Verification of Cooperating Systems An Approach Based on Formal Languages

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

Behaviour of systems is described by formal languages: the sets of all sequences of actions. Regarding abstraction, alphabetic language homomorphisms are used to compute abstract behaviours. To avoid loss of important information when moving to the abstract level, abstracting homomorphisms have to satisfy a certain property called simplicity on the concrete (i.e. not abstracted) behaviour. To be suitable for verification of so called cooperating systems, a modified type of satisfaction relation for system properties (approximate satisfaction) is considered. The well known state space explosion problem is tackled by a compositional method formalised by so called cooperation products of formal languages.

Keywords: Simple language homomorphisms; Approximate satisfaction of safety and liveness properties; Cooperation products of formal languages

Introduction

By cooperating systems we mean distributed systems which are characterized by freedom of decision and loose coupling of their components. This causes a high degree of nondeterminism which is handled by our methods. Typical examples of cooperating systems are telephone systems, communication protocols, smartcard systems, electronic money, contract systems, etc.

In that context verification is the proof that system components work together in a desired manner. So the dynamic behaviour of the system has to be investigated. One usual approach is to start with a formal specification of the dynamic behaviour of the system which is represented by a *labelled transition system* LTS [1], and then to prove properties of such an LTS (Kurshan 1994; Baeten Weijland 1990). But for real life applications the corresponding LTS are often too complex to apply this naive approach.

[1]labelled directed graph with an initial node

In contrast to the immense number of transitions of such an LTS usually only a few characteristic actions of the system are of interest with respect to verification. So it is evident to define abstractions with respect to the actions of interest and to compute a representation of such an abstract behaviour, which usually is much smaller than the LTS of the specification. For such a small representation dynamic properties can be proven more efficiently. Now, under certain conditions, properties of the system specification can be deduced from properties of the abstract behaviour. For such an approach the following questions have to be answered:

Question 1 What does it formally mean, that a system satisfies a property (especially in the context of cooperating systems)?

Question 2 How can we formally define abstractions?

Question 3 For what kind of abstractions is there a sufficiently strong relation between system properties and properties of the abstract behaviour?

Question 4 How can we compute a representation of the abstract behaviour efficiently?

For finite state systems these questions will be answered in the following chapters. The presented verification method is supported by the sh-verification tool (Ochsenschläger, Repp, Rieke 1999).

Approximately Satisfied Properties

To formalise behaviour abstraction we use terms of *formal language theory*. An LTS is completely determined by the set of its paths starting at the initial state. This set is a formal language, called the *local language* of the LTS (Eilenberg 1974). Its letters are the transitions (state, transition label, successor state) of the LTS. Σ denotes the set of all transitions of the LTS. Consequently, there is a one to one correspondence [2]

[2]Even in case of nondeterministic LTS the triple elements of Σ guarantee the one to one correspondence.

between the LTS and its local language $L \subset \Sigma^*$, where Σ^* is the set of all sequences of elements of Σ including the empty sequence ϵ. Now behaviour abstraction can be formalized by *language homomorphisms*, more precisely by alphabetic language homomorphisms $h : \Sigma^* \to \Sigma'^*$ (**answer to question 2**). By these homomorphisms certain transitions are ignored and others are renamed, which may have the effect, that different transitions are identified with one another. A mapping $h : \Sigma^* \to \Sigma'^*$ is called a language homomorphism if $h(\epsilon) = \epsilon$ and $h(yz) = h(y)h(z)$ for each $y, z \in \Sigma^*$. It is called alphabetic, if $h(\Sigma) \subset \Sigma' \cup \{\epsilon\}$.

An automaton representation (*minimal automaton*) (Eilenberg 1974) for the abstract behaviour of a specification (homomorphic image of the LTS's local language) can be computed by the sh-verification tool.

The usual concept of *linear satisfaction of properties* (each infinite run of the system satisfies the property) is not suitable in this context because no fairness constraints are considered. We put a very abstract notion of fairness into the satisfaction relation for properties, which considers that independent of a finitely long computation of a system certain desired events may occure eventually . To formalise such "possibility properties", which are of interest when considering what we call cooperating systems, the notion of *approximate satisfaction* of properties is defined in (Nitsche Ochsenschläger 1996) (**answer to question 1**):

Definition. *An LTS approximately satisfies a property if and only if each finite path can be continued to an infinite path, which satisfies the property.*

As it is well known, system properties are divided into two types: *safety* (what happens is not wrong) and *liveness* properties (eventually something desired happens) (Alpern Schneider 1985). For safety properties linear satisfaction and approximate satisfaction are equivalent (Nitsche Ochsenschläger 1996). Examples are given in (Ochsenschläger, Repp, Rieke 1999). The notion of approximate satisfaction is related to machine-closure as defined in (Apt, Frances, Katz 1988).

Simple Homomorphisms as an Abstraction Concept

It is now the question of main interest, whether, by investigating an abstract behaviour, we may verify the correctness of the underlying concrete behaviour. Generally under abstraction the problem occurs, that an incorrect subbehaviour can be hidden by a correct one. We will answer this question positively, requiring a restriction of the permitted abstraction techniques. To deduce approximately satisfied properties of a specification from properties of its abstract behaviour an additional property of abstractions is required: called *simplicity of homomorphisms* on a specification (Ochsenschläger 1992). Simplicity of homomorphisms on specifications is a very technical condition concerning the possible continuations of finite behaviours.

Concerning abstractions $h : \Sigma^* \to \Sigma'^*$ the crucial point are the liveness properties of a language $L \subset \Sigma^*$. To define simplicity formally we need $w^{-1}(L) = \{y \in \Sigma^* | wy \in L\}$, the *set of continuations* of a word w in a language L (Eilenberg 1974). These continuations in some sense "represent" the liveness properties of L. Generally $h(x^{-1}(L))$ is a (proper) subset of $h(x)^{-1}(h(L))$, but we want to have that $h(x^{-1}(L))$ "eventually" equals $h(x)^{-1}(h(L))$.

Definition. *A homomorphism h is called simple on L, if for each $x \in L$ there exists $w \in h(x)^{-1}(h(L))$ such that $w^{-1}(h(x^{-1}(L))) = (h(x)w)^{-1}(h(L))$.*

For regular languages simplicity is decidable, but by a very complex algorithm. In (Ochsenschläger 1992; 1994) a sufficient condition based on the strongly connected components of an LTS has been proven.

Theorem. *Let L be a language recognized by a finite automaton $\mathcal{A}$ and let h be a homomorphism on L. If for each $x \in L$ there exists $y \in x^{-1}(L)$ leading to a dead component [3] of $\mathcal{A}$, such that each $z \in L$ with $h(z) = h(xy)$ leads to the same dead component, then h is simple on L.*

This condition is satisfied for example, if each dead component contains a label a of an edge with $h(a) \neq \epsilon$, such that no edge exists outside of this component, whose label has the same image $h(a)$. If $\mathcal{A}$ is strongly connected, then each homomorphism is simple on L. In (Ochsenschläger 1992; 1994) also a necessary condition for simplicity has been proven.

The following theorem (Nitsche Ochsenschläger 1996) shows that approximate satisfaction of properties and simplicity of homomorphisms exactly fit together for verifying cooperating systems (**answer to question 3**):

Theorem. *Simple homomorphisms define exactly the class of such abstractions, for which holds that each property is approximately satisfied by the abstract behaviour if and only if the "corresponding" property is approximately satisfied by the concrete behaviour of the system.*

Formally, the "corresponding" property is expressed by the inverse image of the abstract property with respect to the homomorphism.

Our verification method, which is based on the very general notions of approximate satisfaction of properties and simple language homomorphisms, does not de-

[3] a component without outgoing edges

pend on a specific formal specification method. It can be applied to all specification techniques with an LTS semantics.

A Compositional Approach to Avoid State Space Explosion

Simple homomorphisms establish the coarsest, i.e. most abstract notion of system equivalence with respect to a given (abstract) requirement specification. In some sense this equivalence is weaker than failure equivalence (Baeten Weijland 1990), which is too strong in the context of cooperationg systems, as it is shown in (Ochsenschläger 1994). What still remains open is the question of how to construct an abstract behaviour to a given specification without an exhaustive construction of its state space.

To handle the well known state space explosion problem, *a compositional method* has been developed (Ochsenschläger 1996) and implemented in the sh-verification tool. In case of well structured specifications, by applying a divide and conquer strategy this method allows to compute a representation of the abstract behaviour and to check simplicity of homomorphisms efficiently without having to compute the complex LTS of the complete specification (**answer to question 4**). This compositional method is combined with a *partial order method* based on *partially commutative languages* (Ochsenschläger 1997).

The main goal of our compositional method is to compute minimal automata of homomorphic images and to check simplicity of homomorphisms efficiently even in case of complex specifications. The fundamental idea is to embed each component of a structured system (X and Y in Figure 4) in a "simplified environment" (Y' and X' in Figure 1), which shows at the interface an "equivalent behaviour" compared to the rest of the system (shaded areas in Figure 1). This can be checked using special homomorphisms, called *boundary homomorphisms*. The complexity of this check is reduced by our partial order method.

For each of these smaller systems minimal automata related to corresponding homomorphisms (which have to be finer than the boundary homomorphisms) are computed (Figure 2) and are composed (Figure 3) to obtain the desired automaton (Figure 4).

Composition of system behaviour is formalised by the notion of *cooperation products of formal languages*, a restricted kind of shuffle product (Eilenberg 1974). Simplicity of homomorphisms on cooperation products is guaranteed by a particular property of the boundary homomorphisms, which is called *cooperativity*. For more details we refer to (Ochsenschläger *et al.* 1998).

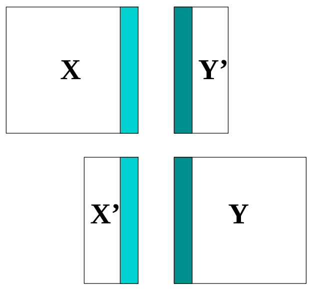

Figure 1:

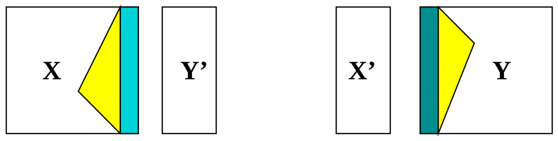

Figure 2:

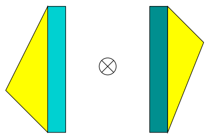

Figure 3:

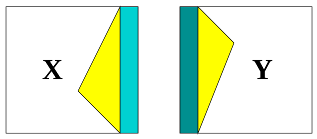

Figure 4:

Cooperation Products

Cooperation products of formal languages describe behaviour of composed systems in terms of their components behaviour and of the communication systems behaviour. To express specific properties of a communication system cooperation products are more flexible than rendezvous (Baeten Weijland 1990), as they are used in algebraic specification techniques.

As an example let us consider a system that consists of a client and a server as its main components. The client sends requests to the server, expecting the server to produce particular results. Nevertheless, for some reasons, the server may not always respond a request by sending a result, but may, as well, reject a request. The clients and servers behaviour is given in Figure 5 and Figure 6. The initial states are shaded.

As the clients and servers LTS is deterministic, it is sufficient to consider the transition labels instead of the triples (state,transition label,

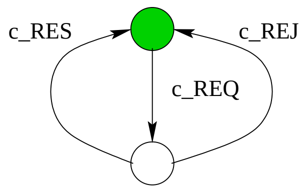

Figure 5: Client Behaviour

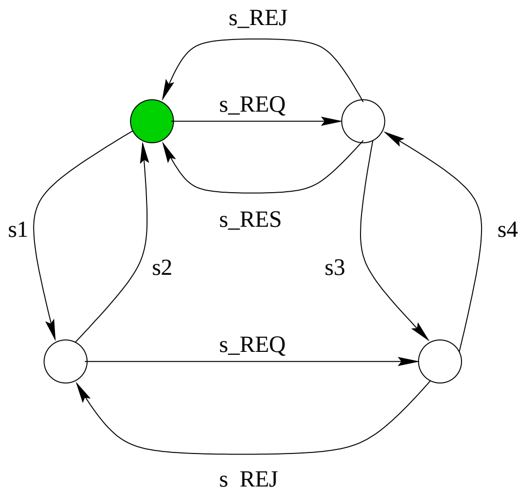

Figure 6: Server Behaviour

succ. state) as the corresponding actions. So the clients behaviour is described by a formal language $F \subset \Phi^*$ with $\Phi = \{c\_REQ, c\_RES, c\_REJ\}$ and the servers behaviour by $G \subset \Gamma^*$ with $\Gamma = \{s\_REQ, s\_RES, s\_REJ, s1, s2, s3, s4\}$. F and G are the languages accepted by the automata of Figure 5 and Figure 6, where all states are accepting states.

By usual interleaving semantics (Baeten Weijland 1990) the global system behaviour is described by a language $L \subset (\Phi \cup \Gamma)^*$ with $\pi_\Phi(L) \subset F$ and $\pi_\Gamma(L) \subset G$, where $\pi_\Phi : (\Phi \cup \Gamma)^* \longrightarrow \Phi^*$ and $\pi_\Gamma : (\Phi \cup \Gamma)^* \longrightarrow \Gamma^*$ are homomorphisms defined by $\pi_\Phi(x) = x$ for $x \in \Phi, \pi_\Phi(x) = \epsilon$ for $x \in \Gamma, \pi_\Gamma(x) = \epsilon$ for $x \in \Phi$ and $\pi_\Gamma(x) = x$ for $x \in \Gamma$. Notice that $\Phi \cap \Gamma = \emptyset$, which is generally assumed to define the cooperation product of F and G.

The global behaviour L not only depends on F and G but also on the kind of communication. For example let us consider a common buffer of capacity 1. The behaviour of this communication system is given in Figure 7.

The label a_M means accepting a message M and d_M means delivering message M. This automaton defines a language $C \subset \Sigma^*$ with $\Sigma = \{a\_REQ, d\_REQ, a\_RES, d\_RES, a\_REJ, d\_REJ\}$.
To formally express $F's$ and $G's$ "effect" on the com-

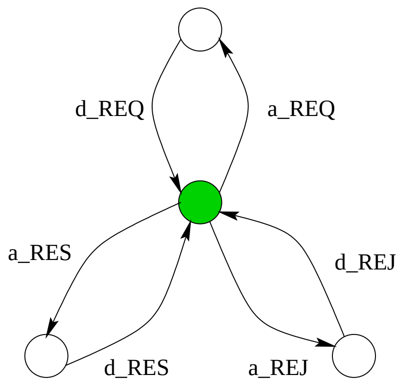

Figure 7: Communication System

munication system we use alphabetic homomorphisms $\phi : \Phi^* \longrightarrow \Sigma^*$ and $\gamma : \Gamma^* \longrightarrow \Sigma^*$ defined by

$$\phi(c\_REQ) = a\_REQ, \phi(c\_RES) = d\_RES,$$
$$\phi(c\_REJ) = d\_REJ \text{ and}$$
$$\gamma(s\_REQ) = d\_REQ, \gamma(s\_RES) = a\_RES,$$
$$\gamma(s\_REJ) = a\_REJ,$$
$$\gamma(s1) = \gamma(s2) = \gamma(s3) = \gamma(s4) = \epsilon.$$

With these homomorphisms L has to satisfy the condition $[\phi, \gamma](L) \subset C$, where $[\phi, \gamma] : (\Phi \cup \Gamma)^* \longrightarrow \Sigma^*$ is the homomorphism defined by $[\phi, \gamma](x) = \phi(x)$ for $x \in \Phi$ and $[\phi, \gamma](x) = \gamma(x)$ for $x \in \Gamma$.

Now the tree conditions $\pi_\Phi(L) \subset F, \pi_\Gamma(L) \subset G$ and $[\phi, \gamma](L) \subset C$ together completely define the global behaviour: $L = \pi_\Phi^{-1}(F) \cap \pi_\Gamma^{-1}(G) \cap [\phi, \gamma]^{-1}(C)$.

Definition. $[F, G]_c = \pi_\Phi^{-1}(F) \cap \pi_\Gamma^{-1}(G) \cap [\phi, \gamma]^{-1}(C)$ *is called the cooperation product of* F *and* G *with respect to* $c = (\Phi, \Gamma, \Sigma, \phi, \gamma, C)$.
In (Ochsenschläger *et al.* 1998) this example is discussed in more detail.

Using cooperation products the notion of *cooperativity* is formulated in (Ochsenschläger 1996), which gives a sufficient condition for simplicity of homomorphisms on cooperation products. The automata pendant to cooperation products are *asynchronous product automata* (APA), as they are defined in (Ochsenschläger 1996; Ochsenschläger *et al.* 1998). It is a very general class of communicating automata. APA can be regarded as families of automata (*elementary automata*), whose sets of states are cartesian products and whose elementary automata are "glued together" by common components of these products. In (Ochsenschläger *et al.* 1998) our compositional method is formulated in terms of APA.

Conclusions

We have presented a verification method which comprises a satisfaction relation with an inherent fairness assumption and an abstraction concept adequate for the particular, practically useful satisfaction relation. Our approach, which is based on the very general notions of approximate satisfaction of properties and simple language homomorphisms, does not depend on a specific formal specification method. It can be applied to all those specification techniques having an LTS-semantics.

Summarizing, using simple abstractions and approximate satisfaction verification can be done in two ways:

- System properties are explicitly given by temporal logic formulae or Büchi-automata. They can be checked on the abstract behaviour (under a simple homomorphism).
- Specifications of different abstraction levels are compared by corresponding simple homomorphisms. In that case system properties are given implicitly.

The presented approach can be compared with automata based methods as described in (Alur Henzinger 1995) or (Kurshan 1994) as well as with the concurrency workbench (Cleaveland, Parrow, Steffen 1993), which uses the modal μ-calculus. We consider the main strength of our method to be the combination of an inherent fairness assumption in the satisfaction relation, a very flexible abstraction technique compatible with approximate satisfaction, and a suitable compositional and partial order method for the construction of only a partial state space.

Though our method has been developed for finite state systems we think that it also can be applied to infinite state systems. For that purpose it has to be combined with a theorem prover.

Our recent interest is focused on the verification of binding cooperations like electronic money and contract systems (Grimm Ochsenschläger 1999). In that context the use of cooperation products leads to formal requirement specifications for cryptographic protocols.

References

Alpern, B., and Schneider, F. B. 1985. Defining liveness. *Information Processing Letters* 21(4):181–185.

Alur, R., and Henzinger, T. A. 1995. Local liveness for compositional modeling of fair reactive systems. In Wolper, P., ed., *Computer Aided Verification (CAV) '95*, volume 939 of *Lecture Notes in Computer Science*, 166–179. Springer.

Apt, K.; Frances, N.; and Katz, S. 1988. Appraising fairness in languages for distributed programming. *Distributed Computing* 2:226–241.

Baeten, J., and Weijland, W. 1990. *Process Algebra.* Cambridge University Press.

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.

Eilenberg, S. 1974. *Automata, Languages and Machines*, volume A. New York: Academic Press.

Grimm, R., and Ochsenschläger, P. 1999. Verbindliche Telekooperation. Ein Modell für Electonic Commerce auf Basis formaler Sprachen. In Röhm, Fox, G. S., ed., *Sicherheit und Electronic Commerce. DuD-Fachbeiträge*, 1–14. Braunschweig, Wiesbaden: Vieweg.

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

Nitsche, U., and Ochsenschläger, P. 1996. Approximately satisfied properties of systems and simple language homomorphisms. *Information Processing Letters* 60:201–206.

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. The SH-Verification Tool. 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. 1994. 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. 1996. Kooperationsprodukte formaler Sprachen und schlichte Homomorphismen. Arbeitspapiere der GMD 1029, GMD – Forschungszentrum Informationstechnik, Darmstadt.

Ochsenschläger, P. 1997. Schlichte Homomorphismen auf präfixstabilen partiell kommutativen Sprachen. Arbeitspapiere der GMD 1106, GMD – Forschungszentrum Informationstechnik, Darmstadt.