Compositional Verification of Secure Applet Interactions

Gilles Barthe[1], Dilian Gurov[2], and Marieke Huisman[1]

[1] INRIA Sophia-Antipolis, France
{Gilles.Barthe,Marieke.Huisman}@sophia.inria.fr
[2] Swedish Institute of Computer Science
dilian@sics.se

Abstract. Recent developments in mobile code and embedded systems have led to an increased interest in open platforms, *i.e.* platforms which enable different applications to interact in a dynamic environment. However, the flexibility of open platforms presents major difficulties for the (formal) verification of secure interaction between the different applications. To overcome these difficulties, compositional verification techniques are required.
This paper presents a compositional approach to the specification and verification of secure applet interactions. This approach involves a compositional model of the interface behavior of applet interactions, a temporal logic property specification language, and a proof system for proving correctness of property decompositions. The usability of the approach is demonstrated on a realistic smartcard case study.

1 Introduction

Verification Techniques for Open Platforms. Open platforms allow different software components, possibly originating from different issuers, to interact easily in a single environment. Thanks to their flexibility, such open platforms are becoming pervasive in modern software for mobile code, but also for embedded devices such as smartcards.

Quite unsurprisingly, the flexibility of open platforms raises major difficulties when it comes to establishing global properties of their applications. Such properties, which capture the interface behavior of the platform's components, include many common security properties such as "Component B can only access resource R after being authorized by Component A", or "Component A cannot perform action α between Component B performing action β and Component C performing action γ".

Two problems arise with the verification of such global properties:

– the complexity of the platform. In order to reason about the system, one needs to specify the communication mechanisms supported by the platform. These can be intrinsic (*e.g.* in Java[1] with privileged instructions and visibil-

[1] See http://java.sun.com/java2.

R.-D. Kutsche and H. Weber (Eds.): FASE 2002, LNCS 2306, pp. 15–32, 2002.

ity modifiers; in JavaCard[2], with firewalls and secure object sharing mechanisms) and can complicate reasoning substantially;
- the availability of the software. In the case of platforms supporting dynamic loading of software, one often would like to establish properties that are preserved when new software is loaded. In particular, this is true for security properties such as confidentiality and integrity.

These problems can be tackled by enforcing the desired properties through strict local security checks, as is done *e.g.* in bytecode verification, see *e.g.* [13], or in type systems for information flow, see *e.g.* [17]. However, this requires focusing on a very restricted set of properties, that excludes many useful global properties.

Open Platforms for Smartcards. New generation smartcards such as JavaCards are open platforms that support multiple applications on a single card and post-issuance loading of applets (*i.e.* applications can be loaded on the card after being issued to users). As smartcards are typically used as identity documents and money devices, security issues are particularly at stake, and the need for formal verification is widely recognized, as testified *e.g.* by Common Criteria[3].

Despite the advent of on-card bytecode verification [14], current technology prevents complex verifications to be performed on-card, thus applet verification needs to be performed off-card, presumably prior to loading the applet on the card. In this setting, one needs to analyze the possible interactions between the applet being loaded and the applets already on the card without having access to the code of the latter.

Compositional Verification. One possible verification strategy for programs operating on open platforms consists of:

1. reducing global properties of programs to local properties about their components, using compositional verification techniques;
2. verifying local properties of components by standard means, such as model-checking.

Such a strategy can be used to control state-space explosion and has been employed to good effect to establish the correctness of realistic, industrial-size open distributed telecom systems [9]. The main goal of this paper is to show that such a strategy is also applicable to open platforms such as smartcards. To this end, we develop a framework which allows to reduce global properties to local properties about components. The problem of verifying the local properties is not addressed here, since there are standard algorithmic techniques for this (see *e.g.* [4]). Our framework for compositional verification consists of:

- a model of applet interactions that captures, in a language-independent setting, control flow within applets and procedure calls between applets. This model is inspired by [11], and was motivated and presented informally by the present authors in [2].

[2] See `http://java.sun.com/javacard`.
[3] See `http://www.commoncriteria.org`.

- a specification language based on the modal μ-calculus [12,8], together with a set of specification patterns, inspired from Bandera [1], that allows higher-level specifications and reasoning;
- a compositional proof system in the style of [16,7,9] that is used for proving correctness of property decompositions. This proof system has been proved sound *w.r.t.* the underlying model in PVS [15].

To illustrate the benefits of our method, we also detail an example of property decomposition in the setting of an industrial smartcard case study [3].

Contents. The remainder of the paper is organized as follows. The model, specification language and proof system are introduced in Sections 2, 3 and 5 respectively, whereas Section 4 provides a brief overview of the language of patterns. The case study is detailed in Section 6. Finally, Section 7 concludes with related and future work.

2 Program Model

We focus on the control flow in platforms in which procedure or method calls are the primary means for interaction between components. For proving validity of property decompositions we need a model of program behavior which captures the interaction behavior of programs and program components, and over which the formulae of property specification languages can be interpreted. Standard models of this kind are provided by labeled transition systems (LTS), where the transition labels denote method invocations and returns. Interaction behavior can then be defined in an abstract and language independent fashion, following the approach by Jensen *et al.* [11], as being induced by a transfer/call graph through a set of transition rules. Composition of behaviors is obtained in process algebraic style by using imperfect actions which handshake to produce perfect communications. The program model and its operational semantics have been motivated and described in greater detail (but less formally) in [2].

Model. We formalize the program model we presented in [2].

Definition 1 (Program Model). *A* program model *is a tuple*

$$\mathcal{M} \triangleq (A, V; \mathbf{app}, \mathbf{ret}; \rightarrow^T, \rightarrow^C)$$

where A is a set of applet names*; V is a set of vertices called* program points*;* $\mathbf{app} : V \rightharpoonup A$ *is a partial function mapping program points to applet names;* $\mathbf{ret} : V \rightarrow \text{bool}$ *is a program point predicate identifying the program's* return points*;* $\rightarrow^T \subseteq V \times V$ *is a* transfer relation *respecting applet boundaries,* i.e. $\mathbf{app}(v) = a$ *and* $v \rightarrow^T v'$ *implies* $\mathbf{app}(v') = a$*;* $\rightarrow^C \subseteq V \times V$ *is a* call relation *between program points, of which the elements* $\langle v, v' \rangle$ *are referred to as* calls*.*

We shall use the notation $\mathbf{loc}_a\, v$ for $\mathbf{app}(v) = a$. We next define the notions of applet state and program state.

Definition 2 (State). *Let* $\mathcal{M} = (A, V; \mathbf{app}, \mathbf{ret}; \rightarrow^T, \rightarrow^C)$ *be a program model.*

(i) An applet state *of* $\mathcal{M}$ *is a pair* $a.\pi$*, where* $a \in A$ *is an applet name, and* $\pi \in (V \times V)^*$ *is a sequence of program point pairs called the* applet call stack*. An applet state* $a.\pi$ *is* active *iff the second program point of the last pair in the call stack* π *is local to applet* a*; in this case this program point is referred to as the* active program point*.*

(ii) A program state s *of* $\mathcal{M}$ *is a collection* $a_1.\pi_1 \mid a_2.\pi_2 \mid \ldots \mid a_n.\pi_n$ *of applet states. A program state is* active *iff it contains an active applet state, and* wellformed *iff it mentions applet names at most once and contains at most one active state. Two program states* intersect *iff they mention the same applet name.*

Intuitively, an applet state represents the unfinished calls that have been made from and to an applet. As method calls can cross applet boundaries, both the source and the destination of a call are remembered, to ensure "proper" returning (*i.e.* to the appropriate callee) from a method call. If intra-procedural execution takes place within an applet, *i.e.* by following the transfer relation, the changing of the active program point is reflected by changing the last element in the call stack. This is described by the operational semantics below.

Operational Semantics. The behavior of programs is given in terms of labeled transition systems.

Definition 3 (Induced LTS). *A program model* $\mathcal{M}$ *induces a LTS* $\mathcal{T}_{\mathcal{M}} \stackrel{\Delta}{=} (\mathcal{S}_{\mathcal{M}}, \mathcal{L}_{\mathcal{M}}; \rightarrow_{\mathcal{M}})$ *in a straightforward fashion:* $\mathcal{S}_{\mathcal{M}}$ *is the set of wellformed program states of* $\mathcal{M}$*;* $\mathcal{L}_{\mathcal{M}}$ *is the set of* transition labels *consisting of* τ *and the triples of the shape* $v\ l\ v'$*,* $v\ l?\ v'$*, and* $v\ l!\ v'$*, where* $v, v' \in V$ *and* $l \in \{\mathsf{call}, \mathsf{ret}\}$*; and* $\rightarrow_{\mathcal{M}} \subseteq \mathcal{S}_{\mathcal{M}} \times \mathcal{L}_{\mathcal{M}} \times \mathcal{S}_{\mathcal{M}}$ *is the least transition relation on* $\mathcal{S}_{\mathcal{M}}$ *closed under the transition rules of* [2] *(see Appendix A).*

We write $s \xrightarrow{\alpha}_{\mathcal{M}} s'$ for $(s, \alpha, s') \in \rightarrow_{\mathcal{M}}$, and by convention omit τ from such transition edges, writing $s \longrightarrow_{\mathcal{M}} s'$ for $s \xrightarrow{\tau}_{\mathcal{M}} s'$.

Two kinds of transition rules are used: (1) applet transition rules, describing state changes in a single applet, and (2) transition rules for composite states, describing the behavior of composed applet sets. An example of a transition rule in the first category is the rule

$$[\mathsf{send\ call}]\ \frac{v_1 \rightarrow^C v_2 \qquad \mathbf{loc}_a\, v_1 \qquad \neg\mathbf{loc}_a\, v_2}{a.\pi \cdot \langle v, v_1\rangle \xrightarrow{v_1\ \mathsf{call!}\ v_2} a.\pi \cdot \langle v, v_1\rangle \cdot \langle v_1, v_2\rangle}$$

which describes under which conditions an applet can invoke a method in another applet. This rule specifies that if applet a is in a state where v_1 is the active program point, and if there is an outgoing call edge $v_1 \rightarrow^C v_2$ to an external program point v_2, then sending a call over applet boundaries is enabled. The local state of the applet will be extended with a pair $\langle v_1, v_2\rangle$, to reflect this call.

Notice that this implicitly makes a inactive, because the last program point in the applet state is now no longer local to a.

An example of a transition rule for composite states is the rule

$$[\mathsf{synchro}]\ \frac{\mathcal{A}_1 \xrightarrow{v_1\ l?\ v_2} \mathcal{A}_1' \qquad \mathcal{A}_2 \xrightarrow{v_1\ l!\ v_2} \mathcal{A}_2'}{\mathcal{A}_1 \mid \mathcal{A}_2 \xrightarrow{v_1\ l\ v_2} \mathcal{A}_1' \mid \mathcal{A}_2'}\ l \in \mathsf{call}, \mathsf{ret}$$

which describes how two applets synchronize if one sends out a message call or return, which is received by the other applet. It is said that the imperfectly labeled transitions $v_1\ l?\ v_2$ and $v_1\ l!\ v_2$ synchronize into one perfectly labeled transition $v_1\ l\ v_2$.

3 Property Specification Language

Properties of component interaction can be conveniently expressed in a modal logic with recursion, such as Kozen's modal μ-calculus [12], extended with model-specific atomic formulae, where the modalities are indexed by transition labels. In addition, such a logic is suitable for compositional reasoning (*cf.* [16,8,9]).

Let a range over applet name variables[4], v over program point variables, π over applet call stack variables, Π over applet call stack terms generated by $\Pi ::= \epsilon \mid \pi \mid \Pi \cdot \langle v, v' \rangle$, $\mathcal{X}$ and $\mathcal{Y}$ over program state variables, $\mathcal{A}$ over program state terms generated by $\mathcal{A} ::= a.\Pi \mid \mathcal{A} \,|\, \mathcal{A}$, and α over transition labels, all of these possibly indexed or primed.

Definition 4 (Syntax). Atomic formulae σ *and* formulae ϕ *are generated by the following grammar, where* x *ranges over program point variables and applet call stack variables,* t *over program point variables and applet call stack terms,* α *over transition labels, and* X *over propositional variables:*

$$\begin{aligned} \sigma &::= t = t \mid \mathsf{return}\ v \mid \mathsf{local}_{\mathcal{A}}\ v \\ \phi &::= \sigma \mid \mathsf{active} \mid \neg\phi \mid \phi \wedge \phi \mid \forall x.\phi \mid [\alpha]\,\phi \mid X \mid \nu X.\phi \end{aligned}$$

We write $\mathsf{local}_a\ v$ for $\mathsf{local}_{a.\pi}\ v$.

An occurrence of a subformula ψ in ϕ is *positive*, if ψ appears in the scope of an even number of negation symbols; otherwise the occurrence is negative. The formation of fixed point formulae is subject to the usual formal monotonicity condition that occurrences of X in ϕ are positive. A formula ϕ is *propositionally closed* if ϕ does not have free occurrences of propositional variables. Standard abbreviations apply, like for instance $\exists x.\phi \stackrel{\Delta}{=} \neg\forall x.\neg\phi$, $\langle\alpha\rangle\,\phi \stackrel{\Delta}{=} \neg\,[\alpha]\,\neg\phi$, and $\mu X.\phi \stackrel{\Delta}{=} \neg\nu X.\neg(\phi[\neg X/X])$.

The semantics of formulae is given relative to a program model $\mathcal{M}$, its induced LTS $\mathcal{T}_{\mathcal{M}}$, and an environment ρ mapping variables to members of their

[4] By abuse of notation we use the same letters for the variables of the logic and for arbitrary constants of the respective domain in the model.

respective domains, lifted to expressions Π and $\mathcal{A}$ in the natural way. The semantic interpretation $[\![\sigma]\!]^{\mathcal{M}}_{\rho}$ of an atomic formula σ maps σ to a boolean, while the semantics $\|\phi\|^{\mathcal{M}}_{\rho}$ of a formula ϕ is described as the set of program states satisfying formula ϕ.

Definition 5 (Semantics). *Let $\mathcal{M}$ be a program model, $\mathcal{T}_{\mathcal{M}}$ be its induced LTS, and let ρ be an environment. The semantics of atomic formulae is defined by:*

$$[\![t_1 = t_2]\!]^{\mathcal{M}}_{\rho} \triangleq t_1\rho = t_2\rho$$
$$[\![\mathsf{return}\ v]\!]^{\mathcal{M}}_{\rho} \triangleq \mathbf{ret}(v\rho)$$
$$[\![\mathsf{local}_{\mathcal{A}}\ v]\!]^{\mathcal{M}}_{\rho} \triangleq \mathbf{app}(v\rho) \text{ is defined and occurs in } \mathcal{A}\rho$$

The semantics of formulae is standard [12]*, except for:*

$$\|\sigma\|^{\mathcal{M}}_{\rho} \triangleq \text{if } [\![\sigma]\!]^{\mathcal{M}}_{\rho} \text{ then } \mathcal{S}_{\mathcal{M}} \text{ else } \emptyset$$
$$\|\mathsf{active}\|^{\mathcal{M}}_{\rho} \triangleq \{s \in \mathcal{S}_{\mathcal{M}} \mid s \text{ is active}\}$$

4 Specification Patterns

The property specification language presented above is rather low-level. To facilitate high-level formal reasoning, we introduce a collection of specification patterns, following the approach of the Bandera project [6].

In the context of the present work, the use of specification patterns has an additional purpose: as explained in the introduction, we have two different kind of verification tasks in our framework, namely model-checking the local properties of the individual applets, and proving property decompositions correct. The use of general temporal logic patterns allows us to use different verification techniques (based on different logics) for the different tasks. For example, we can model check the local applet properties by translating, as appropriate, the specifications into CTL (*e.g.* as input for NuSMV [5]) or LTL (*e.g.* as input for SPIN [10]), while we can use the modal μ-calculus to prove the correctness of the property decomposition, as this is more suitable for the task.

A typical specification pattern used to express invariant properties is:

$$\begin{aligned}\textsc{Always}\ \phi \triangleq\ & \nu X.\ \phi \wedge [\tau]\, X \\ & \wedge\ \forall v_1.\ \forall v_2.\ [v_1\ \mathsf{call}\ v_2]\, X \\ & \qquad \wedge\ [v_1\ \mathsf{call?}\ v_2]\, X \\ & \qquad \wedge\ [v_1\ \mathsf{call!}\ v_2]\, X \\ & \qquad \wedge\ [v_1\ \mathsf{ret}\ v_2]\, X \\ & \qquad \wedge\ [v_1\ \mathsf{ret?}\ v_2]\, X \\ & \qquad \wedge\ [v_1\ \mathsf{ret!}\ v_2]\, X\end{aligned}$$

When components communicate via procedure or method calls one frequently needs to specify that some property ϕ holds within a call, *i.e.* from the point of

invocation to the point of return. For these purposes we propose the following pattern:

$$\textsc{Within}\ v\ \phi \stackrel{\Delta}{=} \forall v_1.\ [v_1\ \mathsf{call}\ v]\ \textsc{Always}_{-\mathsf{ret}\ v_1}\phi \wedge [v_1\ \mathsf{call?}\ v]\ \textsc{Always}_{-\mathsf{ret!}\ v_1}\phi$$

where $\textsc{Always}_{-\lambda\, v}\ \phi$ is defined as $\textsc{Always}\ \phi$, but with the corresponding conjunct $[v_1 \lambda\ v_2]\, X$ replaced by $[v_1 \lambda\ v_2]\, ((v_2 = v) \vee X)$.

In the example of Section 6 we also use the abbreviations:

$$\textsc{CallsExtOnly}\ V \stackrel{\Delta}{=} \forall v_1.\ \forall v_2.\ [v_1\ \mathsf{call!}\ v_2]\, v_2 \in V$$

$$\textsc{CannotCall}\ a\ V \stackrel{\Delta}{=} \forall v_1.\ \forall v_2.\ [v_1\ \mathsf{call}\ v_2] \neg\ (\mathsf{local}_a\ v_1 \wedge v_2 \in V) \wedge [v_1\ \mathsf{call!}\ v_2] \neg\ (\mathsf{local}_a\ v_1 \wedge v_2 \in V)$$

where V denotes an explicit enumeration $v_1, \ldots, v_n$ of vertices and $v \in V$ is syntactic sugar for $v = v_1 \vee \cdots \vee v = v_n$.

5 Proof System

For proving correctness of property decompositions, we develop a Gentzen-style proof system based on the compositional approach advocated by Simpson [16]. This approach has been successfully used for the compositional verification of CCS programs [7], and even of complex telecommunications software written in the Erlang programming language [9].

The proof system uses Gentzen style sequents, *i.e.* proof judgments of the form $\phi_1, \ldots, \phi_n \vdash \psi_1, \ldots, \psi_n$. The intuitive interpretation of such a sequent is that the conjunction of the antecedents implies the disjunction of the consequents, *i.e.* $\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \psi_1 \vee \ldots \vee \psi_n$. Formally, we define the building blocks of our proof system as follows.

Definition 6 (Assertion, Sequent).

(i) An assertion γ *is either a* satisfaction assertion $\mathcal{A} : \phi$*, where* ϕ *is a propositionally closed formula, a* transition assertion $\mathcal{A}_1 \xrightarrow{\alpha} \mathcal{A}_2$*, an* atomic formula assertion σ*, a* transfer-edge assertion $v \rightarrow^T v'$*, a* call-edge assertion $v \rightarrow^C v'$*, or a* wellformedness assertion $\mathsf{wf}(\mathcal{A})$.

(ii) Assertion $\mathcal{A} : \phi$ *is* valid *for program model* $\mathcal{M}$ *and environment* ρ *if* $\mathcal{A}\rho \in \|\phi\|^{\mathcal{M}}_{\rho}$. $\mathcal{A}_1 \xrightarrow{\alpha} \mathcal{A}_2$ *is valid for* $\mathcal{M}$ *and* ρ *if* $\mathcal{A}_1\rho \xrightarrow{\alpha\rho}_{\mathcal{M}} \mathcal{A}_2\rho$. σ *is valid for* $\mathcal{M}$ *and* ρ *if* $[\![\sigma]\!]^{\mathcal{M}}_{\rho}$. $v \rightarrow^T v'$ *is valid for* $\mathcal{M}$ *and* ρ *if* $v\rho \rightarrow^T v'\rho$ *in* $\mathcal{M}$. $v \rightarrow^C v'$ *is valid for* $\mathcal{M}$ *and* ρ *if* $v\rho \rightarrow^C v'\rho$ *in* $\mathcal{M}$. $\mathsf{wf}(\mathcal{A})$ *is valid for* $\mathcal{M}$ *and* ρ *if* $\mathcal{A}\rho$ *is wellformed in* $\mathcal{M}$.

(iii) A sequent *is a proof judgment of the form* $\Gamma \vdash \Delta$*, where* Γ *and* Δ *are sets of assertions.*

(iv) Sequent $\Gamma \vdash \Delta$ *is* valid *if, for all program models* $\mathcal{M}$ *and environments* ρ*, whenever all assertions in* Γ *are valid for* $\mathcal{M}$ *and* ρ *then also some assertion in* Δ *is valid for* $\mathcal{M}$ *and* ρ.

Note that wellformedness of program states does not lift to program-state terms in a way which can be captured purely syntactically, and therefore has to be dealt with explicitly in the proof system.

We now present, in groups, the proof rules of our proof system. Since many of these are standard, we only show the most interesting ones here; the remaining rules can be found in Appendix B. The side condition "fresh x" appearing in some of the rules means "x does not appear free in the *conclusion* of the rule".

Structural and Logical Rules. As structural rules, we assume the standard identity, cut and weakening rules of Gentzen-style proof systems. We have rules for the various atomic formula constructs. Equality is handled through standard congruence rules, plus standard rules for freely generated datatypes (for dealing with equality on program stack terms). The rules for $\mathsf{local}_{\mathcal{A}}\, v$ proceed by decomposing the program state terms; in addition we have:

$$(\mathsf{LocInt})\ \frac{\cdot}{\Gamma,\ \mathsf{local}_{\mathcal{A}_1}\, v,\ \mathsf{local}_{\mathcal{A}_2}\, v\ \vdash\ \mathsf{intersect}(\mathcal{A}_1, \mathcal{A}_2),\ \Delta}$$

$$(\mathsf{LocTransf})\ \frac{\cdot}{\Gamma,\ \mathsf{local}_{\mathcal{A}}\, v,\ v \rightarrow^{T} v'\ \vdash\ \mathsf{local}_{\mathcal{A}}\, v',\ \Delta}$$

where $\mathsf{intersect}(\mathcal{A}_1, \mathcal{A}_2)$ is an auxiliary assertion used to capture program state intersection.

Most of the logical rules (dealing with satisfaction assertions $\mathcal{A} : \phi$) are standard. Not so are the proof rules for active:

$$(\mathsf{ActComL})\ \frac{\Gamma,\ \mathcal{A}_1 : \mathsf{active}\ \vdash\ \Delta \qquad \Gamma,\ \mathcal{A}_2 : \mathsf{active}\ \vdash\ \Delta}{\Gamma,\ \mathcal{A}_1 \,|\, \mathcal{A}_2 : \mathsf{active}\ \vdash\ \Delta}$$

$$(\mathsf{ActComR})\ \frac{\Gamma\ \vdash\ \mathcal{A}_1 : \mathsf{active},\ \mathcal{A}_2 : \mathsf{active},\ \Delta}{\Gamma\ \vdash\ \mathcal{A}_1 \,|\, \mathcal{A}_2 : \mathsf{active},\ \Delta}$$

$$(\mathsf{ActL})\ \frac{\Gamma,\ \Pi = \pi \cdot \langle v_1, v_2 \rangle\ ,\ \mathsf{local}_a\, v_2\ \vdash\ \Delta}{\Gamma,\ a.\Pi : \mathsf{active}\ \vdash\ \Delta}\ \text{fresh } \pi, v_1, v_2$$

$$(\mathsf{ActR})\ \frac{\Gamma\ \vdash\ \Pi = \Pi' \cdot \langle v_1, v_2 \rangle\ ,\ \Delta \qquad \Gamma\ \vdash\ \mathsf{local}_a\, v_2,\ \Delta}{\Gamma\ \vdash\ a.\Pi : \mathsf{active},\ \Delta}$$

Fixed–point formulae are handled as in [8] through fixed-point approximation by using explicit ordinal variables κ to represent approximation ordinals:

$$(\mathsf{NuL})\ \frac{\Gamma,\ \mathcal{A} : \phi[\nu X.\phi/X]\ \vdash\ \Delta}{\Gamma,\ \mathcal{A} : \nu X.\phi\ \vdash\ \Delta} \qquad (\mathsf{NuR})\ \frac{\Gamma\ \vdash\ \mathcal{A} : (\nu X.\phi)^{\kappa},\ \Delta}{\Gamma\ \vdash\ \mathcal{A} : \nu X.\phi,\ \Delta}\ \text{fresh } \kappa$$

$$(\mathsf{ApproxR})\ \frac{\Gamma,\ \kappa' < \kappa\ \vdash\ \mathcal{A} : \phi[(\nu X.\phi)^{\kappa'}/X],\ \Delta}{\Gamma\ \vdash\ \mathcal{A} : (\nu X.\phi)^{\kappa},\ \Delta}\ \text{fresh } \kappa'$$

These ordinal variables are examined by a global *discharge rule*, which checks whether the proof tree constitutes a valid well-founded induction scheme. Informally, the discharge rule applies if (1) every non-axiom leaf of the proof tree is an instance (up to a substitution) of some ancestor sequent in the proof tree, (2) for each such sequent, this substitution maps some ordinal variable approximating a fixed-point formula to an ordinal variable which is assumed to be smaller, and (3) these separate induction schemes are *consistent* with each other. For the technical details the interested reader is referred to [8,7,9].

Transition Rules. These rules deal with transition assertions $\mathcal{A}_1 \xrightarrow{\alpha} \mathcal{A}_2$.

We first consider the case where $\mathcal{A}$ is a composite state. The r.h.s. rules follow directly from the transition rules for program states (see Appendix A), after making the wellformedness conditions explicit.

$$(\mathsf{ComTauR})\ \frac{\Gamma \vdash \mathcal{A}_1 \rightarrow \mathcal{A}_1',\ \Delta \qquad \Gamma \vdash \mathsf{wf}(\mathcal{A}_1 \,|\, \mathcal{A}_2),\ \Delta \qquad \Gamma \vdash \mathsf{wf}(\mathcal{A}_1' \,|\, \mathcal{A}_2),\ \Delta}{\Gamma \vdash \mathcal{A}_1 \,|\, \mathcal{A}_2 \rightarrow \mathcal{A}_1' \,|\, \mathcal{A}_2,\ \Delta}$$

$$(\mathsf{ComSyncR})\ \frac{\begin{array}{l}\Gamma \vdash \mathcal{A}_1 \xrightarrow{v_1\ l!\ v_2} \mathcal{A}_1',\ \Delta \qquad \Gamma \vdash \mathsf{wf}(\mathcal{A}_1 \,|\, \mathcal{A}_2),\ \Delta \\ \Gamma \vdash \mathcal{A}_2 \xrightarrow{v_1\ l?\ v_2} \mathcal{A}_2',\ \Delta \qquad \Gamma \vdash \mathsf{wf}(\mathcal{A}_1' \,|\, \mathcal{A}_2'),\ \Delta\end{array}}{\Gamma \vdash \mathcal{A}_1 \,|\, \mathcal{A}_2 \xrightarrow{v_1\ l\ v_2} \mathcal{A}_1' \,|\, \mathcal{A}_2',\ \Delta}$$

$$(\mathsf{ComPropR})\ \frac{\begin{array}{l}\Gamma \vdash \mathcal{A}_1 \xrightarrow{v_1\ \lambda\ v_2} \mathcal{A}_1',\ \Delta \qquad \Gamma,\ \mathsf{local}_{\mathcal{A}_2}\ v_1 \vdash \Delta \qquad \Gamma \vdash \mathsf{wf}(\mathcal{A}_1 \,|\, \mathcal{A}_2),\ \Delta \\ \qquad\qquad\qquad\qquad\qquad\quad \Gamma,\ \mathsf{local}_{\mathcal{A}_2}\ v_2 \vdash \Delta \qquad \Gamma \vdash \mathsf{wf}(\mathcal{A}_1' \,|\, \mathcal{A}_2),\ \Delta\end{array}}{\Gamma \vdash \mathcal{A}_1 \,|\, \mathcal{A}_2 \xrightarrow{v_1\ \lambda\ v_2} \mathcal{A}_1' \,|\, \mathcal{A}_2,\ \Delta}$$

where l is call or ret and λ is l, $l?$ or $l!$. All three rules have symmetric counterparts which we omit.

Notice that in each rule two proof obligations arise on the wellformedness of the state. This may seem a heavy proof burden, but almost all these proof obligations can be discharged immediately. It is future work to derive optimized proof rules which result in less proof obligations.

The l.h.s. rules apply when we assume that a certain transition is possible. By the closure condition of the transition semantics, the possible transitions are exactly those inferable by the transition rules, thus these proof rules have to capture the conditions under which we can assume that this transition is possible.

For example, a transition $\mathcal{A}_1 \,|\, \mathcal{A}_2 \xrightarrow{v_1\ \mathsf{call}\ v_2} \mathcal{X}$ only is possible if the transition rules [synchro] or [propagation] apply. The transition rule [synchro] applies if: (1) $\mathcal{A}_1$ can do a transition to $\mathcal{A}_1'$ labeled $v_1\ \mathsf{call!}\ v_2$, (2) $\mathcal{A}_2$ can do a transition to $\mathcal{A}_2'$, labeled $v_1\ \mathsf{call?}\ v_2$, and (3) $\mathcal{X}$ is of the form $\mathcal{A}_1' \,|\, \mathcal{A}_2'$, or the symmetric counterpart of this applies. Similarly, it can be decided under which conditions the transition rule [propagation] applies. If we assume that such a transition $\mathcal{A}_1 \,|\, \mathcal{A}_2 \xrightarrow{v_1\ \mathsf{call}\ v_2} \mathcal{X}$ is possible, one of these rules must have been applied, and thus for one of these rules all conditions must have been satisfied. This is exactly

captured by the proof rule (ComPerfL), with explicit wellformedness conditions added.

$$\textsf{(ComTauL)}\ \frac{\Gamma[(\mathcal{Y}\,|\,\mathcal{A}_2)/\mathcal{X}],\ \mathcal{A}_1 \to \mathcal{Y},\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{A}_2),\ \mathsf{wf}(\mathcal{Y}\,|\,\mathcal{A}_2)\ \vdash\ \Delta[(\mathcal{Y}\,|\,\mathcal{A}_2)/\mathcal{X}] \quad \Gamma[(\mathcal{A}_1\,|\,\mathcal{Y})/\mathcal{X}],\ \mathcal{A}_2 \to \mathcal{Y},\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{A}_2),\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{Y})\ \vdash\ \Delta[(\mathcal{A}_1\,|\,\mathcal{Y})/\mathcal{X}]}{\Gamma,\ \mathcal{A}_1\,|\,\mathcal{A}_2 \to \mathcal{X}\ \vdash\ \Delta}\ \text{fresh } \mathcal{Y}$$

$$\textsf{(ComImpL)}\ \frac{\Gamma[(\mathcal{Y}\,|\,\mathcal{A}_2)/\mathcal{X}],\ \mathcal{A}_1 \xrightarrow{v_1\, l?!\, v_2} \mathcal{Y},\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{A}_2),\ \mathsf{wf}(\mathcal{Y}\,|\,\mathcal{A}_2)\ \vdash\ \mathsf{local}_{\mathcal{A}_2}\, v_1,\ \mathsf{local}_{\mathcal{A}_2}\, v_2,\ \Delta[(\mathcal{Y}\,|\,\mathcal{A}_2)/\mathcal{X}] \quad \Gamma[(\mathcal{A}_1\,|\,\mathcal{Y})/\mathcal{X}],\ \mathcal{A}_2 \xrightarrow{v_1\, l?!\, v_2} \mathcal{Y},\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{A}_2),\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{Y})\ \vdash\ \mathsf{local}_{\mathcal{A}_1}\, v_1,\ \mathsf{local}_{\mathcal{A}_1}\, v_2,\ \Delta[(\mathcal{A}_1\,|\,\mathcal{Y})/\mathcal{X}]}{\Gamma,\ \mathcal{A}_1\,|\,\mathcal{A}_2 \xrightarrow{v_1\, l?!\, v_2} \mathcal{X}\ \vdash\ \Delta}$$

fresh $\mathcal{Y}$

$$\textsf{(ComPerfL)}\ \frac{\begin{array}{l}\Gamma[(\mathcal{Y}\,|\,\mathcal{A}_2)/\mathcal{X}],\ \mathcal{A}_1 \xrightarrow{v_1\, l\, v_2} \mathcal{Y},\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{A}_2),\ \mathsf{wf}(\mathcal{Y}\,|\,\mathcal{A}_2)\ \vdash\ \mathsf{local}_{\mathcal{A}_2}\, v_1,\ \mathsf{local}_{\mathcal{A}_2}\, v_2,\ \Delta[(\mathcal{Y}\,|\,\mathcal{A}_2)/\mathcal{X}] \\ \Gamma[(\mathcal{A}_1\,|\,\mathcal{Y})/\mathcal{X}],\ \mathcal{A}_2 \xrightarrow{v_1\, l\, v_2} \mathcal{Y},\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{A}_2),\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{Y})\ \vdash\ \mathsf{local}_{\mathcal{A}_1}\, v_1,\ \mathsf{local}_{\mathcal{A}_1}\, v_2,\ \Delta[(\mathcal{A}_1\,|\,\mathcal{Y})/\mathcal{X}] \\ \Gamma[(\mathcal{X}_1\,|\,\mathcal{X}_2)/\mathcal{X}],\ \mathcal{A}_1 \xrightarrow{v_1\, l!\, v_2} \mathcal{X}_1,\ \mathcal{A}_2 \xrightarrow{v_1\, l?\, v_2} \mathcal{X}_2,\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{A}_2),\ \mathsf{wf}(\mathcal{X}_1\,|\,\mathcal{X}_2)\ \vdash\ \Delta[(\mathcal{X}_1\,|\,\mathcal{X}_2)/\mathcal{X}] \\ \Gamma[(\mathcal{X}_1\,|\,\mathcal{X}_2)/\mathcal{X}],\ \mathcal{A}_1 \xrightarrow{v_1\, l?\, v_2} \mathcal{X}_1,\ \mathcal{A}_2 \xrightarrow{v_1\, l!\, v_2} \mathcal{X}_2,\ \mathsf{wf}(\mathcal{A}_1\,|\,\mathcal{A}_2),\ \mathsf{wf}(\mathcal{X}_1\,|\,\mathcal{X}_2)\ \vdash\ \Delta[(\mathcal{X}_1\,|\,\mathcal{X}_2)/\mathcal{X}]\end{array}}{\Gamma,\ \mathcal{A}_1\,|\,\mathcal{A}_2 \xrightarrow{v_1\, l\, v_2} \mathcal{X}\ \vdash\ \Delta}$$

fresh $\mathcal{X}_1, \mathcal{X}_2, \mathcal{Y}$

In rule (ComImpL), $l?!$ stands for either $l?$ or $l!$.

We now turn to the case when $\mathcal{A}$ is a singleton set, *i.e.* an applet state. Again, the r.h.s. rules follow immediately from the transition rules. However, in many transition rules there is an implicit condition on the form of the applet state, *e.g.* to be able to apply the rule [send call], the call stack has to be of the form $\Pi \cdot \langle v_1, v_2 \rangle$. These conditions are made explicit in the proof rules.

$$\textsf{(LocCallR)}\ \frac{\Gamma\ \vdash\ v_1 \to^C v_2,\ \Delta \quad \Gamma\ \vdash\ \mathsf{local}_a\, v_1,\ \Delta \quad \Gamma\ \vdash\ \Pi = \Pi' \cdot \langle v, v_1 \rangle,\ \Delta \quad \Gamma\ \vdash\ \mathsf{local}_a\, v_2,\ \Delta}{\Gamma\ \vdash\ a.\Pi \xrightarrow{v_1\ \mathsf{call}\ v_2} a.(\Pi \cdot \langle v_1, v_2 \rangle),\ \Delta}$$

$$\textsf{(LocRetR)}\ \frac{\begin{array}{ll}\Gamma\ \vdash\ v_1 \to^T v_2,\ \Delta & \Gamma\ \vdash\ \mathsf{local}_a\, v_1,\ \Delta \\ \Gamma\ \vdash\ \Pi = \Pi' \cdot \langle v, v_1 \rangle \cdot \langle v_1, v_3 \rangle,\ \Delta & \Gamma\ \vdash\ \mathsf{local}_a\, v_3,\ \Delta \qquad \Gamma\ \vdash\ \mathsf{return}\ v_3,\ \Delta\end{array}}{\Gamma\ \vdash\ a.\Pi \xrightarrow{v_3\ \mathsf{ret}\ v_1} a.(\Pi' \cdot \langle v, v_2 \rangle),\ \Delta}$$

$$\textsf{(LocTransfR)}\ \frac{\Gamma\ \vdash\ v_1 \to^T v_2,\ \Delta \quad \Gamma\ \vdash\ \mathsf{local}_a\, v_1,\ \Delta \quad \Gamma,\ v_1 \to^C v_3\ \vdash\ \Delta \quad \Gamma\ \vdash\ \Pi = \Pi' \cdot \langle v, v_1 \rangle,\ \Delta}{\Gamma\ \vdash\ a.\Pi \to a.(\Pi' \cdot \langle v, v_2 \rangle),\ \Delta}\ \text{fresh } v_3$$

$$\textsf{(SendCallR)}\ \frac{\Gamma\ \vdash\ v_1 \to^C v_2,\ \Delta \quad \Gamma\ \vdash\ \mathsf{local}_a\, v_1,\ \Delta \quad \Gamma\ \vdash\ \Pi = \Pi' \cdot \langle v, v_1 \rangle,\ \Delta \quad \Gamma,\ \mathsf{local}_a\, v_2\ \vdash\ \Delta}{\Gamma\ \vdash\ a.\Pi \xrightarrow{v_1\ \mathsf{call!}\ v_2} a.(\Pi \cdot \langle v_1, v_2 \rangle),\ \Delta}$$

$$\textsf{(RecCallR)}\ \frac{\Gamma\ \vdash\ v_1 \to^C v_2,\ \Delta \quad \Gamma,\ \mathsf{local}_a\, v_1\ \vdash\ \Delta \quad \Gamma,\ a.\Pi : \mathsf{active}\ \vdash\ \Delta \quad \Gamma\ \vdash\ \mathsf{local}_a\, v_2,\ \Delta}{\Gamma\ \vdash\ a.\Pi \xrightarrow{v_1\ \mathsf{call?}\ v_2} a.(\Pi \cdot \langle v_1, v_2 \rangle),\ \Delta}$$

$$\textsf{(SendRetR)}\ \frac{\Gamma,\ \mathsf{local}_a\, v_1\ \vdash\ \Delta \quad \Gamma\ \vdash\ \mathsf{local}_a\, v_2,\ \Delta \quad \Gamma\ \vdash\ \Pi = \Pi' \cdot \langle v_1, v_2 \rangle,\ \Delta \quad \Gamma\ \vdash\ \mathsf{return}\ v_2,\ \Delta}{\Gamma\ \vdash\ a.\Pi \xrightarrow{v_2\ \mathsf{ret!}\ v_1} a.\Pi',\ \Delta}$$

$$\textsf{(RecRetR)}\ \frac{\begin{array}{ll}\Gamma\ \vdash\ v_1 \to^T v_2,\ \Delta & \Gamma,\ \mathsf{local}_a\, v_3\ \vdash\ \Delta \\ \Gamma\ \vdash\ \Pi = \Pi' \cdot \langle v, v_1 \rangle \cdot \langle v_1, v_3 \rangle,\ \Delta & \Gamma,\ \mathsf{local}_a\, v_4\ \vdash\ \Delta \qquad \Gamma\ \vdash\ \mathsf{local}_a\, v_1,\ \Delta\end{array}}{\Gamma\ \vdash\ a.\Pi \xrightarrow{v_4\ \mathsf{ret?}\ v_1} a.(\Pi' \cdot \langle v, v_2 \rangle),\ \Delta}$$

The l.h.s. rules are constructed from the transition rules in the same way as the l.h.s. rules for composite states above.

$$\text{(LocCallL)}\ \frac{\Gamma[a.(\Pi \cdot \langle v_1, v_2\rangle)/\mathcal{X}],\ \Pi = \pi \cdot \langle v, v_1\rangle,\ v_1 \rightarrow^C v_2,\ \mathsf{local}_a\ v_1,\ \mathsf{local}_a\ v_2\ \vdash \quad \Delta[a.(\Pi \cdot \langle v_1, v_2\rangle)/\mathcal{X}]}{\Gamma,\ a.\Pi \xrightarrow{v_1\ \mathsf{call}\ v_2} \mathcal{X}\ \vdash\ \Delta}\ \text{fresh } v, \pi$$

$$\text{(LocRetL)}\ \frac{\Gamma[a.(\pi \cdot \langle v, v_2\rangle)/\mathcal{X}],\ \Pi = \pi \cdot \langle v, v_1\rangle \cdot \langle v_1, v_3\rangle,\ v_1 \rightarrow^T v_2,\ \mathsf{local}_a\ v_1,\ \mathsf{local}_a\ v_3,\ \mathsf{return}\ v_3\ \vdash \quad \Delta[a.(\pi \cdot \langle v, v_2\rangle)/\mathcal{X}]}{\Gamma,\ a.\Pi \xrightarrow{v_3\ \mathsf{ret}\ v_1} \mathcal{X}\ \vdash\ \Delta}$$

$$\text{fresh } v, v_2, \pi$$

$$\text{(LocTransfL)}\ \frac{\Gamma[a.(\pi \cdot \langle v, v_2\rangle)/\mathcal{X}],\ \Pi = \pi \cdot \langle v, v_1\rangle,\ v_1 \rightarrow^T v_2,\ \mathsf{local}_a\ v_1\ \vdash \quad v_1 \rightarrow^C v_3,\ \Delta[a.(\pi \cdot \langle v, v_2\rangle)/\mathcal{X}]}{\Gamma,\ a.\Pi \rightarrow \mathcal{X}\ \vdash\ \Delta}\ \text{fresh } v, v_1, v_2, \pi$$

$$\text{(SendCallL)}\ \frac{\Gamma[a.(\Pi \cdot \langle v_1, v_2\rangle)/\mathcal{X}],\ \Pi = \pi \cdot \langle v, v_1\rangle,\ v_1 \rightarrow^C v_2,\ \mathsf{local}_a\ v_1\ \vdash \quad \mathsf{local}_a\ v_2,\ \Delta[a.(\Pi \cdot \langle v_1, v_2\rangle)/\mathcal{X}]}{\Gamma,\ a.\Pi \xrightarrow{v_1\ \mathsf{call!}\ v_2} \mathcal{X}\ \vdash\ \Delta}$$

$$\text{fresh } v, \pi$$

$$\text{(RecCallL)}\ \frac{\Gamma[a.(\Pi \cdot \langle v_1, v_2\rangle)/\mathcal{X}],\ v_1 \rightarrow^C v_2,\ \mathsf{local}_a\ v_2\ \vdash \quad \mathsf{local}_a\ v_1,\ a.\Pi : \mathsf{active},\ \Delta[a.(\Pi \cdot \langle v_1, v_2\rangle)/\mathcal{X}]}{\Gamma,\ a.\Pi \xrightarrow{v_1\ \mathsf{call?}\ v_2} \mathcal{X}\ \vdash\ \Delta}$$

$$\text{(SendRetL)}\ \frac{\Gamma[a.\pi/\mathcal{X}],\ \Pi = \pi \cdot \langle v_1, v_2\rangle,\ \mathsf{local}_a\ v_2,\ \mathsf{return}\ v_2\ \vdash \quad \mathsf{local}_a\ v_1,\ \Delta[a.\pi/\mathcal{X}]}{\Gamma,\ a.\Pi \xrightarrow{v_2\ \mathsf{ret!}\ v_1} \mathcal{X}\ \vdash\ \Delta}\ \text{fresh } \pi$$

$$\text{(RecRetL)}\ \frac{\Gamma[a.(\pi \cdot \langle v, v_2\rangle)/\mathcal{X}],\ \Pi = \pi \cdot \langle v, v_1\rangle \cdot \langle v_1, v_3\rangle,\ v_1 \rightarrow^T v_2,\ \mathsf{local}_a\ v_1\ \vdash \quad \mathsf{local}_a\ v_3, \mathsf{local}_a\ v_4,\ \Delta[a.(\pi \cdot \langle v, v_2\rangle)/\mathcal{X}]}{\Gamma,\ a.\Pi \xrightarrow{v_4\ \mathsf{ret?}\ v_1} \mathcal{X}\ \vdash\ \Delta}$$

$$\text{fresh } v, v_2, v_3, \pi$$

Wellformedness Rules. These rules reflect Definition 2, which states that a composed state $\mathcal{A}_1 \,|\, \mathcal{A}_2$ is wellformed *iff* its components are wellformed, at most one of the components is active, and the applet names in the components do not intersect.

$$\text{(WfAppletR)}\ \frac{}{\Gamma\ \vdash\ \mathsf{wf}(a.\Pi),\ \Delta}$$

$$\text{(WfComR)}\ \frac{\begin{array}{cc}\Gamma\ \vdash\ \mathsf{wf}(\mathcal{A}_1),\ \Delta & \Gamma,\ \mathsf{intersect}(\mathcal{A}_1, \mathcal{A}_2)\ \vdash\ \Delta \\ \Gamma\ \vdash\ \mathsf{wf}(\mathcal{A}_2),\ \Delta & \Gamma,\ \mathcal{A}_1 : \mathsf{active},\ \mathcal{A}_2 : \mathsf{active}\ \vdash\ \Delta\end{array}}{\Gamma\ \vdash\ \mathsf{wf}(\mathcal{A}_1 \,|\, \mathcal{A}_2),\ \Delta}$$

$$\text{(WfComL)}\ \frac{\begin{array}{l}\Gamma,\ \mathsf{wf}(\mathcal{A}_1),\ \mathsf{wf}(\mathcal{A}_2),\ \mathcal{A}_1 : \mathsf{active}\ \vdash\ \mathsf{intersect}(\mathcal{A}_1, \mathcal{A}_2),\ \mathcal{A}_2 : \mathsf{active},\ \Delta \\ \Gamma,\ \mathsf{wf}(\mathcal{A}_1),\ \mathsf{wf}(\mathcal{A}_2),\ \mathcal{A}_2 : \mathsf{active}\ \vdash\ \mathsf{intersect}(\mathcal{A}_1, \mathcal{A}_2),\ \mathcal{A}_1 : \mathsf{active},\ \Delta \\ \Gamma,\ \mathsf{wf}(\mathcal{A}_1),\ \mathsf{wf}(\mathcal{A}_2)\ \vdash\ \mathsf{intersect}(\mathcal{A}_1, \mathcal{A}_2),\ \mathcal{A}_1 : \mathsf{active},\ \mathcal{A}_2 : \mathsf{active},\ \Delta\end{array}}{\Gamma,\ \mathsf{wf}(\mathcal{A}_1 \,|\, \mathcal{A}_2)\ \vdash\ \Delta}$$

Soundness. The program model has been formalized and the proof rules have been proven sound *w.r.t.* the underlying model in PVS [15].

6 Example: Electronic Purse

To illustrate the working of the proof system, we take the electronic purse smartcard example of [3], which we discussed in greater detail (by providing the program model) in [2], and we outline the correctness proof of the decomposition of its specification. In this example an electronic purse is presented, which contains three applets: a `Purse` applet, and two loyalty applets: `AirFrance` and `RentACar`, with the standard functionalities. Besides, the `Purse` keeps a log table of bounded size of all transactions. Loyalties can subscribe to a (paying) `logFull` service, which signals that the log table is full and entries will be overridden. In the example, `AirFrance` is subscribed to this service. If it gets a `logFull` message, it will update its local balance, by asking the entries of the log table of the `Purse`, and by asking the balances of loyalty partners (`RentACar` in this example). In this way, `RentACar` can implicitly deduce that the log table is full, because it receives a `getBalance` message from `AirFrance`. A malicious implementation of `RentACar` can therefore request the information stored in the log table, before returning the value of its local balance. This is unwanted, because `RentACar` has not paid for the `logFull` service.

Thus, an invocation of `logFull` in the `AirFrance` applet by the `Purse` should not trigger a call from `RentACar` to `getTrs` (to ask the transactions) in the `Purse`. Using the macro definitions from Section 4 we can formally specify this as:

$$\textsc{Spec} \triangleq \textsc{Within}\ \texttt{AirFrance.logFull}\ \textsc{Spec}'$$
$$\textsc{Spec}' \triangleq \textsc{CannotCall}\ \texttt{RentACar}\ \texttt{Purse.getTrs}$$

The individual applets are specified as follows:

$$\textsc{Spec}_{\mathsf{P}} \triangleq \mathsf{local}_{\texttt{Purse}}\,\texttt{Purse.getTrs} \wedge \textsc{Spec}'_{\mathsf{P}}$$
$$\textsc{Spec}'_{\mathsf{P}} \triangleq \textsc{Always}\ (\textsc{Within}\ \texttt{Purse.getTrs}\ (\textsc{CallsExtOnly}\ \emptyset))$$

$$\textsc{Spec}_{\mathsf{AF}} \triangleq \mathsf{local}_{\texttt{AirFrance}}\,\texttt{AirFrance.logFull} \wedge \textsc{Spec}'_{\mathsf{AF}}$$
$$\textsc{Spec}'_{\mathsf{AF}} \triangleq \textsc{Always}\ (\textsc{Within}\ \texttt{AirFrance.logFull}\ \textsc{Spec}''_{\mathsf{AF}})$$
$$\textsc{Spec}''_{\mathsf{AF}} \triangleq \textsc{CallsExtOnly}\ \texttt{Purse.getTrs}, \texttt{RentACar.getBalance}$$

$$\textsc{Spec}_{\mathsf{RaC}} \triangleq \mathsf{local}_{\texttt{RentACar}}\,\texttt{RentACar.getBalance} \wedge \textsc{Spec}'_{\mathsf{RaC}}$$
$$\textsc{Spec}'_{\mathsf{RaC}} \triangleq \textsc{Always}\ (\textsc{Within}\ \texttt{RentACar.getBalance}(\textsc{CallsExtOnly}\ \emptyset))$$

To show that this property decomposition is correct, we have to parameterize these specifications by replacing the concrete applet names `Purse`, `AirFrance` and `RentACar` by the applet variables a_P, a_{AF} and a_{RaC}, and the concrete method names `Purse.getTrs`, `AirFrance.logFull` and `RentACar.getBalance` by the program point variables v_{GT}, v_{LF} and v_{GB}, respectively. We employ the proof system presented above to prove validity of the following sequent:

$$a_P.\pi_P : \textsc{Spec}_{\mathsf{P}},\ a_{AF}.\pi_{AF} : \textsc{Spec}_{\mathsf{AF}},\ a_{RaC}.\pi_{RaC} : \textsc{Spec}_{\mathsf{RaC}}$$
$$\vdash\ a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \,|\, a_{RaC}.\pi_{RaC} : \textsc{Spec}$$

There is a systematic method of proving validity of such sequents based on stepwise symbolic execution and loop detection. *Symbolic execution* refers to the process of computing the symbolic next-states of a program-state term (here $a_P.\pi_P \mid a_{AF}.\pi_{AF} \mid a_{RaC}.\pi_{RaC}$) guided by the modalities of the formula (here Spec). In this process some parameter terms of the program-state term might change. This requires the assumptions on these parameter terms to be updated. Some of the resulting symbolic next-states might be impossible, for example due to the accumulation of contradicting assumptions about the locality of program points, or because they violate the wellformedness restrictions on program states. *Loop detection* refers to detecting when a sequent is an instance of some ancestor sequent in the proof tree. This is necessary for checking the discharge condition.

We exemplify the method on the sequent above. First, we unfold the pattern and apply logical rules based on the outermost logical connectives of Spec until reaching a box-formula. In this way two subgoals are obtained; we focus on the first (the correctness proof of the second subgoal will follow the same structure):

$$\mathsf{local}_{a_P}\, v_{GT},\ \mathsf{local}_{a_{AF}}\, v_{LF},\ \mathsf{local}_{a_{RaC}}\, v_{GB},$$
$$a_P.\pi_P : \textsc{Spec}'_{\mathsf{P}},\ a_{AF}.\pi_{AF} : \textsc{Spec}'_{\mathsf{AF}},\ a_{RaC}.\pi_{RaC} : \textsc{Spec}'_{\mathsf{RaC}}$$
$$\vdash\ a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \,|\, a_{RaC}.\pi_{RaC} : [v_1\ \mathsf{call}\ v_{LF}]\ \textsc{Always}_{-\mathsf{ret}\ v_1}\textsc{Spec}'$$

Second, we apply rule BoxR followed by left transition rules for composite states until possible; this yields nine sequents corresponding to the nine different ways in which a perfect call action can come about in a system composed of three applets. Of these, four subgoals consider the cases where a_{AF} is not involved in the communication, and these can immediately be discarded (by applying the Id rule) due to contradicting assumptions about locality of v_{LF}. Two other sequents contain an assumption $a_{AF}.\pi_{AF} \xrightarrow{v_1\ \mathsf{call!}\ v_{LF}} \mathcal{X}$, *i.e.* a_{AF} sends a call to an external v_{LF}, and these can be discarded immediately by applying SendCallL and Id. The remaining three subgoals consider the three possible ways of producing a perfect call to vertex v_{LF} which is local to a_{AF}: by making a local call from within a_{AF}, or by calling from a_P or a_{RaC}.

Here, we focus on the case that v_{LF} is invoked by a local call from within a_{AF}. Verification of the other two cases continues along the same lines.

$$\mathsf{local}_{a_P}\, v_{GT},\ \mathsf{local}_{a_{AF}}\, v_{LF},\ \mathsf{local}_{a_{RaC}}\, v_{GB},$$
$$a_P.\pi_P : \textsc{Spec}'_{\mathsf{P}},\ a_{AF}.\pi_{AF} : \textsc{Spec}'_{\mathsf{AF}},\ a_{RaC}.\pi_{RaC} : \textsc{Spec}'_{\mathsf{RaC}},$$
$$\mathsf{wf}(a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \,|\, a_{RaC}.\pi_{RaC}),\ \mathsf{wf}(a_P.\pi_P \,|\, \mathcal{X} \,|\, a_{RaC}.\pi_{RaC}),$$
$$a_{AF}.\pi_{AF} \xrightarrow{v_1\ \mathsf{call}\ v_{LF}} \mathcal{X}$$
$$\vdash\ a_P.\pi_P \,|\, \mathcal{X} \,|\, a_{RaC}.\pi_{RaC} : \textsc{Always}_{-\mathsf{ret}\ v_1}\textsc{Spec}'$$

Next, we derive from the assumption(s) about $a_{AF}.\pi_{AF}$ assumptions about $\mathcal{X}$. We do this by applying l.h.s. logical rules (including NuL), until we obtain a box-

formula with label v_1 $\mathsf{call}\, v_{LF}$. To this formula we apply BoxL (taking $\mathcal{X}$ for $\mathcal{A}'$), which results in two subgoals. The first subgoals requires to show the transition assertion $a_{AF}.\pi_{AF} \xrightarrow{v_1\ \mathsf{call}\ v_{LF}} \mathcal{X}$, and can immediately be discarded (by Id). The other sequent looks as follows (after weakening):

$$\begin{array}{l}
\mathsf{local}_{a_P}\, v_{GT},\ \mathsf{local}_{a_{AF}}\, v_{LF},\ \mathsf{local}_{a_{RaC}}\, v_{GB},\\
a_P.\pi_P : \textsc{Spec}'_{\mathsf{P}},\ \mathcal{X} : \textsc{Always}_{-\mathsf{ret}\ v_1}\textsc{Spec}''_{\mathsf{AF}},\ a_{RaC}.\pi_{RaC} : \textsc{Spec}'_{\mathsf{RaC}},\\
a_{AF}.\pi_{AF} \xrightarrow{v_1\ \mathsf{call}\ v_{LF}} \mathcal{X},\\
\mathsf{wf}(a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \,|\, a_{RaC}.\pi_{RaC}),\ \mathsf{wf}(a_P.\pi_P \,|\, \mathcal{X} \,|\, a_{RaC}.\pi_{RaC})\\
\vdash\ a_P.\pi_P \,|\, \mathcal{X} \,|\, a_{RaC}.\pi_{RaC} : \textsc{Always}_{-\mathsf{ret}\ v_1}\textsc{Spec}'
\end{array}$$

And fourth, the transition assertion on the left is eliminated by applying the appropriate l.h.s. local transition rule, here LocCallL:

$$\begin{array}{l}
\mathsf{local}_{a_P}\, v_{GT},\ \mathsf{local}_{a_{AF}}\, v_{LF},\ \mathsf{local}_{a_{AF}}\, v_1,\ \mathsf{local}_{a_{RaC}}\, v_{GB},\\
a_P.\pi_P : \textsc{Spec}'_{\mathsf{P}},\ a_{AF}.\pi_{AF} \cdot \langle v_1, v_{LF}\rangle : \textsc{Always}_{-\mathsf{ret}\ v_1}\textsc{Spec}''_{\mathsf{AF}},\\
a_{RaC}.\pi_{RaC} : \textsc{Spec}'_{\mathsf{RaC}},\ \pi_{AF} = \pi \cdot \langle v, v_1\rangle,\ v_1 \rightarrow^C v_{LF},\\
\mathsf{wf}(a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \,|\, a_{RaC}.\pi_{RaC}),\ \mathsf{wf}(a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \cdot \langle v_1, v_{LF}\rangle \,|\, a_{RaC}.\pi_{RaC})\\
\vdash\ a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \cdot \langle v_1, v_{LF}\rangle \,|\, a_{RaC}.\pi_{RaC} : \textsc{Always}_{-\mathsf{ret}\ v_1}\textsc{Spec}'
\end{array}$$

As a result of the above steps we computed a symbolic next-state $a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \cdot \langle v_1, v_{LF}\rangle \,|\, a_{RaC}.\pi_{RaC}$ from the original symbolic state $a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \,|\, a_{RaC}.\pi_{RaC}$ and updated the assumptions on its parameters.

Proof search continues by showing that in this symbolic next-state the formula $\textsc{Always}_{-\mathsf{ret}\ v_1}\textsc{Spec}'$ is true. This amounts to showing that (by applying NuR, ApproxR, and AndR) in this state $\textsc{Spec}'$ is true, and in all possible next states within the call (*i.e.* those that are reached through transitions which are not labeled v $\mathsf{ret}\ v_1$), again $\textsc{Always}_{-\mathsf{ret}\ v_1}\textsc{Spec}'$ holds. $\textsc{Spec}'$ says that a_{RaC} does not call v_{GT}, thus it follows immediately that $a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \cdot \langle v_1, v_{LF}\rangle \,|\, a_{RaC}.\pi_{RaC} : \textsc{Spec}'$ is satisfied from the wellformedness of the current state: a_{AF} is the active applet, thus a_{RaC} cannot issue calls.

We consider all possible next states of $a_P.\pi_P \,|\, a_{AF}.\pi_{AF} \cdot \langle v_1, v_{LF}\rangle \,|\, a_{RaC}.\pi_{RaC}$ within the call. In most cases, we detect a loop and immediately can apply the discharge condition, but in the case that the next state is reached because a_{AF} has sent out an external call, we cannot do this. Here we have to use the assumption on a_{AF}, which says that such a call can only be to v_{GT} or v_{GB}. Thus there are two possible symbolic next states and for both these states we have to show that $\textsc{Always}_{-\mathsf{ret}\ v_1}\textsc{Spec}'$ holds. This is done by showing that in this state $\textsc{Spec}'$ holds (either because a_P is active, thus a_{RaC} cannot send a message, or because of the specification on a_{RaC}, which says that it does not make outgoing calls from within v_{GB}), and that in all possible next states again $\textsc{Always}_{-\mathsf{ret}\ v_1}\textsc{Spec}'$ holds. Thus, proof search continues in the same way from these states, considering all possible computations, until all branches of the proof tree can be discharged, therewith concluding our proof.

Notice that the construction of the proof is exactly prescribed by the structure of the formula. Therefore we believe that having a tailored proof tool and well-developed proof strategies will help us to achieve a sufficiently high degree of automation in constructing the decomposition correctness proofs.

7 Conclusion and Future Work

This paper introduces a language-independent framework for the specification and verification of secure applet interactions in open platforms. It is shown that the framework can be instantiated to JavaCard and that it allows the decomposition of global properties about applet interactions into local properties of applets, as shown on a realistic case study.

Related Work. Our program models can alternatively be cast in terms of context-free processes; for these there exist algorithmic verification techniques *w.r.t.* modal μ-calculus specifications [4]. The development of our program model follows earlier work by Jensen *et al.* [11] which addresses security properties expressible as stack invariants. These form a strict subset of the properties which can be expressed in our framework, but allow for more efficient model checking procedures.

Future Work. Our primary objective is to complete our work on the proof system by studying completeness and decidability issues for suitable fragments of the logic. This is crucial for providing adequate automated tools for property decomposition. Further, we intend to combine such tools with off-the-shelf model checkers, so that local properties of applets can be checked automatically. We believe that such a combination will provide an effective environment to address further, more challenging, case studies.

In a different line of work, it would be of interest to enhance our model with data – so as to capture properties such as "Action Credit increases the balance of the Purse Component" – and with multi-threading, but the theoretical underpinnings of such extensions remain to be unveiled.

Acknowledgment

The authors would like to thank Christoph Sprenger at SICS and the anonymous referees for many useful remarks on the manuscript.

References

1. Bandera project. `http://www.cis.ksu.edu/santos/bandera`
2. G. Barthe, D. Gurov, and M. Huisman. Compositional specification and verification of control flow based security properties of multi-application programs. In *Proceedings of Workshop on Formal Techniques for Java Programs (FTfJP)*, 2001.

3. 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, *Proceedings of the workshop on secure architectures and information flow*, volume 32 of *Elect. Notes in Theor. Comp. Sci.* Elsevier Publishing, 2000.
4. O. Burkart and B. Steffen. Model checking the full modal mu-calculus for infinite sequential processes. In *Proceedings of ICALP'97*, number 1256 in LNCS, pages 419–429, 1997.
5. A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *Software Tools for Technology Transfer (STTT)*, 2/4:410–425, 2000.
6. J. Corbett, M. Dwyer, J. Hatcliff, and Robby. A language framework for expressing checkable properties of dynamic software. In K. Havelund, J. Penix, and W. Visser, editors, *SPIN Model Checking and Software Verification*, number 1885 in LNCS. Springer, 2000.
7. M. Dam and D. Gurov. Compositional verification of CCS processes. In D. Bjørner, M. Broy, and A.V. Zamulin, editors, *Proceedings of PSI'99*, number 1755 in LNCS, pages 247–256, 1999.
8. M. Dam and D. Gurov. μ-calculus with explicit points and approximations. *Journal of Logic and Computation*, 2001. To appear.
9. L.-å. Fredlund, D. Gurov, T. Noll, M. Dam, T. Arts, and G. Chugunov. A verification tool for Erlang. *Software Tools for Technology Transfer (STTT)*, 2002. To appear.
10. G. Holzmann. The model checker SPIN. *Transactions on Software Engineering*, 23(5):279–295, 1997.
11. T. Jensen, D. Le Métayer, and T. Thorn. Verification of control flow based security policies. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 89–103. IEEE Computer Society Press, 1999.
12. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
13. X. Leroy. Java bytecode verification: an overview. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of CAV'01*, number 2102 in LNCS, pages 265–285. Springer, 2001.
14. X. Leroy. On-card bytecode verification for JavaCard. In I. Attali and T. Jensen, editors, *Smart Card Programming and Security (E-Smart 2001)*, number 2140 in LNCS, pages 150–164. Springer, 2001.
15. S. Owre, J. Rushby, N. Shankar, and F von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, 1995.
16. A. Simpson. Compositionality via cut-elimination: Hennesy-Milner logic for an arbitrary GSOS. In *Proceedings of the Tenth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 420–430, 1995.
17. G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings of POPL'98*, pages 355–364. ACM Press, 1998.

A Transition Rules

We use $\mathbf{loc}_a\, v$ as abbreviation for $\mathbf{app}\, v = a$.

Transition Rules for Composite States (With Symmetric Counterparts).

$$[\mathsf{tau}]\ \frac{\mathcal{A}_1 \xrightarrow{\tau} \mathcal{A}_1'}{\mathcal{A}_1 \mid \mathcal{A}_2 \xrightarrow{\tau} \mathcal{A}_1' \mid \mathcal{A}_2}$$

$$[\mathsf{synchro}]\ \frac{\mathcal{A}_1 \xrightarrow{v_1\ l?\ v_2} \mathcal{A}_1' \qquad \mathcal{A}_2 \xrightarrow{v_1\ l!\ v_2} \mathcal{A}_2'}{\mathcal{A}_1 \mid \mathcal{A}_2 \xrightarrow{v_1\ l\ v_2} \mathcal{A}_1' \mid \mathcal{A}_2'}\ l \in \mathsf{call}, \mathsf{ret}$$

$$[\mathsf{propagation}]\ \frac{\mathcal{A}_1 \xrightarrow{v_1\ l?!\ v_2} \mathcal{A}_1' \qquad \neg\mathbf{loc}_{\mathcal{A}_2}\, v_1 \qquad \neg\mathbf{loc}_{\mathcal{A}_2}\, v_2}{\mathcal{A}_1 \mid \mathcal{A}_2 \xrightarrow{v_1\ l?!\ v_2} \mathcal{A}_1' \mid \mathcal{A}_2}\ l \in \mathsf{call}, \mathsf{ret}$$

Applet Transition Rules.

$$[\mathsf{local\ call}]\ \frac{v_1 \rightarrow^{C} v_2 \qquad \mathbf{loc}_a\, v_1 \qquad \mathbf{loc}_a\, v_2}{a.\pi \cdot \langle v, v_1\rangle \xrightarrow{v_1\ \mathsf{call}\ v_2} a.\pi \cdot \langle v, v_1\rangle \cdot \langle v_1, v_2\rangle}$$

$$[\mathsf{local\ return}]\ \frac{v_1 \rightarrow^{T} v_2 \qquad \mathbf{loc}_a\, v_1 \qquad \mathbf{loc}_a\, v_3 \qquad \mathbf{ret}\, v_3}{a.\pi \cdot \langle v, v_1\rangle \cdot \langle v_1, v_3\rangle \xrightarrow{v_3\ \mathsf{ret}\ v_1} a.\pi \cdot \langle v, v_2\rangle}$$

$$[\mathsf{local\ transfer}]\ \frac{v_1 \rightarrow^{T} v_2 \qquad \mathbf{loc}_a\, v_1 \qquad v_1 \not\rightarrow^{C}}{a.\pi \cdot \langle v, v_1\rangle \longrightarrow a.\pi \cdot \langle v, v_2\rangle}$$

$$[\mathsf{send\ call}]\ \frac{v_1 \rightarrow^{C} v_2 \qquad \mathbf{loc}_a\, v_1 \qquad \neg\mathbf{loc}_a\, v_2}{a.\pi \cdot \langle v, v_1\rangle \xrightarrow{v_1\ \mathsf{call!}\ v_2} a.\pi \cdot \langle v, v_1\rangle \cdot \langle v_1, v_2\rangle}$$

$$[\mathsf{receive\ call}]\ \frac{v_1 \rightarrow^{C} v_2 \qquad \neg\mathbf{loc}_a\, v_1 \qquad \mathbf{loc}_a\, v_2 \qquad \neg\mathsf{active}\, a}{a.\pi \xrightarrow{v_1\ \mathsf{call?}\ v_2} a.\pi \cdot \langle v_1, v_2\rangle}$$

$$[\mathsf{send\ return}]\ \frac{\neg\mathbf{loc}_a\, v_1 \qquad \mathbf{loc}_a\, v_2 \qquad \mathbf{ret}\, v_2}{a.\pi \cdot \langle v_1, v_2\rangle \xrightarrow{v_2\ \mathsf{ret!}\ v_1} a.\pi}$$

$$[\mathsf{receive\ return}]\ \frac{v_1 \rightarrow^{T} v_2 \qquad \mathbf{loc}_a\, v_1 \qquad \neg\mathbf{loc}_a\, v_3 \qquad \neg\mathbf{loc}_a\, v_4}{a.\pi \cdot \langle v, v_1\rangle \cdot \langle v_1, v_3\rangle \xrightarrow{v_4\ \mathsf{ret?}\ v_1} a.\pi \cdot \langle v, v_2\rangle}$$

B Remaining Proof Rules of the Proof System

Structural Rules.

$$(\mathsf{Id})\ \frac{}{\Gamma,\ \gamma \vdash \gamma,\ \Delta}$$

$$(\mathsf{Cut})\ \frac{\Gamma,\ \gamma \vdash \Delta \qquad \Gamma \vdash \gamma,\ \Delta}{\Gamma \vdash \Delta}$$

$$(\mathsf{WeakL})\ \frac{\Gamma \vdash \Delta}{\Gamma,\ \gamma \vdash \Delta} \qquad (\mathsf{WeakR})\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \gamma,\ \Delta}$$

Atomic Formula Rules.

$$(\mathsf{EqSymL})\ \frac{\Gamma,\ t_2 = t_1\ \vdash\ \Delta}{\Gamma,\ t_1 = t_2\ \vdash\ \Delta} \qquad (\mathsf{EqReflR})\ \frac{\cdot}{\Gamma\ \vdash\ t = t,\ \Delta}$$

$$(\mathsf{EqSubstL})\ \frac{\Gamma[t/x]\ \vdash\ \Delta[t/x]}{\Gamma,\ x = t\ \vdash\ \Delta}$$

$$(\mathsf{EqNilL})\ \frac{\cdot}{\Gamma,\ \epsilon = \Pi \cdot \langle v_1, v_2\rangle\ \vdash\ \Delta}$$

$$(\mathsf{EqConsL})\ \frac{\Gamma,\ \Pi = \Pi',\ v_1 = v_1',\ v_2 = v_2'\ \vdash\ \Delta}{\Gamma,\ \Pi \cdot \langle v_1, v_2\rangle = \Pi' \cdot \langle v_1', v_2'\rangle\ \vdash\ \Delta}$$

$$(\mathsf{EqConsR})\ \frac{\Gamma\ \vdash\ \Pi = \Pi',\ \Delta \qquad \Gamma\ \vdash\ v_1 = v_1',\ \Delta \qquad \Gamma\ \vdash\ v_2 = v_2',\ \Delta}{\Gamma\ \vdash\ \Pi \cdot \langle v_1, v_2\rangle = \Pi' \cdot \langle v_1', v_2'\rangle,\ \Delta}$$

$$(\mathsf{LocComL})\ \frac{\Gamma,\ \mathsf{local}_{\mathcal{A}_1}\ v\ \vdash\ \Delta \qquad \Gamma,\ \mathsf{local}_{\mathcal{A}_2}\ v\ \vdash\ \Delta}{\Gamma,\ \mathsf{local}_{\mathcal{A}_1 | \mathcal{A}_2}\ v\ \vdash\ \Delta}$$

$$(\mathsf{LocComR})\ \frac{\Gamma\ \vdash\ \mathsf{local}_{\mathcal{A}_1}\ v,\ \mathsf{local}_{\mathcal{A}_2}\ v,\ \Delta}{\Gamma\ \vdash\ \mathsf{local}_{\mathcal{A}_1 | \mathcal{A}_2}\ v,\ \Delta}$$

Logical Rules.

$$(\mathsf{PredL})\ \frac{\Gamma,\ \sigma\ \vdash\ \Delta}{\Gamma,\ \mathcal{A} : \sigma\ \vdash\ \Delta} \qquad (\mathsf{PredR})\ \frac{\Gamma\ \vdash\ \sigma, \Delta}{\Gamma\ \vdash\ \mathcal{A} : \sigma,\ \Delta}$$

$$(\mathsf{NotL})\ \frac{\Gamma\ \vdash\ \mathcal{A} : \phi,\ \Delta}{\Gamma,\ \mathcal{A} : \neg\phi\ \vdash\ \Delta} \qquad (\mathsf{NotR})\ \frac{\Gamma,\ \mathcal{A} : \phi\ \vdash\ \Delta}{\Gamma\ \vdash\ \mathcal{A} : \neg\phi,\ \Delta}$$

$$(\mathsf{AndL})\ \frac{\Gamma,\ \mathcal{A} : \phi,\ \mathcal{A} : \psi\ \vdash\ \Delta}{\Gamma,\ \mathcal{A} : \phi \wedge \psi\ \vdash\ \Delta} \qquad (\mathsf{AndR})\ \frac{\Gamma\ \vdash\ \mathcal{A} : \phi,\ \Delta \qquad \Gamma\ \vdash\ \mathcal{A} : \psi,\ \Delta}{\Gamma\ \vdash\ \mathcal{A} : \phi \wedge \psi,\ \Delta}$$

$$(\mathsf{AllL})\ \frac{\Gamma,\ \mathcal{A} : \phi[t/x]\ \vdash\ \Delta}{\Gamma,\ \mathcal{A} : \forall x.\phi\ \vdash\ \Delta} \qquad (\mathsf{AllR})\ \frac{\Gamma\ \vdash\ \mathcal{A} : \phi,\ \Delta}{\Gamma\ \vdash\ \mathcal{A} : \forall x.\phi,\ \Delta}\ \text{fresh } x$$

$$(\mathsf{BoxL})\ \frac{\Gamma\ \vdash\ \mathcal{A} \xrightarrow{\alpha} \mathcal{A}',\ \Delta \qquad \Gamma,\ \mathcal{A}' : \phi\ \vdash\ \Delta}{\Gamma,\ \mathcal{A} : [\alpha]\,\phi\ \vdash\ \Delta} \qquad (\mathsf{BoxR})\ \frac{\Gamma, \mathcal{A} \xrightarrow{\alpha} \mathcal{X}\ \vdash\ \mathcal{X} : \phi,\ \Delta}{\Gamma\ \vdash\ \mathcal{A} : [\alpha]\,\phi,\ \Delta}\ \text{fresh } \mathcal{X}$$

Auxiliary Rules for Intersection.

$$(\mathsf{IntR})\ \frac{\cdot}{\Gamma\ \vdash\ \mathsf{intersect}(a.\Pi_1, a.\Pi_2),\ \Delta} \qquad (\mathsf{IntReflR})\ \frac{\cdot}{\Gamma\ \vdash\ \mathsf{intersect}(\mathcal{A}, \mathcal{A}),\ \Delta}$$

$$(\mathsf{IntComL})\ \frac{\Gamma,\ \mathsf{intersect}(\mathcal{A}_1, \mathcal{A}_3)\ \vdash\ \Delta \qquad \Gamma,\ \mathsf{intersect}(\mathcal{A}_2, \mathcal{A}_3)\ \vdash\ \Delta}{\Gamma,\ \mathsf{intersect}(\mathcal{A}_1 \,|\, \mathcal{A}_2, \mathcal{A}_3)\ \vdash\ \Delta}$$

$$(\mathsf{IntComR})\ \frac{\Gamma\ \vdash\ \mathsf{intersect}(\mathcal{A}_1, \mathcal{A}_3),\ \mathsf{intersect}(\mathcal{A}_2, \mathcal{A}_3),\ \Delta}{\Gamma\ \vdash\ \mathsf{intersect}(\mathcal{A}_1 \,|\, \mathcal{A}_2, \mathcal{A}_3),\ \Delta}$$

(with symmetric counterparts).