Simulation Logic, Applets and Compositional Verification*

Christoph Sprenger
INRIA Sophia Antipolis
Sophia Antipolis, France
christoph.sprenger@inria.fr

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 a compositional verification method for control flow based safety properties of smart card applets. Our method rests on a close correspondence between transition system models ordered by simulation and Hennessy-Milner logic extended with simultaneous greatest fixed points. We show that simulation can be characterised logically and, vice versa, logical satisfaction can be represented behaviourally by a maximal model for a given formula. Based on these results and earlier ideas by Grumberg and Long we develop a compositional verification technique, where maximal models replace logical assumptions to reduce compositional verification to standard model checking. However, in the context of applets, equipped with interfaces, this technique needs to be refined. Since for a given behavioural formula and interface a maximal applet does not always exist, we propose a two-level approach, where local assumptions restrict the control flow *structure* of applets, while the global property restricts the control flow *behaviour* of the system. By separating the tasks of verifying global and local properties of applets, our method supports secure post-issuance loading of new applets onto a smart card.

Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]; F.1.1 [**Models of Computation**]; F.3.1 [**Specifying and Verifying and Reasoning about Programs**]; F.4.1 [**Mathematical Logic**].

General Terms

Algorithms, Reliability, Theory, Verification.

Keywords

Applets, security, temporal logic, compositional verification, context-free processes.

*This work is partially supported by the European Union as part of the VerifiCard project IST-2000-26328.

1. INTRODUCTION

With the emergence of small secure devices, such as open platform smart cards and secure modules as Palladium[1] and Embassy[2], 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 privacy.

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, the user can install new applets on the card himself; he does not have to go back to the card issuer in order to do this. In this case automatic checks are needed to ensure that the new applet 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: to be able to safely install an applet post-issuance on a smart card, it needs to respect certain control flow properties as specified. More precisely, we study sequential (single-threaded) applets and propose a specification and verification method for safety properties of interprocedural control flow, *i.e.* properties describing sequences of method invocations which are deemed safe for the given application. Since we are interested in post-issuance loading of applets, the implementation of applets might not be available at verification time. We therefore propose a *compositional* verification method, which allows the verification problem to be reduced to the following three tasks:

(i) decomposing the global behavioural property by finding local structural properties of the components (here applets),

(ii) proving correctness of this decomposition, that is, verifying that the local applet properties (assumptions) are sufficient to guarantee the global property (guarantee), and

(iii) verifying that applets satisfy their assumptions.

As explained below, assumptions are structural rather than behavioural to allow algorithmic checking of correctness of property decompositions. This paper focuses on task (ii),

[1]http://www.microsoft.com/resources/ngscb/default.mspx
[2]http://www.wavesys.com/technology/embassy.html

while for task (iii) standard algorithmic techniques already exist.

The compositional verification method proposed here supports different scenarios for secure post-issuance loading of applets *w.r.t.* control flow safety properties. 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 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 in this scenario also the property decomposition needs to be verified at loading time, potentially on-card.

1.1 Our Approach

In earlier work [2], a proof system based approach at proving correctness of property decompositions is investigated, which aims at semi-automatic verification. However, for many applications an algorithmic verification method is preferable, even more so if such a check is to be performed frequently as in our second scenario.

The approach that we take here is inspired by the work on modular verification by Grumberg and Long [9]. Their framework is based on a behavioural *simulation* preorder which (i) is preserved under (parallel) composition, and (ii) preserves satisfaction of properties specified in ACTL, the universal-path fragment of the (branching-time) temporal logic CTL [8]. This justifies compositional verification in the following style: to verify that the composition of components (that is, behaviours) X and Y satisfies a global temporal property ψ, one finds an abstraction of X, that is, a behaviour X' simulating X, such that the composition of X' and Y satisfies ψ. Component Y can be treated similarly. In addition, a *maximal model* construction θ is given, with the property that X satisfies ϕ exactly when X is simulated by $\theta(\phi)$. This construction allows behavioural abstractions to be given through temporal logic formulae (rather than through behaviours), supporting verification in the following style: to verify that the composition of X and Y satisfies ψ, one finds a property ϕ of X, such that the composition of $\theta(\phi)$ and Y satisfies ψ. Components X and Y are assumed to be finite-state behaviours, allowing the verification of both resulting sub-problems to be performed with standard model-checking techniques.

In contrast to the above, we are faced with potentially infinite context-free applet behaviours, generated from finite applet structures. We consider sequential applets only, where applet composition is structural, that is, joins their structures, without introducing concurrency in the behaviour. In this context, the decidability of the correctness problem of property decompositions is an open problem. In our setup, even when restricting to safety properties as mentioned above, in general there is no maximal applet structure for a given behavioural property ψ. For this reason, we adopt a scheme where

(i) local specifications (assumptions) are *structural* properties, restricting the control-flow structure of applets;

(ii) global specifications (guarantees) are *behavioural* properties, restricting the control-flow behaviour of applets.

To verify that the composition of applet structures X and Y satisfies the behavioural property ψ, one finds a *structural* property ϕ of X, such that the composition of $\theta(\phi)$ and Y satisfies ψ. Again, the resulting verification sub-problems are algorithmically checkable: showing that X satisfies ϕ is a standard model-checking problem (if the applet structure X is viewed as a Kripke structure, see e.g. [17]), while showing that the composition of $\theta(\phi)$ and Y satisfies ψ can be checked by standard techniques for model-checking temporal properties of context-free processes [7].

To be able to handle applet structure and behaviour in a uniform way, we first develop a general framework for abstract specifications (models with designated entry points). Then, the method outlined above is obtained by combining instantiations of this framework on both the structural and the behavioural level, with additional results to connect the two levels.

1.2 Summary of Results

Section 2 develops the general framework in the setting of abstract specifications. After the introduction of simulation and a corresponding logic, called simulation logic, we connect these formally by defining maps between specifications and logical formulae. We then present two characterisation results. The first is a logical characterisation of simulation that states that, for any (finite) specification $\mathcal{T}$, there is a characteristic formula $\chi(\mathcal{T})$ such that

$$\mathcal{S} \leq \mathcal{T} \iff \mathcal{S} \models \chi(\mathcal{T}) \tag{1}$$

that is, $\mathcal{T}$ simulates $\mathcal{S}$ precisely if $\mathcal{S}$ satisfies $\chi(\mathcal{T})$. The second, complementary, result is a behavioural characterisation of logical satisfaction that says that, for any formula ϕ of simulation logic, there is a *maximal* specification $\theta(\phi)$ such that

$$\mathcal{S} \models \phi \iff \mathcal{S} \leq \theta(\phi) \tag{2}$$

Thus a specification satisfies a formula ϕ precisely if it simulates the maximal specification $\theta(\phi)$ obtained from ϕ. The map θ is first defined on formulas in so-called simulation normal form and then extended to all formulas by defining an effective stepwise transformation of formulas into simulation normal form. The two characterisations (1) and (2) combine into a Galois connection between the preorder of finite specifications ordered by simulation and the preorder of logical formulae ordered by logical consequence. As another corollary, simulation preserves the satisfaction of formulae of simulation logic.

Next, Section 3 instantiates these general results to the notion of applets. An applet is defined as a collection of method specifications, which are essentially control graph structures together with entry points. Further, to each applet we associate an interface, defining which methods it provides and which methods it uses. The behaviour of an applet is then a context-free specification derived via a set of transition rules. By instantiation of the framework from Section 2, we obtain appropriate notions of simulation and logic on both the structural and the behavioural levels. Our compositional reasoning principle then looks as follows

$$\frac{\mathcal{A} \models_s \phi \qquad \theta_I(\phi) \uplus \mathcal{B} \models_b \psi}{\mathcal{A} \uplus \mathcal{B} \models_b \psi} \tag{3}$$

where $\mathcal{A}$ and $\mathcal{B}$ are applets, $\uplus$ is applet composition, ϕ is an assumption in the structural logic and ψ is a behavioural guarantee. Its correctness mainly rests on an instantiation of the characterisation (2) on the structural level. In order to make sure that the maximal specification $\theta(\phi)$ is itself an applet, we introduce the characteristic formula ϕ_I for a given applet interface I, which is conjoined to ϕ before applying θ. Doing this we obtain a variant of (2) for applets, where θ is replaced by θ_I, so the maximal specification is guaranteed to be an applet with interface I.

The link between the structural and behavioural levels is provided by the additional result that structural simulation is contained in behavioural simulation. Together with the facts that simulation is preserved by applet composition and that behavioural simulation preserves satisfaction of behavioural formulae, this justifies principle (3). Moreover, by using the characteristic formula $\chi(\mathcal{A})$ as the local assumption on ϕ and invoking the structural version of (1), we also establish the completeness of principle (3).

This paper focuses on the theoretical underpinning of the proposed compositional approach. Section 4 sketches how our techniques can be applied to an example. This example is distilled from a larger case study, described elsewhere in full detail [10], which supports our claim that this setup is sufficient to handle relevant practical applications. An accompanying technical report [16] contains more details and proofs.

1.3 Related Work

As stated above, our approach to compositional verification of applets is inspired by the work on modular verification by Grumberg and Long [9] (later developed further by Kupferman and Vardi [14]). We explained why and how we deviate from it; in addition, it should be pointed out that the logic ACTL on which their framework is based, allows safety as well as liveness properties to be expressed, and that the models they consider contain fairness constraints, these being crucial for the existence and construction of maximal models for liveness properties. Since the properties we are mainly interested in are safety properties, such as, e.g., the absence of illicit control flow, there is no need to add fairness constraints to our models. Apart from these differences in setup, the maximal model construction in the paper [9], being is a global one, generates a model that is always exponentially larger than the property formula. In contrast, our construction involves a step-wise transformation of the property formula into simulation normal form which then directly corresponds to a maximal model. Thus, our approach is of a more local nature and avoids unnecessary exponential blow-ups.

The general treatment of simulation and its logical characterisation that we adopt here follows the approach to logical characterisation of refinement by Larsen and Boudol [15, 5]. While on one hand we are more restrictive in our notion of behaviour, using labelled transition systems rather than the more general notion of modal transition systems considered by these authors, using simulation as refinement, and dropping the diamond modality from the logic, on the other hand we extend their results to modal logic with recursion and recursive processes.

Our framework is also influenced in part by work by Jensen *et al.* [12], who provide a (non-compositional) algorithmic verification method for control flow safety properties of applets. In particular, this work motivated us to represent applets as graphs.

2. SIMULATION VERSUS LOGIC

This section develops several general results about simulation and its relation to logic. After the introduction of specifications and simulations between specifications, we present simulation logic, which is a subset of Hennessy-Milner logic [11] with (co-)recursion added. 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. These two results combine into a Galois connection between the preorder of specifications ordered by simulation and the preorder of logical formulae ordered by logical consequence. 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 specifications satisfying the formula. This will serve as the basis for our compositional verification method for applets explained in the next section.

2.1 Specifications and Simulation

First we introduce the general notion of a model over a set of labels L and a set of atomic propositions A.

Definition 1. (Model) A *model* over L and A is a structure $\mathcal{M} = (S, L, \rightarrow, A, \lambda)$, where

- S is a set of states,
- L is a finite set of labels,
- $\rightarrow \subseteq S \times L \times S$ is a transition relation,
- A is a finite set of atomic propositions, and
- $\lambda\colon S \rightarrow \mathcal{P}(A)$ is a valuation assigning to each state s the atomic propositions that hold at s.

A *specification* $\mathcal{S}$ over L and A is a pair $(\mathcal{M}, E)$, where $\mathcal{M}$ is a model over L and A and $E \subseteq S$ is a set of states.

Intuitively, one can think of E as the set of entry states of the model. As usual, we will write $s \xrightarrow{a} t$ to denote $(s, a, t) \in\rightarrow$. For convenience, we define $\lambda^*(p) = \{s \in S \mid p \in \lambda(s)\}$ for $p \in A$, *i.e.* the set of all states satisfying atomic proposition p. A model $\mathcal{M}$ is *finitely branching* models if for all $s \in S$ and $a \in L$ the set $\{t \mid s \xrightarrow{a} t\}$ is finite. A specification $(\mathcal{M}, E)$ is finitely branching if $\mathcal{M}$ is finitely branching and E is finite. A model is *finite* if its set of states is finite, while a specification is finite if the underlying model is. The next step is to define the (usual) notion of simulation on models and specifications.

Definition 2. (Simulation) A *simulation* is a binary relation R on S such that whenever $(s, t) \in R$ then

(i) $\lambda(s) = \lambda(t)$, and

(ii) if $s \xrightarrow{a} s'$ 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 $(s,t) \in R$. States s and t are *simulation equivalent*, written $s = t$, if $s \leq t$ and $t \leq s$. The notion of simulation is extended to specifications $(\mathcal{M}, E)$ and $(\mathcal{M}, E')$ by defining $(\mathcal{M}, E) \leq (\mathcal{M}, E')$ if there is a simulation R such that for each $s \in E$ there is some $t \in E'$ with $(s,t) \in R$.

Next, we extend the definition of simulation to the case where we have two specifications $(\mathcal{M}_1, E_1)$ and $(\mathcal{M}_2, E_2)$, based on different models. We do this by first defining the "disjoint union" $\mathcal{M}_1 \uplus \mathcal{M}_2$ of the underlying models as $(S, L, \rightarrow, A, \lambda)$, where $S = (S_1 \times \{1\}) \cup (S_2 \times \{2\})$, $L = L_1 \cup L_2$, $A = A_1 \cup A_2$, $\lambda(s,i) = \lambda_i(s)$ and $(s,i) \xrightarrow{a} (t,j)$ if and only if $i = j$ and $s \xrightarrow{a}_i t$. Then we work with simulations on $\mathcal{M}_1 \uplus \mathcal{M}_2$, by defining $(\mathcal{M}_1, E_1) \leq (\mathcal{M}_2, E_2)$ if $(\mathcal{M}_1 \uplus \mathcal{M}_2, E_1 \times \{1\}) \leq (\mathcal{M}_1 \uplus \mathcal{M}_2, E_2 \times \{2\})$.

2.2 Simulation Logic

Next we define a logic that characterises simulation. This logic is defined in two steps: first we define a basic logic and then we add recursion by using modal equation systems. The resulting logic is equivalent to modal μ-calculus [13] with greatest fixed points and box modalities only.

Definition 3. (Basic simulation logic: syntax) Let $\mathcal{V}$ be a countably infinite set of variables over sets of states, ranged over by $X, Y, Z, \ldots$. The formulae of *basic simulation logic* over a set L of labels and a set A of atomic propositions are inductively defined by

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

where $p \in A$ and $a \in L$. The set of free variables $\mathsf{fv}(\phi) \subseteq \mathcal{V}$ of a formula ϕ is defined as usual. Formulae of the shape p or $\neg p$ are called *literals*.

Definition 4. (Basic simulation logic: semantics) The *semantics* of a formula ϕ of basic simulation logic over L and A with respect to a model $\mathcal{M}$ over L and A and an environment $\rho\colon \mathcal{V} \to \mathcal{P}(S)$ is defined inductively by

$$\begin{array}{lcl}
\|p\|\rho & = & \lambda^*(p) \\
\|\neg p\|\rho & = & S - \lambda^*(p) \\
\|X\|\rho & = & \rho(X) \\
\|\phi_1 \wedge \phi_2\|\rho & = & \|\phi_1\|\rho \cap \|\phi_2\|\rho \\
\|\phi_1 \vee \phi_2\|\rho & = & \|\phi_1\|\rho \cup \|\phi_2\|\rho \\
\|[a]\,\phi\|\rho & = & \{s \in S \mid \forall t \in S \,.\, s \xrightarrow{a} t \text{ implies } t \in \|\phi\|\rho\}
\end{array}$$

We implicitly assume the existence of the false proposition ff with $\lambda^*(\mathsf{ff}) = \varnothing$ in all models. We then define $\mathsf{tt} = \neg\,\mathsf{ff}$.

Let us introduce some useful notations. We often use finite generalisations of the boolean connectives for which we use notations such as $\bigvee_i \phi_i$ and $\bigvee \Phi$ for a finite set Φ of formulae. For the special case of an empty set of formulae, we make the identifications $\bigvee \varnothing = \mathsf{ff}$ and $\bigwedge \varnothing = \mathsf{tt}$. For a more compact representation of modal formulae we will use $[K]\,\phi$ for $K \subseteq L$ to denote the formula $\bigwedge_{a \in K} [a]\,\phi$. In concrete cases we will omit the curly brackets and write $[a,b]\,\phi$ instead of $[\{a,b\}]\,\phi$. As a special case, we write $[-]\,\phi$ for $[L]\,\phi$.

To make the logic expressive enough to characterise all finite models, we follow Larsen [15] and introduce modal equation systems over formulae of basic simulation logic.

Definition 5. (Modal equation system) A *modal equation system* $\Sigma = \{X_i = \phi_i \mid i \in I\}$ over L and A is a finite set of equations such that the variables X_i are pairwise distinct and each ϕ_i is a formula of basic simulation logic over L and A. The set of variables occurring in Σ is partitioned into the set of *bound* variables, defined by $\mathsf{bv}(\Sigma) = \{X_i \mid i \in I\}$, and the set of *free* variables $\mathsf{fv}(\Sigma)$.

We will henceforth often use ϕ_X to refer to the formula ϕ in an equation $(X = \phi) \in \Sigma$.

Example 1. An example of a modal equation system is $\Sigma = \{X_1 = X_2 \wedge X_3, X_2 = Y, X_3 = Z\}$. For this system we have $\mathsf{bv}(\Sigma) = \{X_1, X_2, X_3\}$ and $\mathsf{fv}(\Sigma) = \{Y, Z\}$.

The next step is to define the semantics of a modal equation system, in terms of its greatest solution. A solution of a modal equation system Σ is a map $\eta : \mathsf{bv}(\Sigma) \to \mathcal{P}(S)$, assigning to each variable $X \in \mathsf{bv}(\Sigma)$ a set of states, such that all equations in Σ are satisfied. Maps η are ordered by point-wise inclusion. We first define the environment update $\rho[\eta]$, as $\rho[\eta](X) = \eta(X)$ if $X \in \mathsf{bv}(\Sigma)$ and $\rho[\eta](X) = \rho(X)$ otherwise. Then we define the map $\Psi_{\Sigma,\rho}\colon \mathcal{P}(S)^{\mathsf{bv}(\Sigma)} \to \mathcal{P}(S)^{\mathsf{bv}(\Sigma)}$ induced by the equations in Σ by $\Psi_{\Sigma,\rho}(\eta)(X) = \|\phi_X\|\rho[\eta]$.

Definition 6. (Solutions) A *solution* of a modal equation system Σ with respect to a model $\mathcal{M}$ and an environment ρ is a map $\eta : \mathsf{bv}(\Sigma) \to \mathcal{P}(S)$ such that $\Psi_{\Sigma,\rho}(\eta) = \eta$. The *semantics* of a modal equation system Σ with respect to $\mathcal{M}$ and ρ, denoted $\|\Sigma\|\rho$, is its greatest solution.

Note that by the well-known Knaster-Tarski fixed point theorem [18] the greatest solution of $\Psi_{\Sigma,\rho}$ always exists, since $\Psi_{\Sigma,\rho}$ is a monotone map on the lattice $\mathcal{P}(S)^{\mathsf{bv}(\Sigma)}$ ordered by point-wise inclusion.

Example 2. For the example above, there is a unique solution $\|\Sigma\|\rho = \{X_1 \mapsto \rho(Y) \cap \rho(Z), X_2 \mapsto \rho(Y), X_3 \mapsto \rho(Z)\}$.

We use modal equation systems to add recursion to basic simulation logic. A formula $\phi[\Sigma]$ of *simulation logic* is composed of a formula ϕ of basic simulation logic and a modal equation system Σ. The free variables of ϕ are interpreted by the greatest solution of Σ. Formally:

Definition 7. (Simulation Logic) The formulae of *simulation logic* over L and A are defined by $\phi[\Sigma]$, where ϕ is a formula of basic simulation logic and Σ is a modal equation system. The set of free and bound variables are $\mathsf{fv}(\phi[\Sigma]) = (\mathsf{fv}(\phi) \cup \mathsf{fv}(\Sigma)) - \mathsf{bv}(\Sigma)$ and $\mathsf{bv}(\phi[\Sigma]) = \mathsf{bv}(\Sigma)$, respectively.

The *semantics* of $\phi[\Sigma]$ with respect to model $\mathcal{M}$ and environment ρ is defined by

$$\|\phi[\Sigma]\|\rho = \|\phi\|\rho[\|\Sigma\|\rho].$$

We say that a state s of a model $\mathcal{M}$ satisfies $\phi[\Sigma]$, written $(\mathcal{M}, s) \models \phi[\Sigma]$, if $s \in \|\phi[\Sigma]\|\rho$ for all ρ. For specifications $(\mathcal{M}, E)$ we define $(\mathcal{M}, E) \models \phi[\Sigma]$ if $(\mathcal{M}, s) \models \phi[\Sigma]$ for all $s \in E$.

We henceforth often omit the equation system Σ from $\phi[\Sigma]$ if no confusion can arise. Further, from now one we restrict our attention to *closed* formulae with no free variables, for which the semantics is independent of the environment.

We say that ϕ_1 is a logical consequence of ϕ_0, written $\phi_0 \models \phi_1$, if for all specifications $\mathcal{S}$, $\mathcal{S} \models \phi_0$ implies $\mathcal{S} \models \phi_1$.

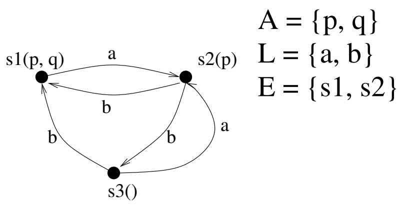

Figure 1: Example specification $\mathcal{S}$

The formula ϕ_0 is logically equivalent to ϕ_1, written $\phi_0 \equiv \phi_1$, if $\phi_0 \models \phi_1$ and $\phi_1 \models \phi_0$.

Simulation logic is equally expressive as the modal μ-calculus [13] with box modalities and greatest fixed points only. The translation from this fragment of the modal μ-calculus to simulation logic is straightforward. As an example, the formula $\nu X.p_1 \wedge (\nu Y.X \wedge (p_2 \vee Y))$ is translated to the equivalent simulation logic formula $X[X = p_1 \wedge Y, Y = X \wedge (p_2 \vee Y)]$. The translation in the other direction is based on Bekič's principle [3, 1], 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 will relate simulation logic to simulation. We proceed by defining two translations: χ and θ. The map χ translates a *finite* specification into a formula, characterising the specification, 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 SNF. We show that χ logically characterises simulation and θ behaviourally characterises logical satisfaction. These two maps form a Galois connection between finite specifications and formulas: $\mathcal{S} \leq \theta(\phi)$ if and only if $\mathcal{S} \models \phi$ if and only if $\chi(\mathcal{S}) \models \phi$.

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 $\Sigma_\mathcal{M}$ is defined by an equation

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

for each $s \in S$, and $\phi_E = \bigvee_{s \in E} X_s$. Recall that we identify $\bigvee \varnothing$ with ff and $\bigwedge \varnothing$ with tt, so for example an empty set of a-transitions from state s will yield the box formula $[a]\,\mathsf{ff}$.

Example 3. To illustrate this definition, suppose we have the specification $\mathcal{S}$ displayed in Figure 1 (where the notation $s_1(p,q)$ is used to denote a state s_1 for which $\lambda(s_1) = \{p,q\}$). The corresponding formula for this model is $\chi(\mathcal{S}) = X_{s_1} \vee X_{s_2}[\Sigma]$, where Σ is given by

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

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][3].

THEOREM 1. *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 $\mathcal{S}_1 \models \chi(\mathcal{S}_2)$.*

Now that we have a translation from finite specifications to formulae, we are interested in defining the inverse mapping. However, not all formulae correspond directly to a specification, but those in so-called simulation normal form do.

Definition 8. (Simulation normal form) A formula $\phi[\Sigma]$ of simulation logic over L and A 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 of Σ have 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 formula $\chi(\mathcal{S})$ is always in SNF. From a formula $(\bigvee \mathcal{X})[\Sigma]$ over L and A 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 the equation for X induces transitions $\{X \xrightarrow{a} Y \mid Y \in \mathcal{Y}_{X,a}\}$ and truth assignment $\lambda(X) = B_X$.

LEMMA 1. *χ and θ are each others inverse up to equivalence, that is,*

(i) $\theta(\chi(\mathcal{S})) \cong \mathcal{S}$ ($\cong$ is isomorphism[4]) for finite $\mathcal{S}$, and

(ii) $\chi(\theta(\phi)) \equiv_\alpha \phi$ ($\equiv_\alpha$ is α-convertibility) for ϕ in SNF.

Finally we are ready to relate simulation logic to simulation.

THEOREM 2. *For ϕ in simulation normal form, we have $\mathcal{S} \leq \theta(\phi)$ if and only if $\mathcal{S} \models \phi$.*

PROOF. Follows from Theorem 1 by Lemma 1(2). ☐

Transformation to SNF

As mentioned above, all formulae of simulation logic have a semantically equivalent simulation normal form. To see this we present a stepwise transformation process of a formula of the form $X_0[\Sigma_0]$ over given sets of labels L and atomic propositions A into simulation normal form. If necessary, we first transform a formula $\phi[\Sigma]$ into the equivalent formula $X[X = \phi, \Sigma]$, where X is fresh. The transformation produces a formula $\psi = (\bigvee \mathcal{X})[\Sigma]$ for a set of variables $\mathcal{X}$ and an equation system Σ such that

(i) ψ is equivalent to $X_0[\Sigma_0]$, and

(ii) ψ is in simulation normal form.

Before describing the transformation in detail, we introduce some auxiliary definitions. First, we need a slightly non-standard variant of disjunctive normal form (DNF).

[3]By using infinite equation systems this theorem easily generalises to finitely branching $\mathcal{S}_2$.

[4]Here, isomorphism means a bijection of states and transitions, but labels have to match on the nose.

Definition 9. (DNF) A formula ϕ of basic simulation logic is in *disjunctive normal form* if it is a disjunction of conjunctions of box formulae and literals, that is, it has the shape

$$\phi = \bigvee_i \bigwedge_j [a_{ij}]\, \psi_{ij} \wedge \bigwedge \mathcal{L}_i$$

where each $\mathcal{L}_i$ is a set of literals and the ψ_{ij}'s are arbitrary formulae in basic simulation logic.

Definition 10. 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 by convention we have $c(\mathsf{tt}) = \varnothing$. We call an occurrence of a subformula *top-level* if it is not under the scope of a box operator. Next, we introduce the notions of variable dependency and guardedness.

Definition 11. (Variable dependency) For $X, Y \in \mathsf{bv}(\Sigma)$, we say X *depends on* Y, written $X \succ Y$, if Y occurs in ϕ_X. We call $(\mathsf{bv}(\Sigma), \succ)$ the variable dependency graph of Σ.

Definition 12. (Guardedness) Let $X, Y \in \mathsf{bv}(\Sigma)$. Variable Y is *guarded* in ϕ_X if all occurrences of Y in ϕ_X are in the scope of a box formula, and is unguarded otherwise. Define the restriction of $\succ$ to unguarded dependencies by $X \succ_u Y$ if $X \succ Y$ and Y occurs unguarded in ϕ. An equation system Σ is called *weakly guarded* if $\succ_u$ is acyclic and *strongly guarded* if $\succ_u$ is empty. A formula $\phi[\Sigma]$ is weakly (strongly) guarded if Σ is weakly (strongly) guarded.

Example 4. Consider the modal equation system

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

Its variable dependency graph is given by $\succ = \{X, Y\} \times \{X, Y\}$. In the equation for X variable X is guarded but Y is not. In the equation for Y both X and Y are guarded. Hence, $\succ_u = \{(X, Y)\}$ being acyclic but not empty, Σ is weakly guarded but not strongly guarded.

LEMMA 2. *Any formula ϕ of simulation logic is equivalent to a weakly and to a strongly guarded formula.*

PROOF. (Sketch) To transform ϕ into a weakly guarded formula we first translate ϕ into a modal μ-calculus formula, use Kozen's procedure to obtain a guarded formula [19], and then translate this back into a weakly guarded formula ϕ_w of simulation logic. In order to obtain a strongly guarded formula from ϕ_w, we repeatedly rewrite unguarded occurrences of variables by their defining equations. This process terminates by weak guardedness of ϕ_w. □

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

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

Phase II eliminates the top-level disjunctions by introducing a new equation for each disjunct, and

Phase III is a cleanup phase that removes duplicated and unreachable equations.

The transformation relies on a partial function h mapping sets of formulae to variables. The purpose of this map is to avoid the repeated introduction of a new equation for the same formula. If h maps a set of formulae Ψ to variable X, this means that at some point a new equation $X = \bigwedge \Psi$ has been added to Σ (in step I.4), so X is reused instead of introducing another equation for $\bigwedge \Psi$. This bookkeeping is essential for the termination of the transformation.

Starting the transformation with the formula $X_0[\Sigma_0]$, we initially have $\mathcal{X} = \{X_0\}$, $\Sigma = \Sigma_0$ and $h = \varnothing$. By Lemma 2 we assume w.l.o.g. that Σ_0 is weakly guarded.

Phase I (Disjunction of state normal forms). All steps of this phase are applied to each equation including the new ones introduced in step I.4 below. Phase I must be completed for all equations before moving on to phase II. The description of most steps ends by an equation indicating the equation shape achieved by the respective transformation step.

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

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

(iii) (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 using $\mathsf{tt} \equiv [a]\, \mathsf{tt}$.

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

(iv) (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 $h \cup \{(c(\psi_{ia}), Z)\}$.

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

(v) (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

$$X = \bigvee_i \bigwedge_{a \in L} [a]\, Z_{ia} \wedge \bigwedge_{p \in B} p \wedge \bigwedge_{q \in A - B} \neg q \tag{4}$$

for some $B \subseteq A$.

Phase II (Push disjunctions inside). Remove an equation of shape $X = \bigvee_i \phi_i$ from Σ (unless there is exactly one disjunct). Add a new equation $X_i = \phi_i$ for each non-variable disjunct ϕ_i of ϕ_X. Then substitute $\bigvee_i X_i$ for X in all equations of Σ, where X_i is a variable disjunct or the fresh variable introduced for ϕ_i. Finally, in case X in $\mathcal{X}$, replace $\mathcal{X}$ by $(\mathcal{X} - \{X\}) \cup \{X_i \mid i\}$.

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

where $\mathcal{Y}_a$ is a (possibly empty) set of variables. Repeat this step until all equations are in SNF.

Phase III (Cleanup). This optimisation phase iteratively removes duplicated and unreachable equations.

(i) If there are equations $Z_1 = \psi_1$ and $Z_2 = \psi_2$ in Σ such that $\psi_2[Z_1/Z_2] = \psi_1$, then remove $Z_2 = \psi_2$ from Σ and substitute Z_1 for Z_2 in the remaining equations as well as in $\mathcal{X}$.

(ii) Remove an equation $Z = \psi$ from Σ in case Z can not be reached from any variable in $\mathcal{X}$ via the variable dependency graph.

The algorithm transforms any formula ϕ into its simulation normal form, denoted by $\mathsf{snf}(\phi)$.

Remark 1. There are many ways to optimise the translation above. For example, steps III.1-2 of phase III can be applied at any point of the transformation if h is updated in the obvious way. However, for the sake of a clearer presentation, we only do these optimisations at the end.

Example 5. Here we present a very simple example of the transformation to SNF. A more elaborate example appears in Section 4. Let $\phi = \mathsf{tt}$ be interpreted as a formula over $L = \{a, b\}$ and $A = \{p\}$. We first translate this to $X[\Sigma_0]$ with $\Sigma_0 = \{X = \mathsf{tt}\}$. This is clearly strongly guarded (I.1) and in DNF (I.2). Box completion (step I.3) transforms the equation to $X = [a]\,\mathsf{tt} \wedge [b]\,\mathsf{tt}$. Then step I.4 produces $\Sigma = \{X = [a]\,Y \wedge [b]\,Y, Y = \mathsf{tt}\}$ and $h = \{(\varnothing, Y)\}$ (recall $c(\mathsf{tt}) = \varnothing$). Applying steps I.1-3 to Y yields $Y = [a]\,\mathsf{tt} \wedge [b]\,\mathsf{tt}$. Step I.4 then yields $Y = [a]\,Y \wedge [b]\,Y$, since $h(\varnothing) = Y$. Now we have $\phi_X = \phi_Y[X/Y]$, so, using remark 1, we drop the equation for Y and get $\Sigma = \{X = [a]\,X \wedge [b]\,X\}$. Step I.5 turns this into $\Sigma = \{X = ([a]\,X \wedge [b]\,X \wedge p) \vee ([a]\,X \wedge [b]\,X \wedge \neg p)\}$. Phase II produces $\mathcal{X} = \{X_1, X_2\}$ and

$$\Sigma = \begin{bmatrix} X_1 & = & [a]\,(X_1 \vee X_2) \wedge [b]\,(X_1 \vee X_2) \wedge p \\ X_2 & = & [a]\,(X_1 \vee X_2) \wedge [b]\,(X_1 \vee X_2) \wedge \neg p \end{bmatrix}$$

which is in SNF, so $\mathsf{snf}(\phi) = (X_1 \vee X_2)[\Sigma]$.

THEOREM 3. *For any formula of simulation logic, there is a logically equivalent formula in simulation normal form.*

PROOF. By Lemma 2, we assume w.l.o.g. that the equations in Σ_0 are weakly guarded. We think of the transformation as producing a series of triples $(\mathcal{X}_i, \Sigma_i, h_i)$ consisting of the current values of $\mathcal{X}$, Σ and h after each transformation step. We have to show that

(i) each of the transformation steps preserves the semantics of the formula, that is, $(\bigvee \mathcal{X}_i)[\Sigma_i] \equiv X_0[\Sigma_0]$

(ii) the transformation terminates after, say, n steps with $(\bigvee \mathcal{X}_n)[\Sigma_n]$ in SNF.

We concentrate here on showing that phase I preserves the meaning of formulae and terminates with all equations in shape (4). It is then not difficult to see that phases II and III terminate and produce an equivalent formula in SNF.

Partial correctness. For phase I we establish 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 $Z \in \mathsf{bv}(\Sigma_i)$ and $Z[\Sigma_i] \equiv (\bigwedge \Psi)[\Sigma_i]$.

The preservation of meaning (point 1) in phase I is then a consequence of J1 - since $\mathcal{X}$ remains unchanged, while J2 is an auxiliary invariant expressing a property of each h_i, which will be needed to establish invariant J1.

Both invariants hold trivially for $i = 0$. We have to check that they are preserved by steps I.1-I.5. This is easy to see for steps I.2 and I.5, which are justified by propositional equivalences and the fact that $h_{i+1} = h_i$. Similarly, for step I.3 which is based on modal equivalences. We discuss the proof for the remaining steps I.1 and I.4 in some detail.

Given equation $(X = \phi) \in \Sigma_i$, in step I.1 we repeatedly rewrite unguarded occurrences of variables in ϕ using equations of Σ_0. Suppose we rewrite equation $(Y = \psi) \in \Sigma_0$ in ϕ. By the fixed point property we have $Y[\Sigma_0] \equiv \psi[\Sigma_0]$. It then follows by invariant J1 and a routine induction on the structure of ψ that $Y[\Sigma_i] \equiv \psi[\Sigma_i]$. Hence, $X[\Sigma_i] \equiv X[\Sigma_i']$, where Σ_i' is obtained from Σ_i by replacing $X = \phi$ with $X = \phi[\psi/Y]$. By repeatedly applying this argument we get $X[\Sigma_i] \equiv X[\Sigma_{i+1}]$, and thus the preservation of J1. Since $h_{i+1} = h_i$, invariant J2 is also preserved.

In step I.4 we consider an equation $(X = \phi) \in \Sigma_i$ and a top-level occurrence of a subformula $[a]\,\psi$ in ϕ such that ψ is not a variable. Two cases are distinguished. If $(c(\psi), Y) \in h_i$ then $\Sigma_i' = (\Sigma_i - \{X = \phi\}) \cup \{X = \phi'\}$, where the given occurrence of $[a]\,\psi$ in ϕ is replaced by $[a]\,Y$ in ϕ'. By invariant J2 we know that $Y[\Sigma_i] \equiv \psi[\Sigma_i]$, so both J1 and J2 are preserved. If, on the other hand, $c(\psi) \notin \mathsf{dom}(h_i)$ then we set $\Sigma_i' = (\Sigma_i - \{X = \phi\}) \cup \{X = \phi', Z = \psi\}$, where the given occurrence of $[a]\,\psi$ in ϕ is replaced by $[a]\,Z$ in ϕ' and $Z \notin \mathsf{bv}(\Sigma_i)$ is a fresh variable. We also set h_i' to $h_i \cup \{(c(\psi), Z)\}$. Invariant J1 is preserved. In particular, $X[\Sigma_i] \equiv X[\Sigma_i']$, as we are just introducing a new name for the subformula ψ of ϕ. Invariant J2 certainly holds for the new mapping $(c(\psi), Z) \in h_i'$ and it holds for all elements in h_i by J1. Again, by repeating this argument one can easily see that both invariants hold for $(\mathcal{X}_{i+1}, \Sigma_{i+1}, h_{i+1})$.

Termination. It remains to be proven that phase I terminates. There are two points to consider. First, the rewriting process in step I.1 might fail to terminate. However, this is prevented by our assumption that the initial equation system Σ_0 is weakly guarded (see also Lemma 2). All transformation steps preserve the weak guardedness of equations. In particular, any new equation introduced in step I.4 is weakly guarded, since it simply gives name to a subformula of an existing equation and does therefore not add any cycles to the variable dependency graph.

The second potential source of non-termination is the introduction of new equations in step I.4. Let us call a formula ψ *boxed in* ϕ if there is some $a \in L$ such that $[a]\,\psi$ is a subformula of ϕ. Non-termination of step I.4 is ruled out by the following invariant:

J3. whenever $(\Psi, Z) \in h_i$ then each $\psi \in \Psi$ appears boxed in the right-hand side of an original equation in Σ_0.

Since there are only a finite number of boxed subformulae, the map h eventually fills up into a total function at which point no further equations will be added. Then phase I terminates, since each of the individual steps does.

To see that J3 holds, observe that steps I.1 and I.2 do not affect boxed formulae and in step I.3 boxed formulae might be combined into conjunctions. Therefore, these steps preserve the property

P. all boxed formulae of an equation $(X = \phi) \in \Sigma_i$ are conjunctions of boxed formulae of Σ_0.

This property trivially holds initially for the equations in Σ_0. But then it also holds for any new equations introduced in step I.4. Hence, any pair (Ψ, Z) that is added to h_i satisfies invariant J3. This completes the proof of termination. $\square$

We extend the mapping θ to all formulae of simulation logic by defining $\theta(\phi) = \theta(\mathsf{snf}(\phi))$. Then, since snf preserves the semantics of formulae, Theorem 2 can be extended to all formulae.

THEOREM 4. *For all specifications $\mathcal{S}$ and formulae ϕ, we have $\mathcal{S} \leq \theta(\phi)$ if and only if $\mathcal{S} \models \phi$.*

Thus, for any specification $\mathcal{S}$ and any property ϕ, if we want to check whether the specification satisfies the property, it is sufficient to check that $\mathcal{S}$ is simulated by $\theta(\phi)$.

Consequences. We mention a few consequences of Theorems 1 and 4. Let $(\mathsf{S}, \leq)$ be the preorder of (isomorphism classes of) *finite* specifications over given L and A ordered by simulation and let $(\mathsf{L}, \models)$ be the preorder of formulae of simulation logic over L and A ordered by the logical consequence relation.

COROLLARY 1. *χ and θ are monotone.*

Simulation preserves logical properties:

COROLLARY 2. *For all specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ we have $\mathcal{S}_1 \leq \mathcal{S}_2$ and $\mathcal{S}_2 \models \phi$ imply $\mathcal{S}_1 \models \phi$.*

The pair (χ, θ) of maps forms a Galois connection between the preorders $(\mathsf{L}, \models)$ and $(\mathsf{S}, \leq)$:

COROLLARY 3. *For finite specifications $\mathcal{S}$ and all formulas ϕ, we have $\mathcal{S} \leq \theta(\phi)$ if and only if $\chi(\mathcal{S}) \models \phi$.*

We end this section by remarking that these results can be transferred to the setting of weak simulation and logic.

3. COMPOSITIONAL VERIFICATION OF APPLETS

So far, all our results have been developed for arbitrary specifications. From now on, we shall concentrate on a particular application, namely the representation of applets (*i.e.* smart card applications) in our general notion of specification. 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:

(i) the structural level, where a specification represents the control flow graph of a method, and applets are viewed as collections of methods, with an appropriate interface;

(ii) the behavioural level, where a specification represents the behaviour of an applet.

This yields a version of simulation logic for each level. The compositional verification principle that we develop allows us to state assumptions about individual applets in the structural simulation logic, in order to establish behavioural simulation logic properties for the composed system. For this and the next section, it will be convenient to make the following assumption.

Assumption 1. (Reachability) In all specifications $(\mathcal{M}, E)$, each state of $\mathcal{M}$ is reachable from a state in E.

Note that all results of Section 2 carry over to this restricted setting.

3.1 Applets

Applets are defined as a collection of methods, where each method is an instance of a model. However, for a realistic program model of applets, it is necessary to know which methods exist and/or are used. Therefore, we first define the notion of an applet interface, which specifies which methods are provided and required by an applet.

Let $\mathcal{M}eth$ be a countably infinite set of method names.

Definition 13. (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 $I_1 \cup I_2 = (I_1^+ \cup I_2^+, I_1^- \cup I_2^-)$.

As mentioned above, methods are an instance of the general notion of specification.

Definition 14. (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, $L_m = M \cup \{\varepsilon\}$, $A_m = \{m, r\}$, $\lambda_m^*(m) = V_m$, *i.e.* each node is tagged with the method name, and $\lambda_m^*(r) \subseteq V_m$ is a set of *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.

An applet is basically a collection of method specifications. For the formal definition we extend the notion of disjoint union from models (as defined for simulation between models, below Definition 2) to specifications: given specifications $(\mathcal{M}_1, E_1)$ and $(\mathcal{M}_2, E_2)$ define $(\mathcal{M}_1, E_1) \uplus (\mathcal{M}_2, E_2) = (\mathcal{M}_1 \uplus \mathcal{M}_2, in_1(E_1) \cup in_2(E_2))$ where in_1 and in_2 are injections from states of $\mathcal{M}_1$ and $\mathcal{M}_2$ to states of $\mathcal{M}_1 \uplus \mathcal{M}_2$, respectively.

Definition 15. (Applet) An *applet* $\mathcal{A}$ with interface I, written $\mathcal{A} : I$, is defined inductively by

- $\varnothing_M : (\varnothing, M)$, where $\varnothing_M = ((\varnothing, M \cup \{\varepsilon\}, \varnothing, \{r\}, \varnothing), \varnothing)$ is the *empty applet* over M,
- $(\mathcal{M}_m, E_m) : (\{m\}, M)$ 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 $\mathcal{A} : (I^+, I^-)$ is *closed* if $I^- \subseteq I^+$.

Note that, up to isomorphism, $\uplus$ is associative and commutative and has the neutral element $\varnothing_M$. For an arbitrary specification $(\mathcal{M}, E)$ we say that it is an applet with interface I if and only if we can decompose $(\mathcal{M}, E)$ following this definition. Notice that if $\mathcal{A} : I$, then for each method $m \in I^+$ the applet has to contain a corresponding method graph. Thus, an applet can only provide methods that it actually implements.

LEMMA 3. *Suppose $\mathcal{A} = (\mathcal{M}, E)$ is an applet with $\mathcal{M} = (V, L, \rightarrow, A, \lambda)$. Then its interface is $(A - \{r\}, L - \{\varepsilon\})$.*

PROOF. By induction on the structure of the applet. □

3.2 Structural Level

As mentioned above, we express properties over applets at two levels: structural and behavioural. The next subsection considers behavioural properties, here we study structural ones, *i.e.* properties that allow one to restrict the possible method graphs in an applet.

Simulation. An applet $\mathcal{A}_1$ structurally simulates another applet $\mathcal{A}_2$ if each entry point for a method m of $\mathcal{A}_1$ is simulated by some entry point for the same method of $\mathcal{A}_2$. Thus, structural simulation coincides directly with simulation on models, as defined above. 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 5. (Struct-Mon) *If $\mathcal{A}_1 \leq_s \mathcal{B}_1$ and $\mathcal{A}_2 \leq_s \mathcal{B}_2$ then $\mathcal{A}_1 \uplus \mathcal{A}_2 \leq_s \mathcal{B}_1 \uplus \mathcal{B}_2$.*

PROOF. Suppose R_1 and R_2 are witnesses of $\mathcal{A}_1 \leq_s \mathcal{B}_1$ and $\mathcal{A}_2 \leq_s \mathcal{B}_2$, respectively. Then $R = \{((s,i),(t,i)) \mid i \in \{1,2\} \wedge (s,t) \in R_i\}$ is a simulation between $\mathcal{A}_1 \uplus \mathcal{A}_2$ and $\mathcal{B}_1 \uplus \mathcal{B}_2$. □

Logic. We also instantiate the notion of satisfaction to the structural level. An applet satisfies a formula of simulation logic if all its entry points satisfy the formula. For clarity, we define *structural satisfaction* $\mathcal{A} \models_s \phi$ as $\mathcal{A} \models \phi$.

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

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

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

Example 6. Given interface $I = (\{m_1, m_2\}, \{m_1, m_3\})$, the basic structure of applets with interface I is characterised by the formula $\phi_I[\Sigma_I]$, where $\phi_I = X_{m_1} \vee X_{m_2}$ and

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

The following proposition characterises applets interface I as those specifications that satisfy the characteristic formula ϕ_I for interface I.

PROPOSITION 1. *Let $I = (I^+, I^-)$ be an applet interface, let $(\mathcal{M}, E)$ be any specification over $L = I^- \cup \{\varepsilon\}$ and $A = I^+ \cup \{r\}$. We have $(\mathcal{M}, E) \models_s \phi_I[\Sigma_I]$ if and only if $(\mathcal{M}, E)$ is an applet with interface I,* i.e. $(\mathcal{M}, E) : I$.

Finally, using the characteristic formula for interface I, we define the maximal applet with interface I *w.r.t.* a formula $\phi[\Sigma]$ as the conjunction of $\phi[\Sigma]$ and $\phi_I[\Sigma_I]$.

Definition 16. (Maximal applet) The *maximal applet* w.r.t. interface I satisfying $\phi[\Sigma]$ 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).

Notice that by Proposition 1 $\theta_I(\phi) : I$, *i.e.* the maximal applet has interface I. Also, by definition $\theta_I(\phi[\Sigma]) \models_s \phi[\Sigma]$, since $\phi \wedge \phi_I[\Sigma, \Sigma_I] \models \phi[\Sigma]$. Finally, we show that to prove that an applet satisfies formula $\phi[\Sigma]$, it is sufficient to show that it is simulated by the maximal applet *w.r.t.* $\phi[\Sigma]$.

THEOREM 6. (Struct-Max) *Let $\mathcal{A} : I$ be an applet. Then $\mathcal{A} \leq_s \theta_I(\phi[\Sigma])$ if and only if $\mathcal{A} \models_s \phi[\Sigma]$.*

PROOF. By Proposition 1, for applet $\mathcal{A} : I$, $\mathcal{A} \models_s \phi[\Sigma]$ if and only if $\mathcal{A} \models_s \phi \wedge \phi_I[\Sigma, \Sigma_I]$. The result then follows from Theorem 4. □

3.3 Behavioural Level

Next, we change our focus to properties on the behavioural level. The advantage of also having this level for specifications is that it allows us to write more abstract specifications. The first step is to define the behaviour of an applet.

Definition 17. (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 as follows.

- $S_b = V \times V^*$, *i.e.* 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 rules of Table 1;
- $A_b = A$; and
- $p \in \lambda_b((v, \sigma))$ if $p \in \lambda(v)$ for $p \in A$.

The set of initial states E_b is defined by $E_b = E \times \{\varepsilon\}$.

Note that applet behaviour defines a context-free process (see, e.g., [6] for an survey of infinite process structures).

Simulation. Also on the behavioural level, we instantiate the general definition of simulation. Applet $\mathcal{A}_1$ *behaviourally simulates* applet $\mathcal{A}_2$, written $\mathcal{A}_1 \leq_b \mathcal{A}_2$, if $b(\mathcal{A}_1) \leq b(\mathcal{A}_2)$. Any two applets that are related by structural simulation, are also related by behavioural simulation.

THEOREM 7. (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 $\mathcal{A}_1 = (\mathcal{M}_1, E_1)$ and $\mathcal{A}_2 = (\mathcal{M}_2, E_2)$. We lift R on the structural level to R_b on the behavioural level by defining

$$\begin{array}{rl} R_b = \{(v, \sigma), (v', \sigma') \mid & (v, v') \in R,\ |\sigma| = |\sigma'| \text{ and} \\ & (\sigma(i), \sigma'(i)) \in R \text{ for all } 0 \leq i < |\sigma|\} \end{array}$$

It is then easy to check that R_b is a behavioural simulation between $\mathcal{A}_1$ and $\mathcal{A}_2$. □

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

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

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

Table 1: Applet Transition Rules

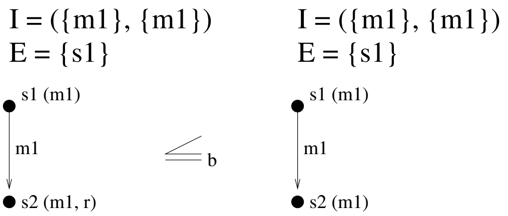

Figure 2: Counterexample, reverse of Theorem 9

The reverse is not the case. Consider for example the two applets in Figure 2. The left applet is behaviourally simulated by the right applet (in fact, they are behaviourally equivalent), but there is no structural simulation between these applets - in any direction - since in the left applet the state s_2 satisfies the atomic predicate r, while in the right applet it does not.

Logic. Finally, we instantiate simulation logic on the behavioural level by defining *behavioural satisfaction* $\mathcal{A} \models_b \psi$ as $b(\mathcal{A}) \models \psi$.

3.4 Compositional Reasoning

Having instantiated the results from Section 2 both at the structural and at the behavioural level, we are now ready to relate the two. The main result of this paper is the following *compositional reasoning principle*. Let $\mathcal{A} : I$ and $\mathcal{B} : J$ be applets, let ϕ be a formula in structural simulation logic, and ψ be a formula in behavioural simulation logic. Then we have

$$\frac{\mathcal{A} \models_s \phi \qquad \theta_I(\phi) \uplus \mathcal{B} \models_b \psi}{\mathcal{A} \uplus \mathcal{B} \models_b \psi} \ (\mathsf{beh\text{-}comp})$$

This principle says that in order to show that a a composed applet $\mathcal{A} \uplus \mathcal{B}$ has a behavioural property ψ, it is sufficient to find a structural property ϕ, satisfied by $\mathcal{A}$, such that $\theta_I(\phi) \uplus \mathcal{B} \models_b \psi$. By using the characteristic formula $\chi(\mathcal{A})$ as instantiation for ϕ we can also show that this principle is complete.

THEOREM 8. (Soundness and Completeness) *Suppose $\mathcal{A}$: I. Let ψ be a behavioural formula. Then there is a structural formula ϕ such that $\mathcal{A} \models_s \phi$ and $\theta_I(\phi) \uplus \mathcal{B} \models_b \psi$ if and only if $\mathcal{A} \uplus \mathcal{B} \models_b \psi$.*

PROOF. "$\Rightarrow$" Suppose $\mathcal{A} \models_s \phi$ and $\theta_I(\phi) \uplus \mathcal{B} \models_b \psi$. By Theorem 6 and the first assumption, we have $\mathcal{A} \leq_s \theta_I(\phi)$. It follows that $\mathcal{A} \uplus \mathcal{B} \leq_b \theta_I(\phi) \uplus \mathcal{B}$ by Theorems 5 and 7. Hence, $\mathcal{A} \uplus \mathcal{B} \models_b \psi$ by Corollary 2 (instantiated to the behavioural level) and the second assumption.

"$\Leftarrow$" Suppose $\mathcal{A} \uplus \mathcal{B} \models_b \psi$ and set $\phi = \chi(\mathcal{A})$. We have to show that $\mathcal{A} \models_s \chi(\mathcal{A})$ and $\theta_I(\chi(\mathcal{A})) \uplus \mathcal{B} \models_b \psi$. The former follows from Theorem 1 (for $\mathcal{S}_1 = \mathcal{S}_2$, instantiated to structural level). To see the latter, we start by the observation that $\chi(\mathcal{A}) \wedge \phi_I[\Sigma_I] \models_s \chi(\mathcal{A})$. By the monotonicity of θ (Corollary 1), we get $\theta_I(\chi(\mathcal{A})) \leq \theta(\chi(\mathcal{A}))$. Lemma 1 states that $\theta(\chi(\mathcal{A})) \cong \mathcal{A}$. Hence, using the definition of structural simulation, $\theta_I(\chi(\mathcal{A})) \leq_s \mathcal{A}$. It follows by Theorems 5 and 7 that $\theta_I(\chi(\mathcal{A})) \uplus \mathcal{B} \leq_b \mathcal{A} \uplus \mathcal{B}$. Finally, Corollary 2 and the assumption imply that $\theta_I(\chi(\mathcal{A})) \uplus \mathcal{B} \models_b \psi$. $\square$

Note that by taking $\mathcal{B}$ to be the empty applet $\varnothing_{J^-}$, the compositional reasoning principle above relates behavioural properties to structural ones. Given applet $\mathcal{A} : I$, the satisfaction of behavioural property ψ can be reduced to the satisfaction of structural property ϕ if and only if the maximal applet *w.r.t.* I and ϕ (behaviourally) satisfies property ψ.

$$\frac{\mathcal{A} \models_s \phi \qquad \theta_I(\phi) \models_b \psi}{\mathcal{A} \models_b \psi} \ (\mathsf{struct\text{-}beh})$$

Other useful compositional reasoning principles are also thinkable. For example, a rule of the shape of the above rule, but involving structural properties only, is easily justifiable with the results presented above.

$$\frac{\mathcal{A} \models_s \phi \qquad \theta_I(\phi) \uplus \mathcal{B} \models_s \psi}{\mathcal{A} \uplus \mathcal{B} \models_s \psi} \ (\mathsf{struct\text{-}comp})$$

Apart from being able to show soundness and completeness for this rule (using a similar proof as for the rule beh-comp), we can also show that it possesses the weakest-cut property.

4. EXAMPLE

Finally, to demonstrate the use of our approach in practice, we present a small example. This example is a smaller, distilled version of a larger case study on verification of behavioural safety properties for an electronic purse. This case study is described elsewhere in more detail [10], and we refer to this paper for a more detailed motivation why this kind of security properties are important for smart card applications and how they should be formalised.

Suppose we have a smart card, on which we allow instances of applets $\mathcal{A}$ and $\mathcal{B}$ with the following interfaces: $\mathcal{A}$: $(\{m_1, m_2\}, \{m_1, m_2, m_3\})$ and $\mathcal{B} : (\{m_3\}, \{m_1, m_2, m_3\})$, respectively. Now, suppose that the method m_1 is a method that is called by an instance of the applet $\mathcal{B}$ when it is in a particular state. However, it might be the case that only certain instances of applet $\mathcal{A}$ are supposed to know that this instance of $\mathcal{B}$ is in this state - possibly because they have paid to get this information. Thus, as a global security property we require that when method m_1 is called, this does not trigger any other calls to instances of $\mathcal{A}$ (until the method has finished). We specify this as the following global security property.

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

where $K = \{\ m_1 \,\mathsf{call}\, m_1, m_1 \,\mathsf{call}\, m_2, m_2 \,\mathsf{call}\, m_1, m_2 \,\mathsf{call}\, m_2, m_3 \,\mathsf{call}\, m_1, m_3 \,\mathsf{call}\, m_2\ \}$. Notice that this can be considered as a confidentiality property: it prevents certain information to flow to unauthorised applications [4].

This formula expresses that within method m_1, *i.e.* until the model describing the behaviour of the applet reaches a state satisfying $m_1 \wedge r$, there cannot be any calls to other methods declared in the interface of $\mathcal{A}$. Notice that this also restricts the calls that can be made from an instance of applet $\mathcal{B}$, *i.e.* m_3 is also not allowed to call a method declared in the interface of $\mathcal{A}$. To make it easier to express this kind of properties, we are developing a common set of specification patterns, with appropriate definitions as modal μ-calculus formulae.

There are several ways in which this property can be established. A trivial one is by specifying that method m_1 should not make any method calls. However, this would exclude many sensible implementations of m_1, therefore we prefer to be less restrictive, and we propose the following structural specifications for $\mathcal{A}$ and $\mathcal{B}$.

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

The specification for $\mathcal{A}$ expresses that the method graph for m_1 should not contain any call edge labelled m_1 or m_2. The specification for $\mathcal{B}$ expresses a similar property for the method graph of m_3.

Applying the compositional reasoning principle beh-comp twice, we know that for any instances of applets $\mathcal{A}$ and $\mathcal{B}$, in order to prove that their composition $\mathcal{A} \uplus \mathcal{B}$ respects the property φ, it is sufficient to prove for these instances that $\mathcal{A} \models_s \sigma_{\mathcal{A}}$ and $\mathcal{B} \models_s \sigma_{\mathcal{B}}$, and finally, that $\theta_{I_{\mathcal{A}}}(\sigma_{\mathcal{A}}) \uplus \theta_{I_{\mathcal{B}}}(\sigma_{\mathcal{B}}) \models_b \varphi$. The properties for the individual applets can be checked using existing model checking techniques, so here we focus on the proof of the last sequent, using the maximal model construction presented earlier. Following Definition 16, the maximal applet is constructed as follows:

(i) take the conjunction of the structural specification and the characteristic formula for the given interface;

(ii) transform the resulting formula into simulation normal form; and

(iii) use the mapping θ to construct the applet corresponding to this formula in SNF.

We present in some detail the construction of the maximal applet for $\sigma_{\mathcal{B}}$; the construction of the maximal applet for $\sigma_{\mathcal{A}}$ is similar. As a first step, the characteristic formula for interface $I_{\mathcal{B}}$ is the following.

$$(\phi_{I_{\mathcal{B}}}) \quad X_{m_3}[X_{m_3} = [m_1, m_2, m_3, \varepsilon]X_{m_3} \wedge m_3]$$

Thus, the conjunction with $\sigma_{\mathcal{B}}$ gives the following formula, after desugaring.

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

The next step is to transform this formula into SNF. First, in phase 1 of the transformation, each equation is transformed into a disjunction of state normal forms. Suppose that we start with the equation defining Z.

(i) Make the equation strongly guarded, by rewriting with the original equation system.

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

(ii) Put the equation into DNF

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

and simplify

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

(iii) Group and complete boxes. Here no boxes are missing, therefore we only group boxes.

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

(iv) Introduce new equations for formula under boxes. Since the map h does not contain a mapping for $\{Y, X_{m_3}\}$ yet, we introduce a new variable U and add the mapping $(\{Y, X_{m_3}\}, U)$ to h. The equation defining Z then becomes

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

while we introduce the equation

$$U = Y \wedge X_{m_3}$$

(v) Finally, we complete the equation for Z by adding missing literals and putting the formula into DNF again. In this case, the only literal that is missing is r. Adding this literal gives us the following result.

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

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 1 is applied to the new equation $U = Y \wedge X_{m_3}$.

(i) Rewriting into strongly guarded form gives

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

(ii) The formula is already in DNF, and cannot be simplified.

(iii) Grouping boxes results in the following.

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

(iv) The mapping h contains a map $(\{Y, X_{m_3}\}, U)$. Therefore, we replace $Y \wedge X_{m_3}$ by U.

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

(v) Literal completion again introduces r.

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

Now, phase I has been completed for all equations. Phase II introduces a single equation for each disjunction, and it replaces the variables by the disjunctions. For example, the equation defining U gets replaced by the two following equations.

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

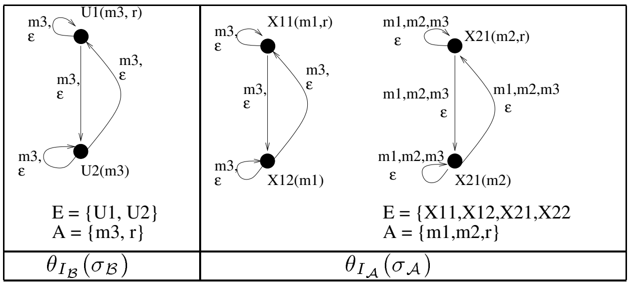

Figure 3: Maximal applets for $\sigma_{\mathcal{B}}$ and $\sigma_{\mathcal{A}}$

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.

Finally, we are ready for the cleanup 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 we replace $\{Z_1, Z_2\}$ in $\mathcal{X}$ by $\{U_1, U_2\}$. We also find that the equations Y_1, Y_2, $X_{m_3 1}$ and $X_{m_3 2}$ (replacing Y and X_{m_3} in Phase II, respectively), are not reachable from any variable in $\mathcal{X} = \{U_1, U_2\}$. Therefore, the final result of the transformation is the following formula.

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

Figure 3 displays the maximal applet corresponding to this equation system (in its left column). Also, it displays the maximal applet found for the property $\sigma_{\mathcal{A}}$, which is found in a similar way. Using a model checking algorithm for context-free processes [7], it can easily be verified that the composition of these two maximal applets indeed satisfies the global behavioural specification ϕ, and thus that the property decomposition is correct.

5. CONCLUSIONS

We propose a compositional verification method for control flow based safety properties of smart card applets. Our method supports different scenarios for secure post-issuance loading of applets. Local applet assumptions are structural, while global guarantees are behavioural, both written in a modal logic with greatest fixed point recursion.

In a general setting, we establish the correspondence between models (which can be structures as well as behaviours) and properties by means of a Galois connection. Maximal (or characteristic) models are used to algorithmically decide correctness of property decompositions by reducing the problem to a standard model checking problem for context-free processes. A distilled version of a realistic case study illustrates the practical applicability of the approach.

Future work will focus on extending our results in two directions: (i) adding diamond modalities to the simulation logic, and (ii) investigating under what restrictions the proposed method can be adapted to behavioural assumptions.

6. REFERENCES

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

[2] 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 (FASE'02)*, number 2306 in LNCS, pages 15–32. Springer, 2002.

[3] H. Bekič. Definable operators in general algebras, and the theory of automata and flowcharts. Technical report, IBM Laboratory, 1967.

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

[6] 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.

[7] O. Burkart and B. Steffen. Model checking for context-free processes. In *CONCUR 92*, number 630 in LNCS, pages 123–137. Springer, 1992.

[8] E. Clarke and E. Emerson. Design and synthesis of synchronisation skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, number 131 in LNCS, pages 52–71. Springer, 1981.

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

[10] D. Gurov, G. Chugunov, M. Huisman, and C. Sprenger. Checking absence of illicit applet interactions: a case study, 2003. Manuscript.

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

[12] T. Jensen, D. L. Métayer, and T. Thorn. Verification of control flow based security policies. In *IEEE Symposium on Research in Security and Privacy*, pages 89–103. IEEE Computer Society Press, 1999.

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

[14] O. Kupferman and M. Vardi. Model checking of safety properties. In *CAV'99*, number 1633 in LNCS, pages 172–183. Springer, 1999.

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

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

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

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

[19] I. Walukiewicz and D. Niwiński. Games for the μ-calculus. *TCS*, 163:99–116, 1997.