The State of PEP*

Bernd Grahlmann

Universität Oldenburg, FB 10, Postfach 2503, D-26111 Oldenburg
grahlmann@informatik.uni-oldenburg.de,
http://theoretica.informatik.uni-oldenburg.de/~bernd

Abstract. The PEP tool can be considered to be one of the most widely distributed Petri net based tools. A continuously increasing functionality and an adequate graphical user interface may have been good reasons for its acceptance. Currently the tool contains approximately 500,000 lines of source code, and supports (to the best of our knowledge) the widest variety of input formalisms and verification methods of all verification tools. We briefly review the most recent developments.
Ftp-able versions of the tool and PEP related papers are available via http://theoretica.informatik.uni-oldenburg.de/~pep.

Keywords: 3D visualisation, C code generation, Parallel programs, PEP, Petri nets, SDL, Simulation, Verification.

1 New Petri net generators

One of the key features of the PEP tool [3] is that simulation, analysis and verification of different input formalisms

- SDL (Specification Description Language [6]) systems,
- parallel programs written in $B(PN)^2$ [4],
- parallel finite automata (PFA) [11],
- process algebra terms expressed in the PBC [1],
- high-level (HL) Petri nets in the M-nets algebra [2], and
- low-level (LL) Petri nets [1]

are based on Petri net theory. PEP's existing transformers between different formalisms (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) were accomplished with two new compilers.

An HL Petri net semantics of SDL [10] was implemented. In particular, dynamic creation as well as termination of processes and (also recursive) procedures are covered. Its integration in PEP supports almost the same simulation, analysis and verification methods as there are available for $B(PN)^2$ programs:

- An SDL simulation may be triggered by a Petri net simulation.
- Standard analyses, such as deadlock detection are possible.
- CTL and LTL properties of SDL specifications may be checked. The user may, e.g., ask whether there exists an instance of a certain process Sender, such that whenever this instance is in state wait while its input queue is non-empty, then it may reach a state send while its variable X has value 3:

* PEP has been a joint project between the 'Universität Hildesheim', the 'Carl-von-Ossietzky Universität Oldenburg' and the 'Humboldt-Universität zu Berlin' which has been financed by the DFG (German Research Foundation).

$$\texttt{EXISTS}\ id\ \texttt{IDSET(}\mathsf{Sender}\texttt{):}$$
$$\mathsf{AG}\Big(\big(\mathsf{Sender}[id].\texttt{state} = \mathsf{wait} \land \mathsf{Sender}[id].\texttt{noqueue} > 0\big) \Rightarrow \big(\mathsf{EF}\big(\mathsf{Sender}[id].\texttt{state} = \mathsf{send} \land \mathsf{Sender}[id].\mathsf{X} = 3\big)\big)\Big).$$

Such a formula is transparently tranformed into a Petri net formula, which is then checked against the Petri net semantics. A resulting counter example may finally be simulated in the SDL editor.

In order to improve the efficiency and to support hybrid modelling of parallel systems we added a compiler from PFA into M-nets [14]. A collection of finite automata together with a number of variable declarations is compiled in a $B(PN)^2$ specific or SDL specific way into an HL Petri net.

2 New HL Petri net editing facilities

In addition to the usual net editing facilities, the HL Petri net editor of the PEP tool now supports a 'programming with nets' approach.

1. The user may compose nets in parallel, in sequence, or in choice; transitions may be substituted by nets; and nets may be synchronised.
2. Moreover, parameterised macro nets (e.g., variable, channel, or procedure nets) may be loaded from libraries and be instantiated (e.g., with the name or type of a variable). Likewise, program semantics may be constructed.

3 New HL Petri net simulation improvements

The performance of the HL Petri net simulator was increased:

- More elaborated analyses of the inscriptions as well as different caching mechanisms give a speed up.
- Petri net simulation may now be distributed over a network of computers (with possibly different operating systems).

4 New 3D visualisation component

Simulation plays a major role within the PEP tool. Based on a Petri net simulation engine, not only Petri nets but also, e.g., parallel programs can be simulated using a dedicated reference scheme [13, 15].

In particular for non Petri net experts, an interactive 3D visualisation of the modelled parallel system (e.g., an elevator) is more appropriate than a Petri net simulation. Thus, we included such an additional Java based feature [19].

We have chosen to base this visualisation on VRML (Virtual Reality Modelling Language). Thus, the user models, e.g., an elevator as a 3D VRML world using an arbitrary VRML editor. After this, the PEP tool offers the possibility to define certain graphical actions (e.g., that the door of the elevator is closed, the elevator moves up one floor, and the door is opened) on top of these worlds, and to bind such actions either directly to transition occurrences of the corresponding Petri net or to action occurrences of the corresponding parallel program. This is done in such an abstract way, that (in general) these bindings are automatically updated upon changes of the corresponding program (and thus resulting net).

The new component may be used in three complementary ways:

1. A random Petri net simulation which triggers a 3D visualisation may exhibit errors (even to non-experts) if, e.g., an elevator moves with an open door.
2. Test scenarios may be created and checked easily interacting with the 3D visualisation (e.g, mouse clicks on elevator buttons may guide the simulation).
3. Transition sequences resulting from a verification may be used to trigger the simulation of a 3D elevator visualising specification errors to non-experts.

5 New code generation

Jäger [17] extended the C code generator for the parallel programming language $B(PN)^2$, which uses the parallelism of the operating system, to the handling of procedures. POSIX conformity and thus operating system and compiler independency was improved. Thus, PEP now supports generation of executable code.

6 New verification facilities

PEP's verification component (a partial order based model checker [8, 9, 16], an interface to the BDD based CTL model checker SMV [7, 23], and an interface to the structural analysis component of INA [22]) was extended and improved:

- We speeded up (by a factor of up to more than 100) the program verification with the SMV verifier using information provided by the different net generators. Compared with the original approach, we are, e.g., able to replace different variables which correspond to control flow points by one single variable using invariant information calculated in almost zero time [15].
- INA was extended by a CTL model checker which performs state graph analysis supporting reductions based on stubborn sets and symmetries.
- Spin [18] was integrated [12] including editing facilities for LTL formulae and a graphical interface to the options of the LTL model checker which is almost identical to the XSPIN interface. Based on an automatic translation of Petri nets into PROMELA, the user may now use the SPIN verifier in a transparent way for the verification of all kinds of input formalisms of PEP.
- We integrated more algorithms based on linear programming:
  - a deadlock checker using net unfoldings [21] and
  - an extended reachability analyser which inputs the LL net [20].
- Moreover, we implemented a compiler from Petri nets into networks of communicating automata in the FC2 format. Based on this translation we are integrating parts of the FC2tool set [5]. As a first step we provided deadlock detection based on BDD/implicit representations of states.

7 Conclusion

We briefly presented some of the new features of the PEP tool. For a more detailed overview we refer to [15] and the various papers which are available together with the tool at http://theoretica.informatik.uni-oldenburg.de/~pep.

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

References

1. E. Best, R. Devillers, and J. G. Hall. The Box Calculus: a New Causal Algebra with Multi-Label Communication. In *APN'92*, *LNCS* 609, 21–69. Springer, 1992.
2. E. Best, H. Fleischhack, W. Frączak, R. P. Hopkins, H. Klaudel, and E. Pelz. A Class of Composable High Level Petri Nets. In *Proc. of ATPN'95*, *LNCS* 935, 103–118. Springer, 1995.
3. E. Best and B. Grahlmann. *PEP: Documentation and User Guide*. Universität Hildesheim. Available together with the tool via: http://theoretica.informatik.uni-oldenburg.de/∼pep.
4. E. Best and R. P. Hopkins. $B(PN)^2$ – a Basic Petri Net Programming Notation. In *Proc. of PARLE*, *LNCS* 694, 379–390, Springer, 1993.
5. A. Bouali, A. Ressouche, V. Roy, and R. de Simone. The FC2tools Set. In *Proc. of CAV'96*, *LNCS* 1102, 441–444, Springer, 1996.
6. CCITT. *Specification and Description Language*, CCITT Z.100, Geneva, 1992.
7. E. Clarke, K. McMillan, S. Campos, and V. Hartonas-Garmhausen. Symbolic Model Checking. In *Proc. of CAV'96*, *LNCS* 1102, 419–422, Springer, 1996.
8. J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. In *Proc. of TACAS'96*, *LNCS* 1055, 87–106, Springer, 1996.
9. J. Esparza. *Model Checking Using Net Unfoldings.* In Number 23 in *Science of Computer Programming*, 151–195, Elsevier, 1994.
10. H. Fleischhack and B. Grahlmann. A Compositional Petri Net Semantics for SDL. In *Proc. of ATPN'98*, *LNCS 1420*, 144–164, Springer, 1998.
11. B. Grahlmann, M. Moeller, and U. Anhalt. A New Interface for the PEP Tool – Parallel Finite Automata. In *Proc. of AWPN'95*, AIS 22, 21–26. FB 10 Universität Oldenburg, 1995.
12. B. Grahlmann and C. Pohl. Profiting from Spin in PEP. In *Proc. of the SPIN'98 Workshop*, 1998.
13. B. Grahlmann. The Reference Component of PEP. In *Proc. of TACAS'97*, *LNCS 1217*, 65–80, Springer, 1997.
14. B. Grahlmann. Combining Finite Automata, Parallel Programs and SDL using Petri Nets. In *Proc. of TACAS'98*, *LNCS 1384*, 102–117, Springer, 1998.
15. B. Grahlmann. *Parallel Programs as Petri Nets.* Ph.D. thesis, Universität Hildesheim, 1998.
16. B. Graves. Computing Reachability Properties Hidden in Finite Net Unfoldings. In *Proc. of FST&TCS'97*, *LNCS 1346*, 327–341, Springer. 1997.
17. J. Jäger. *Portable Codegenerierung für eine parallele Programmiersprache.* Diploma thesis, Universität Hildesheim, 1997.
18. G. J. Holzmann and D. Peled. The State of SPIN. In *Proc. of CAV'96*, *LNCS 1102*, 385–389. Springer, 1996.
19. M. Kater. SimPEP: 3D-Visualisierung und Animation paralleler Prozesse. Diploma thesis, Universität Hildesheim, 1998.
20. S. Melzer and J. Esparza. Checking System Properties via Integer Programming. In *Proc. of ESOP'96*, *LNCS 1058*, 250–264, Springer, 1996.
21. S. Melzer and S. Römer. Deadlock Checking using Net Unfoldings. In *Proc. of CAV'97*, *LNCS 1254*, 352–363, Springer, 1997.
22. S. Roch and P. H. Starke. INA: Integrated Net Analyzer Version 2.1 Handbuch. Humboldt-Universität zu Berlin, 1998. Available together with the tool via: http://www.informatik.hu-berlin.de/lehrstuehle/automaten/ina.
23. G. Wimmel. A BDD-based Model Checker for the PEP Tool. *Technical Report, University of Newcastle upon Tyne*, 1997. Available at PEP's Web page.