The PEP Tool

Bernd Grahlmann

Institut für Informatik, Universität Hildesheim,
bernd@informatik.uni-hildesheim.de

Abstract The **PEP** tool embeds sophisticated programming and verification components in a user-friendly graphical interface. The basic idea is that the programming component allows the user to design concurrent algorithms in an imperative language, and that the **PEP** system then generates Petri nets from such programs in order to use Petri net theory for simulation and verification purposes. A key feature is flexibility; its modular design eases the task of adding new interfaces to other verification packages, such as 'INA', 'PROD' or 'SMV'.

PEP has been implemented on Solaris 2.x, Sun OS 4.1.x and Linux. Ftp-able versions are available via `http://www.informatik.uni-hildesheim.de/~pep`.

Keywords: Binary decision diagrams, $B(PN)^2$, Model checking, **PEP**, Petri nets, Simulation, Stubborn sets, Temporal logic, Tool.

1 System overview

The **PEP**[1] tool is a **P**rogramming **E**nvironment based on **P**etri Nets [5]. The main phases of the development of parallel systems (modelling, simulation, analysis and verification) are supported. From an abstract point of view the tool consists of the following three components (see Fig. 1):

1. The editing and simulation component contains editors and simulators for parallel systems from which Petri nets can be generated automatically using the different compilers integrated in this component.
2. The formula component offers the possibility to enter formulae which can either refer to a parallel program or a net. Furthermore, program formulae can be transformed into net formulae [14].
3. The verification component consisted of a model checker [11,15] based on the finite prefix of the branching process [20,10]. Interfaces to other verification packages, like 'INA' [21], 'PROD' [24] or 'SMV' [8], have been added recently in order to offer as many different verification methods as possible and to support stronger logics.

2 Editing and simulation component

The editing and simulation component of the **PEP** system has grown continuously. Corresponding to the main intention of the tool – i.e. to use Petri net theory in order to provide program verification – a program editor and a net editor have been the starting point.

With the first version of **PEP** it was possible to edit parallel algorithms expressed in $B(PN)^2$, an imperative / predicative programming language. $B(PN)^2$

[1] PEP is a joint project between the 'Universität Hildesheim' and the 'Humboldt-Universität zu Berlin' which is financed by the DFG (German Research Foundation). Furthermore, the work has been partially supported by the projects CALIBAN (ESPRIT) and EXPRESS (HCM)

is called **B**asic **P**etri **N**et **P**rogramming **N**otation because it has a compositional semantics in terms of Petri nets [2]. In addition to sequential composition, nondeterministic choice and iteration, parallel composition is available. Parallel processes can share common memory (asynchronous communication) or use (synchronous) channel communication or both.

Originally, the compilation of these programs into Petri boxes, a special class of 1-safe low-level (LL) Petri nets [2], via terms of a process algebra called PBC (**P**etri **B**ox **C**alculus) [2], which is an extension / modification of CCS, was supported. Later on, an alternative net semantics of $B(PN)^2$ programs was given by means of M-nets, a special class of high-level (HL) Petri nets [3]. The implementation of a procedure concept [12,18,19] extends $B(PN)^2$ to a complete programming language.

Furthermore, a new input interface was added. Parallel finite automata (PFA) with $B(PN)^2$ actions as arc annotations can be edited and compiled into $B(PN)^2$ programs [13].

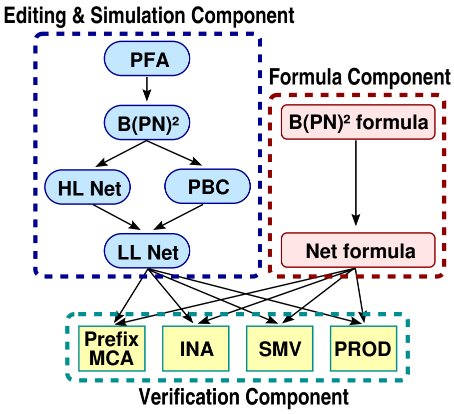

Figure1. The different components of the **PEP** system.

Thus the editing and simulation component of **PEP** presently contains:

1. Editors for PFA, $B(PN)^2$ programs, PBC terms, HL nets and LL nets.
2. Simulators for $B(PN)^2$ programs, HL nets and LL nets.
3. Compilers as follows: PFA $\Rightarrow$ $B(PN)^2$, $B(PN)^2$ $\Rightarrow$ PBC, PBC $\Rightarrow$ LL net, $B(PN)^2$ $\Rightarrow$ HL net and HL net $\Rightarrow$ LL net.

3 Verification component

One of the most important aims during the development of the **PEP** tool was to provide verification in addition to simulation and analysis of standard properties. Therefore, the implementation of a model checking algorithm, in order to allow temporal logic formulae to be checked, was envisaged and carried out. Due to the well-known state explosion problem, efficiency was strongly required.

First a very efficient model checking algorithm for a restricted class of low-level Petri nets was implemented [22], which avoids any state space generation taking the net directly as input. Unfortunately, the class of nets the model checker can handle is too restricted for the **PEP** approach.

Therefore, another model checking algorithm [11,15] was implemented. State graph generation is avoided. The finite prefix of a branching process (an optimised version of the McMillan unfolding [20,10]) of the low-level Petri net is taken as input for this partial order based method. A drawback, which had to be accepted, was that efficiency was gained on the cost of the expressiveness of the supported logic, S_4, a propositional logic on place names, augmented with $\Box$ for 'always' (`AG` in CTL) and $\Diamond$ for 'possibly at some future point' (`EF` in CTL).

It is widely accepted, that no verification method exists (at least not yet) which is superior to all other methods [9,16]. Thus we decided to provide interfaces to other verification packages, so far to INA, PROD and SMV.

There have been three reasons for selecting the SMV [8] package: it offers CTL (Computational Tree Logic) model checking; it uses BDDs (Binary Decision Diagrams) [7] to encode the transition relation; and it is known to be good and easy to integrate. We tested different ways to encode low-level nets using the SMV formalism [25]. Encoding the transition relation directly seems to be the most promising solution. Place invariants, which may be provided by the net generators, tend to speed up the verification. Furthermore, we spent some efforts to improve the way results are reported to the user. S/he can for example choose to animate the firing sequences in the net and/or program editor (as for other analysis method included in the **PEP** tool [14]).

The PROD package [24] has similar advantages: it offers linear time temporal model checking including CTL; state graph generation can be improved by stubborn set reduction [23] as well as symmetrical reduction methods; the user can additionally profit from the on-the-fly verification method.

INA [21] was integrated in a more complete way. Apart from state graph based methods – stubborn set reduction as well as symmetrical reduction are provided on demand – INA includes the most complete available set of Petri net analysis methods. Among others, many elementary properties, e.g. structural properties and different invariants, can be computed.

Even though not every feature of the three packages is available, **PEP** can also be considered to be a comfortable graphical interface to these originally textual tools, in addition to being a stand-alone tool.

4 Conclusion and future work

We briefly presented some of the main features of the **PEP** tool. For a more detailed overview of the **PEP** system we refer the reader to [1,5] and the various papers which are available at `http://www.informatik.uni-hildesheim.de/~pep`.

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

- A model checker for high-level Petri nets should be developed and implemented in order to avoid the unfolding into low-level Petri nets.
- The integration of the verification packages will be enhanced and interfaces to other verification packages (such as SPIN [17]) will be provided.

Acknowledgement: A lot of people (theoreticians and implementors) contributed to the development of the **PEP** system. Thanks!

References

1. E. Best. Partial Order Verification with PEP. *Proc. of POMIV'96, Princeton.*
2. E. Best, R. Devillers and J. G. Hall. The Box Calculus: a New Causal Algebra with Multi-Label Communication. *Advances in Petri Nets 92*, Springer *LNCS 609.*
3. E. Best, H. Fleischhack, W. Frączak, R. P. Hopkins, H. Klaudel, and E. Pelz. An M-Net Semantics of $B(PN)^2$. *Proc. of STRICT'95*, Springer.
4. E. Best, H. Fleischhack, W. Frączak, R. P. Hopkins, H. Klaudel, and E. Pelz. A Class of Composable High Level Petri Nets. *Proc. of ATPN'95.*
5. E. Best and B. Grahlmann. *PEP: Documentation and User Guide.* Universität Hildesheim. Available together with the tool via: `http://www.informatik.uni-hildesheim.de/~pep/HomePage.html`.
6. E. Best and R. P. Hopkins. $B(PN)^2$ – a Basic Petri Net Programming Notation. *Proc. of PARLE*, Springer *LNCS 694.*
7. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.
8. E. Clarke and K. McMillan and S. Campos and V. Hartonas-Garmhausen. Symbolic Model Checking. *Proc. of CAV'96*, New Brunswick, Springer *LNCS 1102.*
9. J. C. Corbett. Evaluating Deadlock Detection Methods for Concurrent Software. *Technical report*, University of Hawaii at Manoa, 1995.
10. J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. *Proc. of TACAS'96*, Springer *LNCS 1055.*
11. J. Esparza. *Model Checking Using Net Unfoldings*, 151–195. Number 23 in Science of Computer Programming. ELSEVIER, 1994.
12. H. Fleischhack and B. Grahlmann. *A Petri Net Semantics for $B(PN)^2$ with Procedures. Proc. of PDSE'97*, Boston.
13. B. Grahlmann, M. Moeller, and U. Anhalt. A New Interface for the PEP Tool – Parallel Finite Automata. *Proc. of 2. AWPN'95.*
14. B. Grahlmann. The Reference Component of PEP. *Proc. of TACAS'97, Enschede.* Springer *LNCS 1217.*
15. B. Graves. Identification of Specific Processes Contained in McMillan's Finite Unfolding. Submitted.
16. M. Heiner and P. Deussen. Petri Net Based Design and Analysis of Reactive Systems. *Proc. of WODES'96, Edinburgh.*
17. G. Holzmann and D. Peled. The State of SPIN. *Proc. of CAV'96, New Brunswick.*
18. L. Jenner. A Low-Level Net Semantics for $B(PN)^2$ with Procedures. In E. Best and H. Fleischhack, editors. *PEP: Programming Environment Based on Petri Nets.* Hildesheimer Informatik-Berichte 14/95. 1995.
19. J. Lilius and E. Pelz. An M-Net Semantics for $B(PN)^2$ with Procedures. *Proc. of ISCIS-XI'96.*
20. K. McMillan. A Technique of a State Space Search Based on Unfolding. *Formal Methods in System Design 6(1)*, 1996.
21. P. H. Starke. *INA: Integrated Net Analyzer.* Handbuch, 1992.
22. T. Thielke. Implementierung eines effizienten Modelchecking-Algorithmus. *Petri-Netze im Einsatz für Entwurf und Entwicklung von Informationssystemen*, 1993.
23. A. Valmari. Stubborn Sets for Reduced State Space Generation. In *APN'90*, Springer *LNCS 483.*
24. K. Varpaaniemi and J. Halme and K. Hiekkanen and T. Pyssysalo. PROD Reference Manual. *Technical Report B 13, University of Helsinki*, 1995.
25. G. Wimmel. A BDD-based Model Checker for the PEP Tool. *Technical Report, University of Newcastle upon Tyne*, 1997.