Game Semantics for Access Control

Samson Abramsky
Oxford University Computing Laboratory
Joint work with Radha Jagadeesan, DePaul University
Paper in MFPS XXV, Oxford April 3–7 2009

Access Control

Access Control

‘Control’ — *i.e.* policies for *restricting* access to informatic resources.

Access Control

‘Control’ — *i.e.* policies for *restricting* access to informatic resources.

Idea (since 1970’s): *Security Lattice* $\mathcal{L}$.

Often generated by poset (Principals, $<$).

Access Control

'Control' — *i.e.* policies for *restricting* access to informatic resources.

Idea (since 1970's): *Security Lattice* $\mathcal{L}$.
Often generated by poset (Principals, $<$).

Reading of $\ell < \ell'$:

$$\ell \text{ is at a } \{\text{higher/lower}\}\ \{\text{security/authorization}\} \text{ level than } \ell'$$

Access Control

'Control' — *i.e.* policies for *restricting* access to informatic resources.

Idea (since 1970's): *Security Lattice* $\mathcal{L}$.
Often generated by poset (Principals, $<$).

Reading of $\ell < \ell'$:

$$\ell \text{ is at a } \{\text{higher/lower}\} \ \{\text{security/authorization}\} \text{ level than } \ell'$$

Traditional: Hi and Lo (variables, procedures/actions).

'Control' — *i.e.* policies for *restricting* access to informatic resources.

Idea (since 1970's): *Security Lattice* $\mathcal{L}$.
Often generated by poset $($Principals, $<)$.

Reading of $\ell < \ell'$:

ℓ is at a $\{$higher/lower$\}$ $\{$security/authorization$\}$ level than ℓ'

Traditional: Hi and Lo (variables, procedures/actions).

A Hi thread can access a Lo resource, but not vice versa.

Access Control

'Control' — *i.e.* policies for *restricting* access to informatic resources.

Idea (since 1970's): *Security Lattice* $\mathcal{L}$.
Often generated by poset (Principals, $<$).

Reading of $\ell < \ell'$:

> ℓ is at a $\{$higher/lower$\}$ $\{$security/authorization$\}$ level than ℓ'

Traditional: Hi and Lo (variables, procedures/actions).

> A Hi thread can access a Lo resource, but not vice versa.

Then $\ell < \ell'$ means that ℓ is (relatively) Lo and ℓ' is (relatively) Hi.

Authorization Logic (Abadi, Pfenning, Garg et al.)

Authorization Logic (Abadi, Pfenning, Garg et al.)

Basic notion 'ℓ says ϕ'.

ϕ is uttered at Authorization level ℓ.

Basic notion 'ℓ says ϕ'.
ϕ is uttered at Authorization level ℓ.

In this reading, the underlying principle we want to enforce is:

> No proof of a formula of the form "P says ϕ" can make any essential use of formulas of the form "Q says ψ" unless Q is at the same or higher security level as P. In other words, we cannot rely on a lower standard of "evidence" or authorization in passing to a higher level.

Basic notion 'ℓ says ϕ'.
ϕ is uttered at Authorization level ℓ.

In this reading, the underlying principle we want to enforce is:

> No proof of a formula of the form "P says ϕ" can make any essential use of formulas of the form "Q says ψ" unless Q is at the same or higher security level as P. In other words, we cannot rely on a lower standard of "evidence" or authorization in passing to a higher level.

In this context, it is natural to read the security lattice in the opposite direction!

Formalization (Abadi, Pfenning, Garg et al.)

Formalization (Abadi, Pfenning, Garg et al.)

Take a standard type theory — could be a typed λ-calculus or a Linear version — as a base.

Formalization (Abadi, Pfenning, Garg et al.)

Take a standard type theory — could be a typed λ-calculus or a Linear version — as a base.
We may think of this as a programming language (in which case it will have features such as recursion), or as a logical calculus.

Formalization (Abadi, Pfenning, Garg et al.)

Take a standard type theory — could be a typed λ-calculus or a Linear version — as a base.
We may think of this as a programming language (in which case it will have features such as recursion), or as a logical calculus.

We then extend this with a family of monads T_ℓ, indexed by elements of the security lattice $\mathcal{L}$. Some additional axioms are given relating these monads.

Take a standard type theory — could be a typed λ-calculus or a Linear version — as a base.
We may think of this as a programming language (in which case it will have features such as recursion), or as a logical calculus.

We then extend this with a family of monads T_ℓ, indexed by elements of the security lattice $\mathcal{L}$. Some additional axioms are given relating these monads.

In the Authorization Logic context, we read $T_\ell A$ directly as 'ℓ says A'.

Formalization (Abadi, Pfenning, Garg et al.)

Take a standard type theory — could be a typed λ-calculus or a Linear version — as a base.
We may think of this as a programming language (in which case it will have features such as recursion), or as a logical calculus.

We then extend this with a family of monads T_ℓ, indexed by elements of the security lattice $\mathcal{L}$. Some additional axioms are given relating these monads.

In the Authorization Logic context, we read $T_\ell A$ directly as 'ℓ says A'.

In the flow or dependency analysis context, $T_\ell A$ is 'wrapping' the type A in a protection level ℓ, and hence preventing objects of that type being accessed by lower-level sub-computations.

Formalization (Abadi, Pfenning, Garg et al.)

Take a standard type theory — could be a typed λ-calculus or a Linear version — as a base.
We may think of this as a programming language (in which case it will have features such as recursion), or as a logical calculus.

We then extend this with a family of monads T_ℓ, indexed by elements of the security lattice $\mathcal{L}$. Some additional axioms are given relating these monads.

In the Authorization Logic context, we read $T_\ell A$ directly as 'ℓ says A'.

In the flow or dependency analysis context, $T_\ell A$ is 'wrapping' the type A in a protection level ℓ, and hence preventing objects of that type being accessed by lower-level sub-computations.

The main results are *non-interference theorems*, stating that the desired restrictions are enforced by the type system.

A Small Example (Garg and Abadi)

Let there be two principals, Bob (a user) and admin (standing for administration). Let dfile stand for the proposition that a certain file should be deleted. Consider the collection of assertions:

1. (admin says dfile) $\Rightarrow$ dfile
2. admin says ((Bob says dfile) $\Rightarrow$ dfile)
3. Bob says dfile

Using the unit of the monad with (3) yields (admin says (Bob says dfile)). Using modal consequence with (2) yields:

- (admin says (Bob says dfile)) $\Rightarrow$ (admin says dfile)

dfile now follows using modus ponens.

Our Approach

Our Approach

- Previous work in this area has been syntactic in nature. Natural models of these notions have not been forthcoming. Non-interference results are proved syntactically.

Our Approach

- Previous work in this area has been syntactic in nature. Natural models of these notions have not been forthcoming. Non-interference results are proved syntactically.

- We take a semantic approach. We show that Game Semantics provides an intuitive and illuminating account of access control, and moreover leads to strikingly simple and robust proofs of interference-freedom.

Our Approach

- Previous work in this area has been syntactic in nature. Natural models of these notions have not been forthcoming. Non-interference results are proved syntactically.
- We take a semantic approach. We show that Game Semantics provides an intuitive and illuminating account of access control, and moreover leads to strikingly simple and robust proofs of interference-freedom.
- Advantages of the semantic approach: more robust and general. Still applicable to syntactic systems.

Our Approach

- Previous work in this area has been syntactic in nature. Natural models of these notions have not been forthcoming. Non-interference results are proved syntactically.
- We take a semantic approach. We show that Game Semantics provides an intuitive and illuminating account of access control, and moreover leads to strikingly simple and robust proofs of interference-freedom.
- Advantages of the semantic approach: more robust and general. Still applicable to syntactic systems.
- Some novelties in the Game Semantics: justified AJM games (with no justification pointers), eliminating the need for an 'intensional equivalence' on strategies.

Game Semantics

Justified AJM Games

Why AJM games?

Why AJM games?

- Want to cover linear type theories

Why AJM games?

- Want to cover linear type theories
- An occasion to revisit AJM game, rethink some basic issues

Justified AJM Games

Why AJM games?

- Want to cover linear type theories
- An occasion to revisit AJM game, rethink some basic issues

We will need *justifiers* to formulate the access control constraints.

Why AJM games?

- Want to cover linear type theories
- An occasion to revisit AJM game, rethink some basic issues

We will need *justifiers* to formulate the access control constraints.

Intuition for justifiers in terms of procedural control-flow:

Why AJM games?

- Want to cover linear type theories
- An occasion to revisit AJM game, rethink some basic issues

We will need *justifiers* to formulate the access control constraints.

Intuition for justifiers in terms of procedural control-flow:

- A call of procedure P will have as its justifier the currently active call of the procedure in which P was (statically) declared. ‘Link in the “static chain”’.

Why AJM games?

- Want to cover linear type theories
- An occasion to revisit AJM game, rethink some basic issues

We will need *justifiers* to formulate the access control constraints.

Intuition for justifiers in terms of procedural control-flow:

- A call of procedure P will have as its justifier the currently active call of the procedure in which P was (statically) declared. ‘Link in the “static chain”’.
- A procedure return will have the corresponding call as its justifier.

Look - no pointers!

Look - no pointers!

Why do we need justification pointers in HO games?

Look - no pointers!

Why do we need justification pointers in HO games?

Two reasons:

Look - no pointers!

Why do we need justification pointers in HO games?

Two reasons:

- Moves can have multiple occurrences in plays — this is how copying is performed.

Look - no pointers!

Why do we need justification pointers in HO games?

Two reasons:

- Moves can have multiple occurrences in plays — this is how copying is performed.
- The justification or enabling relation is often allowed to be many-valued.

Look - no pointers!

Why do we need justification pointers in HO games?

Two reasons:

- Moves can have multiple occurrences in plays — this is how copying is performed.
- The justification or enabling relation is often allowed to be many-valued.

AJM games are naturally *linear*: moves only occur once. Different copies are tagged or named explicitly - this is how the exponential works.

Look - no pointers!

Why do we need justification pointers in HO games?

Two reasons:

- Moves can have multiple occurrences in plays — this is how copying is performed.
- The justification or enabling relation is often allowed to be many-valued.

AJM games are naturally *linear*: moves only occur once. Different copies are tagged or named explicitly - this is how the exponential works.

Moreover, justification can be made single-valued (original HO games did this). This only requires a minor modification to the definition of the linear implication.

Look - no pointers!

Why do we need justification pointers in HO games?

Two reasons:

- Moves can have multiple occurrences in plays — this is how copying is performed.
- The justification or enabling relation is often allowed to be many-valued.

AJM games are naturally *linear*: moves only occur once. Different copies are tagged or named explicitly - this is how the exponential works.

Moreover, justification can be made single-valued (original HO games did this). This only requires a minor modification to the definition of the linear implication.

So our games will have a ‘static’ justification function

$$\mathrm{j}_A : M_A \rightharpoonup M_A$$

but no justification pointers — plays are just sequences of moves.

The Games Format

The Games Format

Justified AJM games have the structure $A = (M_A, \lambda_A, \mathrm{j}_A, P_A, \approx_A)$.

The Games Format

Justified AJM games have the structure $A = (M_A, \lambda_A, \mathrm{j}_A, P_A, \approx_A)$.

The justifier function inverts O/P labelling, and takes answers to questions. Those moves it is undefined on are *initial*.

The Games Format

Justified AJM games have the structure $A = (M_A, \lambda_A, \mathrm{j}_A, P_A, \approx_A)$.

The justifier function inverts O/P labelling, and takes answers to questions. Those moves it is undefined on are *initial*.

Global conditions on plays $s \in M_A^{\circledast}$:

(p1) Opponent starts If s is non-empty, it starts with an O-move.

(p2) Alternation Moves in s alternate between O and P.

(p3) Linearity Any move occurs at most once in s.

(p4) Well-bracketing Write each answer a as $)_a$ and the corresponding question $q = \mathrm{j}_A(a)$ as $(_a$. Then we require that s is well-bracketed in the obvious sense.

(p5) Justification If m occurs in s, $s = s_1 m s_2$, then the justifier $\mathrm{j}_A(m)$ must occur in s_1.

Constructions: Bang

The game $!A$ is defined as the "infinite symmetric tensor power" of A. The symmetry is built in via the equivalence relation on positions.

- $M_{!A} = \omega \times M_A = \sum_{i \in \omega} M_A$.
- Labelling is by source tupling: $\lambda_{!A}(i, a) = \lambda_A(a)$.
- Justification is componentwise: $\mathsf{j}_{!A}(i, m) = (i, \mathsf{j}_A(m))$.
- We write $s \upharpoonright i$ to indicate the restriction to moves with index i.

$$P_{!A} = \{s \in M_{!A}^{\circledast} \mid (\forall i \in \omega)\ s \upharpoonright i \in P_A\}\,.$$

- Let $S(\omega)$ be the set of permutations on ω. Then $s \approx_{!A} t$ iff:

$$(\exists \pi \in S(\omega))[(\forall i \in \omega.\, s \upharpoonright i \approx_A t \upharpoonright \pi(i))\ \wedge\ (\pi \circ \mathtt{fst})^*(s) = \mathtt{fst}^*(t)].$$

Linear Implication

- $M_{A \multimap B} = (\Sigma_{b \in \mathsf{Init}_B} M_A) + M_B$.
- $\lambda_{A \multimap B} = [[\overline{\lambda_A} \mid b \in \mathsf{Init}_B], \lambda_B]$.
- We define justification by cases. We write m_b, for $m \in M_A$ and $b \in \mathsf{Init}_B$, for the b-th copy of m.

$$\begin{array}{rcl} \mathsf{j}_{A \multimap B}(m_b) & = & \begin{cases} b, & m \in \mathsf{Init}_A \\ (\mathsf{j}_A(m))_b, & m \notin \mathsf{Init}_A \end{cases} \\ \mathsf{j}_{A \multimap B}(m) & = & \mathsf{j}_B(m), \qquad m \in M_B \,. \end{array}$$

- We write $s{\upharpoonright}A$ to indicate the restriction to moves in $\Sigma_{b \in \mathsf{Init}_B} M_A$, replacing each m_b by m.

$$P_{A \multimap B} = \{s \in M_{A \multimap B}^{\circledast} \mid s{\upharpoonright}A \in P_A \ \wedge \ s{\upharpoonright}B \in P_B\}$$

Note that Linearity for A implies that *only one copy* m_b of each $m \in M_A$ can occur in any play $s \in P_{A \multimap B}$.

Strategies

Strategies

We also show that AJM strategies can be simplified.

We also show that AJM strategies can be simplified.

A *strategy* on a game A is a non-empty set $\sigma \subseteq P_A^{\mathsf{even}}$ of even-length plays satisfying the following conditions:

Causal Consistency $sab \in \sigma \implies s \in \sigma$

Representation Independence $s \in \sigma \ \wedge \ s \approx_A t \implies t \in \sigma$

Determinacy $sab, ta'b' \in \sigma \ \wedge \ sa \approx_A ta' \implies sab \approx_A ta'b'$.

Strategies

We also show that AJM strategies can be simplified.

A *strategy* on a game A is a non-empty set $\sigma \subseteq P_A^{\text{even}}$ of even-length plays satisfying the following conditions:

Causal Consistency $sab \in \sigma \implies s \in \sigma$

Representation Independence $s \in \sigma \ \wedge \ s \approx_A t \implies t \in \sigma$

Determinacy $sab, ta'b' \in \sigma \ \wedge \ sa \approx_A ta' \implies sab \approx_A ta'b'$.

We can recover the usual notion as a 'skeleton', a subset of the strategy satisfying

Uniformization $\forall sab \in \sigma . s \in \phi \implies \exists! b' . sab' \in \phi$.

We also show that AJM strategies can be simplified.

A *strategy* on a game A is a non-empty set $\sigma \subseteq P_A^{\text{even}}$ of even-length plays satisfying the following conditions:

Causal Consistency $sab \in \sigma \implies s \in \sigma$

Representation Independence $s \in \sigma \land s \approx_A t \implies t \in \sigma$

Determinacy $sab, ta'b' \in \sigma \land sa \approx_A ta' \implies sab \approx_A ta'b'$.

We can recover the usual notion as a ‘skeleton’, a subset of the strategy satisfying

Uniformization $\forall sab \in \sigma. s \in \phi \implies \exists! b'. sab' \in \phi$.

Then the ‘intensional preorder’ $\precapprox$ on skeletons (old-style AJM strategies) reduces to subset inclusion on the new strategies.

Strategies

We also show that AJM strategies can be simplified.

A *strategy* on a game A is a non-empty set $\sigma \subseteq P_A^{\text{even}}$ of even-length plays satisfying the following conditions:

Causal Consistency $sab \in \sigma \implies s \in \sigma$

Representation Independence $s \in \sigma \ \wedge \ s \approx_A t \implies t \in \sigma$

Determinacy $sab, ta'b' \in \sigma \ \wedge \ sa \approx_A ta' \implies sab \approx_A ta'b'$.

We can recover the usual notion as a 'skeleton', a subset of the strategy satisfying

Uniformization $\forall sab \in \sigma . s \in \phi \implies \exists! b'. sab' \in \phi$.

Then the 'intensional preorder' $\sqsubseteq\!\!\!\!\approx$ on skeletons (old-style AJM strategies) reduces to subset inclusion on the new strategies.

Everything works out just fine!

The Model

Games Over A Lattice

Games Over A Lattice

Given a security semilattice $(\mathcal{L}, \sqcup, \perp)$, we define a category $\mathcal{G}_{\mathcal{L}}$ with objects

$$A = (M_A, \lambda_A, \mathsf{j}_A, P_A, \approx_A, \mathsf{lev}_A)$$

Justified AJM games with one new component $\mathsf{lev}_A : M_A \to \mathcal{L}$.

Games Over A Lattice

Given a security semilattice $(\mathcal{L}, \sqcup, \bot)$, we define a category $\mathcal{G}_{\mathcal{L}}$ with objects

$$A = (M_A, \lambda_A, \mathsf{j}_A, P_A, \approx_A, \mathsf{lev}_A)$$

Justified AJM games with one new component $\mathsf{lev}_A : M_A \to \mathcal{L}$.

This is carried componentwise through all the constructions on games, e.g.

$$\mathsf{lev}_{A \multimap B} = [[\mathsf{lev}_A \mid b \in \mathsf{Init}_B], \mathsf{lev}_B].$$

Games Over A Lattice

Given a security semilattice $(\mathcal{L}, \sqcup, \bot)$, we define a category $\mathcal{G}_{\mathcal{L}}$ with objects

$$A = (M_A, \lambda_A, \mathsf{j}_A, P_A, \approx_A, \mathsf{lev}_A)$$

Justified AJM games with one new component $\mathsf{lev}_A : M_A \to \mathcal{L}$.
This is carried componentwise through all the constructions on games, e.g.

$$\mathsf{lev}_{A \multimap B} = [[\mathsf{lev}_A \mid b \in \mathsf{Init}_B], \mathsf{lev}_B].$$

There is a single additional condition on plays:

(p6) Levels A non-initial move m can only be played if $\mathsf{lev}_A(m) \leq \mathsf{lev}_A(\mathsf{j}_A(m))$.

Given a security semilattice $(\mathcal{L}, \sqcup, \bot)$, we define a category $\mathcal{G}_{\mathcal{L}}$ with objects

$$A = (M_A, \lambda_A, \mathsf{j}_A, P_A, \approx_A, \mathsf{lev}_A)$$

Justified AJM games with one new component $\mathsf{lev}_A : M_A \to \mathcal{L}$.
This is carried componentwise through all the constructions on games, e.g.

$$\mathsf{lev}_{A \multimap B} = [[\mathsf{lev}_A \mid b \in \mathsf{Init}_B], \mathsf{lev}_B].$$

There is a single additional condition on plays:

(p6) Levels A non-initial move m can only be played if $\mathsf{lev}_A(m) \leq \mathsf{lev}_A(\mathsf{j}_A(m))$.

This constraint has a clear motivation: a principal can only affirm a proposition at its own level of authorization based on *assertions made at the same level or higher*. In terms of control flow (where the lattice has the opposite interpretation): a procedure can only perform an action *at its own security level or lower*.

Games Over A Lattice

Given a security semilattice $(\mathcal{L}, \sqcup, \bot)$, we define a category $\mathcal{G}_{\mathcal{L}}$ with objects

$$A = (M_A, \lambda_A, \mathsf{j}_A, P_A, \approx_A, \mathsf{lev}_A)$$

Justified AJM games with one new component $\mathsf{lev}_A : M_A \to \mathcal{L}$.
This is carried componentwise through all the constructions on games, e.g.

$$\mathsf{lev}_{A \multimap B} = [[\mathsf{lev}_A \mid b \in \mathsf{Init}_B], \mathsf{lev}_B].$$

There is a single additional condition on plays:

(p6) Levels A non-initial move m can only be played if $\mathsf{lev}_A(m) \leq \mathsf{lev}_A(\mathsf{j}_A(m))$.

This constraint has a clear motivation: a principal can only affirm a proposition at its own level of authorization based on *assertions made at the same level or higher*. In terms of control flow (where the lattice has the opposite interpretation): a procedure can only perform an action *at its own security level or lower*.

Note that formally, this is a purely *static constraint* (on types rather than strategies)!

The Level Monads

Fixing a level ℓ, we can embed $\mathcal{G}$ fully and faithfully into $\mathcal{G}_\mathcal{L}$ by giving every move of every game the level ℓ. Interesting things start to happen when there are moves at different levels.

Fixing a level ℓ, we can embed $\mathcal{G}$ fully and faithfully into $\mathcal{G}_{\mathcal{L}}$ by giving every move of every game the level ℓ. Interesting things start to happen when there are moves at different levels.

We define, for each $\ell \in \mathcal{L}$, a construction T_ℓ on games, which acts only on the level assignment:

$$\mathsf{lev}_{T_\ell A}(m) = \mathsf{lev}_A(m) \sqcup \ell.$$

Fixing a level ℓ, we can embed $\mathcal{G}$ fully and faithfully into $\mathcal{G}_\mathcal{L}$ by giving every move of every game the level ℓ. Interesting things start to happen when there are moves at different levels.

We define, for each $\ell \in \mathcal{L}$, a construction T_ℓ on games, which acts only on the level assignment:

$$\mathsf{lev}_{T_\ell A}(m) = \mathsf{lev}_A(m) \sqcup \ell.$$

The following commutation properties of T_ℓ are immediate.

Proposition 1 *The following equations hold:*

$$\begin{array}{lcl}
T_\ell I & = & I \\
T_\ell(A \otimes B) & = & T_\ell A \otimes T_\ell B \\
T_\ell(A \multimap B) & = & T_\ell A \multimap T_\ell B \\
T_\ell(A \,\&\, B) & = & T_\ell A \,\&\, T_\ell B \\
T_\ell\, !A & = & !T_\ell A \\
T_\ell(A \Rightarrow B) & = & T_\ell A \Rightarrow T_\ell B
\end{array}$$

The semilattice structure on $\mathcal{L}$ acts on the $\mathcal{L}$-indexed family of monads in the evident fashion:

Proposition 2 *The following equations hold:*

$$\begin{array}{rcl} T_\ell(T_{\ell'}A) & = & T_{\ell \sqcup \ell'}A \\ T_\perp A & = & A. \end{array}$$

We can extend each T_ℓ with a functorial action: if $\sigma : A \rightarrow B$ then we can define $T_\ell\sigma : T_\ell A \rightarrow T_\ell B$ simply by taking $T_\ell\sigma = \sigma$. To justify this, note that

$$P_{A \multimap B} = P_{T_\ell(A \multimap B)} = P_{T_\ell A \multimap T_\ell B}$$

Copycats and Levels

Proposition 3 *The copy-cat strategy is well defined on* $A \multimap T_\ell A$.

Proposition 3 *The copy-cat strategy is well defined on $A \multimap T_\ell A$.*

Proof Consider a play of the copy-cat strategy

$$
\begin{array}{cccc}
 & A & \multimap & T_\ell A \\
 & \vdots & & \vdots \\
O & & & m_1 \\
P & m_1 & & \\
O & m_2 & & \\
P & & & m_2
\end{array}
$$

One shows that the Level condition holds for each of these moves. □

Proposition 3 *The copy-cat strategy is well defined on* $A \multimap T_\ell A$.

Proof Consider a play of the copy-cat strategy

$$\begin{array}{llcl} & A & \multimap & T_\ell A \\ & \vdots & & \vdots \\ O & & & m_1 \\ P & m_1 & & \\ O & m_2 & & \\ P & & & m_2 \end{array}$$

One shows that the Level condition holds for each of these moves. □

Thus we can define a natural transformation $\eta_A : A \rightarrow T_\ell A$, where η_A is the copy-cat strategy. Furthermore, by Proposition 2, $T_\ell T_\ell A = T_\ell A$.

Proposition 3 *The copy-cat strategy is well defined on* $A \multimap T_\ell A$.

Proof Consider a play of the copy-cat strategy

$$\begin{array}{cccc} & A & \multimap & T_\ell A \\ & \vdots & & \vdots \\ O & & & m_1 \\ P & m_1 & & \\ O & m_2 & & \\ P & & & m_2 \end{array}$$

One shows that the Level condition holds for each of these moves. □

Thus we can define a natural transformation $\eta_A : A \rightarrow T_\ell A$, where η_A is the copy-cat strategy. Furthermore, by Proposition 2, $T_\ell T_\ell A = T_\ell A$.

Proposition 4 *Each* T_ℓ *is an idempotent commutative monad.*

Results

No Natural Contraflow

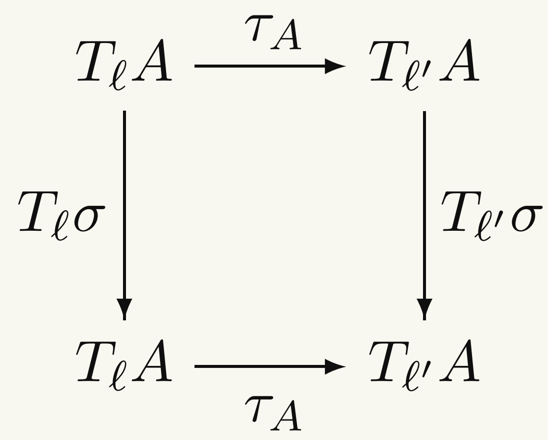
Firstly, we prove a strong form of converse of Proposition 3.

Proposition 5 *If* $\neg(\ell \leq \ell')$*, then there is no natural transformation from* T_ℓ *to* $T_{\ell'}$*.*

Firstly, we prove a strong form of converse of Proposition 3.

Proposition 5 *If* $\neg(\ell \leq \ell')$*, then there is no natural transformation from* T_ℓ *to* $T_{\ell'}$*.*

Proof Suppose for a contradiction that there is such a natural transformation τ. Given any flat game $X_\perp^\flat$, with $\mathsf{lev}_{X_\perp^\flat}(m) = \perp$ for all moves $m \in M_{X_\perp^\flat}$, the strategy $\tau_{X_\perp^\flat} : T_\ell X_\perp^\flat \to T_{\ell'} X_\perp^\flat$ can only play in $T_{\ell'} X_\perp^\flat$, since playing the initial move in $T_\ell X_\perp^\flat$ would violate the Level condition.
We now work the naturality square



with $A = \mathbf{Nat}_\perp^\flat$ to yield the required contradiction. $\square$

Formalizing Non-Flow

Formalizing Non-Flow

Consider the following situation. We have a term in context $\Gamma \vdash t : T$, and we wish to guarantee that t is not able to access some part of the context. For example, we may have $\Gamma = x : U, \Gamma'$, and we may wish to verify that t cannot access x. Rather than analyzing the particular term t, we may wish to guarantee this purely at the level of the types, in which case it is reasonable to assume that this should be determined by the types U and T, and independent of Γ'.

Formalizing Non-Flow

Consider the following situation. We have a term in context $\Gamma \vdash t : T$, and we wish to guarantee that t is not able to access some part of the context. For example, we may have $\Gamma = x : U, \Gamma'$, and we may wish to verify that t cannot access x. Rather than analyzing the particular term t, we may wish to guarantee this purely at the level of the types, in which case it is reasonable to assume that this should be determined by the types U and T, and independent of Γ'.

This can be expressed in terms of the categorical semantics as follows. Note that the denotation of such a term in context will be a morphism of the form $f : A \otimes C \to B$, where $A = [\![U]\!]$, $C = [\![\Gamma']\!]$, $B = [\![T]\!]$.

Formalizing Non-Flow

Consider the following situation. We have a term in context $\Gamma \vdash t : T$, and we wish to guarantee that t is not able to access some part of the context. For example, we may have $\Gamma = x : U, \Gamma'$, and we may wish to verify that t cannot access x. Rather than analyzing the particular term t, we may wish to guarantee this purely at the level of the types, in which case it is reasonable to assume that this should be determined by the types U and T, and independent of Γ'.

This can be expressed in terms of the categorical semantics as follows. Note that the denotation of such a term in context will be a morphism of the form $f : A \otimes C \to B$, where $A = [\![U]\!]$, $C = [\![\Gamma']\!]$, $B = [\![T]\!]$.

Definition 6 *Let $\mathcal{C}$ be an affine category, i.e. a symmetric monoidal category in which the tensor unit I is the terminal object. We write $\top_A : A \to I$ for the unique arrow. We define $A \not\rightarrow B$ if for all objects C, and $f : A \otimes C \to B$, f factors as*

$$f = A \otimes C \xrightarrow{\top_A \otimes \mathsf{id}_C} I \otimes C \xrightarrow{\cong} C \xrightarrow{g} B.$$

The idea is that no information from A can be used by f — it is "constant in A". Note that $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{G}_{\mathcal{L}}^{\mathsf{hf}}$ are affine, so this definition applies directly to our situation.

The No-Flow Theorem

Firstly, we characterize this notion in $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{G}_{\mathcal{L}}^{\mathsf{hf}}$.

Lemma 7 *In $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{G}_{\mathcal{L}}^{\mathsf{hf}}$, $A \not\rightarrow B$ if and only if, for any strategy $\sigma : A \otimes C \rightarrow B$, σ does not play any move in A.*

Firstly, we characterize this notion in $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{G}_{\mathcal{L}}^{\mathsf{hf}}$.

Lemma 7 *In $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{G}_{\mathcal{L}}^{\mathsf{hf}}$, $A \not\rightarrow B$ if and only if, for any strategy $\sigma : A \otimes C \rightarrow B$, σ does not play any move in A.*

We now give a simple characterization for when this “no-flow” relation holds between games.

Firstly, we characterize this notion in $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{G}_{\mathcal{L}}^{\mathsf{hf}}$.

Lemma 7 *In $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{G}_{\mathcal{L}}^{\mathsf{hf}}$, $A \not\rightarrow B$ if and only if, for any strategy $\sigma : A \otimes C \longrightarrow B$, σ does not play any move in A.*

We now give a simple characterization for when this "no-flow" relation holds between games.

Given a game A, we define:

$$
\begin{aligned}
\mathsf{Level}(A) \quad &= \quad \{\mathsf{lev}_A(m) \mid m \in \mathsf{Init}_A\} \\
A \rhd B \quad &\equiv \quad \forall \ell \in \mathsf{Level}(A), \ell' \in \mathsf{Level}(B). \neg(\ell \leq \ell')
\end{aligned}
$$

Firstly, we characterize this notion in $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{G}^{\mathsf{hf}}_{\mathcal{L}}$.

Lemma 7 *In $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{G}^{\mathsf{hf}}_{\mathcal{L}}$, $A \not\rightarrow B$ if and only if, for any strategy $\sigma : A \otimes C \rightarrow B$, σ does not play any move in A.*

We now give a simple characterization for when this “no-flow” relation holds between games.

Given a game A, we define:

$$\begin{array}{lcl} \mathsf{Level}(A) & = & \{\mathsf{lev}_A(m) \mid m \in \mathsf{Init}_A\} \\ A \rhd B & \equiv & \forall \ell \in \mathsf{Level}(A), \ell' \in \mathsf{Level}(B).\, \neg(\ell \leq \ell') \end{array}$$

Theorem 8 (No-Flow) *For any games A, B in $\mathcal{G}_{\mathcal{L}}$:*

$$A \not\rightarrow B \iff A \rhd B.$$

Computing Levels

Computing Levels

The characterization of no-flow in terms of the levels of types means that we can obtain useful information by computing levels.

Computing Levels

The characterization of no-flow in terms of the levels of types means that we can obtain useful information by computing levels.

We consider a syntax of types built from basic types (to be interpreted as flat games at a stipulated level) using the connectives of ILL extended with the level monads. For any such type T, we can give a simple inductive definition of $\mathsf{Level}(A)$ where $A = [\![T]\!]$:

$$
\begin{array}{lcl}
\mathsf{Level}(X^{\flat}_{\ell}) & = & \{\ell\} \\
\mathsf{Level}(I) & = & \varnothing \\
\mathsf{Level}(A \otimes B) & = & \mathsf{Level}(A) \cup \mathsf{Level}(B) \\
\mathsf{Level}(A \multimap B) & = & \mathsf{Level}(B) \\
\mathsf{Level}(A \,\&\, B) & = & \mathsf{Level}(A) \cup \mathsf{Level}(B) \\
\mathsf{Level}(A \Rightarrow B) & = & \mathsf{Level}(B) \\
\mathsf{Level}(!A) & = & \mathsf{Level}(A) \\
\mathsf{Level}(T_{\ell}A) & = & \{\ell \sqcup \ell' \mid \ell' \in \mathsf{Level}(A)\}
\end{array}
$$

The characterization of no-flow in terms of the levels of types means that we can obtain useful information by computing levels.

We consider a syntax of types built from basic types (to be interpreted as flat games at a stipulated level) using the connectives of ILL extended with the level monads. For any such type T, we can give a simple inductive definition of $\mathsf{Level}(A)$ where $A = [\![T]\!]$:

$$
\begin{array}{lcl}
\mathsf{Level}(X^{\flat}_{\ell}) & = & \{\ell\} \\
\mathsf{Level}(I) & = & \varnothing \\
\mathsf{Level}(A \otimes B) & = & \mathsf{Level}(A) \cup \mathsf{Level}(B) \\
\mathsf{Level}(A \multimap B) & = & \mathsf{Level}(B) \\
\mathsf{Level}(A \mathbin{\&} B) & = & \mathsf{Level}(A) \cup \mathsf{Level}(B) \\
\mathsf{Level}(A \Rightarrow B) & = & \mathsf{Level}(B) \\
\mathsf{Level}(!A) & = & \mathsf{Level}(A) \\
\mathsf{Level}(T_{\ell}A) & = & \{\ell \sqcup \ell' \mid \ell' \in \mathsf{Level}(A)\}
\end{array}
$$

This yields a simple, computable analysis which by Theorem 8 can be used to guarantee access constraints of the kind described above.

Protected Types

We give a semantic account of *protected types*, which play a key rôle in the DCC type system (Abadi, Bannerjee, Heintze, Riecke).

Definition 9 *We say that a game A is* protected at level ℓ *if* $\mathsf{Level}(A) \geq \ell$, *meaning that $\ell' \geq \ell$ for all $\ell' \in \mathsf{Level}(A)$.*

This notion extends immediately to types via their denotations as games.
The following (used as an inductive *definition* of protection in Abadi et al.) is an immediate *consequence* of our definition.

Lemma 10

1. *If $\ell \leq \ell'$, then $T_{\ell'} A$ is protected at level ℓ.*
2. *If B is protected at level ℓ, so are $A \multimap B$ and $A \Rightarrow B$.*
3. *If A and B are protected are level ℓ, so are $A \,\&\, B$ and $A \otimes B$.*
4. *If A is protected at level ℓ, so is $!A$.*
5. *I is protected at level ℓ.*

We also have the following *protected promotion* lemma, which shows the soundness of the key typing rule in DCC.

Lemma 11 *If $\sigma : !A \rightarrow T_\ell B$, $\tau : !B \rightarrow C$, and C is protected at level ℓ, then the coKleisli composition*

$$\sigma^\dagger ; \tau : !A \rightarrow C$$

is well-defined.

Proof Firstly, by Proposition 1, $T_\ell\, !B = !T_\ell B$. So it suffices to show that τ is well-defined as a strategy $\tau : T_\ell\, !B \rightarrow C$. If we consider an initial move m in $T_\ell\, !B$ played by τ, we must have $\mathsf{lev}_{!B}(m) \leq \mathsf{lev}(\mathsf{j}(m))$ since $\tau : !B \rightarrow C$ is well-defined. Moreover, $\ell \leq \mathsf{lev}(\mathsf{j}(m))$ since C is protected at ℓ. Hence $\mathsf{lev}_{T_\ell\, !B}(m) \leq \mathsf{lev}(\mathsf{j}(m))$. □

Stability Under Erasure

Stability Under Erasure

We now give a semantic version of the main result in Abadi's ICFP 06 paper (Theorem 7.6), which shows stability of the type theory under erasure of level constraints. This is used by Abadi to derive several other results relating to non-interference.

Stability Under Erasure

We now give a semantic version of the main result in Abadi's ICFP 06 paper (Theorem 7.6), which shows stability of the type theory under erasure of level constraints. This is used by Abadi to derive several other results relating to non-interference.

Firstly, given $\ell \in \mathcal{L}$, we define the erasure A^{ℓ} of a type A, which replaces every sub-expression of A of the form $T_{\ell'} B$, with $\ell' \geq \ell$, by $\top$. Semantically, this corresponds to erasing all moves m in the game (denoted by) A such that lev$(m) \geq \ell$, and all plays containing such moves.

Stability Under Erasure

We now give a semantic version of the main result in Abadi's ICFP 06 paper (Theorem 7.6), which shows stability of the type theory under erasure of level constraints. This is used by Abadi to derive several other results relating to non-interference.

Firstly, given $\ell \in \mathcal{L}$, we define the erasure A^ℓ of a type A, which replaces every sub-expression of A of the form $T_{\ell'} B$, with $\ell' \geq \ell$, by $\top$. Semantically, this corresponds to erasing all moves m in the game (denoted by) A such that $\mathsf{lev}(m) \geq \ell$, and all plays containing such moves.

Abadi's result is that, if we can derive a typed term in context $\Gamma \vdash e : A$, then we can derive a term $\Gamma^\ell \vdash e' : A^\ell$. To obtain an appropriate semantic version, we need to introduce the notion of *total* strategies. A strategy σ is total if when $s \in \sigma$, and $sa \in P_A$, then $sab \in \sigma$ for some b.

Stability Under Erasure

We now give a semantic version of the main result in Abadi's ICFP 06 paper (Theorem 7.6), which shows stability of the type theory under erasure of level constraints. This is used by Abadi to derive several other results relating to non-interference.

Firstly, given $\ell \in \mathcal{L}$, we define the erasure A^ℓ of a type A, which replaces every sub-expression of A of the form $T_{\ell'} B$, with $\ell' \geq \ell$, by $\top$. Semantically, this corresponds to erasing all moves m in the game (denoted by) A such that lev$(m) \geq \ell$, and all plays containing such moves.

Abadi's result is that, if we can derive a typed term in context $\Gamma \vdash e : A$, then we can derive a term $\Gamma^\ell \vdash e' : A^\ell$. To obtain an appropriate semantic version, we need to introduce the notion of *total* strategies. A strategy σ is total if when $s \in \sigma$, and $sa \in P_A$, then $sab \in \sigma$ for some b. This is the direct analogue of totality for functions, and will hold for the strategies denoted by terms in a logical type theory — although not in general for terms in a programming language equipped with general recursion. One can show that total strategies which are *finite* (or alternatively *winning*) in a suitable sense form a category with the appropriate structure to model intuitionistic and linear type theories.

Theorem 12 *Suppose that $\sigma : A \to B$ is a total strategy. Then so is $\sigma' : A^\ell \to B^\ell$ for any $\ell \in \mathcal{L}$, where σ' is the restriction of σ to plays in $A^\ell \multimap B^\ell$.*

Proof Suppose for a contradiction that σ' is not total, and consider a witness $sab \in \sigma \setminus \sigma'$, with $sa \in P_{A^\ell \multimap B^\ell}$. Then $\mathsf{lev}(b) \geq \ell$; but by the Level constraint, we must have $\mathsf{lev}(\mathsf{j}(b)) \geq \ell$, which by the Justification condition contradicts $sa \in P_{A^\ell \multimap B^\ell}$. $\square$

Further Directions

Further Directions

- We have considered a semantic setting which is adequate for both intuitionistic and (intuitionistic-)linear type theories. It would also be interesting to look at access control in the context of *classical type theories* such as $\lambda\mu$, particularly since it is suggested by Abadi and Garg and Pfenning that there are problems with access control logics in classical settings.

Further Directions

- We have considered a semantic setting which is adequate for both intuitionistic and (intuitionistic-)linear type theories. It would also be interesting to look at access control in the context of *classical type theories* such as $\lambda\mu$, particularly since it is suggested by Abadi and Garg and Pfenning that there are problems with access control logics in classical settings.

- The development of algorithmic game semantics suggests that it may be promising to look at automated analysis based on our semantic approach.

Further Directions

- We have considered a semantic setting which is adequate for both intuitionistic and (intuitionistic-)linear type theories. It would also be interesting to look at access control in the context of *classical type theories* such as $\lambda\mu$, particularly since it is suggested by Abadi and Garg and Pfenning that there are problems with access control logics in classical settings.

- The development of algorithmic game semantics suggests that it may be promising to look at automated analysis based on our semantic approach.

- We have developed our semantics in the setting of AJM games, equipped with a notion of justification. One could alternatively take HO-games as the starting point, but these would also have to be used in a hybridized form, with "AJM-like" features, in order to provide models for linear type theories. In fact, one would like a form of game semantics which combined the best features (and minimized the disadvantages) of the two approaches. Some of the ideas introduced in the present paper may be useful steps in this direction.