Distributed programming with distributed authorization

Kumar Avijit Anupam Datta Robert Harper
Carnegie Mellon University
kavijit@cs.cmu.edu danupam@andrew.cmu.edu rwh@cs.cmu.edu

Abstract

We propose a programming language, called PCML$_5$, for building distributed applications with distributed access control. Target applications include web-based systems in which programs must compute with stipulated resources at different sites. In such a setting, access control policies are *decentralized* (each site may impose restrictions on access to its resources without the knowledge of or cooperation with other sites) and *spatially distributed* (each site may store its policies locally). To enforce such policies PCML$_5$ employs a distributed proof-carrying authorization framework in which sensitive resources are governed by reference monitors that authenticate principals and demand logical proofs of compliance with site-specific access control policies. The language provides primitive operations for *authentication*, and *acquisition* of proofs from local policies. The type system of PCML$_5$ enforces locality restrictions on resources, ensuring that they can only be accessed from the site at which they reside, and enforces the authentication and authorization obligations required to comply with local access control policies. This ensures that a well-typed PCML$_5$ program cannot incur a runtime access control violation at a reference monitor for a controlled resource.

1. Introduction

The increasing importance of web-based services motivates the consideration of languages to support the construction of applications that involve computation and resources distributed over multiple sites. In such a setting, site administrators naturally wish to restrict access to their resources to certain authorized principals. These access control policies are inherently *decentralized* because it is unreasonable to assume a central authority governing all resources in a networked system. Morever, these policies are *spatially distributed* since each site may store its policy locally. This paper is concerned with the design of a kernel programming language, which we call PCML$_5$, to support the construction of distributed programs that comply with such distributed access control policies. PCML$_5$ employs a *distributed proof-carrying authorization* (DPCA) framework to enforce such policies.

Proof-carrying authorization (PCA) was introduced by Appel and Felten (Appel and Felten 1999) and developed further in subsequent work (Bauer et al. 2001, 2005; Garg 2009). In this framework, access control policies are specified as theories in an *authorization logic* (Lampson et al. 1992; Abadi et al. 1993; Abadi 2003, 2006a; Garg and Pfenning 2006). Each protected resource is governed by a reference monitor that authenticates principals and demands a formal proof of authorization to control access to it. Access control is thereby reduced to proof checking (not proof search!) in the authorization logic. The formal proofs of authorization required for access may be logged by the reference monitor to provide an audit trail that may be used to analyze permissible, but unintended, accesses.

In a centralized setting the authorization policy may be thought of as a single logical theory that resides in one place and governs access to all resources in a system. In a distributed setting it is natural to consider a *distributed access control policy* consisting of the *local* authorization policies residing at each site in a distributed system, each being a logical theory within a *global* authorization logic specifying the rules of access control. The administrative authority for each site controls the access control policy for its resources, stating the conditions under which they may be accessed. These policies are thus both decentralized and spatially distributed. To be sufficiently expressive in a distributed setting, the authorization logic must include the means to make assertions on behalf of a principal, and may involve principals other than the controlling authority. As a simple example, a conventional access control list may be thought of as a conjunction of propositions of the form $\langle k \rangle \texttt{mayrd}(d, k')$, which should be read as stating that the controlling principal k says that the principal k' may read the database d. A number of authorization logics, beginning with the seminal work by Lampson et al. (Lampson et al. 1992), provide support for such assertions. The problem of specifying and enforcing policies in a scenario with spatially distributed policies was first studied in the context of authentication in the Taos operating system (Wobber et al. 1994), and later in the context of trust management systems (Blaze et al. 1996; Clarke et al. 2001; Li et al. 2002).

PCML$_5$ enforces distributed access control policies using a *distributed proof-carrying authorization (DPCA)* framework. DPCA is a generalization of PCA to a setting in which both computing resources and access control policies are distributed across multiple sites, each with its own controlling authority. Resources at each site are controlled by a proof-carrying reference monitor that demands a proof in distributed authorization logic as a condition for access. The proofs themselves are constructed according to the rules of the logic augmented by the policies in force at each site in a distributed system. A distributed program must execute a *distributed proof construction* algorithm to gather the relevant fragments of the policy from the various sites in the system, combining them to form proofs that are presented to the reference monitor governing each resource. Moreover, a program is executed on behalf of a principal, as evidenced by the possession of a credential obtained through a standard authentication protocol. The reference monitor for a resource requires both the authentication credentials and the authorization proof to control access to it. (In a fuller treatment than is considered here, the reference monitor could also require further

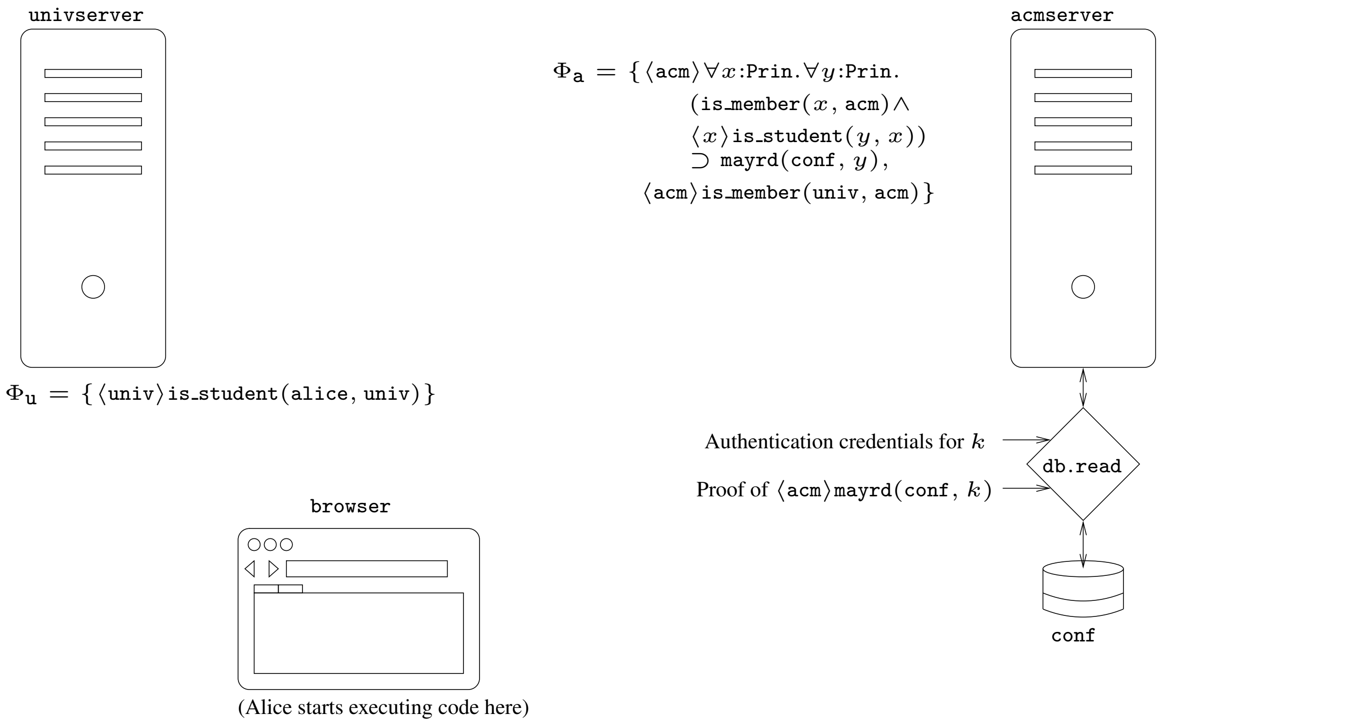
univserver
acmserver
$\Phi_{\mathtt{a}} = \{ \langle \mathtt{acm} \rangle \forall x{:}\mathtt{Prin}.\forall y{:}\mathtt{Prin}.(\mathtt{is\_member}(x, \mathtt{acm}) \wedge \langle x \rangle \mathtt{is\_student}(y, x)) \supset \mathtt{mayrd}(\mathtt{conf}, y), \langle \mathtt{acm} \rangle \mathtt{is\_member}(\mathtt{univ}, \mathtt{acm}) \}$
$\Phi_{\mathtt{u}} = \{ \langle \mathtt{univ} \rangle \mathtt{is\_student}(\mathtt{alice}, \mathtt{univ}) \}$
Authentication credentials for k
Proof of $\langle \mathtt{acm} \rangle \mathtt{mayrd}(\mathtt{conf}, k)$
db.read
browser
conf
(Alice starts executing code here)

Figure 1. An example scenario for distributed PCA

information such as the time of day, or a certificate revocation list to enforce access control restrictions.)

A simple example of such a system is given in Figure 1. Here we consider three sites, viz., `browser`, `univserver`, and `acmserver`. The site `acmserver` is governed by the principal `acm`, and has a database `conf` protected by a reference monitor that mediates accesses to it. For illustration, we only consider an API `db.read` for reading the database. Each site w is associated with a local security policy, denoted by Φ_w, that governs access to its resources. As an example application written in PCML$_5$, we consider in Section 8 a distributed program for reading the database at `acmserver` using its proof-carrying interface `db.read`. The program first authenticates the principal on whose behalf it is running. Then it constructs a proof for the authenticated principal. In this example, a proof of $\langle \mathtt{acm} \rangle \mathtt{mayrd}(\mathtt{conf}, \mathtt{alice})$ can be constructed by suitably combining the three facts from local security policies. These facts are acquired by doing *local* proof searches. One way to construct the proof is to first assemble a proof of $\langle \mathtt{univ} \rangle \mathtt{is\_student}(\mathtt{alice}, \mathtt{univ})$ at `univserver`, followed by constructing a proof of

$$\langle \mathtt{univ} \rangle \mathtt{is\_student}(\mathtt{alice}, \mathtt{univ}) \supset \langle \mathtt{acm} \rangle \mathtt{mayrd}(\mathtt{conf}, \mathtt{alice})$$

at `acmserver`. The final proof is assembled by moving both the proofs to `acmserver` where they are combined using an $\supset$-elimination. This proof is then used at the call-site for database access at `acmserver`. The results of this call are brought back to `browser`.

As we noticed above, an authorization proof usually consists of facts obtained at various sites. Such proofs, however, are verified by reference monitors that have apriori access only to the local policies at their sites. In practice, digital certificates are used as atomic proofs of assertions made by principals. In order to assert P, a principal k simply signs P using his/her private key. In this manner, digital certificates provide an unforgeable, irrefutable, and univerally checkable means of establishing assertions. Local security policies contain a digital certificate for each proposition of the form $\langle k \rangle P$ in the policy. The certificate is substituted for any hypothesis about the proposition in any proof acquired locally, thereby yielding a closed proof that can be verified by any reference monitor.

The run-time attack model we consider is based on the DPCA framework just sketched. In particular, all programs, whether written in PCML$_5$ or not, are assumed to execute against a DPCA-based access control system to ensure the security of protected resources. Untyped programs may incur run-time failures caused by improper access to a monitored resource, in particular, failure to provide proper credentials or to provide a valid proof of authority for an access.

PCML$_5$ is designed to facilitate construction of non-malicious distributed programs that comply with a distributed access control policy. The type system of PCML$_5$ extends that of the ML$_5$ language for distributed computing (Murphy 2008; Murphy et al. 2004) to express the authentication and authorization obligations imposed by the DPCA run-time. As the central theorem of this paper, we obtain the guarantee that a precisely characterized class of well-typed PCML$_5$ program cannot incur a run-time access control violation at a reference monitor of a protected resource (Theorems 7.6. 7.5). This is achieved by the following mechanisms:

- As in its precursor, ML$_5$, the PCML$_5$ type system enforces *locality* restrictions on resources that ensure that access must take place at the site governing that resource.
- A distributed authorization logic is integrated into the type system of PCML$_5$ so that the demands for proofs at the call sites of reference monitors can be expressed and enforced within the language. PCML$_5$ does not depend on any particular authorization logic; we use a particularly simple one developed by Garg et al. (Garg and Pfenning 2006) to illustrate the main ideas.

- A primitive is provided for doing proof search using local authorization policies at sites. Proofs obtained locally can be combined together to obtain proofs valid with respect to the global authorization policy.
- A primitive is provided to authenticate as a principal in the system. Authentication demands of a reference monitor can be expressed using its type.

2. Authorization logic

The presentation of PCML$_5$ in this paper uses the GP authorization logic (Garg and Pfenning 2006), which we summarize in this section. However, we emphasize that we pick a specific logic for illustration only. Since the logic is integrated into the type system using higher-order abstract syntax (as described in Section 5), the PCML$_5$ language can be easily coupled with other authorization logics that can be encoded in the framework.

$$\begin{array}{rlll}
\text{Sorts} & s & ::= & \mathtt{prin} \mid \mathtt{db} \\
\text{First-order terms} & a & ::= & \alpha \mid \mathbf{a} \\
\text{Predicate symbols} & p & & \\
\text{Propositions} & P & ::= & P_1 \supset P_2 \mid P_1 \wedge P_2 \\
 & & \mid & \forall \alpha{:}s.P \mid \langle a \rangle P \mid p(a_1, \ldots, a_n) \\
\text{Basic judgments} & J & ::= & a{:}s \mid P\ \mathtt{true} \mid a\ \mathtt{affirms}\ P \\
\text{Proposition context} & \Phi & ::= & \cdot \mid \Phi, P\ \mathtt{true} \\
\text{Term context} & \mathbf{\Delta} & ::= & \cdot \mid \mathbf{\Delta}, \alpha{:}s \\
\text{Signature} & \mathbf{\Sigma} & ::= & \cdot \mid \mathbf{\Sigma}, \mathbf{a}{:}s \mid \mathbf{\Sigma},\ p{:}(s_1, \ldots, s_n)
\end{array}$$

The logic is parameterized by a signature $\mathbf{\Sigma}$ that fixes the arity of the predicates; $p{:}(s_1, \ldots, s_n)$ denotes that p is a predicate of arity n with s_i being the sort of its ith argument. There are atleast two sorts, viz., `prin` and `db`, representing principals and databases respectively. The signature also fixes principal and database constants $\mathbf{a}$.

Apart from the standard propositional connectives, the logic features a family of modalities indexed by principals, to express propositions affirmed by principals. The modal proposition $\langle a \rangle P$ (read as "a says P") expresses that proposition P is endorsed by principal a. There are three forms of basic judgments: the sorting judgment $a{:}s$, which means that the first-order term a is of the sort s; the truth judgment P `true`; and the affirmation judgment a `affirms` P, which means that a (a term of the sort `prin`) endorses P.

We use $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} J$ to mean the hypothetical judgment J from assumptions $\mathbf{\Delta}$ and Φ, under the signature $\mathbf{\Sigma}$. The superscript $\mathcal{L}$ differentiates entailment in the logic from that in hypothetical judgments of PCML$_5$ which we shall introduce later. We collect hypotheses about sorting judgments $x{:}s$ under the context $\mathbf{\Delta}$; hypotheses of truth judgments P `true` are written as Φ. We do not need to form hypotheses about affirmation judgments in this logic.

In order to illustrate the *says* modality, we present selected rules from a natural deduction for the logic in Figure 2. Rule (`truaff`) says that true propositions are affirmed by all principals. Rule ($\langle\rangle$-I) internalizes the judgment a `affirms` P as the proposition $\langle a \rangle P$. Rule($\langle\rangle$-E) eliminates the modality in $\langle a \rangle P_1$ by using a proof of $\langle a \rangle P_1$ to cut a hypothesis P_1 `true` in a derivation of affirmation a `affirms` P_2 by the same principal a.

3. Distributed policies and proof-checking

In our setup, each site is governed by an administrative principal who decides the authorization policy at that site. We shall use the notation $\bar{w}$ to denote the governing principal at site w. We model the local policy as a context of propositions declared to be true by the governing principal. From a global perspective, a proposition P declared by the principal $\bar{w}$ has the force of the proposition $\langle \bar{w} \rangle P$.

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{true} \qquad \mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a : \mathtt{prin}}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \mathtt{affirms}\ P}(\mathtt{truaff})$$

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \mathtt{affirms}\ P}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \langle a \rangle P\ \mathtt{true}}(\langle\rangle\text{-I})$$

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \langle a \rangle P_1\ \mathtt{true} \qquad \mathbf{\Delta}; \Phi, P_1\ \mathtt{true} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \mathtt{affirms}\ P_2}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \mathtt{affirms}\ P_2}(\langle\rangle\text{-E})$$

Figure 2. Selected rules from natural deduction for the authorization logic. (Reproduced from (Garg and Pfenning 2006)).

Thus the local policy at w, called Φ_w, is a context of the form:

$$\Phi_w ::= \langle \bar{w} \rangle P_1\ \mathtt{true}, \ldots, \langle \bar{w} \rangle P_n\ \mathtt{true}$$

A proof presented to a reference monitor may contain facts from local policies at various sites. Logically, such a proof is valid under the union of local policies at all sites. We refer to this union as the *amalgamated* authorization policy and denote it by

$$\Phi = \bigcup_{w \in \mathtt{Wld}} \langle \bar{w} \rangle \Phi_w$$

where `Wld` is the set of all sites. We use the same notation for amalgamated policy as a proposition context from Section 2 in order to highlight that a policy is just a context in the logic. In implementations, evidence for the local policy at a site is provided by digitally signed certificates, which can be verified by reference monitors at any site. A certificate for proposition P signed by a principal a is unforgeable and establishes the judgment $\langle a \rangle P$ `true` irrefutably. Verification of a proof constructed from such evidences requires checking the validity of these certificates. We view these evidences as implementing the amalgamated authorization policy. Therefore, our formalism directly uses the full policy Φ for the purpose of proof-checking at reference monitors.

In practical distributed authorization systems for trust management (Blaze et al. 1996) and other applications (Bauer et al. 2005) policy statements made by an administrator at its site w are often cached at another site w' in order to aid distributed proof construction. For example, consider the authorization policy at `univserver` from Figure 1. The evidence for $\langle \mathtt{univ} \rangle \mathtt{is\_student}(\mathtt{alice}, \mathtt{univ})$ may be cached at Alice's machine `browser`. In this paper, we do not consider caching of proofs at sites. The policy available at a site is restricted to only consist of declarations made by the principal governing that site.

4. Overview of PCML$_5$

We shall now informally discuss the key ideas behind PCML$_5$. There are three main aspects of PCML$_5$:

1. PCML$_5$ is a language for writing distributed programs that interact with resources which are distributed across different administrative domains. The type system tracks the site where a computation is executed. This allows the language to enforce the requirement that a resource can be accessed only from the site where it is situated.
2. PCML$_5$ uses an authorization logic as a part of its type system. The logic is used to express within the language authorization proof obligations that are imposed by reference monitors.
3. PCML$_5$ allows construction of proofs by combining the results of a distributed proof search with proof constructors from au-

thorization logic. The type system statically ensures that proofs passed to a reference monitor satisfy its proof specifications. This ensures that a call made to a reference monitor from a well-typed PCML$_5$ program never incurs a runtime error due to failure of verification of proof.

We now elaborate on each of these aspects.

4.1 Distribution of computation

PCML$_5$ is based on ML$_5$ (Murphy 2008; Murphy et al. 2004), which is a distributed programming language based on propositions-as-types interpretation of the modal logic Intuitionistic S5 where modal worlds are interpreted as sites in a distributed system. The central idea that PCML$_5$ inherits from ML$_5$ is that terms are classified using types *relative* to sites. The typing judgment $\Gamma \vdash m : A@w$ means that under the hypotheses about variables in Γ, the term m is typed as A for the site w where it is to be evaluated.

The idea of typing relative to worlds is central to writing distributed programs since it allows the type system to track where different pieces of code are meant to execute. A term m for the site w' can be remotely executed from a world w by doing a $\mathtt{get}[w']m$ at w. This causes computation to move to w' to execute m, and the result is brought back to w, provided type of the result is mobile. We discuss mobility later in Section 5.3 after having introduced the formalism.

4.2 Authorization logic

PCML$_5$ incorporates an authorization logic as static constructors using higher-order abstract syntax. Constructors are classified by kinds. Propositions belong to the kind Prop and proofs of a proposition P are kinded as $\mathtt{Prf}(A)$ where A is the representation of the proposition P. Proof of an affirmation judgment a affirms P is represented as a constructor of kind A_1 Affirms A_2, where A_1 represents the principal a, and A_2 is the representation of P. As we shall see later, this translation provides a compositional bijection between propositions, proofs, and affirmations, on one hand, and constructors of kind Prop, $\mathtt{Prf}()$, and Affirms , on the other.

The main requirement from this translation is to be able to reflect the consequence relation of logic in an adequate manner using PCML$_5$ as is done in LF (Harper et al. 1993). Thus if there is a proof of P_2 hypothetical in a proof of P_1 in the logic, one can write a constructor of kind $\mathtt{Prf}(A_2)$ with a free variable of kind $\mathtt{Prf}(A_1)$ in the language, and vice versa, where A_i's are the representations of propositions P_i's. We shall make this relation precise once we introduce the formalism.

In order to ensure that adequacy is without doubt, we follow a phase distinction (Cardelli 1988) between static and dynamic phases of PCML$_5$. The constructor level is constrained to be pure in the sense that constructors do not depend on runtime terms. This way terms can be arbitrarily effectful, but they do not affect the types in any way.

4.3 Representation of principals and resources

Principals and resources appear in propositions and proofs, and also occur as runtime values. In order to maintain a phase distinction, we differentiate between the runtime and the compile-time representations of principals and resources, yielding a dual representation for them:

1. They appear as static constructors of kind Prin and Db. These constructors appear in types and propositions, such as $\mathtt{mayrd}(d, p)$.
2. They are also represented as runtime values that mediate access to them. The type system tracks the identities of these runtime values by linking them to their static counterparts using their types. The type $\mathtt{db}(A)$ is a singleton type that represents runtime database values indexed by A, which has the kind Db. We refer to the static representations, like A, as indices.

The two representations are created together. For databases, this is done using a primitive operation db.open: $\exists\alpha{::}\mathtt{Db}.\mathtt{db}(\alpha)$.

In the case of principals, we need runtime representations of only those principals that are authenticated. The type $\mathtt{Iam}(A)$ represents the type of authentication tokens for $A{::}\mathtt{Prin}$.

4.4 Runtime acquisition of proofs

PCML$_5$ enables querying local security policies at all sites for proofs. This is done using a primitive operation $\mathtt{acquire}[A]\{\alpha.m_1 \mid m_2\}$ that, when executed at a site w, does a local proof search for a proof of A using only Φ_w, the policy at w. The language itself does not specify or depend on any particular proof search strategy. It is however reasonable to assume that any strategy used in an implementation should at least succeed in finding a proof when one is present as an atomic fact in the policy.

The key idea behind acquire is that it enables acquiring facts from security policies and incorporating them in the program at runtime. The results of the runtime query, if successful, discharge the hypothesis α bound in the branch m_1. In case the query succeeds, the result is substituted for α in m_1 and execution proceeds with m_1. In case the query fails, m_2 is executed.

5. Syntax and static semantics

Figure 3 presents the syntax of PCML$_5$. We divide the syntax into three levels: runtime terms, constructors, and kinds. Runtime terms are classified by types, which are a subset of constructors, and constructors are classified by kinds. The language is parameterized by two forms of signatures:

1. Σ_c: is the signature that introduces constructor constants, $\mathbf{a}{\Rightarrow}K$. Besides constant principals and databases, this signature introduces the encoding of authorization logic.
2. Σ_t: is the term signature and it gives types to externally implemented API constants $\mathbf{c}{:}\forall\langle\alpha_1{\Rightarrow}K_1, \ldots, \alpha_n{\Rightarrow}K_n\rangle.A \mathbin{@} w$.

In order to avoid normalization at the constructor level, we present only the normal (canonical) forms in the style of Canonical LF (Harper and Licata 2007). The idea is to exclude β and η redexes altogether by rearranging the constructors as *neutral* (N) and *normal* (A) forms so that all occurrences of eliminations appear before any introduction. Since we admit only canonical constructors, we use hereditary substitution (Watkins et al. 2002) to keep constructors canonical upon substitution, details of which are omitted here.

5.1 Judgment forms

The basic judgment forms used in static semantics are:

K kind	K is a well-formed kind
$A \Leftarrow K$	A is checked for the kind K
$N \Rightarrow K$	K is synthesized as the kind of N
$m : A@w$	m is of type A at the world w

We maintain a phase-distinction between static and dynamic phases in PCML$_5$. This means that runtime terms do not appear in constructors. Accordingly, we divide the context into a static part, Δ, which gives kinds to constructor variables α, and a dynamic part Γ which types term variables x. Because of the phase-distinction, the dynamic context is needed only for term typing judgments. Selected rules defining well-formed kinds are shown here:

Kinds	K	$::=$	$\texttt{Type} \mid \texttt{Wld} \mid \texttt{Prop} \mid \texttt{Prin} \mid \texttt{Prf}(A)$
		$\mid$	$A_1\ \texttt{Affirms}\ A_2 \mid \texttt{Db} \mid \Pi\alpha{::}K_1.K_2$
Constructors			
(Neutral forms)	N	$::=$	$\alpha \mid \mathbf{a} \mid A_1 \to A_2 \mid A_1 \times A_2 \mid A_1 + A_2 \mid \texttt{unit}$
		$\mid$	$\exists\alpha{::}K.A \mid \texttt{Iam}(A) \mid (N\ A)$
(Normal forms)	A, w, k	$::=$	$N \mid \lambda\alpha{::}K.A$
Polytypes	τ	$::=$	$\forall\langle\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n\rangle A$
Terms	m	$::=$	$x \mid \lambda x{:}A.m \mid (m_1\ m_2) \mid \langle m_1, m_2\rangle \mid \pi_1 m \mid \pi_2 m \mid \texttt{inl}\ m \mid \texttt{inr}\ m$
		$\mid$	$\texttt{case}\ m\ \texttt{of}\ x.(m_1 \mid m_2) \mid \langle\rangle$
		$\mid$	$\texttt{let}\ x = m_1\ \texttt{in}\ m_2$
		$\mid$	$\texttt{get}[w]m$
		$\mid$	$\texttt{acquire}[A] \mid \texttt{authenticate}$
		$\mid$	$\mathbf{c}[A_1, \ldots, A_n](m)$
		$\mid$	$\{\alpha = A; m : A'\}$
		$\mid$	$\texttt{open}\ \{\alpha, x\} = m_1\ \texttt{in}\ m_2$
		$\mid$	$\texttt{iam}[\mathbf{a}]$
Constructor signature	Σ_c	$::=$	$\cdot \mid \Sigma_c, \mathbf{a}{:}K$
Term signature	Σ_t	$::=$	$\cdot \mid \Sigma_t, \mathbf{c}{:}\tau@w$
Constructor context	Δ	$::=$	$\cdot \mid \Delta, \alpha{\Rightarrow}K$
Term context	Γ	$::=$	$\cdot \mid \Gamma, x{:}A@w$

Figure 3. PCML$_5$ syntax: Shaded parts are PCML$_5$'s additions to ML$_5$

$$\frac{\Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prop}}{\Delta \vdash_{\Sigma_c} \texttt{Prf}(A)\ \texttt{kind}}(\texttt{Proof})$$

$$\frac{\Delta \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Prin} \qquad \Delta \vdash_{\Sigma_c} A_2 \Leftarrow \texttt{Prop}}{\Delta \vdash_{\Sigma_c} A_1\ \texttt{Affirms}\ A_2\ \texttt{kind}}(\texttt{Affirms})$$

The kind $\texttt{Prf}(A)$ classifies proofs of proposition A, and $A_1\ \texttt{Affirms}\ A_2$ classifies evidences for the judgment $A_1\ \texttt{affirms}\ A_2$ from Section 2.

All types except one in PCML$_5$ are inherited from ML$_5$. PCML$_5$ adds an abstract type $\texttt{Iam}(A)$ of authentication credentials of the principal A. We shall return to this type in Section 5.4 when we discuss authentication.

$$\frac{\Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prin}}{\Delta \vdash_{\Sigma_c} \texttt{Iam}(A) \Rightarrow \texttt{Type}}(\texttt{auth})$$

5.2 Representing authorization logic

The kind level of PCML$_5$ is sufficiently rich to encode the authorization logic from Section 2 using higher-order abstract syntax. For instance, the proposition $P_1 \supset P_2$ can be encoded as the constructor $\texttt{imp}\ A_1\ A_2$, where $\texttt{imp} \Rightarrow \texttt{Prop} \to \texttt{Prop} \to \texttt{Prop}$ is a constructor constant. Propositions in the logic become constructors of kind $\texttt{Prop}$ in PCML$_5$. Proofs, i.e., derivations of the judgment $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{true}$ get translated into open constructors of kind $\texttt{Prf}(A)$, where A is translation of P, under the context Δ that represents $\mathbf{\Delta}, \Phi$. Affirmations, i.e., derivations of judgment $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \texttt{affirms}\ P$ are translated into open constructors of kind $A_1\ \texttt{Affirms}\ A_2$ where A_1 and A_2 are representations of a and P respectively.

The central principle behind the translation of logic is that hypothetical reasoning in PCML$_5$ represents the consequence relation of the logic. This enables constructing proofs in a program that are hypothetical in proofs of certain propositions (such as those where a runtime policy query is involved), and later discharging those hypotheses using facts from authorization policies. In order to state an adequacy theorem, we first need some terminology: We use the notation Σ_c^+ to denote the signature consisting of constants in Σ_c and the constants defining the embedding of authorization logic in PCML$_5$. The following judgments show representation of an entity from the logic into a construct in the language (rules are omitted here but can be found in our full report (Avijit et al. 2009)):

$\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} A \Leftarrow \texttt{Prop}$	Propositions
$\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{true} \gg \Delta \vdash_{\Sigma_c^+} A \Leftarrow \texttt{Prf}(A)$	Proofs
$\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \texttt{affirms}\ P \gg \Delta \vdash_{\Sigma_c^+} A \Leftarrow A_1\ \texttt{Affirms}\ A_2$	Affirmations

We lift the translation to contexts and signatures in the straightforward manner, representing a hypothesis $P\ \texttt{true}$ using as $\alpha{\Rightarrow}\texttt{Prf}(A)$ where A is the representation of P, and α is a fresh variable. In the following theorem, we shall assume that Σ_c is a representation of $\mathbf{\Sigma}$ and Δ_1, Δ_2 represent the contexts $\mathbf{\Delta}$ and Φ resp..

Theorem 5.1 (Adequacy). *Let* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1\ \texttt{prop} \gg \Delta_1 \vdash_{\Sigma_c^+} A_1 \Leftarrow \texttt{Prop}$*, and* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_2\ \texttt{prop} \gg \Delta_1 \vdash_{\Sigma_c^+} A_2 \Leftarrow \texttt{Prop}$*. Then* $\mathbf{\Delta}; \Phi, P_1\ \texttt{true} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_2\ \texttt{true}$ *iff there exists* A_3 *such that* $\Delta_1, \Delta_2, \alpha{\Rightarrow}\texttt{Prf}(A_1) \vdash_{\Sigma_c^+} A_3 \Leftarrow \texttt{Prf}(A_2)$.

In the rest of the paper, we freely use logical notation instead of its translation for readability. For instance, we use $A_1 \supset A_2$ to mean $\texttt{imp}\ A_1\ A_2$. The notation $\hat{\Phi}$ denotes the representation of the authorization policy Φ in the language.

5.3 Situated typing and distributed computation

PCML$_5$ inherits the notion of typing relative to worlds from ML$_5$. Here we briefly recap the central ideas from ML$_5$: distribution of computation and mobility of data.

The main motivation behind situated typing in ML$_5$ is to express locality of resources, and use it to restrain where a piece of code can run, with the idea being that a resource can only be accessed by code running at the same location. Distribution of execution is achieved by moving result of a computation from one location to another using a `get`. This is expressed in the following typing rule:

$$\frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w' \qquad \Delta \vdash_{\Sigma_c} A\ \texttt{mobile}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \texttt{get}[w']m : A@w}(\texttt{get})$$

If m is well-typed for w', then it may be executed using a remote call from another world w. ML$_5$ however restricts the remote invocation to terms of *mobile* types only. The first requirement for a type to be mobile is that all values of that type should be typable at all worlds. This ensures that when the result of m is brought back from w' to w, the value makes sense at w. Base types such as `string` are mobile as their values can be typed at any site. Types representing local resources such as reference cells are not mobile. Function types are also not mobile in ML$_5$ because, by design, computations are never shipped across sites. Instead code meant to be run at a site is compiled and stored at that site. Only the locus of execution shifts from one site to another, as in a remote procedure call.

PCML$_5$ adds a new type $\texttt{Iam}(A)$ for typing authentication credentials to the types inherited from ML$_5$. The type $\texttt{Iam}(A)$ is mobile because authentication credentials are typable at all worlds. An existential type $\exists\alpha{::}K.A$ is mobile if the type A of the packed value is mobile, regardless of α. Thus, for instance, the type $\exists\alpha{::}\texttt{Prin}.\texttt{Iam}(\alpha)$ is mobile. All constructors are considered mobile in the sense that their kinding is independent of their location. In particular, principal and resource indices, and proofs are typable at all sites.

5.4 Authentication

For each constant principal $\mathbf{a}$ declared in the constructor signature Σ_c, we use a constant $\texttt{iam}[\mathbf{a}]$ as its authentication token which is typed as $\texttt{Iam}(\mathbf{a})$.

$$\frac{\mathbf{a}{\Rightarrow}\texttt{Prin} \in \Sigma_c}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \texttt{iam}[\mathbf{a}] : \texttt{Iam}(\mathbf{a})@w}(\texttt{Iam-I})$$

PCML$_5$ provides an abstract operation `authenticate` to authenticate the program on behalf of a principal. In an implementation, this could be achieved through well-established protocols for authentication, e.g. Kerberos (Neuman and Ts'o 1994). The operation returns a principal together with an authentication token that serves as a proof of the principal having been authenticated.

The operation `authenticate` is typed as:

$$\frac{}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \texttt{authenticate} : \exists\alpha{::}\texttt{Prin}.\texttt{Iam}(\alpha)\ \texttt{option}@w}$$

The primitive is typed using an `option` type because it may fail to return any meaningful value. Let us consider the case when the primitive is successful. The result is typed as $\exists\alpha{::}\texttt{Prin}.\texttt{Iam}(\alpha)$ in this case. Notice that the authentication credential is typed as $\texttt{Iam}(\alpha)$ linking it to the abstract component α of the abstraction. The runtime term of type $\texttt{Iam}(\alpha)$ and the static proxy α can be viewed as dual representations of an authenticated principal. Because of the abstraction, the static component uniquely represents a particular instance of opening up the package, since it is assumed to be different from everything else. Thus a credential of type $\texttt{Iam}(\alpha)$ represents a particular instance of doing authentication in the program.

5.5 Authorization

An authorization proof is constructed by doing proof search using local security policies at various sites. This is done using a primitive operation $\texttt{acquire}[A]$. The type system tracks the kinds of proofs starting with $\texttt{acquire}[A]$ to API calls where the proofs are used.

5.5.1 Distributed proof acquisition

$\texttt{acquire}[A]$ optionally returns a proof of A. We use an existential type to model this.

$$\frac{\Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prop}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \texttt{acquire}[A] : \exists\alpha{::}\texttt{Prf}(A).\texttt{unit}\ \texttt{option}@w}$$

The success case is typed as $\exists\alpha{::}\texttt{Prf}(A).\texttt{unit}$. When this existential type is eliminated as $\texttt{open}\ \{\alpha, x\} = m_1\ \texttt{in}\ m_2$, an assumption $\alpha{\Rightarrow}\texttt{Prf}(A)$ is introduced in the scope of m_2 during type-checking. We have mentioned before that all kinds are mobile meaning that constructors are typable at all sites. Thus the variable α stands for a globally-valid proof of proposition A in m_2 even though it is discharged using a proof obtained locally at a site.

5.5.2 Proof checking at API call sites

API constants, $\mathbf{c}$, are introduced using the term signature Σ_t. In order to call an API, its polymorphic arguments have to be fully instantiated. The typing rule statically reflects the proof-verification that happens at the reference monitor during such a call. All the constructor arguments are typed statically according to the type of the API constant.

$$\frac{\begin{array}{c}\Sigma_t(\mathbf{c}) = \forall\langle\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n\rangle(A \to A') \\ \Delta \vdash_{\Sigma_c} A_k \Leftarrow [A_1/\alpha_1]\ldots[A_{k-1}/\alpha_{k-1}]K_k\ (k = 1..n) \\ \Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : [A_1/\alpha_1]\ldots[A_n/\alpha_n]A@w\end{array}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathbf{c}[A_1, \ldots, A_n](m) : [A_1/\alpha_1]\ldots[A_n/\alpha_n]A'@w}$$

Notice that for an API call to be well-typed, the constructor arguments should be of the proper kind under the context Δ as per the type specification of the API. At runtime, the hypotheses are discharged using closed constructors. The hypothetical judgment ensures that typing of arguments is preserved upon discharging these hypotheses. This in turn ensures success of runtime verification of these proofs at reference monitors. We elaborate on proof verification at refence monitors when we give the dynamics of API calls in Section 6.6.

An important technical detail here is the dependency among the constructor arguments. Consider the type $\forall\langle\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n\rangle A \to A'$. The kind of an argument potentially depends on all previous arguments. This is manifest in the premiss $\Delta \vdash_{\Sigma_c} A_k \Leftarrow [A_1/\alpha_1]\ldots[A_{k-1}/\alpha_{k-1}]K_k$ that substitutes arguments A_1 through A_{k-1} in K_k while checking the kind of A_j.

6. Runtime semantics

Term evaluation in PCML$_5$ has the following aspects, in addition to distribution:

1. **Principals, databases and APIs :** The transition system is parameterized by a constructor signature Σ_c, and an API signature Σ_t. The signatures remain fixed throughout the evaluation of the program. We assume a fixed set of principal and database indices, introduced through the constructor signature Σ_c. In addition to principals and databases, Σ_c also introduces representation of an authorization logic.
2. **Execution under a security policy:** Each location has a fixed authorization policy associated with it. The policy at site w consists of assertions made by the principal $\bar{w}$ who governs that site. We formulate the local policy at site w as:

   $$\hat{\Phi}_w ::= \alpha_1{\Rightarrow}\texttt{Prf}(\langle\bar{w}\rangle A_1), \ldots, \alpha_n{\Rightarrow}\texttt{Prf}(\langle\bar{w}\rangle A_n)$$

   where $\alpha_1, \ldots, \alpha_n$ are chosen fresh.

   We assume that all local policies are well-formed with respect to the signature Σ_c that defines the authorization logic. As mentioned before, the amalgamated authorization policy $\hat{\Phi}$ is the union of all local policies.

3. **Authorization checks at reference monitors** An API call $\mathbf{c}[A_1, \ldots, A_n](m)$ results in the reference monitor type-checking A_1 through A_n (as shown by Rule (`api-reduce`) later). Some of these A_i's may be proofs constructed by a distributed acquisition of local policy assertions from various sites. As discussed in Section 3, local assertions make sense globally and are implemented in an unforgeable and irrefutable manner by signing the asserted proposition with the principal's signing key. In the formalism, we directly use the global security policy $\hat{\Phi}$ to serve as the context for type-checking the proofs at API call-sites at runtime.
4. **Authentication and active principals:** As evaluation progresses, principals may be authenticated using the `authenticate` primitive. Principals who have been authenticated during a program run are called *active* principals. We keep a record of all active principals as a set of principals $\mathcal{A} \subseteq \{\mathbf{a} \mid \mathbf{a} \Rightarrow \mathtt{Prin} \in \Sigma_c\}$. The program starts execution with $\mathcal{A} = \phi$, i.e. no principal is assumed to be authenticated at the beginning of the program.

6.1 Judgment forms

We describe the dynamic semantics using a transition relation between terms. We use the following judgment forms:

1. $m; \mathcal{A} \ \mapsto_w^{\Sigma_c;\Sigma_t;\hat{\Phi}} \ m'; \mathcal{A}'$: Under the signatures $\Sigma_c; \Sigma_t$ and the amalgamated authorization policy $\hat{\Phi}$, the term m, executed at world w, steps to the term m' in a single transition; the set of active principals changes from $\mathcal{A}$ to $\mathcal{A}'$. Since Σ_c, Σ_t and $\hat{\Phi}$ remain fixed during evaluation, we often omit them while presenting the transition system.
2. $m \ \mathtt{val}_{\mathcal{A}}$: means that the term m is a value under the runtime record $\mathcal{A}$, and is not evaluated further.

6.2 The value judgment

The value judgment $m \ \mathtt{val}_{\mathcal{A}}$ defines values with respect to the runtime state $\mathcal{A}$. The rules for determining values of the types such as function, product, sum and existential types are straightforward; we adopt eager evaluation for products, sums and existential packages.

The crucial part of this definition is the case for values of type $\mathtt{Iam}(A)$. An authentication token $\mathtt{iam}[\mathbf{a}]$ is a value only if $\mathbf{a}$ is an authenticated principal, i.e. if $\mathbf{a}$ is part of the active set $\mathcal{A}$. The need for this assumption is explained in Section 6.6.

$$\frac{\mathbf{a} \in \mathcal{A}}{\mathtt{iam}[\mathbf{a}] \ \mathtt{val}_{\mathcal{A}}} (\mathtt{Iam\text{-}V})$$

6.3 Distribution

A remote call $\mathtt{get}[w']m$ is evaluated at a world w as follows:

$$\frac{m; \mathcal{A} \ \mapsto_{w'} \ m'; \mathcal{A}'}{\mathtt{get}[w']m; \mathcal{A} \ \mapsto_w \ \mathtt{get}[w']m'; \mathcal{A}'} (\mathtt{get\text{-}eval})$$

$$\frac{m \ \mathtt{val}_{\mathcal{A}}}{\mathtt{get}[w']m; \mathcal{A} \ \mapsto_w \ m; \mathcal{A}} (\mathtt{get\text{-}reduce})$$

Rule (`get-eval`) expresses the remote execution of m at the world w'. In case m is a value, it is brought to w from w' (Rule (`get-reduce`)). This transfer is safe, i.e., m is well-typed at w because the type system guarantees that m has a mobile type.

6.4 Local policy acquisition

The $\mathtt{acquire}[A]$ primitive does a proof search for the proposition A using the local policy $\hat{\Phi}_w$ at the site where it is executed. This proof search may fail. We do not stipulate the procedure used for local theorem-proving. For this reason, we model this primitive using a non-deterministic step: the Rule (`acq-succ`) non-deterministically chooses a proof A' such that A' has the kind $\mathtt{Prf}(A)$ under the assumptions $\hat{\Phi}_w$.

$$\frac{\hat{\Phi}_w \vdash_{\Sigma_c} A' \Leftarrow \mathtt{Prf}(A)}{\mathtt{acquire}[A]; \mathcal{A} \ \mapsto_w \ \mathtt{SOME}\, \{\alpha = A'; \langle\rangle : \mathtt{unit}\}; \mathcal{A}} (\mathtt{acq\text{-}succ})$$

The failure case is modeled by a transition to the value `NONE`:

$$\frac{}{\mathtt{acquire}[A]; \mathcal{A} \ \mapsto_w \ \mathtt{NONE}; \mathcal{A}} (\mathtt{acq\text{-}fail})$$

6.5 Authentication

Authentication, if successful, results in the production of a token for the authenticated principal. We use a non-deterministic rule which may result in a token for any principal that has been declared in the signature Σ_c.

$$\frac{\mathbf{a}{::}\mathtt{Prin} \in \Sigma_c}{\mathtt{authenticate}; \mathcal{A} \ \mapsto_w \ \mathtt{SOME}\, \{\alpha = \mathbf{a}; \mathtt{iam}[\mathbf{a}] : \mathtt{Iam}(\alpha)\}; \mathcal{A} \cup \{\mathbf{a}\}} (\mathtt{auth\text{-}succ})$$

The record $\mathcal{A}$ is augmented with $\mathbf{a}$ to note this authentication.

In case of failure, `authenticate` returns `NONE`, and $\mathcal{A}$ is left unchanged:

$$\frac{}{\mathtt{authenticate}; \mathcal{A} \ \mapsto_w \ \mathtt{NONE}; \mathcal{A}} (\mathtt{auth\text{-}fail})$$

An important technical detail to note here is that we require Σ_c to contain all possible principals. This is manifest in the Rule (`auth-succ`) where `authenticate` does not generate a new principal identity but simply picks one up from Σ_c.

6.6 API calls to reference monitors

API constants represent externally implemented functions. We model their behavior using a non-deterministic transition to an arbitrary term of the appropriate type.

$$\frac{m; \mathcal{A} \ \mapsto_w \ m'; \mathcal{A}'}{\mathbf{c}[A_1, \ldots, A_n](m); \mathcal{A} \ \mapsto_w \ \mathbf{c}[A_1, \ldots, A_n](m'); \mathcal{A}'} (\mathtt{api\text{-}eval})$$

$$\frac{\begin{array}{c}\Sigma_t(\mathbf{c}) = \forall\langle \alpha_1{::}K_1, \ldots, \alpha_n{::}K_n\rangle A \to A'@w \\ \forall i \in [0..n-1] \ \hat{\Phi} \vdash_{\Sigma_c} A_{i+1} \Leftarrow [A_1/\alpha_1] \ldots [A_i/\alpha_i] K_{i+1} \\ m_1 \ \mathtt{val}_{\mathcal{A}}\end{array}}{\mathbf{c}[A_1, \ldots, A_n](m_1); \mathcal{A} \ \mapsto_w^{\Sigma_c;\Sigma_t;\hat{\Phi}} \ m_2; \mathcal{A}} (\mathtt{api\text{-}reduce})$$

The Rule (`api-reduce`) illustrates the central actions that take place during a call to a reference monitor:

1. **Authorization checks** The central point of PCA is that reference monitors for APIs check proofs. Proofs are passed as constructor arguments to the API. In order to verify them, the reference monitor needs the amalgamated policy $\hat{\Phi}$.

   Consider the Rule (`api-reduce`). Before evaluating the API call, the reference monitor type-checks all the constructor arguments $A_1 \ldots A_n$ under $\hat{\Phi}$. For well-typed programs, the type system (cf. Section 5.5.2) guarantees that all type checks succeed.
2. **Authentication checks** In addition to verifying proofs, the reference monitor checks authentication credentials. APIs for such monitors are polymorphic in the authenticated principal α, and require an extra parameter of type $\mathtt{Iam}(\alpha)$. In order to verify an authentication credential $\mathtt{iam}[\mathbf{a}]$, the reference monitor simply checks whether $\mathbf{a} \in \mathcal{A}$, where $\mathcal{A}$ is the active set at the time of API call. In the formalism, authentication checks are reflected implicitly in the value judgment. An authentication token $\mathtt{iam}[\mathbf{a}]$ is considered a value under an active set $\mathcal{A}$ only if

$\mathbf{a} \in \mathcal{A}$ (Rule `Iam-V`). This way, we reduce authentication checking to checking that the argument to the API call (m_1 in Rule (`api-reduce`)) is a value.

6.7 PCA runtime errors

A term may incur a runtime fault at a reference monitor in the following two ways:

1. Either the proofs passed to the monitor API do not type-check,
2. An authentication credential is not valid, in the sense that the purported principal does not appear in the authentication record $\mathcal{A}$.

We formalize these errors explicitly using the judgment $m \uparrow_{\mathcal{A}}$. This judgment formalizes the intuition that m is not a value, and $m; \mathcal{A} \not\mapsto_w$.

$$\frac{\mathbf{a} \notin \mathcal{A}}{\mathtt{iam}[\mathbf{a}] \uparrow_{\mathcal{A}}}(\mathtt{iam}^{\uparrow})$$

$$\frac{\hat{\Phi} \nvdash_{\Sigma_c} A_i \Leftarrow [A_1/\alpha_1] \dots [A_{i-1}/\alpha_{i-1}]K_i}{\mathbf{c}[A_1, \dots, A_n](m) \uparrow_{\mathcal{A}}}(\mathbf{c}^{\uparrow})$$

In addition, we have rules to propagate errors through evaluation of other terms, of which we present only a sample here:

$$\frac{m_1 \uparrow_{\mathcal{A}}}{\langle m_1, m_2 \rangle \uparrow_{\mathcal{A}}}(\mathtt{pair}_1^{\uparrow}) \qquad \frac{m_1\ \mathtt{val}_{\mathcal{A}} \qquad m_2 \uparrow_{\mathcal{A}}}{\langle m_1, m_2 \rangle \uparrow_{\mathcal{A}}}(\mathtt{pair}_2^{\uparrow})$$

The rules for the error judgment follow the evaluation order. A pair $\langle m_1, m_2 \rangle$ can incur an error at two instances: either when m_1 which is evaluated first incurs an error (as in Rule ($\mathtt{pair}_1^{\uparrow}$)), or when m_1 has been evaluated to a value and m_2 incurs an error (as shown in Rule ($\mathtt{pair}_2^{\uparrow}$)).

7. Metatheory

We now prove the central result of this paper: a class of well-typed PCML$_5$ programs do not incur runtime faults at reference monitors. We have already summarized a notion of *error* states (Section 6.7) that formalizes exactly the runtime failures we are trying to avoid.

We start by proving progress and preservation for PCML$_5$. This means that well-typed terms do not get stuck. However, static typing alone does not rule out authentication errors, i.e. failures that happen when the reference monitor gets a token $\mathtt{iam}[\mathbf{a}]$ and $\mathbf{a}$ is not an active principal. This happens because the type system does not track runtime authentication events, i.e. calls to `authenticate`.

Throughout this section we shall assume that $\Sigma_c; \Sigma_t$ are well-formed signatures and $\hat{\Phi}$ is an amalgamated security policy which is well-formed under Σ_c.

7.1 Type safety

Theorem 7.1 (Progress). *Let $\mathcal{A}$ be a set of active principals from Σ_c. If $\Delta; \cdot \vdash_{\Sigma_c;\Sigma_t} m : A@w$, then*

1. *either m $\mathtt{val}_{\mathcal{A}}$,*
2. *or $\exists m', \mathcal{A}'. m; \mathcal{A} \mapsto_w^{\Sigma_c;\Sigma_t;\hat{\Phi}} m'; \mathcal{A}'$,*
3. *or $m \uparrow_{\mathcal{A}}$ without using the Rule ($\mathbf{c}^{\uparrow}$).*

Theorem 7.2 (Preservation of typing). *Assume that all API constants declared in Σ_t preserve the typing. If $m; \mathcal{A} \mapsto_w^{\Sigma_c;\Sigma_t;\hat{\Phi}} m'; \mathcal{A}'$ and $\Delta; \cdot \vdash_{\Sigma_c;\Sigma_t} m : A@w$, then $\Delta, \hat{\Phi}; \cdot \vdash_{\Sigma_c;\Sigma_t} m' : A@w$.*

In order to guarantee that terms do not incur authentication errors in addition to not getting stuck, we define a notion of *authentication safety* for terms with respect to a set of active principals. A well-typed term is regarded as authentication safe if it never evaluates to an unauthenticated token. We prove that an authentication safe term does not evaluate to an error state. In addition, we prove that authentication safety is preserved by evaluation; these two theorems together ensure that authentication safe terms do not incur runtime access violations at reference monitors. Finally we show how authentication safety for the case where the set of active principals is empty can be enforced using the type system.

7.2 Authentication history

The runtime semantic ensures that `authenticate` is the only way in which a new authentication token can be generated. We further wish to enforce that a token $\mathtt{iam}[\mathbf{a}]$ appears in a term only when the associated history $\mathcal{A}$ mentions $\mathbf{a}$ as one of the authenticated principals. We use a judgment of the form $\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$ to denote that, in addition to the term being well-typed, all authentication tokens in m have the corresponding principal recorded in $\mathcal{A}$. This judgment resembles the static typing judgment $\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w$ except for the case of $\mathtt{iam}[\mathbf{a}]$, where we demand that the principal $\mathbf{a}$ be present in $\mathcal{A}$. We present a sample of rules here:

$$\frac{\mathbf{a}{::}\mathtt{Prin} \in \Sigma_c \qquad \mathbf{a} \in \mathcal{A}}{\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{iam}[\mathbf{a}] : \mathtt{Iam}(\mathbf{a})@w}$$

$$\frac{\Delta; \Gamma, x{:}A@w \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A'@w}{\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \lambda x{:}A.m : A \to A'@w}$$

$$\frac{\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_1 \to A_2@w \qquad \Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_1@w}{\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} (m_1\ m_2) : A_2@w}$$

Theorem 7.3. *If $\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$ then $\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w$.*

We assume that API calls do not directly introduce spurious authentication tokens in their results. The following assumption about API calls summarizes this:

Definition 7.4 (Authentication safety for APIs). *Let $\Sigma_t(\mathbf{c}) = \forall \langle \alpha_1{::}K_1, \dots, \alpha_n{::}K_n \rangle A_1 \to A_2@w$. Let $\mathcal{P}$ be the set of constants $\mathbf{a}$ s.t. $\mathbf{a}{::}\mathtt{Prin} \in \Sigma_c$. The API $\mathbf{c}$ is* authentication safe *if the following holds:*

For all constructors $B_1, \dots, B_n$ such that $\cdot \vdash_{\Sigma_c} B_{i+1} \Leftarrow [B_1/\alpha_1] \dots [B_i/\alpha_i]K_{i+1}$. Let v be well-typed, and be safe for $\mathcal{A}$ as $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} v : [B_1/\alpha_1] \dots [B_n/\alpha_n]A_1@w$. If $\mathbf{c}[B_1, \dots, B_n](v); \mathcal{A} \mapsto_w m; \mathcal{A}$, then m is safe for $\mathcal{A}$, i.e. $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : [B_1/\alpha_1] \dots [B_n/\alpha_n]A_2@w$.

This restriction forces the resulting term to be safe w.r.t. all the possible $\mathcal{A}$'s for which the argument v is safe, thus ensuring that all authentication tokens in the result of the call are inherited from the argument v, i.e. no new $\mathtt{iam}[A]$ are introduced in the result.

If all APIs are authentication safe, then authentication safety is preserved by evaluation:

Theorem 7.5 (Preservation of authentication safety). *Let all API constants declared in Σ_t be safe in the sense of Def. 7.4. If $m; \mathcal{A} \mapsto_w^{\Sigma_c;\Sigma_t;\hat{\Phi}} m'; \mathcal{A}'$ and $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$, then $\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m' : A@w$.*

Furthermore, authentication safety ensures that reference monitors cannot reject calls from well-typed and authentication-safe programs.

Theorem 7.6 (Progress under authentication safety). *Let $\mathcal{A}$ be a set of active principals from Σ_c. If $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$, then either m $\mathtt{val}_{\mathcal{A}}$, or $\exists m', \mathcal{A}'. m; \mathcal{A} \mapsto_w^{\Sigma_c;\Sigma_t;\hat{\Phi}} m'; \mathcal{A}'$.*

Notice that in contrast to Theorem 7.1 which has a failure clause ($m \uparrow_{\mathcal{A}}$), Theorem 7.6 excludes the possibility of an authentication-safe term getting stuck and ensures that it is either a value, or that it takes a step to another term. The progress theorem (Theorem 7.6), together with preservation (Theorem 7.5) ensures that an authentication safe, well-typed term never incurs a runtime fault at reference monitors.

7.3 Initializing evaluation

A closed well-typed program m begins evaluation under an empty set of authenticated principals. By virtue of the progress and preservation theorems for authentication safety, in order to ensure that evaluation never incurs a runtime error, it is sufficient to ensure that $\cdot;\cdot \vdash^{\{\}}_{\Sigma_c;\Sigma_t} m : A@w$. This can be enforced easily using the typing rules by simply disallowing all constants $\mathtt{iam}[\mathbf{c}]$ as well-typed terms. This is possible because while checking for authentication safety of a term under $\mathcal{A}$, all the sub-terms are checked under the same $\mathcal{A}$.

8. Example

We revisit the example scenario from Figure 1. The program shown in Figure 4 illustrates how a distributed program running at `browser` can access the database `conf` at `acmserver` using a proof constructed using distributed proof-search at `univserver` and `acmserver`.

8.1 External API and other declarations

The program in Figure 4 first declares principal, database and world constants as `extern` declarations. The `extern principal` declaration runs an initialization code to acquire the runtime representation of the principal (using the name supplied) and binds it to the variable in the declaration. The classifiers `bytecode` and `javascript` in world declarations determine the language of the generated code for the respective worlds. Lines 10-12 declare the addresses for the three sites.

Lines 14-16 declare predicates using the `extern prop` declaration. A declaration `extern prop` ($\mathtt{s}_1$, ..., $\mathtt{s}_n$) `p` declares `p` to be a predicate with $\mathtt{s}_1$ through $\mathtt{s}_n$ being the sorts of its arguments.

Next we declare types for the database API at `acmserver`. The constructor `dbhandle` has the kind $\mathtt{Db} \rightarrow \mathtt{Type}$. The API `db.open` takes a database name of type `string` and returns the runtime identity of the database packaged with its static proxy, as discussed in Section 4.3. We use the syntax `{a:K, A}` for the existential type $\exists a{::}K.A$. For simplicity, databases in this paper are simply *(key, value)* pairs, where both the *key* and the *value* are strings. Line 21 introduces the proof-carrying API for reading databases: `db.read` is typed polymorphically in the accessing principal p, the database d, and the proof f of `acm` affirming that p is allowed to read d. The argument to `db.read` is a triple consisting of (1) a value of type $\mathtt{Iam}(p)$ which forms an evidence of p's authentication, (2) the runtime structures associated with the database d, and (3) the key to be read. Both the database operations `db.open`, and `db.read` are local to `acmserver`.

8.2 Distributed proof construction

We now discuss the steps involved in reading `conf` at `acmserver`. This is done in Lines 28-57 by the function `readpaper` which is typed at `acmserver`.

The first operation (Lines 29-31) is to authenticate the principal. In case authentication fails, the program raises the `Abort` exception and halts. In case of a successful authentication, `authenticate` returns a value of type $\exists\alpha{::}\mathtt{Prin}.\mathtt{Iam}(\alpha)$, which is bound to `authpkg`.

Let us consider the case when authentication succeeds. In this case `authpkg` gets bound to a package $\{\alpha = k; \mathtt{iam}[k] : \mathtt{Iam}(\alpha)\}$, where k is the authenticated principal. The side-effect of this successful authentication is that k gets added to the set of active principals.

Assume that the amalgamated policy is given as the following context $\hat{\Phi}$, with the variables $\alpha_1, \alpha_2, \alpha_3$ chosen fresh. The whole program is run under this context. In an implementation, these variables would be bound to digital certificates.

$$\hat{\Phi} = \left\{ \begin{array}{lll} \alpha_1 & \Rightarrow & \langle\mathtt{univ}\rangle\mathtt{is\_student}(\mathtt{alice}, \mathtt{univ}), \\ \alpha_2 & \Rightarrow & \langle\mathtt{acm}\rangle\mathtt{is\_member}(\mathtt{univ}, \mathtt{acm}), \\ \alpha_3 & \Rightarrow & \langle\mathtt{acm}\rangle\forall x{:}\mathtt{prin}.\forall y{:}\mathtt{prin}. \\ & & \quad \mathtt{is\_member}(x, \mathtt{acm}) \wedge \langle x\rangle\mathtt{is\_student}(y, x) \\ & & \quad \supset \mathtt{mayrd}(\mathtt{conf}, y) \\ & \vdots & \end{array} \right\}$$

The task now is to construct a proof of

$$\langle\mathtt{acm}\rangle\mathtt{mayrd}(\mathtt{conf}, k).$$

This task depends on the identity of the authenticated principal. The static index of the authenticated principal is obtained by opening the package `authpkg` (Line 33), binding `me` to k, and `cookie` to $\mathtt{iam}[k]$. The rest of the code in the function executes in the scope of this `open`.

We begin by first acquiring the certificate for studentship for k. This is done by moving evaluation to the site `univserver` and searching for a proof of $\langle\mathtt{univ}\rangle\mathtt{is\_student}(k, \mathtt{univ})$ there. The query `acquire` on Line 36 succeeds if k is `alice` because in this case there is a direct proof, α_1, in the policy. In the absence of any other proofs in `univserver`'s local policy, this query fails for other principals k. A successful `acquire` on Line 36 binds `studentcert` to the optional package `SOME` $\{\alpha = \alpha_1; \langle\rangle : \mathtt{unit}\}$, and the result is brought back to `acmserver` by the `get`. Notice that the result of an `acquire` is of a mobile type and therefore can be brought over from `univserver` to `acmserver`. The program raises the exception `abort` and terminates in case this `acquire` fails.

Next we need to acquire a proof that k is allowed access to the database `conf`. This proof search depends on the static identity of the database `conf`. This identity is obtained using `db.open` at Line 40, which, if successful binds the proxy to `d`, and the runtime handle to `h` upon opening the package returned by `db.open`. We proceed (Line 44) with a query for the following proposition at `acmserver`:

```
univ says is_student(me, univ)
implies
acm says mayrd(d, me)
```

This particular proof search illustrated the idea of combining proofs from various sites. The proof search is done at `acmserver` even though the antecedents of the implication represent non-local facts. The hypothesis is discharged (Line 50) using proofs α_1 acquired separately at `univserver`. This discharge of assumptions using non-local facts to obtain a globally valid proof is possible because constructors are typeable at all locations.

Notice that in order to construct the accessibility proof, it is critical to assemble `univ`'s policy from `univserver`. The policy of `acm` alone does not suffice for the required proof. That is, one cannot hope for the direct query

```
acquire[acm says mayrd (d, me)]
```

to be successful at `acmserver`. The proof query in Lines 44-46 can be viewed as temporarily extending the policy at `acmserver` to include the non-local fact $\langle\mathtt{univ}\rangle\mathtt{is\_student}(k, \mathtt{univ})$, whereupon the accessibility proof becomes derivable at `acmserver`.

```
unit
import "std.mlh"

extern principal acm = "acm"
extern principal univ = "univ"
extern bytecode world acmserver
extern bytecode world univserver
extern javascript world browser

extern val acmaddr ~ acmserver addr
extern val univaddr ~ univserver addr
extern val browseraddr ~ browser addr

extern prop (prin, prin) is_member
extern prop (db, prin) mayrd
extern prop (prin, prin) is_student

extern type (d:db) dbhandle
extern val db.open :
   string -> {d:db, dbhandle(d)} option @ acmserver
extern val (p: prin, d: db, f: says acm mayrd(d, p))
           db.read :
           Iam(p) * dbhandle(d) * string -> string
        @ acmserver

exception Abort of string

fun readpaper () =
  let val authpkg = case authenticate of
                     NONE => raise Abort "authentication failed"
                   | SOME p => p
  in
  open authpkg as {me: prin, cookie} in
    let val studcert =
     case from univaddr
          get acquire [univ says is_student(me, univ)] of
       NONE => raise Abort "studentship search failed"
     | SOME c => c
    in
      case db.open "conf" of
          NONE => raise Abort "db.open failed"
        | SOME dbase =>
            open dbase as {d:db, h} in
              case acquire [univ says is_student(me, univ)
                            implies
                            acm says mayrd(d, me)] of
                  NONE => raise Abort "mayrd search failed"
                | SOME prf => open {studpf, _} = studcert in
                             open {pf, _} = prf in
                                db.read[me, d, impE pf studpf]
                                       (cookie, h, "paper.pdf")
                             end
                             end
            end
    end
  end
 end

do from acmaddr get readpaper ()

end
```

Figure 4. PCML$_5$ code for accessing a PCA-enabled resource under a distributed authorization policy. The code corresponds to the setup introduced in Figure 1.

8.3 Database access using required proofs

Lines 48-53 show the code that makes an API call to read `conf` using the proof constructed above. First the proof packages returned by `acquire`'s are opened binding the proof constructors to `studpf` and `pf`. The accessibility proof is constructed as `impE pf studpf`. `impE` is a constructor from the embedding of the authorization logic representing $\supset$-elimination; it has the kind

$$\Pi\alpha{::}\mathtt{Prop}.\Pi\beta{::}\mathtt{Prop}.\mathtt{Prf}(\alpha \supset \beta) \to \mathtt{Prf}(\alpha) \to \mathtt{Prf}(\beta)$$

The constructor `pf` is a proof of

$$\langle\mathtt{univ}\rangle\mathtt{is\_student}(k, \mathtt{univ}) \supset \langle\mathtt{acm}\rangle\mathtt{mayrd}(\mathtt{conf}, k)$$

under $\hat{\Phi}_{\mathtt{acmserver}}$, and `studpf` is a proof of $\langle\mathtt{univ}\rangle\mathtt{is\_student}(k, \mathtt{univ})$ under $\hat{\Phi}_{\mathtt{univserver}}$. Both proofs are therefore well-typed under $\hat{\Phi}$. Thus the proof `impE pf studpf` proves $\langle\mathtt{acm}\rangle\mathtt{mayrd}(\mathtt{conf}, k)$ under the amalgamated policy $\hat{\Phi}$.

Apart from the authorization check, the monitor checks if the credential `cookie` represents an authenticated principal by consulting the set of active principals. This check also succeeds because `authenticate` done at Line 29 augments the active set with k. Since both checks succeed, the API call is successful. The result, which is of the mobile type `string`, is returned back to `browser` on Line 59.

9. Related work

Aura (Jia et al. 2008; Vaughan et al. 2008) is a language for enforcing authorization policies. It is based on DCC (Abadi et al. 1999). Our language differs from Aura in terms of the domain of use because ours is a distributed programming language for distributed authorization policies. Aura is neither a distributed programming language, nor does it handle distributed policies.

As a matter of technique, PCML$_5$ differs from Aura in the way they incorporate the authorization logic. While PCML$_5$ uses a higher-order encoding of an authorization logic as static constructors, with a phase distinction between static and dynamic phases, the proof level in Aura is a Curry-Howard interpretation of an authorization logic, and is based on DCC (Abadi 2006b).

Also, Aura does not have a notion of authentication of the principal executing the program. A special principal identifier called `self` is used to refer to the executing principal. In contrast, PCML$_5$ uses authentication tokens as indicators of the fact that a certain principal had been authenticated. This also allows for a program to acquire multiple authenticities during its evaluation.

RCF (Bengtson et al. 2008) uses refinement types together with dependent types to express pre- and post-conditions. The proof obligations are represented as preconditions in the API. Thus a function for reading databases may be typed as

$$\mathtt{read} : \mathtt{file}{:}\mathtt{string}\{\mathtt{mayrd}(\mathtt{file})\} \to \mathtt{string}$$

where $\mathtt{file}{:}\mathtt{string}\{\mathtt{mayrd}(\mathtt{file})\}$ is the refinement type of strings f for which $\mathtt{mayrd}(f)$ holds. Refinement types are introduced using a term of the form `assume` C, which is typed as $\_{:}\mathtt{unit}\{C\}$. The typing context can be thought of as defining a theory which is the set of all the formulae appearing in it. A prominent difference between PCML$_5$ and RCF is that PCML$_5$ uses explicit proof terms unlike RCF. In absence of proof terms, the type-checking algorithm of RCF uses an SMT solver to verify if the typing context proves a particular logic formula. Another difference is that RCF does not have a phase distinction since runtime values can appear inside types, because formulas need to mention runtime entities.

PCAL (Chaudhuri and Garg 2009) is another language that relies on external SMT solvers during compile-time to construct PCA proofs. Users annotate program points with propositions that they expect to hold there. The compiler first checks that the annotation at a point is sufficient to guarantee access to the command executed at that point. Then it attempts to construct a proof statically as per the annotation. In case it cannot construct a proof statically, the compiler produces code to dynamically construct the required proof. Using a combination of both methods, the compiler ensures compliance with the PCA interface.

Fable(Swamy et al. 2008) is another language that provides statically enforced compliance with security policies. The idea is to have an abstract type of tagged program values that can only be manipulated using trusted policy functions. A program is divided into two fragments: the policy fragment that provides the abstraction, and the application fragment that functions as a client for the abstraction, treating tagged values abstractly. Different kinds of security properties can be expressed by having different interpretations for the tags. Thus instead of designing a language around a particular form of policies, such as is PCML$_5$, Fable attempts to provide a general framework in which different kinds of policies can be expressed. This however comes at a cost: the language itself does not guarantee any security property itself (other than type safety); the relevant properties need to be proved separately for every policy fragment.

The idea of statically checking the permission for accessibility has been used in a completely different setting as compared to ours by Krishnaswami *et al.* (Krishnaswami and Aldrich 2005). They use the notion of *domains* with inter-domain accessibility permissions to statically enforce that code from a domain may access an element of another domain only if there is a chain of access permissions from the former to the latter domain. Analogous to the proof-carrying APIs in PCML$_5$, their proposal allows specification of access permissions associated with a domain. They use domains to encapsulate stateful parts of modules for which it is desirable to restrict access from outside domains. The type system enforces compliance of the module and its interface to a high-level policy of accessibility, i.e., protected parts of the module are not leaked out by the interface.

10. Conclusion and future work

We have presented a language-based approach for enforcing distributed authorization policies. We are currently working on a prototype implementation of PCML$_5$, building upon the implementation of ML$_5$ (Murphy 2008). We plan to implement distributed applications using PCML$_5$. We also plan to mechanize the metatheory of PCML$_5$. In this paper, we have assumed a very simple model of spatial distribution of policies: the local policy at a site w contains only assertions made by the principal who governs w. In practice, however, assertions made by one principal may be cached at other sites, as is done in trust management systems (Blaze et al. 1996). In future work, we plan to support such richer forms of distribution of authorization policies.

References

Martín Abadi. Logic in access control. In *LICS*, pages 228–233, 2003.

Martín Abadi. Access control in a core calculus of dependency. In *ICFP*, pages 263–273, 2006a.

Martín Abadi. Access control in a core calculus of dependency. *SIGPLAN Notices*, 41(9):263–273, 2006b.

Martín Abadi, Michael Burrows, Butler W. Lampson, and Gordon D. Plotkin. A calculus for access control in distributed systems. *ACM Transactions Program. Lang. Syst.*, 15(4):706–734, 1993.

Martin Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *Proceedings 26th ACM Symp. on Principles of Programming Languages (POPL)*, pages 147–160, 1999.

Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*, pages 52–62, 1999.

Kumar Avijit, Anupam Datta, and Robert Harper. Distributed programming with distributed authorization, 2009. Available at http://www.cs.cmu.edu/˜kavijit/papers/pcml5-full.pdf.

Lujo Bauer, Michael A. Schneider, and Edward W. Felten. A proof-carrying authorization system. Technical Report TR-638-01, Princeton University, April 2001. URL `http://www.ece.cmu.edu/~lbauer/papers/pcaprototr.pdf`.

Lujo Bauer, Scott Garriss, Jonathan M. Mccune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the grey system. In *Proceedings of the 8th Information Security Conference (ISC05*, pages 431–445, 2005.

Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. Refinement types for secure implementations. In *Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium*, pages 17–32, 2008.

M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 164–173, May 1996.

Luca Cardelli. Phase distinctions in type theory, January 1988. Unpublished manuscript.

Avik Chaudhuri and Deepak Garg. PCAL: Language support for proof-carrying authorization systems. In *Proceedings of the European Symposium on Research in Computer Security*, pages 184–199, 2009.

Dwaine E. Clarke, Jean-Emile Elien, Carl M. Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

Deepak Garg. *Proof Theory for Authorization Logic and its Application to a Practical File System*. PhD thesis, School of Computer Science, Carnegie Mellon University, 2009. Available as Technical Report CMU-CS-09-123.

Deepak Garg and Frank Pfenning. Non-interference in constructive authorization logic. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW 19)*, pages 283–296, 2006.

Robert Harper and Daniel R. Licata. Mechanizing metatheory in a logical framework. *Journal of Functional Programming*, 17(4–5):613–673, July 2007.

Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

Limin Jia, Jeffrey A. Vaughan, Karl Mazurak, Jianzhou Zhao, Luke Zarko, Joseph Schorr, and Steve Zdancewic. Aura: a programming language for authorization and audit. In *ICFP '08: Proceeding of the 13th ACM SIGPLAN International Conference on Functional programming*, pages 27–38, 2008.

Neel Krishnaswami and Jonathan Aldrich. Permission-based ownership: encapsulating state in higher-order typed languages. *SIGPLAN Notices*, 40(6):96–106, 2005.

Butler W. Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions Comput. Syst.*, 10(4):265–310, 1992.

Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

Tom Murphy, VII. *Modal Types for Mobile Code*. PhD thesis, Carnegie Mellon, January 2008. Available as technical report CMU-CS-08-126.

Tom Murphy, VII, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda calculus for distributed computing. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science (LICS 2004)*, pages 286–295,, 2004.

B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33–38, 1994.

Nikhil Swamy, Brian J. Corcoran, and Michael Hicks. Fable: A language for enforcing user-defined security policies. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, May 2008.

Jeffrey A. Vaughan, Limin Jia, Karl Mazurak, and Steve Zdancewic. Evidence-based audit. In *CSF '08: Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium*, pages 177–191, 2008.

Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Carnegie Mellon University, School of Computer Science, 2002.

Edward Wobber, Martn Abadi, Michael Burrows, and Butler Lampson. Authentication in the taos operating system. *ACM Transactions on Computer Systems*, 12:256–269, 1994.

A. PCML$_5$ static semantics

$$\boxed{\Sigma_c\ \texttt{ok}}$$

$$\frac{}{\cdot\ \texttt{ok}} \qquad \frac{\Sigma_c\ \texttt{ok} \qquad \cdot \vdash_{\Sigma_c} K\ \texttt{kind} \qquad \mathbf{a} \notin \texttt{dom}(\Sigma_c)}{\Sigma_c, \mathbf{a}{::}K\ \texttt{ok}}$$

$$\boxed{\Sigma_t\ \texttt{ok}[\Sigma_c]}$$

$$\frac{}{\cdot\ \texttt{ok}[\Sigma_c]} \qquad \frac{\Sigma_t\ \texttt{ok}[\Sigma_c] \qquad \alpha_1{\Rightarrow}K_1, \ldots, \alpha_{j-1}{\Rightarrow}K_{j-1} \vdash_{\Sigma_c} K_j\ \texttt{kind}(j \in [1..n]) \qquad \alpha_1{\Rightarrow}K_1, \ldots, \alpha_n{\Rightarrow}K_n \vdash_{\Sigma_c} A \Leftarrow \texttt{Type}}{\Sigma_t, \mathbf{c}{:}\forall\langle\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n\rangle A\ \texttt{ok}[\Sigma_c]}$$

$$\boxed{\Delta \vdash_{\Sigma_c} K\ \texttt{kind}}$$

$$\frac{}{\Delta \vdash_{\Sigma_c} \texttt{Type}\ \texttt{kind}} \qquad \frac{}{\Delta \vdash_{\Sigma_c} \texttt{Prin}\ \texttt{kind}} \qquad \frac{}{\Delta \vdash_{\Sigma_c} \texttt{Wld}\ \texttt{kind}} \qquad \frac{}{\Delta \vdash_{\Sigma_c} \texttt{Db}\ \texttt{kind}} \qquad \frac{}{\Delta \vdash_{\Sigma_c} \texttt{Prop}\ \texttt{kind}} \qquad \frac{\Delta \vdash_{\Sigma_c} P \Leftarrow \texttt{Prop}}{\Delta \vdash_{\Sigma_c} \texttt{Prf}(P)\ \texttt{kind}}$$

$$\frac{\Delta \vdash_{\Sigma_c} P \Leftarrow \texttt{Prop} \qquad \Delta \vdash_{\Sigma_c} k \Leftarrow \texttt{Prin}}{\Delta \vdash_{\Sigma_c} k\ \texttt{Affirms}\ P\ \texttt{kind}} \qquad \frac{\Delta \vdash_{\Sigma_c} K_1\ \texttt{kind} \qquad \Delta, \alpha{\Rightarrow}K_1 \vdash_{\Sigma_c} K_2\ \texttt{kind}}{\Delta \vdash_{\Sigma_c} \Pi\alpha{::}K_1.K_2\ \texttt{kind}}$$

$$\boxed{\text{(Normal constructors) } \Delta \vdash_{\Sigma_c} A \Leftarrow K}$$

$$\frac{\Delta \vdash_{\Sigma_c} N \Rightarrow K \qquad K \neq \Pi\alpha{::}K_1.K_2}{\Delta \vdash_{\Sigma_c} N \Leftarrow K} \qquad \frac{\Delta, \alpha{\Rightarrow}K \vdash_{\Sigma_c} A \Leftarrow K'}{\Delta \vdash_{\Sigma_c} \lambda\alpha{::}K.A \Leftarrow \Pi\alpha{::}K.K'}$$

$$\boxed{\text{(Neutral Constructors) } \Delta \vdash_{\Sigma_c} N \Rightarrow K}$$

$$\frac{\Delta(\alpha) = K}{\Delta \vdash_{\Sigma_c} \alpha \Rightarrow K} \qquad \frac{\Sigma_c(\mathbf{a}) = K}{\Delta \vdash_{\Sigma_c} \mathbf{a} \Rightarrow K} \qquad \frac{\Delta \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Type} \qquad \Delta \vdash_{\Sigma_c} A_2 \Leftarrow \texttt{Type}}{\Delta \vdash_{\Sigma_c} A_1 \rightarrow A_2 \Rightarrow \texttt{Type}} \qquad \frac{\Delta \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Type} \qquad \Delta \vdash_{\Sigma_c} A_2 \Leftarrow \texttt{Type}}{\Delta \vdash_{\Sigma_c} A_1 \times A_2 \Rightarrow \texttt{Type}}$$

$$\frac{\Delta \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Type} \qquad \Delta \vdash_{\Sigma_c} A_2 \Leftarrow \texttt{Type}}{\Delta \vdash_{\Sigma_c} A_1 + A_2 \Rightarrow \texttt{Type}} \qquad \frac{}{\Delta \vdash_{\Sigma_c} \texttt{unit} \Rightarrow \texttt{Type}} \qquad \frac{\Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Type} \qquad \Delta \vdash_{\Sigma_c} w \Leftarrow \texttt{Wld}}{\Delta \vdash_{\Sigma_c} A\ \texttt{at}\ w \Rightarrow \texttt{Type}}$$

$$\frac{\Delta \vdash_{\Sigma_c} N \Rightarrow \Pi\alpha{::}K_1.K_2 \qquad \Delta \vdash_{\Sigma_c} A \Leftarrow K_1}{\Delta \vdash_{\Sigma_c} (N\ A) \Rightarrow [A/\alpha]K_2} \qquad \frac{\Delta, \alpha{\Rightarrow}K \vdash_{\Sigma_c} A \Leftarrow \texttt{Type}}{\Delta \vdash_{\Sigma_c} \exists\alpha{::}K.A \Rightarrow \texttt{Type}} \qquad \frac{\Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prin}}{\Delta \vdash_{\Sigma_c} \texttt{Iam}(A) \Rightarrow \texttt{Type}}$$

Figure 5. Well-formed signatures and kinds, and constructor kinding

B. Runtime Semantics

The operational semantics is described using judgments of the form $m; \mathcal{A} \ \mapsto_w^{\Sigma_c;\Sigma_t;\Phi}\ m'; \mathcal{A}'$ indexed by the signatures $\Sigma_c; \Sigma_t; \Phi$, and a world w. The signatures Σ_c, Σ_t and Φ remain constant during evaluation, and we shall omit them whenever they are evident from the context.

$\boxed{\text{(Terms) } \Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w}$

$$\frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m_1 : A_1@w \qquad \Delta; \Gamma, x{:}A_1@w \vdash_{\Sigma_c;\Sigma_t} m_2 : A_2@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{let}\, x = m_1 \,\mathtt{in}\, m_2 : A_2@w} \qquad \frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m_1 : A_2 \to A_1@w \qquad \Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m_2 : A_2@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} (m_1\, m_2) : A_1@w}$$

$$\frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A_1 \times A_2@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \pi_1 m : A_1@w} \qquad \frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A_1 \times A_2@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \pi_2 m : A_2@w}$$

$$\frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m_1 : A_1 \,\mathtt{at}\, w'@w \qquad \Delta; \Gamma, x{:}A_1@w' \vdash_{\Sigma_c;\Sigma_t} m_2 : A_2@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{leta}\, x \;=\; m_1 \,\mathtt{in}\, m_2 : A_2@w}$$

$$\frac{\Delta \vdash_{\Sigma_c} w' \Leftarrow \mathtt{Wld} \qquad \Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w' \qquad \Delta \vdash_{\Sigma_c} A \,\mathtt{mobile}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{get}[w']m : A@w} \qquad \frac{\Delta \vdash_{\Sigma_c} A \Leftarrow \mathtt{Prop}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{acquire}[A] : \exists\alpha{::}\mathtt{Prf}(A).\mathtt{unit}\ \mathtt{option}@w}$$

$$\frac{\begin{array}{c}\Sigma_t(\mathbf{c}) = \forall\langle\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n\rangle(A \to A') \\ \Delta \vdash_{\Sigma_c} A_k \Leftarrow [A_1/\alpha_1]\ldots[A_{k-1}/\alpha_{k-1}]K_k \ (k = 1..n) \qquad \Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : [A_1/\alpha_1]\ldots[A_n/\alpha_n]A@w\end{array}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathbf{c}[A_1, \ldots, A_n](m) : [A_1/\alpha_1]\ldots[A_n/\alpha_n]A'@w}$$

$$\frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A_1 + A_2@w \qquad \Delta; \Gamma, x{:}A_1@w \vdash_{\Sigma_c;\Sigma_t} m_1 : A@w \qquad \Delta; \Gamma, x{:}A_2@w \vdash_{\Sigma_c;\Sigma_t} m_2 : A@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{case}\, m \,\mathtt{of}\, x.(m_1 \mid m_2) : A@w}$$

$$\frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m_1 : \exists\alpha{::}K.A@w \qquad \Delta, \alpha{\Rightarrow}K; \Gamma, x{:}A@w \vdash_{\Sigma_c;\Sigma_t} m_2 : A_2@w \qquad \Delta \vdash_{\Sigma_c} A_2 \Leftarrow \mathtt{Type}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{open}\, \{\alpha, x\} = m_1 \,\mathtt{in}\, m_2 : A_2@w}$$

$$\frac{\Delta \vdash_{\Sigma_c} w \Leftarrow \mathtt{Wld}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{authenticate} : \exists\alpha{::}\mathtt{Prin}.\mathtt{Iam}(\alpha)\ \mathtt{option}@w} \qquad \frac{}{\Delta; \Gamma, x{:}A@w, \Gamma' \vdash_{\Sigma_c;\Sigma_t} x : A@w}$$

$$\frac{\Delta; \Gamma, x{:}A_1@w \vdash_{\Sigma_c;\Sigma_t} m : A_2@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \lambda x{:}A_1.m : A_1 \to A_2@w} \qquad \frac{\Delta \vdash_{\Sigma_c} w \Leftarrow \mathtt{Wld}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \langle\rangle : \mathtt{unit}@w} \qquad \frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m_1 : A_1@w \qquad \Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m_2 : A_2@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \langle m_1, m_2\rangle : A_1 \times A_2@w}$$

$$\frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{hold}\, m : A \,\mathtt{at}\, w@w} \qquad \frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w \qquad \Delta \vdash_{\Sigma_c} w' \Leftarrow \mathtt{Wld}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{held}\, m : A \,\mathtt{at}\, w@w'} \qquad \frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A_1@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{inl}\, m : A_1 + A_2@w}$$

$$\frac{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A_2@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{inr}\, m : A_1 + A_2@w} \qquad \frac{\Delta \vdash_{\Sigma_c} A \Leftarrow K \qquad \Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : [A/\alpha]A'@w}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \{\alpha = A; m : A'\} : \exists\alpha{::}K.A'@w}$$

$$\frac{\Delta \vdash_{\Sigma_c} k \Leftarrow \mathtt{Prin} \qquad \Delta \vdash_{\Sigma_c} w \Leftarrow \mathtt{Wld}}{\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} \mathtt{iam}[k] : \mathtt{Iam}(k)@w}$$

Figure 6. Typing rules for terms and values. We use A `option` to mean $A + \mathtt{unit}$.

$$\frac{\Delta \vdash_{\Sigma_c} A_1 \,\mathtt{mobile} \qquad \Delta \vdash_{\Sigma_c} A_2 \,\mathtt{mobile}}{\Delta \vdash_{\Sigma_c} A_1 \times A_2 \,\mathtt{mobile}} \qquad \frac{\Delta \vdash_{\Sigma_c} A_1 \,\mathtt{mobile} \qquad \Delta \vdash_{\Sigma_c} A_2 \,\mathtt{mobile}}{\Delta \vdash_{\Sigma_c} A_1 + A_2 \,\mathtt{mobile}} \qquad \frac{}{\Delta \vdash_{\Sigma_c} \mathtt{unit}\ \mathtt{mobile}}$$

$$\frac{}{\Delta \vdash_{\Sigma_c} A \,\mathtt{at}\, w \,\mathtt{mobile}} \qquad \frac{}{\Delta \vdash_{\Sigma_c} \mathtt{Iam}(A)\ \mathtt{mobile}} \qquad \frac{\Delta, \alpha{:}K \vdash_{\Sigma_c} A \,\mathtt{mobile}}{\Delta \vdash_{\Sigma_c} \exists\alpha{::}K.A \,\mathtt{mobile}}$$

Figure 7. Mobile types

$$\frac{}{\lambda x{:}A.m\ \mathtt{val}_{\mathcal{A}}}(\to\texttt{-V}) \qquad \frac{m_1\ \mathtt{val}_{\mathcal{A}} \qquad m_2\ \mathtt{val}_{\mathcal{A}}}{\langle m_1, m_2\rangle\ \mathtt{val}_{\mathcal{A}}}(\times\texttt{-V}) \qquad \frac{}{\langle\rangle\ \mathtt{val}_{\mathcal{A}}}(\texttt{unit-V}) \qquad \frac{m\ \mathtt{val}_{\mathcal{A}}}{\mathtt{inl}\ m\ \mathtt{val}_{\mathcal{A}}}(+\texttt{-V}_1) \qquad \frac{m\ \mathtt{val}_{\mathcal{A}}}{\mathtt{inr}\ m\ \mathtt{val}_{\mathcal{A}}}(+\texttt{-V}_2)$$

$$\frac{}{\mathtt{held}\ m\ \mathtt{val}_{\mathcal{A}}}(\texttt{at -V}) \qquad \frac{\mathbf{a} \in \mathcal{A}}{\mathtt{iam}[\mathbf{a}]\ \mathtt{val}_{\mathcal{A}}}(\texttt{Iam-V}) \qquad \frac{m\ \mathtt{val}_{\mathcal{A}}}{\{\alpha = A; m : A'\}\ \mathtt{val}_{\mathcal{A}}}(\exists\texttt{-V})$$

$$\frac{m_1; \mathcal{A} \ \mapsto_w\ m_1'; \mathcal{A}'}{\mathtt{let}\ x = m_1\ \mathtt{in}\ m_2; \mathcal{A} \ \mapsto_w\ \mathtt{let}\ x = m_1'\ \mathtt{in}\ m_2; \mathcal{A}'} \qquad \frac{m_1\ \mathtt{val}_{\mathcal{A}}}{\mathtt{let}\ x = m_1\ \mathtt{in}\ m_2; \mathcal{A} \ \mapsto_w\ [m_1/x]m_2; \mathcal{A}}$$

$$\frac{m_1; \mathcal{A} \ \mapsto_w\ m_1'; \mathcal{A}'}{(m_1\ m_2); \mathcal{A} \ \mapsto_w\ (m_1'\ m_2); \mathcal{A}'} \qquad \frac{m_2; \mathcal{A} \ \mapsto_w\ m_2'; \mathcal{A}'}{(\lambda x{:}A.m_1\ m_2); \mathcal{A} \ \mapsto_w\ (\lambda x{:}A.m_1\ m_2'); \mathcal{A}'} \qquad \frac{m_2\ \mathtt{val}_{\mathcal{A}}}{(\lambda x{:}A.m_1\ m_2); \mathcal{A} \ \mapsto_w\ [m_2/x]m_1; \mathcal{A}}$$

$$\frac{m_1; \mathcal{A} \ \mapsto_w\ m_1'; \mathcal{A}'}{\langle m_1, m_2\rangle; \mathcal{A} \ \mapsto_w\ \langle m_1', m_2\rangle; \mathcal{A}'} \qquad \frac{m_1\ \mathtt{val}_{\mathcal{A}} \qquad m_2; \mathcal{A} \ \mapsto_w\ m_2'; \mathcal{A}'}{\langle m_1, m_2\rangle; \mathcal{A} \ \mapsto_w\ \langle m_1, m_2'\rangle; \mathcal{A}'} \qquad \frac{m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'}{\pi_1 m; \mathcal{A} \ \mapsto_w\ \pi_1 m'; \mathcal{A}'}$$

$$\frac{\langle m_1, m_2\rangle\ \mathtt{val}_{\mathcal{A}}}{\pi_1\langle m_1, m_2\rangle; \mathcal{A} \ \mapsto_w\ m_1; \mathcal{A}} \qquad \frac{m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'}{\pi_2 m; \mathcal{A} \ \mapsto_w\ \pi_2 m'; \mathcal{A}'} \qquad \frac{\langle m_1, m_2\rangle\ \mathtt{val}_{\mathcal{A}}}{\pi_2\langle m_1, m_2\rangle; \mathcal{A} \ \mapsto_w\ m_2; \mathcal{A}} \qquad \frac{m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'}{\mathtt{hold}\ m; \mathcal{A} \ \mapsto_w\ \mathtt{hold}\ m'; \mathcal{A}'}$$

$$\frac{m\ \mathtt{val}_{\mathcal{A}}}{\mathtt{hold}\ m; \mathcal{A} \ \mapsto_w\ \mathtt{held}\ m; \mathcal{A}} \qquad \frac{m_1; \mathcal{A} \ \mapsto_w\ m_1'; \mathcal{A}'}{\mathtt{leta}\ x \ = \ m_1\ \mathtt{in}\ m_2; \mathcal{A} \ \mapsto_w\ \mathtt{leta}\ x \ = \ m_1'\ \mathtt{in}\ m_2; \mathcal{A}'}$$

$$\frac{m_1\ \mathtt{val}_{\mathcal{A}}}{\mathtt{leta}\ x \ = \ \mathtt{held}\ m_1\ \mathtt{in}\ m_2; \mathcal{A} \ \mapsto_w\ [m_1/x]m_2; \mathcal{A}} \qquad \frac{m; \mathcal{A} \ \mapsto_{w'}\ m'; \mathcal{A}'}{\mathtt{get}[w']m; \mathcal{A} \ \mapsto_w\ \mathtt{get}[w']m'; \mathcal{A}'} \qquad \frac{m\ \mathtt{val}_{\mathcal{A}}}{\mathtt{get}[w']m; \mathcal{A} \ \mapsto_w\ m; \mathcal{A}}$$

$$\frac{\mathbf{a}{::}\mathtt{Prin} \in \Sigma_c}{\mathtt{authenticate}; \mathcal{A} \ \mapsto_w^{\Sigma_c;\Sigma_t;\Phi}\ \mathtt{SOME}\ \{\alpha = \mathbf{a}; \mathtt{iam}[\mathbf{a}] : \mathtt{Iam}(\alpha)\}; \mathcal{A} \cup \{\mathbf{a}\}} \qquad \frac{}{\mathtt{authenticate}; \mathcal{A} \ \mapsto_w^{\Sigma_c;\Sigma_t;\Phi}\ \mathtt{NONE}; \mathcal{A}}$$

$$\frac{}{\mathtt{acquire}[A]; \mathcal{A} \ \mapsto_w\ \mathtt{NONE}; \mathcal{A}} \qquad \frac{\hat{\Phi}_w \vdash_{\Sigma_c} A' \Leftarrow \mathtt{Prf}(A)}{\mathtt{acquire}[A]; \mathcal{A} \ \mapsto_w^{\Sigma_c;\Sigma_t;\hat{\Phi}}\ \mathtt{SOME}\ \{\alpha = A'; \langle\rangle : \mathtt{unit}\}; \mathcal{A}}$$

$$\frac{m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'}{\mathbf{c}[A_1, \ldots, A_n](m); \mathcal{A} \ \mapsto_w\ \mathbf{c}[A_1, \ldots, A_n](m'); \mathcal{A}'}$$

$$\frac{\Sigma_t(\mathbf{c}) = \forall\langle \alpha_1{::}K_1, \ldots, \alpha_n{::}K_n\rangle A \to A'@w \qquad \forall i \in [0..n-1] \cdot \vdash_{\Sigma_c,\Phi} A_{i+1} \Leftarrow [A_1/\alpha_1]\ldots[A_i/\alpha_i]K_{i+1} \qquad m_1\ \mathtt{val}_{\mathcal{A}}}{\mathbf{c}[A_1, \ldots, A_n](m_1); \mathcal{A} \ \mapsto_w^{\Sigma_c;\Sigma_t;\Phi}\ m_2; \mathcal{A}}$$

$$\frac{m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'}{\mathtt{inl}\ m; \mathcal{A} \ \mapsto_w\ \mathtt{inl}\ m'; \mathcal{A}'} \qquad \frac{m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'}{\mathtt{inr}\ m; \mathcal{A} \ \mapsto_w\ \mathtt{inr}\ m'; \mathcal{A}'} \qquad \frac{m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'}{\mathtt{case}\ m\ \mathtt{of}\ x.(m_1 \mid m_2); \mathcal{A} \ \mapsto_w\ \mathtt{case}\ m'\ \mathtt{of}\ x.(m_1 \mid m_2); \mathcal{A}}$$

$$\frac{\mathtt{inl}\ m\ \mathtt{val}_{\mathcal{A}}}{\mathtt{case}\ \mathtt{inl}\ m\ \mathtt{of}\ x.(m_1 \mid m_2); \mathcal{A} \ \mapsto_w\ [m/x]m_1; \mathcal{A}} \qquad \frac{\mathtt{inr}\ m\ \mathtt{val}_{\mathcal{A}}}{\mathtt{case}\ \mathtt{inr}\ m\ \mathtt{of}\ x.(m_1 \mid m_2); \mathcal{A} \ \mapsto_w\ [m/x]m_2; \mathcal{A}'}$$

$$\frac{m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'}{\{\alpha = A; m : A'\}; \mathcal{A} \ \mapsto_w\ \{\alpha = A; m' : A'\}; \mathcal{A}'} \qquad \frac{m_1; \mathcal{A} \ \mapsto_w\ m_1'; \mathcal{A}'}{\mathtt{open}\ \{m_1, \alpha\} = x\ \mathtt{in}\ m_2; \mathcal{A} \ \mapsto_w\ \mathtt{open}\ \{m_1', \alpha\} = x\ \mathtt{in}\ m_2; \mathcal{A}'}$$

$$\frac{\{\alpha = A; m : A'\}\ \mathtt{val}_{\mathcal{A}}}{\mathtt{open}\ \{\alpha, x\} = \{\alpha = A; m : A'\}\ \mathtt{in}\ m'; \mathcal{A} \ \mapsto_w\ [A/\alpha][m/x]m'; \mathcal{A}}$$

Figure 8. Dynamic semantics

$\boxed{m \uparrow_{\mathcal{A}}^{\Sigma_t;\Sigma_c}}$

$$\frac{\mathbf{a} \notin \mathcal{A}}{\mathtt{iam}[\mathbf{a}] \uparrow_{\mathcal{A}}^{\Sigma_t;\Sigma_c}} \qquad \frac{\Sigma_t(\mathbf{c}) = \forall\langle\alpha_1{::}K_1, \ldots, \alpha_n{::}K_n\rangle A \to A' \qquad \cdot \nvdash_{\Sigma_c} A_j \Leftarrow [A_1/\alpha_1]\ldots[A_{j-1}/\alpha_{j-1}]K_j}{\mathbf{c}[A_1, \ldots, A_n](m) \uparrow_{\mathcal{A}}^{\Sigma_t;\Sigma_c}}$$

$$\frac{m_1 \uparrow_{\mathcal{A}}}{\mathtt{let}\; x = m_1\; \mathtt{in}\; m_2 \uparrow_{\mathcal{A}}} \quad \frac{m_1 \uparrow_{\mathcal{A}}}{(m_1\; m_2) \uparrow_{\mathcal{A}}} \quad \frac{m_2 \uparrow_{\mathcal{A}}}{(\lambda x{:}A.m_1\; m_2) \uparrow_{\mathcal{A}}} \quad \frac{m_1 \uparrow_{\mathcal{A}}}{\langle m_1, m_2\rangle \uparrow_{\mathcal{A}}} \quad \frac{m_1\; \mathtt{val}_{\mathcal{A}} \qquad m_2 \uparrow_{\mathcal{A}}}{\langle m_1, m_2\rangle \uparrow_{\mathcal{A}}} \quad \frac{m \uparrow_{\mathcal{A}}}{\pi_1 m \uparrow_{\mathcal{A}}}$$

$$\frac{m \uparrow_{\mathcal{A}}}{\pi_2 m \uparrow_{\mathcal{A}}} \quad \frac{m \uparrow_{\mathcal{A}}}{\mathtt{hold}\; m \uparrow_{\mathcal{A}}} \quad \frac{m_1 \uparrow_{\mathcal{A}}}{\mathtt{leta}\; x \;=\; m_1\; \mathtt{in}\; m_2 \uparrow_{\mathcal{A}}} \quad \frac{m \uparrow_{\mathcal{A}}}{\mathtt{get}[w]m \uparrow_{\mathcal{A}}} \quad \frac{m \uparrow_{\mathcal{A}}}{\mathtt{inl}\; m \uparrow_{\mathcal{A}}} \quad \frac{m \uparrow_{\mathcal{A}}}{\mathtt{inr}\; m \uparrow_{\mathcal{A}}}$$

$$\frac{m \uparrow_{\mathcal{A}}}{\mathtt{case}\; m\; \mathtt{of}\; x.(m_1 \mid m_2) \uparrow_{\mathcal{A}}} \quad \frac{m \uparrow_{\mathcal{A}}}{\{\alpha = A; m : A'\} \uparrow_{\mathcal{A}}} \quad \frac{m \uparrow_{\mathcal{A}}}{\mathtt{open}\; \{\alpha, x\} = m\; \mathtt{in}\; m' \uparrow_{\mathcal{A}}}$$

Figure 9. Stuck states

C. Metatheory

C.1 Type safety

Definition C.1 (Authentication safety)**.** *We informally say that a well-typed term m under signatures $\Sigma_c; \Sigma_t$ and contexts $\Delta; \Gamma$ is authentication safe wrt. an authentication history $\mathcal{A}$ if:*

1. *$\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w$,*
2. *Any $\mathtt{iam}[\mathbf{a}]$ appearing in the term also appears in $\mathcal{A}$.*

We formally define this concept using a judgment $\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$ which is similar to the term typing judgment except for the rule corresponding to $\mathtt{iam}[\mathbf{a}]$:

$$\frac{\mathbf{a}{::}\mathtt{Prin} \in \Sigma_c \qquad \mathbf{a} \in \mathcal{A}}{\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{iam}[\mathbf{a}] : \mathtt{Iam}(\mathbf{a})@w}$$

Lemma C.2. *If $\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$, then $\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w$.*

Proof. By induction on derivation of $\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$. □

Lemma C.3 (Values and progress-making terms are not stuck)**.** *Let m be a term and $\mathcal{A}$ be a set of active principals. If $m\; \mathtt{val}_{\mathcal{A}}$ or $\exists m', \mathcal{A}'.m; \mathcal{A} \;\mapsto_w^{\Sigma_c;\Sigma_t;\Phi}\; m'; \mathcal{A}'$, then it is not the case that $m \uparrow_{\mathcal{A}}^{\Sigma_t;\Sigma_c}$.*

Proof. By induction on the derivation of $m\; \mathtt{val}_{\mathcal{A}}$ and $m; \mathcal{A} \;\mapsto_w^{\Sigma_c;\Sigma_t;\Phi}\; m'; \mathcal{A}'$. □

Theorem C.4 (Progress)**.** *Let $\Sigma_c; \Sigma_t$ be well-formed signatures and $\hat{\Phi}$ be an access control theory. Let $\mathcal{A}$ be a set of active principals from Σ_c. If $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c,\Phi;\Sigma_t} m : A@w$, then either $m\; \mathtt{val}_{\mathcal{A}}$, or $\exists m', \mathcal{A}'.m; \mathcal{A} \;\mapsto_w^{\Sigma_c;\Sigma_t;\hat{\Phi}}\; m'; \mathcal{A}'$.*

Proof. By induction on derivation of $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c,\Phi;\Sigma_t} m : A@w$.

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c,\Phi;\Sigma_t} m_1 : A_1@w \qquad \Delta; x{:}A_1@w \vdash^{\mathcal{A}}_{\Sigma_c,\Phi;\Sigma_t} m_2 : A_2@w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c,\Phi;\Sigma_t} \mathtt{let}\; x = m_1\; \mathtt{in}\; m_2 : A_2@w}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c,\hat{\Phi};\Sigma_t} m_1 : A_1@w$ — Premiss

Either $m_1\; \mathtt{val}_{\mathcal{A}}$,
or $\exists m_1', \mathcal{A}'.m_1; \mathcal{A} \;\mapsto_w\; m_1'; \mathcal{A}'$ — I.H.

Subcase: $m_1\; \mathtt{val}_{\mathcal{A}}$

$$\mathtt{let}\; x = m_1\; \mathtt{in}\; m_2; \mathcal{A} \;\mapsto_w\; [m_1/x]m_2; \mathcal{A}$$

Subcase: $m_1; \mathcal{A} \;\mapsto_w\; m_1'; \mathcal{A}'$

$$\mathtt{let}\; x = m_1\; \mathtt{in}\; m_2; \mathcal{A} \;\mapsto_w\; \mathtt{let}\; x = m_1'\; \mathtt{in}\; m_2; \mathcal{A}'$$

Case:

$$\frac{\Delta; x{:}A_1 \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_2@w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \lambda x{:}A_1.m : A_1 \to A_2@w}$$

$\lambda x{:}A_1.m\; \mathtt{val}_{\mathcal{A}}$ — Rule ($\to$ -V)

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_2 \to A_1 @ w \qquad \Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_2 @ w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} (m_1\ m_2) : A_1 @ w}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_2 \to A_1 @ w$ Premiss
Either m_1 $\mathtt{val}_{\mathcal{A}}$,
or $\exists m_1', \mathcal{A}'.m_1; \mathcal{A} \mapsto_w m_1'; \mathcal{A}'$ I.H.

Subcase: m_1 $\mathtt{val}_{\mathcal{A}}$.
$m_1 = \lambda x{:}A.m$ Lemma C.5
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_2 @ w$ Premiss
Either m_2 $\mathtt{val}_{\mathcal{A}}$,
or $\exists m_2', \mathcal{A}'.m_2; \mathcal{A} \mapsto_w m_2'; \mathcal{A}'$ I.H.

Subcase: m_2 $\mathtt{val}_{\mathcal{A}}$
$(\lambda x{:}A.m\ m_2); \mathcal{A} \mapsto_w [m_2/x]m; \mathcal{A}$

Subcase: $m_2; \mathcal{A} \mapsto_w m_2'; \mathcal{A}'$
$(\lambda x{:}A.m\ m_2); \mathcal{A} \mapsto_w (\lambda x{:}A.m\ m_2'); \mathcal{A}'$

Subcase: $m_1; \mathcal{A} \mapsto_w m_1'; \mathcal{A}'$
$(m_1\ m_2); \mathcal{A} \mapsto_w (m_1'\ m_2); \mathcal{A}'$

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_1 @ w \qquad \Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_2 @ w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \langle m_1, m_2\rangle : A_1 \times A_2 @ w}$$

By I.H. we get the following three cases:

Subcase: $m_1; \mathcal{A} \mapsto_w m_1'; \mathcal{A}'$
$\langle m_1, m_2\rangle; \mathcal{A} \mapsto_w \langle m_1', m_2\rangle; \mathcal{A}'$

Subcase: m_1 $\mathtt{val}_{\mathcal{A}}$ and $m_2; \mathcal{A} \mapsto_w m_2'; \mathcal{A}'$
$\langle m_1, m_2\rangle; \mathcal{A} \mapsto_w \langle m_1, m_2'\rangle; \mathcal{A}'$

Subcase: m_1 $\mathtt{val}_{\mathcal{A}}$ and m_2 $\mathtt{val}_{\mathcal{A}}$.
$\langle m_1, m_2\rangle$ $\mathtt{val}_{\mathcal{A}}$ Rule $(\times\text{-}\mathtt{V})$

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1 \times A_2 @ w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \pi_1 m : A_1 @ w}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1 \times A_2 @ w$ Premiss
Either m $\mathtt{val}_{\mathcal{A}}$,
or $\exists m', \mathcal{A}, .m; \mathcal{A} \mapsto_w m'; \mathcal{A}'$ I.H.

Subcase: m $\mathtt{val}_{\mathcal{A}}$
$m = \langle m_1, m_2\rangle$, Lemma C.5
$\pi_1\langle m_1, m_2\rangle; \mathcal{A} \mapsto_w m_1; \mathcal{A}$

Subcase: $m; \mathcal{A} \mapsto_w m'; \mathcal{A}'$
$\pi_1 m; \mathcal{A} \mapsto_w \pi_1 m'; \mathcal{A}'$

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1 \times A_2 @ w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \pi_2 m : A_2 @ w}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1 \times A_2 @ w$ Premiss
Either m $\mathtt{val}_{\mathcal{A}}$,
or $\exists m', \mathcal{A}, .m; \mathcal{A} \mapsto_w m'; \mathcal{A}'$ I.H.

Subcase: m $\mathtt{val}_{\mathcal{A}}$
$m = \langle m_1, m_2\rangle$, Lemma C.5
$\pi_2\langle m_1, m_2\rangle; \mathcal{A} \mapsto_w m_2; \mathcal{A}$

Subcase: $m; \mathcal{A} \mapsto_w m'; \mathcal{A}'$
$\pi_2 m; \mathcal{A} \mapsto_w \pi_2 m'; \mathcal{A}'$

Case:

$$\frac{\Delta \vdash_{\Sigma_c} w' \Leftarrow \mathtt{Wld} \qquad \Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A @ w' \qquad \Delta \vdash_{\Sigma_c} A\ \mathtt{mobile}}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{get}[w']m : A @ w}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A @ w'$ Premiss
Either m $\mathtt{val}_{\mathcal{A}}$
or $\exists m', \mathcal{A}, .m; \mathcal{A} \mapsto_{w'} m'; \mathcal{A}'$ I.H.

Subcase: m $\mathtt{val}_{\mathcal{A}}$

$$\mathtt{get}[w']m; \mathcal{A} \mapsto_w m; \mathcal{A}$$

Subcase: $m; \mathcal{A} \ \mapsto_{w'} \ m'; \mathcal{A}'$

$$\texttt{get}[w']m; \mathcal{A} \ \mapsto_w \ \texttt{get}[w']m'; \mathcal{A}'$$

Case:

$$\frac{\Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prop}}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{acquire}[A] : \exists\alpha{::}\texttt{Prf}(A).\texttt{unit option}@w}$$

$$\texttt{acquire}[A]; \mathcal{A} \ \mapsto_w \ \texttt{NONE}; \mathcal{A}$$

Case:

$$\frac{\Sigma_t(\mathbf{c}) = \forall\langle\alpha_1{:}K_1, \ldots, \alpha_n{:}K_n\rangle(A \to A') \quad \Delta \vdash_{\Sigma_c} A_k \Leftarrow [A_1/\alpha_1]\ldots[A_{k-1}/\alpha_{k-1}]K_k \ (k = 1..n) \qquad \Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : [A_1/\alpha_1]\ldots[A_n/\alpha_n]A@w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathbf{c}[A_1, \ldots, A_n](m) : [A_1/\alpha_1]\ldots[A_n/\alpha_n]A'@w}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : [A_1/\alpha_1]\ldots[A_n/\alpha_n]A@w$ Premiss

Either m $\texttt{val}_{\mathcal{A}}$,

or $\exists m', \mathcal{A}, .m; \mathcal{A} \ \mapsto_w \ m'; \mathcal{A}'$ I.H.

Subcase: $m; \mathcal{A} \ \mapsto_w \ m'; \mathcal{A}'$

$$\mathbf{c}[A_1, \ldots, A_n](m); \mathcal{A} \ \mapsto_w \ \mathbf{c}[A_1, \ldots, A_n](m'); \mathcal{A}'$$

Subcase: m $\texttt{val}_{\mathcal{A}}$

$\exists m. \cdot; \cdot \vdash_{\Sigma_c;\Sigma_t} m : [A_1/\alpha_1]\ldots[A_n/\alpha_n]A'@w$ Assumption

$\mathbf{c}[A_1, \ldots, A_n](v); \mathcal{A} \ \mapsto_w \ m; \mathcal{A}$

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1@w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{inl}\, m : A_1 \times A_2@w}$$

Applying I.H., we have the following two subcases:

Subcase: m $\texttt{val}_{\mathcal{A}}$:

$\texttt{inl}\, m$ $\texttt{val}_{\mathcal{A}}$ Rule $+$-$\texttt{V}_1$

Subcase: $\exists m', \mathcal{A}'.m; \mathcal{A} \ \mapsto_w \ m'; \mathcal{A}'$

$\texttt{inl}\, m; \mathcal{A} \ \mapsto_w \ \texttt{inl}\, m'; \mathcal{A}'$

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1@w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{inr}\, m : A_1 \times A_2@w}$$

Applying I.H., we have the following two subcases:

Subcase: m $\texttt{val}_{\mathcal{A}}$:

$\texttt{inr}\, m$ $\texttt{val}_{\mathcal{A}}$ Rule $+$-$\texttt{V}_2$

Subcase: $\exists m', \mathcal{A}'.m; \mathcal{A} \ \mapsto_w \ m'; \mathcal{A}'$

$\texttt{inr}\, m; \mathcal{A} \ \mapsto_w \ \texttt{inr}\, m'; \mathcal{A}'$

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1 + A_2@w \qquad \Delta; x{:}A_1@w \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A@w \qquad \Delta; x{:}A_2@w \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A@w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{case}\, m\, \texttt{of}\, x.(m_1 \mid m_2) : A@w}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1 + A_2@w$ Premiss

Applying I.H. we have the following subcases:

Subcase: m $\texttt{val}_{\mathcal{A}}$

Either $m = \texttt{inl}\, m'$, Lemma $C.12$

$\texttt{case inl}\, m'\, \texttt{of}\, x.(m_1 \mid m_2); \mathcal{A} \ \mapsto_w \ [m'/x]m_1; \mathcal{A}$

or $m = \texttt{inr}\, m'$ Lemma $C.12$

$\texttt{case inr}\, m'\, \texttt{of}\, x.(m_1 \mid m_2); \mathcal{A} \ \mapsto_w \ [m'/x]m_2; \mathcal{A}$

Subcase: $m; \mathcal{A} \ \mapsto_w \ m'; \mathcal{A}'$

$$\texttt{case}\, m\, \texttt{of}\, x.(m_1 \mid m_2); \mathcal{A} \ \mapsto_w \ \texttt{case}\, m'\, \texttt{of}\, x.(m_1 \mid m_2); \mathcal{A}'$$

Case:

$$\frac{\Delta \vdash_{\Sigma_c} A \Leftarrow K \qquad \Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : [A/\alpha]A'@w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \{\alpha = A; m : A'\} : \exists\alpha{::}K.A'@w}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : [A/\alpha]A'@w$ Premiss

Applying I.H., we get the following two subcases:

Subcase: m $\texttt{val}_{\mathcal{A}}$:

$\{\alpha = A; m : A'\}$ $\texttt{val}_{\mathcal{A}}$ Rule$\exists$-$\texttt{V}$

Subcase: $\exists m', \mathcal{A}'.m; \mathcal{A} \ \mapsto_w \ m'; \mathcal{A}'$

$\{\alpha = A; m : A'\}; \mathcal{A} \ \mapsto_w \ \{\alpha = A; m' : A'\}; \mathcal{A}'$

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : \exists\alpha{::}K.A@w \qquad \Delta, \alpha{::}K; x{:}A@w \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_2@w \qquad \Delta \vdash_{\Sigma_c} A_2 \Leftarrow \texttt{Type}}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{open}\, \{\alpha, x\} = m_1\, \texttt{in}\, m_2 : A_2@w}$$

$$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : \exists\alpha{::}K.A@w \quad \text{Premiss}$$

Applying I.H. we get the following subcases:

Subcase: $m_1\ \mathtt{val}_{\mathcal{A}}$
$m_1 = \{\alpha = A'; m' : A\}$ — Lemma $C.5$
$\mathtt{open}\ \{\alpha, x\} = \{\alpha = A'; m' : A\}\ \mathtt{in}\ m_2; \mathcal{A} \mapsto_w [A'/\alpha][m'/x]m_2; \mathcal{A}$

Subcase: $m_1; \mathcal{A} \mapsto_w m_1'; \mathcal{A}'$

$$\mathtt{open}\ \{\alpha, x\} = m_1\ \mathtt{in}\ m_2; \mathcal{A} \mapsto_w \mathtt{open}\ \{\alpha, x\} = m_1'\ \mathtt{in}\ m_2; \mathcal{A}'$$

Case:

$$\frac{\Delta \vdash_{\Sigma_c} w \Leftarrow \mathtt{Wld}}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{authenticate} : \exists\alpha{::}\mathtt{Prin}.\mathtt{Iam}(\alpha)\ \mathtt{option}@w}$$

$$\mathtt{authenticate}; \mathcal{A} \mapsto_w \mathtt{NONE}; \mathcal{A}$$

Case:

$$\frac{}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \langle\rangle : \mathtt{unit}@w}$$

$\langle\rangle\ \mathtt{val}_{\mathcal{A}}$

Case:

$$\frac{\mathbf{a} \in \mathcal{A}}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{iam}[\mathbf{a}] : \mathtt{Iam}(\mathbf{a})@w}$$

$\mathbf{a} \in \mathcal{A}$ — Assumption
$\mathtt{iam}[\mathbf{a}]\ \mathtt{val}_{\mathcal{A}}$

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{hold}\ m : A\ \mathtt{at}\ w@w}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$ — Premiss

By I.H. we have the following subcases:

Subcase: $m\ \mathtt{val}_{\mathcal{A}}$
$\mathtt{hold}\ m; \mathcal{A} \mapsto_w \mathtt{held}\ m; \mathcal{A}$

Subcase: $\exists m', \mathcal{A}'.m; \mathcal{A} \mapsto_w m'; \mathcal{A}'$.
$\mathtt{hold}\ m; \mathcal{A} \mapsto_w \mathtt{hold}\ m'; \mathcal{A}'$

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w \qquad \Delta \vdash_{\Sigma_c} w' \Leftarrow \mathtt{Wld}}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{held}\ m : A\ \mathtt{at}\ w@w'}$$

$\mathtt{held}\ m\ \mathtt{val}_{\mathcal{A}}$

Case:

$$\frac{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_1\ \mathtt{at}\ w'@w \qquad \Delta; x{:}A_1@w' \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_2@w}{\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{leta}\ x\ =\ m_1\ \mathtt{in}\ m_2 : A_2@w}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_1\ \mathtt{at}\ w'@w$ — Premiss

By I.H. we have the following subcases:

Subcase: $m_1\ \mathtt{val}_{\mathcal{A}}$
$m_1 = \mathtt{held}\ m,$ — Lemma $C.5$
$\mathtt{leta}\ x\ =\ \mathtt{held}\ m\ \mathtt{in}\ m_2; \mathcal{A} \mapsto_w [m/x]m_2; \mathcal{A}$

Subcase: $\exists m_1', \mathcal{A}'.m_1; \mathcal{A} \mapsto_w m_1'; \mathcal{A}'$
$\mathtt{leta}\ x\ =\ m_1\ \mathtt{in}\ m_2; \mathcal{A} \mapsto_w \mathtt{leta}\ x\ =\ m_1'\ \mathtt{in}\ m_2; \mathcal{A}'$

□

Lemma C.5 (Canonical forms)**.** *Let* $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$ *and* $m\ \mathtt{val}_{\mathcal{A}}$. *Then:*

1. *If* $A = A_1 \to A_2$, *then* $m = \lambda x{:}A_1.m_2$.
2. *If* $A = A_1 \times A_2$, *then* $m = \langle m_1, m_2\rangle$.
3. *If* $A = \mathtt{unit}$, *then* $m = \langle\rangle$.
4. *If* $A = \mathtt{Iam}(A')$, *then* $A' = \mathbf{a}$, *s.t.* $\Sigma_c(\mathbf{a}) = \mathtt{Prin}$, *and* $m = \mathtt{iam}[\mathbf{a}]$. *Furthermore* $\mathbf{a} \in \mathcal{A}$.
5. *If* $A = A\ \mathtt{at}\ w$, *then* $m = \mathtt{held}\ m'$,
6. *If* $A = A_1 + A_2$, *then either* $m = \mathtt{inl}\ m'$, *or* $m = \mathtt{inr}\ m'$.
7. *If* $A = \exists\alpha{::}K.A'$, *then* $m = \{\alpha = A_1; v : A_2\}$.

Proof. By induction on derivation of $m\ \mathtt{val}_{\mathcal{A}}$. □

Lemma C.6 (Term typing inversion)**.** *All the rules for $\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$ and $\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w$ are invertible.*

Proof. By induction on derivation of the resp. judgment forms. □

Definition C.7 (Consistency of APIs)**.** *Let $\Sigma_t(\mathbf{c}) = \forall\langle\alpha_1{::}K_1, \ldots, \alpha_n{::}K_n\rangle(A_1 \to A_2)@w$. Let $\mathcal{P}$ be the set of constants $\mathbf{a}$ s.t. $\mathbf{a}{::}\texttt{Prin} \in \Sigma_c$. The API $\mathbf{c}$ is* consistent *if the following holds:*

$$\begin{array}{ll}
\forall \mathcal{A} \in 2^{\mathcal{P}}.\forall B_1 \ldots B_n.\forall m_1.\forall m_2. & \\
& (\forall i \in [0..n-1].\cdot \vdash_{\Sigma_c} B_{i+1} \Leftarrow [B_1/\alpha_1] \ldots [B_i/\alpha_i]K_{i+1} \\
& \wedge \\
& \Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : [B_1/\alpha_1] \ldots [B_n/\alpha_n]A_1@w \\
& \wedge \\
& m_1\ \texttt{val}_{\mathcal{A}} \\
& \wedge \\
& \mathbf{c}[B_1, \ldots, B_n](m_1); \mathcal{A} \ \mapsto_w\ m_2; \mathcal{A}) \\
\supset & \\
& \Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : [B_1/\alpha_1] \ldots [B_n/\alpha_n]A_2@w
\end{array}$$

Lemma C.8 (Monotonicity of $\mathcal{A}$)**.** *If $m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'$, then $\mathcal{A} \subseteq \mathcal{A}'$.*

Proof. By induction on the definition of $m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'$. □

Lemma C.9 (Weakening of $\vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t}$ wrt $\mathcal{A}$)**.** *Let $\mathcal{A} \subseteq \mathcal{A}'$. If $\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$, then $\Delta; \Gamma \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m : A@w$*

Proof. We proceed by induction on derivation of $\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$. The only interesting case is the following:

Case:

$$\frac{\mathbf{a}{::}\texttt{Prin} \in \Sigma_c \qquad \mathbf{a} \in \mathcal{A}}{\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{iam}[\mathbf{a}] : \texttt{Iam}(\mathbf{a})@w}$$

$$\begin{array}{ll}
\mathbf{a} \in \mathcal{A} & \text{Premiss} \\
\mathcal{A} \subseteq \mathcal{A}' & \text{Assumption} \\
\mathbf{a} \in \mathcal{A}' & \\
\Delta; \Gamma \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m : A@w &
\end{array}$$

All other cases are proved by using the induction hypothesis in a straightforward manner. □

Lemma C.10 (Substitution for $\vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t}$)**.** *If $\Delta; \Gamma, x{:}A'@w' \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$, and $\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m' : A'@w'$, then $\Delta; \Gamma \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [m'/x]m : A@w$.*

Proof. By induction on the derivation of $\Delta; \Gamma, x{:}A'@w' \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$. □

Theorem C.11 (Preservation of consistency)**.** *Let $\Sigma_c; \Sigma_t$ be well-formed signatures, and $\hat{\Phi}$ be an access control theory. Further let all API constants declared in Σ_t be consistent in the sense of Def. C.7.*
If $m; \mathcal{A} \ \mapsto^{\Sigma_c;\Sigma_t;\hat{\Phi}}_w\ m'; \mathcal{A}'$ and $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$,
then $\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m' : A@w$.

Proof. Assume $m; \mathcal{A} \ \mapsto^{\Sigma_c;\Sigma_t;\Phi}_w\ m'; \mathcal{A}'$ and $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c,\Phi;\Sigma_t} m : A@w$.
We proceed by induction on derivation of $m; \mathcal{A} \ \mapsto^{\Sigma_c;\Sigma_t;\Phi}_w\ m'; \mathcal{A}'$.

Case:

$$\frac{m_1; \mathcal{A} \ \mapsto_w\ m_1'; \mathcal{A}'}{\texttt{let}\ x = m_1\ \texttt{in}\ m_2; \mathcal{A} \ \mapsto_w\ \texttt{let}\ x = m_1'\ \texttt{in}\ m_2; \mathcal{A}'}$$

$$\begin{array}{lr}
\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{let}\ x = m_1\ \texttt{in}\ m_2 : A@w & \text{Assumption} \\
\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_1@w, & \\
\Delta; x{:}A_1 \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A@w & \text{Lemma C.6} \\
\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_1' : A@w & \text{I.H.} \\
\Delta, \hat{\Phi}; x{:}A_1 \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_2 : A@w & \text{Lemma C.9, Lemma } C.15 \\
\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \texttt{let}\ x = m_1\ \texttt{in}\ m_2 : A@w &
\end{array}$$

Case:

$$\frac{m' \; \mathtt{val}_{\mathcal{A}}}{\mathtt{let}\; x = m' \;\mathtt{in}\; m; \mathcal{A} \;\mapsto_w\; [m'/x]m; \mathcal{A}}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{let}\; x = m' \;\mathtt{in}\; m : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m' : A_1@w$,
$\Delta; x{:}A_1 \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [m'/x]m : A@w$ Lemma C.10
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [m'/x]m : A@w$ Lemma $C.15$

Case:

$$\frac{m_1; \mathcal{A} \;\mapsto_w\; m_1'; \mathcal{A}'}{(m_1\; m_2); \mathcal{A} \;\mapsto_w\; (m_1'\; m_2); \mathcal{A}'}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} (m_1\; m_2) : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_2 \to A@w$,
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_2@w$ Lemma C.6
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_1' : A_2 \to A@w$ I.H.
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_2 : A_2@w$ Lemma C.9, Lemma $C.15$
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} (m_1'\; m_2) : A@w$

Case:

$$\frac{m_2; \mathcal{A} \;\mapsto_w\; m_2'; \mathcal{A}'}{(\lambda x{:}A_2.m_1\; m_2); \mathcal{A} \;\mapsto_w\; (\lambda x{:}A_2.m_1\; m_2'); \mathcal{A}'}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} (\lambda x{:}A_2.m_1\; m_2) : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \lambda x{:}A_2.m_1 : A_2 \to A@w$,
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_2@w$ Lemma C.6
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_2' : A_2@w$ I.H.
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \lambda x{:}A_2.m_1 : A_2 \to A@w$ Lemma C.9, Lemma $C.15$
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} (\lambda x{:}A_2.m_1\; m_2') : A@w$

Case:

$$\frac{m' \; \mathtt{val}_{\mathcal{A}}}{(\lambda x{:}A'.m\; m'); \mathcal{A} \;\mapsto_w\; [m'/x]m; \mathcal{A}}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} (\lambda x{:}A'.m\; m') : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \lambda x{:}A'.m : A' \to A@w$,
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m' : A'@w$ Lemma C.6
$\Delta; x{:}A'@w \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [m'/x]m : A@w$ Lemma C.10
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [m'/x]m : A@w$ Lemma $C.15$

Case:

$$\frac{m_1; \mathcal{A} \;\mapsto_w\; m_1'; \mathcal{A}'}{\langle m_1, m_2\rangle; \mathcal{A} \;\mapsto_w\; \langle m_1', m_2\rangle; \mathcal{A}'}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \langle m_1, m_2\rangle : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_1@w$,
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_2@w$,
$A = A_1 \times A_2$ Lemma $C.6$
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_1' : A_1@w$ I.H.
$\mathcal{A} \subseteq \mathcal{A}'$ Lemma $C.8$
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_2 : A_2@w$ Lemma $C.9$, Lemma $C.15$
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \langle m_1', m_2\rangle : A_1 \times A_2@w$

Case:

$$\frac{m_1 \; \mathtt{val}_{\mathcal{A}} \qquad m_2; \mathcal{A} \;\mapsto_w\; m_2'; \mathcal{A}'}{\langle m_1, m_2\rangle; \mathcal{A} \;\mapsto_w\; \langle m_1, m_2'\rangle; \mathcal{A}'}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \langle m_1, m_2\rangle : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_i : A_i@w$,
$A = A_1 \times A_2$ Lemma $C.6$
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_2' : A_2@w$ I.H.
$\mathcal{A} \subseteq \mathcal{A}'$ Lemma $C.8$
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_1 : A_1@w$ Lemma $C.9$, Lemma $C.15$
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \langle m_1, m_2\rangle : A@w$

Case:

$$\frac{m;\mathcal{A} \;\mapsto_w\; m';\mathcal{A}'}{\pi_1 m;\mathcal{A} \;\mapsto_w\; \pi_1 m';\mathcal{A}'}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \pi_1 m : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A \times A'@w$ Lemma C.6
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m' : A \times A'@w$ I.H.
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \pi_1 m' : A@w$

Case:

$$\frac{\langle m_1, m_2\rangle \;\texttt{val}_{\mathcal{A}}}{\pi_1\langle m_1, m_2\rangle;\mathcal{A} \;\mapsto_w\; m_1;\mathcal{A}}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \pi_1\langle m_1, m_2\rangle : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \langle m_1, m_2\rangle : A_1 \times A_2@w$,
$A = A_1$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_1@w$ Lemma C.6
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A_1@w$ Lemma $C.15$

Case:

$$\frac{m;\mathcal{A} \;\mapsto_w\; m';\mathcal{A}'}{\pi_2 m;\mathcal{A} \;\mapsto_w\; \pi_2 m';\mathcal{A}'}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \pi_2 m : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A' \times A@w$ Lemma C.6
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m' : A' \times A@w$ I.H.
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \pi_2 m' : A@w$

Case:

$$\frac{\langle m_1, m_2\rangle \;\texttt{val}_{\mathcal{A}}}{\pi_2\langle m_1, m_2\rangle;\mathcal{A} \;\mapsto_w\; m_2;\mathcal{A}}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \pi_2\langle m_1, m_2\rangle : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \langle m_1, m_2\rangle : A_1 \times A_2@w$,
$A = A_2$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_2@w$ Lemma C.6
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A_2@w$ Lemma $C.15$

Case:

$$\frac{m;\mathcal{A} \;\mapsto_{w'}\; m';\mathcal{A}'}{\texttt{get}[w']m;\mathcal{A} \;\mapsto_w\; \texttt{get}[w']m';\mathcal{A}'}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{get}[w']m : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w'$, $\cdot \vdash_{\Sigma_c} A\,\texttt{mobile}$ Lemma C.6
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m' : A@w'$ I.H.
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \texttt{get}[w']m' : A@w$

Case:

$$\frac{m\;\texttt{val}_{\mathcal{A}}}{\texttt{get}[w']m;\mathcal{A} \;\mapsto_w\; m;\mathcal{A}}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{get}[w']m : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w'$,
$\Delta \vdash_{\Sigma_c} A\,\texttt{mobile}$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$ Lemma C.13
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$ Lemma $C.15$

Case:

$$\frac{\mathbf{a}{::}\texttt{Prin} \in \Sigma_c}{\texttt{authenticate};\mathcal{A} \;\mapsto^{\Sigma_c;\Sigma_t;\Phi}_w\; \texttt{SOME}\,\{\alpha = \mathbf{a}; \texttt{iam}[\mathbf{a}] : \texttt{Iam}(\alpha)\};\mathcal{A} \cup \{\mathbf{a}\}}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{authenticate} : \exists\alpha{::}\texttt{Prin}.\texttt{Iam}(\alpha)\;\texttt{option}@w$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}\cup\{\mathbf{a}\}}_{\Sigma_c;\Sigma_t} \texttt{SOME}\,\{\alpha = \mathbf{a}; \texttt{iam}[\mathbf{a}] : \texttt{Iam}(\alpha)\} : \exists\alpha{::}\texttt{Prin}.\texttt{Iam}(\alpha)\;\texttt{option}@w$
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}\cup\{\mathbf{a}\}}_{\Sigma_c;\Sigma_t} \texttt{SOME}\,\{\alpha = \mathbf{a}; \texttt{iam}[\mathbf{a}] : \texttt{Iam}(\alpha)\} : \exists\alpha{::}\texttt{Prin}.\texttt{Iam}(\alpha)\;\texttt{option}@w$ Lemma $C.15$

Case:

$$\frac{}{\texttt{authenticate};\mathcal{A} \;\mapsto^{\Sigma_c;\Sigma_t;\Phi}_w\; \texttt{NONE};\mathcal{A}}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{authenticate} : \exists\alpha{::}\texttt{Prin}.\texttt{Iam}(\alpha)\;\texttt{option}@w$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{NONE} : \exists\alpha{::}\texttt{Prin}.\texttt{Iam}(\alpha)\;\texttt{option}@w$

Case:

$$\frac{}{\texttt{acquire}[A'];\mathcal{A} \ \mapsto_w \ \texttt{NONE};\mathcal{A}}$$

$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{acquire}[A'] : \exists\alpha{::}\texttt{Prf}(A').\texttt{unit option}@w$ Lemma C.6
$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{NONE} : \exists\alpha{::}\texttt{Prf}(A').\texttt{unit option}@w$
$\Delta,\hat{\Phi};\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{NONE} : \exists\alpha{::}\texttt{Prf}(A').\texttt{unit option}@w$ Lemma $C.15$

Case:

$$\frac{\hat{\Phi}_w \vdash_{\Sigma_c} A'' \Leftarrow \texttt{Prf}(A')}{\texttt{acquire}[A'];\mathcal{A} \ \mapsto_w \ \texttt{SOME}\,\{\alpha = A''; \langle\rangle : \texttt{unit}\};\mathcal{A}}$$

$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{acquire}[A'] : \exists\alpha{::}\texttt{Prf}(A').\texttt{unit option}@w$ Lemma C.6
$\hat{\Phi}_w \vdash_{\Sigma_c} A'' \Leftarrow \texttt{Prf}(A')$ Premiss
$\Delta,\hat{\Phi} \vdash_{\Sigma_c} A'' \Leftarrow \texttt{Prf}(A')$ Lemma C.14
$\Delta,\hat{\Phi};\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{SOME}\,\{\alpha = A''; \langle\rangle : \texttt{unit}\} : \exists\alpha{::}\texttt{Prf}(A').\texttt{unit option}@w$

Case:

$$\frac{m;\mathcal{A} \ \mapsto_w \ m';\mathcal{A}'}{\mathbf{c}[A_1,\ldots,A_n](m);\mathcal{A} \ \mapsto_w \ \mathbf{c}[A_1,\ldots,A_n](m');\mathcal{A}'}$$

$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathbf{c}[A_1,\ldots,A_n](m) : A@w$ Assumption
$\Sigma_t(\mathbf{c}) = \forall\langle\alpha_1{:}K_1,\ldots,\alpha_n{:}K_n\rangle(A' \to A'')$,
$\forall k \in [1..n].\Delta \vdash_{\Sigma_c} A_k \Leftarrow [\alpha_1/K_1]\ldots[\alpha_{k-1}/K_{k-1}]K_k$
$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : [A_1/\alpha_1]\ldots[A_n/\alpha_n]A'@w$ Lemma C.6
$A = [A_1/\alpha_1]\ldots[A_n/\alpha_n]A''$
$\Delta,\hat{\Phi};\cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m' : [A_1/\alpha_1]\ldots[A_n/\alpha_n]A'@w$ I.H.
$\forall k \in [1..n].\Delta,\hat{\Phi} \vdash_{\Sigma_c} A_k \Leftarrow [\alpha_1/K_1]\ldots[\alpha_{k-1}/K_{k-1}]K_k$ Lemma $C.14$
$\Delta,\hat{\Phi};\cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \mathbf{c}[A_1,\ldots,A_n](m') : A@w$

Case:

$$\frac{\Sigma_t(\mathbf{c}) = \forall\langle\alpha_1{::}K_1,\ldots,\alpha_n{::}K_n\rangle A \to A'@w \qquad \forall i \in [0..n-1]\cdot \vdash_{\Sigma_c} A_{i+1} \Leftarrow [A_1/\alpha_1]\ldots[A_i/\alpha_i]K_{i+1} \qquad m_1\ \texttt{val}_{\mathcal{A}}}{\mathbf{c}[A_1,\ldots,A_n](m_1);\mathcal{A} \ \mapsto^{\Sigma_c;\Sigma_t;\Phi}_w \ m_2;\mathcal{A}}$$

$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathbf{c}[A_1,\ldots,A_n](m_1) : A@w$ Assumption
$A = [A_1/\alpha_1]\ldots[A_n/\alpha_n]A''$
$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A@w$ Assumption about API consistency
$\Delta,\hat{\Phi};\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A@w$ Lemma $C.15$

Case:

$$\frac{m;\mathcal{A} \ \mapsto_w \ m';\mathcal{A}'}{\texttt{inl}\,m;\mathcal{A} \ \mapsto_w \ \texttt{inl}\,m';\mathcal{A}'}$$

$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{inl}\,m : A@w$ Assumption
$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A'@w$,
$A = A' + A''$ Lemma $C.6$
$m;\mathcal{A} \ \mapsto_w \ m';\mathcal{A}'$ Premiss
$\Delta,\hat{\Phi};\cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m' : A'@w$ I.H.
$\Delta,\hat{\Phi};\cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \texttt{inl}\,m' : A' + A''@w$

Case:

$$\frac{m;\mathcal{A} \ \mapsto_w \ m';\mathcal{A}'}{\texttt{inr}\,m;\mathcal{A} \ \mapsto_w \ \texttt{inr}\,m';\mathcal{A}'}$$

$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{inr}\,m : A@w$ Assumption
$\Delta;\cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A'@w$,
$A = A'' + A'$ Lemma $C.6$
$m;\mathcal{A} \ \mapsto_w \ m';\mathcal{A}'$ Premiss
$\Delta,\hat{\Phi};\cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m' : A'@w$ I.H.
$\Delta,\hat{\Phi};\cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \texttt{inr}\,m' : A'' + A'@w$

Case:

$$\frac{m;\mathcal{A} \ \mapsto_w \ m';\mathcal{A}'}{\texttt{case}\,m\,\texttt{of}\,x.(m_1 \mid m_2);\mathcal{A} \ \mapsto_w \ \texttt{case}\,m'\,\texttt{of}\,x.(m_1 \mid m_2);\mathcal{A}'}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t}$ $\texttt{case}\ m\ \texttt{of}\ x.(m_1 \mid m_2) : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1 + A_2@w,$
$\Delta; x{:}A_i@w \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_i : A@w,\ i \in [1..2]$ Lemma C.6
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m : A_1 + A_2@w$ I.H.
$\mathcal{A} \subseteq \mathcal{A}'$ Lemma C.8
$\Delta, \hat{\Phi}; x{:}A_i@w \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_i : A@w$ Lemma C.9, Lemma C.15
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \texttt{case}\ m\ \texttt{of}\ x.(m_1 \mid m_2) : A@w$

Case:

$$\frac{m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'}{\{\alpha = A; m : A'\}; \mathcal{A} \ \mapsto_w\ \{\alpha = A; m' : A'\}; \mathcal{A}'}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \{\alpha = A_1; m : A_2\} : A@w$ Assumption
$A = \exists\alpha{::}K.A_2,$
$\Delta \vdash_{\Sigma_c} A_1 \Leftarrow K,$
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : [A_1/\alpha]A_2@w$ Lemma C.6
$m; \mathcal{A} \ \mapsto_w\ m'; \mathcal{A}'$ Premiss
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m' : [A_1/\alpha]A_2@w$ I.H.
$\Delta, \hat{\Phi} \vdash_{\Sigma_c} A_1 \Leftarrow K$ Lemma C.14
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \{\alpha = A_1; m' : A_2\} : A@w$

Case:

$$\frac{m\ \texttt{val}_{\mathcal{A}}}{\texttt{case inl}\ m\ \texttt{of}\ x.(m_1 \mid m_2); \mathcal{A} \ \mapsto_w\ [m/x]m_1; \mathcal{A}}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{case inl}\ m\ \texttt{of}\ x.(m_1 \mid m_2) : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{inl}\ m : A_1 + A_2@w,$
$\Delta; x{:}A_1 \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : A@w$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [m/x]m_1 : A@w$ Lemma C.10
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [m/x]m_1 : A@w$ Lemma C.15

Case:

$$\frac{m\ \texttt{val}_{\mathcal{A}}}{\texttt{case inr}\ m\ \texttt{of}\ x.(m_1 \mid m_2); \mathcal{A} \ \mapsto_w\ [m/x]m_2; \mathcal{A}}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{case inr}\ m\ \texttt{of}\ x.(m_1 \mid m_2) : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{inr}\ m : A_1 + A_2@w,$
$\Delta; x{:}A_2 \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_2 : A@w$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_2@w$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [m/x]m_2 : A@w$ Lemma C.10
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [m/x]m_2 : A@w$ Lemma C.15

Case:

$$\frac{m_1; \mathcal{A} \ \mapsto_w\ m_1'; \mathcal{A}'}{\texttt{open}\ \{m_1, \alpha\} = x\ \texttt{in}\ m_2; \mathcal{A} \ \mapsto_w\ \texttt{open}\ \{m_1', \alpha\} = x\ \texttt{in}\ m_2; \mathcal{A}'}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{open}\ \{m_1, \alpha\} = x\ \texttt{in}\ m_2 : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_1 : \exists\alpha{::}K.A'@w,$
$\Delta, \alpha{::}K; x{:}A'@w \vdash_{\Sigma_c;\Sigma_t} m_2 : A@w$ Lemma C.6
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_1' : \exists\alpha{::}K.A'@w$ I.H.
$\mathcal{A} \subseteq \mathcal{A}'$ Lemma C.8
$\Delta, \hat{\Phi}, \alpha{::}K; x{:}A'@w \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} m_2 : A@w$ Lemma C.9, Lemma C.15
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}'}_{\Sigma_c;\Sigma_t} \texttt{open}\ \{m_1', \alpha\} = x\ \texttt{in}\ m_2 : A@w$

Case:

$$\frac{m\ \texttt{val}_{\mathcal{A}}}{\texttt{open}\ \{\{\alpha = A'; m : A''\}, \alpha\} = x\ \texttt{in}\ m; \mathcal{A} \ \mapsto_w\ [A'/\alpha][m/x]m'; \mathcal{A}}$$

$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \texttt{open}\ \{\{\alpha = A'; m : A''\}, \alpha\} = x\ \texttt{in}\ m' : A@w$ Assumption
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \{\alpha = A'; m : A''\} : \exists\alpha{::}K.A''@w,$
$\Delta, \alpha{::}K; x{:}A'' \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m' : A@w$ Lemma C.6
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [A/\alpha][m/x]m' : A@w$ Lemma C.10
$\Delta, \hat{\Phi}; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} [A/\alpha][m/x]m' : A@w$ Lemma C.15

□

Lemma C.12 (Inversion of term typing)**.** *All the typing rules for terms and values are invertible.*

Proof. By induction on the typing derivation. In each case, there is a single rule that could have applied. □

Lemma C.13 (Validity of mobile values)**.** *1. If* $\Delta; \cdot \vdash_{\Sigma_c;\Sigma_t} m : A@w$*, and* $m\ \mathtt{val}_{\mathcal{A}}$ *and* $\Delta \vdash_{\Sigma_c} A\ \mathtt{mobile}$*, and* $\Delta \vdash_{\Sigma_c} w' \Leftarrow \mathtt{Wld}$*, then* $\Delta; \cdot \vdash_{\Sigma_c;\Sigma_t} m : A@w'$

2. If $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w$*, and* $m\ \mathtt{val}_{\mathcal{A}}$ *and* $\Delta \vdash_{\Sigma_c} A\ \mathtt{mobile}$*, and* $\Delta \vdash_{\Sigma_c} w' \Leftarrow \mathtt{Wld}$*, then* $\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A@w'$

Proof. Proofs for the two parts are similar in structure and we illustrate only the first proof here.
We proceed by induction on the derivation of $m\ \mathtt{val}_{\mathcal{A}}$.

Case: $\dfrac{}{\lambda x{:}A_1.m\ \mathtt{val}_{\mathcal{A}}}$.
By Inversion Lemma C.12, we get $A = A_1 \to A_2$. However $A_1 \to A_2$ is not mobile. So this case does not arise.

Case: $\dfrac{m_1\ \mathtt{val}_{\mathcal{A}} \qquad m_2\ \mathtt{val}_{\mathcal{A}}}{\langle m_1, m_2\rangle\ \mathtt{val}_{\mathcal{A}}}$

$m_i\ \mathtt{val}_{\mathcal{A}},\ i \in \{1,2\}$	Premiss
$A = A_1 \times A_2, \Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_i : A_i@w$	Lemma C.12
$\Delta \vdash_{\Sigma_c} A_i\ \mathtt{mobile}$	Lemma C.16
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m_i : A_i@w'$	I.H.
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \langle m_1, m_2\rangle : A_1 \times A_2@w'$	

Case: $\dfrac{}{\langle\rangle\ \mathtt{val}_{\mathcal{A}}}$

$A = \mathtt{unit}$	Lemma C.12
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \langle\rangle : \mathtt{unit}@w'$	

Case: $\dfrac{m\ \mathtt{val}_{\mathcal{A}}}{\mathtt{inl}\ m\ \mathtt{val}_{\mathcal{A}}}$

$A = A_1 + A_2,$	
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1@w$	Lemma C.12
$\Delta \vdash_{\Sigma_c} A_1 + A_2\ \mathtt{mobile}$	Assumption
$\Delta \vdash_{\Sigma_c} A_1\ \mathtt{mobile}$	Lemma C.16
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1@w'$	I.H.
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{inl}\ m : A_1 + A_2@w'$	

Case: $\dfrac{m\ \mathtt{val}_{\mathcal{A}}}{\mathtt{inr}\ m\ \mathtt{val}_{\mathcal{A}}}$

$A = A_1 + A_2,$	
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_2@w$	Lemma C.12
$\Delta \vdash_{\Sigma_c} A_1 + A_2\ \mathtt{mobile}$	Assumption
$\Delta \vdash_{\Sigma_c} A_1\ \mathtt{mobile}$	Lemma C.16
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_2@w'$	I.H.
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{inr}\ m : A_1 + A_2@w'$	

Case: $\dfrac{}{\mathtt{held}\ m\ \mathtt{val}_{\mathcal{A}}}$

$A = A_1\ \mathtt{at}\ w_1,$	
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : A_1@w_1$	Lemma C.12
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{held}\ m : A_1\ \mathtt{at}\ w_1@w'$	

Case: $\dfrac{\mathbf{a} \in \mathcal{A}}{\mathtt{iam}[\mathbf{a}]\ \mathtt{val}_{\mathcal{A}}}$

$A = \mathtt{Iam}(\mathbf{a}), \Delta \vdash_{\Sigma_c} \mathbf{a} \Leftarrow \mathtt{Prin}$	Lemma C.12
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \mathtt{iam}[\mathbf{a}] : \mathtt{Iam}(\mathbf{a})@w'$	

Case: $\dfrac{m\ \mathtt{val}_{\mathcal{A}}}{\{\alpha = A_1; m : A_2\}\ \mathtt{val}_{\mathcal{A}}}$

$A = \exists\alpha{::}K.A_2, \Delta \vdash_{\Sigma_c} A_1 \Leftarrow K,$	
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : [A_1/\alpha]A_2@w$	Lemma C.12
$\Delta \vdash_{\Sigma_c} [A_1/\alpha]A_2\ \mathtt{mobile}$	Lemma C.17
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} m : [A_1/\alpha]A_2@w'$	I.H.
$\Delta; \cdot \vdash^{\mathcal{A}}_{\Sigma_c;\Sigma_t} \{A_1 = \alpha; m : A_2\} : \exists\alpha{::}K.A_2@w'$	

□

Lemma C.14 (Constructor kinding weakening)**.** *If* $\Delta \vdash_{\Sigma_c} A \Leftarrow K$*, then* $\Delta, \alpha{::}K' \vdash_{\Sigma_c} A \Leftarrow K$*.*

Proof. By induction on derivation of $\Delta \vdash_{\Sigma_c} A \Leftarrow K$. □

Lemma C.15 (Term typing weakening)**.** *1. If* $\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w$ *then* $\Delta, \alpha{::}K; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w$*.*

$\supset$	`imp`	$: \Pi\alpha{::}\mathtt{Prop}.\Pi\beta{::}\mathtt{Prop}.\mathtt{Prop}$
$\wedge$	`conj`	$: \Pi\alpha{::}\mathtt{Prop}.\Pi\beta{::}\mathtt{Prop}.\mathtt{Prop}$
$\langle\rangle$	`says`	$: \Pi\alpha{::}\mathtt{Prin}.\Pi\beta{::}\mathtt{Prop}.\mathtt{Prop}$
$\forall x{:}\mathtt{prin}$	`allp`	$: \Pi\alpha{::}\Pi\beta{::}\mathtt{Prin}.\mathtt{Prop}.\mathtt{Prop}$
$\forall x{:}\mathtt{db}$	`allr`	$: \Pi\alpha{::}\Pi\beta{::}\mathtt{Db}.\mathtt{Prop}.\mathtt{Prop}$

`impI`	$: \Pi\alpha{::}\mathtt{Prop}.\Pi\beta{::}\mathtt{Prop}.(\mathtt{Prf}(\alpha) \to \mathtt{Prf}(\beta)) \to \mathtt{Prf}(\alpha \supset \beta)$
`impE`	$: \Pi\alpha{::}\mathtt{Prop}.\Pi\beta{::}\mathtt{Prop}.\mathtt{Prf}(\alpha \supset \beta) \to \mathtt{Prf}(\alpha) \to \mathtt{Prf}(\beta)$
`conjI`	$: \Pi\alpha{::}\mathtt{Prop}.\Pi\beta{::}\mathtt{Prop}.\mathtt{Prf}(\alpha) \to \mathtt{Prf}(\beta) \to \mathtt{Prf}(\alpha \wedge \beta)$
$\mathtt{conjE}_1$	$: \Pi\alpha{::}\mathtt{Prop}.\Pi\beta{::}\mathtt{Prop}.\mathtt{Prf}(\alpha \wedge \beta) \to \mathtt{Prf}(\alpha)$
$\mathtt{conjE}_2$	$: \Pi\alpha{::}\mathtt{Prop}.\Pi\beta{::}\mathtt{Prop}.\mathtt{Prf}(\alpha \wedge \beta) \to \mathtt{Prf}(\beta)$
`allpI`	$: \Pi\alpha{::}\mathtt{Prin} \to \mathtt{Prop}.(\Pi\kappa{::}\mathtt{Prin}.\mathtt{Prf}((\alpha\ \kappa))) \to \mathtt{Prf}(\forall_p\alpha)$
`allrI`	$: \Pi\alpha{::}\mathtt{Db} \to \mathtt{Prop}.(\Pi\delta{::}\mathtt{Db}.\mathtt{Prf}((\alpha\ \delta))) \to \mathtt{Prf}(\forall_r\alpha)$
`allpE`	$: \Pi\alpha{::}\mathtt{Prin} \to \mathtt{Prop}.\Pi\kappa{::}\mathtt{Prin}.\mathtt{Prf}(\forall_p\alpha) \to \mathtt{Prf}((\alpha\ \kappa))$
`allrE`	$: \Pi\alpha{::}\mathtt{Db} \to \mathtt{Prop}.\Pi\delta{::}\mathtt{Db}.\mathtt{Prf}(\forall_p\alpha) \to \mathtt{Prf}((\alpha\ \delta))$
`saysI`	$: \Pi\alpha{::}\mathtt{Prop}.\Pi\kappa{::}\mathtt{Prin}.\kappa\ \mathtt{Affirms}\ \alpha \to \mathtt{Prf}(\langle\kappa\rangle\alpha)$
`truaff`	$: \Pi\alpha{::}\mathtt{Prop}.\Pi\kappa{::}\mathtt{Prin}.\mathtt{Prf}(\alpha) \to \kappa\ \mathtt{Affirms}\ \alpha$
`saysE`	$: \Pi\alpha{::}\mathtt{Prop}.\Pi\beta{::}\mathtt{Prop}.\Pi\kappa{::}\mathtt{Prin}.\mathtt{Prf}(\langle\kappa\rangle\alpha) \to (\mathtt{Prf}(\alpha) \to \kappa\ \mathtt{Affirms}\ \beta) \to \kappa\ \mathtt{Affirms}\ \beta$

Figure 10. $\Sigma_c\{\mathtt{logic}\}$: The static signature encoding the authorization logic. The propositional connectives are shown to the left of constants. We write $\Pi\alpha{::}K_1.K_2$ as $K_1{\to}K_2$ whenever α is not free in K_2.

$$\boxed{s \gg K}$$

$$\frac{}{\mathtt{prin} \gg \mathtt{Prin}} \qquad \frac{}{\mathtt{db} \gg \mathtt{Db}}$$

$$\boxed{\mathbf{\Sigma} \gg \Sigma_c}$$

$$\frac{}{\cdot \gg \cdot} \qquad \frac{s \gg K}{\mathbf{\Sigma}, \mathbf{a}{:}s \gg \Sigma_c, \mathbf{a}{::}K} \qquad \frac{\mathbf{\Sigma} \gg \Sigma_c \qquad \forall i.s_i \gg K_i}{\mathbf{\Sigma}, p{:}(s_1, \ldots, s_n) \gg \Sigma_c, p{::}K_1 \to \ldots \to K_n \to \mathtt{Prop}}$$

$$\boxed{\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \mathtt{ctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}}$$

$$\frac{\mathbf{\Sigma} \gg \Sigma_c}{\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \cdot\ \mathtt{ctx} \gg\ \vdash_{\Sigma_c} \cdot\ \mathtt{ctx}} \qquad \frac{\mathbf{\Sigma} \gg \Sigma_c \qquad \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \mathtt{ctx} \gg\ \vdash^{\mathcal{L}}_{\Sigma_c} \Delta\ \mathtt{ctx} \qquad s \gg K}{\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}, \alpha : s\ \mathtt{ctx} \gg\ \vdash_{\Sigma_c} \Delta, \alpha{::}K\ \mathtt{ctx}}$$

$$\boxed{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \mathtt{tctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}}$$

$$\frac{\mathbf{\Sigma} \gg \Sigma_c \qquad \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \mathtt{ctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}}{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \cdot\ \mathtt{tctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}}$$

$$\frac{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \mathtt{tctx} \gg\ \vdash_{\Sigma_c} \Delta_1, \Delta_2\ \mathtt{ctx} \qquad \mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{prop} \gg \Delta_1 \vdash_{\Sigma_c} A \Leftarrow \mathtt{Prop} \qquad \alpha \text{ fresh}}{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi, P\ \mathtt{true}\ \mathtt{tctx} \gg\ \vdash_{\Sigma_c} \Delta_1, \Delta_2, \alpha{::}\mathtt{Prf}(A)\ \mathtt{ctx}}$$

Figure 11. Encoding signatures and contexts

2. *If* $\Delta; \Gamma \vdash_{\Sigma_c;\Sigma_t} m : A@w$ *then* $\Delta; \Gamma, x{:}A'@w' \vdash_{\Sigma_c;\Sigma_t} m : A@w$.
   The same results hold for the relation $\vdash^{A}_{\Sigma_c;\Sigma_t}$.

Lemma C.16 (Inversion of mobility)**.** *All rules for* $\Delta \vdash_{\Sigma_c} A\ \mathtt{mobile}$ *are invertible.*

Proof. By induction on the derivations of $\Delta \vdash_{\Sigma_c} A\ \mathtt{mobile}$ for various A's. □

Lemma C.17 (Substitution for mobility)**.** *If* $\Delta, \alpha{::}K \vdash_{\Sigma_c} A\ \mathtt{mobile}$*, and* $\Delta \vdash_{\Sigma_c} A' \Leftarrow K$*, then* $[A'/\alpha]\Delta \vdash_{\Sigma_c} [A'/\alpha]A\ \mathtt{mobile}$.

Proof. By induction on the derivation of $\Delta, \alpha{::}K \vdash_{\Sigma_c} A\ \mathtt{mobile}$. □

Lemma C.18 (Inversion of SOS rules)**.** *All rules defining the operational semantics are invertible.*

D. Adequacy of encoding of authorization logic

D.1 The encoding

Figure 10 shows the constructor constants used to encode propositions and proofs from the authorization in the language. Using these constructors, the various judgment forms of the logic are encoded as constructor kinding judgments in PCML$_5$. We use the judgment form $J_1 \gg J_2$ to represent the encoding of judgment J_1 of logic as the judgment J_2 of PCML$_5$.

Lemma D.1 (Correctness of encoding)**.** *1. If* $\mathbf{\Sigma} \gg \Sigma_c$*, then* Σ_c ok.
2. *If* $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \mathtt{ctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}$*, then* $\mathbf{\Sigma} \gg \Sigma_c$*, and* $\vdash_{\Sigma_c} \Delta\ \mathtt{ctx}$.
3. *If* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \mathtt{tctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}$*, then* $\mathbf{\Sigma} \gg \Sigma_c$*,* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \mathtt{tctx}$*, and* $\vdash_{\Sigma_c} \Delta\ \mathtt{ctx}$.
4. *If* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{prop} \gg \Delta \vdash_{\Sigma_c} A \Leftarrow K$*, then*

$$\frac{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \texttt{Prop} \qquad \mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_2\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} A_2 \Leftarrow \texttt{Prop}}{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1 \supset P_2\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} ((\texttt{imp}\ A_1)\ A_2) \Leftarrow \texttt{Prop}}(\supset\gg)$$

$$\frac{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \texttt{Prop} \qquad \mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_2\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} A_2 \Leftarrow \texttt{Prop}}{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1 \wedge P_2\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} ((\texttt{conj}\ A_1)\ A_2) \Leftarrow \texttt{Prop}}(\wedge\gg)$$

$$\frac{s \gg K \qquad \mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \texttt{Prop} \qquad \mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a : s \gg \Delta \vdash_{\Sigma_c^+} A_2 \Leftarrow K}{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \langle a \rangle P\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} ((\texttt{says}\ A_1)\ A_2) \Leftarrow \texttt{Prop}}(\texttt{says}\gg)$$

$$\frac{s \gg \texttt{Prin} \qquad \mathbf{\Delta}, \alpha{:}s \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{prop} \gg \Delta, \alpha{::}\texttt{Prin} \vdash_{\Sigma_c^+} A \Leftarrow \texttt{Prop}}{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \forall \alpha{:}s.P\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} (\texttt{allp}\ \lambda\alpha{::}\texttt{Prin}.A) \Leftarrow \texttt{Prop}}(\forall_{\texttt{p}}\gg)$$

$$\frac{s \gg \texttt{Db} \qquad \mathbf{\Delta}, \alpha{:}s \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{prop} \gg \Delta, \alpha{::}\texttt{Db} \vdash_{\Sigma_c^+} A \Leftarrow \texttt{Prop}}{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \forall \alpha{:}s.P\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c^+} (\texttt{allr}\ \lambda\alpha{::}\texttt{Db}.A) \Leftarrow \texttt{Prop}}(\forall_{\texttt{r}}\gg)$$

Figure 12. Encoding propositions

(a) $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \texttt{ctx} \gg\ \vdash_{\Sigma_c} \Delta\ \texttt{ctx}$,
(b) $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{prop}$
(c) $K = \texttt{Prop}$, $\Delta \vdash_{\Sigma_c} A \Leftarrow K$

5. *If* $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} A \Leftarrow K$*, then*
   (a) $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{true}$
   (b) $K = \texttt{Prf}(A')$, $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c} A' \Leftarrow \texttt{Prop}$, $\Delta \vdash_{\Sigma_c} A \Leftarrow K$
6. *If* $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \texttt{affirms}\ P \gg \Delta \vdash_{\Sigma_c} A \Leftarrow K$*, then*
   (a) $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \texttt{affirms}\ P$
   (b) $K = A_1\ \texttt{Affirms}\ A_2$, $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{prop} \gg \Delta \vdash_{\Sigma_c} A_2 \Leftarrow \texttt{Prop}$, $\mathbf{\Delta} \vdash_{\mathbf{\Sigma}} a : \texttt{prin} \gg \Delta \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Prin}$.
   (c) $\Delta \vdash_{\Sigma_c} A \Leftarrow K$

The strongest subsumption relation for PCML$_5$ kinds under an empty signature Σ_c is denoted by

$$\begin{array}{rcrcl} \preceq_{\texttt{pcml}} & = & \{ \quad \texttt{Prin}, \texttt{Db}, \texttt{Wld} & \preceq & \texttt{Type} \\ & & \texttt{Prop} & \preceq & \texttt{Prf} \\ & & \texttt{Prop}, \texttt{Prin} & \preceq & \texttt{Affirm} \\ & & \}^\star & & \end{array}$$

In the following, we shall use $\preceq_1 \otimes \preceq_2$ to denote the least upper bound of $\preceq_1$ and $\preceq_2$, i.e. the strongest subordination relation weaker than both $\preceq_1$ and $\preceq_2$.

Lemma D.2 (Transport of canonical forms)**.** *Let* Σ_c *be a signature and* $\preceq_{\Sigma_c}$ *be the strongest subordination relation for* Σ_c*. Let* $\preceq = \preceq_{\texttt{pcml}} \otimes \preceq_{\Sigma_c}$*. Let* $\vdash_{\Sigma_c, \preceq} \Delta\ \texttt{ctx}$*,* $\Delta \vdash_{\Sigma_c, \preceq} K\ \texttt{kind}$*,* $\Delta \vdash_{\Sigma_c, \preceq} K'\ \texttt{kind}$*, and* $K \preceq K'$*. Let* $\Delta' = \Delta|^{\preceq}_{K'}$*, and* $\Sigma_c' = \Sigma_c|^{\preceq}_{K'}$*. Then,*

1. $\Delta' \vdash_{\Sigma_c', \preceq} K\ \texttt{kind}$
2. *For any* A*,* $\Delta \vdash_{\Sigma_c} A \Leftarrow K$ *iff* $\Delta' \vdash_{\Sigma_c'} A \Leftarrow K$*.*
3. *For any* N*,* $\Delta \vdash_{\Sigma_c} N \Rightarrow K$ *iff* $\Delta' \vdash_{\Sigma_c'} N \Rightarrow K$*.*

Proof. The "if" part follows from Weakening. For the other direction, we proceed by a simultaneous induction on the mutually recursive definitions of the above three judgments. We present the three cases itemwise, showing only the typical cases in each:

1. $\Delta \vdash_{\Sigma_c} K\ \texttt{kind}$:
Case:

$$\frac{\Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prop}}{\Delta \vdash_{\Sigma_c} \texttt{Prf}(A)\ \texttt{kind}}$$

$\Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prop}$	Premiss
$\texttt{Prf} \preceq K'$	Assumption
$\texttt{Prop} \preceq \texttt{Prf}$	By definition of $\preceq$
$\texttt{Prop} \preceq K'$	Transitivity of $\preceq$
$\Delta' \vdash_{\Sigma_c'} A \Leftarrow \texttt{Prop}$	I.H.
$\Delta \vdash_{\Sigma_c} \texttt{Prf}(A)\ \texttt{kind}$	

2. $\Delta \vdash_{\Sigma_c} A \Leftarrow K$:

$$\frac{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \texttt{tctx} \gg\ \vdash_{\Sigma_c} \Delta_1, \Delta_2\ \texttt{ctx} \qquad \mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{prop} \gg \Delta_1 \vdash_{\Sigma_c} A \Leftarrow \texttt{Prop} \qquad \alpha\ \text{fresh}}{\mathbf{\Delta}; \Phi, P\ \texttt{true} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{true} \gg \Delta_1, \Delta_2, \alpha{::}\texttt{Prf}(A) \vdash_{\Sigma_c} \alpha \Leftarrow \texttt{Prf}(A)}\,(\texttt{hyp} \gg)$$

$$\frac{\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1\ \texttt{prop} \gg \Delta_1 \vdash_{\Sigma_c^+} A_1 \Leftarrow \texttt{Prop} \qquad \mathbf{\Delta}; \Phi, P_1\ \texttt{true} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_2\ \texttt{true} \gg \Delta_1, \Delta_2, \alpha{::}\texttt{Prf}(A_1) \vdash_{\Sigma_c^+} A \Leftarrow \texttt{Prf}(A_2)}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1 \supset P_2\ \texttt{true} \gg \Delta_1, \Delta_2 \vdash_{\Sigma_c^+} (\texttt{impI}\ A_1\ A_2\ \lambda\alpha{::}\texttt{Prf}(A_1).A) \Leftarrow \texttt{Prf}(A_1 \supset A_2)}\,(\supset\text{-I} \gg)$$

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1 \supset P_2\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prf}(A_1' \supset A_2') \qquad \mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Prf}(A_1')}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_2\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} \texttt{impE}\ A_1'\ A_2'\ A\ A_1 \Leftarrow \texttt{Prf}(A_2')}\,(\supset\text{-E} \gg)$$

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Prf}(A_1') \qquad \mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_2\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} A_2 \Leftarrow \texttt{Prf}(A_2')}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1 \wedge P_2\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} \texttt{conjI}\ A_1'\ A_2'\ A_1\ A_2 \Leftarrow \texttt{Prf}(A_1' \wedge A_2')}\,(\wedge\text{-I} \gg)$$

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1 \wedge P_2\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prf}(A_1 \wedge A_2)}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} \texttt{conjE}_1\ A_1\ A_2\ A \Leftarrow \texttt{Prf}(A_1)}\,(\wedge\text{-E}_1 \gg)$$

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1 \wedge P_2\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prf}(A_1 \wedge A_2)}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_2\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} \texttt{conjE}_2\ A_1\ A_2\ A \Leftarrow \texttt{Prf}(A_2)}\,(\wedge\text{-E}_2 \gg)$$

$$\frac{\mathbf{\Delta}, \alpha{:}\texttt{prin}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{true} \gg \Delta, \alpha{::}\texttt{Prin} \vdash_{\Sigma_c} A \Leftarrow \texttt{Prf}(A')}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \forall\alpha{:}\texttt{prin}.P\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} \texttt{allpI}\ \lambda\alpha{::}\texttt{Prin}.A'\ \lambda\alpha{::}\texttt{Prin}.A \Leftarrow \texttt{Prf}(\forall_p \lambda\alpha{::}\texttt{Prin}.A')}\,(\forall_{\mathrm{p}}\text{-I} \gg)$$

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \forall\alpha{:}\texttt{prin}.P\ \texttt{true} \gg \Delta_1, \Delta_2 \vdash_{\Sigma_c} A \Leftarrow \texttt{Prf}(\texttt{allp}\ A') \qquad \mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a : \texttt{prin} \gg \Delta_1 \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Prin}}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} [a/\alpha]P\ \texttt{true} \gg \Delta_1, \Delta_2 \vdash_{\Sigma_c} \texttt{allpE}\ A'\ A_1\ A \Leftarrow \texttt{Prf}(A'\ A_1)}\,(\forall_{\mathrm{p}}\text{-E} \gg)$$

$$\frac{\mathbf{\Delta}, \alpha{:}\texttt{db}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{true} \gg \Delta, \alpha{::}\texttt{Db} \vdash_{\Sigma_c} A \Leftarrow \texttt{Prf}(A')}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \forall\alpha{:}\texttt{db}.P\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} \texttt{allrI}\ \lambda\alpha{::}\texttt{Db}.A'\ \lambda\alpha{::}\texttt{Db}.A \Leftarrow \texttt{Prf}(\forall_p \lambda\alpha{::}\texttt{Db}.A')}\,(\forall_{\mathrm{r}}\text{-I} \gg)$$

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \forall\alpha{:}\texttt{db}.P\ \texttt{true} \gg \Delta_1, \Delta_2 \vdash_{\Sigma_c} A \Leftarrow \texttt{Prf}(\texttt{allr}\ A') \qquad \mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a : \texttt{db} \gg \Delta_1 \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Db}}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} [a/\alpha]P\ \texttt{true} \gg \Delta_1, \Delta_2 \vdash_{\Sigma_c} \texttt{allrE}\ A'\ A_1\ A \Leftarrow \texttt{Prf}(A'\ A_1)}\,(\forall_{\mathrm{r}}\text{-E} \gg)$$

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \texttt{affirms}\ P \gg \Delta \vdash_{\Sigma_c} A \Leftarrow A_1\ \texttt{Affirms}\ A_2}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \langle a \rangle P\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} \texttt{saysI}\ A_2\ A_1\ A \Leftarrow \texttt{Prf}(\langle A_1 \rangle A_2)}\,(\texttt{says}\text{-I} \gg)$$

$$\frac{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{true} \gg \Delta_1, \Delta_2 \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Prf}(A_1') \qquad \mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a : \texttt{prin} \gg \Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prin}}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \texttt{affirms}\ P \gg \Delta_1, \Delta_2 \vdash_{\Sigma_c} \texttt{truaff}\ A_1'\ A\ A_1 \Leftarrow A\ \texttt{Affirms}\ A_1'}\,(\texttt{truaff} \gg)$$

$$\frac{\begin{array}{c}\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \langle a \rangle P_1\ \texttt{true} \gg \Delta \vdash_{\Sigma_c} A_1 \Leftarrow \texttt{Prf}(\langle A \rangle A_1') \\ \mathbf{\Delta}; \Phi, P_1\ \texttt{true} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \texttt{affirms}\ P_2 \gg \Delta, \alpha{::}\texttt{Prf}(A_1') \vdash_{\Sigma_c} A_2 \Leftarrow A\ \texttt{Affirms}\ A_2'\end{array}}{\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} a\ \texttt{affirms}\ P_2 \gg \Delta \vdash_{\Sigma_c} \texttt{saysE}\ A_1'\ A_2'\ A\ A_1\ (\lambda\alpha{::}\texttt{Prf}(A_1').A_2) \Leftarrow A\ \texttt{Affirms}\ A_2'}\,(\texttt{says}\text{-E} \gg)$$

Figure 13. Encoding proofs and affirmations

Case:

$$\frac{\Delta \vdash_{\Sigma_c} N \Rightarrow K \qquad K \neq \Pi\alpha{::}K_1.K_2}{\Delta \vdash_{\Sigma_c} N \Leftarrow K}$$

$K \preceq K'$	Assumption
$\Delta' \vdash_{\Sigma_c'} N \Rightarrow K$	I.H.
$K \neq \Pi\alpha{::}K_1.K_2$	Premiss
$\Delta' \vdash_{\Sigma_c'} N \Leftarrow K$	

Case:

$$\frac{\Delta, \alpha{::}K_1 \vdash_{\Sigma_c} A \Leftarrow K_2}{\Delta \vdash_{\Sigma_c} \lambda\alpha{::}K_1.A \Leftarrow \Pi\alpha{::}K_1.K_2}$$

$\lvert\Pi\alpha{::}K_1.K_2\rvert \preceq \lvert K'\rvert$	Assumption
$\lvert K_2\rvert \preceq \lvert K'\rvert$	$\lvert\Pi\alpha{::}K_1.K_2\rvert = \lvert K_2\rvert$

Subcase: $K_1 \preceq K'$:

$(\Delta, \alpha{::}K_1)\rvert^{\preceq}_{K'} = \Delta\rvert^{\preceq}_{K'}, \alpha{::}K_1$	
$\Delta\rvert^{\preceq}_{K'}, \alpha{::}K_1 \vdash_{\Sigma_c'} A \Leftarrow K_2$	I.H.
$\Delta\rvert^{\preceq}_{K'} \vdash_{\Sigma_c'} \lambda\alpha{::}K_1.A \Leftarrow \Pi\alpha{::}K_1.K_2$	

Subcase: $K_1 \not\preceq K'$:

$(\Delta, \alpha{::}K_1)\rvert^{\preceq}_{K'} = \Delta\rvert^{\preceq}_{K'}$	
$\Delta\rvert^{\preceq}_{K'} \vdash_{\Sigma_c'} A \Leftarrow K$	I.H.
$\Delta\rvert^{\preceq}_{K'}, \alpha{::}K_1 \vdash_{\Sigma_c'} A \Leftarrow K$	Weakening
$\Delta\rvert^{\preceq}_{K'} \vdash_{\Sigma_c'} \lambda\alpha{::}K_1.A \Leftarrow \Pi\alpha{::}K_1.K_2$	

3. $\Delta \vdash_{\Sigma_c} N \Rightarrow K$:

Case:

$$\frac{}{\Delta_1, \alpha{::}K, \Delta_2 \vdash_{\Sigma_c} \alpha \Rightarrow K}$$

$K \preceq K'$	Assumption
$\Delta' = \Delta_1\rvert^{\preceq}_{K'}, \alpha{::}K, \Delta_2\rvert^{\preceq}_{K'}$	
$\Delta' \vdash_{\Sigma_c} \alpha \Rightarrow K$	

Case:

$$\frac{\Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Type} \qquad \Delta \vdash_{\Sigma_c} w \Leftarrow \texttt{Wld}}{\Delta \vdash_{\Sigma_c} A \; \texttt{at} \; w \Rightarrow \texttt{Type}}$$

$\texttt{Type} \preceq K'$	Assumption
$\texttt{Wld} \preceq \texttt{Type}$	Definition of $\preceq$
$\texttt{Wld} \preceq K'$	Transitivity of $\preceq$
$\Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Type}$	Premiss
$\Delta \vdash_{\Sigma_c} w \Leftarrow \texttt{Wld}$	Premiss
$\Delta' \vdash_{\Sigma_c'} A \Leftarrow \texttt{Type}$	I.H.
$\Delta' \vdash_{\Sigma_c'} w \Leftarrow \texttt{Wld}$	I.H.
$\Delta' \vdash_{\Sigma_c'} A \; \texttt{at} \; w \Rightarrow \texttt{Type}$	

Case:

$$\frac{\Delta \vdash_{\Sigma_c} A \Leftarrow K_1 \qquad \Delta \vdash_{\Sigma_c} N \Rightarrow \Pi\alpha{::}K_1.K_2}{\Delta \vdash_{\Sigma_c} (N\; A) \Rightarrow [A/\alpha]K_2}$$

$\lvert[A/\alpha]K_2\rvert \preceq K'$	Assumption
$\lvert[A/\alpha]K_2\rvert = \lvert K_2\rvert$	By induction on structure of kinds
$\lvert\Pi\alpha{::}K_1.K_2\rvert = \lvert K_2\rvert$	Definition of $\lvert\cdot\rvert$
$\lvert\Pi\alpha{::}K_1.K_2\rvert \preceq K'$	
$\Delta \vdash_{\Sigma_c} N \Rightarrow \Pi\alpha{::}K_1.K_2$	Premiss
$\Delta' \vdash_{\Sigma_c'} N \Rightarrow \Pi\alpha{::}K_1.K_2$	I.H.
$\Delta \vdash_{\Sigma_c} A \Leftarrow K_1$	Premiss
$\Delta' \vdash_{\Sigma_c'} A \Leftarrow K_1$	I.H.
$\Delta \vdash_{\Sigma_c} (N\; A) \Rightarrow [A/\alpha]K_2$	

□

Lemma D.3 (Inversion of encoding). *1. All the rules for* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P \; \texttt{prop} \gg \Delta \vdash_{\Sigma_c} A \Leftarrow \texttt{Prop}$ *are invertible.*

Lemma D.4 (Inversion of canonical forms of kind Prop). *If* $\Delta \vdash_{\Sigma_c^+} A \Leftarrow \texttt{Prop}$ *where* $\mathbf{\Sigma} \gg \Sigma_c$*, and* $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta} \; \texttt{ctx} \gg \; \vdash_{\Sigma_c} \Delta \; \texttt{ctx}$*, then the following holds:*

1. *Either* $A = \texttt{imp}\; A_1\; A_2$ *where* $\Delta \vdash_{\Sigma_c} A_i \Leftarrow \texttt{Prop}$*, or,*
2. $A = \texttt{conj}\; A_1\; A_2$ *where* $\Delta \vdash_{\Sigma_c} A_i \Leftarrow \texttt{Prop}$*, or,*

3. $A = \mathtt{says}\ A_1\ A_2$ *where* $\Delta \vdash_{\Sigma_c} A_1 \Leftarrow \mathtt{Prin}$, *and* $\Delta \vdash_{\Sigma_c} A_2 \Leftarrow \mathtt{Prop}$, *or,*
4. $A = \mathtt{allp}\ \lambda\alpha{::}\mathtt{Prin}.A'$ *where* $\Delta, \alpha{::}\mathtt{Prin} \vdash_{\Sigma_c} A' \Leftarrow \mathtt{Prop}$, *or,*
5. $A = \mathtt{allr}\ \lambda\alpha{::}\mathtt{Prin}.A'$ *where* $\Delta, \alpha{::}\mathtt{Db} \vdash_{\Sigma_c} A' \Leftarrow \mathtt{Prop}$

Proof. By induction of derivation of $\Delta \vdash_{\Sigma_c^+} A \Leftarrow \mathtt{Prop}$. □

Lemma D.5 (Inversion of canonical forms of kind $\mathtt{Prf}()$ and $\mathtt{Affirms}$). *1. If* $\Delta \vdash_{\Sigma_c^+} A \Leftarrow \mathtt{Prf}(A')$ *and* $\mathbf{\Sigma} \gg \Sigma_c$ *and* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \mathtt{tctx} \gg\ \vdash_{\Sigma_c^+} \Delta\ \mathtt{ctx}$*, then one of the following holds:*

- $\Delta = \Delta', \alpha{::}\mathtt{Prf}(A')$ *and* $\Delta' \vdash_{\Sigma_c^+} A' \Leftarrow \mathtt{Prop}$.
- $A = \mathtt{impI}\ A_1\ A_2\ \lambda\alpha{::}\mathtt{Prf}(A_1).A_3$, $A' = A_1 \supset A_2$. *Also* $\Delta, \alpha{::}\mathtt{Prf}(A_1) \vdash_{\Sigma_c^+} A_3 \Leftarrow \mathtt{Prf}(A_2)$ *and* $\Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \mathtt{Prop}$ *are derived as strict sub-derivations.*
- $A = \mathtt{impE}\ A_1' A' A_2 A_1$. *Moreover* $\Delta \vdash_{\Sigma_c^+} A_2 \Leftarrow \mathtt{Prf}(A_1' \supset A')$ *and* $\Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \mathtt{Prf}(A_1')$ *are derived as strict sub-derivations.*
- $A = \mathtt{conjI}\ A_1'\ A_2'\ A_1\ A_2$ *and* $A' = A_1' \wedge A_2'$. *Moreover* $\Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \mathtt{Prf}(A_1')$ *and* $\Delta \vdash_{\Sigma_c^+} A_2 \Leftarrow \mathtt{Prf}(A_2')$ *are derived as strict sub-derivations.*
- $A = \mathtt{conjE}_1\ A'\ A''\ A_1$, *and* $\Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \mathtt{Prf}(A' \wedge A'')$ *is derived as a strict sub-derivation.*
- $A = \mathtt{conjE}_2\ A''\ A'\ A_2$, *and* $\Delta \vdash_{\Sigma_c^+} A_2 \Leftarrow \mathtt{Prf}(A'' \wedge A')$ *is derived as a strict sub-derivation.*
- $A = \mathtt{allpI}\ \lambda\alpha{::}\mathtt{Prin}.A_1\ \lambda\alpha{::}\mathtt{Prin}.A_2$ *and* $A' = (\mathtt{allp}\ \lambda\alpha{::}\mathtt{Prin}.A_1)$. *Moreover* $\Delta, \alpha{::}\mathtt{Prin} \vdash_{\Sigma_c^+} A_2 \Leftarrow \mathtt{Prf}(A_1)$ *is derived as a strict sub-derivation.*
- $A = \mathtt{allpE}\ A_3\ A_1\ A_2$ *and* $A' = (A_3\ A_1)$. *Moreover* $\Delta \vdash_{\Sigma_c^+} A_2 \Leftarrow \mathtt{Prf}((\mathtt{allp}\ A_3))$ *and* $\Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \mathtt{Prin}$ *are derived as strict sub-derivations.*
- $A = \mathtt{allrI}\ \lambda\alpha{::}\mathtt{Db}.A_1\ \lambda\alpha{::}\mathtt{Db}.A_2$ *and* $A' = (\mathtt{allp}\ \lambda\alpha{::}\mathtt{Db}.A_1)$. *Moreover* $\Delta, \alpha{::}\mathtt{Db} \vdash_{\Sigma_c^+} A_2 \Leftarrow \mathtt{Prf}(A_1)$ *is derived as a strict sub-derivation.*
- $A = \mathtt{allrE}\ A_3\ A_1\ A_2$ *and* $A' = (A_3\ A_1)$. *Moreover* $\Delta \vdash_{\Sigma_c^+} A_2 \Leftarrow \mathtt{Prf}((\mathtt{allr}\ A_3))$ *and* $\Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \mathtt{Db}$ *are derived as strict sub-derivations.*
- $A = \mathtt{saysI}\ A_2\ A_1\ A_3$ *and* $A' = \mathtt{says}\ A_1\ A_2$. *Moreover* $\Delta \vdash_{\Sigma_c^+} A_3 \Leftarrow A_1\ \mathtt{Affirms}\ A_2$ *is derived as a strict sub-derivation.*

2. *If* $\Delta \vdash_{\Sigma_c^+} A \Leftarrow A_1\ \mathtt{Affirms}\ A_2$ *and* $\mathbf{\Sigma} \gg \Sigma_c$ *and* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \mathtt{tctx} \gg\ \vdash_{\Sigma_c^+} \Delta\ \mathtt{ctx}$ *then one of the following holds:*
   - $A = \mathtt{truaff}\ A_2\ A_1\ A_3$, *and* $\Delta \vdash_{\Sigma_c^+} A_3 \Leftarrow \mathtt{Prf}(A_2)$ *and* $\Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \mathtt{Prin}$ *are derived as strict sub-derivations.*
   - $A = \mathtt{saysE}\ A_1'\ A_2\ A_1\ A_3\ \lambda\alpha{::}\mathtt{Prf}(A_1').A_4$ *and* $\Delta \vdash_{\Sigma_c^+} A_3 \Leftarrow \mathtt{Prf}(\mathtt{says}\ A_1\ A_1')$ *and* $\Delta \vdash_{\Sigma_c^+} A_4 \Leftarrow A_1\ \mathtt{Affirms}\ A_2$ *are derived as strict sub-derivations.*

Lemma D.6 (Context conversion). *1. Let* $\mathbf{\Sigma}$ *be a logic signature. Then there exists a unique* Σ_c *s.t.* $\mathbf{\Sigma} \gg \Sigma_c$.

2. *Let* $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \mathtt{ctx}$, *and* $\mathbf{\Sigma} \gg \Sigma_c$. *Then there exists a unique* Δ *s.t.* $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \mathtt{ctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}$.
3. *Let* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \mathtt{tctx}$, *and* $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \mathtt{ctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}$. *Then there exists* Δ' *s.t.* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \mathtt{tctx} \gg\ \vdash_{\Sigma_c} \Delta, \Delta'\ \mathtt{ctx}$. *Moreover* Δ' *is unique upto the choice of variable names.*

Proof. By induction on the derivations of context formation. □

Theorem D.7 (Adequacy of encoding). *1. (a) If* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{prop}$ *and* $\mathbf{\Sigma} \gg \Sigma_c$ *and* $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \mathtt{ctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}$*, then there exists a unique constructor* A *such that* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{prop} \gg \Delta \vdash_{\Sigma_c^+} A \Leftarrow \mathtt{Prop}$.

   (b) Let Σ_c *be a signature such that* $\mathbf{\Sigma} \gg \Sigma_c$. *If* $\Delta \vdash_{\Sigma_c^+} A \Leftarrow \mathtt{Prop}$ *and* $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \mathtt{ctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}$*, then there exists a unique* P *such that* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{prop}$.

2. *(a) If* $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{true}$ *and* $\mathbf{\Sigma} \gg \Sigma_c$ *and* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \mathtt{tctx} \gg\ \vdash_{\Sigma_c} \Delta\ \mathtt{ctx}$*, then there exist unique (upto* α*-equivalence) constructors* A_1 *and* A_2 *such that*
   i. $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{true} \gg \Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \mathtt{Prf}(A_2)$.
   ii. $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{prop} \gg \Delta \vdash_{\Sigma_c^+} A_2 \Leftarrow \mathtt{Prop}$.

   (b) If $\Delta_1, \Delta_2 \vdash_{\Sigma_c^+} A_1 \Leftarrow \mathtt{Prf}(A_2)$*, and* $\mathbf{\Sigma} \gg \Sigma_c$*, and* $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \mathtt{ctx} \gg\ \vdash_{\Sigma_c} \Delta_1\ \mathtt{ctx}$ *and* $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \mathtt{tctx} \gg\ \vdash_{\Sigma_c} \Delta_1, \Delta_2\ \mathtt{ctx}$*, then* $\exists P$ *s.t.*
   i. $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{true} \gg \Delta_1, \Delta_2 \vdash_{\Sigma_c^+} A_1 \Leftarrow \mathtt{Prf}(A_2)$,
   ii. $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{prop} \gg \Delta_1 \vdash_{\Sigma_c^+} A_2 \Leftarrow \mathtt{Prop}$.

Proof. 1. (a) The existence can be shown by induction on the derivation of $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{prop}$. Uniqueness can be established by induction using Lemma D.3.

(b) We use Lemma D.4 to establish the various constructors that A can be. For each of these options, Lemma D.3 establishes the unique proposition P.

2. (a) The lemma can be proven by induction on $\mathbf{\Delta}; \Phi \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \mathtt{true}$. For each of the inference rules in the logic, there is exactly one conversion rule that applies. Uniqueness upto α-equivalence follows from uniqueness of proposition, database, and, principal conversions, and from the I.H..

(b) By Lemma D.5, we get the various possible choices for A_1 and A_2. Here we illustrate the proof steps for a few choices:

Case: $\Delta = \Delta', \alpha{::}\texttt{Prf}(A_2)$ and $\Delta' \vdash_{\Sigma_c^+} A_2 \Leftarrow \texttt{Prop}$: By Lemma D.6, we get $\Delta' = \Delta_1, \Delta_2$ s.t. $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \texttt{ctx} \gg \vdash_{\Sigma_c} \Delta_1\ \texttt{ctx}$, and $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \Phi\ \texttt{tctx} \gg \vdash_{\Sigma_c^+} \Delta_1, \Delta_2\ \texttt{ctx}$.

Since $\Delta_1, \Delta_2|^{\texttt{Prop}}_{\preceq} = \Delta_1$, applying Lemma D.2, we get $\Delta_1 \vdash_{\Sigma_c^+} A_2 \Leftarrow \texttt{Prop}$. By adequacy for propositions, we obtain $\Delta_1 \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{prop} \gg \Delta_1 \vdash_{\Sigma_c^+} A_2 \Leftarrow \texttt{Prop}$, and applying Rule($\texttt{hyp} \gg$) we get $\mathbf{\Delta}; \Phi, P\ \texttt{true} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P\ \texttt{true} \gg \Delta_1, \Delta_2, \alpha{::}\texttt{Prf}(A_2) \vdash_{\Sigma_c^+} \alpha \Leftarrow \texttt{Prf}(A_2)$.

Case: $A_1 = \texttt{impI}\ A_1\ A_2\ \lambda\alpha{::}\texttt{Prf}(A_1)A.$ and $A_2 = A_1 \supset A_2$, and $\Delta \vdash_{\Sigma_c^+} A_1 \Leftarrow \texttt{Prop}$, and $\Delta, \alpha{::}\texttt{Prf}(A_1) \vdash_{\Sigma_c^+} A \Leftarrow \texttt{Prf}(A_2)$:

By a similar argument as above, we strengthen kinding of A_1 to be $\Delta_1 \vdash_{\Sigma_c^+} A_1 \Leftarrow \texttt{Prop}$ where $\vdash^{\mathcal{L}}_{\mathbf{\Sigma}} \mathbf{\Delta}\ \texttt{ctx} \gg \vdash_{\Sigma_c} \Delta_1\ \texttt{ctx}$. By adequacy for propositions, we get $\mathbf{\Delta} \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_1\ \texttt{prop} \gg \Delta_1 \vdash_{\Sigma_c^+} A_1 \Leftarrow \texttt{Prop}$. By I.H. we get $\mathbf{\Delta}; \Phi' \vdash^{\mathcal{L}}_{\mathbf{\Sigma}} P_2\ \texttt{true} \gg \Delta_1, \Delta_2, \alpha{::}\texttt{Prf}(A_1) \vdash_{\Sigma_c^+} A \Leftarrow \texttt{Prf}(A_2)$. By Lemma D.1 we get $\Phi' = \Phi, P_1\ \texttt{true}$. Rule ($\supset$ -I $\gg$) gives us the desired result now.

□