Compositional Verification for Secure Loading of Smart Card Applets[*]

Christoph Sprenger[†]
Swiss Federal Institute of Technology, Zurich, Switzerland
christoph.sprenger@inf.ethz.ch

Dilian Gurov
Royal Institute of Technology
Kista, Sweden
dilian@imit.kth.se

Marieke Huisman
INRIA Sophia Antipolis
Sophia Antipolis, France
marieke.huisman@inria.fr

Abstract

We present an algorithmic compositional verification method for smart card applets and control flow based safety properties expressed in a modal logic with simultaneous greatest fixed points. Our method builds on a technique proposed by Grumberg and Long who use maximal models to reduce compositional verification of finite-state parallel processes to standard model checking. We adapt this technique to applets, a class of infinite-state sequential processes. This requires a refinement of the method, since for a given applet interface and behavioural formula a maximal applet does not always exist. We therefore propose a two-level approach, where local assumptions restrict the control flow structure *of applets, while the global guarantee restricts the control flow* behaviour *of the system. We present a novel maximal model construction for our logic and then adapt it to applets. By separating the tasks of verifying global and local properties our method supports secure post-issuance loading of applets onto a smart card.*

1 Introduction

With the emergence of small secure devices, such as open platform smart cards, it becomes important to set criteria to decide whether an application can be accepted on a device. Since such devices are typically used to store privacy-sensitive data, for the acceptance of this new technology it is important that potential users have full trust in the protection of their data.

For the new generation of smart cards, an interesting possibility is to have *post-issuance* loading of applications (applets). This means that once the card is issued and given to the user, new applets can be installed on the card without the mediation of the card provider. Post-issuance loading opens many possibilities for new and powerful applications, but so far has not found its way to industrial practice, mainly because of security concerns. The method proposed here is a first step toward a framework for secure post-issuance loading of smart card applets. Automatic checks are needed to ensure that new applets can be trusted. These checks can involve for example type safety, memory consumption, and illicit data or control flow.

In this paper we focus on the last category of properties. More precisely, we study sequential (single-threaded) applets and propose a specification and verification method for safety properties of inter-procedural control flow, *i.e.* properties describing safe sequences of method invocations. Since applets can be loaded on a card post-issuance, their implementation might not be available at the time global properties are verified. Therefore we propose a *compositional* verification method, according to the following proof principle:

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

This reduces the problem of showing that the composition of applets A and B satisfies ψ to three tasks:

1. decompose the global property ψ by finding a local property ϕ of applet A,
2. prove correctness of the decomposition, *i.e.*, verify that, for *any* applet X satisfying ϕ, X composed with B satisfies ψ (second premise), and
3. verify that A satisfies the local property ϕ (first premise), when its implementation becomes available for post-issuance loading.

The compositionality of the method supports different scenarios for secure post-issuance loading of applets, where the tasks above can potentially be delegated to different authorities. In the first scenario, the card issuer specifies both the global and local properties and verifies – using the techniques described in this paper – that the decomposition is

[*]This work was partially supported by the EU VerifiCard project IST-2000-26328. This work was also partially supported by the Zurich Information Security Center. It represents the views of the authors.

[†]Research done while at INRIA Sophia Antipolis, partially supported by ERCIM Fellowship No. 2002-10.

correct, meaning that the local specification is sufficient to establish the global specification. Each time an applet is loaded post-issuance, an algorithm provided by the card issuer checks whether the applet implementation satisfies the required specification. An alternative scenario is that the card issuer only provides the global specification (and local specifications for its own applets), and leaves it to the applet provider to come up with an appropriate local specification for each post-issuance loaded applet. As in the previous scenario, an algorithm provided by the card issuer checks the applet against the local specification upon loading, but now also the property decomposition needs to be verified at loading time, potentially on-card.

Task (1) above is a manual one and requires some insight into the system, while the other two can be automated in our approach. We concentrate here on task (2); for task (3) standard algorithmic techniques exist. In earlier work [3], we explored deductive verification of correctness of decompositions based on a proof system. However, the generality of this approach requires considerable time and expertise from the user. Hence, an algorithmic method such as the one presented here is preferable in many situations.

The approach that we take is inspired by the pioneering work on automatic modular verification by Grumberg and Long [10]. To check whether $X : \phi \vdash X \otimes B : \psi$ holds we replace X by a *maximal model* $\theta(\phi)$ and then verify $\vdash \theta(\phi) \otimes B : \psi$ algorithmically. The maximal model $\theta(\phi)$ represents all models satisfying ϕ in the sense that it simulates exactly those models and thus satisfies precisely the properties enjoyed by all these models. For this technique to be sound it is required that $\otimes$ preserves simulation and that logical properties are preserved by simulation.

Contributions Most existing work on compositional model checking focuses on the verification of parallel compositions of finite-state processes. Our main contribution is the adaptation of this technique to infinite-state sequential programs, more precisely to applets. We model applets by a collection of method control flow graphs equipped with an interface of provided and required methods. Applet composition essentially forms the disjoint union of the respective collections of method graphs and allows the composed applets to communicate by method invocation. Our applet models induce a subclass of pushdown processes, with potentially infinite-state behaviour (*cf.* [8]).

We are interested in safety properties, which can be adequately expressed in our *simulation logic*, a modal logic with simultaneous greatest fixed points. This logic is equivalent to the modal μ-calculus [13] without diamond modalities and least fixed points. We establish a logical characterisation of simulation and, vice versa, a behavioural characterisation of logical satisfaction in terms of maximal models. In particular, we present a novel maximal model construction, consisting of a step-wise transformation of the formula into a semantically equivalent normal form, which is isomorphic to a maximal model for the formula.

When tailoring the maximal model technique to applets, we require that the maximal model for a given property is itself an applet. This ensures that, if the verification of $\vdash \theta(\phi) \otimes B : \psi$ fails, there is indeed an *applet* among the set of models F such that F satisfies ϕ but $F \otimes B$ does not satisfy ψ. In this case we can try to strengthen ϕ and iterate the process. However, for a given property ϕ and interface I, a maximal applet in general does not exist due to the possibility of re-entrance of methods, even if there are applets with interface I satisfying ϕ. To overcome this problem, we propose a two-level solution, distinguishing between structural and behavioural levels. We instantiate our logic and simulation at each level. Local specifications are *structural* properties, restricting the finite control flow *structure* of applets, while global specifications are *behavioural* properties, restricting their potentially infinite control flow *behaviour*. We define the maximal applet for a given interface I and structural property ϕ by $\theta_I(\phi) = \theta(\phi_I \wedge \phi)$, where ϕ_I is a structural formula capturing the basic properties of all applets with this interface, thus ensuring that the resulting maximal model is indeed an applet structure. This maximal applet is composed with applet B and the result can be verified against ψ using a model checker for pushdown processes such as Alfred [17] (based on an algorithm by Bouajjani *et al.* [5]). Combining our characterisation results with results linking the structural and behavioural levels, we establish the soundness and completeness of our method.

Some additional results, examples and full proofs can be found in a technical report accompanying the present paper [19]. In a companion paper [12] we present a tool set for our framework and show its practical usability by applying it on an industrial electronic purse case study.

Related Work There is a wealth of methods for compositional verification of concurrent programs, most notably assumption/commitment based reasoning about processes with synchronous message passing, and the rely/guarantee method for shared-variable concurrency. A systematic overview of these and related proof methods, some of which have been adapted to support algorithmic verification is given by de Roever *et al.* [9]. However, these techniques do not address programs with recursive procedures.

As an example of an approach to compositional verification based on theorem proving we mention the work by Prensa [18]. She formalises the rely/guarantee method for parallel programs with shared variables (but without procedures) in Isabelle/HOL. The method is based on an extension of Hoare logic to parallel programs and does as such not cover temporal properties.

The original maximal model technique by Grumberg and

Long [10] was designed for the universal fragment of CTL and later extended to CTL* by Kupferman and Vardi [14]. These works study synchronous parallel compositions of sequential processes under fairness assumptions. Since we are interested in safety properties of sequential programs, we do not need to add fairness to our models. Our transformational approach to the maximal model construction avoids some unnecessary exponential blowups appearing in their global constructions.

Laster and Grumberg [16] present a compositional method for sequential programs written in a high-level While language (without procedures). Their technique partitions the program text into a sequence of sequentially composed subprograms, which can be model checked individually using assumptions on the properties holding at the cut points.

Characterisation results connecting logics and behavioural preorders similar to ours are described by Larsen and Boudol [7] (see also [15]), who construct maximal modal transition systems w.r.t. the refinement preorder for Hennessy-Milner logic [11]. Since this logic does not include fixed points, the constructed models are essentially finite forests. Bouajjani *et al.* [6] define maximal models for a co-recursive modal logic to express safety properties. Their logic has an expressive power similar to ours, but is somewhat less standard as it includes a connective corresponding to non-deterministic choice.

The method of partial model checking introduced by Andersen [1] is based on a reduction procedure that removes the top-level operator from a process algebra term and computes a new property for the reduced term. To verify that the product $P \times Q$ of two processes has some property ϕ, the reduction "divides" the property ϕ by Q to yield ϕ/Q, which can be effectively computed only if Q is finite.

Structure Section 2 defines models, simulation and logic, and describes a procedure to construct maximal models. Section 3 instantiates this to applets (both at structural and behavioural level) and derives the compositional proof principle. Section 4 illustrates our approach with an example. Finally, Section 5 contains some concluding remarks.

2 Simulation and Logic

This section develops several general results about simulation and its relation to logic. After the introduction of specifications and simulations between them, we present simulation logic, a subset of Hennessy-Milner logic [11] extended with simultaneous greatest fixed points. By defining maps between specifications and logical formulae, we establish a logical characterisation of simulation in terms of simulation logic and, vice versa, a behavioural characterisation of logical satisfaction. In particular, the behavioural characterisation of satisfaction involves the construction of a model from a formula, which is *maximal* in the sense that it simulates all models satisfying the formula. This will serve as the basis for our compositional verification method for applets explained in the next section.

For the rest of this section we fix two arbitrary finite sets of *labels* L and *atomic propositions* A, parameterising the models and logic introduced next.

2.1 Specifications and Simulation

First we define models, specifications and simulation. These notions are standard up to some minor variations.

Definition 2.1. ***(Model and specification)*** A *model* is a structure $\mathcal{M} = (S, L, \rightarrow, A, \lambda)$, where S is a set of states, $\rightarrow \subseteq S \times L \times S$ is a transition relation and $\lambda \colon S \rightarrow \mathcal{P}(A)$ is a valuation assigning to each state a set of atomic propositions. A *specification* $\mathcal{S}$ is a pair $(\mathcal{M}, E)$, where $\mathcal{M}$ is a model and $E \subseteq S$ is a set of entry states.

The reachable part of a specification $\mathcal{S} = (\mathcal{M}, E)$ is defined by $\mathcal{R}(\mathcal{S}) = (\mathcal{M}', E)$, where $\mathcal{M}'$ is obtained from $\mathcal{M}$ by deleting all states and transitions not reachable from any entry state in E.

Example 2.2. Figure 1 shows the graphical representation of a specification (where $s_1(p, q)$ means $\lambda(s_1) = \{p, q\}$).

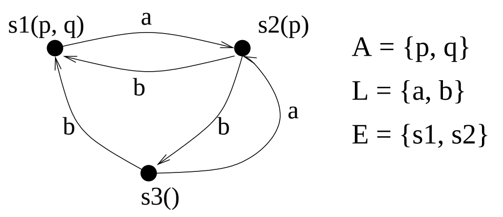

Figure 1. Example specification $\mathcal{S}$

Definition 2.3. ***(Simulation)*** A *simulation* is a binary relation R on S such that whenever $(s, t) \in R$ then $\lambda(s) = \lambda(t)$, and whenever $s \xrightarrow{a} s'$ then there is some $t' \in S$ such that $t \xrightarrow{a} t'$ and $(s', t') \in R$. We say that t *simulates* s, written $s \leq t$, if there is a simulation R such that $(s, t) \in R$.

Simulation on two models $\mathcal{M}_1$ and $\mathcal{M}_2$ is defined as simulation on their disjoint union $\mathcal{M}_1 \uplus \mathcal{M}_2$. The transitions of $\mathcal{M}_1 \uplus \mathcal{M}_2$ are defined by $in_i(s) \xrightarrow{a} in_i(s')$ if $s \xrightarrow{a} s'$ in $\mathcal{M}_i$ and its valuation by $\lambda(in_i(s)) = \lambda_i(s)$, where in_i injects S_i into $S_1 \uplus S_2$. Simulation is extended to specifications $(\mathcal{M}_1, E_1)$ and $(\mathcal{M}_2, E_2)$ by defining $(\mathcal{M}_1, E_1) \leq (\mathcal{M}_2, E_2)$ if there is a simulation R on $\mathcal{M}_1 \uplus \mathcal{M}_2$ such that for each $s \in E_1$ there is some $t \in E_2$ with $(in_1(s), in_2(t)) \in R$.

2.2 Simulation Logic

We define simulation logic in two steps: first we define a basic logic and then we add recursion by using modal equation systems. Let $\mathcal{V}$ be a countably infinite set of propositional variables. *Basic simulation logic* is a variant of Hennessy-Milner logic without diamond modalities:

$$\phi ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg p \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid [a]\,\phi$$

where $p \in A$, $a \in L$ and $X \in \mathcal{V}$. The interpretation $\|\phi\|\rho$ of a basic formula ϕ is defined with respect to a model $\mathcal{M}$ and an environment ρ interpreting the propositional variables. The definition is standard (*cf.* [20]), in particular, for the box modality we have $s \in \|[a]\,\phi\|\rho$ if and only if for all $t \in S$ such that $s \xrightarrow{a} t$ we have $t \in \|\phi\|\rho$. Formulae like p or $\neg p$ are called *literals*. We use n-ary versions of conjunction and disjunction, setting $\bigvee \varnothing = \mathsf{ff}$ (false) and $\bigwedge \varnothing = \mathsf{tt}$ (true). As usual, for $K \subseteq L$, we write $[K]\,\phi$ for $\bigwedge_{a\in K} [a]\,\phi$ and $[-]\,\phi$ for $[L]\,\phi$.

To make the logic expressive enough to characterise all finite models, we follow Larsen [15] and add recursion to basic simulation logic by introducing modal equation systems. A *modal equation system* Σ is a finite set of defining equations of the shape $X = \phi_X$, where X is a propositional variable and ϕ_X is a formula of basic simulation logic. The defined variables X are pairwise distinct and bound in Σ, while all other variables are free. For a simpler presentation, we restrict our attention here to closed equation systems without free variables.

Since our equations systems are closed, it is sufficient to work with environments $\rho : \mathsf{bv}(\Sigma) \to \mathcal{P}(S)$ mapping the bound variables of Σ to sets of states. The equations in Σ induce a map $\Psi_\Sigma : \mathcal{P}(S)^{\mathsf{bv}(\Sigma)} \to \mathcal{P}(S)^{\mathsf{bv}(\Sigma)}$ on such environments ρ defined by $\Psi_\Sigma(\rho)(X) = \|\phi_X\|\rho$. A solution of Σ is an environment ρ such that all equations in Σ are satisfied, that is, $\Psi_\Sigma(\rho) = \rho$. Environments are ordered by point-wise inclusion. The *semantics* of a modal equation system Σ w.r.t. a model $\mathcal{M}$, denoted $\|\Sigma\|$, is its greatest solution. By the Knaster-Tarski fixed point theorem [21] a greatest solution always exists, since Ψ_Σ is monotone.

Definition 2.4. ***(Simulation Logic)*** A (closed) formula of *simulation logic* has the shape $\phi[\Sigma]$, where ϕ is a formula of basic simulation logic and Σ is a (closed) modal equation system such that all variables occurring in ϕ are bound in Σ. The *semantics of* $\phi[\Sigma]$ with respect to model $\mathcal{M}$ is defined by $\|\phi[\Sigma]\| = \|\phi\|\|\Sigma\|$. We say a specification $(\mathcal{M}, E)$ satisfies $\phi[\Sigma]$, written $(\mathcal{M}, E) \models \phi[\Sigma]$, if $E \subseteq \|\phi[\Sigma]\|$.

Example 2.5. Consider the formula $\phi = (X \vee Y)[\Sigma]$, where

$$\Sigma = \left[\begin{array}{lcl} X & = & [a]\,Y \wedge [b]\,X \wedge p \\ Y & = & [a]\,(X \wedge Y) \wedge \neg q \end{array} \right].$$

We would like to determine the semantics of this formula with respect to the specification $\mathcal{S}$ in Figure 1. The greatest fixed point $\|\Sigma\|$ of Σ with respect to $\mathcal{S}$ can be computed in the standard way by iteration of Ψ_Σ starting with $\rho_0 = \{X \mapsto S, Y \mapsto S\}$, where $S = \{s_1, s_2, s_3\}$. This yields $\|\Sigma\| = \{X \mapsto \{s_1\}, Y \mapsto \{s_2\}\}$. So, $E = \{s_1, s_2\} = \|X \vee Y\|\|\Sigma\|$, and hence the specification $\mathcal{S}$ satisfies ϕ.

Henceforth, we often omit the equation system Σ from $\phi[\Sigma]$ if no confusion can arise. We say that ϕ_1 is a logical consequence of ϕ_0, written $\phi_0 \sqsubseteq \phi_1$, if for all specifications $\mathcal{S}$, $\mathcal{S} \models \phi_0$ implies $\mathcal{S} \models \phi_1$. The formula ϕ_0 is logically equivalent to ϕ_1, written $\phi_0 \equiv \phi_1$, if $\phi_0 \sqsubseteq \phi_1$ and $\phi_1 \sqsubseteq \phi_0$.

Simulation logic is equally expressive as the modal μ-calculus [13] without diamond modalities and least fixed points. The translation from this fragment of the modal μ-calculus to simulation logic is straightforward and replaces each fixed point by an equation. As an example, the formula $\nu X.p_1 \wedge (\nu Y.X \wedge [a]\,(p_2 \vee Y))$ is translated into the equivalent formula $X[X = p_1 \wedge Y, Y = X \wedge [a]\,(p_2 \vee Y)]$ of simulation logic. The translation in the other direction is based on Bekič's principle (*cf.* [2]), which expresses a fixed point in a product lattice in terms of a vector of component-wise fixed points.

2.3 Representation Results

Next, we relate simulation logic to simulation by defining two functions, χ and θ. The map χ translates each *finite* specification into a formula, while θ translates formulae into (finite) specifications. The latter map is first defined on formulae in so-called simulation normal form (SNF) and then extended to all formulae by showing that any formula can be transformed into an equivalent one in SNF. We show that χ logically characterises simulation and θ behaviourally characterises logical satisfaction. These two maps form a Galois connection between finite specifications ordered by simulation and formulae ordered by logical consequence. Similar results for somewhat different settings appear in [7, 15, 6]. In this paper, we present a novel procedure θ to construct maximal models. While its complexity is exponential in the worst case, it avoids by its transformational nature some unnecessary exponential blowups occurring in the respective procedures for the universal fragments of CTL [10] and CTL* [14].

2.3.1 Characteristic Formulae

First we define the mapping from finite specifications to formulae. A finite specification $(\mathcal{M}, E)$ is translated into its *characteristic formula* $\chi(\mathcal{M}, E) = \phi_E[\Sigma_\mathcal{M}]$, where $\phi_E = \bigvee_{s\in E} X_s$ and $\Sigma_\mathcal{M}$ defines X_s for each $s \in S$ by

$$X_s = \bigwedge_{a\in L} [a] \bigvee_{s \xrightarrow{a} t} X_t \;\wedge\; \bigwedge_{p\in\lambda(s)} p \;\wedge\; \bigwedge_{q\in A-\lambda(s)} \neg q$$

Recall that $\bigvee \varnothing = \mathsf{ff}$ (false) and $\bigwedge \varnothing = \mathtt{tt}$ (true).

Example 2.6. Consider the specification $\mathcal{S}$ displayed in Figure 1. Its characteristic formula is $\chi(\mathcal{S}) = X_{s_1} \vee X_{s_2}[\Sigma]$, where

$$\Sigma = \begin{bmatrix} X_{s_1} & = & [a]X_{s_2} \wedge [b]\,\mathsf{ff} \wedge\ p \wedge q \\ X_{s_2} & = & [a]\,\mathsf{ff} \wedge [b](X_{s_1} \vee X_{s_3}) \wedge p \wedge \neg q \\ X_{s_3} & = & [a]X_{s_2} \wedge [b]X_{s_1} \wedge \neg p \wedge \neg q \end{bmatrix}.$$

We can prove that if specification $\mathcal{S}_1$ is simulated by the finite specification $\mathcal{S}_2$, this is equivalent to saying that $\mathcal{S}_1$ satisfies the characteristic formula of $\mathcal{S}_2$. This is a variation of an earlier result by Larsen [15].

Theorem 2.7. *Let $\mathcal{S}_1$, $\mathcal{S}_2$ be specifications and suppose $\mathcal{S}_2$ is finite. Then $\mathcal{S}_1 \leq \mathcal{S}_2$ if and only if $\mathcal{S}_1 \models \chi(\mathcal{S}_2)$.*

Note that using infinite equation systems this theorem generalises to finitely branching $\mathcal{S}_2$.

2.3.2 Maximal Models

The next step is to define the inverse mapping. Not all formulae correspond directly to a specification, but those in simulation normal form do.

Definition 2.8. ***(Simulation normal form)*** A formula $\phi[\Sigma]$ of simulation logic is in *simulation normal form (SNF)* if ϕ has the form $\bigvee \mathcal{X}$ for some finite set $\mathcal{X} \subseteq \mathsf{bv}(\Sigma)$ and all equations in Σ are in the following *state normal form*

$$X = \bigwedge_{a \in L} [a] \bigvee \mathcal{Y}_{X,a} \ \wedge \bigwedge_{p \in B_X} p \ \wedge \bigwedge_{q \in A - B_X} \neg q$$

where each $\mathcal{Y}_{X,a} \subseteq \mathsf{bv}(\Sigma)$ is a finite set of variables and $B_X \subseteq A$ is a set of atomic propositions.

Notice that any characteristic formula $\chi(\mathcal{S})$ is in SNF. From a formula $(\bigvee \mathcal{X})[\Sigma]$ in SNF we derive the specification $\theta((\bigvee \mathcal{X})[\Sigma]) = ((S, L, \rightarrow, A, \lambda), E)$ where $S = \mathsf{bv}(\Sigma)$, $E = \mathcal{X}$ and, for each $X \in \mathsf{bv}(\Sigma)$, the equation for X induces the transitions $\{X \xrightarrow{a} Y \mid Y \in \mathcal{Y}_{X,a}\}$ and the valuation $\lambda(X) = B_X$.

Lemma 2.9. *χ and θ are each others inverse up to equivalence, that is,*

1. *$\theta(\chi(\mathcal{S})) \cong \mathcal{S}$ ($\cong$ is isomorphism[1]) for finite $\mathcal{S}$, and*
2. *$\chi(\theta(\phi)) \equiv_\alpha \phi$ ($\equiv_\alpha$ is α-convertibility) for ϕ in SNF.*

Theorem 2.10. *For ϕ in SNF, we have $\mathcal{S} \leq \theta(\phi)$ if and only if $\mathcal{S} \models \phi$.*

[1]Here, isomorphism means a label-preserving bijection between states and transitions.

Transformation to SNF We now present one of the main results of our paper, a step-wise transformation of any simulation logic formula into a logically equivalent formula in SNF. Before describing the transformation in detail, we introduce some auxiliary notions. First, we use a slightly non-standard variant of disjunctive normal form: we say that a formula ϕ of basic simulation logic is in *disjunctive normal form* (*DNF*) if it is a disjunction of conjunctions of box formulae and literals, *i.e.*, it has the shape $\phi = \bigvee_i (\bigwedge_j [a_{ij}]\, \psi_{ij} \wedge \bigwedge \mathcal{L}_i)$ where $\mathcal{L}_i$ are sets of literals and ψ_{ij} arbitrary formulae in basic simulation logic. Furthermore, the *conjunctive decomposition* $c(\psi)$ of a formula ψ into its conjuncts is given by $c(\psi) = \{\psi_1, \ldots, \psi_m\}$ such that no ψ_i is a conjunction and $\psi = \bigwedge_i \psi_i$ (modulo associativity and commutativity). Note that $c(\mathtt{tt}) = \varnothing$.

We call an occurrence of a subformula *top-level* if it is not under the scope of a box operator. We say that Y is *unguarded* in ϕ_X, written $X \triangleright Y$, if there is a top-level occurrence of Y in ϕ_X. A modal equation system Σ (or formula $\phi[\Sigma]$) is *weakly guarded* if the relation $\triangleright$ is acyclic and *strongly guarded* if $\triangleright$ is empty.

Example 2.11. Consider the modal equation system

$$\Sigma = \begin{bmatrix} X & = & [a]\, X \vee (q \wedge Y) \\ Y & = & [b]\, (X \wedge [a]\, Y) \wedge p \end{bmatrix}$$

Variable X is guarded in ϕ_X (the only occurrence of X is under the scope of a box operator), but Y is not (it occurs on the top-level). Both X and Y are guarded in ϕ_Y. Hence, $\triangleright\ = \{(X, Y)\}$ being acyclic but not empty, Σ is weakly guarded but not strongly guarded.

Any weakly guarded formula can be transformed into a strongly guarded one by repeatedly rewriting each unguarded occurrence of a variable by its defining equation. Moreover, using a result in [22] we can also show:

Lemma 2.12. (Weak Guardedness) *Any formula of simulation logic can be transformed into an equivalent weakly guarded one.*

After these auxiliary definitions, we are ready to present the transformation. It consists of three phases:

Phase I transforms each equation into a disjunction of formulae in state normal form, where only single variables appear under modalities,

Phase II splits top-level disjunctions in each equation into a set of new equations, one for each disjunct, yielding an equation system in state normal form, and

Phase III is an optimisation phase removing unreachable and redundant equations.

The transformation into SNF uses a partial function h mapping sets of formulae to variables. Its termination relies on this map that avoids the repeated introduction of new equations for the same formula. If h maps a set of formulae Ψ to variable X, this means that an equation $X = \bigwedge \Psi$ (such that $c(\bigwedge \Psi) = \Psi$) has been introduced earlier and that variable X should be used instead of introducing any further equation for $\bigwedge \Psi$. This bookkeeping is essential for the termination of the transformation.

Before going into the details, let us illustrate the basic ideas on a simple example. A more elaborate example appears in Section 4.

Example 2.13. Let $\phi = [b]\,\mathsf{ff} \wedge p$ be interpreted as a formula over $L = \{a, b\}$ and $A = \{p\}$. This formula holds for specifications, where each initial state satisfies p and has no outgoing b transition. We first translate ϕ to $(\bigvee \mathcal{X}_0)[\Sigma_0]$ with $\mathcal{X}_0 = \{X\}$ and $\Sigma_0 = \{X = [b]\,\mathsf{ff} \wedge p\}$. In the following, the numbers in parentheses refer to the transformation steps detailed below.

The equation for X is already strongly guarded (I.1) and in DNF (I.2). Next, we add the missing box $[a]$ using the equivalence $\mathsf{tt} \equiv [a]\,\mathsf{tt}$ (I.3), yielding $X = [a]\,\mathsf{tt} \wedge [b]\,\mathsf{ff} \wedge p$. In the next step (I.4), we introduce new variables for the formulae under the boxes: $Y = \mathsf{tt}$ and $Z = \mathsf{ff}$. This is recorded in h with two new entries: $(\varnothing, Y)$ (since $\mathsf{tt} = \bigwedge \varnothing$) and $(\{\mathsf{ff}\}, Z)$. The equation for X becomes

$$X = [a]\,Y \wedge [b]\,Z \wedge p$$

which is already in state normal form. We proceed with $Y = \mathsf{tt}$. Again, the first step with an effect adds the missing boxes (I.3), producing $Y = [a]\,\mathsf{tt} \wedge [b]\,\mathsf{tt}$. Next, since $c(\mathsf{tt}) = \varnothing$ and $h(\varnothing) = Y$, we know that Y stands for tt, so we replace the subformulae tt under the boxes by Y, yielding $Y = [a]\,Y \wedge [b]\,Y$. To get a disjunction of state normal forms, we add the missing literals in positive and negative form, yielding

$$Y = ([a]\,Y \wedge [b]\,Y \wedge p) \vee ([a]\,Y \wedge [b]\,Y \wedge \neg p).$$

The third equation $Z = \mathsf{ff}$ $(= \bigvee \varnothing)$ is already a (trivial) disjunction of state normal forms. Note that $\mathcal{X}$ remains unchanged in Phase I. Next, Phase II splits each top-level disjunction into a set of new equations and substitutes the disjunction of new variables for the original variable. Concretely, all occurrences of Y are replaced by $Y_1 \vee Y_2$ and $Z = \mathsf{ff}$ $(= \bigvee \varnothing)$ is substituted back into ϕ_X, yielding

$$\Sigma = \left[\begin{array}{lcl} X & = & [a]\,(Y_1 \vee Y_2) \wedge [b]\,\mathsf{ff} \wedge p \\ Y_1 & = & [a]\,(Y_1 \vee Y_2) \wedge [b]\,(Y_1 \vee Y_2) \wedge p \\ Y_2 & = & [a]\,(Y_1 \vee Y_2) \wedge [b]\,(Y_1 \vee Y_2) \wedge \neg p \end{array}\right]$$

Since X is not split into several equations, $\mathcal{X} = \{X\}$ remains unchanged. Phase III is the identity transformation in this example as there are no unreachable or duplicate equations. Thus, the final result is $X[\Sigma]$, which is in simulation normal form. The derived maximal model $\theta(X[\Sigma])$ is displayed in Figure 2.

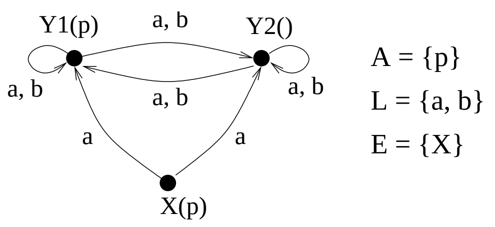

Figure 2. Maximal model for $\phi = [b]\,\mathsf{ff} \wedge p$

Now we describe the actual transformation in detail. We assume w.l.o.g. that the initial formula has the shape $X_0[\Sigma_0]$, where Σ_0 is weakly guarded (by Lemma 2.12). We initialise $\mathcal{X} = \{X_0\}$, $\Sigma = \Sigma_0$ and $h = \varnothing$. Here are the three phases of the transformation in detail.

Phase I (Disjunction of state normal forms) This phase transforms each equation into a disjunction of formulae in state normal form. Its steps are applied once to each equation including the new ones introduced in step I.4 below.

1. (Strong guardedness) Make equation strongly guarded by repeatedly rewriting unguarded occurrences of variables using the original system Σ_0.

2. (DNF) Put equation into disjunctive normal form and remove inconsistent disjuncts (those where ff or both p and $\neg p$ appear).

3. (Box grouping and completion) Group boxes together using $[a]\,\phi_1 \wedge [a]\,\phi_2 \equiv [a]\,(\phi_1 \wedge \phi_2)$ and add missing boxes to each disjunct using $\mathsf{tt} \equiv [a]\,\mathsf{tt}$ such that there is a box formula for each $a \in L$. Resulting equation shape is

$$X = \bigvee_i \big(\bigwedge_{a \in L} [a]\,\psi_{ia} \wedge \bigwedge \mathcal{L}_i \big)$$

4. (Modal depth reduction) Apply the following to each top-level box subformula $[a]\,\psi_{ia}$ where ψ_{ia} is not a variable. If $(c(\psi_{ia}), Y) \in h$ for some variable Y then replace $[a]\,\psi_{ia}$ by $[a]\,Y$; otherwise, choose a fresh variable $Z \notin \mathsf{bv}(\Sigma)$, add the new equation $Z = \psi_{ia}$ to Σ, replace $[a]\,\psi_{ia}$ by $[a]\,Z$ and extend h to $h \cup \{(c(\psi_{ia}), Z)\}$. Equation shape is then

$$X = \bigvee_i \big(\bigwedge_{a \in L} [a]\,Z_{ia} \wedge \bigwedge \mathcal{L}_i \big)$$

5. (Literal completion) Replace equation $X = \phi$ by $X = \phi \wedge \bigwedge_{p\in A}(p \vee \neg p)$, then repeat step (2) to put equation back into DNF. Equation shape is (for some $B \subseteq A$)

$$X = \bigvee_i \,(\bigwedge_{a\in L} [a]\, Z_{ia} \wedge \bigwedge_{p\in B} p \wedge \bigwedge_{q\in A-B} \neg q) \qquad (1)$$

Phase II (Push disjunctions inside) This phase eliminates the top-level disjunctions by introducing a new equation for each disjunct, thus pushing these disjunctions under box modalities. It is applied once to each equation in Σ.

1. remove an equation of shape $X = \bigvee_{i=1}^{n} \phi_i$ with $n \neq 1$ from Σ,
2. add a new equation $X_i = \phi_i$ for each non-variable disjunct ϕ_i and substitute $\bigvee_{i=1}^{n} X_i$ for X in all equations of Σ (where each X_i is either equal to ϕ_i or X_i is the fresh variable introduced for ϕ_i),
3. if $X \in \mathcal{X}$ then replace $\mathcal{X}$ by $(\mathcal{X} - \{X\}) \cup \{X_1, \ldots, X_n\}$.

The resulting equation is in state normal form.

Phase III (Optimisation) This optimisation phase iteratively removes unreachable and duplicate equations.

1. Remove equations $Z = \psi$ from Σ in case Z can not be reached from any variable in $\mathcal{X}$ via variable dependencies (X depends on Y if Y occurs in ϕ_X)
2. If there are equations $Z_1 = \psi_1$ and $Z_2 = \psi_2$ in Σ such that $\psi_1[Z_1/Z_2] = \psi_2[Z_1/Z_2]$, then remove $Z_2 = \psi_2$ from Σ and substitute Z_1 for Z_2 in the remaining equations as well as in $\mathcal{X}$.

Theorem 2.14. *The algorithm above terminates and transforms any formula ϕ of simulation logic into an equivalent formula $\mathsf{snf}(\phi)$ in simulation normal form.*

Proof. (Sketch; full proof in [19]) Let $\mathcal{X}_i$, Σ_i and h_i denote the values of $\mathcal{X}$, Σ and h after i transformation steps. We concentrate in this sketch on phase I, which preserves the following two invariants:

J1. for all $Y \in \mathsf{bv}(\Sigma_0)$ we have $Y \in \mathsf{bv}(\Sigma_i)$ and $Y[\Sigma_i] \equiv Y[\Sigma_0]$, and

J2. if $(\Psi, Z) \in h_i$ then each $\psi \in \Psi$ occurs *boxed*, that is, as a top-level subformula of shape $[a]\,\psi$, in some equation of Σ_0.

Preservation of the semantics by the transformation steps follows from J1 and the fact that $\mathcal{X}$ is constant in phase I. To see that Phase I terminates, note first that step I.1 terminates, because Σ_0 is weakly guarded (by assumption) and all steps preserve weak guardedness. Overall non-termination of Phase I due to the introduction of equations in step I.4 is ruled out by J2: since there are only a finite number of subsets Ψ of boxed subformulae in Σ_0, the map h fills up and thus phase I eventually terminates. $\square$

We extend the mapping θ to all formulae of simulation logic by defining $\theta(\phi) = \theta(\mathsf{snf}(\phi))$. Since snf preserves the semantics, Theorem 2.10 can be extended to all formulae, showing that $\theta(\phi)$ is the maximal model of ϕ with respect to the simulation preorder.

Theorem 2.15. *$\mathcal{S} \leq \theta(\phi)$ if and only if $\mathcal{S} \models \phi$.*

We conclude this section with two important consequences of Theorems 2.7 and 2.15. The first one is that simulation preserves logical properties.

Corollary 2.16. *$\mathcal{S}_1 \leq \mathcal{S}_2$ and $\mathcal{S}_2 \models \phi$ implies $\mathcal{S}_1 \models \phi$.*

The second corollary expresses that the maps χ and θ form a Galois connection between the preorder $(\mathsf{S}, \leq)$ of (isomorphism classes of) *finite* specifications ordered by simulation and be the preorder $(\mathsf{L}, \sqsubseteq)$ of formulae of simulation logic ordered by logical consequence.

Corollary 2.17. *χ and θ are monotone and, for finite specifications $\mathcal{S}$, $\mathcal{S} \leq \theta(\phi)$ if and only if $\chi(\mathcal{S}) \sqsubseteq \phi$.*

Finally, it is worth noting that all results can be transferred to the setting of weak simulation and logic (see [19]).

3 Compositional Verification of Applets

Having so far developed our results for arbitrary specifications, we shall now concentrate on a particular application, namely, the representation of applets (*i.e.* smart card applications) as specifications. We study sequential (single-threaded) applets and safety properties of their interprocedural control flow. As explained above, we are interested in the decomposition of properties, in order to guarantee the secure post-issuance loading of applets. We do this by instantiating the general framework of the previous section on two different levels: (1) the structural level, where a specification represents the control flow graph of an applet, and (2) the behavioural level, where a specification represents the behaviour of an applet. This yields a version of simulation and simulation logic for each level. We develop a compositional verification principle, where assumptions about individual applets are stated in the structural simulation logic and properties of the composed system are expressed in the behavioural logic.

3.1 Applets

We model the control structure of an applet as a collection of method specifications. However, for compositional reasoning about applets, we need to know which methods exist and/or are used. Therefore, we first define the notion of an applet interface. Let $\mathcal{M}eth$ be an infinite set of method names not containing the special symbols r and ε.

Definition 3.1. ***(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. We say I is *closed* if $I^- \subseteq I^+$. The *composition* of two interfaces $I_1 = (I_1^+, I_1^-)$ and $I_2 = (I_2^+, I_2^-)$ is defined by $I_1 \cup I_2 = (I_1^+ \cup I_2^+, I_1^- \cup I_2^-)$.

Next, we define method specifications, which are the basic building blocks of applets.

Definition 3.2. ***(Method specification)*** A *method graph* for $m \in \mathcal{M}eth$ over a set M of method names is a finite model $\mathcal{M}_m = (V_m, L_m, \rightarrow_m, A_m, \lambda_m)$, where V_m is the set of control nodes of m, $L_m = M \cup \{\varepsilon\}$, $A_m = \{m, r\}$, $m \in \lambda_m(v)$ for all $v \in V_m$, *i.e.* each node is tagged with the method name. A *method specification* for $m \in \mathcal{M}eth$ over M is a specification $(\mathcal{M}_m, E_m)$ such that $\mathcal{M}_m$ is a method graph for m over M.

The nodes labelled with the distinguished atomic proposition r are the *return points* of m. An applet is a collection of method specifications. For the formal definition we extend the notion of disjoint union from models (as defined below Definition 2.3) to specifications by $(\mathcal{M}_1, E_1) \uplus (\mathcal{M}_2, E_2) = (\mathcal{M}_1 \uplus \mathcal{M}_2, E_1 \uplus E_2)$.

Definition 3.3. ***(Applet)*** *Applets $\mathcal{A}$ with interface I,* written $\mathcal{A}\colon I$, are inductively defined by

- $\mathbf{0}_M : (\varnothing, M)$, where $\mathbf{0}_M$ is the *empty applet* over M defined by $\mathbf{0}_M = ((\varnothing, M \cup \{\varepsilon\}, \varnothing, \{r\}, \varnothing), \varnothing)$,
- $(\mathcal{M}_m, E_m) : (\{m\}, M)$ if $(\mathcal{M}_m, E_m)$ is a method specification for m over M,
- $\mathcal{A}_1 \uplus \mathcal{A}_2 : I_1 \cup I_2$ if $\mathcal{A}_1 : I_1$ and $\mathcal{A}_2 : I_2$.

An applet $\mathcal{A} : I$ is *closed* if its interface I is closed.

The definition requires that each provided method $m \in I^+$ of an applet $\mathcal{A} : I$ has to be implemented in a method graph for m. The interface of an applet can be derived from its implementation: a straightforward induction shows that if $\mathcal{A}$ is an applet built from a model over L and A then its interface is $(A - \{r\}, L - \{\varepsilon\})$. We write $\mathcal{S} : I$ for an arbitrary specification $\mathcal{S}$ to mean that $\mathcal{S}$ is (isomorphic to) an applet with interface I. Note that, up to isomorphism, $\uplus$ is associative and commutative with neutral element $\mathbf{0}_\varnothing$.

3.2 Structural Level

Structural simulation on applets coincides with simulation on the specifications defining applets. For convenience we write $\mathcal{A}_1 \leq_s \mathcal{A}_2$ instead of $\mathcal{A}_1 \leq \mathcal{A}_2$ to denote *structural simulation*. Structural simulation is preserved by applet composition.

Theorem 3.4. *If $\mathcal{A}_1 \leq_s \mathcal{B}_1$ and $\mathcal{A}_2 \leq_s \mathcal{B}_2$ then $\mathcal{A}_1 \uplus \mathcal{A}_2 \leq_s \mathcal{B}_1 \uplus \mathcal{B}_2$.*

We also instantiate simulation logic to this level. For an applet $\mathcal{A} : I$ and a formula ϕ of simulation logic over $L = I^- \cup \{\varepsilon\}$ and $A = I^+ \cup \{r\}$, we write $\mathcal{A} \models_s \phi$ instead of $\mathcal{A} \models \phi$ for clarity.

Maximal Applets Let $I = (I^+, I^-)$ be an applet interface. Define $\phi_I[\Sigma_I]$, the *characteristic formula for I*, by

$$\begin{array}{lcl} \phi_I & = & \bigvee_{m \in I^+} X_m \\ \Sigma_I & = & \{X_m = [I^-, \varepsilon]\, X_m \wedge p_m \mid m \in I^+\} \\ p_m & = & m \wedge \bigwedge\{\neg m' \mid m' \in I^+, m' \neq m\} \end{array}$$

The formula $\phi_I[\Sigma_I]$ axiomatises the basic structure of an applet with interface I, namely, each initial node belongs to a unique method m and no transition leaves m. Note that Σ_I is *not* in SNF (proposition r is missing).

Example 3.5. The characteristic formula for interface $I = (\{m_1, m_2\}, \{m_1, m_3\})$ is given by the formula $\phi_I[\Sigma_I]$, where $\phi_I = X_{m_1} \vee X_{m_2}$ and

$$\Sigma_I = \left[\begin{array}{l} X_{m_1} = [m_1, m_3, \varepsilon] X_{m_1} \wedge m_1 \wedge \neg m_2 \\ X_{m_2} = [m_1, m_3, \varepsilon] X_{m_2} \wedge m_2 \wedge \neg m_1 \end{array}\right]$$

The specifications satisfying ϕ_I are essentially the applets with interface I as we will show below. Using the characteristic formula for interfaces, we can define maximal applets.

Definition 3.6. ***(Maximal applet)*** The *maximal applet* w.r.t. interface I and formula $\phi[\Sigma]$ is defined as $\theta_I(\phi[\Sigma]) = \theta(\phi \wedge \phi_I[\Sigma, \Sigma_I])$ (where it is assumed w.l.o.g. that the bound variables of Σ and Σ_I are disjoint).

The following result records the main properties of characteristic formulae and maximal applets.

Theorem 3.7. *Let I be an applet interface. For any specification $\mathcal{S} = (\mathcal{M}, E)$ over labels $L = I^- \cup \{\varepsilon\}$ and atomic propositions $A = I^+ \cup \{r\}$ we have*

(i) $\mathcal{S} \models_s \phi_I$ if and only if $\mathcal{R}(\mathcal{S}) : I$, and

(ii) $\mathcal{S} \leq_s \theta_I(\phi)$ if and only if $\mathcal{S} \models_s \phi$ and $\mathcal{R}(\mathcal{S}) : I$.

Proof. (i) (Sketch) "$\Rightarrow$" By an induction on the size of I^+. The restriction to the reachable part of $\mathcal{S}$ is required, because the ϕ_I does not constrain the unreachable parts of $\mathcal{S}$. "$\Leftarrow$" By inspection of the definition of applets. (ii) Using the definition of $\theta_I(\phi)$ and Theorem 2.15 we know that $\mathcal{S} \leq_s \theta_I(\phi)$ is equivalent to $\mathcal{S} \models_s \phi$ and $\mathcal{S} \models_s \phi_I$. The result then follows from (i). □

Point (i) of the theorem essentially expresses that the formula ϕ_I characterises those specifications that are applets with interface I, while point (ii) extends Theorem 2.15 from specifications to applets. As a consequence of (ii) we have $\theta_I(\phi) \models \phi_I$ and $\theta_I(\phi) : I$, since all nodes of $\theta_I(\phi)$ are reachable by construction.

3.3 Behavioural Level

Next, we change our focus to the behavioural level, where we first define the operational semantics of a closed applet. Since our compositional method uses structural assumptions, there is no need to compose applets on the behavioural level, so an operational semantics of *closed* applets is sufficient. In contrast, in previous work on semi-automatic compositional applet verification [3], the use of behavioural assumptions required a more involved open semantics of applets.

Definition 3.8. ***(Behaviour)*** Let $\mathcal{A} = (\mathcal{M}, E) : (I^+, I^-)$ be a *closed* applet and let $\mathcal{M} = (V, L, \rightarrow, A, \lambda)$. The *behaviour* of $\mathcal{A}$ is described by the specification $b(\mathcal{A}) = (\mathcal{M}_b, E_b)$, where $\mathcal{M}_b = (S_b, L_b, \rightarrow_b, A_b, \lambda_b)$ is defined by $S_b = V \times V^*$, that is, states are pairs of control points and stacks, $L_b = \{m_1\, l\, m_2 \mid l \in \{\mathsf{call}, \mathsf{ret}\},\ m_1, m_2 \in I^+\} \cup \{\varepsilon\}$, $\rightarrow_b$ is defined by the transition rules of Table 1, $A_b = A$ and $\lambda_b((v,\sigma)) = \lambda(v)$. The set of initial states E_b is defined by $E_b = E \times \{\varepsilon\}$.

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

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

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

Table 1. Applet Transition Rules

Note that the applet transition rules define a pushdown process (*cf.* survey paper [8]).

Applet $\mathcal{A}_1$ *behaviourally simulates* applet $\mathcal{A}_2$, written $\mathcal{A}_1 \leq_b \mathcal{A}_2$, if $b(\mathcal{A}_1) \leq b(\mathcal{A}_2)$. The notions of applet structure and behaviour have been carefully chosen to ensure that any two applets related by structural simulation are also related by behavioural simulation. In general, the inverse does not hold.

Theorem 3.9. (Simulation Correspondence) *If* $\mathcal{A}_1 \leq_s \mathcal{A}_2$ *then* $\mathcal{A}_1 \leq_b \mathcal{A}_2$.

Proof. Let R be a structural simulation between $\mathcal{A}_1$ and $\mathcal{A}_2$. We lift R from the structural level to R_b on the behavioural level by defining $((v,\sigma),(v',\sigma')) \in R_b$ if and only if $(v,v') \in R$, $|\sigma| = |\sigma'|$ and $(\sigma(i), \sigma'(i)) \in R$ for all $0 \leq i < |\sigma|$. It is easy to check that R_b is a behavioural simulation between $\mathcal{A}_1$ and $\mathcal{A}_2$. □

Finally, we instantiate simulation logic on the behavioural level. Behavioural properties are more abstract than structural ones as they do not refer to the program control structure. We define *behavioural satisfaction* $\mathcal{A} \models_b \psi$ as $b(\mathcal{A}) \models \psi$ for applets $\mathcal{A} : I$ and ψ a formula of simulation logic over L_b and A_b.

3.4 Compositional Reasoning

The general results of Section 2 together with those of this section form the basis for the main contribution of our paper, the following *compositional reasoning principle* for applets, which combines reasoning at the structural and behavioural levels. Let $\mathcal{A} : I$ and $\mathcal{B} : J$ be applets such that $I \cup J$ is closed and let ϕ and ψ be formulae of structural and behavioural simulation logic, respectively. Then we have

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

This principle says that in order to show that a composed applet $\mathcal{A} \uplus \mathcal{B}$ has a behavioural property ψ, it is sufficient to find a structural property ϕ that is satisfied by $\mathcal{A}$ and such that $\theta_I(\phi) \uplus \mathcal{B} \models_b \psi$. We prove its soundness and completeness using the following result, which characterises the second premise of rule (compos).

Proposition 3.10. *Let* $\mathcal{B} : J$ *be an applet and* I *an interface s.t.* $I \cup J$ *is closed. Then* $\theta_I(\phi) \uplus \mathcal{B} \models_b \psi$ *if and only if for all* $\mathcal{A} : I$ *with* $\mathcal{A} \models_s \phi$ *we have* $\mathcal{A} \uplus \mathcal{B} \models_b \psi$.

Proof. "$\Rightarrow$" Suppose $\theta_I(\phi) \uplus \mathcal{B} \models_b \psi$, $\mathcal{A} : I$ and $\mathcal{A} \models_s \phi$. Then certainly also $\mathcal{R}(\mathcal{A}) : I$ and so we get $\mathcal{A} \leq_s \theta_I(\phi)$ by Theorem 3.7(ii). From Theorems 3.4 and 3.9 we derive that $\mathcal{A} \uplus \mathcal{B} \leq_b \theta_I(\phi) \uplus \mathcal{B}$. Hence, $\mathcal{A} \uplus \mathcal{B} \models_b \psi$ by Corollary 2.16. "$\Leftarrow$" By Theorem 3.7(ii) we have $\theta_I(\phi) : I$ and $\theta_I(\phi) \models_s \phi$, thus $\theta_I(\phi) \uplus \mathcal{B} \models_b \psi$. □

Theorem 3.11. *Rule* (compos) *is sound and complete.*

Proof. Soundness is immediate by Proposition 3.10. For completeness suppose $\mathcal{A} \uplus \mathcal{B} \models_b \psi$ and set $\phi = \chi(\mathcal{A})$. By Theorem 2.7 we have $\mathcal{A} \models_b \chi(\mathcal{A})$. To establish the second premise of the rule, we use Proposition 3.10 and show $\mathcal{C} \uplus \mathcal{B} \models_b \psi$ for an arbitrary $\mathcal{C} : I$ with $\mathcal{C} \models_s \mathcal{X}(\mathcal{A})$. We use Theorem 2.7 to derive $\mathcal{C} \leq_s \mathcal{A}$. The result then follows by Theorems 3.4 and 3.9 and Corollary 2.16. □

Other useful compositional reasoning principles are thinkable. For example, a rule with a similar shape as the one above, but involving structural properties only, is easily justifiable with the results presented above (see [19]).

4 Example

To demonstrate the use of our approach in practice, we present a small example, which is a highly distilled version of a larger case study on the verification of security properties for an electronic purse. We refer the interested reader to [12] for more details, including a more detailed motivation of why this kind of security properties are important for smart card applications and how they can be formalised.

Suppose we have a smart card on which we allow instances of applets $\mathcal{A}$ and $\mathcal{B}$ with the following respective interfaces: $I_{\mathcal{A}} = (\{m_1, m_2\}, \{m_1, m_2, m_3\})$ and $I_{\mathcal{B}} = (\{m_3\}, \{m_1, m_2, m_3\})$. Suppose method m_1 is called by an instance of applet $\mathcal{B}$ whenever this instance is in a particular state. However, it could be that only certain instances of applet $\mathcal{A}$ are supposed to know when an instance of $\mathcal{B}$ is in such a state - possibly because they have paid to get this information. Thus, as a global security property we require that *within invocations of method m_1, no other calls to instances of $\mathcal{A}$ are triggered*. This can be considered as a confidentiality property: it prevents certain information (namely: m_1 is called) to flow to unauthorised applications [4]. We specify this as the global property

$$(\psi) \quad \neg m_1 \vee Z\,[Z = [K]\,\mathsf{ff} \wedge [-]Z]$$

where $K = \{m_i \,\mathsf{call}\, m_j \mid 1 \leq i \leq 3 \text{ and } 1 \leq j \leq 2\}$. Formula ψ expresses that within method m_1 there cannot be any calls to other methods of $\mathcal{A}$. Notice that this also disallows indirect calls via an instance of applet $\mathcal{B}$, *i.e.* m_3 is not allowed either to call methods declared in the interface of $\mathcal{A}$. The verification of property ψ can be decomposed in several ways. A trivial way is to disallow any method calls in method m_1. However, this would also exclude implementations of method m_1 that communicate with instances of $\mathcal{B}$. Hence, we propose less restrictive structural specifications for $\mathcal{A}$ and $\mathcal{B}$:

$$\begin{array}{ll} (\sigma_{\mathcal{A}}) & \neg m_1 \vee X[X = [m_1, m_2]\mathsf{ff} \wedge [\varepsilon, m_3]X] \\ (\sigma_{\mathcal{B}}) & \neg m_3 \vee Y[Y = [m_1, m_2]\mathsf{ff} \wedge [\varepsilon, m_3]Y] \end{array}$$

Formula $\sigma_{\mathcal{A}}$ says that the method graph for m_1 does not contain any call edge labelled m_1 or m_2, while $\sigma_{\mathcal{B}}$ expresses a similar property for the graph of m_3.

Applying the compositional reasoning principle (compos) twice, we know that to establish $\mathcal{A} \uplus \mathcal{B} \models_b \psi$, it is sufficient to prove $\mathcal{A} \models_s \sigma_{\mathcal{A}}$, $\mathcal{B} \models_s \sigma_{\mathcal{B}}$ and $\theta_{I_{\mathcal{A}}}(\sigma_{\mathcal{A}}) \uplus \theta_{I_{\mathcal{B}}}(\sigma_{\mathcal{B}}) \models_b \psi$. We have assembled a tool set to support all these verification tasks [12]. The former two local properties are checked using existing finite-state model checking techniques. The maximal applets for $\sigma_{\mathcal{A}}$ and $\sigma_{\mathcal{B}}$ are generated by an implementation of the algorithm in Section 2. Finally, $\theta_{I_{\mathcal{A}}}(\sigma_{\mathcal{A}}) \uplus \theta_{I_{\mathcal{B}}}(\sigma_{\mathcal{B}}) \models_b \psi$ is verified using Alfred [17], a model checker for pushdown systems based on the algorithm by Bouajjani *et al.* [5], thus establishing the correctness of the property decomposition.

For illustration, we present in some detail the construction of the maximal applet for $\sigma_{\mathcal{B}}$; the construction is similar for $\sigma_{\mathcal{A}}$. First, we build the characteristic formula for interface $I_{\mathcal{B}}$, that is, $\phi_{I_{\mathcal{B}}} = X_{m_3}[X_{m_3} = [m_1, m_2, m_3, \varepsilon]X_{m_3} \wedge m_3]$. Introducing a new variable Z for $\phi_{I_{\mathcal{B}}} \wedge \sigma_{\mathcal{B}}$ yields

$$Z \left[\begin{array}{rcl} Z & = & (\neg m_3 \vee Y) \wedge X_{m_3} \\ Y & = & [m_1, m_2]\mathsf{ff} \wedge [m_3, \varepsilon]Y \\ X_{m_3} & = & [m_1, m_2, m_3, \varepsilon]X_{m_3} \wedge m_3 \end{array} \right]$$

The next step is to transform this formula into SNF. First, in Phase I of the transformation, each equation is transformed into a disjunction of state normal forms. Suppose we start with the equation defining Z.

1. Make equation strongly guarded, by rewriting with the original equations:

$$\begin{array}{rcl} Z & = & (\neg m_3 \vee ([m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon]Y)) \wedge \\ & & [m_1, m_2, m_3, \varepsilon]X_{m_3} \wedge m_3 \end{array}$$

2. Put equation into DNF and simplify:

$$\begin{array}{rcl} Z & = & [m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon]Y \wedge \\ & & [m_1, m_2, m_3, \varepsilon]X_{m_3} \wedge m_3 \end{array}$$

3. Group and complete boxes. No boxes are missing, thus we only group them:

$$Z = [m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon](Y \wedge X_{m_3}) \wedge m_3$$

4. Introduce new equations for formulae under boxes. Since the map h does not yet contain an entry for $\{Y, X_{m_3}\}$, we choose a fresh variable U and add $(\{Y, X_{m_3}\}, U)$ to h. The equation defining Z becomes

$$Z = [m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon]U \wedge m_3$$

while we introduce the new equation $U = Y \wedge X_{m_3}$.

5. Finally, complete the equation by adding missing literals and put the formula into DNF again. Here, only literal r is missing. Adding this gives:

$$\begin{array}{rcl} Z & = & ([m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon]U \wedge m_3 \wedge r) \vee \\ & & ([m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon]U \wedge m_3 \wedge \neg r) \end{array}$$

The equations defining Y and X_{m_3} are handled in a similar way. The only step that has some effect is step 5, which introduces the missing literal r. More interesting is to look how Phase I is applied to the new equation $U = Y \wedge X_{m_3}$.

1. Rewriting into strongly guarded form yields:

$$\begin{array}{rcl} U & = & [m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon]Y \wedge \\ & & [m_1, m_2, m_3, \varepsilon]X_{m_3} \wedge m_3 \end{array}$$

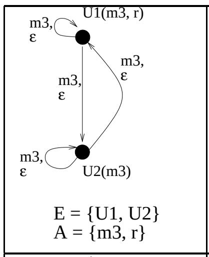	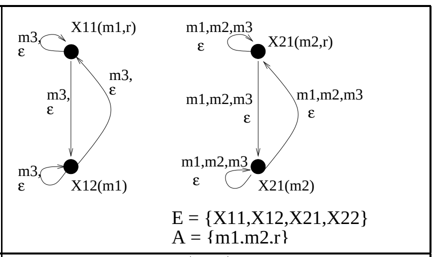
$\theta_{I_{\mathcal{B}}}(\sigma_{\mathcal{B}})$	$\theta_{I_{\mathcal{A}}}(\sigma_{\mathcal{A}})$

Figure 3. Maximal applets for $\sigma_{\mathcal{B}}$ and $\sigma_{\mathcal{A}}$

2. Formula ϕ_U is already in DNF and cannot be simplified.

3. Grouping boxes results in the following equation:

$$U = [m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon](Y \wedge X_{m_3}) \wedge m_3$$

4. The map h contains the pair $(\{Y, X_{m_3}\}, U)$, so we replace $Y \wedge X_{m_3}$ by U.

$$U = [m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon]U \wedge m_3$$

5. Literal completion again introduces r.

$$\begin{array}{rcl} U & = & ([m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon]U \wedge m_3 \wedge r) \vee \\ & & ([m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon]U \wedge m_3 \wedge \neg r) \end{array}$$

After applying Phase I to all equations, Phase II introduces a new equation for each disjunct and replaces each old variable by the disjunction of the new variables. For example, the equation defining U gets replaced by:

$$\begin{array}{rcl} U_1 & = & [m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon](U_1 \vee U_2) \wedge m_3 \wedge r \\ U_2 & = & [m_1, m_2]\,\mathsf{ff} \wedge [m_3, \varepsilon](U_1 \vee U_2) \wedge m_3 \wedge \neg r \end{array}$$

The remaining equations are treated similarly. Notice that also Z in $\mathcal{X}$ gets replaced by $\{Z_1, Z_2\}$, where Z_1 and Z_2 are the equations replacing Z.

During the optimisation in Phase III, we find that the equations for Z_1 and U_1, and Z_2 and U_2 are duplicates of each other. Therefore, we remove the equations for Z_1 and Z_2, and replace $\{Z_1, Z_2\}$ in $\mathcal{X}$ by $\{U_1, U_2\}$. Further, the equations Y_1, Y_2, X_{m_31} and X_{m_32} (replacing Y and X_{m_3} in Phase II), are not reachable from any variable in $\mathcal{X} = \{U_1, U_2\}$. Hence, the final result is $U_1 \vee U_2[\Sigma]$, where

$$\Sigma = \left[\begin{array}{l} U_1 = [m_1, m_2]\mathsf{ff} \wedge [m_3, \varepsilon](U_1 \vee U_2) \wedge m_3 \wedge r \\ U_2 = [m_1, m_2]\mathsf{ff} \wedge [m_3, \varepsilon](U_1 \vee U_2) \wedge m_3 \wedge \neg r \end{array} \right]$$

Figure 3 displays the maximal applet corresponding to this equation system (in its left column). It also shows the maximal applet for $\sigma_{\mathcal{A}}$, found in a similar way.

5 Conclusions

We propose an algorithmic compositional verification method for control flow based safety properties of smart card applets, where local assumptions on individual applets are structural, while global guarantees are behavioural. Safety properties are adequately expressed in our simulation logic, a modal logic with simultaneous greatest fixed points. We establish representation results connecting logical satisfaction to simulation in a general setting, including a characterisation of logical satisfaction in terms of maximal models. Our novel maximal model construction transforms the formula into an equivalent simulation normal form, isomorphic to a maximal model.

For compositional applet verification we define maximal applets at the structural level as the maximal model of the local structural property restricted by a formula characterising the given interface. From these results and the fact that structural simulation implies behavioural simulation, we derive a sound and complete compositional method, reducing the correctness of property decompositions to a model checking problem for pushdown processes and thus extending existing compositional techniques for finite-state systems to a useful class of infinite-state systems.

The companion paper [12] presents a tool set that we have developed and applied to an industrial electronic purse case study to demonstrate the practical applicability of the approach. Section 4 contains a highly distilled version of this work. It is noteworthy that, in the present setting, the method supports secure post-issuance loading of applets, but it could be applied to any type of sequential programs with recursive procedures for which compositional verification of control flow properties is desired.

There are several possible directions for future work, including (i) adding diamond modalities to the simulation logic, (ii) refining the notion of interface, by defining public and private interfaces, and (iii) investigating under what restrictions the proposed method can be adapted to allow behavioural assumptions in place of structural ones.

References

[1] H. R. Andersen. Partial model checking (extended abstract). In *Logic in Computer Science (LICS 95)*, pages 398–407. IEEE Computer Society Press, 1995.

[2] A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Publishing, 2001.

[3] G. Barthe, D. Gurov, and M. Huisman. Compositional verification of secure applet interactions. In R.-D. Kutsche and H. Weber, editors, *Fundamental Approaches to Software Engineering 2002*, number 2306 in LNCS, pages 15–32. Springer, 2002.

[4] P. Bieber, J. Cazin, V. Wiels, G. Zanon, P. Girard, and J.-L. Lanet. Electronic purse applet certification: extended abstract. In S. Schneider and P. Ryan, editors, *Workshop on secure architectures and information flow*, volume 32 of *Elect. Notes in Theor. Comp. Sci.* Elsevier Publishing, 2000.

[5] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *International Conference on Concurrency Theory*, pages 135–150, 1997.

[6] A. Bouajjani, J. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for branching time semantics. In *Automata, Languages and Programming*, pages 76–92, 1991.

[7] G. Boudol and K. Larsen. Graphical versus logical specifications. *TCS*, 106:3–20, 1992.

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

[9] W.-P. de Roever, F. de 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, Nov. 2001.

[10] O. Grumberg and D. Long. Model checking and modular verification. *ACM Trans. on Prog. Lang. & Syst.*, 16(3):843–871, 1994.

[11] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

[12] M. Huisman, 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 2004*, number 2984 in LNCS, pages 84–98. Springer, 2004.

[13] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.

[14] O. Kupferman and M. Vardi. An automata-theoretic approach to modular model checking. *ACM Trans. on Prog. Lang. & Syst.*, 22(1):87–128, 2000.

[15] K. Larsen. Modal specifications. In *Automatic Verification Methods for Finite State Systems*, number 407 in LNCS. Springer, 1989.

[16] K. Laster and O. Grumberg. Modular model checking of software. In *Tools and Algorithms for the Analysis and Construction of Software, TACAS 98*, LNCS. Springer, 1998.

[17] D. Polanský. Verifying properties of infinite-state systems. Master's thesis, Masaryk University, Faculty of Informatics, Brno, 2000.

[18] L. Prensa Nieto. The Rely-Guarantee method in Isabelle/HOL. In P. Degano, editor, *European Symposium on Programming (ESOP'03)*, volume 2618 of *LNCS*, pages 348–362. Springer, 2003.

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

[20] C. Stirling. *Modal and Temporal Logics of Processes*. Springer, 2001.

[21] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. of Math.*, 5, 1955.

[22] I. Walukiewicz. Pushdown processes: games and model checking. In *Proceedings of CAV'96*, number 1102 in LNCS, pages 62–75, 1996.