# Abstraction Based Verification of a Parameterised Policy Controlled System

Peter Ochsenschläger and Roland Rieke [*]

Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, Germany
{ochsenschlaeger,rieke}@sit.fraunhofer.de

**Abstract.** Safety critical and business critical systems are usually controlled by policies with the objective to guarantee a variety of safety, liveness and security properties. Traditional model checking techniques allow a verification of the required behaviour only for systems with very few components. To be able to verify entire families of systems, independent of the exact number of replicated components, we developed an abstraction based approach to extend our current tool supported verification techniques to such families of systems that are usually parameterised by a number of replicated identical components. We demonstrate our technique by an exemplary verification of security and liveness properties of a simple parameterised collaboration scenario. Verification results for configurations with fixed numbers of components are used to choose an appropriate property preserving abstraction that provides the basis for an inductive proof that generalises the results for a family of systems with arbitrary settings of parameters.

**Key words:** Formal analysis of security and liveness properties, security modelling and simulation, security policies, parameterised models.

## 1 Introduction

In a typical policy controlled system, a set of policy rules, posing restrictions on the system's behaviour, is used to enforce the required security objectives, such as confidentiality, integrity and availability. For safety critical systems as well as for business critical systems or parts thereof, assuring the correctness - conformance to the intended purpose - is imperative. These systems must guarantee a variety of safety, liveness and security properties.

*The problem approached.* Traditional model checking techniques can be used to analyse such systems and to understand and verify how they behave subject to different policy constraints. However, because of well known state explosion problems, the usage of these techniques is limited to systems with very few components. In this paper we propose an extension of these techniques to a

[*] Part of the work presented in this paper was developed within the project SicAri being funded by the German Ministry of Education and Research.

particularly interesting class of systems called *parameterised systems*. A parameterised system describes a family of systems that are finite-state in nature but scalable. A formal specification of a parameterised system thus covers a family of systems, each member of which has a different number of replicated components. Instances of the family can be obtained by fixing the parameters. Extensions of model checking techniques are required that support verification of properties that are valid independently of given concrete parameters.

*Contributions.* To be able to verify entire families of critical systems, independent of the exact number of replicated components, we developed an *abstraction based approach* to extend our current tool supported verification techniques to such parameterised systems. Abstraction is a fundamental and widely-used verification technique. It can be used to reduce the verification of a property over a concrete system, to checking a related property over a simpler abstract system [1]. In this paper however we need an inductive proof on the construction of the behaviour of the parameterised system to show that it results in identical abstract system behaviour for any given parameter configuration. This allows the verification of parameterised systems by constructing abstract systems that can be model checked.

In the case of our abstraction based approach, the key problem is the choice of an appropriate abstraction that, (1) is *property preserving*, (2) results in identical abstract system behaviour for any given parameter configuration, and, (3) is sufficiently precise to express the required properties at the chosen abstraction level. To solve this problem, we

- compute the system behaviour and verify the required properties for some configurations with fixed numbers of components;
- we then use the results to choose an appropriate property preserving abstraction that results in identical abstract system behaviour for any given parameter configuration;
- based on this abstraction, we provide an inductive proof (by hand) that generalises the results for a family of systems with arbitrary settings of parameters.

In this paper we demonstrate our technique by an exemplary verification of security and liveness properties of a simple parameterised collaboration scenario.

The subsequent paper is structured as follows. In Sect. 2 we review some related work. Section 3 introduces a collaboration scenario that we will use throughout this paper to illustrate the usage of the proposed method for analysis of parameterised models. Section 4 describes the formal modelling technique, the abstraction based verification concept and the verification tool while Sect. 5 presents an exemplary verification of the collaboration scenario. Finally, the paper ends with conclusions and an outlook in Sect. 6.

## 2 Related Work

*Analysis of security policies.* The research in the field of security policies has gained increasing attention in the past few years. Many research papers appeared that investigated security policies on its own and abstracted from the systems needed to enforce these policies. These activities concentrated on the examination of specific properties of policies like consistency, freedom of conflicts, information flow implications and effects to system safety. This allows shifting the attention from specifics of computer system towards the analysis of properties that are inherent to the policy itself.

In the information flow analysis approach presented in [2] for the *SELinux* system, a labelled transition system (LTS) is generated from the policy specifications that models the information flow policy. Temporal logic formulas are used to specify the security goals. The *NuSMV* (`http://nusmv.irst.itc.it/`) model-checker verifies the security goals on this LTS.

A method to enforce rigorous automated network security management using a network access control policy is presented in [3]. This method is illustrated using examples based on enforcement strategy by distributed packet filtering and confidentiality/authenticity goals enforced by IPsec mechanisms.

In [4] a model-based approach focussing on the validation of network security policies and the interplay of threats and vulnerabilities and system's behaviour is proposed. This approach is based on Asynchronous Product Automata (APA) [5]. APA are also used as a basis of the work presented in this paper.

*Verification approaches for parameterised systems.* An extension to the *Mur$\varphi$* verifier to verify systems with replicated identical components through a new data type called RepetitiveID (with restricted usage) is presented in [6]. The verification is performed by explicit state enumeration in an abstract state space where states do not record the exact numbers of components. Mur$\varphi$ automatically checks the soundness of this abstraction and translates the system description to an abstract state graph for a system of a fixed size. During the verification of this system, Mur$\varphi$ uses a run-time check to determine if the result can be generalised for a family of systems. The soundness of the abstraction algorithm is guaranteed by the restrictions on the use of repetitiveIDs. These restrictions allow Mur$\varphi$ to decide which components are abstractable using the repetition constructors, enforce symmetry in the system, which enables the automatic construction of abstract states, and, enforce the repetitive property in the system, which enables the automatic construction of the abstract successors. A typical application area of this tool are cache coherence protocols. Many cache coherence protocols satisfy the above restrictions.

The aim of [7] is an abstraction method through symmetry which works also when using variables holding references to other processes which is not possible in Mur$\varphi$. An implementation of this approach for the *SPIN* model-checker (`http://spinroot.com/`) is described.

In [8] a methodology for constructing abstractions and refining them by analysing counter-examples is presented. The method combines abstraction, model-checking and deductive verification and in particular, allows to use the set of reachable states of the abstract system in a deductive proof even when the abstract model does not satisfy the specification and when it simulates the concrete system with respect to a weaker simulation notion than Milner's. The tool *InVeSt* supports this approach and makes use of *PVS* (`http://pvs.csl.sri.com/`) and *SMV* (`http://www.cs.cmu.edu/ modelcheck/smv.html`). This approach however does not consider liveness properties.

In [9] a technique for automatic verification of parameterised systems based on process algebra *CCS* [10] and the logic modal *mu-calculus* [11] is presented. This technique views processes as property transformers and is based on computing the limit of a sequence of mu-calculus formula generated by these transformers.

The above-mentioned approaches demonstrate, that finite state methods combined with deductive methods can be applied to analyse parameterised systems. The approaches differ in varying amounts of user intervention and their range of application. A survey of a number of approaches to combine model checking and theorem proving methods is given in [12].

Characteristic of our approach is the flexibility of abstractions defined by language homomorphisms and the consideration of liveness properties.

## 3 Collaboration Scenario

There are manifold uses and aspects of the terms *policy* in general and *security policy* specifically. In the context of this paper we use the concepts of the eXtensible Access Control Markup Language (XACML [13]) to express a security policy, but for readability we use a much simpler syntax.

We consider three *roles* (classes of collaboration partners with a uniform security policy) in this scenario namely *trustworthy clients* (TC), *observers* (OB) and a *manager* (M) representing the collaboration infrastructure. There is only one role player for the manager but an unspecified number of role players for the two types of clients. The set of *subjects* is defined by $subject = \{trustworthy\ client, observer, manager\}$. For our collaboration scenario we now assume that a group of trustworthy clients hold a session. The session can be in state *public* (*pub*) or *confidential* (*conf*). The set of possible session states is thus defined by $s\_state = \{pub, conf\}$. The initial session state is *pub*. We furthermore assume that the set of possible *actions* is defined by $action = \{join, leave, close, open\}$ and that the following policy rules govern the session.

$rule_1$ When the session is in state *pub*, then observers are permitted to *join*.
$rule_2$ Observers are permitted to *leave* at any time.
$rule_3$ When no observers participate in the session, then the manager can *close* the session (change state to *conf*).
$rule_4$ The manager can *open* the session (change state to *pub*) at any time.

To be able to decide whether observers are currently participating in a session, we furthermore use a counter $o\_count \in \mathbb{N}_0$ for the current count of observers in the session. The initial value of *o_count* is 0. We don't consider any actions of the trustworthy clients in the model because we consider this irrelevant for the security goals.

In XACML a policy is given by a set of rules and a rule-combining algorithm. Each rule is composed of a condition, an effect, and a target. The conditions (predicates on attributes of subject, resource, action) associated with a policy rule specify when the policy rule is applicable. If the condition returns *False*, the rule returns *NotApplicable*. If the condition returns *True*, the value of the effect element (*Permit* or *Deny*) is returned.

For better readability we use an abbreviated syntax in this paper and define the rules from our example now by

$$rule_x : subject \times s\_state \times action \times o\_count \rightarrow \{permit, deny, not\_applicable\}.$$

$$rule_1(s,a,z,c) = \begin{cases} permit \mid & s = observer \ \wedge \ a = join \ \wedge \ z = pub \\ not\_applicable \mid & else \end{cases}$$

$$rule_2(s,a,z,c) = \begin{cases} permit \mid & s = observer \ \wedge \ a = leave \\ not\_applicable \mid & else \end{cases}$$

$$rule_3(s,a,z,c) = \begin{cases} permit \mid & s = manager \ \wedge \ a = close \ \wedge \ c = 0 \\ not\_applicable \mid & else \end{cases}$$

$$rule_4(s,a,z,c) = \begin{cases} permit \mid & s = manager \ \wedge \ a = open \ \wedge \ z = conf \\ not\_applicable \mid & else \end{cases}$$

The rule-combining algorithm we use to derive the policy result from the given rules is the permit-overrides algorithm, if a single permit result is encountered, then the combined result is permit. So we define the policy for our example now by

$$policy : subject \times action \times s\_state \times o\_count \rightarrow \{permit, deny\}.$$

$$policy(s,a,z,c) = \begin{cases} permit \mid & rule_1(s,a,z,c) = permit \ \vee \\ & rule_2(s,a,z,c) = permit \ \vee \\ & rule_3(s,a,z,c) = permit \ \vee \\ & rule_4(s,a,z,c) = permit \\ deny \mid & else \end{cases}$$

Generally, security policies have to guarantee certain security properties of a system and moreover they must not prevent the system from working.

In our example we define the following security properties:

– the collaboration is in state $conf$ only if no observer is present (security), and
– always eventually state changes between *pub* and $conf$ are possible (liveness).

These properties are formally verified in Sect. 5.

## 4 Verification of System Properties

Our operational finite state model of the behaviour of the given collaboration scenario is based on *Asynchronous Product Automata (APA)*, a flexible operational specification concept for cooperating systems [5]. An APA consists of a family of so called *elementary automata* communicating by common components of their state (shared memory).

### 4.1 Formal Modelling Technique

We now introduce the formal modelling techniques used, and illustrate the usage by our collaboration example.

**Definition 1.** *An* Asynchronous Product Automaton *consists of*

- *a family of* state sets $Z_s, s \in \mathbb{S}$,
- *a family of* elementary automata $(\Phi_e, \Delta_e), e \in \mathbb{E}$ *and*
- *a* neighbourhood relation $N : \mathbb{E} \to \mathcal{P}(\mathbb{S})$

$\mathbb{S}$ *and* $\mathbb{E}$ *are index sets with the names of state components and of elementary automata and* $\mathcal{P}(\mathbb{S})$ *is the power set of* $\mathbb{S}$.

*For each elementary automaton* $(\Phi_e, \Delta_e)$ *with* Alphabet $\Phi_e$*, its* state transition relation *is* $\Delta_e \subseteq \bigtimes_{s \in N(e)}(Z_s) \times \Phi_e \times \bigtimes_{s \in N(e)}(Z_s)$. *For each element of* $\Phi_e$ *the state transition relation* $\Delta_e$ *defines state transitions that change only the state components in* $N(e)$. *An APA's (global)* states *are elements of* $\bigtimes_{s \in \mathbb{S}}(Z_s)$. *To avoid pathological cases it is generally assumed that* $\mathbb{S} = \bigcup_{e \in \mathbb{E}}(N(e))$ *and* $N(e) \neq \emptyset$ *for all* $e \in \mathbb{E}$. *Each APA has one* initial state $q_0 = (q_{0s})_{s \in \mathbb{S}} \in \bigtimes_{s \in \mathbb{S}}(Z_s)$. *In total, an APA* $\mathbb{A}$ *is defined by*

$$\mathbb{A} = ((Z_s)_{s \in \mathbb{S}}, (\Phi_e, \Delta_e)_{e \in \mathbb{E}}, N, s_0)$$

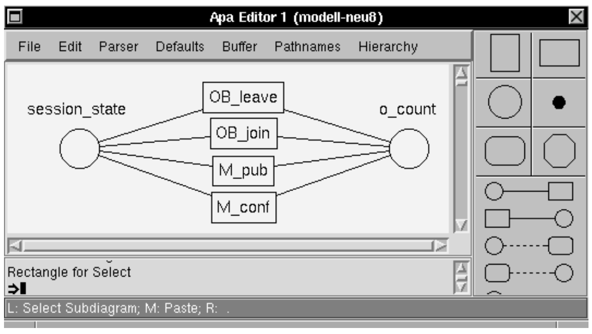
**Finite state model of the collaboration scenario.** The collaboration model described in Sect. 3 is specified for the proposed analysis method using the following *APA state components*:

$\mathbb{S} = \{s\_state, o\_count\}$ with $Z_{s\_state} = \{pub, conf\}$ and $Z_{o\_count} = \mathbb{N}_0$,
$q_0 = (q_{0s\_state}, q_{0o\_count}) = (pub, 0)$.

The set of *elementary automata* $\mathbb{E} = \{OB\_join, OB\_leave, M\_conf, M\_pub\}$ represents the possible actions that the subjects (manager and observers) can take. These specifications are represented in the data structures and initial configuration of the state components in the APA model. The lines in Fig. 1 between state components and elementary automata represent the neighbourhood relation.

From Fig. 1 we conclude that $N(e) = \mathbb{S}$ for each $e \in \mathbb{E}$.
For each $e \in \mathbb{E}$ we choose $\Phi_e = \{\#\}$. Therefore we can omit the middle component of the state transition relation $\Delta_e$.

Using the abbreviation $state = \{pub, conf\}$, it holds $\Delta_e \subset (state \times \mathbb{N}_0) \times (state \times \mathbb{N}_0)$ for each $e \in \mathbb{E}$.



*OB_join* - observer *join* collaboration,
*OB_leave* - observer *leave* collaboration,
*M_pub* - manager changes state to *pub*,
*M_conf* - manager changes state to *conf*

**Fig. 1.** Collaboration model

In detail:

$$\begin{aligned}
\Delta_{OB_{leave}} &= \{((x,y),(x,y-1)) \in (state \times \mathbb{N}_0) \times (state \times \mathbb{N}_0) \mid \\
&\quad y > 0 \land policy(observer, leave, x, y) = permit\} \\
\Delta_{OB_{join}} &= \{((x,y),(x,y+1)) \in (state \times \mathbb{N}_0) \times (state \times \mathbb{N}_0) \mid \\
&\quad y > maxOB \land policy(observer, join, x, y) = permit\} \\
\Delta_{M_{conf}} &= \{((x,y),(conf,y)) \in (state \times \mathbb{N}_0) \times (state \times \mathbb{N}_0) \mid \\
&\quad policy(manager, close, x, y) = permit\} \\
\Delta_{M_{pub}} &= \{((x,y),(pub,y)) \in (state \times \mathbb{N}_0) \times (state \times \mathbb{N}_0) \mid \\
&\quad policy(manager, open, x, y) = permit\}
\end{aligned}$$

Note that this APA is parameterised by $maxOB \in \mathbb{N}_0$.

**Definition 2.** *An elementary automaton* $(\Phi_e, \Delta_e)$ *is* activated *in a state* $q = (q_s)_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$ *as to an* interpretation $i \in \Phi_e$*, if there are* $(p_s)_{s\in N(e)} \in \bigtimes_{s\in N(e)}(Z_s)$ *with* $((q_s)_{s\in N(e)}, i, (p_s)_{s\in N(e)}) \in \Delta_e$*. An activated elementary automaton* $(\Phi_e, \Delta_e)$ *can execute a state transition and produce a* successor state $p = (p_s)_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$*, if* $q_r = p_r$ *for* $r \in \mathbb{S} \setminus N(e)$ *and* $(q_s)_{s\in N(e)}, i, (p_s)_{s\in N(e)} \in \Delta_e$*. The corresponding state transition is* $(q, (e, i), p)$*.*

For example $((conf, 0), (M\_pub, \#), (pub, 0))$ is a state transition of our example. As mentioned above, we omit # in the sequel.

**Definition 3.** *The behaviour of an APA is represented by all possible coherent sequences of state transitions starting with initial state* $q_0$*. The sequence* $(q_0, (e_1, i_1), q_1)\ (q_1, (e_2, i_2), q_2)\ (q_2, (e_3, i_3), q_3) \ldots (q_{n-1}, (e_n, i_n), q_n)$ *with* $i_k \in \Phi_{e_k}$ *represents one possible sequence of actions of an APA.* $q_n$ *is called the* goal *of this action sequence.*

*State transitions* $(p, (e, i), q)$ *may be interpreted as labelled edges of a directed graph whose nodes are the states of an APA:* $(p, (e, i), q)$ *is the edge leading from* $p$ *to* $q$ *and labelled by* $(e, i)$*. The subgraph reachable from the node* $q_0$ *is called the* reachability graph *of an APA.*

*Let* $\mathbb{Q}$ *denote the set of all states* $q \in \bigtimes_{s\in\mathbb{S}}(Z_s)$ *that are reachable from the initial state* $q_0$ *and let* $\Psi$ *denote the set of all state transitions with the first component in* $\mathbb{Q}$*.*

*The set $L \subset \Psi^*$ of all action sequences with initial state $q_0$ including the empty sequence $\epsilon$ denotes the* action language *of the corresponding APA. The action language is prefix closed. By definition $q_0$ is the goal of $\epsilon$.*

The *reachability graph* of the example depends on the parameter $maxOB \in \mathbb{N}_0$; its set of nodes is given by $\mathbb{Q}_{maxOB}$ and its set of edges is $\Psi_{maxOB}$.

It is $\mathbb{Q}_{maxOB} \subset \{pub, conf\} \times \mathbb{N}_0$ and $\Psi_{maxOB} \subset \mathbb{Q}_{maxOB} \times \mathbb{E} \times \mathbb{Q}_{maxOB}$. The reachability graph for $maxOB = 0$ is shown in Fig. 2. The reachability graph for $maxOB = 1$ is depicted by the solid lines in Fig. 3, whereas the dashed lines in the same figure show the reachability graph for $maxOB = 2$.

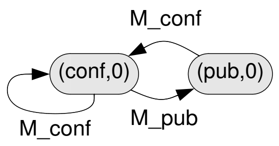


**Fig. 2.** Reachability graph for $maxOB = 0$

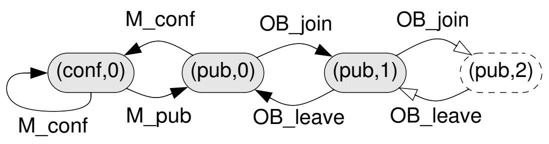


**Fig. 3.** Reachability graphs for $maxOB = 1$ (solid lines) and $maxOB = 2$ (dashed)

For example $((pub, 0), M\_conf, (conf, 0))((conf, 0), M\_pub, (pub, 0))$ is an element of the action language.

## 4.2 Abstraction Based Verification Concept

Now behaviour abstraction of an APA can be formalised by language homomorphisms, more precisely by alphabetic language homomorphisms $h : \Sigma^* \to \Sigma'^*$.

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 $h(\Sigma) \subset \Sigma' \cup \{\epsilon\}$.

It is now the question, 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 [1].

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, e.g. availability) [14].

On account of liveness aspects system properties are formalised by $\omega$-languages (sets of infinite long words). So to investigate satisfaction of properties "infinite system behaviour" has to be considered. This is formalised by so called Eilenberg limits of action languages (more precisely: by Eilenberg limits of modified action languages where maximal words are continued by an unbounded repetition of a dummy action) [15].

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 [15].

**Definition 4.** *A system* approximately satisfies *a property if and only if each finite behaviour can be continued to an infinite behaviour, which satisfies the property.*

For safety properties linear satisfaction and approximate satisfaction are equivalent [15]. To deduce approximately satisfied properties of a specification from properties of its abstract behaviour an additional property of abstractions called simplicity of homomorphisms on an action language [16] is required. Simplicity of homomorphisms is a very technical condition concerning the possible continuations of finite behaviours.

For regular languages simplicity is decidable. In [16] a sufficient condition based on the strongly connected components of corresponding automata is given, which easily can be checked. Especially: If the automaton or reachability graph is strongly connected, then each homomorphism is simple.

The following theorem [15] shows that approximate satisfaction of properties and simplicity of homomorphisms exactly fit together for verifying cooperating systems.

**Theorem 1.** *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.

In the example of this paper the desired security properties are safety and liveness properties. Generally there are more complex security properties. In [17] and [18] it has been shown how authenticity, provability and confidentiality are also treated in terms of prefix closed languages and property preserving language homomorphisms.

### 4.3 Verification Tool

The *Simple Homomorphism (SH) verification tool* [5] is used to analyse the collaboration model for different concrete values of $maxOB$. It has been developed at the *Fraunhofer-Institute for Secure Information Technology.* The SH verification tool provides components for the complete cycle from formal specification to exhaustive validation as well as visualisation and inspection of computed reachability graphs and minimal automata. The applied specification method based on *Asynchronous Product Automata (APA)* is supported by this tool. The tool manages the components of the model, allows to select alternative parts of the specification and automatically *glues* together the selected components to generate a combined model of the APA specification. After an initial state is selected, the reachability graph is automatically computed by the SH verification tool.

The tool provides an editor to define homomorphisms on action languages, it computes corresponding minimal automata [19] for the homomorphic images and checks simplicity of the homomorphisms.

*Model checking.* If it is required to inspect some or all paths of the graph to check for the violation of a security property, as it is usually the case for liveness properties, then the tool's temporal logic component can be used. Temporal logic formulae can also be checked on the abstract behaviour (under a simple homomorphism). The method for checking approximate satisfaction of properties fits exactly to the built-in simple homomorphism check [5].

The SH verification tool successfully has been applied in several security projects such as Valikrypt (`http://www.bsi.bund.de/fachthem/valikrypt/`) and CASENET[1].

## 5 Verification of the Collaboration Scenario

An outline of our verification concept for parameterised models, exemplary realised for the collaboration scenario, is given in Fig. 4.

The abstraction based verification concept introduced in Sect. 4.2 and the tool support described in Sect. 4.3 cover the part marked by solid lines in Fig. 4 whereas we now prove the components marked by dashed lines.

Using the graphs of Fig. 2 and Fig. 3 as induction base we will now prove Lemma 1 below by induction on $maxOB$. We use the abbreviations
$T_{mjoin}$ for $((pub, maxOB), OB\_join, (pub, maxOB + 1))$ and
$T_{mleave}$ for $((pub, maxOB + 1), OB\_leave, (pub, maxOB))$.

[1] The EU project CASENET (`http://www.casenet-eu.org/`) has provided a tool-supported framework for the systematic specification, design and analysis of e-commerce and e-government transactions to produce protocols with proven security properties, and to assist in code generation for these protocols.

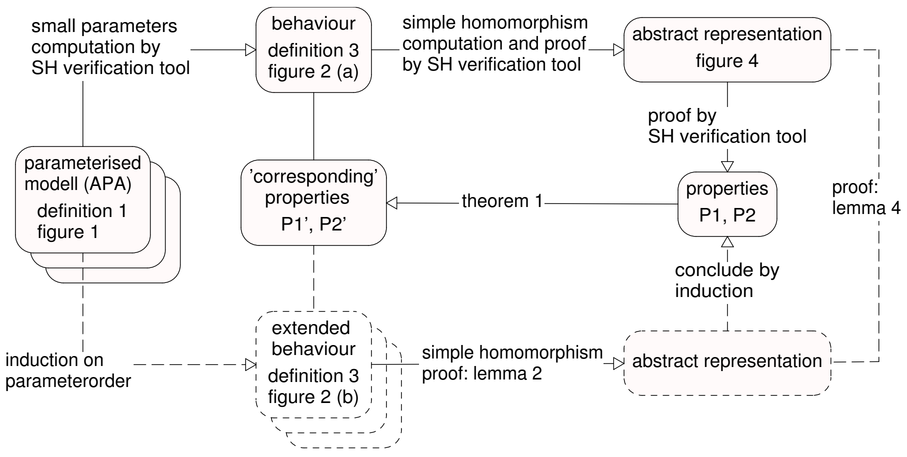

small parameters computation by SH verification tool
behaviour
definition 3
figure 2 (a)
simple homomorphism computation and proof by SH verification tool
abstract representation
figure 4
proof by SH verification tool
parameterised modell (APA)
definition 1
figure 1
'corresponding' properties
P1', P2'
theorem 1
properties
P1, P2
proof: lemma 4
conclude by induction
extended behaviour
definition 3
figure 2 (b)
induction on parameterorder
simple homomorphism proof: lemma 2
abstract representation

**Fig. 4.** Verification concept for parameterised APA

**Lemma 1.** *(a)* $\mathbb{Q}_{maxOB} = \{(pub, i) | 0 \leq i \leq maxOB\} \cup \{(conf, 0)\}$
*(b)* $\Psi_{maxOB+1} = \Psi_{maxOB} \dot\cup \{T_{mjoin}, T_{mleave}\}$

*Proof.* Figure 3 shows the reachability graph with $maxOB = 1$. Together with Fig. 2, Fig. 3 proves the induction base.
*Induction step.*
By inspection of the 4 elementary automata we get: $\Psi_{maxOB} \subset \Psi_{maxOB+1}$.
Starting from the nodes in $\mathbb{Q}_{maxOB}$ from $maxOB + 1$ only the additional transitions $T_{mjoin}$ and $T_{mleave}$ are possible.

□

It follows by induction:

**Lemma 2.** *For each* $maxOB \in \mathbb{N}_0$ *the corresponding reachability graph is finite and strongly connected.*

Let $L_{maxOB} \subset \Psi^*_{maxOB}$ denote the *action language*, then using Lemma 1(b) we can derive

**Lemma 3.** *(a)* $L_{maxOB} \subset L_{maxOB+1}$ *and*
*(b) for each* $u \in L_{maxOB+1}$*:* $h(u) \in L_{maxOB}$ *with the homomorphism* $h : \Psi^*_{maxOB+1} \to \Psi^*_{maxOB}$
*with* $h(T_{mjoin}) = \varepsilon = h(T_{mleave})$ *and* $h(x) = x$ *for* $x \in \Psi_{maxOB}$
*(c) The goal of* $u$ *is identical to the goal of* $h(u)$ *or the goal of* $u$ *is* $(pub, maxOB + 1)$ *and the goal of* $h(u)$ *is* $(pub, maxOB)$*.*

*Proof of Lemma 3 (b) by induction on the length of u.*
*Induction base.* Lemma 3(b) is true for $u = \varepsilon$. Note that by definition the goal of the empty transition sequence is equal to the initial state of the APA.

*Induction step.* Consider $ua \in L_{maxOB+1}$ with $a \in \Psi_{maxOB+1}$. From induction hypothesis there are 2 different cases:

*Case 1.* The goal of $u$ is equal to the goal of $h(u)$ and therefore an element of $\mathbb{Q}_{maxOB}$.
Therefore $a \in \Psi_{maxOB} \cup \{T_{mjoin}\}$.
For $a \in \Psi_{maxOB}$ holds: $h(ua) = h(u)h(a) = h(u)a$
Therefore from induction hypothesis $h(ua) \in L_{maxOB}$ and goals of $ua$ and $h(ua)$ are equal.
For $a = T_{mjoin}$ the goal of $u$ and therefore also the goal of $h(u)$ is $(pub, maxOB)$.
Now it holds that $h(ua) = h(u)h(a) = h(u)$.
From induction hypothesis we get that $h(ua) \in L_{maxOB}$ and goal of $ua$ is $(pub, maxOB + 1)$ and goal of $h(ua)$ is $(pub, maxOB)$.

*Case 2.* The goal of $u$ is $(pub, maxOB + 1)$ and goal of $h(u)$ is $(pub, maxOB)$.
Then: $a = T_{mleave}$
And so:
$h(ua) = h(u)h(a) = h(u) \in L_{maxOB}$ and $ua$ and also $h(ua)$ have the same goal, namely $(pub, maxOB)$. □

Now from Lemma 3 (a) we get $L_{maxOB} = h(L_{maxOB}) \subset h(L_{maxOB+1})$
and from 3 (b) we get $h(L_{maxOB+1}) \subset L_{maxOB}$
together $L_{maxOB} = h(L_{maxOB+1})$.
For each homomorphism $f : \Psi^*_{maxOB+1} \to \Sigma'^*$ with $f(T_{mjoin}) = \varepsilon = f(T_{mleave})$ it holds that: $f(L_{maxOB+1}) = f(h(L_{maxOB+1})) = f(L_{maxOB})$
and so:

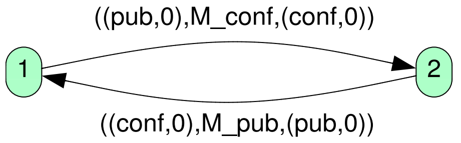
**Lemma 4.** *With the assumptions above holds:* $f(L_{maxOB+1}) = f(L_{maxOB})$

### 5.1 Proving Security and Liveness of the Collaboration Example

To consider our example's correctness we have to observe the state changes between $pub$ and $conf$. So we define an appropriate homomorphism
$c : \Psi^*_{maxOB} \to \Psi^*_{maxOB}$ by
$c(((x_1, x_2), e, (y_1, y_2))) = ((x_1, x_2), e, (y_1, y_2))$ if $x_1 \neq y_1$ , and
$c(((x_1, x_2), e, (y_1, y_2))) = \varepsilon$ if $x_1 = y_1$ .

This homomorphism $c$ fulfils the condition of Lemma 4 and therefore we get $c(L_{maxOB+1}) = c(L_{maxOB})$.

This implies $c(L_{maxOB}) = c(L_0)$ for each $maxOB \in \mathbb{N}_0$.



Initial state is 1.
All states are final states.

**Fig. 5.** Minimal automaton of $c(L_0)$

It is easy to see, that the automaton of Fig. 5 is the minimal automaton of $c(L_0)$.

This automaton shows that the collaboration is in state $conf$ only if no observer is present (P1). Moreover always state changes between *pub* and $conf$ are possible (P2).

By Lemma 2 $c$ is simple on each $L_{maxOB}$ and therefore (Theorem 1) corresponding properties P1' and P2' hold for each concrete behaviour $L_{maxOB}$. In content P1' is the same as P1. P2' is the property that always eventually state changes between *pub* and $conf$ are possible. The difference between P2 and P2' is caused by actions of the concrete behaviour which are mapped to $\varepsilon$ by the homomorphism $c$. P1' and P2' are the desired properties of the collaboration as formulated in Sect. 3.

## 6 Conclusions and Future Work

Based on property preserving abstractions (simple homomorphisms) we combined our tool supported finite state methods with induction proofs to verify security and liveness properties of a parameterised system.

We have shown how abstractions serve as a framework for individual proofs of problem specific security properties. So our results are no contradictions to well known undecidability properties of general security models e.g. Harrison-Ruzzo-Ullman.

This paper focussed on properties which are independent of concrete parameter values. Considering parameterised abstract behaviours we will extend our method to verify parameter dependent properties. The induction proofs in this paper are "handmade". So it would be desirable to support such proofs by a theorem prover. For that purpose our system specifications based on parameterised APA have to be represented in a corresponding theorem prover.

**Acknowledgements.** We would like to thank Carsten Kunz, Carsten Rudolph and Björn Steinemann for cooperation on early versions of this work and many productive discussions on the subject.

## References

1. Ochsenschläger, P., Repp, J., Rieke, R.: Abstraction and composition – a verification method for co-operating systems. Journal of Experimental and Theoretical Artificial Intelligence **12** (2000) 447–459 Copyright: ©2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.
2. Guttman, J.D., Herzog, A.L., Ramsdell, J.D.: Information flow in operating systems: Eager formal methods. IFIP WG 1.7 Workshop on Issues in the Theory of Security (2003)
3. Guttman, J.D., Herzog, A.L.: Rigorous automated network security management. International Journal of Information Security **4**(1-2) (2005) 29–48

4. Rieke, R.: Modelling and Analysing Network Security Policies in a Given Vulnerability Setting. In: Critical Information Infrastructures Security, First International Workshop, CRITIS 2006, Samos Island, Greece. Volume 4347 of LNCS., Springer (2006) 67–78 © Springer.
5. Ochsenschläger, P., Repp, J., Rieke, R., Nitsche, U.: The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. Formal Aspects of Computing, The International Journal of Formal Method **11** (1999) 1–24
6. Ip, C.N., Dill, D.L.: Verifying Systems with Replicated Components in Mur$\varphi$. Formal Methods in System Design **14**(3) (1999) 273–310
7. Derepas, F., Gastin, P.: Model checking systems of replicated processes with spin. In: SPIN '01: Proceedings of the 8th international SPIN workshop on Model checking of software, New York, NY, USA, Springer-Verlag New York, Inc. (2001) 235–251
8. Lakhnech, Y., Bensalem, S., Berezin, S., Owre, S.: Incremental verification by abstraction. In Margaria, T., Yi, W., eds.: TACAS. Volume 2031 of Lecture Notes in Computer Science., Springer (2001) 98–112
9. Basu, S., Ramakrishnan, C.R.: Compositional analysis for verification of parameterized systems. Theor. Comput. Sci. **354**(2) (2006) 211–229
10. Milner, R.: Communication and Concurrency. International Series in Computer Science. Prentice Hall (1989)
11. Bradfield, J., Stirling, C.: Modal logics and mu-calculi: an introduction (2001)
12. Uribe, T.E.: Combinations of model checking and theorem proving. In: FroCoS '00: Proceedings of the Third International Workshop on Frontiers of Combining Systems, London, UK, Springer-Verlag (2000) 151–170
13. Moses, T.: eXtensible Access Control Markup Language (XACML), Version 2.0. Technical report, OASIS Standard (2005)
14. Alpern, B., Schneider, F.B.: Defining liveness. Information Processing Letters **21**(4) (1985) 181–185
15. Nitsche, U., Ochsenschläger, P.: Approximately satisfied properties of systems and simple language homomorphisms. Information Processing Letters **60** (1996) 201–206
16. Ochsenschläger, P.: Verification of cooperating systems by simple homomorphisms using the product net machine. In Desel, J., Oberweis, A., Reisig, W., eds.: Workshop: Algorithmen und Werkzeuge für Petrinetze, Humboldt Universität Berlin (1994) 48–53
17. Gürgens, S., Ochsenschläger, P., Rudolph, C.: On a formal framework for security properties. International Computer Standards & Interface Journal (CSI), Special issue on formal methods, techniques and tools for secure and reliable applications (2004)
18. Gürgens, S., Ochsenschläger, P., Rudolph, C.: Abstractions preserving parameter confidentiality. In: Computer Security – ESORICS 2005. (2005) 418–437 Copyright: ©2005, Springer Verlag.
19. Eilenberg, S.: Automata, Languages and Machines. Volume A. Academic Press, New York (1974)