System Specification and Verification Using High Level Concepts – A Tool Demonstration

Christian Stehno

Carl von Ossietzky Universität Oldenburg
FB Informatik, Parallel systems group
D-26111 Oldenburg
Christian.Stehno@informatik.uni-oldenburg.de

Abstract. This paper describes a sample modelling and verification session using SDL and SPIN modelchecker via the PEP tool[1]. We will focus on the tight integration of all involved tools allowing the user to stay within his known environment of SDL specifications. Thus the user need not know about the underlying Petri net or the Promela language even while formulating the properties to be checked.

1 Introduction and motivation

The PEP tool [1] provides an integrated development and verification environment for a selection of formal modelling techniques, including the Specification and Description Language (SDL, [10]). SDL is widely used in industry. It provides synchronous and asynchronous channels for communication of different processes, that run in parallel. In addition to the usual parts of most languages, like variables and control flow with choice and sequence, SDL also offers a procedure concept and dynamic process creation during runtime. This facilitates the system development and allows compact and readable models.

To support the user during the development process and further while verifying the model, it does not suffice to group different tools in one user interface. Instead, all tools have to be tightly connected making use of all features from a single point of view. It is usually best to stay at the top level of system description for ease of use and understanding, i.e. to allow a user to stay within his known environment even for verification purposes. Thus all of the involved steps have to relay their results to upper levels, providing simulation, verification and debugging in terms of the specification language.

This paper presents features offered by the PEP tool, that support all topics mentioned above, with an emphasis on temporal logics. It is structured as follows: Section 2 describes the modelling and simulation of SDL specifications. The verification of such specifications are presented in Sec. 3, while Sec. 4 concludes the paper and shows possible further developments.

[1] http://parsys.informatik.uni-oldenburg.de/~pep

2 High level modelling

An SDL system may be directly modelled within the PEP interface by entering SDL code in PEP's text editor, but may also be read from external specifications. The editor allows the selection of most SDL language blocks with the mouse and offers online syntax checking.

Before further action can take place, the SDL specification is translated into an M-net [2] representing its formal semantics. In conjunction with this and following transformations, a set of references is created [7], providing feedback from lower levels to the original specification and facilitating the methods described in this paper.

The first step to occur after the specification is usually simulation. Simple design flaws and unwanted behaviour may be detected this way. Due to the references, it is not only possible to simulate the net and gain the SDL behaviour from annotations, but also to simulate the SDL program directly. Simulation cannot guarantee properties though, it only helps in understanding the system. To verify properties for all possible states of the system, model checking provides an effective and widely used method. The next section will present some algorithms and PEP support of them.

3 Verification of the specification

Various verification tools are integrated into the PEP tool to offer the user a large base of formal concepts to check properties of the system, e.g. partial order representation [4] and BDD based [11] algorithms. The SPIN tool [9] is used to verify LTL formulae over Petri net state properties.

To transform the high level M-net into Promela code, the net is first unfolded into a semantically equivalent low level net and subsequently compiled into Promela code according to [8]. This yields a SPIN compatible process version of the Petri net which emulates the net behaviour.

Properties that may be checked have to be defined over Petri nets. Depending on the model checker used, the temporal logic is determined. The SPIN tool provides LTL checking, while other tools may be used for branching time logics. The formula may be entered in the formula editor shown in Fig. 1. This editor not only offers a simple text entry to enter and save terms, but also allows the creation of formulae using the mouse. As shown in Fig. 1, all syntactical components of the logic are choosable.

While simple properties may be formulated directly in terms of net entities, e.g. [](P1), stating ”the place P1 is always marked”, this method is tedious and error prone for more complex ones. Instead, a high-level syntax allows to state explicitly model properties within the formulae, which are automatically transformed into net formulae, using the mentioned references. A sample formula using this concept is shown in (1), stating the property ”The send state in client process 1 is always reachable again” (liveness).

$$[\,] <> (\text{client}[1].\text{state} = \text{send}) \tag{1}$$

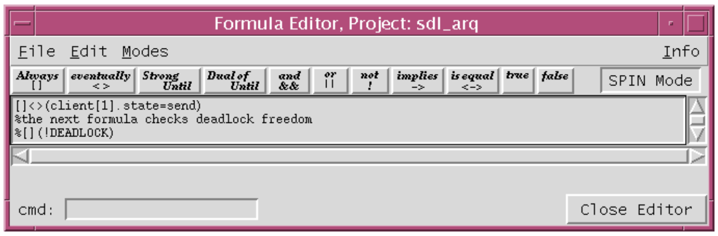

Fig. 1. Formula editor in PEP

To use these types of formulae, PEP provides two different mechanisms. The first is a reference mode of the SDL editor. The user may select states or components of the SDL specification and send their corresponding places into the formula editor. The second approach allows the high level terms to be entered into the formula and thus making it invariant to changes of the underlying Petri net, as the corresponding places are regenerated each time. Both techniques are available at the same time, so the user may choose depending on his needs.

Continuing the verification process, the formula is expanded into the corresponding net formula if some high level terms were used. Then it is translated into a "never claim" and included in the promela code. The model checking interface is started via the SPIN button and creates a window (cf. Fig. 2) offering a result display and buttons to change SPIN options and start checking. When the SPIN tool is invoked, the promela code is compiled into a binary which is automatically executed. The result is displayed in the model checker window. Additionally to the complete SPIN output, a transition sequence of the Petri net is calculated where applicable as shown in Fig. 2. Using the references ranging from the low level Petri net back to the SDL specification, the user may simulate the counterexample (if one is found) not only in the Petri net, but also in the original specification as defined at the beginning of the session. This allows a debugging technique solely based on high level terms of the chosen modelling language, reducing the efforts of users to a minimum.

4 Conclusion and future work

We have briefly presented the features of the PEP tool supporting the entire modelling process of SDL specifications, including simulation and verification, at the abstract SDL level. The user does not have to cope with different formal models but gets all results in terms of the specification. The development process will be completed by a graphical editor, that is currently being implemented, and an extension of the SDL fragment used towards SDL-2000.

While the PEP tool supports the transformation of low level Petri nets to Promela code only, [8] describes also possible transformation of high level nets and code of $B(PN)^2$ [3]. A further, very promising direction is the translation of time Petri nets into Promela code. Due to the variable concept in Promela it should be fairly easy to add this functionality.

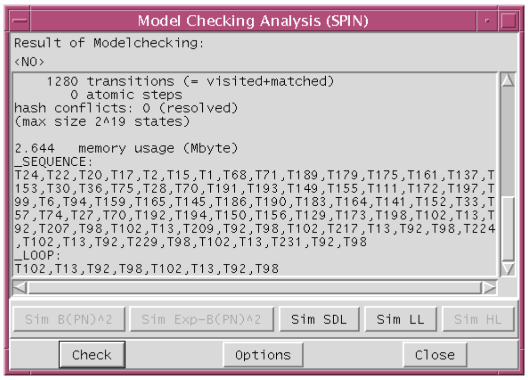

Fig. 2. Result of SPIN and counterexample

The PEP tool is currently extended by some additional LTL model checkers from [5] and [12]. This will give an opportunity to compare the different approaches and choose the algorithms best suiting the particular problem.

Acknowledgements. The author would like to thank Eike Best and an anonymous referee for their comments on this paper.

References

1. Best, E.: Partial Order Verification with PEP. In Proc. of POMIV'96. Am. Math. Soc. (1996) 305–328
2. Best, E., Fleischhack, H., Frączak, W., Hopkins, R.P., Klaudel, H., Pelz, E.: A Class of Composable High Level Petri Nets. In Proc. of ATPN'95. Volume 935 of LNCS. Springer-Verlag (1995) 103–118
3. Best, E., Hopkins, R.P.: $B(PN)^2$– a Basic Petri Net Programming Notation. Proc. of PARLE'93. Volume 694 of LNCS. Springer-Verlag (1993) 379–390
4. Esparza, J.: Model Checking Using Net Unfoldings. Science of Computer Programming **23**. Elsevier (1994) 151–195
5. Esparza, J., and Heljanko, K.: A new unfolding approach to LTL model checking. In Proc. ICALP 2000. Volume 1853 of LNCS. Springer-Verlag (2000) 475–486
6. Fleischhack, H., Grahlmann, B.: A Compositional Petri Net Semantics for SDL. In Proc. of ATPN'98. Volume 1420 of LNCS. Springer-Verlag (1998)
7. Grahlmann, B.: The Reference Component of PEP. In Proc. of TACAS'97. Volume 1254 of LNCS. Springer-Verlag (1997) 65–80
8. Grahlmann, B., Pohl, C.: Profiting from SPIN in PEP. In Proc. of SPIN'98. (1998)
9. Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice Hall (1991)
10. CCITT: Functional Specification and Description Language (SDL). CCITT Z.100. Geneva (1992)
11. McMillan, K.: Symbolic model checking: An approach to the state explosion problem. Kluwer Academic Publishers (1993)
12. Wallner, F.: Model checking LTL using net unfoldings. In Proc. of CAV'98. Volume 1427 of LNCS. Springer-Verlag (1998) 207–218