Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 203 (2009) 87–101

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

# Composing Modal Properties of Programs with Procedures

Marieke Huisman[1] ,[2]

*INRIA Sophia Antipolis, France*

Dilian Gurov[3] ,[4]

*Royal Institute of Technology, Stockholm, Sweden*

Abstract

In component based software design, formal reasoning about programs has to be compositional, allowing global, program-wide properties to be inferred from the properties of its components. The present paper addresses the problem of compositional verification of behavioural control flow properties of sequential programs with procedures, expressed in a modal logic. We use as a starting point a maximal model based method previously developed by the authors, which assumes the local properties to be structural (rather than behavioural). To handle local behavioural properties, we propose the combination of the above method with a translation from behavioural properties to sets of structural ones. The present paper presents a direct solution for the logic, and prepares the ground for a translation for the considerably more expressive logic obtained by adding greatest fixed-point recursion.

*Keywords:* Program Behaviour, Program Structure, Compositional Verification

## 1 Introduction

### Background

Component based software design is a design paradigm where several software components, each with their own well-described functionality, are combined into a single application. This technique is becoming increasingly widespread as a means of constructing advanced, complex software systems. However, to ensure the well-functioning and security of the application as a whole, one needs compositional

[1] Partially funded by the IST FET programme of the EC, under the IST-2005-015905 MOBIUS project.

[2] Email: `Marieke.Huisman@sophia.inria.fr`

[3] Partially funded by the IST FP6 programme of the EC, under the IST-FP6-STREP-27004 S3MS project.

[4] Email: `dilian@csc.kth.se`

doi:10.1016/j.entcs.2009.03.028

verification techniques that allow to conclude this from the functionality and security properties of the components. Such guarantees are in particular necessary when the different components in the system can communicate and collaborate with each other. In the context of mobile code, development of a compositional verification techniques becomes even more pertinent, because here new applications can be downloaded post-issuance on a running system.

In general, the problem of *compositional verification* that we study is the following: if we wish to ensure that a composed application has global property $\psi$, can we find local properties $\phi$ for the different components, that are sufficient to ensure that the composed application has property $\psi$. This idea can be described by the following proof principle (where $X$ is a component variable):

$$\frac{\vdash A : \phi \qquad X : \phi \vdash X \otimes B : \psi}{\vdash A \otimes B : \psi}$$

This principle reduces the problem of showing that the composition of $A$ and $B$ satisfies $\psi$ to the following tasks:

- decompose the global property $\psi$ by finding a local property $\phi$ of $A$,
- prove correctness of the decomposition, *i.e.*, verify that, for *any* $X$ satisfying $\phi$, $X$ composed with $B$ satisfies $\psi$ (second premise), and
- when a concrete implementation of $A$ has been chosen, verify that it satisfies the local property $\phi$ (first premise).

In our work, we focus on *applets*, that are components described in a sequential procedural language. Applets come equipped with an applet interface $I$ that describes the methods that are defined and required by its implementation.

## Compositional verification with maximal models

In earlier work we proposed an approach to compositional verification based on *maximal models* [9], provided tool support, and evaluated its practical applicability by means of an industrial, electronic purse case study provided by smart card producer Gemplus [3]. To show that a local property $\phi$ is sufficient to ensure a global property $\psi$, we compute a so-called maximal model *w.r.t.* $\phi$. This model is maximal in the sense that it simulates all applets having property $\phi$. We have shown how maximal models can be computed for our *simulation logic*, a fragment of the modal $\mu$-calculus with box modalities and greatest fixed-points only. We distinguish two kind of properties: structural and behavioural. Structural properties restrict the set of possible applet implementations, while behavioural properties restrict the possible behaviours. For structural property $\phi$ and applet interface $I$, we can define a maximal applet $\theta_I(\phi)$ that simulates exactly all applets with interface $I$, having property $\phi$. Therefore we can prove that the following proof principle is sound and complete (Theorem 3.11 in [9]):

$$(\mathsf{structure})\ \frac{\mathcal{A} \models_s \phi \qquad \theta_I(\phi) \uplus \mathcal{B} \models_b \psi}{\mathcal{A} \uplus \mathcal{B} \models_b \psi}\ \mathcal{A} : I$$

This principle should be understood as follows. Suppose we have a local structural assumption $\phi$, describing applets with interface $I$[5]. If we wish to show that any applet having this property can safely interact with some already known applet $\mathcal{B}$, and in particular that this respects some global behavioural safety property $\psi$, we do this by constructing the maximal applet $\theta_I(\phi)$ for $\phi$ and $I$, compose it with $\mathcal{B}$, and model check (using a suitable algorithm, see for a survey [1]) that the resulting applet satisfies $\psi$. To safely compose applets $\mathcal{A}$ and $\mathcal{B}$, one has then also to model check that applet $\mathcal{A}$ indeed satisfies the local structural assumption $\phi$.

However, for local behavioural properties the situation is more problematic. In general, given behavioural property $\phi$ and interface $I$, there does not exist a unique maximal applet that simulates all applets (with interface $I$) with property $\phi$. Suppose we have the following behavioural formula $[a\ \mathsf{call}\ b]\,r$, meaning that immediately after a call from method $a$ to method $b$ (that is, at the entry point of $b$), the atomic proposition $r$ should hold. Even for this simple formula there are two maximal applets (providing and requiring the methods $a$ and $b$): the first is 'maximal' but has no call edges labelled $b$ with source an entry node of $a$ (unless this entry node is also a return node, meaning that the call will never be reached), while the second is 'maximal' but has no entry point of $b$ which is valuated $r$. Every applet satisfying the formula is simulated by one of these two applets; however, the two applets do not simulate each other.

## Characterising behavioural properties with applet structure

There are several ways of addressing this problem. A first possibility is to over–approximate the behaviour with a model which is not necessarily a behaviour. The maximal model of the behavioural formula gives one such approximation. A better approximation can be obtained through a standard product construction between the applet PDA induced by the maximal applet for the given interface and this maximal model. However, this inherently results in an incomplete verification principle.

Therefore, we take another approach and we aim at computing the whole set of maximal applet structures, by characterising a behavioural formula by a set of structural formulae. This paper describes how behavioural modal logic formulae using box modalities only can be characterised by a set of structural formulae. We call this fragment of simulation logic *modal simulation logic.* It is ongoing work to extend this work to greatest fixed-points; this requires the use of a tableau construction and a global discharge condition.

We show that in case the formula does not contain disjunctions, this characterisation is exact. Below, we present a *translation* $\Pi$ from behavioural modal simulation logic formulae into (equivalent) sets of structural properties. This is a first step towards the possibility to apply our compositional verification techniques also when local assumptions are behavioural properties. If a behavioural property can be characterised by a set of structural properties, we can apply our maximal applet construction defined for structural properties (see [9]) to obtain a set of maximal

[5] Interfaces will be defined formally below. They specify the set of methods that is known by the applet.

models for the behavioural property. The compositional verification principle then becomes the following:

$$\frac{\mathcal{A} \models_b \phi \qquad\qquad \{\theta_I(\sigma) \uplus \mathcal{B} \models_b \chi\}_{\sigma \in \Pi_I(\phi)}}{\mathcal{A} \uplus \mathcal{B} \models_b \chi} \mathcal{A} \colon I_{\mathcal{A}}$$

Notice that instead of showing $\mathcal{A} \models_b \phi$ it suffices to show $\mathcal{A} \models_s \sigma$ for some $\sigma \in \Pi_I(\phi)$.

Apart from filling up a gap in our compositional verification method, this characterisation also has a more general interest in itself, as it reveals the relation between structure and behaviour. In particular, in our translation we exploit that a behavioural property is trivially satisfied by an applet if the applet structure does not allow this behaviour.

### Related work

Compositional verification of concurrent programs has been studied extensively, especially in the form of assumption/commitment based reasoning about processes with synchronous message passing, and in the form of rely/guarantee based reasoning for shared-variable concurrency; see *e.g.* De Roever *et al.* [7] for a systematic overview of these and related proof methods. However, these techniques do not address programs with recursive procedures. The use of maximal models for algorithmic compositional verification is due to Grumberg and Long [2] for the universal fragment of CTL, later extended to CTL* by Kupferman and Vardi [5]. These works study synchronous parallel compositions of sequential processes under fairness assumptions. We adopt this approach to simulation logic in the present context of compositional verification of control-flow properties of sequential programs with procedures in [9]. To the best of our knowledge, the present work is the first to address the characterisation of behavioural control-flow properties in terms of structural ones.

### Structure of the paper

Section 2 gives a short overview of our earlier results, in particular the program structure and behaviour, the logic and the definition of maximal model. It also introduces useful notations. Section 3 defines the translation from behavioural modal simulation logic formulae to structural formulae, and proves its correctness. Section 4 proves soundness and completeness of a compositional verification principle for behavioural properties without disjunctions, provided the local assumptions are described by behavioural modal simulation logic. Finally, Section 5 gives conclusions and discusses future work.

## 2 Preliminaries: model and logic

We briefly recall some definitions and results that form the basis for our compositional verification method. For a full overview, the reader is referred to [8,9].

### 2.1 Modal simulation logic

We use a subset of modal logic as our specification language. In our work on compositional verification, we exploited that formulae in this logic, extended with greatest fixed-points can be characterised by simulation, and vice versa; therefore we call that logic simulation logic. The subset of simulation logic that we consider in this paper is *modal simulation logic.* Throughout, we fix a set of labels $L$ and a set of atomic propositions $A$.

**Definition 2.1 (Modal Simulation Logic)** *The formulae of* modal simulation logic *are inductively defined by:*

$$\phi ::= p \mid \neg p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid [a]\, \phi$$

*where* $p \in A$ *and* $a \in L$*.*

The semantics of the logic is standard for modal logics and is given in terms of models (Kripke structures). In particular, formula $[a]\, \phi$ holds of a state ("possible world") if $\phi$ holds in all states accessible from the former state via an edge labelled $a$. Notice that the constant formulae *true* and *false* are definable; they shall be denoted $\mathsf{tt}$ and $\mathsf{ff}$, respectively. For convenience, we shall use $\phi_1 \Rightarrow \phi_2$ to abbreviate $\neg\phi_1 \vee \phi_2$.

Next, we define a general notion of model and specifications.

**Definition 2.2 (Model)** *A* model *is a structure* $\mathcal{M} = (S, L, \rightarrow, A, \lambda)$*, where* $S$ *is a set of states,* $\rightarrow \subseteq S \times L \times S$ *a labelled transition relation, and* $\lambda \colon S \rightarrow \mathcal{P}(A)$ *a valuation, assigning to each state* $s$ *the atomic propositions that hold in* $s$*. A* specification $\mathcal{S}$ *is a pair* $(\mathcal{M}, E)$*, where* $\mathcal{M}$ *is a model and* $E \subseteq S$ *is a set of states.*

Intuitively, one can think of $E$ as the set of entry states of the model. We define the usual notions of satisfaction $(\mathcal{M}, s) \models \phi$ and simulation $(\mathcal{M}_1, s_1) \leq (\mathcal{M}_2, s_2)$ (where related states satisfy the same atomic propositions). A specification satisfies a formula if all its entry states satisfy the formula. A specification is simulated by another specification if for all its entry states there exists an entry state in the other specification, that simulates this first entry state. This simulation relation preserves (backwards) logical properties.

**Theorem 2.3** $\mathcal{S}_1 \leq \mathcal{S}_2$ *and* $\mathcal{S}_2 \models \phi$ *implies* $\mathcal{S}_1 \models \phi$

**Proof** Corollary 2.16 in [9] □

### 2.2 Applet structure and behaviour

Our program model, inspired by [4], is control–flow based and thus over–approximates actual program behaviour. It defines two different views on applets: a structural and a behavioural view. Both views are instantiations of the general notions of model and specification. Notice in particular that these instantiations yield a structural and a behavioural version of simulation and simulation logic. Again, we refer to [8,9] for more detail.

### *2.2.1 Applet Structure*

Since we abstract away from all data, applet structure is defined as a collection of call graphs for the methods that the applet implements. Let $\mathcal{M}eth$ be a countably infinite set of method names. A method specification is an instance of the general notion of specification.

**Definition 2.4 (Method specification)** *A* method graph *for $m \in \mathcal{M}eth$ over a finite set $M \subseteq \mathcal{M}eth$ of method names is a finite model*

$$\mathcal{M}_m = (V_m, L_m, \to_m, A_m, \lambda_m)$$

*where $V_m$ is the set of control nodes of $m$, $L_m = M \cup \{\varepsilon\}$, $A_m = \{m, r\}$, and $\lambda_m : V_m \to \mathcal{P}(A_m)$ is so that $m \in \lambda_m(v)$ for all $v \in V_m$ (*i.e. *each node is tagged with its method name). The nodes $v \in V_m$ with $r \in \lambda_m(v)$ are called* return points. *A* method specification *for $m \in \mathcal{M}eth$ over $M$ is a pair $(\mathcal{M}_m, E_m)$, where $\mathcal{M}_m$ is a method graph for $m$ over $M$ and $E_m \subseteq V_m$ is a non–empty set of* entry points *of $m$.*

Next we define the notion of applet interface.

**Definition 2.5 (Applet interface)** *An* applet interface *is a pair $I = (I^+, I^-)$, where $I^+, I^- \subseteq \mathcal{M}eth$ are finite sets of names of* provided *and* required *methods, respectively. The* composition *of two interfaces $I_1 = (I_1^+, I_1^-)$ and $I_2 = (I_2^+, I_2^-)$ is defined by $I_1 \cup I_2 = (I_1^+ \cup I_2^+, I_1^- \cup I_2^-)$.*

To formally define the notion *applet with interface*, we use the notion of disjoint union of specifications $\mathcal{S}_1 \uplus \mathcal{S}_2$, where each state is tagged with 1 or 2, respectively, and $(s, i) \xrightarrow{a}_{\mathcal{S}_1 \uplus \mathcal{S}_2} (t, i)$, for $i \in \{1, 2\}$, if and only if $s \xrightarrow{a}_{\mathcal{S}_i} t$.

**Definition 2.6 (Applet)** *An* applet *$\mathcal{A}$ with implementation interface $I$, written $\mathcal{A} : I$, is defined inductively by*

- *$(\mathcal{M}_m, E_m) : (\{m\}, M)$ if $(\mathcal{M}_m, E_m)$ is a method specification for $m \in \mathcal{M}eth$ over $M$, and*
- *$\mathcal{A}_1 \uplus \mathcal{A}_2 : I_1 \cup I_2$ if $\mathcal{A}_1 : I_1$ and $\mathcal{A}_2 : I_2$.*

An applet is *closed* if $I^- \subseteq I^+$, *i.e.* it does not require any external methods.

Simulation and satisfaction, instantiated to this particular type of models are called structural simulation $\leq_s$, and structural satisfaction $\models_s$, respectively.

**Example 2.7** *As an illustration of properties in structural modal simulation logic, consider first formula $a \Rightarrow [b]\,\mathsf{ff}$ (which, as explained above, abbreviates $\neg a \vee [b]\,\mathsf{ff}$). It holds of a control node $v$ if either $v \models \neg a$ (that is, $a \notin \lambda(v)$, meaning $v$ does not belong to method $a$) or else all nodes $v'$ such that $v \xrightarrow{b}_a v'$ satisfy $\mathsf{ff}$, meaning no such nodes $v'$ exist (since no node satisfies $\mathsf{ff}$). The formula holds of an applet if it holds for all its entry nodes; it hence specifies that from any entry node of method $a$, there is no call edge to method $b$. Similarly, the formula $b \Rightarrow r$ specifies that no entry node of method $b$ is a return node.*

Modal logic is not capable of expressing general invariant or reachability properties such as "no call edge to method $b$ is reachable from an entry node of method $b$"; however, such properties are easily expressed in full structural simulation logic (see [9]).

### 2.2.2 Applet behaviour

Next we instantiate specifications on the behavioural level.

$$\text{(transfer)}\ \frac{m \in I^+ \qquad v \to_m v' \qquad v \models \neg r}{(v,\sigma) \xrightarrow{\tau} (v',\sigma)}$$

$$\text{(call)}\ \frac{\begin{array}{ccc} m_1, m_2 \in I^+ & v_1 \xrightarrow{m_2}_{m_1} v_1' & v_1 \models \neg r \\ v_2 \models m_2 & & v_2 \in E \end{array}}{(v_1,\sigma) \xrightarrow{m_1 \mathsf{call}\ m_2} (v_2, v_1' \cdot \sigma)}$$

$$\text{(return)}\ \frac{m_1, m_2 \in I^+ \qquad v_2 \models m_2 \wedge r \qquad v_1 \models m_1}{(v_2, v_1 \cdot \sigma) \xrightarrow{m_2\ \mathsf{ret}\ m_1} (v_1,\sigma)}$$

Figure 1. Applet Transition Rules

**Definition 2.8 (Behaviour)** *Let $\mathcal{A} = (\mathcal{M}, E) : I$ be a closed applet where $\mathcal{M} = (V, L, \to, A, \lambda)$. The* behaviour *of $\mathcal{A}$ is described by the specification $b(\mathcal{A}) = (\mathcal{M}_b, E_b)$, where $\mathcal{M}_b = (S_b, L_b, \to_b, A_b, \lambda_b)$ is such that $S_b = V \times V^*$,* i.e. *states (also called configurations) are pairs of control points and stacks, $L_b = \{m_1\ k\ m_2 \mid k \in \{\mathsf{call}, \mathsf{ret}\},\ m_1, m_2 \in I^+\} \cup \{\tau\}$, $\to_b$ is defined by the rules of Figure 1, $A_b = A$, and $\lambda_b((v,\sigma)) = \lambda(v)$.*

*The set of initial states $E_b$ is defined by $E_b = E \times \{\epsilon\}$, where $\epsilon$ denotes the empty sequence over $V$.*

Note that applet behaviour defines a pushdown automaton. We exploit this by using a model checker for PDAs to verify behavioural properties (see, *e.g.*, [1] for a survey of verification techniques for infinite process structures).

Also on the behavioural level, we instantiate the definitions of simulation $\leq_b$ and satisfaction $\models_b$: $\mathcal{A}_1 \leq_b \mathcal{A}_2 \Leftrightarrow b(\mathcal{A}_1) \leq b(\mathcal{A}_2)$ and $\mathcal{A} \models_b \phi \Leftrightarrow b(\mathcal{A}) \models \phi$. Any two applets that are related by structural simulation, are also related by behavioural simulation (Theorem 3.9 in [9]), but the converse is not true (since behavioural simulation only requires reachable states to be related).

**Example 2.9** *As an example of a property in behavioural modal simulation logic, consider formula $[a\ \mathsf{call}\ b]\, r$. A configuration $(v,\sigma)$ satisfies the formula if all configurations that are reached from $(v,\sigma)$ by performing a call from method $a$ to method $b$ satisfy $r$ (that is, have a return node as control point). The formula holds of an applet if it holds of all its initial configurations; it hence specifies that if the first action of the applet is a call from method $a$ to method $b$, then immediately afterwards* *we should be at a return node.*

Again, invariant behavioural properties such as "method $a$ never calls method $b$" are beyond the expressive power of modal logic, but are easily expressed in full behavioural simulation logic.

### *2.2.3 Clean applet structures*

Notice that method specifications allow return points to have outgoing edges. However, the characterisation of behavioural properties by a set of structural formulae defined later is only correct if the applet has no outgoing edges in return nodes; such applets are referred to as *clean*. We define a unary operation of *cleaning* returning a clean applet having the same behaviour.

**Definition 2.10 [Cleaning]** *Given a method specification* $\mathcal{M}_m = (V_m, L_m, \to_m, A_m, \lambda_m)$*, the unary operation of cleaning is defined by:*

$$\mathcal{M}^\bullet = (V_m, L_m, \{s \xrightarrow{l}_m t \mid s \xrightarrow{l}_m t \wedge r \notin \lambda_m(s)\}, A_m, \lambda_m)$$

It is easy to see that cleaned applets are clean. We list several useful properties of cleaning.

$$(\mathcal{A}^\bullet)^\bullet = \mathcal{A}^\bullet$$
$$\mathcal{A} \models_b \phi \Leftrightarrow \mathcal{A}^\bullet \models_b \phi$$
$$(\mathcal{A}^\bullet, s) \models_s r \Rightarrow \forall l, \sigma.\, (\mathcal{A}^\bullet, s) \models_s [l]\,\sigma$$

Thus, cleaning is idempotent and preserves behavioural properties. And any state that is a return point, trivially satisfies any box formula at the structural level. Below, in the correctness proof of the characterisation, we will define a notion of reachability (the set of nodes that can be reached by a behaviour), and we will use that on clean applets this coincides with the satisfaction of box formulae.

## *2.3 Compositional verification using maximal applets*

Our compositional verification method, presented in [9], is based on the computation of maximal models for a property $\phi$. A model is said to be *maximal* for property $\phi$, if it simulates all other models having property $\phi$. However, when we have a property $\phi$ over applet structure (or applet behaviour), we can not be sure that the maximal model of $\phi$ is also a legal applet structure (or applet behaviour). For applet structures, this problem can be solved, because we can precisely characterise legal applet structures *w.r.t.* an interface $I$ as a formula in simulation logic (instantiated at the structural level). If $\phi_I$ is the characteristic formula for an applet with interface $I$, and $\phi$ is an arbitrary structural formula, then the maximal model of the formula $\phi \wedge \phi_I$ precisely characterises all applets with interface $I$ satisfying $\phi$. Thus, if we define the maximal applet *w.r.t.* $\phi$ and $I$, denoted $\theta_I(\phi)$, to be the maximal model of the property $\phi \wedge \phi_I$, then we can prove that the following compositional verification principle is sound and complete for full simulation logic (Theorem 3.11 in [9]):

$$\text{(structure)}\ \frac{\mathcal{A} \models_s \phi \qquad \theta_I(\phi) \uplus \mathcal{B} \models_b \psi}{\mathcal{A} \uplus \mathcal{B} \models_b \psi}\ \mathcal{A} : I$$

For our subset modal simulation logic, we can only prove soundness of this rule. To establish completeness for full simulation logic, we use that every model can be characterised with a characteristic formula. But, since specifications can contain loops, the availability of greatest fixed-points is essential here.

As explained above, there is no such way to precisely characterise applet behaviour, thus we cannot establish soundness and completeness of the same compositional verification principle for local formulae over applet behaviour. Below we will define a translation of behavioural formulae into sets of structural formulae, which will allow us to exploit the compositional verification principle for applet structure to perform compositional verification of applet behaviour properties.

### 2.4 Notational conventions

We use label $\varepsilon$ for transfer edges in applet structures, and label $\tau$ for silent behavioural transitions.

For sequences $\sigma$, we use $\sigma^\sharp$ to denote the top element (head), and $\sigma^\flat$ to denote the sequence with the top element removed (tail), *i.e.* $(v \cdot \sigma)^\sharp = v$ and $(v \cdot \sigma)^\flat = \sigma$. The empty sequence is denoted as $\epsilon$, where $\epsilon^\sharp$ and $\epsilon^\flat$ are undefined.

In our translation of modal simulation logic formulae we allow sequences $\alpha$ of labels to appear in box modalities, with the obvious translation $\hat{\cdot}$ to standard formulae:

$$\widehat{[\epsilon]\,\psi} = \psi$$
$$\widehat{[l \cdot \alpha]\,\psi} = [l]\,\widehat{[\alpha]\,\psi}$$

## 3 Characterising behavioural properties

We shall assume throughout this section that applets are clean; if they are not, they can be cleaned as explained above without changing the behaviour. We define a mapping $\Pi$ from behavioural properties to sets of structural properties, for which we can prove that

$$\mathcal{A} \models_b \phi \;\Leftarrow\; \exists \sigma \in \Pi(\phi).\mathcal{A} \models_s \sigma$$

The reason for this not being an equivalence is that (unlike the remaining connectives of the logic) disjunction cannot be treated compositionally: the validity of $\mathcal{A} \models_b \phi \vee \psi$ cannot be inferred just from the validity (or invalidity) of $\mathcal{A} \models_b \phi$ and $\mathcal{A} \models_b \psi$, since $\mathcal{A} \models_b \phi \vee \psi$ may hold because some initial configurations of $\mathcal{A}$ satisfy $\phi$ while the rest satisfy $\psi$. However, if behavioural property $\phi$ does not contain disjunction, the mapping is exact, and we obtain an equivalence (*i.e.* we find the precise set of structural formulae characterising a behavioural property).

$$\mathcal{A} \models_b \phi \;\Leftrightarrow\; \exists \sigma \in \Pi(\phi).\mathcal{A} \models_s \sigma \qquad (\phi\,\text{disjunction-free}) \tag{1}$$

$$\begin{aligned}
\pi_{(i,F)\cdot H}(p) &= \{i \Rightarrow [F]\, p\} \cup \{i' \Rightarrow [F']\, \mathsf{ff} \mid (i',F') \in H\}\\
\pi_{(i,F)\cdot H}(\neg p) &= \{i \Rightarrow [F]\, \neg p\} \cup \{i' \Rightarrow [F']\, \mathsf{ff} \mid (i',F') \in H\}\\
\pi_{(i,F)\cdot H}(\phi_1 \wedge \phi_2) &= \{\sigma_1 \wedge \sigma_2 \mid \sigma_1 \in \pi_{(i,F)\cdot H}(\phi_1),\ \sigma_2 \in \pi_{(i,F)\cdot H}(\phi_2)\}\\
\pi_{(i,F)\cdot H}(\phi_1 \vee \phi_2) &= \pi_{(i,F)\cdot H}(\phi_1) \cup \pi_{(i,F)\cdot H}(\phi_2)\\
\pi_{(i,F)\cdot H}([\tau]\, \phi) &= \pi_{(i,F\cdot\varepsilon)\cdot H}(\phi)\\
\pi_{(i,F)\cdot H}([a\ \mathsf{call}\ b]\, \phi) &= \begin{cases} \{\mathsf{tt}\} & \text{if } i \neq a\\ \pi_{(b,\epsilon)\cdot(i,F\cdot b)\cdot H}(\phi) & \text{if } i = a \end{cases}\\
\pi_{(i,F)\cdot H}([a\ \mathsf{ret}\ b]\, \phi) &= \begin{cases} \{\mathsf{tt}\} & i \neq a \ \vee\ H^{\flat} = \epsilon \ \vee\ \pi_1(H^{\sharp}) \neq b\\ \{i \Rightarrow [F]\, \neg r\} \cup \pi_H(\phi) & \text{if } i = a \wedge \pi_1(H^{\sharp}) = b \end{cases}
\end{aligned}$$

Figure 2. The mapping $\pi_H$

## 3.1 Mapping behavioural properties into structural properties

Our translation is based on a symbolic execution of the behavioural property. First, we define the (auxiliary) mapping $\pi_H : \mathit{Behform} \rightarrow \mathcal{P}(\mathit{Structform})$, parametrised by a non-empty *history stack* $H$. Each element in the history stack is a tuple containing the current method name and a sequence (called frame) of edge labels, *i.e.*: $H\colon (I^{+} \times (I^{-} \cup \{\varepsilon\})^{*})^{+}$.

The mapping $\pi$ is initially applied with an initial history stack containing a single element with an empty frame. We use $\varnothing_{H,m}$ to denote this single element sequence $(m, \epsilon)$.

The history stack is updated as follows:

- when the behavioural property prescribes a call from $a$ to $b$, and the top element of $H$ is in method $a$, we add $b$ at the end of this frame, and we push a new element $(b, \epsilon)$ onto $H$;
- when the behavioural property prescribes a return from $a$ to $b$, the top element of $H$ is in method $a$ and the previous element is in method $b$, we pop the top element from $H$; and
- when the behavioural property prescribes an internal transfer, we append $\varepsilon$ to the end of the frame of the top element of $H$.

The mapping $\pi$ is defined in Figure 2 by induction on the structure of the formula. The mapping symbolically executes the formula, and for every box modality that it encounters labelled with a return, it generates one structural formula capturing the possibility that this behaviour cannot happen. The recursive call will then generate other structural formulae, that will have to hold in case the described behaviour actually took place. Symbolic execution of the box modality labelled with a call or an internal transfer does not produce any explicit structural formula (capturing the possibility that this transition is not possible), because such a formula would always be subsumed by the structural formulae generated by the remaining formula. When the translation encounter (the negation of) an atomic proposition, it generates a formula characterising the case that the atomic proposition holds in the current frame. In addition, it generates formulae characterising the possibility that the applet does not satisfy one or more of the structural constraints collected in the history stack (in which case the current atomic proposition would never be evaluated). Notice that $\pi_H(\mathsf{tt}) = \{\mathsf{tt}\}$. Notice further that non-emptiness of the history stack is an invariant of the construction.

Finally, we can define the mapping of $\phi$ *w.r.t.* a given interface $I$:

$$\Pi_I(\phi) = \{\textstyle\bigwedge_{m \in I^+} \sigma_m \mid \sigma_m \in \pi_{\varnothing_{H,m}}(\phi)\}$$

This last expression gives rise to an explosion in the number of formulae, but note that for the compositional verification we only need the weakest structural formulae of this set. In particular, as soon as a set contains tt, all other elements can be removed from the set, because any applet will satisfy the local assumption tt.

### *3.2 Examples*

The working of this mapping is best explained with some examples. Throughout we assume $I = (\{a, b, c\}, \{a, b, c\})$.

**Example 3.1** *Consider the following translations.*

$$\pi_{\varnothing_{H,a}}([a\ \mathsf{call}\ a]\, r) = \pi_{(a,\epsilon)\cdot(a,a)}(r) = \{a \Rightarrow r, a \Rightarrow [a]\,\mathsf{ff}\}$$

$$\pi_{\varnothing_{H,a}}([a\ \mathsf{call}\ b]\,[b\ \mathsf{call}\ a]\,[a\ \mathsf{call}\ b]\, r) = \{b \Rightarrow r, a \Rightarrow [b]\,\mathsf{ff}, b \Rightarrow [a]\,\mathsf{ff}\}$$

$$\pi_{\varnothing_{H,a}}([a\ \mathsf{call}\ b]\,[b\ \mathsf{ret}\ a]\,[a\ \mathsf{call}\ c]\, r) = \{c \Rightarrow r, a \Rightarrow [b \cdot c]\,\mathsf{ff}, b \Rightarrow \neg r\}$$

$$\pi_{\varnothing_{H,a}}([a\ \mathsf{call}\ b]\,[b\ \mathsf{ret}\ d]\,\mathsf{ff}) = \{\mathsf{tt}\}$$

*The first property concerns a selfcall, the second a callback and the third a return. The last property shows how nonsense returns (causing formulae to be vacuously true) are detected.*

The next example shows a computation for the whole interface with a call behaviour.

**Example 3.2** *We examine the property from Example 2.9.*

$$\pi_{\varnothing_{H,a}}([a\ \mathsf{call}\ b]\, r) = \{b \Rightarrow r, a \Rightarrow [b]\,\mathsf{ff}\}$$

$$\pi_{\varnothing_{H,b}}([a\ \mathsf{call}\ b]\, r) = \{\mathsf{tt}\}$$

$$\pi_{\varnothing_{H,c}}([a\ \mathsf{call}\ b]\, r) = \{\mathsf{tt}\}$$

$$\Pi_I([a\ \mathsf{call}\ b]\, r) = \{b \Rightarrow r \wedge \mathsf{tt} \wedge \mathsf{tt}, a \Rightarrow [b]\,\mathsf{ff} \wedge \mathsf{tt} \wedge \mathsf{tt}\}$$

And finally, the last example shows a computation for the whole interface with only internal behaviour.

**Example 3.3**

$$\begin{aligned}\pi_{\varnothing_{H,a}}([\tau]\,\mathsf{ff}) &= \{a \Rightarrow [\varepsilon]\,\mathsf{ff}\}\\ \pi_{\varnothing_{H,b}}([\tau]\,\mathsf{ff}) &= \{b \Rightarrow [\varepsilon]\,\mathsf{ff}\}\\ \pi_{\varnothing_{H,c}}([\tau]\,\mathsf{ff}) &= \{c \Rightarrow [\varepsilon]\,\mathsf{ff}\}\end{aligned}$$

$$\Pi_I([\tau]\,\mathsf{ff}) = \{a \Rightarrow [\varepsilon]\,\mathsf{ff} \wedge b \Rightarrow [\varepsilon]\,\mathsf{ff} \wedge c \Rightarrow [\varepsilon]\,\mathsf{ff}\}$$

## *3.3 Correctness of the Translation*

To show the correctness of the translation, we generalise the notion of satisfaction by relativising it on the history stack. As for the translation, in the proof we also assume applet $\mathcal{A}$ to be clean.

Intuitively, the generalised notion of satisfaction, relativised *w.r.t.* $H$, can be understood as follows: $\mathcal{A} \models_H \phi$ holds for applet $\mathcal{A}$ and formula $\phi$ if for any node $v$ in method $i$ that can be reached by following the path described by the frame in the top element of $H$ and for any callstack $\sigma$ that corresponds to the rest of the history stack we have $(v, \sigma) \models_b \phi$.

To define this formally, we need several auxiliary definitions. First we define a function reach, that computes given applet $\mathcal{A}$, a set of nodes $V$ and sequence of labels $L$, which nodes are reachable in $\mathcal{A}$ from the nodes in $V$, following edges with the labels in $L$.

$$\begin{aligned}\mathsf{reach}_{\mathcal{A}}(V, \epsilon) &= V\\ \mathsf{reach}_{\mathcal{A}}(V, l \cdot L) &= \mathsf{reach}_{\mathcal{A}}(\{v' \mid v \in V \wedge v \xrightarrow{l}_{\mathcal{A}} v'\}, L)\end{aligned}$$

The correspondence between concrete callstack $\sigma$ and history stack $H$ is defined as follows: if $\sigma^\sharp = v$ and $H^\sharp = (i, F)$ then we require that $v \in \mathsf{reach}_{\mathcal{A}}(E_i, F)$ and that $\sigma^\flat$ and $H^\flat$ correspond. Formally this is defined as follows:

$$\gamma_{\mathcal{A}}(\epsilon, \epsilon) = \mathsf{tt} \quad \gamma_{\mathcal{A}}(v \cdot \sigma, \epsilon) = \mathsf{ff} \quad \gamma_{\mathcal{A}}(\epsilon, (i, F) \cdot H) = \mathsf{ff}$$

$$\gamma_{\mathcal{A}}(v \cdot \sigma, (i, F) \cdot H) = v \in \mathsf{reach}_{\mathcal{A}}(E_i, F) \wedge \gamma_{\mathcal{A}}(\sigma, H)$$

Notice that $\gamma_{\mathcal{A}}(v \cdot \sigma, \varnothing_{H,i})$ holds whenever $v \in E_i$ and $\sigma = \epsilon$, since $v \in E_i$ whenever $v \in \mathsf{reach}_{\mathcal{A}}(E_i, \epsilon)$.

We are now ready to define formally the generalised notion of satisfaction.

**Definition 3.4** *Given applet $\mathcal{A}$ and history stack $H$, we define* generalised satisfaction w.r.t. $H$ *by:*

$$\mathcal{A} \models_H \phi \Leftrightarrow \forall v, \sigma.(\gamma_{\mathcal{A}}(v \cdot \sigma, H) \Rightarrow (v, \sigma) \models_b \phi)$$

$$
\begin{array}{lll}
 & \mathcal{A} \models_{(i,F)\cdot H} [a\ \mathsf{call}\ b]\,\phi & \\
\Leftrightarrow & \forall v,\sigma.(v \in \mathsf{reach}_{\mathcal{A}}(E_i,F) \wedge \gamma_{\mathcal{A}}(\sigma,H) \Rightarrow (v,\sigma) \models_b [a\ \mathsf{call}\ b]\,\phi) & \{\text{Def. } 3.4, \gamma_{\mathcal{A}}\} \\
\Leftrightarrow & \forall v,\sigma.(v \in \mathsf{reach}_{\mathcal{A}}(E_i,F) \wedge \gamma_{\mathcal{A}}(\sigma,H) \Rightarrow & \{\text{Def. } \models_b\} \\
 & \qquad\qquad \forall v_1,v_2.(v \xrightarrow{b}_i v_1 \wedge v_2 \in E_b \Rightarrow (v_2, v_1\cdot\sigma) \models_b \phi)) & \\
\Leftrightarrow & \forall v,v_1,v_2,\sigma.(v \in \mathsf{reach}_{\mathcal{A}}(E_i,F) \wedge \gamma_{\mathcal{A}}(\sigma,H) \wedge v \xrightarrow{b}_i v_1 \wedge v_2 \in E_b \Rightarrow & \{\text{Logic}\} \\
 & \qquad\qquad (v_2, v_1\cdot\sigma) \models_b \phi) & \\
\Leftrightarrow & \forall v_1,v_2,\sigma.(v_1 \in \mathsf{reach}_{\mathcal{A}}(E_i,F\cdot b) \wedge \gamma_{\mathcal{A}}(\sigma,H) \wedge v_2 \in E_b \Rightarrow & \{\text{Def. } \mathsf{reach}_{\mathcal{A}}\} \\
 & \qquad\qquad (v_2, v_1\cdot\sigma) \models_b \phi) & \\
\Leftrightarrow & \forall v_1,v_2,\sigma.(v_2 \in E_b \wedge \gamma_{\mathcal{A}}(v_1\cdot\sigma,(i,F\cdot b)\cdot H) \Rightarrow (v_2,v_1\cdot\sigma) \models_b \phi) & \{\text{Def. } \gamma_{\mathcal{A}}\} \\
\Leftrightarrow & \forall v_1,v_2,\sigma.(v_2 \in \mathsf{reach}_{\mathcal{A}}(E_b,\epsilon) \wedge \gamma_{\mathcal{A}}(v_1\cdot\sigma,(i,F\cdot b)\cdot H) \Rightarrow (v_2,v_1\cdot\sigma) \models_b \phi) & \{\text{Def. } \mathsf{reach}_{\mathcal{A}}\} \\
\Leftrightarrow & \forall v_1,v_2,\sigma.(\gamma_{\mathcal{A}}(v_2\cdot v_1\cdot\sigma,(b,\epsilon)\cdot(i,F\cdot b)\cdot H) \Rightarrow (v_2,v_1\cdot\sigma) \models_b \phi) & \{\text{Def. } \gamma_{\mathcal{A}}\} \\
\Leftrightarrow & \mathcal{A} \models_{(b,\epsilon)\cdot(i,F\cdot b)\cdot H} \phi & \{\text{Def. } 3.4\} \\
\Leftrightarrow & \exists\sigma \in \pi_{(b,\epsilon)\cdot(i,F\cdot b)\cdot H}(\phi).\ \mathcal{A} \models_s \sigma & \{\text{Ind. hyp.}\} \\
\Leftrightarrow & \exists\sigma \in \pi_{(i,F)\cdot H}([a\ \mathsf{call}\ b]\,\phi).\ \mathcal{A} \models_s \sigma & \{\text{Def. } \pi\}
\end{array}
$$

Figure 3. Correctness proof for case $[a\ \mathsf{call}\ b]\phi$

The standard notion of satisfaction can be related to this relativised notion of satisfaction as follows.

**Proposition 3.5** $\mathcal{A} \models_b \phi \Leftrightarrow \forall m \in I^+.\ \mathcal{A} \models_{\varnothing_{H,m}} \phi$

We can now state the main theorem that allows us to show the correctness of the translation.

**Theorem 3.6** *Let $\mathcal{A}$ be an applet, $H$ be a history stack, and $\phi$ be a disjunction-free behavioural formula. Then:*

$$\mathcal{A} \models_H \phi \ \Leftrightarrow\ \exists\sigma \in \pi_H(\phi).\ \mathcal{A} \models_s \sigma$$

**Proof** This theorem has been formalised and proven correct with the PVS theorem prover [6] by using induction on the structure of $\phi$. The two main properties used in the proofs are the following:

- $(\forall v.v \in \mathsf{reach}_{\mathcal{A}}(E_i, L) \Rightarrow v \models \sigma) \Leftrightarrow \mathcal{A} \models [L]\,\sigma$
- $(\exists\sigma.\gamma_{\mathcal{A}}(\sigma, H)) \Leftrightarrow (\forall\phi \in \{i \Rightarrow [F]\,\mathsf{ff} \mid (i,F) \in H\}.\mathcal{A} \not\models \phi)$

The rest of the proof is a careful rewriting of definitions, case analysis and use of logic. Figure 3 shows the complete derivation for one of the most interesting cases, namely case $[a\ \mathsf{call}\ b]\,\phi$ when $i = a$. $\square$

Equivalence (1) is a straightforward corollary of Theorem 3.6 and Proposition 3.5.

## 4 Compositional verification for behavioural properties

The results in the preceding section justify the following compositional verification principle where $\phi$ is a behavioural formula of modal simulation logic.

$$\frac{\mathcal{A} \models_b \phi \qquad\qquad \{\theta_{I_{\mathcal{A}}}(\sigma) \uplus \mathcal{B} \models_b \chi\}_{\sigma\in\Pi(\phi)_{I_{\mathcal{A}}}}}{\mathcal{A} \uplus \mathcal{B} \models_b \chi}\,\mathcal{A}\colon I_{\mathcal{A}}$$

Notice that instead of showing $\mathcal{A} \models_b \phi$ it suffices to show $\mathcal{A}^\bullet \models_s \sigma$ for some $\sigma \in \Pi(\phi)_{I_\mathcal{A}}$.

Soundness of this rule follows from equation (1) and the soundness of the compositional verification principle for structural formulae (structure). Since the compositional verification principle is not complete if we restrict ourselves to modal simulation logic, we do not get completeness. However, since the characterisation of behavioural formulae described in this paper is exact, provided the formula does not contain disjunctions, once it has been extended with greatest fixed-points, we can exploit the fact that satisfaction and simulation characterise each other, to also get a complete compositional verification principle for behavioural formulae.

## 5 Conclusion

This paper describes a translation from behavioural to structural properties. The translation is defined for so-called modal simulation logic, which corresponds to modal logic with box modalities only. In earlier work, we defined a compositional verification principle where local properties have to be structural properties. Based on this principle, we developed a compositional verification method, provided machine support by means of a tool set, and evaluated the practical applicability of the method on an industrial case study. By having a translation from behavioural to structural properties, we extend the compositional verification principle to behavioural properties. The translation proceeds by symbolic execution of the behavioural formula: each modality in this formula gives rise to a constraint on the structures that satisfy the behavioural formula. It has been implemented in Ocaml and included in the tool set.

The logic fragment for which we have defined the translation is quite restricted, but the work described in this paper forms part of a bigger project, where we are extending the translation to greatest fixed-points. This extension requires that the translation is recast into a tableau construction, unfolding the greatest fixed-point operator until some global discharge condition holds, meaning that the fixed-point has been sufficiently unfolded to capture all structural properties. However, we feel that the translation of the modal logic fragment in itself merits a detailed description, because this is the point in the translation where the interplay between behaviour and structure is the most prominent.

In a different line of work, we are currently investigating whether we can extend the translation to also take diamond modalities into account. However, since diamond modalities cannot be characterised by simulation of standard models, this would not contribute further to our work on compositional verification.

## References

[1] Burkart, O., D. Caucal, F. Moller and B. Steffen, *Verification on infinite structures*, in: J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, North Holland, 2000 pp. 545–623.

[2] Grumberg, O. and D. Long, *Model checking and modular verification*, ACM TOPLAS **16(3)** (1994), pp. 843–871.

[3] Huisman, M., D. Gurov, C. Sprenger and G. Chugunov, *Checking absence of illicit applet interactions: a case study*, in: M. Wermelinger and T. Margaria, editors, *Fundamental Approaches to Software Engineering, FASE'04*, number 2984 in LNCS (2004), pp. 84–98.

[4] Jensen, T., D. L. Métayer and T. Thorn, *Verification of control flow based security policies*, in: *IEEE Symposium on Research in Security and Privacy* (1999), pp. 89–103.

[5] Kupferman, O. and M. Vardi, *An automata-theoretic approach to modular model checking*, ACM TOPLAS **22** (2000), pp. 87–128.

[6] Owre, S., J. Rushby, N. Shankar and F. v. Henke, *Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS*, IEEE Transactions on Software Engineering **21** (1995), pp. 107–125.

[7] Roever, W.-P. d., F. d. Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel and J. Zwiers, "Concurrency Verification: Introduction to Compositional and Noncompositional Methods," Number 54 in Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge, UK, 2001.

[8] Sprenger, C., D. Gurov and M. Huisman, *Simulation logic, applets and compositional verification*, Technical Report RR-4890, INRIA (2003).

[9] Sprenger, C., D. Gurov and M. Huisman, *Compositional verification for secure loading of smart card applets*, in: *Formal Methods and Models for Co-Design (Memocode 2004)* (2004), pp. 211–222.