The SH-Verification Tool

Peter Ochsenschläger and **Jürgen Repp** and **Roland Rieke**
SIT – Institute for Secure Telecooperation,
GMD – German National Research Center for Information Technology,
Rheinstr. 75, D-64295 Darmstadt, Germany
E-Mail: {ochsenschlaeger,repp,rieke}@darmstadt.gmd.de

Abstract

The sh-verification tool supports a verification method for cooperating systems based on formal languages. It comprises computing abstractions of finite-state behaviour representations as well as automata and temporal logic based verification approaches. A small but typical example shows the steps for analysing its dynamic behaviour using the sh-verification tool.

Keywords: Cooperating Systems; Finite State Systems, Abstraction; Simple Language Homomorphisms; Formal Specification; Verification

Introduction

The sh-verification tool [1] supports the method for verification of cooperating systems described in (Ochsenschläger, Repp, Rieke 1999). The reader is referred to this paper for notations, definitions and theorems. Figure 1 shows the structure of the tool. The main components of the system are the tools for specification, the analysis kernel, the tools for abstraction and the project manager. It is possible to extend the tool by different application oriented user interfaces. A small but typical example shows the steps for analysing a systems behaviour using the sh-verification tool.

Specification

The presented verification method does not depend on a specific formal specification technique. For practical use the sh-verification tool has to be combined with a specification tool generating labeled transition systems LTS [2]. The current implementation uses product nets [3] (Burkhardt, Ochsenschläger, Prinoth 1989; Ochsenschläger Prinoth 1995) as specification environment. A second specification environment based on asynchronous product automata (APA), (Ochsenschläger *et al.* 1998) is planned.

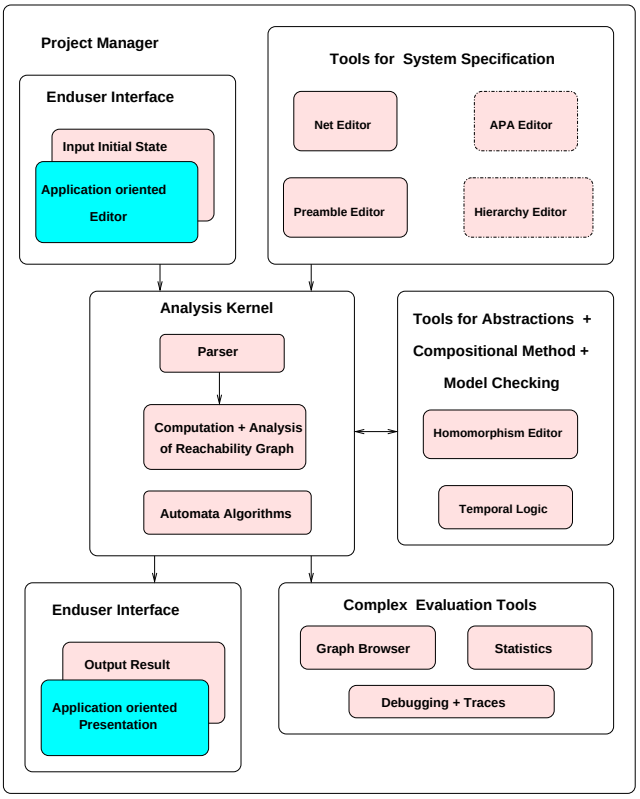

Figure 1: Components of the sh-verification tool

To illustrate the usage of the methods described in (Ochsenschläger, Repp, Rieke 1999) we consider an example of a system that consists of a client and a server as its main components. The client sends requests *REQ* to the server, expecting the server to produce particular results. Nevertheless, for some reasons, the server may not always respond to a request by sending

[1] sh abbreviates simple homomorphism

[2] The semantics of formal specification techniques for distributed systems is usually based on LTS.

[3] a special class of high level petri nets

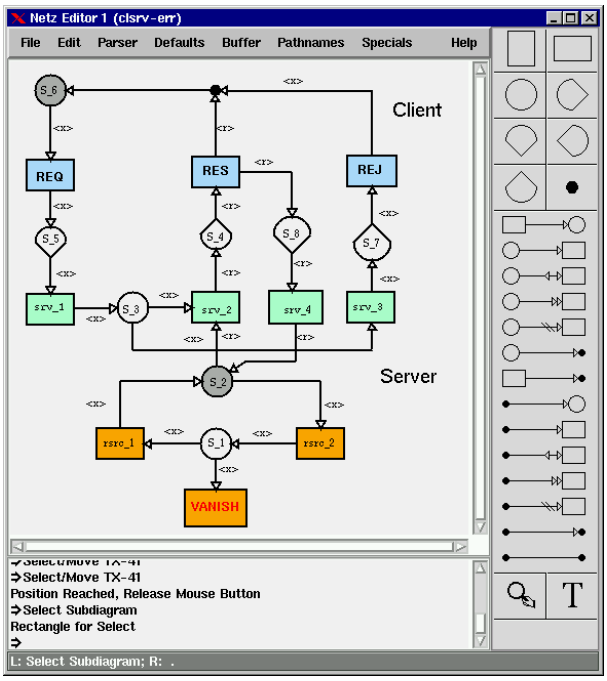

Figure 2: Client Server Example

a result *RES*, but may, as well, reject a request *REJ* (Figure 4).

Figure 2 shows a product net specification of this example. It is a global model for the systems behaviour. Note that the resource may eventually be locked forever. In Figure 2 we do not use most of product nets' possible features. Indeed, it is a product net representation of a Petri net.

Usually complex systems are specified hierarchically. This is supported by the *project manager* of the tool. (In our simple example the specification is flat.)

The LTS in Figure 3, which is the reachability graph of the product net in Figure 2, is computed by the tool. This LTS consists of two strongly connected components (marked by diffent colors). Usually the LTS of a specification is too complex for a complete graphical presentation; there are several features to inspect the LTS.

Abstraction

In the example the important actions with respect to the client's behaviour, are sending a request and receiving a result or rejection.

We will regard the whole system running properly, *if the client, at no time, is prohibited completely from receiving a result after having sent a request* (correctness criterion).

For the moment, we regard the server as a black box; i.e. we neither consider its internal structure nor look

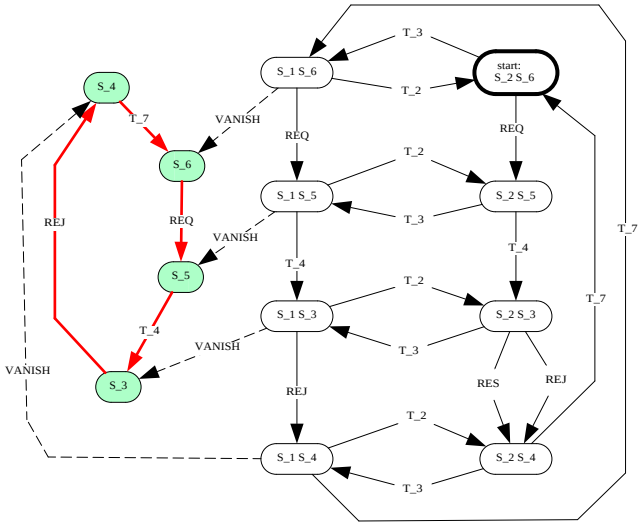

Figure 3: LTS

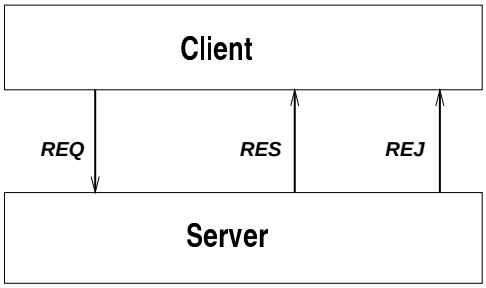

Figure 4: Client Server Abstract View

at its internal actions. Not caring about particular actions of a specification when regarding the specification's behaviour is *behaviour abstraction*. If we define a suitable abstraction for the client/server system with respect to our correctness criterion, we only keep actions *REQ*, *RES*, and *REJ* visible, hiding all other actions. This is supported by the homomorphism editor of the tool (Figure 5).

An automaton [4] representing the abstract behaviour of the specification can be computed by the sh-verification tool (Figure 6). It obviously satisfies the required property. The next step is to check whether the concrete behaviour also satisfies the correctness requirement mentioned above. For that purpose we have to prove simplicity of the defined homomorphism.

Simplicity of an abstraction can be investigated inspecting the strongly connected components of the LTS by a sufficient condition (Ochsenschläger, Repp, Rieke 1999). The component graph in Figure 7 (combined with the homomorphic images of the arc labels of the

[4] *the minimal automaton*

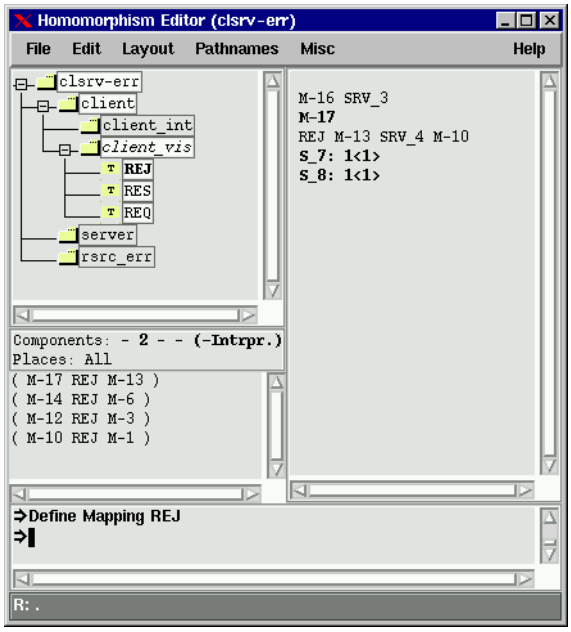

Figure 5: Defining an Abstraction

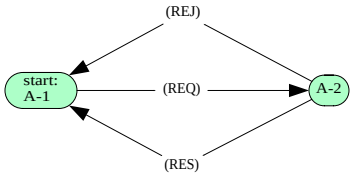

Figure 6: Minimal Automaton.

corresponding graph components) does not satisfy this condition, so nothing can be said about simplicity.

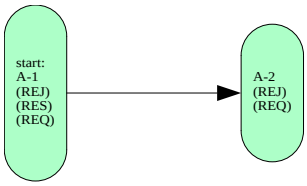

Figure 7: Component Graph

We now try to refine the homomorphism such that the sufficient condition for simplicity can be proven. Inspecting the edge between the two nodes of the component graph shows that the action $VANISH$ causes the transitions between this two components (Figure 8). The refined homomorphism, which additionally keeps $VANISH$ visible, satisfies the sufficient condition for simplicity. Figure 9 shows the corresponding automaton. This automaton obviously violates the required property, so the systems behaviour does not satisfy this property.

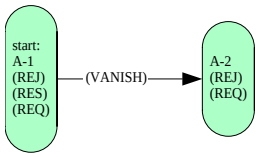

Figure 8: Component Graph

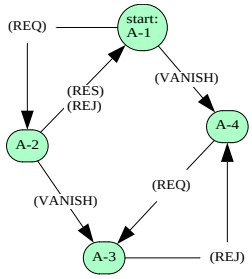

Figure 9: Minimal Automaton (with VANISH)

These simplicity investigations, which are supported by the tool, detect the error in the specification. In (Ochsenschläger 1992; 1994a) a necessary condition for simplicity is given. It is based on so called deadlock languages and shows non-simplicity of our *REQ-RES-REJ*-homomorphism (Ochsenschläger *et al.* 1998).

To handle the well known state space explosion problem *a compositional method* (Ochsenschläger 1996) is implemented in the sh-verification tool. This approach can also be used iteratively and provides a basis for induction proofs in case of systems with several identical components (Ochsenschläger 1996). Using our compositional method a connection establishment and release protocol has been verified by investigating automata with about 100 states instead of 100000 states.

Temporal Logic

Our verification approach can also be combined with *temporal logic* (Ochsenschläger *et al.* 1998). In terms of temporal logic, the automaton of Figure 6 approximately satisfies (Ochsenschläger *et al.* 1998) the formula $\mathcal{G}(\mathcal{F}(RES))$ ($\mathcal{G}$: always-operator, $\mathcal{F}$: eventually-operator; thus $\mathcal{G}(\mathcal{F}(RES))$ means "infinitely often result"), but the system in Figure 3 does not. This is indeed the case because the abstracting homomorphism is not simple. Using an appropriate type of *model checking*, approximate satisfaction of temporal logic formulae can be checked by the sh-verification tool.

Our experience in practical examples shows that the combination of computing a minimal automaton of an LTS and model checking on this abstraction is significantly faster than direct model checking on the LTS.

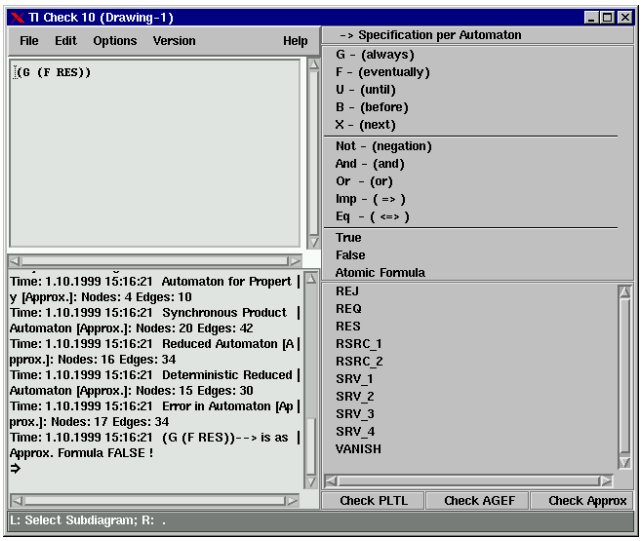

Figure 10: Temporal Logic Formula Editor

Applications

Practical experiences have been gained with large specifications:

- ISDN and XTP protocols (Klug 1992; Schremmer 1992; Ochsenschläger Prinoth 1993)
- Smartcard systems (Nebel 1994; Ochsenschläger 1994b)
- Service interactions in intelligent telecommunication systems (Capellmann *et al.* 1996b; 1996a).
- The tool has also been applied to the analysis of cryptographic protocols (Basak 1999; Rudolph 1998). In this context an application oriented user-interface has been developed for input of cryptographic formulae and presentation of results in this syntax.
- Currently our interest is focused on the verification of binding cooperations including electronic money and contract systems. Recently some examples in that context have been investigated with our tool (Fox 1998; Roßmann 1998).

Technical Requirements

The sh-verification tool is implemented in Allegro Common Lisp. An interpreter-based version of the software is freely available (currently for Solaris, Linux and Windows NT) for non commercial purposes (http://sit.gmd.de/META/projects.html). For investigation of large systems a compiler-based version of the tool is needed. For more information please contact the authors.

Conclusions

We have presented the basic functionality of the sh-verification tool in this article. The tool is equipped with the main features necessary to verify specifications of cooperating systems of industrial size. It supports a verification method based on formal languages (Ochsenschläger, Repp, Rieke 1999).

There exists a variety of verification tools which can be found in the literature. Some are based on model checking, others use proof systems. We consider COSPAN (Kurshan 1994) to be closest to the sh-verification tool. COSPAN is automata based and contains a homomorphism based abstraction concept. Since the transition labels of automata in COSPAN are in a Boolean algebra notation, the abstraction homomorphisms are Boolean algebra homomorphisms which correspond to non-erasing alphabetic language homomorphisms on the automata level. The sh-verification tool, in addition, offers erasing homomorphisms as an abstraction concept. COSPAN also considers only linear satisfaction of properties. Thus fairness assumptions need to be made explicitly in this tool. A tool which uses the modal μ-calculus as a specification language for properties (Stirling 1989) is the concurrency workbench (Cleaveland, Parrow, Steffen 1993). In (Hartel *et al.* 1999) ten tools in this area including ours are compared.

We consider the main strength of our tool to be the combination of an inherent fairness assumption in the satisfaction relation, an abstraction technique compatible with approximate satisfaction, and a suitable compositional and partial order method for the construction of only a partial state space. The sh-verification tool's user interface and general handling has reached a level of maturity that enabled its successful application in the industrial area.

References

Basak, G. 1999. Sicherheitsanalyse von Authentifizierungsprotokollen – model checking mit dem SH-Verification tool. Diploma thesis, University of Frankfurt.

Burkhardt, H. J.; Ochsenschläger, P.; and Prinoth, R. 1989. Product nets — a formal description technique for cooperating systems. GMD-Studien 165, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.

Capellmann, C.; Demant, R.; Fatahi, F.; Galvez-Estrada, R.; Nitsche, U.; and Ochsenschläger, P. 1996a. Verification by behavior abstraction: A case study of service interaction detection in intelligent

telephone networks. In *Computer Aided Verification (CAV) '96*, volume 1102 of *Lecture Notes in Computer Science*, 466–469.

Capellmann, C.; Demant, R.; Galvez-Estrada, R.; Nitsche, U.; and Ochsenschläger, P. 1996b. Case study: Service interaction detection by formal verification under behaviour abstraction. In Margaria, T., ed., *Proceedings of International Workshop on Advanced Intelligent Networks'96*, 71–90.

Cleaveland, R.; Parrow, J.; and Steffen, B. 1993. The concurrency workbench: A semantics-based tool for the verification of finite-state systems. In *TOPLAS 15*, 36–72.

Fox, S. 1998. Sezifikation und Verifikation eines Separation of Duty-Szenarios als verbindliche Telekooperation im Sinne des Gleichgewichtsmodells. GMD Research Series 21, GMD – Forschungszentrum Informationstechnik, Darmstadt.

Hartel, P.; Butler, M.; Currie, A.; Henderson, P.; Leuschel, M.; Martin, A.; Smith, A.; Ultes-Nitsche, U.; and Walters, B. 1999. Questions and answers about ten formal methods. In *Proc. 4th Int. Workshop on Formal Methods for Industrial Critical Systems*, volume II, 179–203. Pisa, Italy: ERCIM.

Klug, W. 1992. OSI-Vermittlungsdienst und sein Verhältnis zum ISDN-D-Kanalprotokoll. Spezifikation und Analyse mit Produktnetzen. Arbeitspapiere der GMD 676, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.

Kurshan, R. P. 1994. *Computer-Aided Verification of Coordinating Processes*. Princeton, New Jersey: Princeton University Press, first edition.

Nebel, M. 1994. Ein Produktnetz zur Verifikation von Smartcard-Anwendungen in der STARCOS-Umgebung. GMD-Studien 234, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.

Ochsenschläger, P., and Prinoth, R. 1993. Formale Spezifikation und dynamische Analyse verteilter Systeme mit Produktnetzen. In *Informatik aktuell Kommunikation in verteilten Systemen*, 456–470. München: Springer Verlag.

Ochsenschläger, P., and Prinoth, R. 1995. *Modellierung verteilter Systeme – Konzeption, Formale Spezifikation und Verifikation mit Produktnetzen*. Wiesbaden: Vieweg.

Ochsenschläger, P.; Repp, J.; Rieke, R.; and Nitsche, U. 1998. The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems. *Formal Aspects of Computing* 10:381–404.

Ochsenschläger, P.; Repp, J.; and Rieke, R. 1999. Verification of Cooperating Systems – An Approach Based on Formal Languages. Submitted to *FLAIRS-2000* Special Track on Validation, Verification & System Certification.

Ochsenschläger, P. 1992. Verifikation kooperierender Systeme mittels schlichter Homomorphismen. Arbeitspapiere der GMD 688, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.

Ochsenschläger, P. 1994a. Verification of cooperating systems by simple homomorphisms using the product net machine. In Desel, J.; Oberweis, A.; and Reisig, W., eds., *Workshop: Algorithmen und Werkzeuge für Petrinetze*, 48–53. Humboldt Universität Berlin.

Ochsenschläger, P. 1994b. Verifikation von Smartcard-Anwendungen mit Produktnetzen. In Struif, B., ed., *Tagungsband des 4. GMD-SmartCard Workshops*. GMD Darmstadt.

Ochsenschläger, P. 1996. Kooperationsprodukte formaler Sprachen und schlichte Homomorphismen. Arbeitspapiere der GMD 1029, GMD – Forschungszentrum Informationstechnik, Darmstadt.

Roßmann, J. 1998. Formale Analyse der Business-Phase des First Virtual Internet Payment Systems basierend auf Annahmen des Gleichgewichtsmodells. Diploma thesis, University of Frankfurt.

Rudolph, C. 1998. Analyse krypotgraphischer Protokolle mittels Produktnetzen basierend auf Modellannahmen der BAN-Logik. GMD Research Series 13/1998, GMD – Forschungszentrum Informationstechnik GmbH.

Schremmer, S. 1992. ISDN-D-Kanalprotokoll der Schicht 3. Spezifikation und Analyse mit Produktnetzen. Arbeitspapiere der GMD 640, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt.

Stirling, C. 1989. An introduction to modal and temporal logics for CCS. In Yonezawa, A., and Ito, T., eds., *Concurrency: Theory, Language, and Architecture*, volume 391 of *Lecture Notes in Computer Science*. Springer Verlag.