

Reasoning about the Consequences of Authorization Policies in a Linear Epistemic Logic

Henry DeYoung Frank Pfenning

Computer Science Department
Carnegie Mellon University

FCS Workshop 2009
August 10, 2009

Observation:

- Authorization policies are not stand-alone objects.
 - Permit actions that change a system's state.
 - Intended to allow only safe consequences.

Observation:

- Authorization policies are not stand-alone objects.
 - Permit actions that change a system's state.
 - Intended to allow only safe consequences.

Example:

Policy “A principal may read file F if F 's owner says so.”

Consequence “A principal may learn F 's contents if granted read access.”

Observation:

- Authorization policies are not stand-alone objects.
 - Permit actions that change a system's state.
 - Intended to allow only safe consequences.

Example:

Policy “A principal may read file F if F 's owner says so.”

Consequence “A principal may learn F 's contents if granted read access.”

Goal:

- Develop a general method for formally:
 - specifying both policies and their semantic consequences; and
 - reasoning about the interface between them.

- 1 Specify policies and consequences in a security linear logic.

- 1 Specify policies and consequences in a security linear logic.
- 2 Define a system-specific notion of state.

- 1 Specify policies and consequences in a security linear logic.
- 2 Define a system-specific notion of state.
- 3 Translate semantic specifications to rewrite rules via a rewriting interpretation of the logic.

Proposed Method

- 1 Specify policies and consequences in a security linear logic.
- 2 Define a system-specific notion of state.
- 3 Translate semantic specifications to rewrite rules via a rewriting interpretation of the logic.
- 4 Analyze the rewrite rules to prove properties of the system.

Running Example: A Simple File System

Simple file system with no directory structure and operations:

$$O^* ::= \text{create} \mid \text{onfile}(F, O)$$

Running Example: A Simple File System

Simple file system with no directory structure and operations:

$$O^* ::= \textit{create} \mid \textit{onfile}(F, O)$$

Running Example: A Simple File System

Simple file system with no directory structure and operations:

$$O^* ::= \text{create} \mid \text{onfile}(F, O)$$

Running Example: A Simple File System

Simple file system with no directory structure and operations:

$$O^* ::= \textit{create} \mid \textit{onfile}(F, O)$$
$$O ::= \textit{read} \mid \textit{write}(S) \mid \textit{delete}$$

Running Example: A Simple File System

Simple file system with no directory structure and operations:

$$\begin{aligned} O^* &::= \textit{create} \mid \textit{onfile}(F, O) \\ O &::= \textit{read} \mid \textit{write}(S) \mid \textit{delete} \end{aligned}$$

Files F are versioned with tags T .

- Writes create new versions.
- $\textit{current}(F, T)$: the current version of F is T .

- 1 Specify policies and consequences in a security linear logic.
- 2 Define a system-specific notion of state.
- 3 Translate semantic specifications to rewrite rules via a rewriting interpretation of the logic.
- 4 Analyze the rewrite rules to prove properties of the system.

To model mutable system state, use a linear logic [Girard87].

- Linear assumptions may be used only once

To model mutable system state, use a linear logic [Girard87].

- Linear assumptions may be used only once

For policies, borrow $\langle K \rangle A$ from [Garg+06].

- $\langle K \rangle A$: K says A
- Family of strong monads indexed by principals

To model mutable system state, use a linear logic [Girard87].

- Linear assumptions may be used only once

For policies, borrow $\langle K \rangle A$ from [Garg+06].

- $\langle K \rangle A$: K says A
- Family of strong monads indexed by principals

For semantics, also borrow $\llbracket K \rrbracket A$ and $[K]A$.

- $\llbracket K \rrbracket A$: K knows A
- $[K]A$: K possesses A
- Families of S4 \Box indexed by principals
- Possession is linear (consumable) knowledge

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- Any registered user may create files.

$\text{maycreate} : \langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- Any registered user may create files.

`maycreate` : $\langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- Any registered user may create files.

$\text{maycreate} : \langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- Any registered user may create files.

$\text{maycreate} : \langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- Any registered user may create files.

$$\text{maycreate} : \langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$$

- A principal may read, write, or delete a file if the owner says so.

$$\text{delegate} : \langle \text{fs} \rangle (\text{owns}(K, F) \otimes \langle K \rangle \text{may}(L, \text{onfile}(F, O)) \multimap \text{may}(L, \text{onfile}(F, O)))$$

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- Any registered user may create files.

$$\text{maycreate} : \langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$$

- A principal may read, write, or delete a file if the owner says so.

$$\text{delegate} : \langle \text{fs} \rangle (\text{owns}(K, F) \otimes \langle K \rangle \text{may}(L, \text{onfile}(F, O)) \multimap \text{may}(L, \text{onfile}(F, O)))$$

- Any registered user may create files.

$\text{maycreate} : \langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$

- A principal may read, write, or delete a file if the owner says so.

$\text{delegate} : \langle \text{fs} \rangle (\text{owns}(K, F) \otimes \langle K \rangle \text{may}(L, \text{onfile}(F, O)) \multimap \text{may}(L, \text{onfile}(F, O)))$

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- Any registered user may create files.

$$\text{maycreate} : \langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$$

- A principal may read, write, or delete a file if the owner says so.

$$\text{delegate} : \langle \text{fs} \rangle (\text{owns}(K, F) \otimes \langle K \rangle \text{may}(L, \text{onfile}(F, O)) \multimap \text{may}(L, \text{onfile}(F, O)))$$

Key to Syntax

$\langle K \rangle A = \text{"K says A"}$

- Any registered user may create files.

$\text{maycreate} : \langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$

- A principal may read, write, or delete a file if the owner says so.

$\text{delegate} : \langle \text{fs} \rangle (\text{owns}(K, F) \otimes \langle K \rangle \text{may}(L, \text{onfile}(F, O)) \multimap$
 $\text{may}(L, \text{onfile}(F, O)))$

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- Any registered user may create files.

$\text{maycreate} : \langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$

- A principal may read, write, or delete a file if the owner says so.

$\text{delegate} : \langle \text{fs} \rangle (\text{owns}(K, F) \otimes \langle K \rangle \text{may}(L, \text{onfile}(F, O)) \multimap \text{may}(L, \text{onfile}(F, O)))$

- Any registered user may create files.

$\text{maycreate} : \langle \text{fs} \rangle (\text{user}(K) \multimap \text{may}(K, \text{create}))$

- A principal may read, write, or delete a file if the owner says so.

$\text{delegate} : \langle \text{fs} \rangle (\text{owns}(K, F) \otimes \langle K \rangle \text{may}(L, \text{onfile}(F, O)) \multimap \text{may}(L, \text{onfile}(F, O)))$

Proving *authorization* should not be effectful.

- Proof-Carrying Authorization (PCA) [AppelFelten99]:
proving authorization and granting access are distinct phases

Specifying the Semantic Consequences of Creating a File

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$\llbracket K \rrbracket A = \text{"}K \text{ knows } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

$\text{create} : \langle K \rangle \text{do}(K, \text{create}) \otimes \langle \text{fs} \rangle \text{may}(K, \text{create}) \multimap$
 $\{ \exists f. \exists t. ! \langle \text{fs} \rangle \text{owns}(K, f) \otimes$
 $\quad [\text{fs}] \text{current}(f, t) \otimes$
 $\quad \llbracket \text{fs} \rrbracket \text{contents}(f, t, \epsilon) \otimes$
 $\quad \llbracket K \rrbracket \text{contents}(f, t, \epsilon) \}$

Specifying the Semantic Consequences of Creating a File

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$\llbracket K \rrbracket A = \text{"}K \text{ knows } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

create : $\langle K \rangle do(K, create) \otimes \langle fs \rangle may(K, create) \multimap$
 $\{ \exists f. \exists t. \neg \langle fs \rangle owns(K, f) \otimes$
 $[fs] current(f, t) \otimes$
 $\llbracket fs \rrbracket contents(f, t, \epsilon) \otimes$
 $\llbracket K \rrbracket contents(f, t, \epsilon) \}$

Specifying the Semantic Consequences of Creating a File

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$\llbracket K \rrbracket A = \text{"}K \text{ knows } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

$\text{create} : \langle K \rangle \text{do}(K, \text{create}) \otimes \langle \text{fs} \rangle \text{may}(K, \text{create}) \multimap$
 $\{ \exists f. \exists t. \langle \text{fs} \rangle \text{owns}(K, f) \otimes$
 $\quad [\text{fs}] \text{current}(f, t) \otimes$
 $\quad \llbracket \text{fs} \rrbracket \text{contents}(f, t, \epsilon) \otimes$
 $\quad \llbracket K \rrbracket \text{contents}(f, t, \epsilon) \}$

Specifying the Semantic Consequences of Creating a File

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$\llbracket K \rrbracket A = \text{"}K \text{ knows } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

$\text{create} : \langle K \rangle \text{do}(K, \text{create}) \otimes \langle \text{fs} \rangle \text{may}(K, \text{create}) \rightarrow$
 $\{ \exists f. \exists t. ! \langle \text{fs} \rangle \text{owns}(K, f) \otimes$
 $[\text{fs}] \text{current}(f, t) \otimes$
 $\llbracket \text{fs} \rrbracket \text{contents}(f, t, \epsilon) \otimes$
 $\llbracket K \rrbracket \text{contents}(f, t, \epsilon) \}$

Specifying the Semantic Consequences of Creating a File

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$\llbracket K \rrbracket A = \text{"}K \text{ knows } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

create : $\langle K \rangle do(K, create) \otimes \langle fs \rangle may(K, create) \multimap$
 $\{ \exists f. \exists t. \neg \langle fs \rangle owns(K, f) \otimes$
 $[fs] current(f, t) \otimes$
 $\llbracket fs \rrbracket contents(f, t, \epsilon) \otimes$
 $\llbracket K \rrbracket contents(f, t, \epsilon) \}$

Specifying the Semantic Consequences of Creating a File

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$\llbracket K \rrbracket A = \text{"}K \text{ knows } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

$\text{create} : \langle K \rangle \text{do}(K, \text{create}) \otimes \langle \text{fs} \rangle \text{may}(K, \text{create}) \multimap$
 $\{ \exists f. \exists t. \langle \text{fs} \rangle \text{owns}(K, f) \otimes$
 $\quad [\text{fs}] \text{current}(f, t) \otimes$
 $\quad \llbracket \text{fs} \rrbracket \text{contents}(f, t, \epsilon) \otimes$
 $\quad \llbracket K \rrbracket \text{contents}(f, t, \epsilon) \}$

Specifying the Semantic Consequences of Creating a File

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$\llbracket K \rrbracket A = \text{"}K \text{ knows } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

$\text{create} : \langle K \rangle \text{do}(K, \text{create}) \otimes \langle \text{fs} \rangle \text{may}(K, \text{create}) \multimap$
 $\{ \exists f. \exists t. ! \langle \text{fs} \rangle \text{owns}(K, f) \otimes$
 $\quad \text{[fs]current}(f, t) \otimes$
 $\quad \llbracket \text{fs} \rrbracket \text{contents}(f, t, \epsilon) \otimes$
 $\quad \llbracket K \rrbracket \text{contents}(f, t, \epsilon) \}$

Specifying the Semantic Consequences of Creating a File

Key to Syntax
$\langle K \rangle A =$ "K says A"
$\llbracket K \rrbracket A =$ "K knows A"
$[K]A =$ "K has A"

create : $\langle K \rangle do(K, create) \otimes \langle fs \rangle may(K, create) \multimap$
 $\{ \exists f. \exists t. ! \langle fs \rangle owns(K, f) \otimes$
 $[fs] current(f, t) \otimes$
 $\llbracket fs \rrbracket contents(f, t, \epsilon) \otimes$
 $\llbracket K \rrbracket contents(f, t, \epsilon) \}$

Specifying the Semantic Consequences of Creating a File

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$\llbracket K \rrbracket A = \text{"}K \text{ knows } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

$$\begin{aligned} \text{create} : & \langle K \rangle \text{do}(K, \text{create}) \otimes \langle \text{fs} \rangle \text{may}(K, \text{create}) \multimap \\ & \{ \exists f. \exists t. !\langle \text{fs} \rangle \text{owns}(K, f) \otimes \\ & \quad [\text{fs}] \text{current}(f, t) \otimes \\ & \quad \llbracket \text{fs} \rrbracket \text{contents}(f, t, \epsilon) \otimes \\ & \quad \llbracket K \rrbracket \text{contents}(f, t, \epsilon) \} \end{aligned}$$

We introduce an **effect monad** to isolate semantic effects from non-effectful authorization *decisions*.

- Stratification of policies from consequences is evident syntactically
 - Policies have top-level $\langle \cdot \rangle$ and do not contain $\{ \cdot \}$

Simulating the Environment with a Semantic Action

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- A principal can issue an arbitrary request at any time.

environment : $\{\langle K \rangle do(K, O^*)\}$

Simulating the Environment with a Semantic Action

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- A principal can issue an arbitrary request at any time.

environment : $\{\langle K \rangle do(K, O^*)\}$

Simulating the Environment with a Semantic Action

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- A principal can issue an arbitrary request at any time.

environment : $\{\langle K \rangle do(K, O^*)\}$

Simulating the Environment with a Semantic Action

Key to Syntax

$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$

- A principal can issue an arbitrary request at any time.

environment : $\{\langle K \rangle do(K, O^*)\}$

We would need a more detailed model to reason about *sequences* of requests.

- But the weak model is sufficient and actually beneficial: properties demonstrate security *regardless* of the sequence.

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \rightarrow$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

delete : $\langle K \rangle do(K, onfile(F, delete)) \otimes$
 $\langle fs \rangle may(K, onfile(F, delete)) \otimes [fs]current(F, T) \multimap$
 $\{\mathbf{1}\}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$\llbracket K \rrbracket A = \text{"}K \text{ knows } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

delete : $\langle K \rangle do(K, onfile(F, delete)) \otimes$
 $\langle fs \rangle may(K, onfile(F, delete)) \otimes [fs]current(F, T) \multimap$
 $\{ \mathbf{1} \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K]A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

delete : $\langle K \rangle do(K, onfile(F, delete)) \otimes$
 $\langle fs \rangle may(K, onfile(F, delete)) \otimes [fs]current(F, T) \multimap$
 $\{\mathbf{1}\}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K] A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

delete : $\langle K \rangle do(K, onfile(F, delete)) \otimes$
 $\langle fs \rangle may(K, onfile(F, delete)) \otimes [fs]current(F, T) \multimap$
 $\{ \mathbf{1} \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K] A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

delete : $\langle K \rangle do(K, onfile(F, delete)) \otimes$
 $\langle fs \rangle may(K, onfile(F, delete)) \otimes [fs]current(F, T) \multimap$
 $\{ \mathbf{1} \}$

Semantic Actions for Reading and Deleting Files

Key to Syntax
$\langle K \rangle A = \text{"K says A"}$
$\llbracket K \rrbracket A = \text{"K knows A"}$
$[K] A = \text{"K has A"}$

read : $\langle K \rangle do(K, onfile(F, read)) \otimes \langle fs \rangle may(K, onfile(F, read)) \otimes$
 $[fs]current(F, T) \otimes \llbracket fs \rrbracket contents(F, T, S) \multimap$
 $\{ \llbracket K \rrbracket contents(F, T, S) \otimes$
 $[fs]current(F, T) \}$

delete : $\langle K \rangle do(K, onfile(F, delete)) \otimes$
 $\langle fs \rangle may(K, onfile(F, delete)) \otimes [fs]current(F, T) \multimap$
 $\{\mathbf{1}\}$

Proposed Method

- 1 Specify policies and consequences in a security linear logic.
- 2 Define a system-specific notion of state.
- 3 Translate semantic specifications to rewrite rules via a rewriting interpretation of the logic.
- 4 Analyze the rewrite rules to prove properties of the system.

We wish to translate the semantic specifications to state rewriting rules (similar to MSR [Cervesato+]).

We wish to translate the semantic specifications to state rewriting rules (similar to MSR [Cervesato+]).

A state is described by judgments about it that are evident to:

- us (e.g., truths); and
- principals (e.g., knowledge).

General Form of States

We wish to translate the semantic specifications to state rewriting rules (similar to MSR [Cervesato+]).

A state is described by judgments about it that are evident to:

- us (e.g., truths); and
- principals (e.g., knowledge).

States are pairs $\Gamma; \Delta$.

- Γ contains persistent judgments:
 - currently evident, and evident in all future states
- Δ contains linear judgments:
 - currently evident, but might fail to be evident in future states

Definition (File System State)

$\Gamma; \Delta$ is a *file system state* if and only if:

Definition (File System State)

$\Gamma; \Delta$ is a *file system state* if and only if:

- 1 Each persistent judgment in Γ is either:
 - a policy or a semantic action
 - fs knows *contents*(F, T, S) or K knows *contents*(F, T, S)
 - $\langle \text{fs} \rangle \text{user}(K)$
 - $\langle \text{fs} \rangle \text{owns}(K, F)$

Definition (File System State)

$\Gamma; \Delta$ is a *file system state* if and only if:

- 1 Each persistent judgment in Γ is either:
 - a policy or a semantic action
 - fs knows *contents*(F, T, S) or K knows *contents*(F, T, S)
 - $\langle \text{fs} \rangle \text{user}(K)$
 - $\langle \text{fs} \rangle \text{owns}(K, F)$
- 2 Each linear judgment in Δ is either:
 - fs has *current*(F, T)
 - $\langle K \rangle \text{do}(K, O^*)$
 - $\langle K \rangle \text{may}(L, O^*)$

Definition (File System State)

$\Gamma; \Delta$ is a *file system state* if and only if:

- 1 Each persistent judgment in Γ is either:
 - a policy or a semantic action
 - fs knows *contents*(F, T, S) or K knows *contents*(F, T, S)
 - $\langle \text{fs} \rangle \text{user}(K)$
 - $\langle \text{fs} \rangle \text{owns}(K, F)$
- 2 Each linear judgment in Δ is either:
 - fs has *current*(F, T)
 - $\langle K \rangle \text{do}(K, O^*)$
 - $\langle K \rangle \text{may}(L, O^*)$
- 3 For each F , there is at most one T such that fs has *current*(F, T) $\in \Delta$.

Proposed Method

- 1 Specify policies and consequences in a security linear logic.
- 2 Define a system-specific notion of state.
- 3 Translate semantic specifications to rewrite rules via a rewriting interpretation of the logic.
- 4 Analyze the rewrite rules to prove properties of the system.

Translating Specifications to Rewrite Steps

We want to characterize possible rewrite steps on states using the specifications.

The step $\Gamma; \Delta \rightarrow \Gamma'; \Delta'$ can occur iff a special form of partial derivation, relating $\Gamma; \Delta$ and $\Gamma'; \Delta'$, exists.

- Ensures that steps have logical origins
- Structured so that steps arise from effectful state judgments (i.e., semantic actions)
- Focusing [Andreoli92] ensures that specs translate to atomic steps.

Rewrite Steps \cong Semantic Actions

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

Specification:

delete : $\langle K \rangle do(K, onfile(F, delete)) \otimes$
 $\langle fs \rangle may(K, onfile(F, delete)) \otimes [fs]current(F, T) \multimap$
 $\{1\}$

Rewrite Steps \cong Semantic Actions

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

Specification:

$$\text{delete} : \langle K \rangle do(K, \text{onfile}(F, \text{delete})) \otimes \\ \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete})) \otimes [\text{fs}] \text{current}(F, T) \multimap \\ \{\mathbf{1}\}$$

Rewrite step:

$\Gamma; \langle K \rangle do(K, \text{onfile}(F, \text{delete})), \Delta_1, [\text{fs}] \text{current}(F, T), \Delta_2 \rightarrow \Gamma; \Delta_2$ such that $\Gamma; \Delta_1 \vdash \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete}))$.

Rewrite Steps \cong Semantic Actions

Key to Syntax
$\langle K \rangle A =$ “ K says A ”
$[K]A =$ “ K has A ”

Specification:

$$\text{delete} : \langle K \rangle \text{do}(K, \text{onfile}(F, \text{delete})) \otimes$$
$$\langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete})) \otimes [\text{fs}] \text{current}(F, T) \multimap$$
$$\{\mathbf{1}\}$$

Rewrite step:

$\Gamma; \langle K \rangle \text{do}(K, \text{onfile}(F, \text{delete})), \Delta_1, [\text{fs}] \text{current}(F, T), \Delta_2 \rightarrow \Gamma; \Delta_2$ such that $\Gamma; \Delta_1 \vdash \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete}))$.

Rewrite Steps \cong Semantic Actions

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

Specification:

$$\text{delete} : \langle K \rangle \text{do}(K, \text{onfile}(F, \text{delete})) \otimes \\ \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete})) \otimes [\text{fs}] \text{current}(F, T) \multimap \\ \{1\}$$

Rewrite step:

$\Gamma; \langle K \rangle \text{do}(K, \text{onfile}(F, \text{delete})), \Delta_1, [\text{fs}] \text{current}(F, T), \Delta_2 \rightarrow \Gamma; \Delta_2$ such that $\Gamma; \Delta_1 \vdash \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete}))$.

Rewrite Steps \cong Semantic Actions

Key to Syntax
$\langle K \rangle A =$ “ K says A ”
$[K]A =$ “ K has A ”

Specification:

$$\text{delete} : \langle K \rangle do(K, \text{onfile}(F, \text{delete})) \otimes \\ \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete})) \otimes [\text{fs}] \text{current}(F, T) \multimap \\ \{\mathbf{1}\}$$

Rewrite step:

$\Gamma; \langle K \rangle do(K, \text{onfile}(F, \text{delete})), \Delta_1, [\text{fs}] \text{current}(F, T), \Delta_2 \rightarrow \Gamma; \Delta_2$ such that $\Gamma; \Delta_1 \vdash \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete}))$.

Rewrite Steps \cong Semantic Actions

Key to Syntax
$\langle K \rangle A = \text{"}K \text{ says } A\text{"}$
$[K]A = \text{"}K \text{ has } A\text{"}$

Specification:

$$\text{delete} : \langle K \rangle do(K, \text{onfile}(F, \text{delete})) \otimes \\ \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete})) \otimes [\text{fs}] \text{current}(F, T) \multimap \\ \{\mathbf{1}\}$$

Rewrite step:

$\Gamma; \langle K \rangle do(K, \text{onfile}(F, \text{delete})), \Delta_1, [\text{fs}] \text{current}(F, T), \Delta_2 \rightarrow \Gamma; \Delta_2$ such that $\Gamma; \Delta_1 \vdash \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete}))$.

Rewrite Steps \cong Semantic Actions

Key to Syntax
$\langle K \rangle A =$ “ K says A ”
$[K]A =$ “ K has A ”

Specification:

$$\text{delete} : \langle K \rangle \text{do}(K, \text{onfile}(F, \text{delete})) \otimes \\ \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete})) \otimes [\text{fs}] \text{current}(F, T) \multimap \\ \{1\}$$

Rewrite step:

$\Gamma; \langle K \rangle \text{do}(K, \text{onfile}(F, \text{delete})), \Delta_1, [\text{fs}] \text{current}(F, T), \Delta_2 \rightarrow \Gamma; \Delta_2$ such that $\Gamma; \Delta_1 \vdash \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete}))$.

Theorem (Rewrite Step Schemata)

Each rewrite step from a file system state is one of:

- $\Gamma; \langle K \rangle \text{do}(K, \text{onfile}(F, \text{delete})), \Delta_1, [\text{fs}] \text{current}(F, T), \Delta_2 \rightarrow \Gamma; \Delta_2$ such that $\Gamma; \Delta_1 \vdash \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete}))$.
- Similar rewrite steps for create, read, write and environment elided.*

Proposed Method

- 1 Specify policies and consequences in a security linear logic.
- 2 Define a system-specific notion of state.
- 3 Translate semantic specifications to rewrite rules via a rewriting interpretation of the logic.
- 4 Analyze the rewrite rules to prove properties of the system.

Theorem (Knowledge Safety)

If $\Gamma; \Delta$ is a file system state such that

$$\Gamma; \Delta \rightarrow \Gamma', \llbracket K \rrbracket \text{contents}(F, T, S); \Delta'$$

then either $\llbracket K \rrbracket \text{contents}(F, T, S) \in \Gamma$ or the step was a create, read, or write step for F triggered by K and permitted by the policy (as evidenced by the $\langle \text{fs} \rangle \text{may}(K, O^)$ derivation of the step).*

Proof.

By case analysis of the possible rewrite step schemata. □

Theorem (Knowledge Safety)

If $\Gamma; \Delta$ is a file system state such that

$$\Gamma; \Delta \rightarrow \Gamma', \llbracket K \rrbracket \text{contents}(F, T, S); \Delta'$$

then either $\llbracket K \rrbracket \text{contents}(F, T, S) \in \Gamma$ or the step was a create, read, or write step for F triggered by K and permitted by the policy (as evidenced by the $\langle \text{fs} \rangle \text{may}(K, O^*)$ derivation of the step).

Proof.

By case analysis of the possible rewrite step schemata. □

Theorem (Knowledge Safety)

If $\Gamma; \Delta$ is a file system state such that

$$\Gamma; \Delta \rightarrow \Gamma', \llbracket K \rrbracket \text{contents}(F, T, S); \Delta'$$

then either $\llbracket K \rrbracket \text{contents}(F, T, S) \in \Gamma$ or the step was a create, read, or write step for F triggered by K and permitted by the policy (as evidenced by the $\langle fs \rangle \text{may}(K, O^*)$ derivation of the step).

Proof.

By case analysis of the possible rewrite step schemata. □

Theorem (Knowledge Safety)

If $\Gamma; \Delta$ is a file system state such that

$$\Gamma; \Delta \rightarrow \Gamma', \llbracket K \rrbracket \text{contents}(F, T, S); \Delta'$$

then either $\llbracket K \rrbracket \text{contents}(F, T, S) \in \Gamma$ or the step was a create, read, or write step for F triggered by K and permitted by the policy (as evidenced by the $\langle \text{fs} \rangle \text{may}(K, O^)$ derivation of the step).*

Proof.

By case analysis of the possible rewrite step schemata. □

Principals do not learn file contents unless permitted by the policy!

Corollary

If

- $\Gamma; \langle K \rangle do(K, onfile(F, delete)), \Delta_1, [fs]current(F, T), \Delta_2$
- $\rightarrow \Gamma; \Delta_2$
- $\rightarrow^* \Gamma', \llbracket L \rrbracket contents(F, T', S); \Delta'$

such that $\Gamma; \Delta_1 \vdash \langle fs \rangle may(K, onfile(F, delete))$, then $\llbracket L \rrbracket contents(F, T', S) \in \Gamma$.

Corollary

If

$$\begin{aligned} & \Gamma; \langle K \rangle \text{do}(K, \text{onfile}(F, \text{delete})), \Delta_1, [\text{fs}] \text{current}(F, T), \Delta_2 \\ \rightarrow & \Gamma; \Delta_2 \\ \rightarrow^* & \Gamma', \llbracket L \rrbracket \text{contents}(F, T', S); \Delta' \end{aligned}$$

such that $\Gamma; \Delta_1 \vdash \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete}))$, then
 $\llbracket L \rrbracket \text{contents}(F, T', S) \in \Gamma$.

Corollary

If

- $\Gamma; \langle K \rangle do(K, onfile(F, delete)), \Delta_1, [fs]current(F, T), \Delta_2$
- $\rightarrow \Gamma; \Delta_2$
- $\rightarrow^* \Gamma', \llbracket L \rrbracket contents(F, T', S); \Delta'$

such that $\Gamma; \Delta_1 \vdash \langle fs \rangle may(K, onfile(F, delete))$, then $\llbracket L \rrbracket contents(F, T', S) \in \Gamma$.

Corollary

If

- $\Gamma; \langle K \rangle \text{do}(K, \text{onfile}(F, \text{delete})), \Delta_1, [\text{fs}] \text{current}(F, T), \Delta_2$
- $\rightarrow \Gamma; \Delta_2$
- $\rightarrow^* \Gamma', \llbracket L \rrbracket \text{contents}(F, T', S); \Delta'$

such that $\Gamma; \Delta_1 \vdash \langle \text{fs} \rangle \text{may}(K, \text{onfile}(F, \text{delete}))$, *then*
 $\llbracket L \rrbracket \text{contents}(F, T', S) \in \Gamma$.

Corollary

If

$$\begin{aligned} & \Gamma; \langle K \rangle do(K, onfile(F, delete)), \Delta_1, [fs]current(F, T), \Delta_2 \\ \rightarrow & \Gamma; \Delta_2 \\ \rightarrow^* & \Gamma', \llbracket L \rrbracket contents(F, T', S); \Delta' \end{aligned}$$

such that $\Gamma; \Delta_1 \vdash \langle fs \rangle may(K, onfile(F, delete))$, *then*
 $\llbracket L \rrbracket contents(F, T', S) \in \Gamma$.

Principals cannot learn the contents of deleted files!

Theorem (Delete Safety)

If $\Gamma; \Delta, [\text{fs}]_{\text{current}}(F, T)$ is a file system state such that

$$\Gamma; \Delta, [\text{fs}]_{\text{current}}(F, T) \rightarrow \Gamma'; \Delta'$$

and there is no T' such that $[\text{fs}]_{\text{current}}(F, T') \in \Delta'$, then the step was a delete.

Theorem (Delete Safety)

If $\Gamma; \Delta, [\text{fs}]_{\text{current}}(F, T)$ is a file system state such that

$$\Gamma; \Delta, [\text{fs}]_{\text{current}}(F, T) \rightarrow \Gamma'; \Delta'$$

and there is no T' such that $[\text{fs}]_{\text{current}}(F, T') \in \Delta'$, then the step was a delete.

Theorem (Delete Safety)

If $\Gamma; \Delta, [\text{fs}]_{\text{current}}(F, T)$ is a file system state such that

$$\Gamma; \Delta, [\text{fs}]_{\text{current}}(F, T) \rightarrow \Gamma'; \Delta'$$

and there is no T' such that $[\text{fs}]_{\text{current}}(F, T') \in \Delta'$, then the step was a delete.

Files disappear only if explicitly deleted (and permitted by policy as evidenced by the $\langle \text{fs} \rangle_{\text{may}}(K, \text{onfile}(F, \text{delete}))$ derivation)!

Stratification is Necessary!

Recall the rewrite steps each have a condition:

$$\Gamma; \Delta_1 \vdash \langle \text{fs} \rangle \text{may}(K, O^*)$$

In principle, $[\text{fs}] \text{current}(F, T)$ might be in Δ_1 .

Stratification is Necessary!

Recall the rewrite steps each have a condition:

$$\Gamma; \Delta_1 \vdash \langle fs \rangle may(K, O^*)$$

In principle, $[fs]current(F, T)$ might be in Δ_1 .

Impossible since policies and semantic effects are stratified!

- Effects are confined to the effect monad.
- With stratification, policies do not contain the effect monad.
- Thus, stratification ensures that authorization decisions do not depend on semantic effects.

Summary:

- 1 Specify policies and consequences in a security linear logic.
- 2 Define a system-specific notion of state.
- 3 Translate semantic specifications to rewrite rules via a rewriting interpretation of the logic.
- 4 Analyze the rewrite rules to prove system properties.

Stratification of policies from semantic effects is crucial!

Summary:

- 1 Specify policies and consequences in a security linear logic.
- 2 Define a system-specific notion of state.
- 3 Translate semantic specifications to rewrite rules via a rewriting interpretation of the logic.
- 4 Analyze the rewrite rules to prove system properties.

Stratification of policies from semantic effects is crucial!

Future Work:

- Obligations as semantic effects
- Real-world case studies
- Dynamic logic for mechanically verifying properties
- Compilation of semantic actions to executable code

Thank You!

I'm happy to answer questions:
hdeyoung@cs.cmu.edu



Weakly Focused Linear Logic

Polarized propositions:

$$\begin{aligned} A^+, B^+ & ::= \rho^+ \mid A^+ \otimes B^+ \mid \exists x:\tau. A^+ \mid \mathbf{1} \mid !A^- \\ & \quad \mid \llbracket K \rrbracket A^- \mid [K]A^- \mid A^- \\ A^-, B^- & ::= \rho^- \mid A^+ \multimap B^- \mid \forall x:\tau. A^- \mid \{A^+\} \mid \langle K \rangle A^+ \end{aligned}$$

Three forms of sequent:

$$\begin{array}{ll} \text{Neutral} & \Gamma; \Delta \vdash J \\ \text{Right Focusing} & \Gamma; \Delta \vdash [A^+] \\ \text{Left Focusing} & \Gamma; \Delta, [A^-] \vdash J \end{array}$$

$$\begin{array}{ll} \text{Unrestricted Hyps} & \Gamma ::= \cdot \mid \Gamma, A^- \text{ valid} \mid \Gamma, K \text{ knows } A^- \\ \text{Linear Hyps} & \Delta ::= \cdot \mid \Delta, A^+ \text{ hyp} \mid \Delta, K \text{ has } A^- \\ \text{Consequents} & J ::= A^- \text{ true} \mid A^+ \text{ lax} \mid K \text{ affirms } A^+ \end{array}$$

$$\overline{\Gamma; [p^-] \vdash p^-} \text{ atom}^-$$

$$\overline{\Gamma; [p^-] \vdash p^-} \text{ atom}^-$$

$$\frac{\Gamma; \Delta_1 \vdash [A^+] \quad \Gamma; \Delta_2, [B^-] \vdash J}{\Gamma; \Delta_1, \Delta_2, [A^+ \multimap B^-] \vdash J} \multimap_L$$

Selected Rules of Weakly Focused Linear Logic

$$\overline{\Gamma; [p^-] \vdash p^-} \text{ atom}^-$$

$$\frac{\Gamma; \Delta_1 \vdash [A^+] \quad \Gamma; \Delta_2, [B^-] \vdash J}{\Gamma; \Delta_1, \Delta_2, [A^+ \multimap B^-] \vdash J} \multimap_L$$

$$\frac{\Gamma; \Delta, A^+ \vdash C^+ \text{ lax}}{\Gamma; \Delta, [\{A^+\}] \vdash C^+ \text{ lax}} \{\}_L$$

- All principals affirm true statements.

$$\frac{\Gamma; \Delta \vdash [A^+]}{\Gamma; \Delta \vdash K \text{ affirms } A^+} \text{ affirms}_R$$

- All principals affirm true statements.

$$\frac{\Gamma; \Delta \vdash [A^+]}{\Gamma; \Delta \vdash K \text{ affirms } A^+} \text{ affirms}_R$$

- Internalize $K \text{ affirms } A^+$ as the proposition $\langle K \rangle A^+$.

$$\frac{\Gamma; \Delta \vdash K \text{ affirms } A^+}{\Gamma; \Delta \vdash \langle K \rangle A^+} \langle \rangle_R$$

Affirmation

- All principals affirm true statements.

$$\frac{\Gamma; \Delta \vdash [A^+]}{\Gamma; \Delta \vdash K \text{ affirms } A^+} \text{affirms}_R$$

- Internalize $K \text{ affirms } A^+$ as the proposition $\langle K \rangle A^+$.

$$\frac{\Gamma; \Delta \vdash K \text{ affirms } A^+}{\Gamma; \Delta \vdash \langle K \rangle A^+} \langle \rangle_R$$

- Principals trust their own affirmations to be true.

$$\frac{\Gamma; \Delta, A^+ \vdash K \text{ affirms } C^+}{\Gamma; \Delta, [\langle K \rangle A^+] \vdash K \text{ affirms } C^+} \langle \rangle_L$$

- If K knows a fact, that fact will be provably true.

$$\frac{\Gamma, K \text{ knows } A^-; \Delta, [A^-] \vdash J}{\Gamma, K \text{ knows } A^-; \Delta \vdash J} \text{ knows}_L$$

- If K knows a fact, that fact will be provably true.

$$\frac{\Gamma, K \text{ knows } A^-; \Delta, [A^-] \vdash J}{\Gamma, K \text{ knows } A^-; \Delta \vdash J} \text{ knows}_L$$

- Internalize $K \text{ knows } A^-$ as the proposition $\llbracket K \rrbracket A^-$.

$$\frac{\Gamma, K \text{ knows } A^-; \Delta \vdash J}{\Gamma; \Delta, \llbracket K \rrbracket A^- \vdash J} \llbracket \rrbracket_L$$

- Knowledge should be:

Reusable: empty linear context

Private: restriction operator $|_K$

$$\frac{\Gamma|_K; \cdot \vdash A^-}{\Gamma; \cdot \vdash \llbracket K \rrbracket A^-} \quad \boxed{R}$$

$$\begin{aligned}(\cdot)|_K &= \cdot \\(\Gamma, A^- \text{ valid})|_K &= \Gamma|_K \\(\Gamma, K \text{ knows } A^-)|_K &= \Gamma|_K, K \text{ knows } A^- \\(\Gamma, L \text{ knows } A^-)|_K &= \Gamma|_K \text{ if } L \neq K\end{aligned}$$

- Knowledge should be:

Reusable: empty linear context

Private: restriction operator $|_K$

$$\frac{\Gamma|_K; \cdot \vdash A^-}{\Gamma; \cdot \vdash \llbracket K \rrbracket A^-} \quad \boxed{R}$$

$$\begin{aligned}(\cdot)|_K &= \cdot \\(\Gamma, A^- \text{ valid})|_K &= \Gamma|_K \\(\Gamma, K \text{ knows } A^-)|_K &= \Gamma|_K, K \text{ knows } A^- \\(\Gamma, L \text{ knows } A^-)|_K &= \Gamma|_K \text{ if } L \neq K\end{aligned}$$

- Knowledge should be:

Reusable: empty linear context

Private: restriction operator $|_K$

$$\frac{\Gamma|_K; \cdot \vdash A^-}{\Gamma; \cdot \vdash \llbracket K \rrbracket A^-} \text{R}$$

$$\begin{aligned}(\cdot)|_K &= \cdot \\(\Gamma, A^- \text{ valid})|_K &= \Gamma|_K \\(\Gamma, K \text{ knows } A^-)|_K &= \Gamma|_K, K \text{ knows } A^- \\(\Gamma, L \text{ knows } A^-)|_K &= \Gamma|_K \text{ if } L \neq K\end{aligned}$$

$$\frac{\Gamma; \Delta, [A^-] \vdash J}{\Gamma; \Delta, K \text{ has } A^- \vdash J} \text{has}_L$$

$$\frac{\Gamma; \Delta, [A^-] \vdash J}{\Gamma; \Delta, K \text{ has } A^- \vdash J} \text{has}_L$$

$$\frac{\Gamma; \Delta, K \text{ has } A^- \vdash J}{\Gamma; \Delta, [K]A^- \vdash J} \square_L$$

$$\frac{\Gamma; \Delta, [A^-] \vdash J}{\Gamma; \Delta, K \text{ has } A^- \vdash J} \text{has}_L$$

$$\frac{\Gamma; \Delta, K \text{ has } A^- \vdash J}{\Gamma; \Delta, [K]A^- \vdash J} \boxed{L}$$

$$\frac{\Gamma|_K; \Delta|_K \vdash A^-}{\Gamma; \Delta|_K \vdash [[K]A^-]} \boxed{R}$$

$$\begin{aligned} (\cdot)|_K &= \cdot \\ (\Delta, A^+ \text{ hyp})|_K &= \Delta|_K \\ (\Delta, K \text{ has } A^-)|_K &= \Delta|_K, K \text{ has } A^- \\ (\Delta, L \text{ has } A^-)|_K &= \Delta|_K \text{ if } L \neq K \end{aligned}$$

Theorem (Admissibility of Cut)

- 1 If $\Gamma; \Delta \vdash A^-$ and $\Gamma; \Delta', [A^-] \vdash J$, then $\Gamma; \Delta', \Delta \vdash J$.
- 2 If $\Gamma; \Delta \vdash [A^+]$ and $\Gamma; \Delta', A^+ \vdash J$, then $\Gamma; \Delta', \Delta \vdash J$.

Theorem (Admissibility of Cut)

- 1 If $\Gamma; \Delta \vdash A^-$ and $\Gamma; \Delta', [A^-] \vdash J$, then $\Gamma; \Delta', \Delta \vdash J$.
- 2 If $\Gamma; \Delta \vdash [A^+]$ and $\Gamma; \Delta', A^+ \vdash J$, then $\Gamma; \Delta', \Delta \vdash J$.

Theorem (Identity)

- 1 For all Γ and A^- , $\Gamma; A^- \vdash A^-$.
- 2 For all Γ and A^+ , $\Gamma; A^+ \vdash A^+ \text{ lax}$.

Theorem (Admissibility of Cut)

- 1 If $\Gamma; \Delta \vdash A^-$ and $\Gamma; \Delta', [A^-] \vdash J$, then $\Gamma; \Delta', \Delta \vdash J$.
- 2 If $\Gamma; \Delta \vdash [A^+]$ and $\Gamma; \Delta', A^+ \vdash J$, then $\Gamma; \Delta', \Delta \vdash J$.

Theorem (Identity)

- 1 For all Γ and A^- , $\Gamma; A^- \vdash A^-$.
- 2 For all Γ and A^+ , $\Gamma; A^+ \vdash A^+$ lax.

- Ensure that our sequent forms respect the defining properties of a hypothetical judgment.
- Can be useful in reasoning about policies and their semantics.

Theorem (Admissibility of Cut)

- 1 If $\Gamma; \Delta \vdash A^-$ and $\Gamma; \Delta', [A^-] \vdash J$, then $\Gamma; \Delta', \Delta \vdash J$.
- 2 If $\Gamma; \Delta \vdash [A^+]$ and $\Gamma; \Delta', A^+ \vdash J$, then $\Gamma; \Delta', \Delta \vdash J$.
- 3 If $\Gamma; \cdot \vdash A^-$ and $\Gamma, A^-; \Delta' \vdash J$, then $\Gamma; \Delta' \vdash J$.
- 4 If $\Gamma; \Delta \vdash A^+ \text{ lax}$ and $\Gamma; \Delta', A^+ \vdash C^+ \text{ lax}$, then $\Gamma; \Delta', \Delta \vdash C^+ \text{ lax}$.
- 5 If $\Gamma; \Delta \vdash K \text{ affirms } A^+$ and $\Gamma; \Delta', A^+ \vdash K \text{ affirms } C^+$, then $\Gamma; \Delta', \Delta \vdash K \text{ affirms } C^+$.
- 6 If $\Gamma|_K; \cdot \vdash A^-$ and $\Gamma, K \text{ knows } A^-; \Delta' \vdash J$, then $\Gamma; \Delta' \vdash J$.
- 7 If $\Gamma|_K; \Delta|_K \vdash A^-$ and $\Gamma; \Delta', K \text{ has } A^- \vdash J$, then $\Gamma; \Delta', \Delta|_K \vdash J$.

- Ensures that our sequent forms respect the substitution property of a hypothetical judgment.

Theorem (η Expansion)

- 1** *If, by assuming a derivation of $\Gamma, \Gamma'; \Delta' \vdash [A^+]$, one can derive $\Gamma, \Gamma'; \Delta, \Delta' \vdash J$ parametrically in Γ' and Δ' , then $\Gamma; \Delta, A^+ \vdash J$.*
 - 2** *If, by assuming a derivation of $\Gamma, \Gamma'; \Delta', [A^-] \vdash J$, one can derive $\Gamma, \Gamma'; \Delta', \Delta \vdash J$ parametrically in Γ' , Δ' , and J , then $\Gamma; \Delta \vdash A^-$.*
- Ensures that our sequent forms respect the identity property of a hypothetical judgment.