Proc. 3rd Workshop on Discrete Event Systems (WoDES '96), Edingburgh/UK, August 1996, pp. 308-313.

PETRI NET BASED DESIGN AND ANALYSIS OF REACTIVE SYSTEMS

Monika Heiner, Peter Deussen

Brandenburg Technical University of Cottbus
Department of Computer Science
Postbox 101344
D-03013 Cottbus
Germany
mh@informatik.tu-cottbus.de, pd@informatik.tu-cottbus.de
Phone: (+ 49 - 355) 69 - 3885
Fax: (+ 49 - 355) 69 - 3830

Abstract. The development of provably error-free concurrent systems is still a challenge of system engineering. Modelling and analysis of concurrent systems by means of Petri nets is one of the well-known approaches using formal methods. To evaluate the reached practicability degree of available methods and tools to at least medium-sized systems, the authors demonstrate the step-wise development and validation of the control software of a reactive system [13]. Strong emphasis has been laid on automation of the analyses to be done. This paper provides a brief outline of the authors' work, stressing especially analysis experience.

Keywords: Parallel software/system engineering, static analysis, Petri nets, reactive system, reliability.

1 Introduction

The development of provably error-free concurrent systems is still a challenge of system engineering. Modelling and analysis of concurrent systems by means of Petri nets is one of the well-known approaches using formal methods. To evaluate the reached practicability degree of available methods and tools to at least medium-sized systems, the authors demonstrate in [13] the step-wise development and validation of the control software of a reactive system - a (really existing) production cell in a metal-processing plant [14]. In this paper, a brief outline of our work is presented, updated by recent analysis results.

The validation of qualitative properties comprises two steps. At first, the context checking of general semantic properties is done by a suitable combination of static and dynamic analysis techniques, mainly of "classical" Petri net theory. Afterwards, the verification of well-defined special semantic properties, especially safety properties, given by a separate specification of the required functionality is performed. Strong emphasis has been laid on automation of the analyses to be done.

The case study has been designed to evaluate different kinds of software development methods. The objective is to develop verified control software, taking into account various safety conditions and performance constraints. This case study has been already investigated by a lot of different formal methods [14], but up to now Petri nets have been used only as supplementing description. This paper provides the possibility to consider also a strongly Petri net-oriented approach to compare advantages and drawbacks inherent to different formal methods.

This paper is organized as follows. Section 2 gives a short overview of the Petri net based framework of software validation and introduces some basic terminology. The essential points of the discussed case study concerning the task description and the modelling of the controller components for the production cell are presented in section 3. Section 4 gives a classification of properties of reactive systems derived from our experience, and related analysis methods are discussed. A short overview of the main analysis results obtained by now is presented in section 5. Finally, section 6 comprises an outlook on further work in preparation and open questions, which have to be solved in order to improve the Petri net based validation.

2 Petri net based framework of software validation

In our case study, Petri nets are constructed right from the beginning to model and prototype the concurrent aspects of the system under development in order to be able to predict, at the chosen abstraction level, the possible behaviour of the system by intermediate validation steps. After being satisfied with the analysis result obtained, the executable code (of the communication skeleton) has been generated based on [18].

Generally spoken, validation tries to minimize the presence of faults in the operation phase by analytical and (as far as possible) computer-aided methods in the pre-operation phase.

Concerning the type of properties to be validated, two classes of qualitative validation techniques can be distinguished:

Context checking deals with general qualitative

properties like freedom from data or control flow anomalies which must be valid in any system independent of its special semantics (for that reason, it is called in the following *general analysis*). These properties are generally accepted or project-oriented consistency conditions of the static semantics of any program structure like boundedness and liveness.

Verification aims at special qualitative properties like functionality or robustness which are determined by the intended special semantics of the system under development (to underline this fact, it is called in the following *special analysis*).

Evidently, a successful general analysis is a prerequisite to prove that some special qualitative properties will be fulfilled under any circumstances. So, the validation of qualitative properties can be divided into two consecutive steps, which supplement each other. First, the context checking of general semantic properties (general analysis) has to be done by a suitable combination of static and dynamic analysis techniques of Petri net theory.

Afterwards, the verification of well-defined special semantic properties (special analysis) given by a separate specification of the required functionality has to be performed. Especially for the second step it is very useful to supplement the power of "classical" Petri net theory by the model checking approach, using temporal logic as a flexible query language for asking questions about the (complete/reduced) set of reachable states.

All **static analysis** techniques have in common that they avoid the construction of the reachability graph. Reduction and structural analysis (e.g. the deadlock trap property) correspond mainly to general analysis (of (un-) boundedness or (un-) liveness). The invariant analysis supports general analysis (if the net is covered with semipositive place invariants) as well as special analysis. In the latter case, program invariants are proven by showing the existence of related net invariants. So first, suitable program invariants have to be hypothesized, and second, the related net invariants have to be found from the (in general non-minimal) basis of invariants provided by a net analysis tool. Generally, this is hardly manageable for larger systems (larger concerning the size of states).

Dynamic analysis techniques have to be used if the static analysis efforts were not successful, or if properties are wanted which cannot be analyzed statically at all (e.g. reversibility, livelock freedom, dynamic conflicts - especially those conflicts, where communication places are involved, firing of facts, etc.). Due to the well-known effect of state explosion, lazy construction methods to consider only reduced state spaces have been elaborated. E.g. the stubborn set reduced reachability graph is usually much smaller than the complete one, but exhibits all dead states, if any.

In **model checking**, the reachability graph (in different versions) of a Petri net is interpreted as a model of a set of formulae given in some formal languages. In the recent years, temporal logics are broadly accepted as a suitable formalism to express properties of reactive systems.

The tool kit used consists of the following components: PED [16], INA [19], PROD [24], and PEP [2].

The graphical Petri net editor **PED** supports basically the construction of hierarchical place/transition nets. Complementary, all necessary attributes of those net types can be assigned, which are analyzable by INA.

INA provides a broad offer of analysis methods of "classical" Petri net theory, e.g. static analysis techniques like testing the deadlock trap property or state machine coverability, invariant analysis, computation of symmetries, structural reduction, and dynamic analysis techniques like reachability/coverability graph generation and stubborn set reduced reachability graph construction.

The reachability graph generator **PROD** provides a query language which is strong enough to express a fully version of propositional *Computational Tree Logic* (CTL) [1]. Unfortunately, the evaluation of CTL formulae requires the previous construction of the complete state space of a Petri net which is unaccaptable at least for medium sized systems (state explosion problem).

Another type of temporal logics provided by PROD is a version of *Linear Time Temporal Logic* (LTL) (see e. g. [6]). LTL formulae not containing the nexttime operator can be checked very efficiently (on-the-fly) by construction of a reduced state space which is in so-called *CFFD-equivalence* to the complete state space using the stubborn set method (see [5, 9, 21, 22, 23]).

Instead of computing a complete or reduced version of the reachability graph, the model checking component of the **PEP** tool is based on the construction of a finite prefix of an unfolding of a Petri net, which does not contain interleavings of concurrent transition occurrences [7, 8, 15]. However, this method works only for 1-bounded nets. PEP provides model checking for a rather restricted version of CTL containing only the temporal operators AG and EF. But, generation of the finite prefix as well as model checking work extraordinary fast.

3 Case study

The focus of our investigations builds a (really existing) industrial facility [14]. This production cell comprises six physical components: two conveyor belts, a rotatable robot equipped with two extendable arms, an elevating rotatable table, a press, and a travelling crane. The machines are organized in a (closed) pipeline. Their common goal is the transport and transformation of metal plates. For details the reader is referred to [14]. In this paper, also a list of safety and progress requirements can be found which are to obey by the implementation of a control program.

We have developed and analyzed the control software in two abstraction levels. The more abstract **cooperation**

model describes the synchronization of the machine controllers. This model has been influenced essentially by [3]. The construction of the model was done stepwise. At first, common patterns concerning the intended communication behaviour of the controllers for the physical devices have been identified and modelled as Petri nets. Then, the complete model was constructed by composition of instances of the communication patterns by merging the so-called communication places. The total cooperation model consists of 51 places and 36 transitions structured into eight hierachical sheets.

After designing and analyzing successfully the cooperation model, a refinement has been done which incorporates the interactions of the controllers with the interface (actuators, sensors) of the production cell. Furthermore, this **control model** comprises a Petri net description of the environment which allows to express certain assumptions on the behaviour of the physical devices in response to the controllers' actions. As before, the construction of the model was done bottom-up: A net structure for an elementary control procedure was defined which involves the controller part and also the environment part of one basic motion step of one device. One basic step consists of the activation of some device, receiving a certain sensor value that indicates that the motion is completed, and the deactivation of the device. Complex processing step controls where constructed by the composition of elementary ones. The cooperation model is structured into 65 sheets. It comprises 231 places and 202 transitions.

4 Properties of reactive systems

At the very first beginning, a specification of a reactive system is given by an informal list of functionality and safety requirements. If a compositional design methodology is used to build up a formal model of the required system, the specification process may be stopped at an arbitrary level of refinement, where complex system components are considered to be atomic ones, and internal details are left to the actual implementation. Hence, the question arises whether this model is adequate to express the required system properties. In our case study, for instance, the cooperation model does not contain any references to the environment. Therefore, the influence of the system on the devices to be controlled is only described indirectly by assumptions on the relation between internal system states and external states (of the environment).

Basically, two different types of system properties can be distinguished:

1. Properties related to system design. This class of properties consists both of general qualitative properties like freedom of dead states, liveness of transitions, or boundedness, and of special properties demanded by system design like (if Petri nets are used as formal specification language) mutual exclusion of pairs of places modelling a message/acknowledgement behaviour of communication channels. Properties of this type can be given in terms of system states itselves without further concerning (implicit or explicit) assumptions on the environment behaviour.

2. Properties related to the influence of a system on its environment. For reactive systems, a specification of the required functionality and safety constrains must be given in terms of the actions of a system in response to external events. Hence, the expressibility of those properties depends on the existence of appropriate terms of the environment in the formal system description. These terms can be given either implicitly by means of assumptions on the relations between internal system states and external states of the environment or explicitly by incorporating a reasonable description of the environment itself.

In the case of control software of manufacturing systems, a refined classification of interaction-related safety properties can be made:

(a) Illegal states of single components. For instance, devices may have restricted mobility. To ensure requirements of this type, certain system actions have to occur fast enough in response to external events. If external errors are observable by the control program, properties of this type are expressible by adding appropriate error states to the environment model. Otherwise, the system model has to contain assumptions on execution times.

(b) Illegal combinations of single components states. For instance, certain combinations of states of the devices result into machine collision. In general, compositional system design ensures that errors of this kind can be related to internal system states and/or to states of an external environment model. Using "classical" Petri net based analysis techniques, properties of this type can be verified e.g. by place invariants. In temporal logics, these properties appear as simple safety formulae of the logical form $\mathsf{AG}(\neg\varphi)$ (in CTL, φ a state formula), which are expressible both by the versions of CTL and LTL supported by PROD and by the CTL version of PEP.

(c) Illegal (sequences of) system actions. For instance, an unloading action of a transport device may not occur unless a transport motion is completed. Most of the analyzing techniques which are supported by the tools cited above are based on the reachability or non-reachability of (classes of) system states. Therefore, requirements of this type are only expressible indirectly in terms of (sets of) system states where such unwanted action (sequences) would be enabled. In CTL, formulae expressing this type of properties are in general of the form $\mathsf{AG}(\varphi \rightarrow \mathsf{A}[\neg\chi\ \mathsf{U}\ \psi])$, i.e. at some state determined by φ, an unwanted system state determined by χ does not occur until ψ holds. Formulae of this logical form can be expressed in PROD's CTL version and also in PROD's LTL version. But PEP's version of CTL is not powerful enough for this purpose.

Table 1: Size of analyzed nets and analysis efforts done (cooperation model).

	places/ transitions	Analysis (INA)			Execution Times		
		DTP[a]	R_{stub}[b]	R[c]	INA[d]	INA[e]	PROD[f]
controllers							
table / press with init part without init part	 13 / 9 12 / 8	 (N) 28	 12 8	 28 24	 1.5" 1.46"	 1.54" 2.16"	 7.96" 7.41"
crane	12 / 8	31	11	48	1.62"	2.12"	7.39"
arm version 1 version 2 version 3	 13 / 8 17 / 12 17 / 12	 38 109 88	 11 15 15	 48 112 96	 1.5" 1.73" 1.77"	 3.12" 1.65" 1.88"	 14.78" 24.92" 24.19"
belts	12 / 8	26	8	36	1.58"	2.23"	7.38"
composed systems							
robot (arm version 3)	33/24	448	221	10,944	2190.18"	11.34"	75.6"
robot/press with arm version 1 arm version 2 arm version 3	 25 / 16 33 / 24 33 / 24	 175 3.851 (N) 725	 47 75 140	 640 1,984 1,800	 7.81" 53.23" 57.07"	 3.64" 7.78" 8.12"	 14.78" 24.92" 24.19"
open system	51 / 36	1145	299	77,760	86132.6"	3653.6"	1073.35"
closed system with 1 plate with 2 plates with 3 plates with 4 plates with 5 plates	51 / 36	1140	 36 72 94 98 121	 864 4,776 12,102 16,362 12,144	 18.58" 365.53" 2401.28" 4167.93" 2373.1"	 6.07" 18.77" 87.1" 150.85" 78.6"	 25.93" 56.21" 125.71" 169.15" 123.8"

a) processed candidates to check the deadlock trap property
b) number of states of the stubborn reduced reachability graph $\mathbf{R}_{stub}$
c) number of states of the complete reachability graph **R**
d) time effort to generate **R** (with coverability test)
e) time effort to generate **R** (without coverability test)
f) time effort to generate **R**

5 Summary of the main analysis results

Cooperation model. In general analysis, we are basically interested in proving the general properties boundedness and liveness. Using INA, both properties can be decided efficiently by showing that the net under consideration is covered by semipositive place invariants and by proving the deadlock trap property, respectively, because of certain structural properties of the considered net models (marked graphs or extended simple nets). Dead states in one discussed controller version (arm version 2) have been found very fast by the stubborn set reduction method. Table 1 compiles some details on the size of the analyzed nets and the analysis efforts done. Please note especially the impressive reduction effect inherent in the stubborn set method. The analyses were done on a SUN sparc station 5 with 32 MB main memory.

The reachability graph analyser PROD has been applied to prove certain safety properties of the cooperation model given by CTL formulae. Two major problems have been arisen:

(a) The expressibility of certain properties. Due to the lack of an explicit environment model, requirements, which are given explicitly in terms of the system's interaction with the environment (like sensor values), cannot be expressed only in terms of internal system states. Hence, the confidence in the verification depends on the degree of assurance that the assumed relation between internal states and the corresponding controlled processes will be correctly realized by the final implementation of the modelled system.

(b) The time effort for model checking. Safety properties of the general logical form $AG(\neg\varphi)$ (φ a state formula) can be checked within an acceptable amount of time (about a few minutes). For systems with a medium sized state space, the time effort to validate more complex formulae like progress properties of the form $AG(\varphi \rightarrow AF\chi)$ (φ, χ state formulae) is extraordinary large (about ten hours on a SUN sparc station 20).

Table 2: Size of analyzed nets and analysis efforts done (control model).

	places/ transitions	PEP		PROD					
		conditions/ events[a]	time[b]	R[c]	time[d]	R_{stub}[e]	time[f]	R_{stub}[g]	time[h]
controllers									
crane	45/34	154/71	0.02"	256	0.78"	51	0.16"	38	0.08"
feed belt	22/16	69/34	0.01"	256	0.20"	31	0.10"	16	0.07"
table	32/24	82/37	0.01"	88	0.38"	36	0.15"	24	0.09"
arm 1/2 (version 3)	66/60	138/65	0.02"	365"	1.19"	62	0.23"	51	0.09"
press	28/20	166/81	0.02"	140	0.42"	48	0.10"	20	0.09"
deposit belt	22/16	69/34	0.01"	69	0.31"	31	0.11"	16	0.07"
composed systems									
robot	124/120	3514/1752	0.02"	63,232	11.26'	992	5.99"	205	0.21"
robot/press	140/132	1280/624	1.07"	18,344	3.10"	557	3.46"	305	0.35"
open system	198/176	2773/1348	5.15"	?	?	798	5.90"	507	0.62"
closed system	231/202								
with 1 plate		690/316	0.57"	30,952	7.54'	162	0.68"	163	0.32"
with 2 plates		1670/792	2.63"	543,480	ca. 3.3 h	406	2.53"	456	0.72"
with 3 plates		2009/960	3.02"	> 1.7 Mio	> 20 h	523	4.51"	635	0.95"
with 4 plates		2164/1035	3.38"	> 3.1 Mio	> 42 h	471	4.02"	678	1.06"
with 5 plates		1619/768	1.68"	1,657,242	ca. 14 h	585	5.05"	608	0.98"

a) size of the finite prefix of the branching process (net unfolding)
b) time effort to generate the finite prefix
c) number of states of the complete reachability graph **R**
d) time effort to generate **R**
e) number of states of the stubborn reduced reachability graph $\mathbf{R_{stub}}$ using the *deletion algorithm*
f) time effort to generate $\mathbf{R_{stub}}$
g) number of states of the stubborn reduced reachability graph $\mathbf{R_{stub}}$ using the *incremental algorithm*
h) time effort to generate $\mathbf{R_{stub}}$

Control model. Again, INA and PROD were applied for general and special analysis. For the control model, a complete construction of it's state space became unacceptable. (See table 2 for a comparison of the analysis efforts using PROD and PEP. The analyses were done on a sparc station 20 with 64 MB main memory.) In spite of this, boundedness is still decidable very efficiently by showing that the net model is covered by semipositive place invariants (again it took only a few seconds). Due to the added environment behaviour, all discussed net models exhibit a net structure beyond the extended simple one. So, the deadlock trap property could only show the freedom of dead states. The construction of the stubborn set reduced reachability graph is still manageable even for larger systems with unknown size of the complete state space. So, the prove of the deadlock freedom seems to be practicable in any case.

It was impossible to prove liveness using INA or PROD, because of the lack of suitable structural properties and the size of the state space. The liveness of a transition t can be expressed in CTL by the formula $\mathsf{AGEF}\,en(t)$ ($en(t)$ denotes the proposition that all preconditions of a transition t are satisfied). But this property is not expressible in LTL and therefore not analyzable by PROD on the fly. In LTL, only a stronger property related to the livelock freedom of a transition can be expressed by $\mathsf{GF}\,en(t)$, and actually analyzed by PROD's on-the-fly verification. Unfortunately, not (but almost) all transitions in the control model enjoy this property. As cited above, the CTL version of PEP contains both the operators AG and EF. Hence, liveness is expressible in this logic. We succeeded in proving the liveness of non-livelock-free transitions in all cases within an negligible amount of time (less then one second).

Safety requirements were expressed in LTL and verified using PROD's stubborn set based on-the-fly verification method. A few simple safety properties were verified by PEP. The following conclusions are worth mentioning:

(a) The existence of an environment model allows to express both assumptions on the behaviour of physical machines and of system properties in terms of the influence of a system on its environment. Evidently,

requirements expressed in this way are comprehensible without a deeper knowledge of the internal structure of a system.

(b) Because of the structure of the environment model, it is impossible to express properties related to illegal states of single devices. A more improved model must contain explicit error states.

(c) The stubborn set based on-the-fly verification method has been proven to be applicable even for systems with a larger state space. The analysis efforts were between 2 and 25 minutes for the formulae for which a model checking were done. However, certain progress properties like the liveness of transitions cannot be expressed in LTL because of the lack of a quantification on computation paths.

(d) Liveness of transitions and simple safety requirements can be verified very fast by PEP's model checking method. However, many of those system requirements related to illegal action sequences cannot be expressed in the restricted CTL version of PEP.

6 Final remarks

Up to now, a Petri net model to control the given production cell has been developed which enjoys provably a lot of valuable qualitative properties - general as well as special ones.

Beyond that, the quantitative analysis is in preparation using different types of time-dependent Petri Nets: Duration Interval Nets [12] based on Interval Nets [17] to prove the meeting of given deadlines by worst-case evaluation, and Stochastic Nets [25] to estimate performance measures like throughput or average processing time.

Throughout our case study, (the rarely available and rather restrictive) compositional approaches of Petri net analysis have not been discussed yet. They have been skipped in order to get a feeling for the borders of those net/state space sizes, which are actually manageable by available analysis tools.

Finally, the main lessons learnt concerning a suitable tool box framework are the following.

(a) The combination of different tools (even if they provide similar features at the very first glance) seems to be unavoidable.

(b) We need user guidelines showing which analysis techniques are recommendable for a given analysis question.

(c) The check of a given system against its functionality and/or safety requirements given by a (more or less large) set of temporal formulae calls for distributed evaluations in batch processing manner.

References

[1] BEN-ARI, M., PNUELI, A, MANNA, Z, The Temporal Logic of Branching Time, Acta Inform. **20**(1983), 207-226.

[2] BEST, E., GRAHLMANN, B., PEP - Programming Environment Based on Petri Nets, Documentation and User Guide: Univ. Hildesheim, Institut für Informatik, Nov. 1995.

[3] CASAIS, E., Eiffel; A Reusable Framework for Production Cells Developed with an Object-oriented Programming Language, in: Lewerentz, C., Lindner, T., ed.: *Case Study "Production Cell" A Comparative Study in Formal Software Development*, FZI-Publication 1/94, Forschungszentrum Informatik, Karlsruhe 1994, 241-256.

[4] CLARKE, E. M., EMERSON, E. A., SISTLA, A. P., Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, ACM Trans. on Programming Languages and Systems **8**(1986)2, 244-263.

[5] COURCOUBETIS, C., VARDI, M. Y., WOLPER, P., YANNAKAKIS, M., Memory Efficient Algorithms for the Verification of Temporal Properties, Formal Methods in System Design **1**, 2/3 (1992), 275-288.

[6] EMERSON, E. A., Temporal and Modal Logic, in: J. v. Leeuwen, ed.: *Handbook of Theoretical Computer Science*, Vol. B, Elsivier, Amsterdam 1990, 995-1072.

[7] ENGELFRIET, J., Branching Processes of Petri Nets, Acta Inform. 25(1991), 575-591.

[8] ESPARZA, J., Model Checking Using Net Unfoldings, Science of Computer Programming **23**(1994), 151-195.

[9] GERTH, R., PELED, D., VARDI, M. Y., WOLPER, P., Simple On-the-fly Automatic Verification of Linear Temporal Logic, in: *Proc. of the 15th International Symposium on Protocol Specification, Testing and Verification (PSTV'95)*, Warsaw, 1995, 3-18.

[10] HEINER, M., Petri Net Based Software Validation, Prospects and Limitations, ICSI-TR-92-022, Berkeley/CA, 3/1992.

[11] HEINER, M., VENTRE, G., WIKARSKI, D., A Petri Net Based Methodology to Integrate Qualitative and Quantitative Analysis, Information and Software Technology **36**(94)7, 435-441.

[12] HEINER, M., Petri Net Based Software Dependability Engineering, Tutorial Notes, Int. Symposium on Software Reliability Engineering, Toulouse, Oct. 1995.

[13] HEINER, M., DEUSSEN, P., Petri Net Based Qualitative Analysis - a Case Study, Techn. Report BTU Cottbus, I-08/1995.

[14] LEWERENTZ, C., LINDNER, T., *Formal Development of Reactive Systems - Case Study Production Cell*, LNCS 891, 1995.

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

[16] PED, http://www-dssz.Informatik.TU-Cottbus.De/~wwwdssz/ped.html

[17] POPOVA, L., On Time Petri Nets, J. Information Processing and Cybernetics EIK **27**(1991)4, 227-244.

[18] SCHWIDDER, K., Petri Net Based Modelling and Simulation of Automation Techniques' Discrete Processes (in German); in Scheschonk, G.; Reisig, W. (eds.): *Petri Net Applications for Design and Development of Information Systems*, Springer 1993, pp. 209-221.

[19] STARKE, P. H., INA - Integrated Net Analyzer, Manual (in German), Berlin 1992.STARKE, P. H., *Analysis of Petri Net Models* (in German), Teubner, Stuttgart 1990.

[20] STARKE, P. H., *Analysis of Petri Net Models* (in German), Teubner, Stuttgart 1990.

[21] VALMARI, A., A Stubborn Attack on State Explosion, Formal Methods in System Design **1**, (1992) 4, 297-322.

[22] VALMARI, A., Alleviating State Explosion during Verification of Behavioral Equivalence, Univ. of Helsinki, Department of Computer Science, Report A-1992-4, Helsinki 1992.

[23] VARPAANIEMI, K., On Computing Symmetries and Stubborn Sets, Helsinki Univ. of Technology, Digital Systems Laboratory Report B 12, Espoo 1994.

[24] VARPAANIEMI, K., HALME, J., HIEKKANEN, K., PYSSYSALO, T., PROD Reference Manual, Helsinki Univ. of Technology, Digital Systems Laboratory, Series B: Techn. Report No. 13, August 1995.

[25] WIKARSKI, D., HEINER, M., On the Application of Markovian Object Nets to Integrated Qualitative and Quantitative Software Analysis; Fraunhofer ISST, Berlin, ISST-Berichte 29/95, Oct. 1995.