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

Motivation

Observation:

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

Motivation

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."

Motivation

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.

Proposed Method

1. Specify policies and semantics in a security linear logic.

Proposed Method

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

Proposed Method

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

Proposed Method

1. Specify policies and semantics in a security linear logic.
2. Define a system-specific notion of state.
3. Interpret semantic specifications as 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:

$$O^* \quad ::= \quad \mathit{insert} \mid \mathit{onfile}(F, O)$$

Running Example: A Simple File System

Simple file system with no directory structure and operations:

$$O^* \quad ::= \quad \mathit{insert} \mid \mathit{onfile}(F, O)$$

Running Example: A Simple File System

Simple file system with no directory structure and operations:

$$O^* \quad ::= \quad \mathit{insert} \mid \mathit{onfile}(F, O)$$

Running Example: A Simple File System

Simple file system with no directory structure and operations:

$$\begin{array}{rcl} O^* & ::= & \mathit{insert} \mid \mathit{onfile}(F, O) \\ O & ::= & \mathit{read} \mid \mathit{write}(S) \mid \mathit{delete} \end{array}$$

Running Example: A Simple File System

Simple file system with no directory structure and operations:

$$\begin{array}{rcl} O^* & ::= & \mathit{insert} \mid \mathit{onfile}(F, O) \\ O & ::= & \mathit{read} \mid \mathit{write}(S) \mid \mathit{delete} \end{array}$$

Files F are versioned with tags T.

- Writes create new versions.

Running Example: A Simple File System

Simple file system with no directory structure and operations:

$$\begin{array}{rcl} O^* & ::= & \mathit{insert} \mid \mathit{onfile}(F, O) \\ O & ::= & \mathit{read} \mid \mathit{write}(S) \mid \mathit{delete} \end{array}$$

Files F are versioned with tags T.

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

Proposed Method

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

A Linear Logic of Authorization and Knowledge

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

- Linear assumptions may be used only once

A Linear Logic of Authorization and Knowledge

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

- Linear assumptions may be used only once

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

- Family of strong monads indexed by principals

A Linear Logic of Authorization and Knowledge

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

- Linear assumptions may be used only once

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

- Family of strong monads indexed by principals

For knowledge and possession, also borrow $[\![K]\!]$ and $[K]$.

- Families of S4 $\Box$ indexed by principals
- Possession is linear (consumable) knowledge

A Linear Logic of Authorization and Knowledge

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

- Linear assumptions may be used only once

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

- Family of strong monads indexed by principals

For knowledge and possession, also borrow $[\![K]\!]$ and $[K]$.

- Families of S4 $\Box$ indexed by principals
- Possession is linear (consumable) knowledge

Modalities provide logical force to these concepts; atoms cannot.

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathsf{fs} \rangle (\mathit{user}^{-}(K) \multimap \mathit{may}^{-}(K, \mathit{insert}))$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathsf{fs} \rangle (\mathit{user}^{-}(K) \multimap \mathit{may}^{-}(K, \mathit{insert}))$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathsf{fs} \rangle (\mathit{user}^{-}(K) \multimap \mathit{may}^{-}(K, \mathit{insert}))$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathsf{fs} \rangle (\mathit{user}^{-}(K) \multimap \mathit{may}^{-}(K, \mathit{insert}))$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathrm{fs} \rangle (\mathit{user}^-(K) \multimap \mathit{may}^-(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \multimap \mathit{may}^-(K, \mathit{onfile}(F, O)))$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathsf{fs} \rangle (\mathit{user}^-(K) \multimap \mathit{may}^-(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathsf{fs} \rangle (\mathit{owns}^-(K, F) \multimap \mathit{may}^-(K, \mathit{onfile}(F, O)))$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathrm{fs} \rangle (\mathit{user}^-(K) \multimap \mathit{may}^-(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \multimap \mathit{may}^-(K, \mathit{onfile}(F, O)))$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathsf{fs} \rangle (\mathit{user}^{-}(K) \multimap \mathit{may}^{-}(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathsf{fs} \rangle (\mathit{owns}^{-}(K, F) \multimap \mathit{may}^{-}(K, \mathit{onfile}(F, O)))$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathrm{fs} \rangle (\mathit{user}^-(K) \multimap \mathit{may}^-(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \multimap \mathit{may}^-(K, \mathit{onfile}(F, O)))$

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

  $$\texttt{delegate} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \otimes \langle K \rangle \mathit{may}^-(L, \mathit{onfile}(F, O)) \multimap \mathit{may}^-(L, \mathit{onfile}(F, O)))$$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathsf{fs} \rangle (\mathit{user}^-(K) \multimap \mathit{may}^-(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathsf{fs} \rangle (\mathit{owns}^-(K, F) \multimap \mathit{may}^-(K, \mathit{onfile}(F, O)))$

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

  $$\texttt{delegate} : \langle \mathsf{fs} \rangle (\mathit{owns}^-(K, F) \otimes \langle K \rangle \mathit{may}^-(L, \mathit{onfile}(F, O)) \multimap \mathit{may}^-(L, \mathit{onfile}(F, O)))$$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathrm{fs} \rangle (\mathit{user}^-(K) \multimap \mathit{may}^-(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \multimap \mathit{may}^-(K, \mathit{onfile}(F, O)))$

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

  $$\texttt{delegate} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \otimes \langle K \rangle \mathit{may}^-(L, \mathit{onfile}(F, O)) \multimap \mathit{may}^-(L, \mathit{onfile}(F, O)))$$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathrm{fs} \rangle (\mathit{user}^-(K) \multimap \mathit{may}^-(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \multimap \mathit{may}^-(K, \mathit{onfile}(F, O)))$

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

  $$\texttt{delegate} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \otimes \langle K \rangle \mathit{may}^-(L, \mathit{onfile}(F, O)) \multimap \mathit{may}^-(L, \mathit{onfile}(F, O)))$$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathrm{fs} \rangle (\mathit{user}^-(K) \multimap \mathit{may}^-(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \multimap \mathit{may}^-(K, \mathit{onfile}(F, O)))$

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

  $\texttt{delegate} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \otimes \langle K \rangle \mathit{may}^-(L, \mathit{onfile}(F, O)) \multimap \mathit{may}^-(L, \mathit{onfile}(F, O)))$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathsf{fs} \rangle (\mathit{user}^-(K) \multimap \mathit{may}^-(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathsf{fs} \rangle (\mathit{owns}^-(K, F) \multimap \mathit{may}^-(K, \mathit{onfile}(F, O)))$

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

  $$\texttt{delegate} : \langle \mathsf{fs} \rangle (\mathit{owns}^-(K, F) \otimes \langle K \rangle \mathit{may}^-(L, \mathit{onfile}(F, O)) \multimap \mathit{may}^-(L, \mathit{onfile}(F, O)))$$

Specifying the Policies

- Any registered user may insert files.

  $\texttt{mayinsert} : \langle \mathrm{fs} \rangle (\mathit{user}^-(K) \multimap \mathit{may}^-(K, \mathit{insert}))$

- A principal may read, write, or delete files he owns.

  $\texttt{owner} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \multimap \mathit{may}^-(K, \mathit{onfile}(F, O)))$

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

  $$\texttt{delegate} : \langle \mathrm{fs} \rangle (\mathit{owns}^-(K, F) \otimes \langle K \rangle \mathit{may}^-(L, \mathit{onfile}(F, O)) \multimap \mathit{may}^-(L, \mathit{onfile}(F, O)))$$

Proving authorization by combining policies should not be effectful.

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

Specifying the Semantic Consequences of Inserting a File

$$\texttt{insert} : \langle K \rangle do^{-}(K, insert) \otimes \langle \mathsf{fs} \rangle may^{-}(K, insert) \multimap$$
$$\{\exists f.\exists t.\ !\langle \mathsf{fs} \rangle owns^{-}(K, f) \otimes$$
$$[\mathsf{fs}] current^{-}(f, t) \otimes$$
$$[\![\mathsf{fs}]\!] contents^{-}(f, t, \epsilon) \otimes$$
$$[\![K]\!] contents^{-}(f, t, \epsilon)\}$$

Specifying the Semantic Consequences of Inserting a File

$$\texttt{insert} : \langle K \rangle do^{-}(K, insert) \otimes \langle \mathsf{fs} \rangle may^{-}(K, insert) \multimap$$
$$\{\exists f.\exists t.\ !\langle \mathsf{fs} \rangle owns^{-}(K, f) \otimes$$
$$[\mathsf{fs}] current^{-}(f, t) \otimes$$
$$[\![\mathsf{fs}]\!] contents^{-}(f, t, \epsilon) \otimes$$
$$[\![K]\!] contents^{-}(f, t, \epsilon)\}$$

Specifying the Semantic Consequences of Inserting a File

$$\texttt{insert} : \langle K \rangle do^{-}(K, insert) \otimes \langle \mathsf{fs} \rangle may^{-}(K, insert) \multimap$$
$$\{\exists f.\exists t.\ !\langle \mathsf{fs} \rangle owns^{-}(K, f) \otimes$$
$$[\mathsf{fs}] current^{-}(f, t) \otimes$$
$$[\![\mathsf{fs}]\!] contents^{-}(f, t, \epsilon) \otimes$$
$$[\![K]\!] contents^{-}(f, t, \epsilon)\}$$

Specifying the Semantic Consequences of Inserting a File

$$\texttt{insert} : \langle K \rangle do^{-}(K, insert) \otimes \langle \mathsf{fs} \rangle may^{-}(K, insert) \multimap$$
$$\{\exists f . \exists t .\ !\langle \mathsf{fs} \rangle owns^{-}(K, f) \otimes$$
$$[\mathsf{fs}] current^{-}(f, t) \otimes$$
$$[\![\mathsf{fs}]\!] contents^{-}(f, t, \epsilon) \otimes$$
$$[\![K]\!] contents^{-}(f, t, \epsilon)\}$$

Specifying the Semantic Consequences of Inserting a File

$$\texttt{insert} : \langle K \rangle do^{-}(K, insert) \otimes \langle \mathsf{fs} \rangle may^{-}(K, insert) \multimap$$
$$\{\exists f.\exists t.\ !\langle \mathsf{fs} \rangle owns^{-}(K, f) \otimes$$
$$[\mathsf{fs}] current^{-}(f, t) \otimes$$
$$[\![\mathsf{fs}]\!] contents^{-}(f, t, \epsilon) \otimes$$
$$[\![K]\!] contents^{-}(f, t, \epsilon)\}$$

Specifying the Semantic Consequences of Inserting a File

$$\texttt{insert} : \langle K \rangle do^{-}(K, insert) \otimes \langle \mathsf{fs} \rangle may^{-}(K, insert) \multimap$$
$$\{\exists f.\exists t.\ !\langle \mathsf{fs} \rangle owns^{-}(K, f) \otimes$$
$$[\mathsf{fs}] current^{-}(f, t) \otimes$$
$$[\![\mathsf{fs}]\!] contents^{-}(f, t, \epsilon) \otimes$$
$$[\![K]\!] contents^{-}(f, t, \epsilon)\}$$

Specifying the Semantic Consequences of Inserting a File

$$\texttt{insert} : \langle K \rangle do^{-}(K, insert) \otimes \langle \mathsf{fs} \rangle may^{-}(K, insert) \multimap$$
$$\{\exists f.\exists t.\ !\langle \mathsf{fs} \rangle owns^{-}(K, f) \otimes$$
$$[\mathsf{fs}] current^{-}(f, t) \otimes$$
$$[\![\mathsf{fs}]\!] contents^{-}(f, t, \epsilon) \otimes$$
$$[\![K]\!] contents^{-}(f, t, \epsilon)\}$$

Specifying the Semantic Consequences of Inserting a File

$$\texttt{insert} : \langle K \rangle do^{-}(K, insert) \otimes \langle \mathsf{fs} \rangle may^{-}(K, insert) \multimap \{\exists f.\exists t.\ !\langle \mathsf{fs} \rangle owns^{-}(K, f) \otimes [\mathsf{fs}] current^{-}(f, t) \otimes [\![\mathsf{fs}]\!] contents^{-}(f, t, \epsilon) \otimes [\![K]\!] contents^{-}(f, t, \epsilon)\}$$

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

- Creating a file and learning contents are effects.
- Stratification of policies from consequences is evident syntactically
  - Policies have top-level $\langle \cdot \rangle$ and do not contain $\{\cdot\}$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K \rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, read)) \otimes [\mathsf{fs}] current^{-}(F, T) \otimes [\![\mathsf{fs}]\!] contents^{-}(F, T, S) \multimap \{[\![K]\!] contents^{-}(F, T, S) \otimes [\mathsf{fs}] current^{-}(F, T)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K \rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, read)) \otimes [\mathsf{fs}] current^{-}(F, T) \otimes [\![\mathsf{fs}]\!] contents^{-}(F, T, S) \multimap \{[\![K]\!] contents^{-}(F, T, S) \otimes [\mathsf{fs}] current^{-}(F, T)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K \rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, read)) \otimes [\mathsf{fs}] current^{-}(F, T) \otimes [\![\mathsf{fs}]\!] contents^{-}(F, T, S) \multimap \{[\![K]\!] contents^{-}(F, T, S) \otimes [\mathsf{fs}] current^{-}(F, T)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K \rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, read)) \otimes [\mathsf{fs}] current^{-}(F, T) \otimes [\![\mathsf{fs}]\!] contents^{-}(F, T, S) \multimap \{[\![K]\!] contents^{-}(F, T, S) \otimes [\mathsf{fs}] current^{-}(F, T)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K \rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, read)) \otimes [\mathsf{fs}] current^{-}(F, T) \otimes [\![\mathsf{fs}]\!] contents^{-}(F, T, S) \multimap \{[\![K]\!] contents^{-}(F, T, S) \otimes [\mathsf{fs}] current^{-}(F, T)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K \rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, read)) \otimes [\mathsf{fs}] current^{-}(F, T) \otimes [\![\mathsf{fs}]\!] contents^{-}(F, T, S) \multimap \{[\![K]\!] contents^{-}(F, T, S) \otimes [\mathsf{fs}] current^{-}(F, T)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K \rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, read)) \otimes [\mathsf{fs}] current^{-}(F, T) \otimes [\![\mathsf{fs}]\!] contents^{-}(F, T, S) \multimap \{[\![K]\!] contents^{-}(F, T, S) \otimes [\mathsf{fs}] current^{-}(F, T)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K \rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, read)) \otimes [\mathsf{fs}] current^{-}(F, T) \otimes [\![\mathsf{fs}]\!] contents^{-}(F, T, S) \multimap \{ [\![K]\!] contents^{-}(F, T, S) \otimes [\mathsf{fs}] current^{-}(F, T) \}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathsf{fs}] current^-(F, T) \otimes [\![\mathsf{fs}]\!] contents^-(F, T, S) \multimap \{[\![K]\!] contents^-(F, T, S) \otimes [\mathsf{fs}] current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathsf{fs}] current^-(F, T) \multimap \{\exists t.\ [\mathsf{fs}] current^-(F, t) \otimes [\![\mathsf{fs}]\!] contents^-(F, t, S) \otimes [\![K]\!] contents^-(F, t, S)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathsf{fs}]current^-(F, T) \otimes [\![\mathsf{fs}]\!]contents^-(F, T, S) \multimap \{[\![K]\!]contents^-(F, T, S) \otimes [\mathsf{fs}]current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathsf{fs}]current^-(F, T) \multimap \{\exists t.\ [\mathsf{fs}]current^-(F, t) \otimes [\![\mathsf{fs}]\!]contents^-(F, t, S) \otimes [\![K]\!]contents^-(F, t, S)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathsf{fs}]current^-(F, T) \otimes [\![\mathsf{fs}]\!]contents^-(F, T, S) \multimap \{[\![K]\!]contents^-(F, T, S) \otimes [\mathsf{fs}]current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathsf{fs}]current^-(F, T) \multimap \{\exists t.\ [\mathsf{fs}]current^-(F, t) \otimes [\![\mathsf{fs}]\!]contents^-(F, t, S) \otimes [\![K]\!]contents^-(F, t, S)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathsf{fs}\rangle may^{-}(K, onfile(F, read)) \otimes [\mathsf{fs}]current^{-}(F, T) \otimes [\![\mathsf{fs}]\!]contents^{-}(F, T, S) \multimap \{[\![K]\!]contents^{-}(F, T, S) \otimes [\mathsf{fs}]current^{-}(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^{-}(K, onfile(F, write(S))) \otimes \langle \mathsf{fs}\rangle may^{-}(K, onfile(F, write(S))) \otimes [\mathsf{fs}]current^{-}(F, T) \multimap \{\exists t.\ [\mathsf{fs}]current^{-}(F, t) \otimes [\![\mathsf{fs}]\!]contents^{-}(F, t, S) \otimes [\![K]\!]contents^{-}(F, t, S)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathsf{fs}\rangle may^{-}(K, onfile(F, read)) \otimes [\mathsf{fs}]current^{-}(F, T) \otimes [\![\mathsf{fs}]\!]contents^{-}(F, T, S) \multimap \{[\![K]\!]contents^{-}(F, T, S) \otimes [\mathsf{fs}]current^{-}(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^{-}(K, onfile(F, write(S))) \otimes \langle \mathsf{fs}\rangle may^{-}(K, onfile(F, write(S))) \otimes [\mathsf{fs}]current^{-}(F, T) \multimap \{\exists t.\ [\mathsf{fs}]current^{-}(F, t) \otimes [\![\mathsf{fs}]\!]contents^{-}(F, t, S) \otimes [\![K]\!]contents^{-}(F, t, S)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathsf{fs}]current^-(F, T) \otimes [\![\mathsf{fs}]\!]contents^-(F, T, S) \multimap \{[\![K]\!]contents^-(F, T, S) \otimes [\mathsf{fs}]current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathsf{fs}]current^-(F, T) \multimap \{\exists t.\ [\mathsf{fs}]current^-(F, t) \otimes [\![\mathsf{fs}]\!]contents^-(F, t, S) \otimes [\![K]\!]contents^-(F, t, S)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^{-}(K, onfile(F, read)) \otimes \langle \mathrm{fs}\rangle may^{-}(K, onfile(F, read)) \otimes [\mathrm{fs}]current^{-}(F, T) \otimes [\![\mathrm{fs}]\!]contents^{-}(F, T, S) \multimap \{[\![K]\!]contents^{-}(F, T, S) \otimes [\mathrm{fs}]current^{-}(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^{-}(K, onfile(F, write(S))) \otimes \langle \mathrm{fs}\rangle may^{-}(K, onfile(F, write(S))) \otimes [\mathrm{fs}]current^{-}(F, T) \multimap \{\exists t.\ [\mathrm{fs}]current^{-}(F, t) \otimes [\![\mathrm{fs}]\!]contents^{-}(F, t, S) \otimes [\![K]\!]contents^{-}(F, t, S)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathsf{fs}]current^-(F, T) \otimes [\![\mathsf{fs}]\!]contents^-(F, T, S) \multimap \{[\![K]\!]contents^-(F, T, S) \otimes [\mathsf{fs}]current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathsf{fs}]current^-(F, T) \multimap \{\exists t.\ [\mathsf{fs}]current^-(F, t) \otimes [\![\mathsf{fs}]\!]contents^-(F, t, S) \otimes [\![K]\!]contents^-(F, t, S)\}$$

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathsf{fs}] current^-(F, T) \otimes [\![\mathsf{fs}]\!] contents^-(F, T, S) \multimap \{[\![K]\!] contents^-(F, T, S) \otimes [\mathsf{fs}] current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathsf{fs}] current^-(F, T) \multimap \{\exists t.\ [\mathsf{fs}] current^-(F, t) \otimes [\![\mathsf{fs}]\!] contents^-(F, t, S) \otimes [\![K]\!] contents^-(F, t, S)\}$$

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

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathrm{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathrm{fs}]current^-(F, T) \otimes [\![\mathrm{fs}]\!]contents^-(F, T, S) \multimap \{[\![K]\!]contents^-(F, T, S) \otimes [\mathrm{fs}]current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathrm{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathrm{fs}]current^-(F, T) \multimap \{\exists t.\ [\mathrm{fs}]current^-(F, t) \otimes [\![\mathrm{fs}]\!]contents^-(F, t, S) \otimes [\![K]\!]contents^-(F, t, S)\}$$

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

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathsf{fs}] current^-(F, T) \otimes [\![\mathsf{fs}]\!] contents^-(F, T, S) \multimap \{[\![K]\!] contents^-(F, T, S) \otimes [\mathsf{fs}] current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathsf{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathsf{fs}] current^-(F, T) \multimap \{\exists t.\ [\mathsf{fs}] current^-(F, t) \otimes [\![\mathsf{fs}]\!] contents^-(F, t, S) \otimes [\![K]\!] contents^-(F, t, S)\}$$

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

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathrm{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathrm{fs}] current^-(F, T) \otimes [\![\mathrm{fs}]\!] contents^-(F, T, S) \multimap \{[\![K]\!] contents^-(F, T, S) \otimes [\mathrm{fs}] current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathrm{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathrm{fs}] current^-(F, T) \multimap \{\exists t.\ [\mathrm{fs}] current^-(F, t) \otimes [\![\mathrm{fs}]\!] contents^-(F, t, S) \otimes [\![K]\!] contents^-(F, t, S)\}$$

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

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathrm{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathrm{fs}]current^-(F, T) \otimes [\![\mathrm{fs}]\!]contents^-(F, T, S) \multimap \{[\![K]\!]contents^-(F, T, S) \otimes [\mathrm{fs}]current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathrm{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathrm{fs}]current^-(F, T) \multimap \{\exists t.\ [\mathrm{fs}]current^-(F, t) \otimes [\![\mathrm{fs}]\!]contents^-(F, t, S) \otimes [\![K]\!]contents^-(F, t, S)\}$$

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

Semantic Actions for Reading, Writing, and Deleting Files

$$\texttt{read} : \langle K\rangle do^-(K, onfile(F, read)) \otimes \langle \mathrm{fs}\rangle may^-(K, onfile(F, read)) \otimes [\mathrm{fs}]current^-(F, T) \otimes [\![\mathrm{fs}]\!]contents^-(F, T, S) \multimap \{[\![K]\!]contents^-(F, T, S) \otimes [\mathrm{fs}]current^-(F, T)\}$$

$$\texttt{write} : \langle K\rangle do^-(K, onfile(F, write(S))) \otimes \langle \mathrm{fs}\rangle may^-(K, onfile(F, write(S))) \otimes [\mathrm{fs}]current^-(F, T) \multimap \{\exists t.\ [\mathrm{fs}]current^-(F, t) \otimes [\![\mathrm{fs}]\!]contents^-(F, t, S) \otimes [\![K]\!]contents^-(F, t, S)\}$$

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

Simulating the Environment with a Semantic Action

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

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

Simulating the Environment with a Semantic Action

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

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

Simulating the Environment with a Semantic Action

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

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

Simulating the Environment with a Semantic Action

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

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

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

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

Proposed Method

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

Definition of State Circumscribes Relevant Objects

Definition (File System State)

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

Definition of State Circumscribes Relevant Objects

Definition (File System State)

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

1. Each assumption in Γ is either:
   - a policy or a semantic action
   - fs knows $\mathit{contents}^-(F, T, S)$ or K knows $\mathit{contents}^-(F, T, S)$
   - $\langle \mathrm{fs} \rangle \mathit{user}^-(K)$
   - $\langle \mathrm{fs} \rangle \mathit{owns}^-(K, F)$

Definition of State Circumscribes Relevant Objects

Definition (File System State)

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

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

Definition of State Circumscribes Relevant Objects

Definition (File System State)

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

1. Each assumption in Γ is either:
   - a policy or a semantic action
   - fs knows $contents^-(F, T, S)$ or K knows $contents^-(F, T, S)$
   - $\langle \text{fs} \rangle user^-(K)$
   - $\langle \text{fs} \rangle owns^-(K, F)$
2. Each assumption in Δ is either:
   - fs has $current^-(F, T)$
   - $\langle K \rangle do^-(K, O^*)$
   - $\langle K \rangle may^-(L, O^*)$
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 semantics in a security linear logic.
2. Define a system-specific notion of state.
3. Interpret semantic specifications as 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

Definition (Rewrite Step)

$\Gamma; \Delta \longrightarrow \Gamma'; \Delta'$ if and only if there exists a derivation

$$\dfrac{\dfrac{\vdots}{\Gamma'; \Delta' \vdash C^+ \text{ lax}}}{\dfrac{\dfrac{\vdots}{\Gamma; \Delta_2, A_2^+ \vdash C^+ \text{ lax}}}{\Gamma; \Delta_2, [\{A_2^+\}] \vdash C^+ \text{ lax}}\ \{\}_L}$$
$$\dfrac{\dfrac{\vdots}{\Gamma; \Delta_1, [A_1^-] \vdash C^+ \text{ lax}}}{\Gamma; \Delta \vdash C^+ \text{ lax}}\ *$$

parametrically in C^+ lax, where only invertible left rules are used above $\{\}_L$ and both $\Gamma; \Delta$ and $\Gamma'; \Delta'$ are states.

Translating Specifications to Rewrite Steps

Definition (Rewrite Step)

$\Gamma; \Delta \longrightarrow \Gamma'; \Delta'$ if and only if there exists a derivation

$$\dfrac{\dfrac{\begin{array}{c}\Gamma'; \Delta' \vdash C^+ \text{ lax} \\ \vdots \\ \Gamma; \Delta_2, A_2^+ \vdash C^+ \text{ lax}\end{array}}{\Gamma; \Delta_2, [\{A_2^+\}] \vdash C^+ \text{ lax}}\ \{\}_L \\ \vdots \\ \Gamma; \Delta_1, [A_1^-] \vdash C^+ \text{ lax}}{\Gamma; \Delta \vdash C^+ \text{ lax}}\ *$$

parametrically in C^+ lax, where only invertible left rules are used above $\{\}_L$ and both $\Gamma; \Delta$ and $\Gamma'; \Delta'$ are states.

Focusing [Andreoli92] ensures specs are translated atomically.

Translating Specifications to Rewrite Steps

Definition (Rewrite Step)

$\Gamma; \Delta \longrightarrow \Gamma'; \Delta'$ if and only if there exists a derivation

$$\dfrac{\dfrac{\dfrac{\Gamma'; \Delta' \vdash C^+ \text{ lax}}{\vdots}}{\Gamma; \Delta_2, A_2^+ \vdash C^+ \text{ lax}}\ \{\}_L}{\Gamma; \Delta_2, [\{A_2^+\}] \vdash C^+ \text{ lax}}$$

$$\dfrac{\dfrac{\dfrac{\Gamma; \Delta_1, [A_1^-] \vdash C^+ \text{ lax}}{\vdots}}{\Gamma; \Delta \vdash C^+ \text{ lax}}\ *}$$

parametrically in C^+ lax, where only invertible left rules are used above $\{\}_L$ and both $\Gamma; \Delta$ and $\Gamma'; \Delta'$ are states.

Use of lax judgment (effect monad) ensures that rewrite steps:

- Capture all effects; come from assumptions with monadic heads.

Translating Specifications to Rewrite Steps

Definition (Rewrite Step)

$\Gamma; \Delta \longrightarrow \Gamma'; \Delta'$ if and only if there exists a derivation

$$\dfrac{\dfrac{\begin{array}{c}\Gamma'; \Delta' \vdash C^+ \text{ lax} \\ \vdots \\ \Gamma; \Delta_2, A_2^+ \vdash C^+ \text{ lax}\end{array}}{\Gamma; \Delta_2, [\{A_2^+\}] \vdash C^+ \text{ lax}}\ \{\}_L \\ \vdots \\ \Gamma; \Delta_1, [A_1^-] \vdash C^+ \text{ lax}}{\Gamma; \Delta \vdash C^+ \text{ lax}}\ *$$

parametrically in C^+ lax, where only invertible left rules are used above $\{\}_L$ and both $\Gamma; \Delta$ and $\Gamma'; \Delta'$ are states.

Use of lax judgment (effect monad) ensures that rewrite steps:

- Capture all effects; come from assumptions with monadic heads.

Rewrite Steps $\cong$ Semantic Actions

If $\Gamma; \Delta$ is a file system state, then, by construction, only the semantic actions have monadic heads.

Rewrite Steps $\cong$ Semantic Actions

If $\Gamma; \Delta$ is a file system state, then, by construction,
only the semantic actions have monadic heads.

Theorem (Rewrite Step Schemata)

Each rewrite step from a file system state is either:

1. $\Gamma; \langle K \rangle do^-(K, onfile(F, delete)), \Delta_1, \mathsf{fs}\ \mathsf{has}\ current^-(F, T), \Delta_2 \longrightarrow \Gamma; \Delta_2$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, onfile(F, delete))$.

Rewrite Steps $\cong$ Semantic Actions

If $\Gamma; \Delta$ is a file system state, then, by construction,
only the semantic actions have monadic heads.

Theorem (Rewrite Step Schemata)

Each rewrite step from a file system state is either:

1. $\Gamma; \langle K \rangle do^-(K, onfile(F, delete)), \Delta_1, \mathsf{fs}\ \mathsf{has}\ current^-(F, T), \Delta_2 \longrightarrow \Gamma; \Delta_2$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, onfile(F, delete))$.

Rewrite Steps $\cong$ Semantic Actions

If $\Gamma; \Delta$ is a file system state, then, by construction,
only the semantic actions have monadic heads.

Theorem (Rewrite Step Schemata)

Each rewrite step from a file system state is either:

1. $\Gamma; \langle K \rangle do^{-}(K, onfile(F, delete)), \Delta_1, \mathsf{fs}\ \mathsf{has}\ current^{-}(F, T), \Delta_2 \rightarrow \Gamma; \Delta_2$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, delete))$.

Rewrite Steps $\cong$ Semantic Actions

If $\Gamma; \Delta$ is a file system state, then, by construction,
only the semantic actions have monadic heads.

Theorem (Rewrite Step Schemata)

Each rewrite step from a file system state is either:

1. $\Gamma; \langle K \rangle do^-(K, onfile(F, delete)), \Delta_1, \mathsf{fs}\ \mathsf{has}\ current^-(F, T), \Delta_2 \longrightarrow \Gamma; \Delta_2$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, onfile(F, delete))$.

Rewrite Steps $\cong$ Semantic Actions

If $\Gamma; \Delta$ is a file system state, then, by construction,
only the semantic actions have monadic heads.

Theorem (Rewrite Step Schemata)

Each rewrite step from a file system state is either:

1. $\Gamma; \langle K \rangle do^{-}(K, onfile(F, delete)), \Delta_1, \mathsf{fs}\ \mathsf{has}\ current^{-}(F, T), \Delta_2 \longrightarrow \Gamma; \Delta_2$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, delete))$.

Rewrite Steps $\cong$ Semantic Actions

If $\Gamma; \Delta$ is a file system state, then, by construction,
only the semantic actions have monadic heads.

Theorem (Rewrite Step Schemata)

Each rewrite step from a file system state is either:

1. $\Gamma; \langle K \rangle do^-(K, onfile(F, delete)), \Delta_1, \mathsf{fs}\ \mathsf{has}\ current^-(F, T), \Delta_2 \longrightarrow \Gamma; \Delta_2$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, onfile(F, delete))$.

Rewrite Steps $\cong$ Semantic Actions

If $\Gamma; \Delta$ is a file system state, then, by construction,
only the semantic actions have monadic heads.

Theorem (Rewrite Step Schemata)

Each rewrite step from a file system state is either:

1. $\Gamma; \langle K \rangle do^{-}(K, onfile(F, delete)), \Delta_1, \mathsf{fs}\ \mathsf{has}\ current^{-}(F, T), \Delta_2 \longrightarrow \Gamma; \Delta_2$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, delete))$.

Rewrite Steps $\cong$ Semantic Actions

If $\Gamma; \Delta$ is a file system state, then, by construction,
only the semantic actions have monadic heads.

Theorem (Rewrite Step Schemata)

Each rewrite step from a file system state is either:

1. $\Gamma; \langle K \rangle do^{-}(K, onfile(F, delete)), \Delta_1, \mathsf{fs}\ \mathsf{has}\ current^{-}(F, T), \Delta_2 \longrightarrow \Gamma; \Delta_2$
   such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^{-}(K, onfile(F, delete))$.
2. *Similar rewrite steps for* `insert`, `read`, `write` *and* `environment` *elided.*

Rewrite Steps $\cong$ Semantic Actions

If $\Gamma; \Delta$ is a file system state, then, by construction, only the semantic actions have monadic heads.

Theorem (Rewrite Step Schemata)

Each rewrite step from a file system state is either:

1. $\Gamma; \langle K \rangle do^-(K, onfile(F, delete)), \Delta_1, \mathsf{fs}\ \mathsf{has}\ current^-(F, T), \Delta_2 \longrightarrow \Gamma; \Delta_2$ *such that* $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, onfile(F, delete))$.
2. *Similar rewrite steps for* `insert`, `read`, `write` *and* `environment` *elided.*

Proof.

Construct derived rules for left focusing on state assumptions with monadic heads. $\square$

Proving that Rewrite Steps $\cong$ Semantic Actions

Proof.

Recall

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

Proving that Rewrite Steps $\cong$ Semantic Actions

Proof.

Recall

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

The derived rule corresponding to left focusing on `delete` is:

$$\frac{\Gamma; \Delta_1' \vdash \langle K \rangle do^-(K, onfile(F, delete)) \quad \Gamma|_{\mathsf{fs}}; \Delta_2'|_{\mathsf{fs}} \vdash current^-(F, T) \qquad \Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, onfile(F, delete)) \quad \Gamma; \Delta_2, \mathbf{1} \vdash C^+ \ \mathsf{lax}}{\Gamma; \Delta_1', \Delta_1, \Delta_2', \Delta_2 \vdash C^+ \ \mathsf{lax}}$$

Proving that Rewrite Steps $\cong$ Semantic Actions

Proof.

Recall

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

The derived rule corresponding to left focusing on `delete` is:

$$\frac{\Gamma; \Delta_1' \vdash \langle K \rangle do^-(K, onfile(F, delete)) \quad \Gamma|_{\mathsf{fs}}; \Delta_2'|_{\mathsf{fs}} \vdash current^-(F, T) \qquad \Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, onfile(F, delete)) \quad \Gamma; \Delta_2, \mathbf{1} \vdash C^+ \ \mathsf{lax}}{\Gamma; \Delta_1', \Delta_1, \Delta_2', \Delta_2 \vdash C^+ \ \mathsf{lax}}$$

Proving that Rewrite Steps $\cong$ Semantic Actions

Proof.

Recall

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

The derived rule corresponding to left focusing on `delete` is:

$$\frac{\Gamma; \Delta_1' \vdash \langle K \rangle do^-(K, onfile(F, delete)) \quad \Gamma|_{\mathsf{fs}}; \Delta_2'|_{\mathsf{fs}} \vdash current^-(F, T) \qquad \Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, onfile(F, delete)) \quad \Gamma; \Delta_2, \mathbf{1} \vdash C^+ \ \mathsf{lax}}{\Gamma; \Delta_1', \Delta_1, \Delta_2', \Delta_2 \vdash C^+ \ \mathsf{lax}}$$

Proving that Rewrite Steps $\cong$ Semantic Actions

Proof.

Recall

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

The derived rule corresponding to left focusing on `delete` is:

$$\frac{\Gamma; \Delta_1' \vdash \langle K \rangle do^-(K, onfile(F, delete)) \quad \Gamma|_{\mathsf{fs}}; \Delta_2'|_{\mathsf{fs}} \vdash current^-(F, T) \qquad \Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, onfile(F, delete)) \quad \Gamma; \Delta_2, \mathbf{1} \vdash C^+ \ \mathsf{lax}}{\Gamma; \Delta_1', \Delta_1, \Delta_2', \Delta_2 \vdash C^+ \ \mathsf{lax}}$$

Proving that Rewrite Steps $\cong$ Semantic Actions

Proof.

Recall

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

The derived rule corresponding to left focusing on `delete` is:

$$\frac{\Gamma; \Delta_1' \vdash \langle K \rangle do^-(K, onfile(F, delete)) \quad \Gamma|_{\mathsf{fs}}; \Delta_2'|_{\mathsf{fs}} \vdash current^-(F, T) \qquad \Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, onfile(F, delete)) \quad \Gamma; \Delta_2, \mathbf{1} \vdash C^+ \ \mathsf{lax}}{\Gamma; \Delta_1', \Delta_1, \Delta_2', \Delta_2 \vdash C^+ \ \mathsf{lax}}$$

Proving that Rewrite Steps $\cong$ Semantic Actions

Proof.

Recall

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

The derived rule corresponding to left focusing on `delete` is:

$$\frac{\Gamma; \Delta_1' \vdash \langle K\rangle do^-(K, onfile(F, delete)) \quad \Gamma|_{\mathsf{fs}}; \Delta_2'|_{\mathsf{fs}} \vdash current^-(F, T) \qquad \Gamma; \Delta_1 \vdash \langle \mathsf{fs}\rangle may^-(K, onfile(F, delete)) \quad \Gamma; \Delta_2, \mathbf{1} \vdash C^+ \ \mathsf{lax}}{\Gamma; \Delta_1', \Delta_1, \Delta_2', \Delta_2 \vdash C^+ \ \mathsf{lax}}$$

With two simple lemmas and invertibility of $\mathbf{1}$, we recover

$$\Gamma; \langle K\rangle do^-(K, onfile(F, delete)), \Delta_1, \mathsf{fs} \ \mathsf{has} \ current^-(F, T), \Delta_2 \longrightarrow \Gamma; \Delta_2$$

such that $\Gamma; \Delta_1 \vdash \langle \mathsf{fs}\rangle may^-(K, onfile(F, delete))$. $\square$

Proposed Method

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

Analysis: Policy Controls Knowledge

Theorem (Knowledge Safety)

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

$$\Gamma; \Delta \longrightarrow \Gamma', K \text{ knows } \mathit{contents}^{-}(F, T, S); \Delta'$$

then either K knows $\mathit{contents}^{-}(F, T, S) \in \Gamma$ *or the step was an insert, read, or write step for* F *triggered by* K *and permitted by the policy.*

Proof.

By case analysis of the possible rewrite step schemata. □

Analysis: Policy Controls Knowledge

Theorem (Knowledge Safety)

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

$$\Gamma; \Delta \longrightarrow \Gamma', K \text{ knows } \mathit{contents}^{-}(F, T, S); \Delta'$$

then either K knows $\mathit{contents}^{-}(F, T, S) \in \Gamma$ *or the step was an insert, read, or write step for* F *triggered by* K *and permitted by the policy.*

Proof.

By case analysis of the possible rewrite step schemata. □

Analysis: Policy Controls Knowledge

Theorem (Knowledge Safety)

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

$$\Gamma; \Delta \longrightarrow \Gamma', K \text{ knows } \mathit{contents}^{-}(F, T, S); \Delta'$$

then either K knows $\mathit{contents}^{-}(F, T, S) \in \Gamma$ *or the step was an insert, read, or write step for* F *triggered by* K *and permitted by the policy.*

Proof.

By case analysis of the possible rewrite step schemata. □

Analysis: Policy Controls Knowledge

Theorem (Knowledge Safety)

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

$$\Gamma; \Delta \longrightarrow \Gamma', K \text{ knows } \mathit{contents}^{-}(F, T, S); \Delta'$$

then either K knows $\mathit{contents}^{-}(F, T, S) \in \Gamma$ *or the step was an insert, read, or write step for* F *triggered by* K *and permitted by the policy.*

Proof.

By case analysis of the possible rewrite step schemata. □

Analysis: Policy Controls Knowledge

Theorem (Knowledge Safety)

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

$$\Gamma; \Delta \longrightarrow \Gamma', K \text{ knows } \mathit{contents}^{-}(F, T, S); \Delta'$$

then either K knows $\mathit{contents}^{-}(F, T, S) \in \Gamma$ *or the step was an insert, read, or write step for* F *triggered by* K *and permitted by the policy.*

Proof.

By case analysis of the possible rewrite step schemata. □

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

Analysis: Delete Shreds Files

Corollary

If

$$\begin{array}{lll} & \Gamma; \langle K \rangle do^-(K, onfile(F, delete)), \Delta_1, \text{fs has } current^-(F, T), \Delta_2 \\ \longrightarrow & \Gamma; \Delta_2 \\ \longrightarrow^* & \Gamma', L \text{ knows } contents^-(F, T', S); \Delta' \end{array}$$

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

Analysis: Delete Shreds Files

Corollary

If

$$\begin{array}{rl} & \Gamma; \langle K \rangle do^-(K, onfile(F, delete)), \Delta_1, \text{fs has } current^-(F, T), \Delta_2 \\ \longrightarrow & \Gamma; \Delta_2 \\ \longrightarrow^* & \Gamma', L \text{ knows } contents^-(F, T', S); \Delta' \end{array}$$

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

Analysis: Delete Shreds Files

Corollary

If

$$\begin{array}{lll} & \Gamma; \langle K \rangle do^-(K, onfile(F, delete)), \Delta_1, \text{fs has } current^-(F, T), \Delta_2 \\ \longrightarrow & \Gamma; \Delta_2 \\ \longrightarrow^* & \Gamma', L \text{ knows } contents^-(F, T', S); \Delta' \end{array}$$

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

Analysis: Delete Shreds Files

Corollary

If

$$\begin{array}{rl} & \Gamma; \langle K \rangle do^-(K, onfile(F, delete)), \Delta_1, \text{fs has } current^-(F, T), \Delta_2 \\ \longrightarrow & \Gamma; \Delta_2 \\ \longrightarrow^* & \Gamma', L \text{ knows } contents^-(F, T', S); \Delta' \end{array}$$

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

Analysis: Delete Shreds Files

Corollary

If

$$\begin{array}{rl} & \Gamma; \langle K \rangle do^-(K, onfile(F, delete)), \Delta_1, \text{fs has } current^-(F, T), \Delta_2 \\ \longrightarrow & \Gamma; \Delta_2 \\ \longrightarrow^* & \Gamma', L \text{ knows } contents^-(F, T', S); \Delta' \end{array}$$

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

Analysis: Delete Shreds Files

Corollary

If

$$\begin{array}{lll} & \Gamma; \langle K \rangle do^{-}(K, onfile(F, delete)), \Delta_1, \text{fs has } current^{-}(F, T), \Delta_2 \\ \longrightarrow & \Gamma; \Delta_2 \\ \longrightarrow^{*} & \Gamma', L \text{ knows } contents^{-}(F, T', S); \Delta' \end{array}$$

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

Principals cannot learn the contents of deleted files!

- Implies the completeness of garbage collection.

Analysis: Policy Controls Deletes

Theorem (Delete Safety)

If $\Gamma; \Delta, \mathsf{fs}$ has $\mathit{current}^-(F, T)$ *is a file system state such that*

$$\Gamma; \Delta, \mathsf{fs} \text{ has } \mathit{current}^-(F, T) \longrightarrow \Gamma'; \Delta'$$

and there is no T' *such that* fs has $\mathit{current}^-(F, T') \in \Delta'$*, then the step was a delete.*

Analysis: Policy Controls Deletes

Theorem (Delete Safety)

If $\Gamma; \Delta,$ fs has $\mathit{current}^{-}(F, T)$ *is a file system state such that*

$$\Gamma; \Delta, \text{fs has } \mathit{current}^{-}(F, T) \longrightarrow \Gamma'; \Delta'$$

and there is no T' *such that* fs has $\mathit{current}^{-}(F, T') \in \Delta'$*, then the step was a delete.*

Analysis: Policy Controls Deletes

Theorem (Delete Safety)

If $\Gamma; \Delta, \text{fs has } \mathit{current}^-(F, T)$ *is a file system state such that*

$$\Gamma; \Delta, \text{fs has } \mathit{current}^-(F, T) \longrightarrow \Gamma'; \Delta'$$

and there is no T' *such that* $\text{fs has } \mathit{current}^-(F, T') \in \Delta'$*, then the step was a delete.*

Files disappear only if explicitly deleted (and permitted by policy)!

Stratification is Necessary!

Recall the rewrite steps each have a condition:

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

In principle, fs has $current^-(F, T)$ might be in Δ_1.

Stratification is Necessary!

Recall the rewrite steps each have a condition:

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

In principle, fs has $\mathit{current}^-(F, T)$ might be in Δ_1. So, we prove:

Lemma (Possession Strengthening)

If $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle \mathit{may}^-(K, O^*)$, *then* Δ_1 *does not contain any* fs has $\mathit{current}^-(F, T)$.

Stratification is Necessary!

Recall the rewrite steps each have a condition:

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

In principle, fs has $current^-(F, T)$ might be in Δ_1. So, we prove:

Lemma (Possession Strengthening)

If $\Gamma; \Delta_1 \vdash \langle \mathsf{fs} \rangle may^-(K, O^*)$, *then* Δ_1 *does not contain any* fs has $current^-(F, T)$.

This lemma requires stratification of policies and semantic effects!

- 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.

Conclusion

Summary:

1. Specify policies and consequences in a security linear logic.
2. Define a system-specific notion of state.
3. Interpret specifications as 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!

Conclusion

Summary:

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