

Access Control in a Core Calculus of Dependencies

Daniel K. Lee
15-819

Fall, 2007

Dependency Core Calculus

- ▶ “A Core Calculus Of Dependency”
Abadi, M. and Banerjee, A. and Heintze, N. and Riecke, J. G.
POPL, 1999
 - ▶ Extension of Moggi’s computational lambda calculus.
 - ▶ Uses lax modalities to model program dependencies such as information flow, binding-time analysis, information flow, etc.
- ▶ “Access Control in a Core Calculus of Dependency”
Abadi, M.
Theoretical Computer Science, 2007
 - ▶ Used DCC to describe access control in distributed systems.

Dependency Core Calculus

- ▶ A programming language that uses types to distinguish dependencies in data.
- ▶ Used to model program dependencies such as *information flow*.
- ▶ Predetermined lattice of information levels.
- ▶ Uses monad $T_\ell(s)$ to denote computations at information level ℓ .

Access Control in DCC

- ▶ By using principals as information levels, DCC becomes an access control calculus.
- ▶ Constructive proof system: a type system for evidence.
 - ▶ Proof witnesses (lambda terms) as evidence for access.
 - ▶ First time Curry-Howard isomorphism used for access control.
(?)
- ▶ Simply Typed DCC captures many necessary properties for access control, but a number of desirable properties require Polymorphic DCC.

Principals in Simply Typed DCC

Principals:

$$A, B, C, \dots \in \mathcal{L}$$

- ▶ Elements of lattice \mathcal{L} are principals.
- ▶ Order induced by a relation \sqsubseteq
- ▶ Elements lower in the lattice are more trusted.

Types in Simply Typed DCC

Types:

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

- ▶ `true` is a unit type.
- ▶ `A says s` is a *lax modality*.

Types protected at level A

For each $A \in \mathcal{L}$, the **says** operation induces a subset of types called the *types protected at level A*.

Types protected at level A

For each $A \in \mathcal{L}$, the **says** operation induces a subset of types called the *types protected at level A* .

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

Types protected at level A

For each $A \in \mathcal{L}$, the **says** operation induces a subset of types called the *types protected at level A* .

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

Types protected at level A

For each $A \in \mathcal{L}$, the **says** operation induces a subset of types called the *types protected at level A* .

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

Types protected at level A

For each $A \in \mathcal{L}$, the **says** operation induces a subset of types called the *types protected at level A* .

- ▶ If $A \sqsubseteq B$, then B **says** s is protected at level A
- ▶ **true** is protected at level A .
- ▶ If t is protected at level A , then B **says** t is protected at level A
- ▶ If s and t are protected at level A , then $(s \wedge t)$ is protected at level A .

Types protected at level A

For each $A \in \mathcal{L}$, the **says** operation induces a subset of types called the *types protected at level A* .

- ▶ If $A \sqsubseteq B$, then **B says s** is protected at level A
- ▶ **true** is protected at level A .
- ▶ If t is protected at level A , then **B says t** 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 **$(s \rightarrow t)$** is protected at level A .

Terms in Simply Typed DCC

Terms:

$$e ::= \dots \mid () \mid (\eta_A e) \mid \text{bind } x = e \text{ in } e'$$

- ▶ $()$ intro-form for type **true**.
- ▶ Usual intro- and elim-forms for \wedge , \vee , and \rightarrow .
- ▶ η_A and **bind** are the monad operations of the **says** modality.

Typing Rules of Simply Typed DCC

Typing Rules:

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

Typing Rules of Simply Typed DCC

Typing Rules:

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

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

- ▶ Side condition on bind rule restricts what we can use terms of type $A \text{ says } s$ to prove.

Logical Reasoning in Simply Typed DCC

- ▶ $\Gamma \vdash s$ when there exists an e such that $\Gamma \vdash e : s$ is derivable.
- ▶ Uses a call-by-name operational semantics.
- ▶ Denotational semantics exists, but not really used in access control paper.

Access Control in Simply Typed DCC

Suppose s is a formula that represents that a certain operation o should be performed:

- ▶ Write s as $Do(o)$.
- ▶ Reasoning is from the perspective of a reference monitor that is in charge of making access control decisions for o .

Access Control in Simply Typed DCC

Suppose s is a formula that represents that a certain operation o should be performed:

- ▶ Write s as $Do(o)$.
- ▶ Reasoning is from the perspective of a reference monitor that is in charge of making access control decisions for o .
- ▶ Represent that principal A can perform o as $(A \text{ says } Do(o)) \rightarrow Do(o)$.

Access Control in Simply Typed DCC

Suppose s is a formula that represents that a certain operation o should be performed:

- ▶ Write s as $Do(o)$.
- ▶ Reasoning is from the perspective of a reference monitor that is in charge of making access control decisions for o .
- ▶ Represent that principal A can perform o as $(A \text{ says } Do(o)) \rightarrow Do(o)$.
- ▶ Represent a request from principal B to perform o as $B \text{ says } Do(o)$.

Access Control in Simply Typed DCC

Suppose s is a formula that represents that a certain operation o should be performed:

- ▶ Write s as $Do(o)$.
- ▶ Reasoning is from the perspective of a reference monitor that is in charge of making access control decisions for o .
- ▶ Represent that principal A can perform o as $(A \text{ says } Do(o)) \rightarrow Do(o)$.
- ▶ Represent a request from principal B to perform o as $B \text{ says } Do(o)$.
- ▶ Reference monitor may try to prove these formulas imply $Do(o)$ and grant access if this is possible. Or could check a proof given by B .

Properties of says

Can produce proof terms for:

- ▶ $\vdash s \rightarrow A \text{ says } s$
- ▶ $\vdash (A \text{ says } (s \rightarrow s')) \rightarrow ((A \text{ says } s) \rightarrow (A \text{ says } s'))$

These exist as axioms or rules in other access control logics.
Usual intuitionistic properties about \vee , \wedge , and \rightarrow still hold.

Properties of says

Can also produce proof terms for:

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

These properties say something about reasoning about chains of **says**, e.g. $A \text{ says } B \text{ says } A \text{ says } C \text{ says } s$: order and multiple appearances in chain do not matter.

Partial Order and “speaks for”

- ▶ Lower elements of lattice are more trusted.
- ▶ Can show $\vdash (A \text{ says } s) \rightarrow (B \text{ says } s)$ when $A \sqsubseteq B$.

Partial Order and “speaks for”

- ▶ Lower elements of lattice are more trusted.
- ▶ Can show $\vdash (A \text{ says } s) \rightarrow (B \text{ says } s)$ when $A \sqsubseteq B$.
- ▶ Could regard $A \sqsubseteq B$ as a representation for A speaksfor B .
- ▶ However, Simply Typed DCC is not rich enough to reason *about* $A \sqsubseteq B$, because it is not a type. e.g. $A \text{ says } (B \sqsubseteq A)$ is not syntactically valid.

Partial Order and “speaks for”

- ▶ Lower elements of lattice are more trusted.
- ▶ Can show $\vdash (A \text{ says } s) \rightarrow (B \text{ says } s)$ when $A \sqsubseteq B$.
- ▶ Could regard $A \sqsubseteq B$ as a representation for A *speaksfor* B .
- ▶ However, Simply Typed DCC is not rich enough to reason *about* $A \sqsubseteq B$, because it is not a type. e.g. $A \text{ says } (B \sqsubseteq A)$ is not syntactically valid.
- ▶ Instead of just adding $B \sqsubseteq A$ as a type, polymorphism can be used to encode and reason about *speaksfor*.

Meets and Joins for Combining Principals

- ▶ Since $A \sqcap B \sqsubseteq A$ and $A \sqcap B \sqsubseteq B$:
 $(A \sqcap B \text{ says } s) \rightarrow (A \text{ says } s \wedge B \text{ says } s)$

Meets and Joins for Combining Principals

- ▶ Since $A \sqcap B \sqsubseteq A$ and $A \sqcap B \sqsubseteq B$:
 $(A \sqcap B \text{ says } s) \rightarrow (A \text{ says } s \wedge B \text{ says } s)$
- ▶ Since $A \sqsubseteq A \sqcup B$ and $A \sqsubseteq A \sqcup B$:
 $(A \text{ says } s \vee B \text{ says } s) \rightarrow (A \sqcup B \text{ says } s)$
- ▶ The converses of these statements are not in general true.

Quoting

- ▶ $A \mid B$ is a principal representing A “quoting” B .
- ▶ In the presence of quoting, want the property that $A \mid B$ says s is equivalent to A says B says s .

Quoting

- ▶ $A | B$ is a principal representing A “quoting” B .
- ▶ In the presence of quoting, want the property that $A | B \text{ says } s$ is equivalent to $A \text{ says } B \text{ says } s$.
- ▶ The paper suggests $A \sqcup B$ as a candidate for $A | B$.
- ▶ This change would require that $A \sqcup B \text{ says } s \rightarrow A \text{ says } B \text{ says } s$, which would require significant changes to Simply Typed DCC.

Polymorphic DCC

- ▶ Simply Typed DCC extended with constructs for second-order polymorphism.
- ▶ Main benefit: addresses inability to reason about *A speaksfor B* as a statement in Simply Typed DCC.

Polymorphic DCC

New Types:

$$s ::= \dots \mid X \mid \forall X.s$$

- ▶ Introduce type variables, and quantification over types.
- ▶ Definition of protected at level A must be extended:
If t is protected at level A , then $\forall X.t$ is protected at level A .

Polymorphic DCC

New Typing Rules:

$$\frac{\Gamma, X \vdash e : s}{\Gamma \vdash \lambda X. s : \forall X. s}$$

Polymorphic DCC

New Typing Rules:

$$\frac{\Gamma, X \vdash e : s}{\Gamma \vdash \lambda X. s : \forall X. s}$$

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

Access Control in Polymorphic DCC

- ▶ Write $A \Rightarrow B$ as an abbreviation for “ A speaks for B ”.
- ▶ $A \Rightarrow B$ is an abbreviation for $\forall X. A \text{ says } X \rightarrow B \text{ says } X$
- ▶ Obtain fundamental property that for every s :
 $\vdash (A \Rightarrow B) \rightarrow ((A \text{ says } s) \rightarrow (B \text{ says } s))$

The Hand-off Property

- ▶ The hand-off property is the property that if A says $(B \Rightarrow A)$ then $B \Rightarrow A$.
- ▶ Often added as an axiom in other formulations of access control.

The Hand-off Property

- ▶ The hand-off property is the property that if A says $(B \Rightarrow A)$ then $B \Rightarrow A$.
- ▶ Often added as an axiom in other formulations of access control.
- ▶ Because $B \Rightarrow A$ is protected at level A , the “hand-off axiom” is derivable:
 $\vdash (A \text{ says } (B \Rightarrow A)) \rightarrow (B \Rightarrow A)$

The Hand-off Property

- ▶ The hand-off property is the property that if A says $(B \Rightarrow A)$ then $B \Rightarrow A$.
- ▶ Often added as an axiom in other formulations of access control.
- ▶ Because $B \Rightarrow A$ is protected at level A , the “hand-off axiom” is derivable:
$$\vdash (A \text{ says } (B \Rightarrow A)) \rightarrow (B \Rightarrow A)$$
- ▶ $A \sqsubseteq B$ implies $A \Rightarrow B$, but the converse is not true.
- ▶ There is some separation and redundancy between \sqsubseteq and \Rightarrow . Is this good or bad?

Example: Access with a Simple Hand-Off

Suppose we have:

- ▶ $A \text{ says } (B \Rightarrow A)$
- ▶ $B \text{ says } Do(o)$
- ▶ $(A \text{ says } Do(o)) \rightarrow Do(o)$

The following is derivable in DCC:

$$\vdash ((A \text{ says } (B \Rightarrow A)) \wedge (B \text{ says } Do(o)) \wedge ((A \text{ says } Do(o)) \rightarrow Do(o))) \rightarrow Do(o)$$

Example: Proof-Carrying Calls

- ▶ Suppose o is an operation that takes in an integer and returns an integer. The interface for performing o should be:
 $Do(o) \rightarrow \text{int} \rightarrow \text{int}$
- ▶ To invoke o , a proof of $Do(o)$ is required.

Example: Proof-Carrying Calls

- ▶ Suppose o is an operation that takes in an integer and returns an integer. The interface for performing o should be:

$Do(o) \rightarrow \text{int} \rightarrow \text{int}$

- ▶ To invoke o , a proof of $Do(o)$ is required.
- ▶ Based on previous slide, we need proofs of $A \text{ says } (B \Rightarrow A)$, $B \text{ says } Do(o)$, and $(A \text{ says } Do(o)) \rightarrow Do(o)$.
- ▶ Proofs of $A \text{ says } (B \Rightarrow A)$ and $B \text{ says } Do(o)$ should be digitally signed statements that checked with the principal's public key verifies they made the statement.
- ▶ Proof of $(A \text{ says } Do(o)) \rightarrow Do(o)$ should come from an authority trusted on o .

Translation to System F

There exists a simple translation from Polymorphic DCC is a result of its translation, $(\cdot)^F$, into System F.

$$(A \text{ says } s)^F = (s)^F$$

$$(\eta_A e)^F = (e)^F$$

$$(\text{bind } x = e \text{ in } e')^F = (\lambda x : t.(e')^F) (e)^F$$

Metatheory

- ▶ *Theorem:* If $\Gamma \vdash e : s$ in Polymorphic DCC, then there exists e' such that $(G)^F \vdash e' : (t')$ in System F.
Proof by induction over $\Gamma \vdash e : s$.

Metatheory

- ▶ *Theorem*: If $\Gamma \vdash e : s$ in Polymorphic DCC, then there exists e' such that $(G)^F \vdash e' : (t')$ in System F.
Proof by induction over $\Gamma \vdash e : s$.
- ▶ *Theorem* (Consistency): In polymorphic DCC, it is neither the case that $\vdash \forall X.X$ nor that $\vdash A \text{ says } \forall X.X$.
Proof by composing previous theorem with consistency of System F.

Noninterference

- ▶ Informally, noninterference is the property that if we have a proof e of A says s and it depends on a proof x of B says t , where A and B are unrelated principals, it does not matter what actual proof is substituted for x .

Noninterference

- ▶ Informally, noninterference is the property that if we have a proof e of A says s and it depends on a proof x of B says t , where A and B are unrelated principals, it does not matter what actual proof is substituted for x .
- ▶ In order to state the non-interference theorem, we must first define a function $(\cdot)^B$.

$$(A \text{ says } s)^B = \text{true} \text{ if } B \sqsubseteq A$$

$$(A \text{ says } s)^B = A \text{ says } (s)^B \text{ otherwise}$$

The intuition is that $(s)^B$ is a variant of s that corresponds to when B is completely untrustworthy.

Noninterference

- ▶ *Theorem:* In Polymorphic DCC, for every type s and $B \in \mathcal{L}$, if $\vdash s$, then $\vdash (s)^B$.
 - ▶ Proof in paper.

Noninterference

- ▶ *Theorem:* In Polymorphic DCC, for every type s and $B \in \mathcal{L}$, if $\vdash s$, then $\vdash (s)^B$.
 - ▶ Proof in paper.
- ▶ Suppose $B \not\sqsubseteq A$, by previous theorem:
 $\vdash (B \text{ says } t) \rightarrow (A \text{ says } \forall X.X)$
implies
 $\vdash \text{true} \rightarrow (A \text{ says } \forall X.X)$
and therefore
 $(A \text{ says } \forall X.X)$
which is impossible, by consistency.

Noninterference

Suppose $B \not\sqsubseteq A$

- ▶ If $(s)^B = s$, then
 $\vdash (B \text{ says } t) \rightarrow (A \text{ says } s)$
implies
 $A \text{ says } s$
- ▶ If $(s)^B \neq s$, then
 $\vdash (B \text{ says } t) \rightarrow (A \text{ says } s)$
implies
 $A \text{ says } (s)^B$

Fin

Questions?

Information Flow

- ▶ DCC was originally used as a target of translation from existing information flow calculi.
- ▶ Instead of principals, used information levels as its lattice.
- ▶ High security was at top of lattice, low security was at bottom.
- ▶ Type system enforces that high security values cannot be used to create low security values, via the typing rule for bind.
- ▶ Translations given from information flow calculi such as the SLam calculus and the Smith-Volpano Calculus into DCC.