This is an Accepted Manuscript of an article published by Taylor & Francis in Journal of Experimental and Theoretical Artificial Intelligence, Volume 12, Issue 4, 2000, available online: http://dx.doi.org/10.1080/095281300454829.

Abstraction and composition a verification method for cooperating systems

Peter Ochsenschläger, Jürgen Repp, 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: Abstraction; Simple language homomorphisms; Approximate satisfaction of safety and liveness properties; Cooperation products of formal languages; Cooperating systems; Finite state systems; Formal specification; Verification

1 Introduction

The complexity of verification in cooperating systems grows with the system size, determined, in particular by the number of components involved and by the number of states and transitions in each component. This growth is so rapid, that reasoning in sizeable systems becomes intractable. However given a system S and a query Q demanding a property P of its behaviour, if it is possible to find an abstraction of S informative enough to provide a correct answer to Q, and small enough to fit into the range of tractable computation, the answer to Q (satisfaction of P) could be produced efficiently. This paper presents a way of implementing this idea.

Conclusions suggested by an abstraction of a system may, of course, differ from those derived from the entire system. Nevertheless, it is proven that if this abstraction is chosen properly, then even though it may be small it provides sufficient information to derive a correct conclusion.

Furthermore we present a way to make use of the typical component structure of cooperating systems to further reduce complexity of the state space to be explored in a semiautomatic manner. 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 the required restrictions to the abstractions efficiently without having to compute the complex dynamic behaviour of the complete system.

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 method. 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] that is usually computed from the specification by a tool. The next step is to prove properties of such an LTS (Kurshan 1994; Baeten and Weijland 1990). But in real life applications the corresponding LTS are often too complex to apply this naive approach.

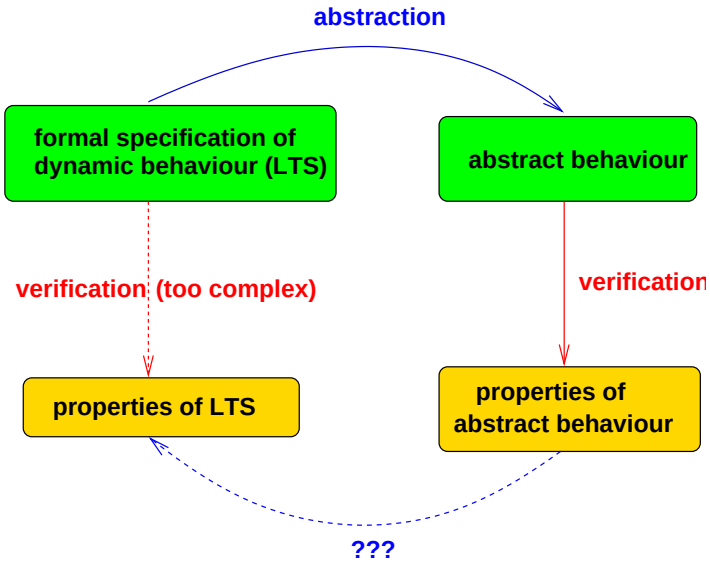


Figure 1: Approach

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

[1] labelled directed graph with an initial node

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. This paper is an extended version of (Ochsenschläger, Repp, and Rieke 2000b). The presented method is supported by the sh-verification tool (Ochsenschläger, Repp, and Rieke 2000a).

2 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] 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 occur 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 and Ochsenschläger 1996) (**answer to question 1**):

Definition 1 An LTS *approximately satisfies* a property if and only if each finite path can be continued to an infinite path, which satisfies the property.

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

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 and Schneider 1985). For safety properties linear satisfaction and approximate satisfaction are equivalent (Nitsche and Ochsenschläger 1996). The notion of approximate satisfaction is related to machine-closure as defined in (Apt, Frances, and Katz 1988).

3 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 to the permitted abstraction techniques. To deduce approximately satisfied properties of a specification from properties of its abstract behaviour an additional property of abstractions called simplicity of homomorphisms on a specification (Ochsenschläger 1992) is required. 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 2 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 $w^{-1}(h(x^{-1}(L))) = (h(x)w)^{-1}(h(L))$.

It is easy to show (Ochsenschläger 1992) that the composition of simple homomorphisms is simple:

Theorem 1 Let $h : \Sigma_1^* \to \Sigma_2^*$ and $g : \Sigma_2^* \to \Sigma_3^*$ be homomorphisms. If h is simple on $L \subset \Sigma_1^*$ and g is simple on $h(L) \subset \Sigma_2^*$ then $g \circ h$ is simple on L.

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

Theorem 2 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

[3] a component without outgoing edges

label has the same image $h(a)$. If $\mathcal{A}$ is strongly connected, then each homomorphism is simple on L. In (Ochsenschläger 1992; 1994a) also a necessary condition for simplicity has been proven.

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

Theorem 3 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 depend on a specific formal specification method. It can be applied to all specification techniques with an LTS semantics. Examples are given in (Ochsenschläger, Repp, and Rieke 2000a; Ochsenschläger *et al.* 1998) and in the next sections.

Using simple abstractions and approximate satisfaction verification can be done in two ways:

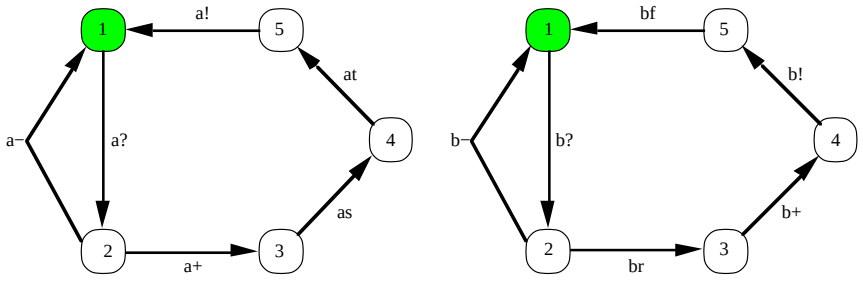
- System properties are explicitly given (by temporal logic formulae or Büchi-automata). They can be checked on the abstract behaviour (under a simple homomorphism).
- Specifications of different abstraction levels are compared by corresponding simple homomorphisms. In that case system properties are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

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 and Weijland 1990), as they are used in algebraic specification techniques.

As an example let us consider a system that consists of a server and a resource manager as its main components. The server F processes jobs and needs particular resources that are allocated by the resource manager G. Therefore F requests the needed resources from G and releases them after the execution of the job. The behaviour of F and G is given by the automata in figure 2. The initial states are shaded.

As the servers and resource managers LTS is deterministic, it is sufficient to consider the transition labels instead of the triples (state,transition label, succ. state) as the corresponding actions. So the servers behaviour is described by a formal language $F \subset \Phi^*$ with $\Phi = \{a?, a+, a-, as, at, a!\}$ and the resource managers behaviour by $G \subset \Gamma^*$ with $\Gamma = \{b?, b+, b-, br, bf, b!\}$. ?, +, − and ! are the actions related to the communication between F and G; ? means *request a resource*, + and − depict a positive or negative answer to this request, ! means *the resource*



(a) Servers Behaviour (b) Resource Managers Behaviour

Figure 2: Components behaviour

is no longer needed. From the context the direction of the message send/receive is obvious. All other actions are *internal* actions. *as* and *at* are the start and termination of a job. *br* and *bf* are the actions to allocate and free a resource. F and G are the languages accepted by the automata of figure 2, where all states are accepting states.

By usual interleaving semantics (Baeten and 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 a cooperation product of F and G.

The global behaviour L not only depends on F and G but also on the kind of communication. In this example the communication system C is a common buffer of capacity 1. It can store just one message out of $\{?, +, -, !\}$. The behaviour of this communication system is given in figure 3.

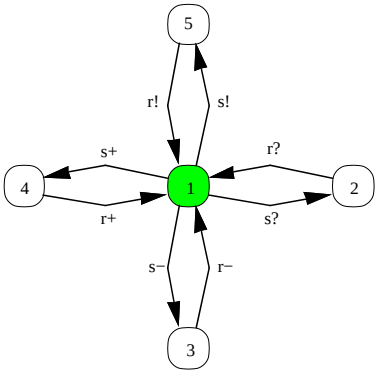


Figure 3: Communication systems behaviour

The label sM means accepting a message M which is sent by a cooperation partner and rM means delivering message M which is received by a cooperation partner. This automaton defines a language $C \subset \Sigma^*$ with $\Sigma = \{s?, s+, s-, s!, r?, r+, r-, r!\}$.

To formally express $F's$ and $G's$ "effect" on the communication system we use alphabetic homomorphisms $\phi : \Phi^* \longrightarrow \Sigma^*$ and $\gamma : \Gamma^* \longrightarrow \Sigma^*$ defined by
$\phi(a?) = s?$, $\phi(a+) = r+$, $\phi(a-) = r-$, $\phi(a!) = s!$, $\phi(as) = \phi(at) = \epsilon$ and
$\gamma(b?) = r?$, $\gamma(b+) = s+$, $\gamma(b-) = s-$, $\gamma(b!) = r!$, $\gamma(br) = \gamma(bf) = \epsilon$.

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 three conditions $\pi_\Phi(L) \subset F, \pi_\Gamma(L) \subset G$ and $[\phi, \gamma](L) \subset C$ together completely define the global behaviour (see figure 4): $L = \pi_\Phi^{-1}(F) \cap \pi_\Gamma^{-1}(G) \cap [\phi, \gamma]^{-1}(C)$ (here $\pi_\Phi^{-1}(F)$ denotes the inverse homomorphic image of $\pi_\Phi(F)$).

Definition 3 $[F, G]_c = \pi_\Phi^{-1}(F) \cap \pi_\Gamma^{-1}(G) \cap [\phi, \gamma]^{-1}(C)$ is called the *cooperation product* of F and G with respect to the *cooperation form* $c = (\Phi, \Gamma, \Sigma, \phi, \gamma, C)$.

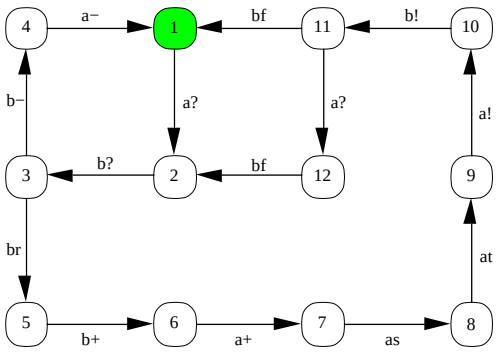


Figure 4: Global system behaviour

5 A compositional approach to avoid state space explosion

On account of theorem 3 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 and Weijland 1990), which is too strong in the context of cooperating systems, as it is shown in (Ochsenschläger 1994a). 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).

To consider this approach in more detail let us continue with the example of the previous section. A correct global system behaviour is given, if a required resource is allocated before each job and is released afterwards. So $v([F,G]_c)$ has to be considered, where $v : (\Phi \cup \Gamma)^* \to \Xi^*$ with $\Xi = \{as, at, br, bf\}$ is an appropriate homomorphism. To formally define v we use the following notation:

For an alphabet Φ, $I_\Phi : \Phi^* \to \Phi^*$ is the *identity homomorphism* that maps each word onto itself and $E_\Phi : \Phi^* \to \emptyset^*$ is the *epsilon homomorphism* that maps each word onto the empty word ϵ ($\emptyset^* = \{\epsilon\}$).

For two homomorphisms $f : \Phi^* \to \Phi'^*$ and $g : \Gamma^* \to \Gamma'^*$ with $\Phi \cap \Gamma = \emptyset$ the *direct sum* $[f,g] : (\Phi \cup \Gamma)^* \to (\Phi' \cup \Gamma')^*$ is defined by $[f,g](x) = f(x)$ for $x \in \Phi$ and $[f,g](x) = g(x)$ for $x \in \Gamma$.

With this notations v can be defined as: $v = [I_\Xi, E_{(\Phi\cup\Gamma)\setminus\Xi}]$. From figure 4 it is easy to construct the minimal automaton for $v([F,G]_c)$ as it is shown in figure 5. This automaton in fact shows the desired behaviour.

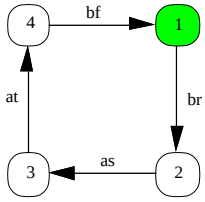


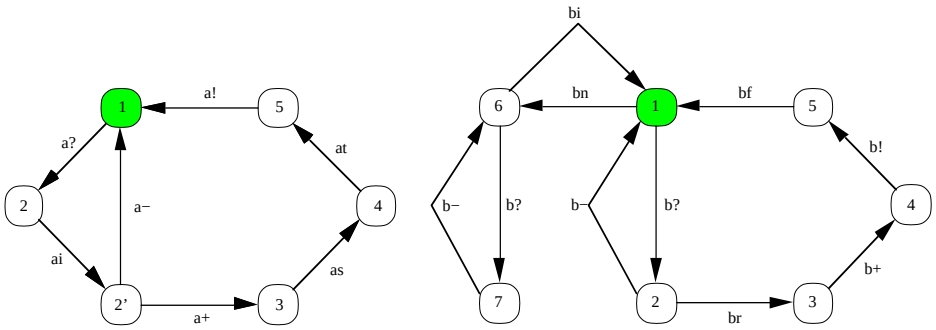
Figure 5: Minimal automaton of $v([F,G]_c)$

Looking at the number of states cooperation products of languages with relatively small minimal automata can become quite complex. This is well known as the state space explosion problem. A slight modification of our simple example shows the problem:

The server is extended to execute an additional internal action ai after requiring the resource and before accepting the answer of the resource manager. The resource manager is able to change with an internal action bn from the initial state into an exception state where it answers every resource request negative. From the exception state it can change back to the initial state with another internal action bi. Figure 6 shows the minimal automata of the modified server and resource manager.

The automata in figure 6 define languages $F1 \subset \Phi1^*$ and $G1 \subset \Gamma1^*$ with $\Phi1 = \Phi \cup \{ai\}$ and $\Gamma1 = \Gamma \cup \{bn, bi\}$. For the cooperation form $c1 = (\Phi1, \Gamma1, \Sigma, \phi1, \gamma1, C)$ let $\phi1 : \Phi1^* \to \Sigma^*$ and $\gamma1 : \Gamma1^* \to \Sigma^*$ be extended homomorphisms based on ϕ and γ with $\phi1(ai) = \gamma1(bn) = \gamma1(bi) = \epsilon$. Though there were only minor modifications to F and G $[F1, G1]_{c1}$ is quite more complex than $[F,G]_c$. The minimal automaton of $[F1, G1]_{c1}$ has 25 states and 44 transitions. Under certain premises the homomorphic image of a cooperation product can be determined without knowing the cooperation product itself.

To state an adequate theorem let $c = (\Phi, \Gamma, \Sigma, \phi, \gamma, C)$ be a cooperation form and $f : \Phi^* \to \Phi'^*$ respectively $g : \Gamma^* \to \Gamma'^*$ two homomorphisms with $\Phi' \cap \Gamma' = \emptyset$ and $f \prec \phi$ (f is finer than ϕ), $g \prec \gamma$ which means there are homomorphisms $\phi' : \Phi'^* \to \Sigma^*$ and $\gamma' : \Gamma'^* \to \Sigma^*$ with $\phi = \phi' \circ f$ and $\gamma = \gamma' \circ g$. Therefore $[\phi, \gamma] = [\phi', \gamma'] \circ [f, g]$. Let c' be the cooperation form defined by $c' = (\Phi', \Gamma', \Sigma, \phi', \gamma', C)$.



(a) $F1$ (b) $G1$

Figure 6: Minimal automata of $F1$ and $G1$

With these premises the following theorem holds:

Theorem 4 $[f,g]([F,G]_c) = [f(F), g(G)]_{c'}$

If an arbitrary homomorphic image of a cooperation product is to be computed using this theorem then usually the homomorphism has to be refined such that it can be represented in a form $[f,g]$ that meets the requirements of theorem 4. In the example this way does not reduce the complexity because only $f = I_\Phi$ and $g = I_\Gamma$ is possible.

To verify $[F1,G1]_{c1}$ the homomorphism v has to be extended by $v1(ai) = v1(bn) = v1(bi) = \epsilon$ to an homomorphism $v1 : (\Phi 1 \cup \Gamma 1)^* \to \Xi^*$. To apply theorem 4 homomorphisms $f1 : \Phi 1^* \to \Phi^*$ and $g1 : \Gamma 1^* \to \Gamma^*$ defined by $f1 = [I_\Phi, E_{\{ai\}}]$ and $g1 = [I_\Gamma, E_{\{bn,bi\}}]$ have to be considered.

Now:
$[f1,g1]([F1,G1]_{c1}) = [f1(F1), g1(G1)]_c = [F,G]_c$
Using $v1 = v \circ [f1,g1]$ it follows that:
$v1([F1,G1]_{c1}) = v([f1,g1]([F1,G1]_{c1})) = v([f1(F1), g1(G1)]_c) = v([F,G]_c)$

Because the minimal automaton of $[F,G]_c$ (figure 4) is strongly connected, v is simple on $[f1,g1]([F1,G1]_{c1})$. To prove simplicity of $v1$ on $[F1,G1]_{c1}$ using theorem 1 simplicity of $[f1,g1]$ on $[F1,G1]_{c1}$ must be proven.

6 Simplicity and cooperativity

To investigate simplicity of a direct sum $[f,g]$ of homomorphisms on a cooperation product $[F,G]_c$ the notion of cooperativity, that is stronger than simplicity, is needed. For an intuitive understanding of the technically difficult definition the reader is referred to the remark after theorem 5.

Let $L \subset (\Phi \cup \Gamma)^*$, $f : \Phi^* \to \Phi'^*$ and $\Phi \cap \Gamma = \Phi' \cap \Gamma = \emptyset$. Let $pre(H)$ represent the set of all prefixes of words of a formal language H and $max(H)$ the set of all words of H that are not prefix of another word of H ($max(H) \subset H \subset pre(H)$).

Definition 4 The language $[I_\Phi, E_\Gamma](L)$ is *cooperativ on L with respect to f*, if for every $x \in L$ a subset $H \subset [f, I_\Gamma](x)^{-1}([f, I_\Gamma](L))$ exists, with the following properties (1) -(3):

1. $H \neq \emptyset$, $H = pre(max(H))$ and $[I_{\Phi'}, E_\Gamma](H)$ are finite.
2. For $u \in max(H)$ holds $u^{-1}([f, I_\Gamma](x^{-1}(L))) = ([f, I_\Gamma](x)u)^{-1}([f, I_\Gamma](L))$.
3. For $u \in H \setminus max(H)$ holds
   (a) $u^{-1}(H) \cap \Gamma = ([f, I_\Gamma](x)u)^{-1}([f, I_\Gamma](L)) \cap \Gamma$ and
   (b) $u^{-1}(H) \cap \Phi' \neq \emptyset$ if $([f, I_\Gamma](x)u)^{-1}([f, I_\Gamma](L)) \cap \Phi' \neq \emptyset$

Because of $H \neq \emptyset$ and (2) cooperativity implies simplicity of $[f, I_\Gamma]$ on L.

For the next three theorems let $F \subset \Phi^*$, $G \subset \Gamma^*$, $\Phi' \cap \Gamma = \Phi' \cap \Gamma = \Phi \cap \Gamma' = \Phi' \cap \Gamma' = \emptyset$, $c = (\Phi, \Gamma, \Sigma, \phi, \gamma, C)$, $f : \Phi^* \to \Phi'^*$, $g : \Gamma^* \to \Gamma'^*$, $\phi' : \Phi'^* \to \Sigma^*$, $\gamma' : \Gamma'^* \to \Sigma^*$ with $\phi = \phi' \circ f$, $\gamma = \gamma' \circ g$, $c^\triangleright = (\Phi', \Gamma, \Sigma, \phi', \gamma, C)$ and $c^\triangleleft = (\Phi, \Gamma', \Sigma, \phi, \gamma', C)$. Proofs of these theorems are given in (Ochsenschläger 1996).

Theorem 5 If $[I_\Phi, E_{\Gamma'}]([F, g(G)]_{c^\triangleleft})$ is cooperative on $[F, g(G)]_{c^\triangleleft}$ with respect to f and $[E_{\Phi'}, I_\Gamma]([f(F), G]_{c^\triangleright}$ is cooperative on $[f(F), G]_{c^\triangleright}$ with respect to g, then $[f, g]$ is simple on $[F, G]_c$.

Theorem 5 reduces investigation of simplicity of $[f, g]$ on $[F, G]_c$ to investigation of cooperativity on two "smaller" systems (figure 7).

In addition to simplicity of the corresponding homomorphisms this cooperativity demands that $g(G)$ has in its cooperation with F the same "freedom of action" as with $f(F)$ and that $f(F)$ has in its cooperation with G the same "freedom of action" as with $g(G)$.

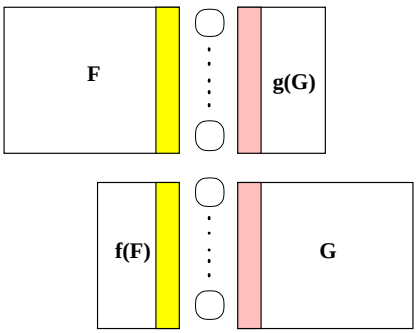


Figure 7:

In this constellation highest reduction of state space is achieved if $f(F)$ and $g(G)$ are as "small" as possible, that is $f = \phi$ and $g = \gamma$. Using theorem 5 directly to prove simplicity of $[f1, g1]$ on $[F1, G1]_{c1}$ does not effect an optimal reduction of complexity. The following theorems 6 and 7 allow such an optimal effect in an indirect manner.

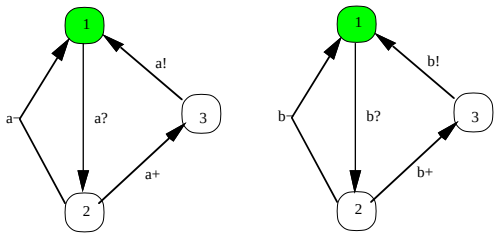
Theorem 6 If $[f, E_\Gamma]$ and $[E_\Phi, g]$ are simple on $[F, G]_c$, so is $[f, g]$.

Theorem 7 If $[f, g]$ is simple on $[F, G]_c$, so is $[I_\Phi, g]$. [4]

To prove simplicity of $[f1, g1]$ on $[F1, G1]_{c1}$ it is sufficient (theorem 6) to prove simplicity of $[f1, E_{\Gamma 1}]$ and $[E_{\Phi 1}, g1]$ on $[F1, G1]_{c1}$. Theorems 5 and 7 allow to investigate simplicity on "smaller" languages than on $[F1, G1]_{c1}$.

Let therefore $\Phi' = \{a?, a+, a-, a!\}$, $\Gamma' = \{b?, b+, b-, b!\}$,
$f' : \Phi 1^* \to \Phi'^*$ with $f' = [I_{\Phi'}, E_{\Phi 1 ? \Phi'}]$,
$g' : \Gamma 1^* \to \Gamma'^*$ with $g' = [I_{\Gamma'}, E_{\Gamma 1 ? \Gamma'}]$,
$\phi' : \Phi'^* \to \Sigma^*$ with $\phi'(a?) = s?$, $\phi'(a+) = r+$, $\phi'(a-) = r-$, $\phi'(a!) = s!$,
$\gamma' : \Gamma'^* \to \Sigma^*$ with $\gamma'(b?) = r?$, $\gamma'(b+) = s+$, $\gamma'(b-) = s-$, $\gamma'(b!) = r!$,
$c^\triangleleft = (\Phi', \Gamma 1, \Sigma, \phi', \gamma 1, C)$ and $c^\triangleright = (\Phi 1, \Gamma', \Sigma, \phi 1, \gamma', C)$.

The minimal automata of $f'(F1)$, $g'(G1)$, $[f'(F1), G1]_{c^\triangleleft}$ and $[F1, g'(G1)]_{c^\triangleright}$ are shown in figures 8, 9 and 10.



(a) $f'(F1)$ (b) $g'(G1)$

Figure 8: Minimal automata of $f'(F1)$ and $g'(G1)$

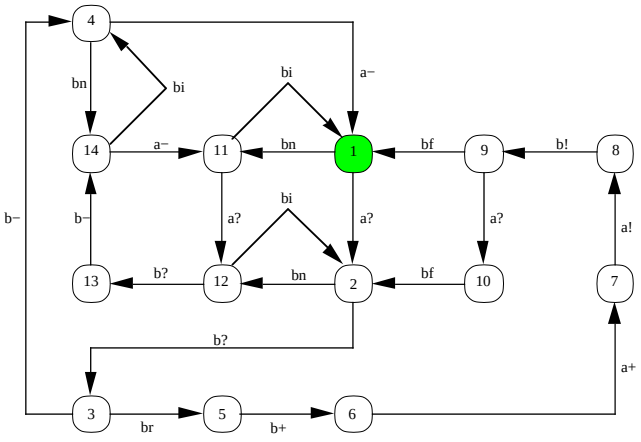


Figure 9: Minimal automaton of $[f'(F1), G1]_{c^\triangleleft}$

Using the minimal automata of $[f'(F1), G1]_{c^\triangleleft}$ and $[F1, g'(G1)]_{c^\triangleright}$ it can be proven, that:

[4] $g \prec \gamma$ is not required here.

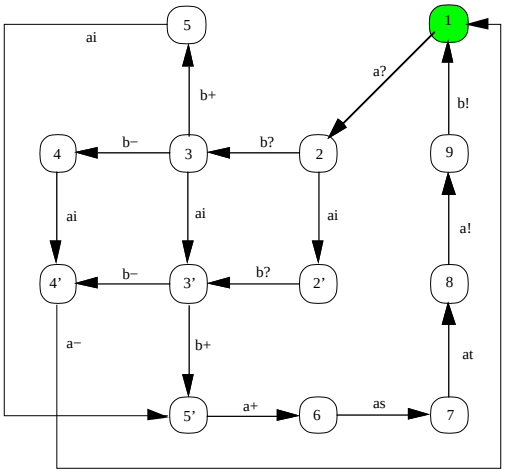


Figure 10: Minimal automaton of $[F1, g'(G1)]_{c^{\triangleright}}$

$[E_{\Phi'}, I_{\Gamma 1}]([f'(F1), G1]_{c^{\triangleleft}})$ is cooperative on $[f'(F1), G1]_{c^{\triangleleft}}$ with respect to g' and $[I_{\Phi 1}, E_{\Gamma'}]([F1, g'(G1)]_{c^{\triangleright}})$ is cooperative on $[F1, g'(G1)]_{c^{\triangleright}}$ with respect to f'.
Applying theorem 5 proves that $[f', g']$ is simple on $[F1, G1]_{c1}$.
Using theorem 7 $[I_{\Phi 1}, g']$ and $[f', I_{\Gamma 1}]$ are simple on $[F1, G1]_{c1}$.
It holds that $[f1, E_{\Gamma 1}] = [f1, E_{\Gamma'}] \circ [I_{\Phi 1}, g']$ and $[E_{\Phi 1}, g1] = [E_{\Phi'}, g1] \circ [f', I_{\Gamma 1}]$.
$[f1, E_{\Gamma'}]$ and $[E_{\Phi'}, g1]$ are simple on $[I_{\Phi 1}, g']([F1, G1]_{c1}) = [F1, g'(G1)]_{c^{\triangleright}}$ respectively $[f', I_{\Gamma 1}]([F1, G1]_{c1}) = [f'(F1), G1]_{c^{\triangleleft}}$, because the corresponding minimal automaton is strongly connected.
This proves simplicity of $[f1, E_{\Gamma 1}]$ and $[E_{\Phi 1}, g1]$ on $[F1, G1]_{c1}$.

In case of systems with several identical components (Ochsenschläger 1996) this approach can also be used iteratively and provides a basis for induction proofs. Using this compositional method a connection establishment and release protocol has been verified by investigating automata with about 100 states instead of 100000 states.

7 Applications

Practical experiences have been gained with large specifications using the sh-verification tool, which supports the presented method (Ochsenschläger, Repp, and Rieke 2000a).

- ISDN and XTP protocols (Ochsenschläger and 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) and a formal framework for binding cooperation has been developed (Grimm and Ochsenschläger 2000a; 2000b).

For a comparison of the sh-verification tool with other verification tools the reader is referred to (Hartel *et al.* 1999).

8 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.

The presented approach can be compared with automata based methods as described in (Alur and Henzinger 1995) or (Kurshan 1994) as well as with the concurrency workbench (Cleaveland, Parrow, and Steffen 1993), which uses the modal μ-calculus as a specification language for properties (Stirling 1989).

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 and Ochsenschläger 2000a; 2000b). 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.

Basak, G. 1999. Sicherheitsanalyse von Authentifizierungsprotokollen – model checking mit dem SH-Verification tool. GMD Research Series 19/1999, GMD – Forschungszentrum Informationstechnik GmbH.

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.

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

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

Grimm, R., and Ochsenschläger, P. 2000a. Binding Cooperation. A Formal Model for Electronic Commerce. GMD Report 96.

Grimm, R., and Ochsenschläger, P. 2000b. Elektronische Verträge und ihre verbindliche Aushandlung. Ein formales Modell für verbindliche Telekooperation. In *Informatik Forschung und Entwicklung, Band 15, Sonderheft E-Commerce (to appear)*. Springer Verlag.

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.

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.

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

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.; 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. 2000a. The SH-Verification Tool. In *Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000)*, 18–22. Orlando, FL, USA: AAAI Press.

Ochsenschläger, P.; Repp, J.; and Rieke, R. 2000b. Verification of Cooperating Systems – An Approach Based on Formal Languages. In *Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000)*, 346–350. Orlando, FL, USA: AAAI Press.

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.

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

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

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.

Papers of our research group are available at
http://www.sit.gmd.de/META/index.html.