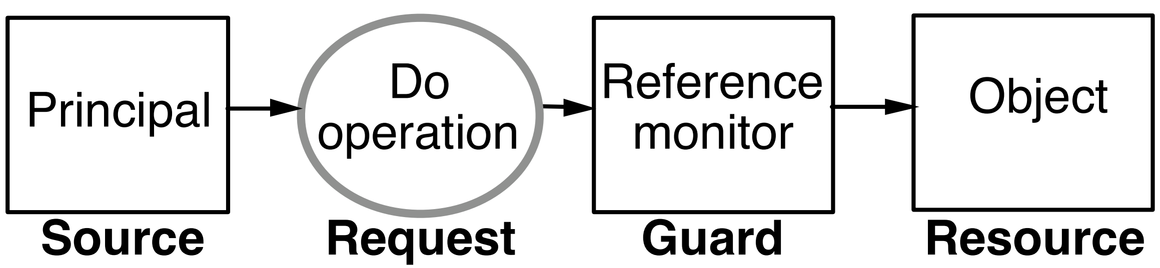
Calculi for Access Control

Martín Abadi

University of California, Santa Cruz
and
Microsoft Research, Silicon Valley

The access control model

- Elements:
  - Objects or resources
  - Requests
  - Sources for requests, called principals
  - A reference monitor to decide on requests



Authentication vs. access control

- Access control (authorization):
  - Is principal A trusted on statement s?
  - If A requests s, is s granted?

- Authentication:
  - Who says s?

An access control matrix
[Lampson, 1971]

objects / principals	file1	file2	file3	file4
user1	rwx	rw	r	x
user2	r	r		x
user3	r	r		x

Access control in current practice

- Access control is pervasive
  - applications
  - virtual machines
  - operating systems
  - firewalls
  - doors
  - …
- Access control seems difficult to get right.
- Distributed systems make it harder.

General theories and systems

- Over the years, there have been many theories and systems for access control.
  - Logics
  - Languages
  - Infrastructures (e.g., PKIs)
  - Architectures
- They often aim to explain, organize, and unify access control.

An approach

- A notation for representing principals and their statements, and perhaps more:
  - objects and operations,
  - trust,
  - channels,
  - …
- Derivation rules

A calculus for access control

[Abadi, Burrows, Lampson, and Plotkin, 1993]

- A simple notation for assertions
  - A says s
  - A speaks for B (sometimes written $A \Rightarrow B$)

- With logical rules
  - $\vdash$ A says $(s \rightarrow t) \rightarrow$ (A says s) $\rightarrow$ (A says t)
  - If $\vdash$ s then $\vdash$ A says s.
  - $\vdash$ A speaks for B $\rightarrow$ (A says s) $\rightarrow$ (B says s)
  - $\vdash$ A speaks for A
  - $\vdash$ A speaks for B $\wedge$ B speaks for C $\rightarrow$ A speaks for C

An example

- Let good-to-delete-file1 be a proposition. Let B controls s stand for
  (B says s) $\rightarrow$ s
- Assume that
  - B controls (A speaks for B)
  - B controls good-to-delete-file1
  - B says (A speaks for B)
  - A says good-to-delete-file1
- We can derive:
  - B says good-to-delete-file1
  - good-to-delete-file1

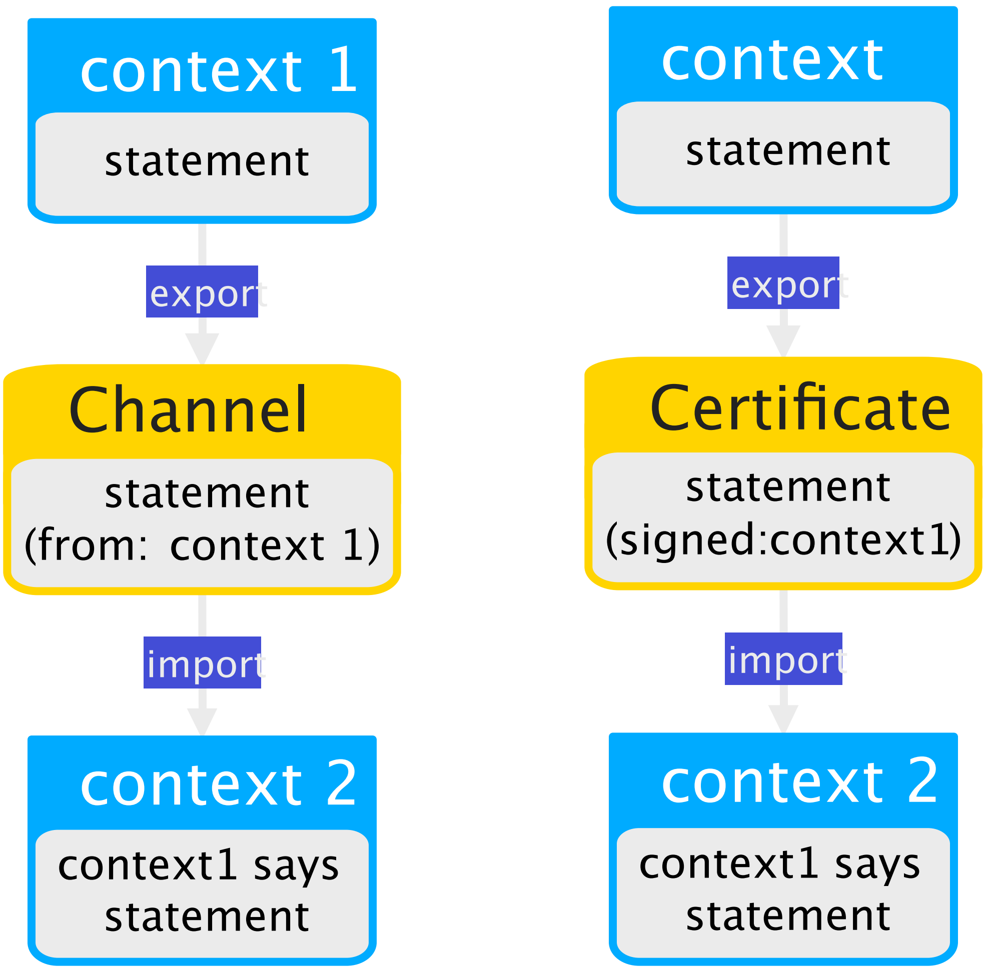
Another example

- Let good-to-delete-file2 be a proposition too.
- Assume that
  - B controls (A speaks for B)
  - B controls good-to-delete-file1
  - B says (A speaks for B)
  - A says (good-to-delete-file1 $\wedge$ good-to-delete-file2)
- We can derive:
  - B says good-to-delete-file1
  - good-to-delete-file1

Says

Says represents communication across contexts.

Says abstracts from the details of authentication.



Choosing axioms

- Standard modal logic?
  - (As above.)
- Less?
  - Treat says “syntactically”, with no special rules (Halpern and van der Meyden, 2001)

Choosing axioms (cont.)

- More?
  - $\vdash$ (A says (B speaks for A)) $\rightarrow$ (B speaks for A)
    The "hand-off axiom";
    in other words, A controls (B speaks for A).
  - $\vdash$ s $\rightarrow$ (A says s)
    (Lampson, 198?; Appel and Felten, 1999)
    but then
    $\vdash$ (A says s) $\rightarrow$ s $\vee$ (A says false)

Semantics

- Following standard semantics of modal logics, a principal may be mapped to a binary relation on possible worlds.

  A says s holds at world w

  iff

  s holds at world w'

  for every w' such that w A w'

- This is formally viable, also for richer logics.
- It does not give much insight on the meaning of authority, but it is sometimes useful.

Proof strategies

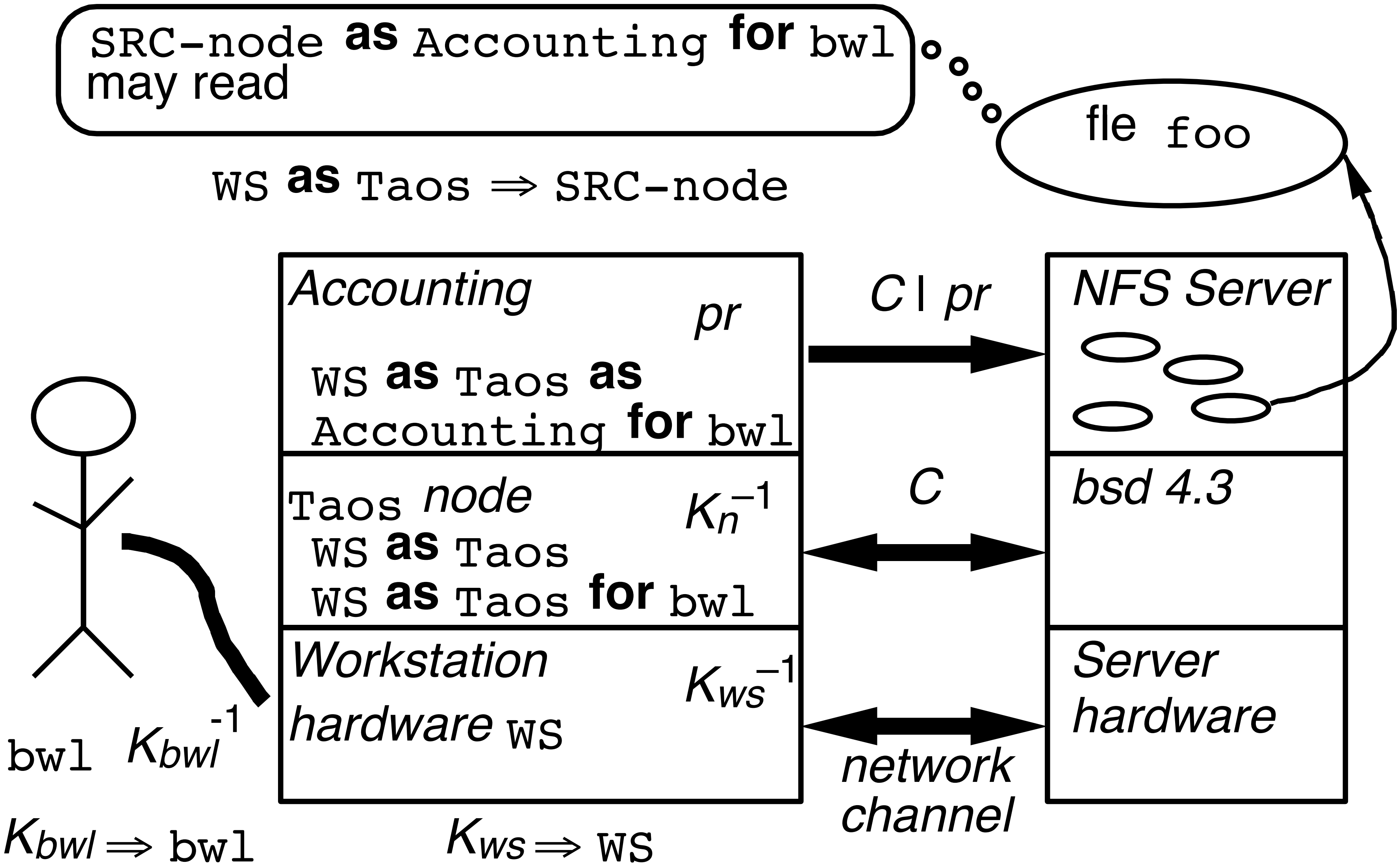
- Style of proofs:
  - Hilbert systems
  - Tableaux
    (Massacci, 1997)
  - …
- Proof distribution:
  - Proofs done at reference monitors
  - Partial proofs provided by clients
    (Wobber et al., 1994; Appel and Felten, 1999)
  - With certificates pulled or pushed

More principals

- Compound principals represent a richer class of sources for requests:
  - $A \wedge B$ Alice and Bob (cosigning)
  - A quoting B server.uxyz.edu quoting Alice
  - A for B server.uxyz.edu for Alice
  - A as R Alice as Reviewer

  $A \wedge B$ speaks for A, etc.
- Groups represent collections of principals, and may be treated as principals themselves.
- Programs may be treated as roles.

Applications (1): Security in an operating system [Wobber et al., 1994]



Applications (2): An account of security in JVMs [Wallach and Felten, 1998]

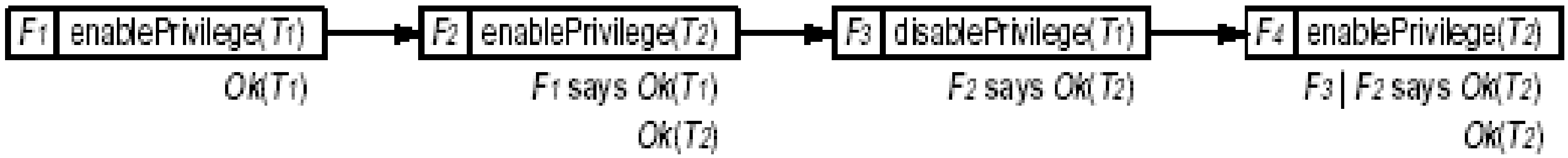
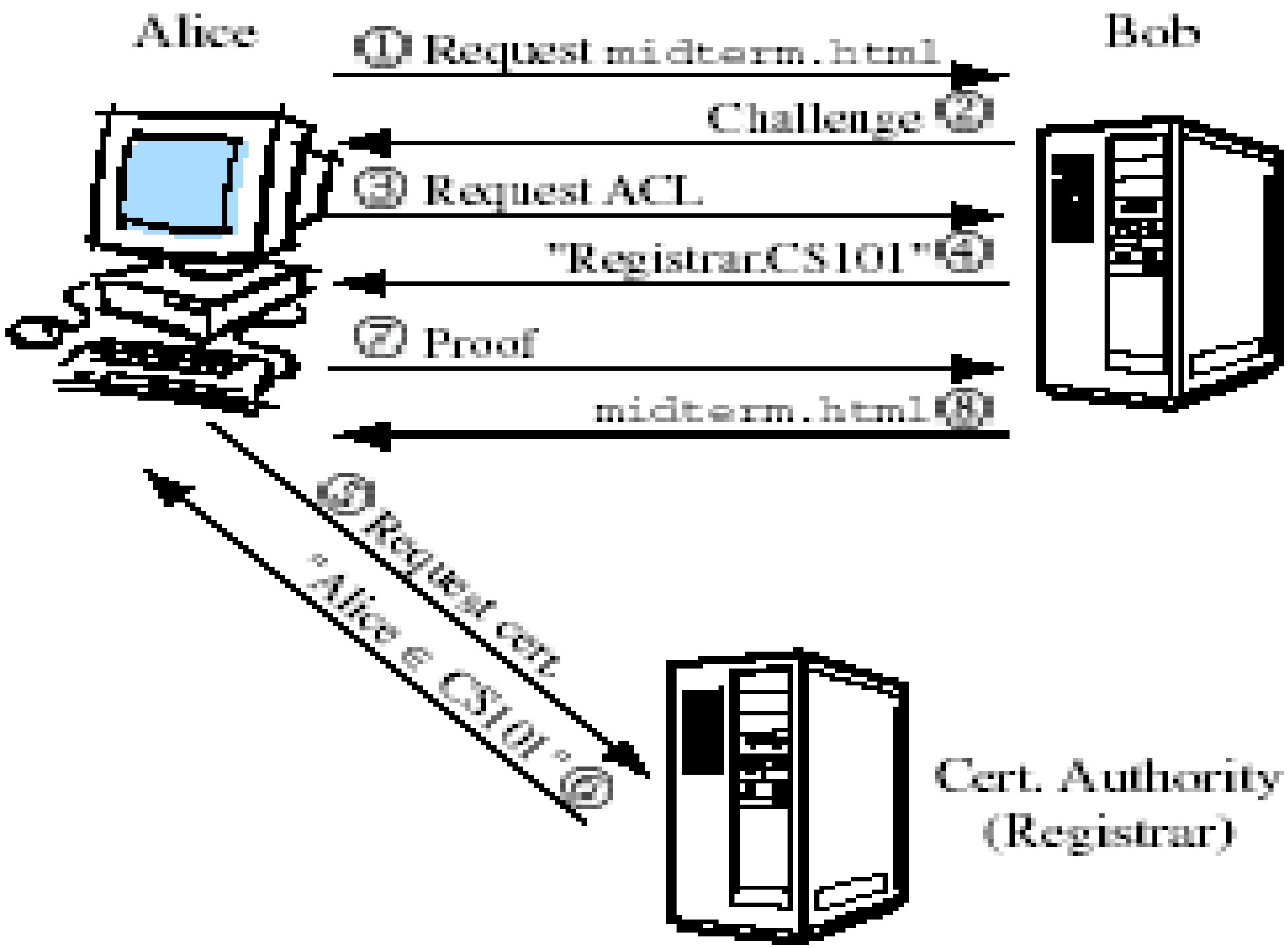


Figure 2: Example of interaction between stack frames. Each rectangle represents a stack frame. Each stack frame is labeled with its name. In this example, each stack frame makes one `enablePrivilege()` or `disablePrivilege()` call, which is also written inside the rectangle. Below each frame is written its belief set after its call to `enablePrivilege()` or `disablePrivilege()`.

Applications (3): A Web access control system [Bauer, Schneider, and Felten, 2002]



Applications (4): The Grey system

[Bauer, Reiter, et al., 2005]

- Converts a cell-phone into a tool for delegating and exercising authority.
- Uses cell phones to replace physical locks and key systems.
- Implemented in part of CMU.
- With access control based on logic and distributed proofs.

Distributed Proving

I can prove that with any of
1) `Jon speaksfor Mike.Student`
2) `Jon speaksfor Mike.Admin`
3) `Jon speaksfor Mike.Wife`
4) `Delegates(Mike, Jon, D208.open)`

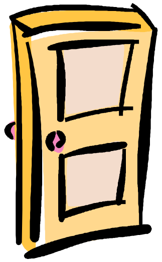
Jon

Open D208

Jon's phone

Phone discovers door

D208



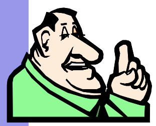
To prove:
`Mike says Goal(D208.open)`

Hmm, I can't prove that. I'll ask Mike's phone for help.

Please help

Mike's phone

Mike



`Jon speaksfor Mike.Student`

Proof of:
`Jon says Goal(D208.open) →`
`Mike says Goal(D208.open)`

Proof of:
`Mike says Goal(D208.open)`

Further applications: Other languages and systems

Several languages rely on logics for access control and on logic programming:

- D1LP and RT [Li, Mitchell, et al.]
- SD3 [Jim]
- Binder [DeTreville]

"speaks for" plays a role in other systems:

- SDSI and SPKI [Lampson and Rivest; Ellison et al.]
- Plan 9 [Pike et al.]
- …

Some issues

- It is easy to add constructs and axioms, but sometimes difficult to decide which are right.
- Explicit representations for proofs are useful.

- Even with logic, access control typically does not provide end-to-end guarantees
(e.g., the absence of flows of information).

The Dependency Core Calculus (DCC) [Abadi, Banerjee, Heintze, and Riecke, 1999]

- A minimal but expressive calculus in which the types capture dependencies.
- A foundation for some static program analyses:
  - information-flow control,
  - binding-time analysis,
  - slicing,
  - …
- Based on the computational lambda calculus.

DCC basics

- Let L be a lattice.
- For each type s and each l in L, there is a type $T_l(s)$.
- If $l \sqsubseteq k$ then terms of type $T_k(t)$ may depend on terms of type $T_l(s)$.

For instance:

- The lattice may have two elements Public and Secret, with Public $\sqsubseteq$ Secret.
- $T_{Public}(int)$ and $T_{Secret}(bool)$ would be two types.
- Then DCC guarantees that outputs of type $T_{Public}(int)$ do not depend on inputs of type $T_{Secret}(bool)$.

A new look at DCC

- We read DCC as a logic,
via the Curry-Howard isomorphism.
  - Types are propositions.
  - Programs are proofs.
- We consider significant but routine variations on the original DCC:
  - We remove fixpoints and related constructs.
  - We add polymorphism in the style of System F.
- We write A says s instead of $T_l(s)$.
- We write A speaks for B as an abbreviation for $\forall X.\ (A \text{ says } X \to B \text{ says } X)$.

A new look at DCC (cont.)

- The result is a logic for access control, with some principles and some useful theorems.
- The logic is intuitionistic
(like a recent system by Garg and Pfenning).
- Terms are proofs to be used in access control.

Simply Typed DCC: Syntax

The types of Simply Typed DCC are given by the grammar:

$$s ::= \mathtt{true} \mid (s \vee s) \mid (s \wedge s) \mid (s \rightarrow s) \mid A \ \mathtt{says} \ s$$

where A ranges over elements of a lattice $\mathcal{L}$, equipped with a partial order $\sqsubseteq$.

Simply Typed DCC: Protected types

If $A \sqsubseteq B$, then B **says** s is protected at level A.

true is protected at level A.

If s and t are protected at level A, then $(s \wedge t)$ is protected at level A.

If t is protected at level A, then B **says** t is protected at level A.

If t is protected at level A, then $(s \rightarrow t)$ is protected at level A.

(It will turn out that, up to equivalence, the types protected at level A are of the form A **says** t.)

Simply Typed DCC: Typing rules

- The typing rules are those of simply typed λ-calculus plus:

$$\frac{\Gamma \vdash e : s}{\Gamma \vdash (\eta_A\ e) : A \ \texttt{says}\ s}$$

$$\frac{\Gamma \vdash e : A \ \texttt{says}\ s \qquad \Gamma, x : s \vdash e' : t}{\Gamma \vdash \texttt{bind}\ x = e\ \texttt{in}\ e' : t} \quad t \text{ protected at level } A$$

$$\Gamma, x : s, \Gamma' \vdash x : s \qquad\qquad \Gamma \vdash () : \mathtt{true}$$

$$\frac{\Gamma, x : s_1 \vdash e : s_2}{\Gamma \vdash (\lambda x : s_1.\, e) : (s_1 \to s_2)} \qquad \frac{\Gamma \vdash e : (s_1 \to s_2) \qquad \Gamma \vdash e' : s_1}{\Gamma \vdash (e\, e') : s_2}$$

$$\frac{\Gamma \vdash e_1 : s_1 \qquad \Gamma \vdash e_2 : s_2}{\Gamma \vdash \langle e_1, e_2 \rangle : (s_1 \wedge s_2)}$$

$$\frac{\Gamma \vdash e : (s_1 \wedge s_2)}{\Gamma \vdash (\mathtt{proj}_1\, e) : s_1} \qquad \frac{\Gamma \vdash e : (s_1 \wedge s_2)}{\Gamma \vdash (\mathtt{proj}_2\, e) : s_2}$$

$$\frac{\Gamma \vdash e : s_1}{\Gamma \vdash (\mathtt{inj}_1\, e) : (s_1 \vee s_2)} \qquad \frac{\Gamma \vdash e : s_2}{\Gamma \vdash (\mathtt{inj}_2\, e) : (s_1 \vee s_2)}$$

$$\frac{\Gamma \vdash e : (s_1 \vee s_2) \qquad \Gamma, x : s_1 \vdash e_1 : s \qquad \Gamma, x : s_2 \vdash e_2 : s}{\Gamma \vdash (\mathtt{case}\ e\ \mathtt{of}\ \mathtt{inj}_1(x).\, e_1 \mid \mathtt{inj}_2(x).\, e_2) : s}$$

$$\frac{\Gamma \vdash e : s}{\Gamma \vdash (\eta_A\, e) : A\ \mathtt{says}\ s}$$

$$\frac{\Gamma \vdash e : A\ \mathtt{says}\ s \qquad \Gamma, x : s \vdash e' : t}{\Gamma \vdash \mathtt{bind}\ x = e\ \mathtt{in}\ e' : t} \quad t \text{ protected at level } A$$

Simply Typed DCC: Logical reading

- Reading the typing rules as a logic can be simply a matter of omitting terms…

$$\Gamma, s, \Gamma' \vdash s \qquad\qquad \Gamma \vdash \texttt{true}$$

$$\frac{\Gamma, s_1 \vdash s_2}{\Gamma \vdash (s_1 \rightarrow s_2)} \qquad \frac{\Gamma \vdash (s_1 \rightarrow s_2) \qquad \Gamma \vdash s_1}{\Gamma \vdash s_2}$$

$$\frac{\Gamma \vdash s_1 \qquad \Gamma \vdash s_2}{\Gamma \vdash (s_1 \wedge s_2)}$$

$$\frac{\Gamma \vdash (s_1 \wedge s_2)}{\Gamma \vdash s_1} \qquad \frac{\Gamma \vdash (s_1 \wedge s_2)}{\Gamma \vdash s_2}$$

$$\frac{\Gamma \vdash s_1}{\Gamma \vdash (s_1 \vee s_2)} \qquad \frac{\Gamma \vdash s_2}{\Gamma \vdash (s_1 \vee s_2)}$$

$$\frac{\Gamma \vdash (s_1 \vee s_2) \qquad \Gamma, s_1 \vdash s \qquad \Gamma, s_2 \vdash s}{\Gamma \vdash s}$$

$$\frac{\Gamma \vdash s}{\Gamma \vdash A \ \texttt{says}\ s}$$

$$\frac{\Gamma \vdash A \ \texttt{says}\ s \qquad \Gamma, s \vdash t}{\Gamma \vdash t} \quad t \text{ protected at level } A$$

Polymorphic DCC

- Polymorphic DCC is obtained by adding type variables and universal quantification, with the standard rules.

$$\frac{\Gamma, X \vdash e : s}{\Gamma \vdash (\Lambda X.\, e) : \forall X.\, s}$$

$$\frac{\Gamma \vdash e : \forall X.\, s}{\Gamma \vdash (et) : s[t/X]} \quad (t \text{ well-formed in } \Gamma)$$

- The definition of "protected" is extended:

If t is protected at level A,
then $\forall X.\, t$ is protected at level A.

Semantics

- Operational semantics (one possibility):
  - usual λ-calculus rules, plus
  - the new rule

$$\texttt{bind}\ x = (\eta_A\ e)\ \texttt{in}\ e' \quad \texttt{reduces to} \quad e'[e/x]$$

(Zdancewic recently checked subject reduction and progress properties for this semantics in Twelf.)

- Denotational semantics? (We have some pieces, but more could be done.)

DCC theorems

- We can rederive the core of the previous logics:
  - $\vdash$ A says (s $\rightarrow$ t) $\rightarrow$ (A says s) $\rightarrow$ (A says t)
  - If $\vdash$ s then $\vdash$ A says s.
  - $\vdash$ A speaks for B $\rightarrow$ (A says s) $\rightarrow$ (B says s)
  - $\vdash$ A speaks for A
  - $\vdash$ A speaks for B $\wedge$ B speaks for C $\rightarrow$ A speaks for C

DCC theorems (cont.)

- DCC has some additional useful theorems.
  - $\vdash$ (A says (B speaks for A)) $\rightarrow$ (B speaks for A)
  - $\vdash$ s $\rightarrow$ (A says s)

  and also
  - $\vdash$ A says A says s $\rightarrow$ A says s
  - $\vdash$ A says B says s $\rightarrow$ B says A says s

These follow from general rules,
apparently without annoying consequences.

DCC theorems (cont.)

- If $A \sqsubseteq B$, then $\vdash$ A speaks for B.
- B says (A speaks for B) does not imply $A \sqsubseteq B$.
- B says ($A \sqsubseteq B$) is not even syntactically correct.

- Lattice elements may represent groups, rather than individual principals.
- The operations $\sqcap$ and $\sqcup$ may represent group intersection and union.
  - $\vdash (A \sqcap B)$ says $s \to A$ says $s \land B$ says s.
  - The converse fails (quite reasonably).

DCC metatheorems

- DCC also has a useful metatheory, which includes old and new non-interference results.

Mapping to System F (warm-up)

- Tse and Zdancewic have defined a clever encoding of Simply Typed DCC in System F.
- We can define a more trivial mapping $(.)^{\mathsf{F}}$ from Polymorphic DCC to System F by letting

$$(A\ \mathtt{says}\ s)^{\mathsf{F}} \;=\; (s)^{\mathsf{F}}$$

- This mapping preserves provability, so Polymorphic DCC is consistent.

Non-interference

- Access control requires the integrity of requests and policies.
  - We would like some guarantees on the possible effect of the statements of principals.
  - E.g., if A and B are unrelated principals, then B’s statements should not interfere with A’s.
- There are previous non-interference theorems for DCC, and we can prove some more.

Another mapping: what a formula means when *B* may say anything

For a type s and $B \in \mathcal{L}$, we define $(s)^B$ as follows:

$$
\begin{array}{rcl}
(\mathtt{true})^B & = & \mathtt{true} \\
(s_1 \vee s_2)^B & = & (s_1)^B \vee (s_2)^B \\
(s_1 \wedge s_2)^B & = & (s_1)^B \wedge (s_2)^B \\
(s_1 \rightarrow s_2)^B & = & (s_1)^B \rightarrow (s_2)^B \\
(A \;\mathtt{says}\; s)^B & = & \begin{cases} \mathtt{true} & \text{if } B \sqsubseteq A \\ A \;\mathtt{says}\; (s)^B & \text{otherwise} \end{cases} \\
(X)^B & = & X \\
(\forall X.\, s)^B & = & \forall X.\, (s)^B
\end{array}
$$

A theorem

In Polymorphic DCC,
for every typing environment Γ,
type s, and $B \in \mathcal{L}$,
if $\Gamma \vdash e : s$
then there exists e'
such that $(\Gamma)^B \vdash e' : (s)^B$.

Some corollaries

If $B \not\sqsubseteq A$, then

$$\nvdash (B \mathtt{\ says\ } t) \rightarrow (A \mathtt{\ says\ } \forall X.\, X)$$

If s mentions no principal C such that $B \sqsubseteq C$ and

$$\vdash (B \mathtt{\ says\ } t) \rightarrow (A \mathtt{\ says\ } s)$$

then

$$\vdash A \mathtt{\ says\ } s$$

Note however that $\vdash B \mathtt{\ says\ } t \rightarrow A \mathtt{\ says\ } B \mathtt{\ says\ } t$.

Further work and open questions

- Rich, convenient languages for writing policies.
- Procedures for analyzing policies.
- Revisiting compound principals.
- Other logics with similar principles
  (but different theorems).
- More semantics.
- Integration of access control into programming.
- Relation to information flow.

Outlook

- We can provide at least partial evidence of the “goodness” of our rules.
- Even with imperfect rules, declarative policies may contribute to improving authorization.
- Logics and types should help.