Logic in Access Control

Martín Abadi

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

Contents

- Introduction to access control.
- Some logical approaches.
- Some systems and languages.
- A closer look at a particular logic based on a type system for tracking dependencies.

Contents (cont.)

- Some retrospective, some news.
- Some technical material, about languages, logic, proofs, security, and their connections.
  - A nonstandard application of language-based security to access control.
  - A surprising relation between access control and information-flow control.

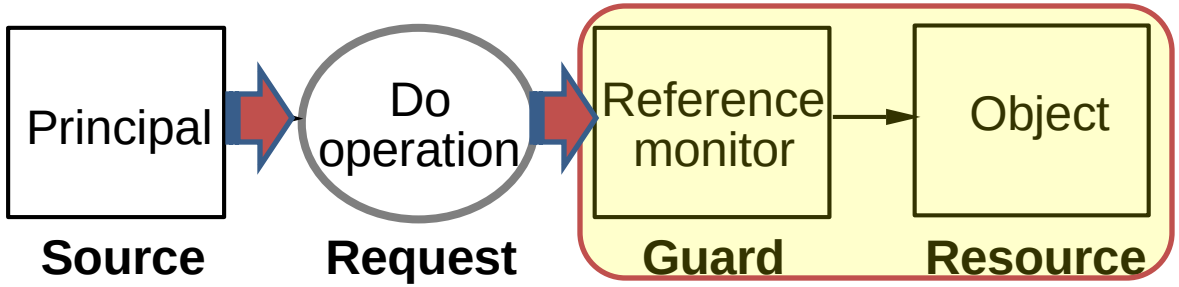
References

- Any general book on security (e.g., Ross Anderson's) for the introductory material.
- The tutorial notes in the FOSAD volume for more logical material, e.g., on the logic CDD.
- The papers cited there for a broader look at the field.

Basics

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

The principle of complete mediation

[Saltzer and Schroeder, 1975]

Every access to every object must be checked for authority.

- This principle can be enforced in several ways:
  - The OS intercepts some of the requests. The hardware catches others.
  - A software wrapper / interpreter intercepts some of the requests. (E.g., as in VMs.)

Implementing access control

Two strategies (often combined):
ACLs and capabilities.

- ACL: a column of an access control matrix, attached to an object.

- Capability: (basically) a pair of an object and an operation, for a given principal.
  It means that the principal may perform the operation on the object.

More on ACLs

- An ACL says which subjects can access a particular object.
  - It is a column of an access control matrix,
  - typically maintained “near” the object that it protects.
- ACLs can be compact and easy to review.
- ACLs may have negative entries (and then evaluation may be order-depedendent).
- Revoking a subject can be painful.

More on capabilities

- An alternative is to associate capabilities with each subject.
  - A capability means that the subject can perform an operation on an object.
- These capabilities form a row of an access control matrix for the subject.
- Capabilities are often easy to pass around (so they enable delegation).
- They can be hard to revoke.

Implementing capabilities

⇒ *Subjects should not be allowed to forge capabilities.*

- This leads to implementations of capabilities
  - stored in a protected address space,
  - with special tags with hardware support,
  - as references in a typed language,
  - with a secret,
  - with cryptography, e.g., certificates.

ACLs and capabilities

- ACLs and capabilities are dual.
- Both can yield practical implementations of access matrices.
- In actual systems, they are often combined.

Some further elaborations and complications

- Joint requests
- Groups
- Roles
- Programs

- Defining principals, objects, and operations

Conjunctions

- Sometimes a request should be granted only if it is made jointly by several principals.
- A conjunction may or may not be made explicit in the access policy.

Groups and roles

- Principals can be organized into groups.
- Principals can play roles.
- These groups and roles may be used as a level of indirection in access control.
  - E.g., any member of a group G may access a file f.

Groups and roles (cont.)

- Suppose that any member of a group G may access a file f owned by A.
  - G may be maintained by someone other A.
  - The group may change over time, without immediate knowledge of A.
  - The ACL for f should be short and clear.
  - Proofs of memberships resemble (are?) capabilities.
  - Access to f might be partly anonymous.
  - Still, A may require a proof of identity at each f access, for auditing.

More on objects and operations

- Objects and operations may also be put in groups, e.g.,
  - all company files,
  - all read operations on an object.
- Sometimes operations should be bundled, e.g.,
  - read a patient's record,
  - write a log record.

Design choices

- Principals, objects, and operations should have the “right” granularity and be at the “right” level of abstraction
  - for ease of understanding,
  - to avoid giving away too much privilege.

Programs

- Programs may be principals too.
- But then:
  - we need to deal with call chains,
    - e.g., application on browser on OS,
  - we still need to connect programs to other principals
    - who write them or edit them,
    - who provide them,
    - who install them,
    - who call them.

Installing programs

- Programs should be set up so that they get appropriate rights when they run.
- Programs should be adequately protected from editing.

Running programs

- What are the run-time rights of a program?
  - those of the caller,
  - those of the program owner, or
  - some combination, or
  - something else, e.g, because of intrinsic properties.
- E.g., a program that moves incoming mail to a user's inbox may need to combine system rights and user rights.

Running programs (cont.)

- Some answers:
  - setuid,
  - program identities,
  - code access security (with stack inspection or alternatives),
  - proof-carrying code,
  - …

Protection and isolation

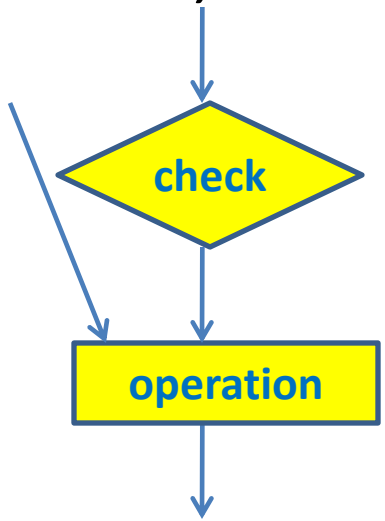
- At run-time, programs should be protected and limited to communicate over proper interfaces.
- This is often the job of the computing platform (OS + hardware).
  - It can implement address spaces
    so that programs in separate spaces cannot interact directly
    (e.g., cannot smash or snoop on one another).
- A language and its run-time system can provide finer control over communication.
  - (Remember capabilities?)

Common dangers

- Access control can be insufficient or irrelevant
  - when it is implemented incorrectly,
  - when the underlying operations are implemented incorrectly,
  - when the policy is wrong,
  - when it is circumvented.

Circumventing access control

- Sometimes the reference monitor does not protect all important objects and operations, or does not protect them all the time.
  - Control-flow subversions.
  - Race conditions.
  - Data recovery from disks.
  - Hostile platforms (e.g., for DRM systems).
  - Users that give out sensitive information.
  - …



Issues

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

Issues (cont.)

- Many characteristics of distributed systems make access control harder:
  - size,
  - faultiness (e.g., revocations may get lost),
  - heterogeneity (e.g., of communication channels and of protection mechanisms),
  - autonomy, lack of central administration and therefore of central trust,
  - …

Logical approaches

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.
- They may be intellectually pleasing.
- They may actually help.

Algorithmic analysis

[starting with Harrison, Ruzzo, and Ullman, 1976]

- A system has finite sets of rights and commands.
- A configuration is an access control matrix.
- A command is of the form "if conditions hold, perform operations" (with some parameters).
  - The conditions are predicates on the matrix.
  - The operations add or delete rights, principals, and objects. E.g.:

```
command CONFER_r (owner, friend, file)
  if own in (owner, file)
  then enter r into (friend,file)
end
```

Algorithmic analysis (cont.)

- Safety means that untrusted subjects cannot access a resource in any reachable state.
- Safety is undecidable (in general).

Algorithmic analysis (cont.)

[in particular, Li, Winsborough, and Mitchell, 2003]

- Not all interesting problems are undecidable!
- Consider the containment problem:

  In every reachable state, does every principal that has one property (e.g., has access to a resource) also have another property (e.g., being an employee)?

  For different classes of systems, this problem is decidable (in coNP or coNEXP).

Formal verification

A formally verified security kernel is widely considered to offer the most promising basis for the construction of truly secure computer systems at least in the short term. A number of kernelized systems have been constructed and various models of security have been formulated to serve as the basis for their verification. Despite the enthusiasm for this approach there remain certain difficulties and problems in its application […]

(Rushby, 1981)

A logic from matrices

- An access control matrix may be represented with a ternary predicate symbol may-access.
- The setting may be a fairly standard, classical predicate calculus.
- We may then write formulas such as:

  may-access(Alice, Foo.txt, Rd)

  and rules such as:

  may-access(p, o, Wr) $\rightarrow$ may-access(p, o, Rd)

A logic from matrices: questions

- Does this really help?
  - In describing policies?
  - In analyzing policies?
- We may need many more constructs and axioms for representing security policies. For example:
  - may-jointly-access(p,q,o,r)
  - owns(p,o)
  - …

(When are we done?)

Logics for distributed systems

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

A calculus for access control

[with 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
  - $\vdash A \text{ says } (s \rightarrow t) \rightarrow (A \text{ says } s) \rightarrow (A \text{ says } t)$
  - If $\vdash s$ then $\vdash A$ says s.
  - $\vdash A \text{ speaks for } B \rightarrow (A \text{ says } s) \rightarrow (B \text{ says } s)$
  - $\vdash A \text{ speaks for } A$
  - $\vdash A \text{ speaks for } B \wedge B \text{ speaks for } C \rightarrow A \text{ speaks for } C$

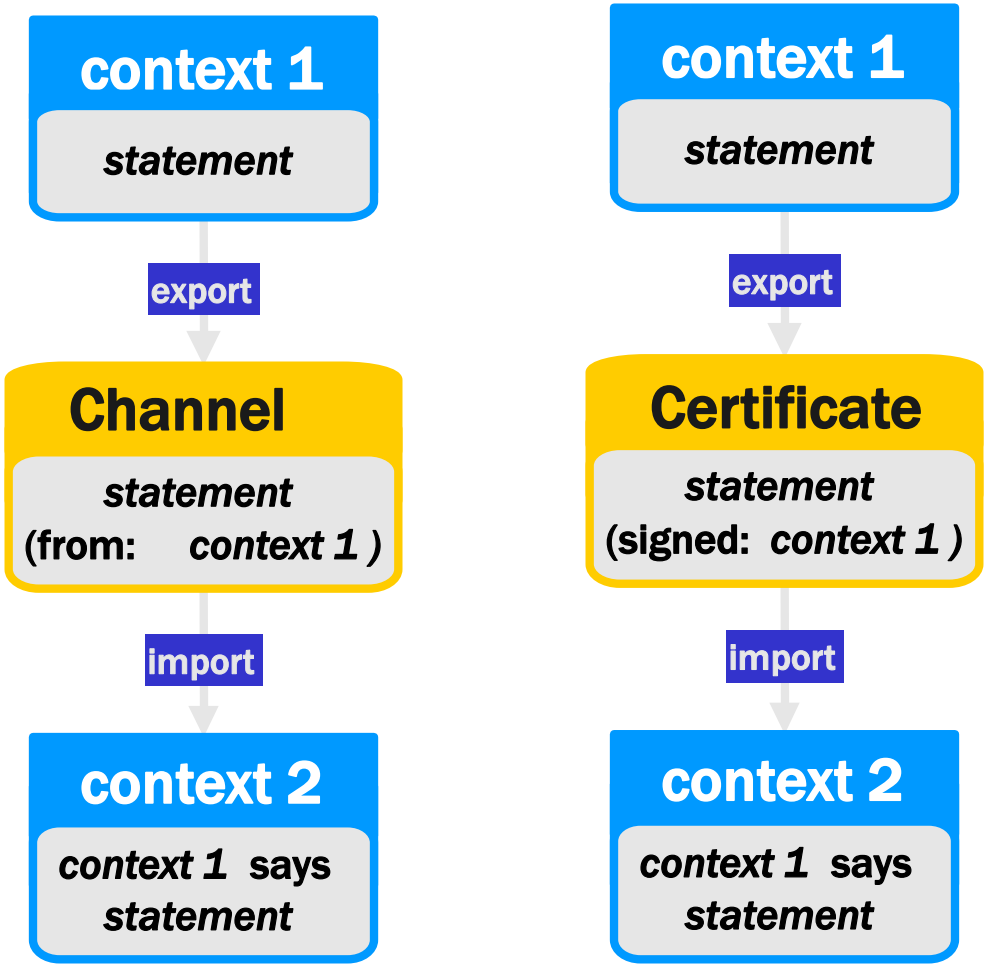
An example

- 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

Says

"says" represents communication across contexts.

"says" abstracts from the details of authentication.



Choosing axioms

- Classical? Intuitionistic? Other?
- Standard modal logic for “says”?
  - (As above.)
- Less?
  - Give “says” no special rules.

    [Halpern and van der Meyden, 2001]

Choosing axioms (cont.)

- More?
  - $\vdash s \rightarrow (A \text{ says } s)$

    [Lampson, 198?; Appel and Felten, 1999]

    but in classical logic this implies that "saying" is black-and-white: $(A \text{ says } s) \rightarrow (s \vee (A \text{ says false}))$
  - $\vdash (A \text{ says } (B \text{ speaks for } A)) \rightarrow (B \text{ speaks for } A)$

    The "hand-off axiom": A controls (B speaks for A)

Semantics (briefly)

- 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., 1993; Appel and Felten, 1999]
  - With certificates pulled or pushed

More principals

- Compound principals represent a richer class of sources for requests:
  - $A \wedge B$
  - A quoting B
  - A for B
  - A as R

  $A \wedge B$ speaks for A, etc.

(Another addition: local naming.)

Groups and programs

- We may represent each group by a principal. Then, when A is a member of G, we may write that A speaks for G.
- In practice, it is harder to know when A is not a member of G.

- Programs may be treated as roles.

An example

- The cast:
  - CA, the certification authority, with public key K_{CA}
  - WS, a workstation, with public key K_{WS}
  - OS, an operating system, with no key
  - (WS as OS), the resulting node, with ephemeral public key K_n
  - bwl, a user, with public key K_{bwl}
  - K_{del}, an ephemeral public key for the node for bwl
  - C, a secure channel to a file server
  - TrustedNode and SysAdm, two groups

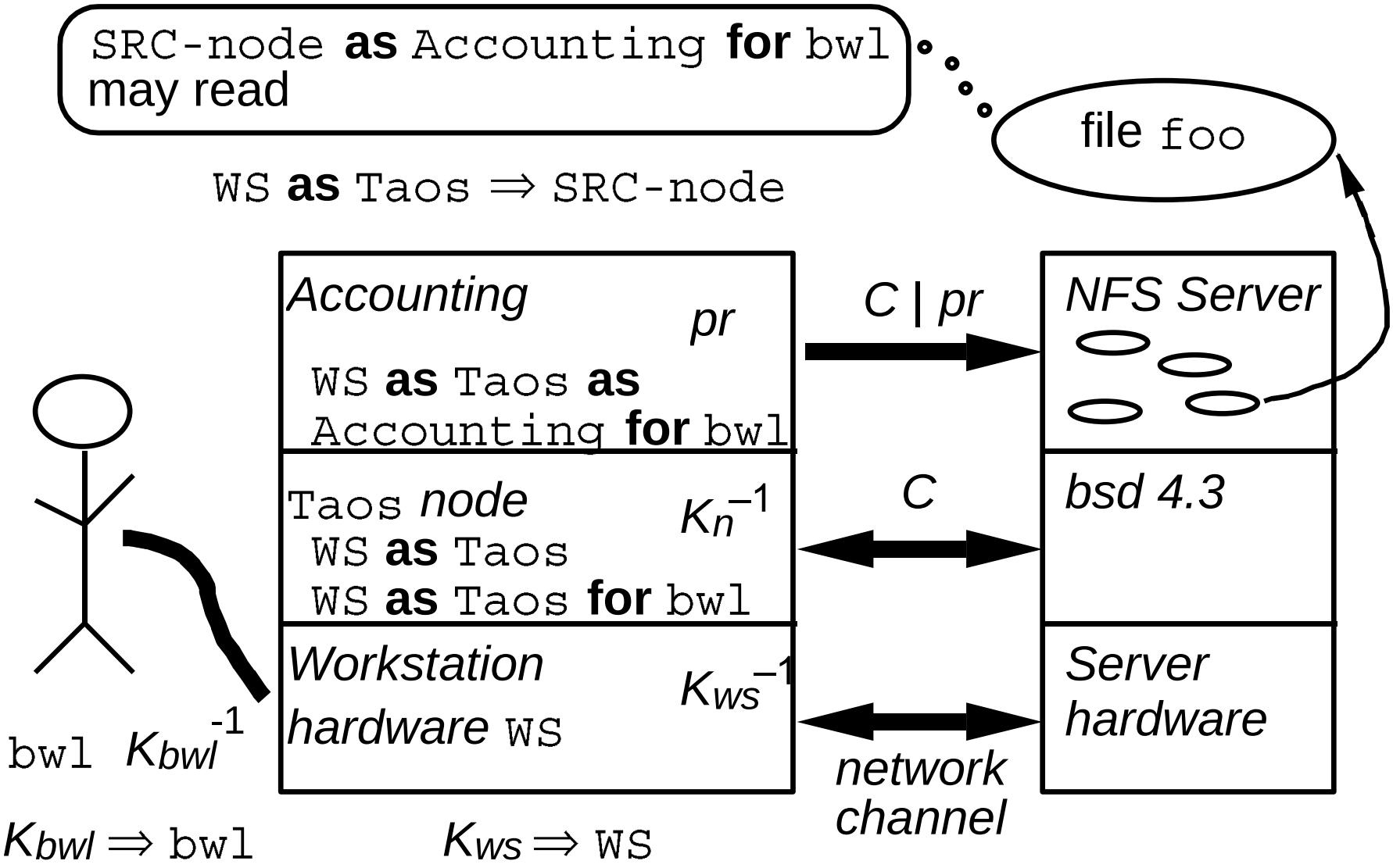
An example (cont.)

- K_{CA} says (K_{WS} speaks for WS)
- K_{WS} says (K_n speaks for (WS as OS))
- K_{CA} says (K_{bwl} speaks for bwl)
- K_{bwl} says (K_{del} speaks for ((WS as OS) for bwl))
- K_n says (K_{del} speaks for ((WS as OS) for bwl))
- K_{del} says (C speaks for ((WS as OS) for bwl))
- C says good-to-delete-file1
- And we may deduce:
  ((WS as OS) for bwl) says good-to-delete-file1

An example (cont.)

- K_{CA} says ((WS as OS) speaks for TrustedNode)
- K_{CA} says (bwl speaks for SysAdm)
- Then we may deduce:
  TrustedNode for SysAdm
  says good-to-delete-file1
- The ACL for file1 may say:
  TrustedNode for SysAdm
  controls good-to-delete-file1
- Then we conclude: good-to-delete-file1

Applications (1): Security in an operating system [Wobber et al., 1993]



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

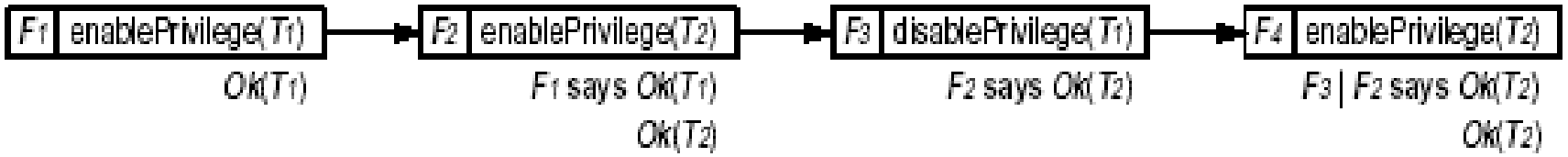
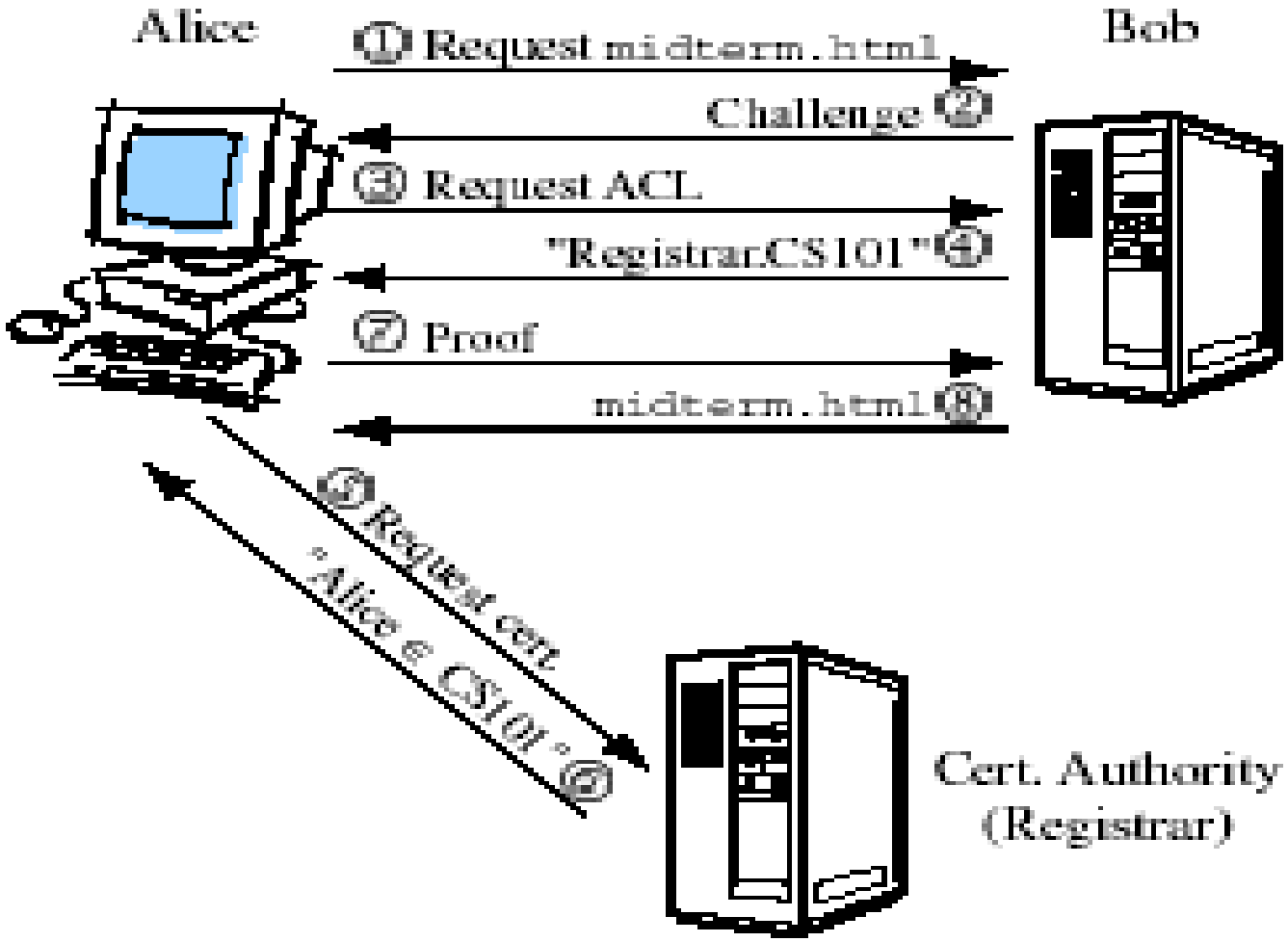


Figure 2: Example of interaction between stack frames. Each rectangle represents a stack frame. Each stack frame is labeled with its name. In this example, each stack frame makes one `enablePrivilege()` or `disablePrivilege()` call, which is also written inside the rectangle. Below each frame is written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

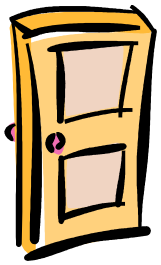
Jon

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] and Binder [DeTreville]
- Daisy [Cirillo et al.]
- SecPAL [Becker, Fournet, and Gordon] and DKAL [Gurevich and Neeman]

"says" and "speaks for" play a role in other systems:

- SDSI and SPKI [Lampson and Rivest; Ellison et al.]
- Alpaca [Lesniewski-Laas et al.]
- Aura [Vaughan et al.]
- Plan 9 [Pike et al.]

Binder

Binder

- Binder is a relative of Prolog.
- Like Datalog, it lacks function symbols.
- It also includes the special construct says.
- It does not include much else.

An example in Binder

- Facts
  - owns(Alice, Foo.txt).
  - Alice says good(Bob).
- Rules
  - may_access(p, o) :-
    owns(q, o), blesses(q, p).
  - blesses(Alice, p) :- Alice says good(p).
- Conclusions
  - may_access(Bob, Foo.txt).

Binder's proof rules

- Binder includes a standard resolution rule.
- In addition, Binder includes a rule for importing formulas from a context F to a context D.
  - The rule adds a "F says" in front of all atoms without a "says".
  - The rule applies only to clauses where the head atom does not have "says".

Binder's proof rules: example

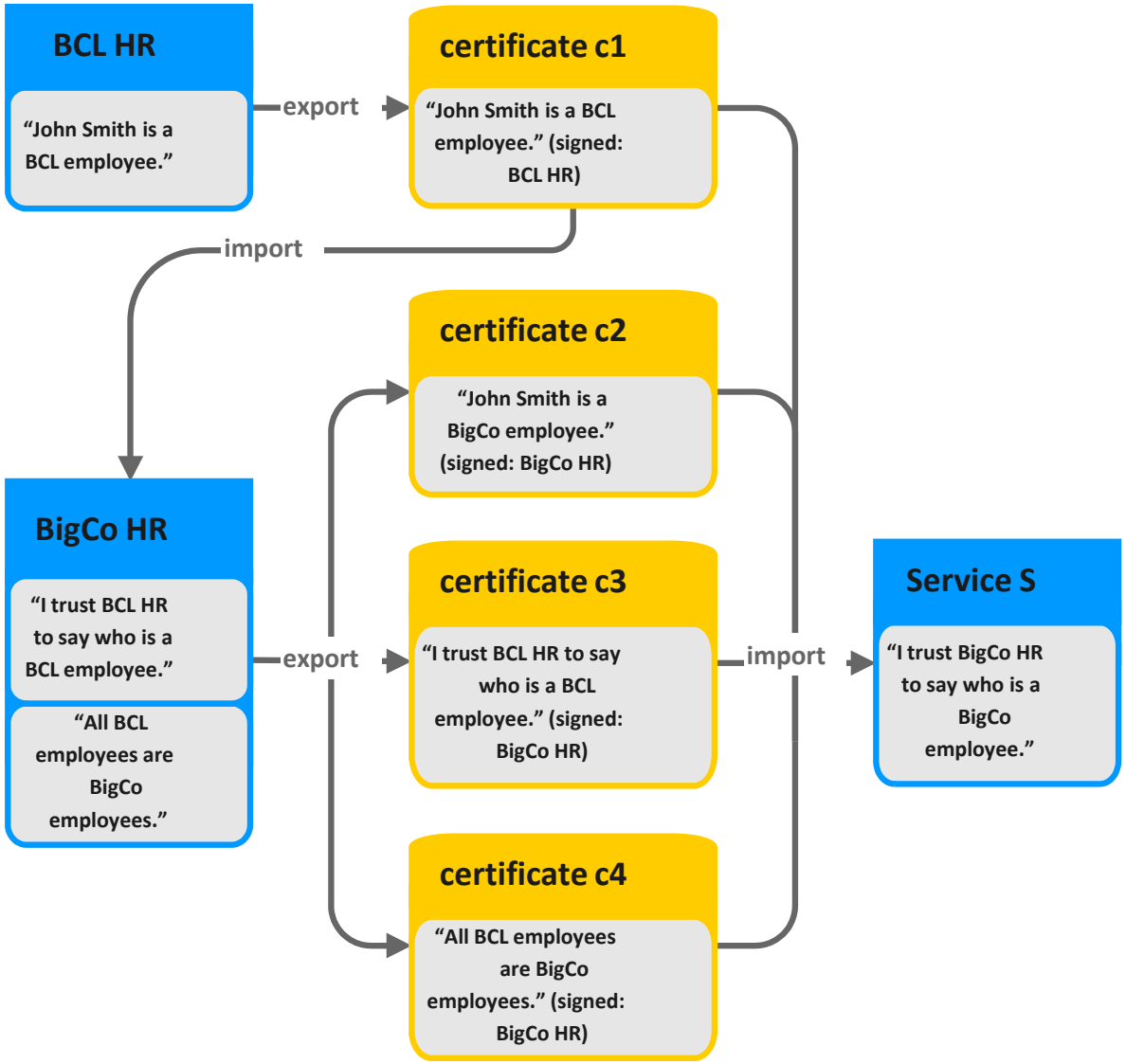
- Suppose F has the rules
  - may_access(p, o) :-
    owns(q, o), blesses(q, p).
  - blesses(Alice, p) :- Alice says good(p).
  - Alice says good(Bob).
- D may import the first two as:
  - F says may_access(p, o) :-
    F says owns(q, o), F says blesses(q, p).
  - F says blesses(Alice, p) :- Alice says good(p).
- D may import good(Bob) directly from Alice.

Binder's proof rules (cont.)

- Suppose F has the rule
  - blesses(Alice, p) :- Alice says good(p).
- D may import it as:
  - F says blesses(Alice, p) :- Alice says good(p).
- D and F should agree on Alice's identity.
- But the meaning of predicates may vary, and it typically will.
  For example, F may also have:
  - blesses(Bob, p) :- Bob says excellent(p).

Another example

[DeTreville]



A logical analysis

- Suppose that A has the rule:

  p :- B says q, r

- C would import this as:

  A says p :- B says q, A says r

- We may represent C's view of A's rule by:

  A says $((B \text{ says } q) \wedge r \rightarrow p)$

- We may represent C's conclusion by:

  $(B \text{ says } q) \wedge (A \text{ says } r) \rightarrow (A \text{ says } p)$

- How did we get here?

A logical analysis (cont.)

- So we assume:
  A says $((B \text{ says } q) \wedge r \rightarrow p)$
  and would like to derive:
  $(B \text{ says } q) \wedge (A \text{ says } r) \rightarrow (A \text{ says } p)$
- Assume the standard modal axiom
  A says $(s \rightarrow t) \rightarrow (A \text{ says } s) \rightarrow (A \text{ says } t)$
  and the necessitation rule.
- We obtain:
  A says $((B \text{ says } q) \wedge r) \rightarrow (A \text{ says } p)$
  and then (only!):
  $(A \text{ says } B \text{ says } q) \wedge (A \text{ says } r) \rightarrow (A \text{ says } p)$

A logical analysis (cont.)

- We can finish with the strong axiom:

  $s \rightarrow (A \text{ says } s)$

- A weaker form suffices:

  $B \text{ says } s \rightarrow (A \text{ says } B \text{ says } s)$

Important properties of Binder

- Binder programs can define and use new, application-specific predicates.
- A statement in Binder can be read as a declarative English sentence.
- Queries in Binder are decidable (in PTime).

- Questions:
  - Should there be more built-in syntax and semantics?
  - Can all reasonable policies be expressed?
    Can the simple ones be expressed simply enough?
  - What about other algorithmic problems?

Data integration

- A classic database problem is how to integrate multiple sources of data.
  - The sources may be heterogeneous. Their contents and structure may be partly unknown.
  - The data may be semi-structured (e.g., XML on the Web).

TSIMMIS and MSL [Garcia-Molina et al., mid 1990's]

- Wrappers translate between a common language and the native languages of sources.
- Mediators then give integrated views of data from multiple sources.
- The mediators may be written in the Mediator Specification Language (MSL).

```
<cs_person {<name N> <relation R> Rest1 Rest2}>@med :-
    <person {<name N> <dept `CS'> <relation R> |
                Rest1}>@whois
    AND decompose_name(N, LN, FN)
    AND <R {<first_name FN> <last_name LN> | Rest2}>@cs
```

Similarities

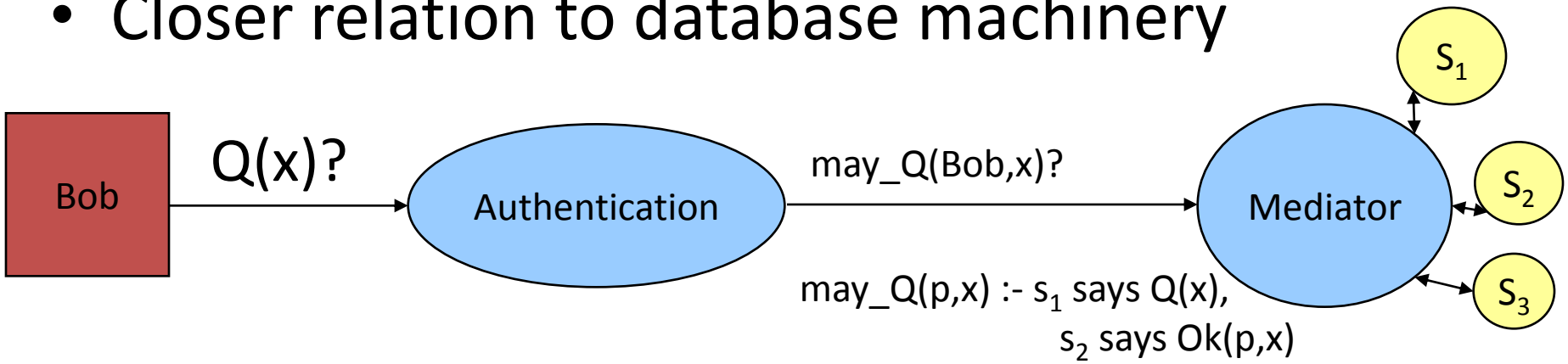
- MSL is remarkably similar to Binder.
  - They start from Datalog.
  - They add sites (or contexts).
  - X@s corresponds to s says X.
  - In X@s, the site s may be a variable.
- More broadly, distributed access control is partly about data integration.
  - Binder follows the "global as view" approach (GAV), in which each relation in the mediator schema is defined by a query over the data sources.
  - The converse "local as view" approach (LAV) might not be as meaningful for access control.

Caveats

- MSL and Binder are used in different environments and for different purposes.
  - Work in databases seems to focus on a messy but benign and tolerant world, full of opportunities.
  - Work in security deals with a hostile world and tries to do less.
- Security is primarily a property of systems, not of languages.
  Coincidences in languages go only so far.

Potential outcomes (speculation)

- Language-design ideas
  - Constructs beyond Datalog
  - Semi-structured data
- More theory, algorithms, tools
- Closer relation to database machinery

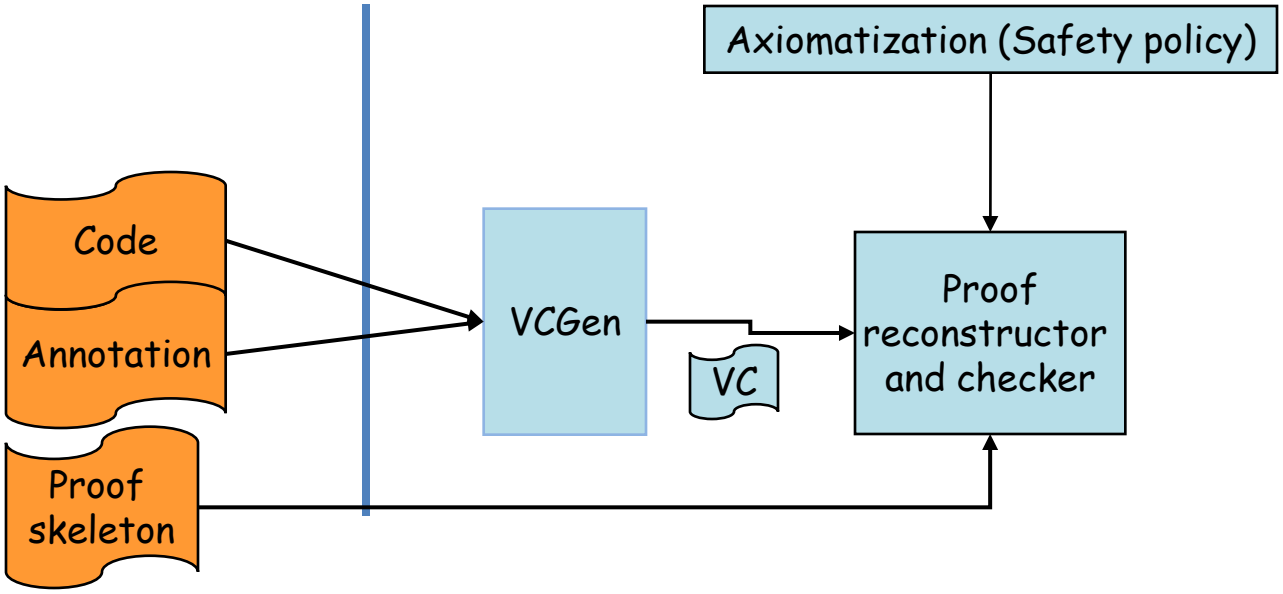


Proof-carrying code, proof-carrying authorization, …

Proof-carrying code (PCC)

[Necula and Lee, 1996]

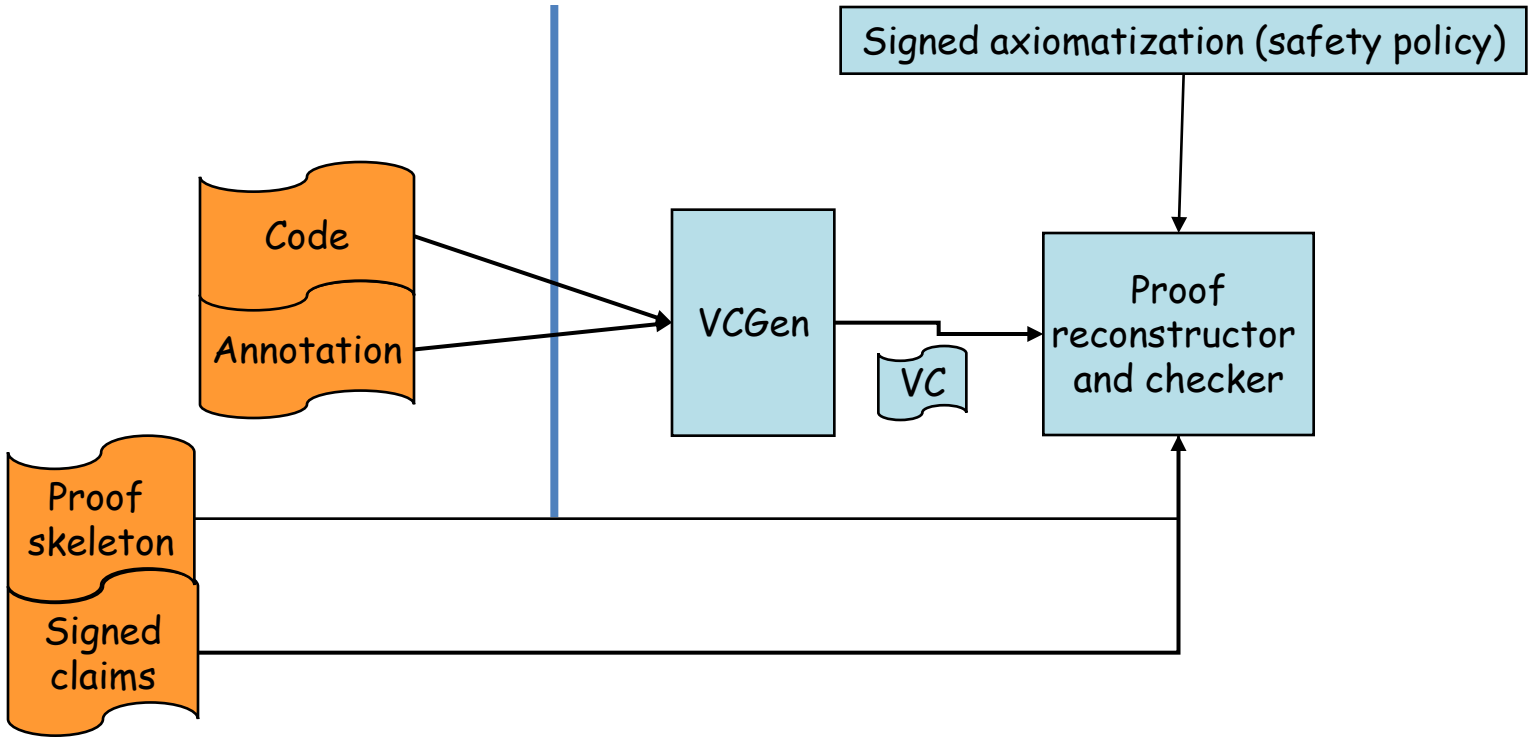
- Proof-carrying code is also based on logic.
- It is also essentially concerned with an authorization decision (running code):



Reason + Authority

[with Whitehead and Necula]

- How does PCC fit into the broader context of access control?
- How about hybrid policies and mechanisms?



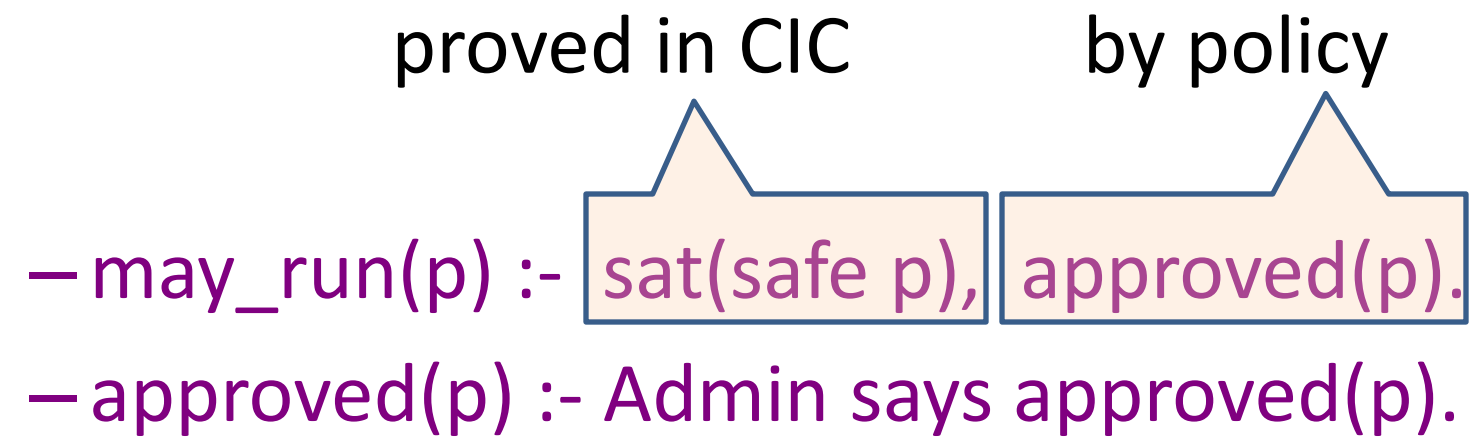
BCIC = Binder + CIC (Coq logic)

- An example of authorizing code execution:

  - may_run(p) :- sat(safe p), approved(p).

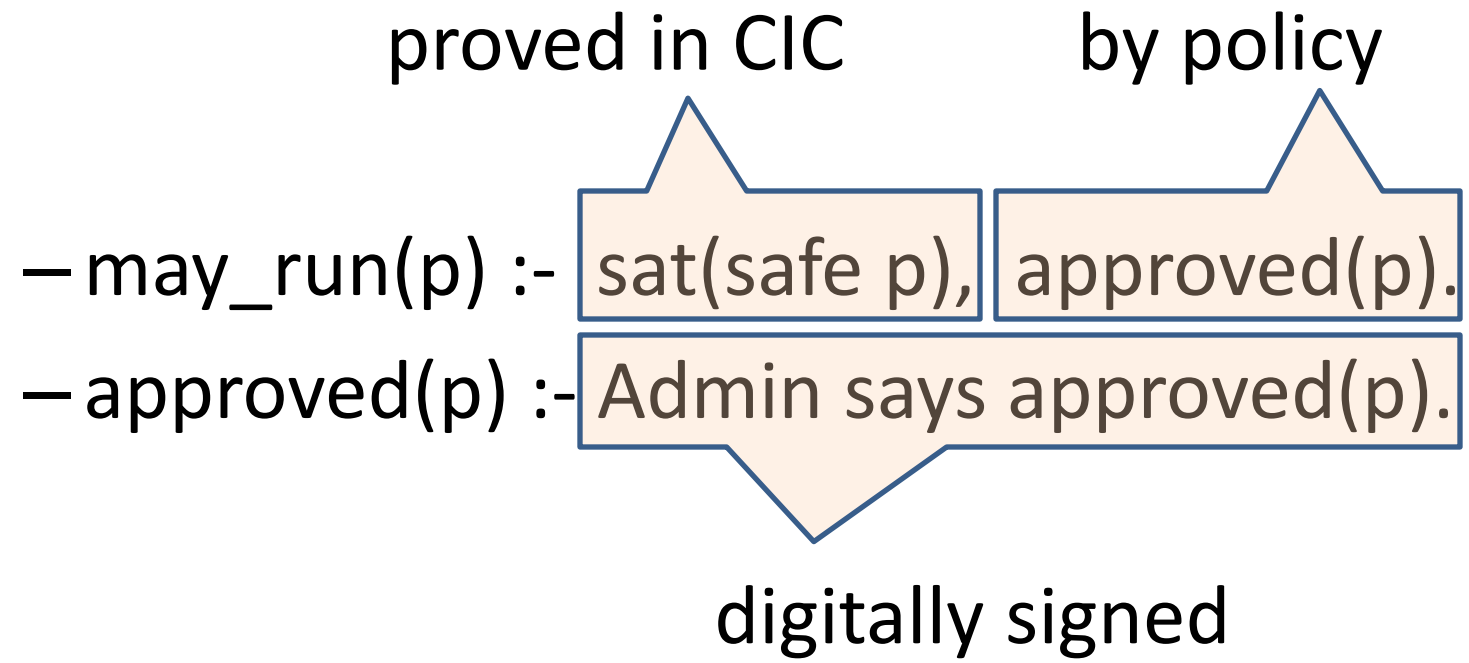
BCIC = Binder + CIC (Coq logic)

- An example of authorizing code execution:



BCIC = Binder + CIC (Coq logic)

- An example of authorizing code execution:



From proof-carrying to code-carrying

[with Maffeis, Fournet, and Gordon]

- Proofs are programs that can be presented as evidence with requests.
- Going further, programs provided by clients may ***do*** some of the access control.
  - In a pi calculus with higher-order features and dynamic typing.
  - With types for authorization.
  - Verifiably in compliance with policy.

Access control in a type system for tracking dependencies

Status and issues

- Calculi for access control have been applied in several languages and systems,
  (but are not in wide day-to-day use).
- 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).

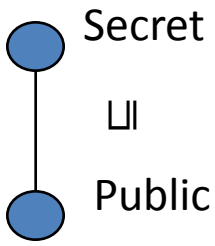
The Dependency Core Calculus

[with Banerjee, Heintze, and Riecke, 1999]

- A minimal but expressive calculus in which the types capture dependencies.
- A foundation for some static 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 j in L, there is a type $T_j(s)$.
- If $j \sqsubseteq k$ then terms of type $T_k(t)$ may depend on terms of type $T_j(s)$.

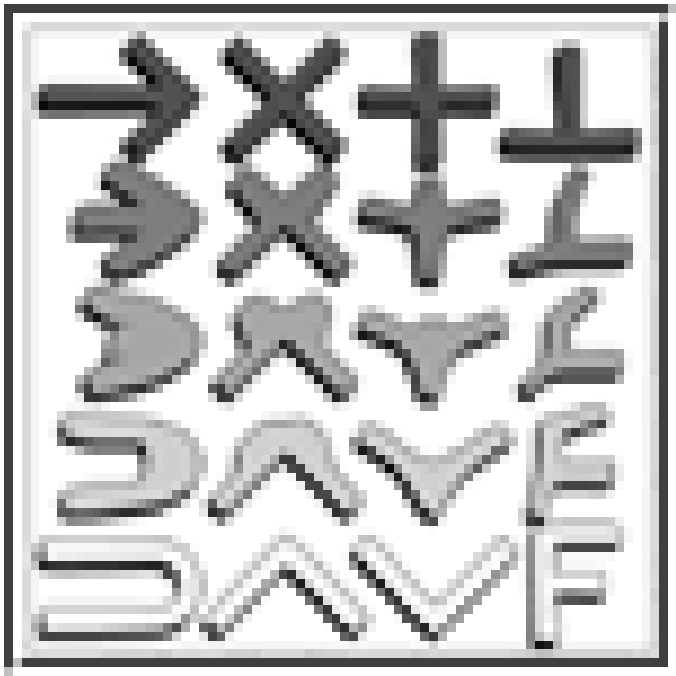


For instance:

- The lattice may have two elements:
- 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).
- This result is a non-interference theorem.

A new look at DCC

- We read DCC as a logic,
  via the ***Curry-Howard isomorphism***.
  - Types are propositions.
  - Programs are proofs.



A new look at DCC (cont.)

- We consider significant but routine variations on the original DCC:
  - We remove recursion.
  - We add polymorphism.
- We write A says s instead of $\mathsf{T}_A(s)$.
- We write A speaks for B as an abbreviation for $\forall X.\ (A \text{ says } X \rightarrow B \text{ says } X)$.

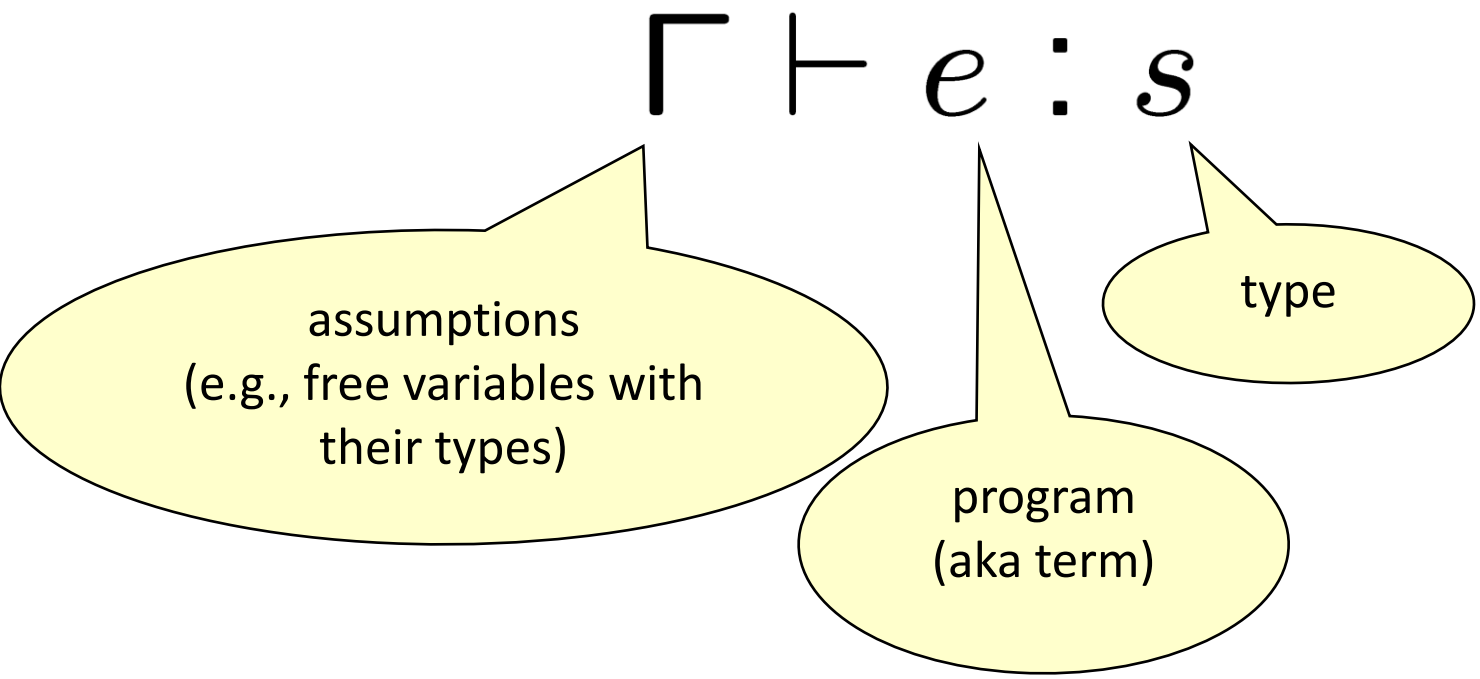
(This presentation omits the lattice aspects, and makes other small simplifications. This turns DCC into CDD.)

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 (not classical).
  - So it does not have “excluded middle”.
- Terms are proofs to be used in access control.

Typing rules

- As usual, typing rules are rules for deducing judgments (assertions) of the form:



The Simply Typed λ-calculus: rules

$$\Gamma, x : s, \Gamma' \vdash x : s$$

$$\Gamma \vdash () : \mathtt{true}$$

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

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

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

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

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

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

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

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

Logical reading

$$\Gamma, s, \Gamma' \vdash s$$

$$\Gamma \vdash \texttt{true}$$

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

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

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

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

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

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

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

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

Rules for says

- The rules are those of the λ-calculus plus:

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

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

- In other words:

$$\frac{\Gamma \vdash s}{\Gamma \vdash A\ \mathtt{says}\ s} \qquad \frac{\Gamma \vdash A\ \mathtt{says}\ s \qquad \Gamma, s \vdash A\ \mathtt{says}\ t}{\Gamma \vdash A\ \mathtt{says}\ t}$$

Rules for quantification

- We use the rules from System F, basically

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

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

Semantics

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

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

(Zdancewic checked subject reduction and progress properties in Twelf.)

- Denotational semantics? (We have some pieces. More could be done. See Abramsky and Jagadeesan; Kammar and Plotkin.)

Theorems

- We can rederive the core of the previous logics:

  $\vdash A \text{ says } (s \to t) \to (A \text{ says } s) \to (A \text{ says } t)$

  If $\vdash s$ then $\vdash A \text{ says } s$.

  $\vdash A \text{ speaks for } B \to (A \text{ says } s) \to (B \text{ says } s)$

  $\vdash A \text{ speaks for } A$

  $\vdash A \text{ speaks for } B \wedge B \text{ speaks for } C \to$
  $A \text{ speaks for } C$

Theorems (cont.)

- We obtain some additional useful theorems, including
  - $\vdash s \rightarrow (A \text{ says } s)$
  - $\vdash (A \text{ says } (B \text{ speaks for } A)) \rightarrow (B \text{ speaks for } A)$
- These follow from general rules, apparently without annoying consequences.

Another theorem

- $\forall X.\ A$ controls (A says $X \rightarrow B$ says X)

  $\rightarrow$

  A speaks for B

The value of this theorem is more debatable.

Non-theorems

- It does ***not*** follow that:
  $(A \text{ says } s) \rightarrow s \vee (A \text{ says false})$

Non-theorems

- It does ***not*** follow that:
  $(A \text{ says } s) \rightarrow s \vee (A \text{ says false})$

This would trivialize “says”.

Non-theorems

- It does ***not*** follow that:
  $(A \text{ says } s) \rightarrow s \vee (A \text{ says false})$

This would trivialize "says".

- ***Nor*** does it follow that control is monotonic:
  $(s \rightarrow t) \rightarrow (A \text{ controls } s) \rightarrow (A \text{ controls } t)$

Non-theorems

- It does ***not*** follow that:
  $(A \text{ says } s) \rightarrow s \vee (A \text{ says false})$

This would trivialize "says".

- ***Nor*** does it follow that control is monotonic:
  $(s \rightarrow t) \rightarrow (A \text{ controls } s) \rightarrow (A \text{ controls } t)$

What about Least Privilege?

Non-theorems

- It does ***not*** follow that:
  $(A \text{ says } s) \rightarrow s \vee (A \text{ says false})$

This would trivialize "says".

- ***Nor*** does it follow that control is monotonic:
  $(s \rightarrow t) \rightarrow (A \text{ controls } s) \rightarrow (A \text{ controls } t)$

What about Least Privilege?

- Both would follow in classical logic.
- Both are equivalent classically,
  but not intuitionistically.

Metatheory

- We also obtain a useful metatheory, including:
  - old and new non-interference results,
  - various interpretations in other systems.
- Thus, we can provide at least partial evidence of the “goodness” of our rules.

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 effects 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 $(s)^B$ by:

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

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 $(\Gamma)^B \vdash e' : (s)^B$.

Non-interference (a special case)

If A and B are distinct, then

$$\not\vdash (B \ \mathbf{says}\ t) \rightarrow (A \ \mathbf{says}\ \forall X.\, X)$$

If s does not mention B and

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

then

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

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

Another good property: simple reasoning with "speaks for"

- We would like:

  if *s* is a formula with "speaks for" but
  no other use of quantifiers, then
  s is provable using propositional rules
  for "speaks for"
  if and only if
  s is provable using the definition
  A speaks for *B* = $\forall X.$ (*A* says $X \rightarrow B$ says X)

- So reasoning about "speaks for" does not require hard reasoning with quantifiers.

Three systems

- ICL = intuitionistic propositional logic + "says"
  with the axioms and rules just given
- ICL⇒ = ICL + primitive "speaks for"
  with the axioms given, i.e.,
  - *A* speaks for *B* → (*A* says *s*) → (*B* says *s*)
  - reflexivity + transitivity of "speaks for"
  - (*A* says (*B* speaks for *A*)) → (*B* speaks for *A*)
- CDD = ICL + ∀
  with the usual rules for ∀

Translation to S4

[with Garg]

- We translate to the classical S4 modal logic. We write $\mathcal{T}(s)$ for the translation of s.
- The translation of propositional connectives consists in adding boxes [Gödel]:

  $\mathcal{T}(p) = \Box\, p$

  $\mathcal{T}(s \to t) = \Box\, (\mathcal{T}(s) \to \mathcal{T}(t))$

- For "says" and primitive "speaks for", we set:

  $\mathcal{T}(A \text{ says } s) = \Box\, (A \vee \mathcal{T}(s))$

  $\mathcal{T}(A \text{ speaks for } B) = \Box\, (A \to B)$

  where A and B are proposition symbols in S4.

Translation to S4 (cont.)

- This translation is sound and complete for ICL and ICL⇒.
- It follows that ICL and ICL⇒ are decidable in PSPACE [via Ladner].
- We obtain possible-worlds semantics.
- Furthermore, the translation suggests a nice logic with Boolean operations on principals.
  - Again, even when *A* is a Boolean expression, we set: $\mathcal{T}(A \text{ says } s) = \Box\,(A \vee \mathcal{T}(s))$

Translation to S4 (cont.)

- We also obtain that ICL⇒ is sound and complete for its fragment of CDD:

  if *s* is a formula in ICL⇒, then

  s is provable in ICL⇒

  if and only if

  s is provable in CDD

  expanding *A* speaks for *B* to

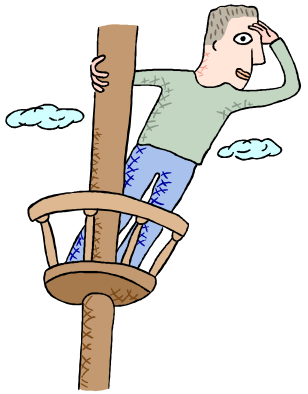
  $\forall X.\ (A \text{ says } X \rightarrow B \text{ says } X)$

- So reasoning about "speaks for" does not require hard reasoning with quantifiers.

Further work and open questions

- Rich, convenient languages for policies.
- More semantics.
- Integrating access control into programming.
- Relation to information-flow control.

Conclusions



- Big stakes.
- A growing body of sophisticated approaches
  - with diverse ideas and techniques from programming languages;
  - as explanations or as possible substitutes for more ad hoc methods.

Questions?