automata (PFA)
program π written in B(PN)²
temporal logic formula Φ referring to program π
C code
process algebraic expression (PBC)
high-level net (M-net)
1-safe low-level Petri net *N*
temporal logic formula Φ' referring to net *N*
finite prefix
property checker
model checker
restricted model checker
verification component

Figure 1: Functionality diagram of the existing PEP system

Figure 1 depicts the components of the existing PEP system and their functional dependencies. The user may choose between modelling a parallel system either as a collection of finite automata [13], writing a concurrent program in a language such as $B(PN)^2$ [7], or specifying a modular high-level Petri net (M-net [2]). The models may be edited and simulated, compiled into Petri nets.

The translations between different models is achieved by means of compilations, as shown by arrows in figure 1. For instance, a parallel finite automaton may be compiled into a $B(PN)^2$ program. The latter may be compiled either into a process algebraic expression of the PBC [6] or into an M-net, and from either, further into a 1-safe low-level net (both routes yield equivalent low-level nets). A 1-safe low-level net may further be subjected to a prefix-builder which constructs the McMillan prefix [9, 15].

Editors and simulators exist for the following objects shown in figure 1: parallel finite automata, $B(PN)^2$ programs, M-nets and 1-safe nets. For PBC expressions and temporal logic formulae, PEP provides only editors but (naturally, for formulae) no specific simulators.

The components of PEP are glued together by a reference component [12] which records information such as, for instance, which place(s) of a net correspond to a given variable or to a given control point of a program. By means of the reference component, formulae referring to a program may be translated automatically into formulae referring to the corresponding net, and the simulation of a net may trigger the simulation of the corresponding program (if there exists any). The reference component also severs a link between a program and a net, once the latter has been edited independently of the former.

Finally, PEP also has output filters such as one which transforms a $B(PN)^2$ program into executable C code. PEP has been implemented on X-platforms and conforms to the usual standards of user-friendly software.

3 The Verification Component of PEP

The starting point of the verification component is a 1-safe Petri net. Most of the implemented algorithms try to avoid the state explosion problem using one of the following two strategies: (1) construct a compact representation of the state space by means of a so-called net unfolding [15, 9], or (2) construct a set of constraints whose solutions are a superset of the state space, and use this set to derive semidecision procedures.

The verification component offers a number of dedicated algorithms to check key properties like deadlock-freedom. A wider range of safety properties can be checked using efficient semidecision algorithms [16]. Finally, PEP also offers a model checker for a small temporal logic [8]. When the Petri net satisfies certain constraints this model checker can be replaced by a more efficient one [3].

Recent developments. We briefly describe three recent developments:

- A new model checker for Linear Temporal Logic (LTL).
- An extension that allows to check Petri nets which are not necessarily 1-safe.
- A new technique that improves the efficiency of the deadlock detection algorithm.

We have developed a model checker for 1-safe Petri nets and LTL formulas [24]. It is based on the so-called automata-theoretic approach to model checking. Given a formula ϕ, a Büchi automaton $A_{\neg\phi}$ is constructed which accepts exactly all the infinite sequences of markings which *violate* ϕ. Then, $A_{\neg\phi}$ is composed in a certain way with the system under consideration to yield a 1-safe net that accepts all the computations of the system that fail ϕ. Finally, the unfolding of the composed net is computed and used to decide if the composed system accepts any sequence.
The advantage of the algorithm lies in the fact that the unfolding of the composed system is usually much smaller than the state space. This allows to solve the state-explosion problem in many cases. When the system does not satisfy the property, a failing run is delivered as fault diagnosis.

The method for the construction of the net unfolding introduced in [9] accepted only 1-safe nets as input. We have recently extended it to arbitrary Petri nets. The algorithm terminates if and when the Petri net has a finite state space [10]. It is planned to extend the algorithm even further to high-level nets (such as M-nets).

We have implemented an algorithm for deadlock detection based on net unfoldings [18]. In contrast to a former algorithm by McMillan for the same purpose [15], the unfolding is explored using a new technique based on linear programming. This technique improves the performance in many cases.

4 Conclusions

We have briefly presented some of the main features of the PEP tool. For an overview of the other analysis and verification methods available in PEP we refer the reader to [1].

Our experiences have prompted us to consider the following future aims:

- A model checker for high-level Petri nets will be developed and implemented.
- The INA system [21] (in particular, its state graph checker and its theorem data base) will be fully integrated in PEP.
- Other verification methods (like a BDD based model checker) will be integrated, and interfaces to other tools (like SPIN [14] or PROD [22]) will be offered.
- The graphical net editors will be improved to meet the needs of industrial applications, e.g. [20].

References

[1] E. Best: Partial Order Verification with PEP. Proc. *POMIV '96, Partial Order Methods in Verification.* G. Holzmann, D. Peled, V. Pratt (eds), American Mathematical Society (1996).

[2] E. Best, H. Fleischhack, W. Frączak, R.P. Hopkins, H. Klaudel and E. Pelz: An M-net Semantics of $B(PN)^2$. Proc. *STRICT '95*, Berlin, J. Desel (ed.). Springer-Verlag, Workshops in Computing, 85–100 (1995).

[3] E. Best and J. Esparza: Model Checking of Persistent Petri Nets. Proc. *5th Workshop Computer Science Logic-91.* Springer-Verlag, LNCS Vol. 626, 35-52 (1992).

[4] E. Best and B. Grahlmann: PEP - more than a Petri Net Tool. Proc. *Tools and Algorithms for the Construction and Analysis of Systems*, 2nd International Workshop, TACAS '96, Passau, March 1996, T. Margaria, B. Steffen (eds). Springer-Verlag, LNCS Vol. 1055, 397–401 (1996).

[5] E. Best and B. Grahlmann: PEP: Documentation and User Guide. Universität Hildesheim, 1995. Available together with the tool via:
`http://www.informatik.uni-hildesheim.de/~pep/HomePage.html`

[6] E. Best, R. Devillers and J.G. Hall: The Petri Box Calculus: a New Causal Algebra with Multilabel Communication. *Advances in Petri Nets 1992*, G. Rozenberg (ed.). Springer-Verlag, LNCS Vol. 609, 21–69 (1992).

[7] E. Best and R.P. Hopkins: $B(PN)^2$ – a Basic Petri Net Programming Notation. Proc. *PARLE'93*, A. Bode, M. Reeve, G. Wolf (eds). Springer-Verlag, LNCS Vol. 694, 379–390 (1993).

[8] J. Esparza: Model Checking based on Branching Processes. Habilitation, Hildesheim (1993). Published as: Model Checking Using Net Unfoldings. Proc. *TAPSOFT'93* (1993), M.C. Gaudel, J.P. Jouannaud (eds). Springer-Verlag, LNCS Vol. 668, 613–628 (1993). Full version in *Science of Computer Programming* Vol. 23, 151–195 (1994).

[9] J. Esparza, St. Römer and W. Vogler: An Improvement of McMillan's Unfolding Algorithm. *Proc. of TACAS'96*, 1996.

[10] J. Esparza, St. Römer and W. Vogler: An Extension of McMillan's Unfolding Algorithm (in preparation).

[11] P. Godefroid: On the Costs and Benefits of using Partial-order Methods for the Verification of Concurrent Systems. Proc. *POMIV'96, Partial Order Methods in Verification.* G. Holzmann, D. Peled, V. Pratt (eds), American Mathematical Society (1996).

[12] B. Grahlmann: PEP: A Reference Component Supports Analysis of Parallel Systems (submitted).

[13] B. Grahlmann, M. Moeller and U. Anhalt: A New Interface for the PEP Tool: Parallel Finite Automata. Proc. 2nd Workshop *Algorithmen und Werkzeuge für Petrinetze 1995*, J. Desel, H. Fleischhack, A. Oberweis, and M. Sonnenschein (eds), Report AIS 22, Universität Oldenburg (1995).

[14] G. Holzmann: *Design and Validation of Computer Protocols.* Prentice Hall (1990).

[15] K.L. McMillan: Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits. Proc. *4th Workshop on Computer Aided Verification*, 164–174 (1992).

[16] St. Melzer and J. Esparza: Checking System Properties via Integer Programming. Proc. *ESOP'96, European Symposium on Programming*, Hanne Riis Nielson (ed.). Springer-Verlag, LNCS 1058, 250–264 (1996).

[17] St. Melzer, St. Römer and J. Esparza: Verification using PEP. Proc. *AMAST '96.* Springer-Verlag, LNCS 1101, pp. 591–594. (1996).

[18] St. Melzer and St. Römer: Checking Deadlock Properties Using Net Unfoldings (in preparation).

[19] D. Peled: Combining Partial Order Reductions with On-the-fly Model-Checking. *Journal of Formal Methods in Systems Design*, Vol. 8(1), 39–64 (1996).

[20] B. Sánchez: Model Checking for State-Graphs via Integer Programming. Internal Report ZF SE 95/96 Sanchez-1, Siemens ZFE, 1996.

[21] P. H. Starke: *INA: Integrated Net Analyzer.* Reference Manual, 1992.

[22] K. Varpaaniemi, J. Halme, K. Hiekkanen and T. Pyssysalo: PROD reference Manual. Helsinki University of Technology, Digital Systems Laboratory, Series B: Technical Report No.13 (1995).

[23] A. Valmari: Stubborn Set Methods for Process Algebras. Proc. *POMIV'96, Partial Order Methods in Verification.* G. Holzmann, D. Peled, V. Pratt (eds), American Mathematical Society (1996).

[24] F. Wallner: Model Checking LTL Using Net Unfoldings (in preparation).